Discourse Modeling with Abstract Categorial Grammars

Aleksandre Maskharashvili

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Discourse Modeling with Abstract Categorial Grammars
Modélisation du Discours avec les Grammaires Catégorielles Abstraites

THÈSE

présentée et soutenue publiquement le

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Aleksandre MASKHARASHVILI

Composition du jury

Rapporteurs :	Laurent PREVOT	Professeur, Université Aix-Marseille
	Matthew STONE	Maître de conférences, Rutgers University
Examinateurs :	Philippe DE GROOTE	Directeur de Recherche, INRIA Nancy – Grand Est (Directeur de thèse)
	Sylvain POGODALLA	Chargés de recherche, INRIA Nancy – Grand Est (co-directeur de thèse)
	Laurence DANLOS	Professeur, Université Paris Diderot, INRIA – Rocquencourt, l'Institut Universitaire de France
	Annie FORET	Maître de conférences, IRISA, Université de Rennes 1
	Christian RETORÉ	Professeur, LIRMM, Université de Montpellier
	Mathieu CONSTANT	Professeur, Université de Lorraine, ATILF

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Résumé

Ce mémoire de thèse traite de la modélisation du discours dans le cadre grammatical des Grammaires Catégorielles Abstraites (Abstract Categorial Grammars, ACG). Les ACG offrent un cadre unifié pour la modélisation de la syntaxe et de la sémantique. Nous nous intéressons en particulier aux formalismes discursifs qui utilisent une approche grammaticale pour rendre compte des régularités des structures discursives. Nous étudions plusieurs formalismes grammaticaux qui s'appuient sur les Grammaires d'Arbres Adjoints (Tree-Adjoining Grammars, TAG): D-LTAG, G-TAG et D-STAG. Dans notre travail, nous proposons un encodage de G-TAG et un encodage de D-STAG. G-TAG est un formalisme introduit pour la génération de textes en langue naturelle à partir de représentations conceptuelles (sémantiques). D-STAG est un formalisme synchrone pour la modélisation de l'interface syntaxe-sémantique du discours. Il a été introduit pour l'analyse et la construction des structures discursives. L'encodage en ACG de G-TAG et de D-STAG permet d'éclairer le problème des connecteurs discursifs médiaux que les formalismes s'appuyant sur TAG ne traitent pas, du moins pas par un mécanisme grammatical. En effet, pour prendre en compte ces connecteurs, D-LTAG, G-TAG et D-STAG utilisent tous une étape extra-grammaticale. Notre encodage offre au contraire une approche purement grammaticale de la prise en compte de ces connecteurs discursifs. La méthode que nous proposons est générique et peut servir de solution à tout encodage des connecteurs médiaux de formalismes fondés sur les TAG. Notre encodage de G-TAG et de D-STAG se fait avec des ACG de second ordre. Les grammaires de cette classe sont réversibles. Elles recourent aux mêmes algorithmes polynômiaux pour construire les structures d'analyse, que ce soit à partir de chaînes de caractères ou à partir de formules logiques. Ainsi, ces grammaires peuvent être utilisées aussi bien en analyse qu'en génération. Les problèmes d'analyse et de génération avec les encodages de G-TAG et de D-STAG en ACG sont donc de complexité polynômiale.

Mots-clés: Grammaire Catégorielle Abstraite, discours, logique, grammaire, sémantique, syntaxe, TAG

Abstract

This dissertation addresses the questions of discourse modeling within a grammatical framework called Abstract Categorial Grammars (ACGs). ACGs provide a unified framework for both syntax and semantics. We focus on the discourse formalisms that make use of a grammatical approach to capture structural regularities of discourse. We study several TAG-based discourse grammar formalisms, D-LTAG, G-TAG, and D-STAG. In the present work, we propose ACG encodings of G-TAG and D-STAG. G-TAG is a formalism introduced for generating natural language texts out of conceptual (semantic) representation inputs. D-STAG is a synchronous formalism for modeling the syntax-semantics interface for discourse. It was introduced for discourse analysis (parsing). The ACG encodings of G-TAG and D-STAG shed light on the problem of clause-medial connectives that TAG-based formalisms leave out of account. To deal with a discourse that contains clause-medial connectives, D-LTAG, G-TAG, and D-STAG, all make use of an extra grammatical step. In contrast, the ACG encodings of G-TAG and D-STAG offer a purely grammatical approach to discourse connectives occupying clause-medial positions. The method we propose is a generic one and can serve as a solution for encoding clause-medial connectives with the formalisms based on TAGs. The ACG encodings of G-TAG and D-STAG are second-order. Importantly, the class of second-order ACGs consists of intrinsically reversible grammars. Grammars of this class use the same polynomial algorithm to build parse structures both for strings and logical formulas. Thus, second-order ACGs can be used both for parsing and generation. Therefore, the problems of parsing and generation with the ACG encodings of G-TAG and D-STAG are of polynomial complexity.

Keywords: Abstract Categorial Grammar, discourse, logic, grammar, semantics, syntax, TAG

Remerciements

I am deeply indebted and grateful to my supervisors, Dr. Philippe de Groote and Dr. Silvain Pogodalla, for their inspiration and guidance as I pursued this work.

Je dédie cette thèse à ma mère.

Contents

List of Tables

List of Figures

Introduction générale

Chapter 0

Le panorama

Contents

0.1 Introduction

Dans cette thèse, nous étudions le problème de la modélisation du discours à l'aide des Grammaires Catégorielles Abstraites (ACG). Une grande variété d'objets linguistiques peuvent se retrouver sous le terme de "discours". Ces travaux ne s'intéressent cependant qu'à une notion limitée de discours, à savoir les monologues écrits.

L'étude du discours a permis de montrer qu'il ne s'agit pas d'un simple sac de phrases, mais il y a encore débat sur ce dont il s'agit vraiment. Un grand nombre de chercheurs affirment que le discours a une *structure*, mais la nature de celle-ci est loin de faire consensus et donne lieu à diverses théories. Certaines de ces théories analysent cette structure comme des *connexions rhétoriques* entre divers éléments signifiants du discours, appelés *unités discursives* (*constituants discursifs*). Deux unités discursives forment une connexion rhétorique quand il existe une *relation rhétorique* entre elles. Il est alors possible de représenter cette structure en identifiant les unités discursives à des nœuds.

On peut ensuite relier deux nœuds par une arête si il y a une relation rhétorique entre unités discursives dénotées. Il en résulte alors une structure de graphe. D'après certaines approches, ces graphes sont toujours des arbres, tandis que d'autres prônent plutôt une structure de graphe acyclique dirigé (DAG). Lors de cette thèse, nous accepterons que *le discours a une structure*, mais nous ne trancherons pas sur la forme que prend cette structure, différente selon la théorie étudiée.

Dans la tradition des approches types logiques, de Groote (2001) a défini les Grammaires Catégorielles Abstraites (ACGs) dans le but de modéliser la syntaxe et la sémantique de manière uniforme. Les Grammaires Catégorielles Abstraites (ACGs) utilisent un cadre grammatical à deux niveaux : un niveau abstrait et un niveau objet.

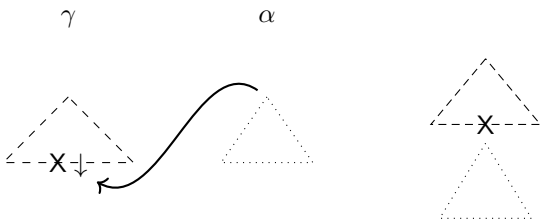
Dans cette thèse, nous étudions la possibilité d'encoder l'interface syntaxe-sémantique du discours à l'aide des ACGs. Un des objectifs de ces travaux est de construire des ACGs permettant de gérer les problèmes d'analyse et de génération. En d'autres termes, nous souhaitons encoder à l'aide des ACGs la manière dont les systèmes d'analyse et de génération de textes traitent le discours. Dans un même temps, un autre objectif est de produire des ACGs utilisables dans le sens où il serait possible de les implémenter pour une application pratique.

0.2 Grammaires Formelles

Étant donné qu'un texte est composé de phrases, l'analyse de texte introduit le problème de l'analyse de phrase. De nombreuses théories linguistiques ont été proposées dans le but de décrire les règles syntactiques régissant cette analyse des phrases en langue naturelle. Parmi elles, celles qui offrent une caractérisation rigoureuse et mathématique de l'ensemble des phrases (grammaticales, acceptables) d'une langue naturelle à l'aide d'un nombre fini de règles sont appelées des *grammaires formelles.*

Un des formalismes ayant prouvé son utilité dans la description de la syntaxe de la langue naturelle est celui des Grammaires d'Arbres Adjoints (TAGs) (A. K. Joshi, Levy, and Takahashi, 1975). Une TAG est constituée d'un ensemble fini d'*arbres élémentaires*, d'un ensemble de symboles *terminaux* et *non-terminaux* ainsi que d'un symbole non-terminal *distingué.* Chaque arbre élémentaire est soit *initial*, soit *auxiliaire.* Dans ce second cas, l'arbre a exactement une de ses feuilles qui est marquée comme étant son *nœud pied* (qui a nécessairement le même label non-terminal que la racine de l'arbre). Étant donné une grammaire TAG, on génère des arbres non-élémentaires, appelés arbres *dérivés*, en combinant les arbres élémentaires entre eux, avec d'autres arbres déjà dérivés ou en combinant des arbres dérivés. Il y a deux manières de combiner ces arbres TAG : par *substitution* ou par *adjonction.* Substituer un arbre A (dérivé à partir d'un arbre initial) dans un arbre B signifie remplacer une feuille non-terminale marquée pour la substitution de B par l'arbre A (voir Figure 0.1). L'adjonction d'un arbre A (dérivé d'un arbre auxiliaire) dans un arbre B est proche de la substitution mais peut être exécutée sur un nœud interne de B. Plus précisément, A remplace un nœud X de B de telle sorte que le père de X devienne le père de la racine de A et que les fils de X deviennent les fils du pied de A (voir Figure 0.1).

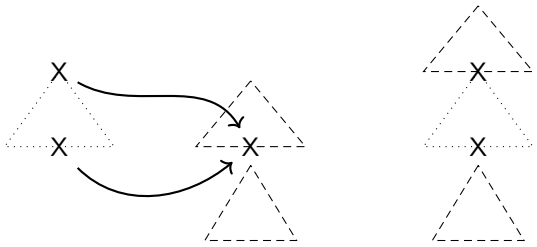
Nous nous intéresserons également aux TAGs synchrones (STAG) (Shieber and



(a) Substitution sur le nœud d'étiquette X

(b) L'arbre résultant de la substitution

Figure 0.1: L'opération de substitution



(a) Adjonction sur le nœud d'étiquette X X

(b) L'arbre résultant de l'adjonction

Figure 0.2: L'opération d'adjonction

Schabes, 1990). Ici, les structures élémentaires sont des paires d'arbres. Dans une paire $\langle \alpha_1, \alpha_2 \rangle$, les sites de substitutions et d'adjonctions de α_1 sont liés à ceux de α_2. Cela permet de définir la substitution/adjonction d'une paire d'arbres avec une autre paire d'arbres en effectuant cette substitution/adjonction sur les nœuds liés.

0.3 Grammaires Catégorielles Abstraites

De Groote (2001) introduit les Grammaires Catégorielles Abstraites (ACGs) pour traiter l'interface syntaxe-sémantique pour laquelle il est intéressant d'encoder à la fois la syntaxe et la sémantique avec la même machinerie. Les ACGs s'inspirent à la fois des idées de Curry, 1960 sur la distinction entre deux niveaux de grammaire et de l'approche de Montague, 1973 de l'interface syntaxe-sémantique où il est possible de traduire les structures syntaxiques en structures sémantiques. Une grammaire ACG définit deux langages, un langage *abstrait* et un langage *objet*, reliés par un *lexique*. Un exemple de grammaire ACG est d'avoir un langage abstrait correspondant à l'ensemble des structures de dérivation d'une grammaire quelconque, tandis que le langage objet correspond aux réalisations de cette grammaire. Le lexique relie alors chaque dérivation à une réalisation, résultant de cette dérivation. Par définition, une grammaire ACG

est un quadruplet composé de deux signatures *linéaires* d'ordre supérieur, d'un type *distingué* et d'un lexique. Une de ces signatures est appelée *vocabulaire abstrait* et l'autre *vocabulaire objet.* Le lexique est un *homomorphisme de type* faisant correspondre les constantes et les types du vocabulaire abstrait à des termes construits sur ceux du vocabulaire objet. Il est nécessaire pour ce lexique que l'image du type d'une constante soit identique au type de l'image d'une constante. Le langage abstrait est alors l'ensemble de tous les termes construits sur le vocabulaire abstrait et étant du type distingué, tandis que le langage objet est l'image de ce langage abstrait par le morphisme du lexique.

On peut encoder les TAGs à l'aide des ACGs en modélisant les arbres de dérivation TAG dans le vocabulaire abstrait et les arbres dérivés dans le vocabulaire objet (de Groote, 2002). Il est ensuite possible de définir une autre ACG dont le langage abstrait est le langage objet de l'ACG précédente. Le langage objet de cette nouvelle ACG est le langage des chaînes de terminaux (des formes de surface). Le nouveau lexique fait correspondre les termes représentant des arbres TAG dérivés aux termes représentant leurs frontières. Par conséquent, nous avons deux ACG : une qui modélise un langage d'arbre TAG dérivé comme son langage objet et une autre qui modélise un langage de chaîne de terminaux TAG comme son langage objet. Comme le langage objet de la première ACG est le langage abstrait de la seconde, on peut s'intéresser à la composition des deux. Son vocabulaire abstrait est celui de la première ACG, son vocabulaire objet est celui de la seconde, son type distingué est celui de la première et son lexique est la *composition* des lexiques des deux ACGs. Le lexique de cette ACG composée traduit donc des arbres de dérivation TAG en des chaînes de terminaux. Cette propriété de composition des ACGs est importante car elle rend possible la construction d'une architecture modulaire et connectée.

Afin d'encoder les TAGs en ACGs, nous montrerons comment encoder les arbres et les chaînes en λ-termes linéaires. Pour les TAGs, nous montrerons que la substitution et l'adjonction deviennent des applications fonctionnelles sur les λ-termes linéaires. Nous construirons ensuite un vocabulaire abstrait pour une TAG. Pour chaque arbre élémentaire de cette TAG, nous introduisons une constante du vocabulaire abstrait dont le type encode les substitutions et adjonctions que cet arbre peut recevoir. Plus précisément, pour chaque symbole non-terminal labellisant un site de substitution ou d'adjonction d'un arbre élémentaire, nous définissons un type *atomique* (les types les plus simples de la hiérarchie des types) dans le vocabulaire abstrait. Ces sites correspondent alors à des arguments de type atomique pour la constante modélisant l'arbre élémentaire. Par conséquent, les arbres adjoints ou substitués dans un arbre donné sont encodés comme des termes ayant un type atomique. Le lexique traduit différemment les types atomiques qui encodent des substitutions et ceux qui encodent des adjonctions dans le vocabulaire objet modélisant les arbres dérivés. Plus précisément, les types des substitutions deviennent des types atomiques alors que ceux des adjonctions deviennent des types fonctionnels. De cette manière, les ACGs encodant des TAGs sont du *second ordre* car toutes les constantes du vocabulaire abstrait ont un type du second ordre, ce qui signifie que les arguments des constantes abstraites ne peuvent être que de type atomique. Les ACGs du second ordre peuvent encoder des formalismes plus expressifs que les TAGs (de Groote and Pogodalla, 2004).

Afin d'encoder des TAGs avec la sémantique de Montague à l'aide des ACGs (Pogodalla, 2004, 2009), il faut relâcher la contrainte de linéarité présente dans la définition des ACGs. Plus précisément, on autorise le lexique à faire correspondre un terme d'un type *quasi-linéaire* à une constante abstraite qui est, elle, d'un type linéaire (Kanazawa, 2007). Ceci provient de la sémantique de Montague, dont les λ-termes sémantiques sont non-linéaires. L'architecture ACG utilisée pour les TAGs avec la sémantique de Montague se compose de deux ACGs partageant un même vocabulaire abstrait, qui modélise les arbres de dérivation TAG. Ces deux ACGs ont cependant des vocabulaires objets distincts, l'un modélisant des arbres TAGs dérivés tandis que l'autre modélise des formules de la sémantique de Montague. Le premier lexique, que nous appelons lexique *syntactique*, traduit donc les arbres de dérivation TAGs en arbres TAGs dérivés quand le second, appelé lexique *sémantique*, les traduit en formules logiques. De cette manière, les arbres de dérivation TAGs deviennent des médiateurs entre la syntaxe et la sémantique. Les ACGs que nous construisons ici sont du second ordre car leur vocabulaire abstrait est du second ordre. Cela est important car la complexité de l'analyse et de la génération dans le cadre des ACGs du second ordre est polynomiale (Kanazawa, 2007; Salvati, 2005).

0.4 Formalismes Discursifs

Nous avons parlé des formalismes qui capturent les régularités de la structure du discours à l'aide de grammaires. Comme l'encodage sous forme d'ACG des TAGs avec la sémantique de Montague permet de résoudre le problème de l'interface syntaxe-sémantique, les formalismes discursifs fondés sur les TAGs nous intéressent tout particulièrement. Nous pouvons en effet nous attendre à pouvoir user d'une approche similaire pour encoder ces formalismes discursifs fondés sur les TAGs dans le cadre des ACGs. Parmi ces formalismes, nous nous concentrons sur les G-TAG (Danlos, 1998) et sur les D-STAG (Danlos, 2011). G-TAG est un formalisme conçu pour la génération de texte alors que D-STAG est conçu pour l'analyse du discours. G-TAG propose un structure discursive en forme d'arbre alors que D-STAG permet d'avoir un structure de graphe acyclique dirigé (DAG).

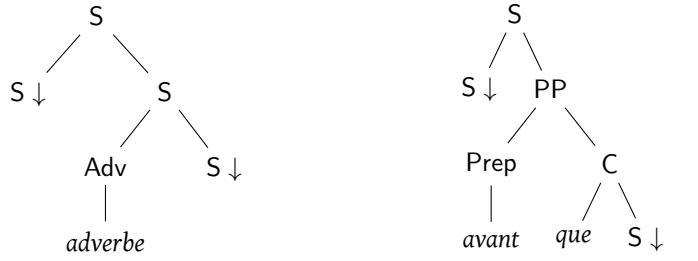
Les grammaires discursives dans ces formalismes incluent des arbres élémentaires représentant des connecteurs discursifs, ces derniers étant soit des conjonctions de subordination, soit des adverbes discursifs, soit la chaîne vide (connecteur non exprimé lexicalement).

0.4.1 G-TAG

Les G-TAG génèrent un texte à partir d'une représentation conceptuelle modélisant un contenu grâce à un langage formel. Elles font correspondre des concepts de cette représentation conceptuelle à une réalisation linguistique. La structure qui en résulte est la *g-dérivation* du texte. Cette g-dérivation rappelle l'arbre de dérivation TAG. Chaque g-dérivation spécifie un unique arbre *g-dérivé*. Contrairement aux arbres dérivés TAG, aucune ancre d'un arbre g-dérivé n'est fléchie, mais ils contiennent les informations

nécessaires pour les fléchir. Ces informations sont utilisés par le module de post-traitement des G-TAG, qui calcule la forme fléchie des mots avant de produire un texte. Les G-TAG ont un traitement spécial pour les textes avec une *conjonction réduite*, comme dans l'exemple suivant :

(1) Jean a passé l'aspirateur pour être récompensé par Marie.



(a) Un arbre élémentaire ancré par un *adverbe*

(b) Un arbre élémentaire ancré par une *conjonction*

Figure 0.3: Arbres élémentaires ancrés par un adverbe et une conjonction de subordination

Comme le montre (1), la clause subordonnée (être récompensé par Marie) et la clause matrice (Jean a passé l'aspirateur) partagent leur sujet sémantique *Jean* qui n'est syntaxiquement le sujet que de la première clause. On appelle ce type de relation des conjonctions réduites. Pour générer de tels textes, les G-TAG ont une approche ad hoc. De plus, pour générer des textes dans lesquels un connecteur est médial, les G-TAG usent également de procédés non-grammaticaux. En effet, comme les connecteurs sont les ancres lexicales d'arbres élémentaires représentés sur la Figure 0.3, ils ne peuvent apparaître qu'en position initiale. Pour générer un texte avec des connecteurs médiaux, le module de post-traitement des G-TAG commence par générer un texte dans lequel tous les connecteurs sont initiaux (comme dans (2)(a)) avant de déplacer certains d'entre eux en position médiale. C'est ainsi qu'une G-TAG peut générer un texte avec des connecteurs médiaux (comme dans (2)(b)).

(2) a. Jean a passé l'aspirateur. *Ensuite*, il a fait une sieste.

b. Jean a passé l'aspirateur. Il a *ensuite* fait une sieste.

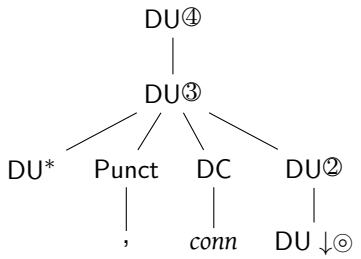
0.4.2 D-STAG

Les D-STAG s'intéressent à l'interface syntaxe-sémantique dans le discours. Elles se fondent sut les TAGs synchrones, ce qui signifie que les structures élémentaires des D-STAG sont des paires d'arbres élémentaires. Un des arbres de ces paires d'arbres élémentaires est ancré par une entrée lexicale tandis que l'autre est ancré par un λ-terme simplement typé, correspondant à l'interprétation sémantique de l'entrée lexicale. Les

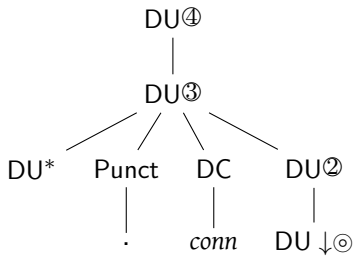
structures dérivés sont alors également des paires d'arbres, l'un d'eux étant appelé l'arbre syntaxique et l'autre l'arbre sémantique.

Lorsque, dans un texte, un connecteur se trouve en position médiale, les D-STAG doivent commencer par le *normaliser* en le déplaçant en début de clause.

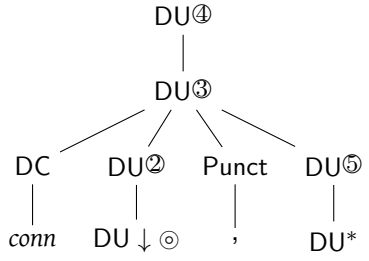
Afin que le discours soit cohérent, l'ajout d'une nouvelle clause doit être relié au reste du discours par le biais d'une relation rhétorique. Dans le cadres des D-STAG, l'arbre correspondant à cette nouvelle clause est substitué dans l'arbre ancré par le connecteur qui exprime cette relation rhétorique. L'arbre dérivé qui en résulte est ensuite adjoint au discours courant.



(a) conn est une conjonction postposée



(b) conn est un adverbe ou un connecteur vide



(c) conn est une conjonction préposée

Figure 0.4: L'arbre auxiliaire ancré par *conn*, où *conn* est soit une conjonction préposée, soit un adverbe discursif, soit une conjonction postposée (DU provient d'une unité de discours)

Il y a trois catégories de connecteurs : les conjonctions préposées, les conjonctions postposés et les adverbiaux. Comme on peut le voir sur la Figure 0.4, les arbres représentant les conjonctions postposées et les adverbiaux ont la même structure alors que ceux représentant les conjonctions préposées ont une structure différente. Cela est dû au fait que les conjonctions préposées créent un ≪cadre de discours≫ (Charolles, 2005).

Si l'on s'intéresse à la forme des arbres élémentaires des D-STAG, on observe que l'adjonction sur divers sites produit des arbres dérivés dont la frontière est identique. En d'autres termes, leur forme de surface est la même. D'un autre côté, les arbres sémantiques dérivés obtenus en parallèle des arbres syntaxiques produisent des formules sémantiques distinctes. Cette propriété implique une ambiguïté inhérente à l'analyse de texte avec les D-STAG.

0.4.3 Le problèmes des connecteurs médiaux

Pour traiter les connecteurs médiaux, tous les formalismes précédemment présentés nécessitent un traitement (externe à la grammaire) supplémentaire des textes. Ainsi, ces formalismes utilisent une approche en deux temps pour s'occuper de la génération et de l'analyse de discours.

0.5 G-TAG comme ACGs

Comme les G-TAG permettent de former une grammaire discursive, on pourrait supposer qu'encoder les G-TAG à l'aide des ACGs nous permette de modéliser le discours dans le cadre des ACGs. En effet, G-TAG s'inspire des TAGs et l'encodage des TAGs en ACGs serait un bon point de départ. Cependant, il y a des différences entre les G-TAG et les TAGs. Par exemple, elles ont une approche différente des informations lexico-syntaxiques. Tandis que les arbres dérivés TAGs ont des symboles terminaux fléchis, les arbres dérivés G-TAG ont seulement des lemmes sur leur terminaux, leurs informations morphologiques et leur flexions se trouvant sur leur nœud père. Ainsi, au lieu d'encoder directement les arbres de g-dérivation et les arbres g-dérivés, nous utiliserons la même approche que les TAGs, à savoir avoir des terminaux fléchis et non des lemmes.

Afin d'encoder les G-TAG avec les ACGs, nous construisons un vocabulaire abstrait modélisant les arbres de g-dérivation. Comme notre approche de la morpho-syntaxe est celle des TAGs, nous typons nos constantes abstraites par des types du second ordre construits à partir des sites de substitution et d'adjonction des arbres élémentaires.

Pour encoder les arbres dérivés et les représentations conceptuelles, nous utilisons de nouveau le vocabulaire objet présenté lors de l'encodage en ACG des TAGs avec la sémantique de Montague. En revanche, au lieu d'interpréter directement les arbres de g-dérivation par des arbres dérivés, nous commençons par les interpréter par des arbres de dérivation TAGs avant d'interpréter ces derniers par des arbres dérivés comme cela est fait dans l'encodage en ACG des TAGs.

Les G-TAG génèrent des textes contenant des conjonctions réduites par le biais de mécanismes extra-grammaticaux. Par opposition, l'approche que nous développons pour générer ces conjonctions réduites est purement grammaticale. Notre analyse est présentée en Figure 3. Nous séparons le sujet de la clause matrice, ce qui nous permet en sémantique d'exprimer le partage de sujet entre la clause matrice et la clause subordonnée. Du côté de la syntaxe, nous rendons le sujet seulement à la clause matrice.

De même, afin de traiter les textes contenant des connecteurs médiaux, les G-TAG ont recours à un module de post-traitement qui déplace ces connecteurs médiaux en position initiale car les mécanismes purement grammaticaux des G-TAG ne permettent pas d'analyser ou de générer de connecteurs médiaux. C'est pourquoi l'encodage actuel des G-TAG par les ACGs ne permet pas de traiter ces textes.

0.6 Chapter 7 - Clause-Medial Connectives

Pour étendre l'encodage ACG des G-TAG de sorte qu'il puisse traiter les textes contenant des connecteurs médiaux, commençons par regarder leur traitement par les TAGs. Dans ce cadre, on obtient un connecteur médial en adjoignant un arbre auxiliaire de racine VP ancré par le connecteur dans un arbre dérivé de racine S. Comme ce procédé place le connecteur dans le VP de la clause, cela provoque un défaut sémantique. En effet, la relation discursive représentée par ce connecteur n'obtient qu'un seul de ses deux arguments. Par opposition, les G-TAG représentent cette même relation discursive par un arbre initial ancré par le connecteur et disposant de deux sites de substitution de type S. Avec cette approche, la relation discursive reçoit bien ses deux arguments mais le connecteur ne peut être qu'en position initiale. Pour traiter correctement cette situation, nous combinons ces deux visions. Plus précisément, nous proposons d'encoder les connecteurs médiaux par des constantes abstraites à deux arguments. Un de ces arguments modélise l'arbre dérivé d'une phrase tandis que l'autre modélise un arbre *incomplet* qui attend l'adjonction d'un VP (qui introduira le connecteur dans le VP) pour être l'arbre dérivé d'une phrase. Cette contrainte pourrait être exprimée par un type fonctionnel (cette *fonction* prendrait un arbre auxiliaire de racine VP et produirait un arbre dérivé de racine S dans lequel le premier arbre a été adjoint au VP). La Figure 0.5 illustre cette analyse. Malheureusement, cela produirait une constante du troisième ordre, et donc un vocabulaire abstrait du troisième ordre. Or, nous ne disposons pas d'algorithme garantissant une analyse et une génération de texte en temps polynomial dans le cadre des ACGs d'ordre supérieur à deux. Afin de contourner ce problème de complexité, nous devons nous limiter à des ACGs du second ordre. Par conséquent, nous attribuons un type du second ordre à nos constantes abstraites en distinguant les deux arguments de la constante encodant le connecteur médial, chacun recevant un type atomique différent. Nous interprétons alors différemment ces deux types lors du passage aux arbres de dérivation TAGs. Le type de l'argument encodant l'arbre dérivé de racine S se traduit en un type atomique tandis que l'autre se traduit par un type fonctionnel indiquant que l'arbre TAG correspondant attend une adjonction VP pour devenir l'arbre dérivé d'une clause.

Un connecteur a la même sémantique, qu'il se trouve en position initiale ou médiale, c'est pourquoi nous interprétons les constantes modélisant ces deux catégories de connecteurs avec les même termes sémantiques. Ainsi, nous obtenons à la fois une interprétation syntaxique et une interprétation sémantique de nos constantes abstraites encodant les connecteurs médiaux. Cette approche nous permet alors d'analyser et de générer des textes avec ces deux catégories de connecteurs en une seule étape (sans avoir recours à plus d'étapes de traitements).

0.7 D-STAG comme ACGs

Comme D-STAG est un formalisme discursif dont la partie grammaticale est fondée sur les TAGs, nous cherchons à le modéliser par le biais des ACGs. Pour encoder les D-STAG avec les ACGs, nous modélisons les arbres de dérivation D-STAG par un

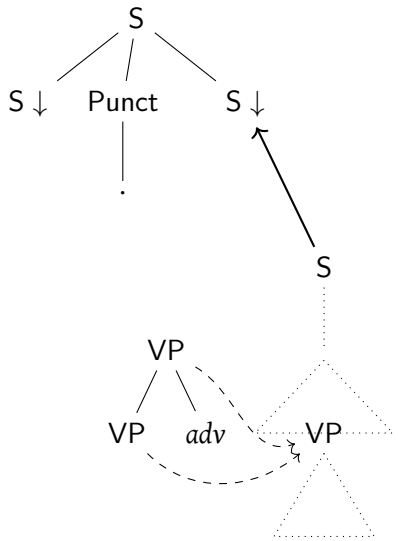


Figure 0.5: Une analyse d'un texte contenant un connecteur adverbial *adv* en position médiale

vocabulaire abstrait tandis que les arbres syntaxiques et sémantiques sont encodés par des vocabulaires objets. En particulier, nous utilisons pour ce vocabulaire objet celui que nous avions développé pour l'encodage des TAGs par les ACGs. Nous construisons donc deux ACGs partageant un même langage abstrait. Pour modéliser le niveau clausal de la grammaire, nous utilisons le niveau clausal de la grammaire obtenue en encodant les TAGs par des ACGs. Nous couvrons donc à la fois le niveau clausal et le niveau discursif de la grammaire.

Notre encodage ne nécessite pas d'étape de traitement supplémentaire pour normaliser les textes contenant des connecteurs médiaux. Il est cependant capable de gérer ces connecteurs, que ce soit pour générer ou pour analyser un texte, quand les D-STAG ne peuvent faire d'analyse qu'après une étape de pré-traitement. Plus précisément, nous développons une approche des connecteurs médiaux similaire à celle décrite précédemment pour les G-TAG. De plus, notre approche des connecteurs discursifs est plus uniforme dans le cadre des D-STAG que dans celui des G-TAG. En effet, nous encodons tous les connecteurs (à l'exception des conjonctions préposées introduisant un ≪cadre de discours≫) par des constantes du même type dans notre vocabulaire abstrait. Ensuite, afin de distinguer les connecteurs médiaux des connecteurs initiaux dans les arbres syntaxiques, nous les interprétons différemment. Plus précisément, alors que nous modélisons les connecteurs médiaux à l'aide d'une adjonction sur un nœud VP, nous modélisons les connecteurs initiaux à l'aide d'une adjonction sur un nœud S.

L'interprétation sémantique des constantes encodant les connecteurs médiaux et initiaux est identique, puisque la seule différence entre eux se trouve uniquement dans leurs emplacements syntaxiques dans le texte.

Étant donné que nous pouvons modéliser tous les arbres constructibles avec les D-STAG, notre encodage hérite de son ambiguïté intrinsèque lors de l'analyse. De plus, notre vocabulaire abstrait est du second ordre et notre lexique est quasi-linéaire.

Dans ce cas, les résultats de Kanazawa, 2007 nous assurent une complexité polynomiale pour la génération et l'analyse de texte. Ainsi, notre approche permet de modéliser le discours car les ACGs rendent possible l'analyse et la génération avec des connecteurs à la fois médiaux et initiaux tout en restant utilisable en pratique et en ne nécessitant qu'une seule étape.

De plus, nous interprétons les arbres de dérivation D-STAG par des formules étiquetées. Cela nous permet de nommer explicitement les arguments de nos connecteurs avec ces étiquettes.

o.8 Conclusion

Dans cette thèse, nous avons encodé des phénomènes discursifs à l'aide des ACGs. Nous avons en particulier encodé deux formalismes du discours, les G-TAG et les D-STAG. Ces encodages permettent de traiter les connecteurs médiaux de manière purement grammaticale. Ces deux encodages sont du second ordre, or les ACGs du second ordre disposent d'algorithmes polynomiaux de génération et d'analyse. Nous sommes donc en mesure de générer et d'analyser des textes avec une complexité polynomiale à l'aide de notre encodage en ACG. La Figure o.6 illustre l'architecture ACG utilisée à la fois pour D-STAG et pour G-TAG. Les cercles correspondent aux langages et les flêches aux lexiques.

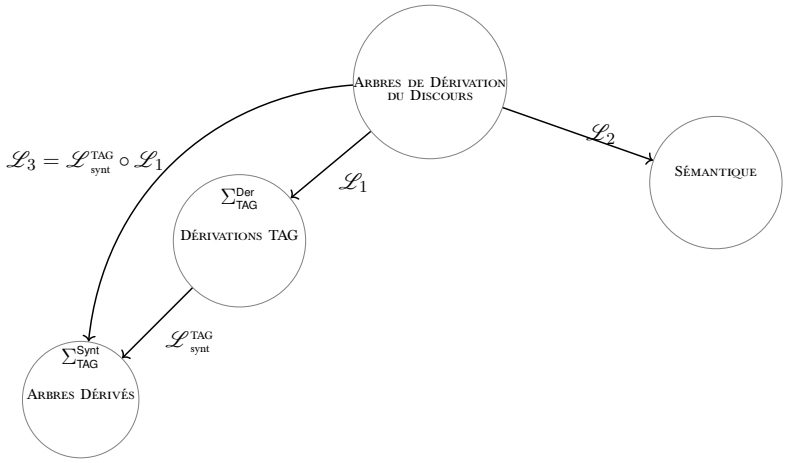


Figure o.6: L'architecture ACG pour G-TAG et D-STAG

Part I

Introduction

Chapter 1

Introduction

Contents

In this chapter we briefly overview some basic concepts and notions that we use throughout the thesis. We briefly discuss the problems of discourse (text) parsing and generation. Afterwards, we determine the questions within the discourse modeling problem that the present work addresses.

Since 1950s a number of theories have been proposed for analyzing natural language sentences, from both the syntactic and semantic points of view. Discourse processing can be seen as a further step in natural language studies. Under 'discourse' one may understand various kinds of natural language acts, either in written or spoken form, or a multi-modal one. In this thesis, we will consider a restricted notion of a discourse by focusing on only monologues. Thus, we consider 'discourse' and 'monologic text' or just 'text' to be synonyms (unless otherwise stated).

1.1 Discourse Coherence, Structure, and Interpretation

> Let us begin with a fact: discourse has structure. Whenever we read something closely, with even a bit of sensitivity, text structure leaps off the page at us. Hobbs (1985)

A discourse is not just a set of meaningful propositions (sentences). By listing sentences, one does not necessarily obtain a *meaningful* discourse, even though each of the sentences might be meaningful on its own. The bits of information presented by different *meaningful pieces* of a discourse must be *related*. Otherwise, a discourse would be hardly comprehensible.

Moreover, even though a discourse may consist of several pieces and they might be *connected* to each other in such ways that their conglomeration makes *sense*, the discourse may still sound *odd*. To illustrate that, let us consider the discourses (3) and (4) from (Asher and Lascarides, 2003).

(3) π_1. A burglar broke into Mary's apartment.
π_2. Mary was asleep.
π_3. He stole some silver.

(4) π_1. A burglar broke into Mary's apartment.
π'_2. A police woman visited her the next day.
π_3. ??He stole some silver.

Both of the discourses (3) and (4) convey messages that are fairly straightforward to perceive. Nevertheless, the discourse (4) sounds odd, whereas the discourse (3) does not. In (4) the anaphoric reference provided by *he* in π_3 to *a bugler* in π_1 makes the discourse (4) infelicitous. One may argue that a problem with the discourse (4) is due to the mispositioned sentence π_3. Indeed, if π_3 were placed right after π_1 in (4), then the resultant discourse would be felicitous. On the other hand, in the discourse (3), π_3 does not come right after π_1, but in spite of that, the discourse (3) is felicitous. As Asher and Lascarides (2003) explain, in the discourse (3), π_2 provides a *background* information to π_1. Since π_2 is a background information, it cannot break the *main story line* established by π_1. Therefore, the discourse π_1 π_2 π_3, where *he* from π_3 refers to *a burglar* introduced in π_1 (not in π_2), is felicitous. However, in the case of the discourse (4), adding π'_2 to π_1 creates a *narrative*. In this case, π_1 cannot create the main story line of the discourse. Adding π_3 after π'_2 makes the resultant discourse infelicitous because the anaphoric pronoun *he* from π_3 cannot find its antecedent in π_1 due to presence of π'_2 between π_1 and π_3.

Various theories address the problem of defining what is a *coherent* (meaningful, felicitous) discourse (e.g. (Asher and Lascarides, 2003; Hobbs, 1985; Mann and Thompson, 1986; Marcu, 1997)). Most of the theories agree that a coherent discourse should have a connected *structure*, where the structure is defined by certain *relations* connecting discourse units, i.e., meaningful pieces of a discourse. These relations are sometimes called *coherence relations* (Hobbs, 1985), or *rhetorical relations* (Mann and Thompson, 1986), or simply *discourse relations*.

Some theories consider the structure to be the main subject of the discourse studies (Hobbs, 1985; Mann and Thompson, 1987). At the same time, the notion of a discourse structure is not universally agreed but varies from theory to theory (Asher and Lascarides, 2003; Mann and Thompson, 1987; Bonnie Webber, Stone, Aravind Joshi,

and Knott, 2003). Given a discourse, one can consider discourse units (constituents) as nodes. By placing an edge between two nodes if they are related by a discourse relation, one obtains a graph representation of a discourse structure. While some authors assume that a discourse structure is tree-shaped (Mann and Thompson, 1987), some others argue against that (Danlos, 2011; Wolf and Gibson, 2004). What *kind* of non-tree shaped graphs discourse structures are is also among the questions that are still subjects to discussions. In the rest of the thesis, we will assume that *a discourse has a structure.* However, what a discourse structure is will depend on a particular theory.

Apart from theories that study discourse structure, theories such as Discourse Representation Theory (DRT) (Kamp, 1988; Kamp, van Genabith, and Reyle, 2011) and Dynamic Predicate Logic (DPL) (Groenendijk and Stokhof, 1991), aim to interpret a discourse as a logical form, similar to Montague's (1973) translations of sentences into logical forms.

Segmented Discourse Representation Theory (SDRT) (Asher and Lascarides, 2003) has similar goals as DRT and DPL, that is, it also interprets a discourse in a logical setting. However, SDRT makes use of the pragmatic knowledge coming from *the structure of discourse*, to which DRT and DPL do not pay significant attention. For instance, in order to explain why the discourse (4) sounds odd, the knowledge obtained from the logical interpretation of (4) would not help much. It is the *discourse structure* and its properties (derived from pragmatic knowledge) that show why the discourse (4) is infelicitous.

1.2 Discourse Processing

In computational linguistics, the problem of discourse processing has been studied from various points of view. Among fundamental problems of discourse processing are discourse *parsing* (analysis) and *generation.*

1.2.1 Discourse Parsing

Given a discourse theory, the main goal of the discourse parsing (analysis) task is to find an analysis of that discourse or show that no analysis is applicable to it. Since discourse theories may significantly differ from each other, some of them may parse a given discourse, whereas others may fail to parse it. It might be also the case that two different theories, both can parse a given discourse, but they produce contrasting analyses of that discourse. Although both an analysis and an output of the analysis of a discourse depends on a particular approach, there are generic problems that the most of the theories consider in their discourse parsing tasks. One of them is identification of *basic* (atomic, minimal, elementary) discourse units (*idea units*) in a given discourse. Then, a problem is to find how these discourse units are related to each other. Another problem is to determine whether it is possible to group the related discourse units to create a larger discourse unit; how larger discourse units are related to other discourse units; when one has to stop the parsing process, etc. All in all, discourse parsing is a nontrivial task and the output of the discourse parsing task depends on a theory.

Therefore, while describing a particular approach to discourse parsing, one has to take into account the assumptions about the discourse structure of that approach.

1.2.2 Discourse Generation

The discourse generation problem is a part of a more generic problem of natural language generation (NLG). Since we identify a discourse with a monologic text, we focus on the part of NLG that is concerned with generating monologues.

NLG focuses on building such systems that can produce as understandable, coherent, texts in natural languages as humans do. As a rule, the starting point of an NLG system is some nonlinguistic representation of information. Such a representation is an input to the NLG system. By making use of knowledge about a natural language, NLG systems try to automatically produce natural language texts. To do that, NLG should be equipped with a considerable amount of linguistic knowledge at the levels of pragmatics, semantics, syntax, morphology, and phonology.

In addition to dealing with linguistic problems of text generation, NLG systems deal with the tasks such as information management, information selection, and information computing.

One of the first views that comes to a mind while explaining what NLG task is, is that NLG may be viewed as the opposite to the natural language understanding (analysis, parsing): While in natural language understanding the system needs to disambiguate the linguistic input (text, sentence) to produce the machine understandable output, in NLG the system needs to make decisions about how to put a concept into words, words into sentences, sentences into paragraphs, and paragraphs into texts. According to McKeown (1992), a text parsing system does not need to provide the reasons explaining why a particular choice is made in a text (for instance, why an active form of a verb is used instead of passive), whereas it is exactly what NLG systems are concerned with.

Thus, ideally, an NLG system should be able to justify why it made a certain choice, i.e., why the chosen one is one of the *best*[1] possible solutions. The following problems are the ones that NLG is particularly concerned with:

- Determining the content to be communicated;
- and representing already determined content by means of a natural language.

Dale (1995) gives a more elaborated view on the questions that NLG systems address:

1. *Deciding how much to say, and what not to say*:
    - Maintaining brevity;
    - avoiding stating the obvious.
2. *Designing text structure*:
    - May need to add material to the basic subject matter;
    - controlling the effects of the structure and ordering of the material;
    - making the text flow smoothly.
3. *Problems in carrying out a detailed text plan once built:*
    - Determining the sentence boundaries and the use of conjunctions;
    - deciding when to use anaphora;

[1]What is the *best* and *for what it is best* are in question as well.

- lexical selection (lexicalization);
- use of marked syntactic structures for particular rhetorical effects.

The tasks 1, 2, and 3 are interrelated, but for the sake of simplicity, in the most of the cases, they are assumed to be independent and thereby are solved separately.

1.3 An Example of a Text Generation System

EasyText is a fully-operational NLG system (Danlos, Frédéric Meunier, and Combet, 2011). It was developed for providing monthly reports for bank customers about their bank account activities. EasyText generates texts (in French) from tables that are filled with numerical data reflecting the account activities of the bank customers. For example, given the input shown in Figure 1.1, EasyText outputs the following text:

(5) *Dans ce secteur, les investissements ont doublé (+130%) pour la variété MULTIPROD.ORG.FINANCIERS en mai 2008 par rapport à mai 2007. Par ailleurs, les investissements pour la variété CREDIT PERSONNEL O.F marquent une progression de 6% pour le cumul à date étudié. Au contraire, pour la variété MULTIPROD.ORG.FINANCIERS, ils voient leur volume diminuer (-3%) sur la même période.*

In this sector, investments have doubled (+ 130%) for the variety MULTI-PROD.ORG.FINANCIERS in May 2008 compared to May 2007. In addition, investments for the variety CREDIT PERSONNEL O.F shows an increase of 6% accumulated in time studied. On the contrary, for the variety MULTI-PROD.ORG.FINANCIERS, they see decrease in their volume (-3%) over the same period.

The EasyText generation process can be viewed as a pipeline, consisting of the *content determination*, *document structuring*, and *tactical components*.

	Mai 2008	Mai 2009	Evol%	Cumul janvier à mai 2008	Cumul janvier à mai 2009	Evol%
ORGANISMES FINANCIERS	16 587	26 312	59 %	216 948	177 353	-18 %
CREDIT PERSONNEL O.F	5 868	11 227	91 %	50 610	53 772	6 %
~MULTIPROD.ORG.FINANCIERS	3 243	7 463	130 %	53 191	51 718	-3 %
CREDIT RENOUVELABLE O.F	3 930	1 994	-49 %	60 094	34 987	-42 %
INTERNET TELEMATIQUE 583	2 648	4 687	77 %	16 460	27 613	68 %
RACHAT DE CREDITS O.F	777	732	-6 %	15 817	5 637	-64 %
CREDIT AUTO MOTO O.F	79	110	39 %	5 638	993	-82 %
CREDIT TRAVAUX O.F		86		535	797	49 %
PARRAINAGE MECENAT O.F				80	0	-100 %

Figure 1.1: An example of an input table of Easyext, borrowed from (Danlos, Frédéric Meunier, and Combet, 2011)

The EasyText *content determination* component selects within a given table the *relevant cells* in terms of the information that has to be communicated. As Danlos, Frédéric

Meunier, and Combet (2011) note, no reasoning module is developed for this task; the rules for determining what are the relevant cells are rather hardcoded.

The EasyText *document structuring* component produces *conceptual representations.* This task involves building the discourse structure by relating the semantic content expressed by the cells in the table. For example, given two cells such that one shows an increase in the income, whereas the other one shows a significant decrease, within the same month, one concludes that these cells contain contrasting information. The document structuring component connects these two cells by the discourse relation CONTRAST. In EasyText, the principles of the discourse structure follow SDRT (Asher and Lascarides, 2003). The output of the document structuring component is a structured representation of concepts, where the structure is a discourse structure, and the concepts stand for events, discourse referents, etc.

The output of the document structuring task, i.e., a conceptual representation is then passed to the *tactical* component of EasyText, whose theoretical basis is G-TAG (Danlos, 1998, 2000). G-TAG makes a number of decisions, including ordering of the sentences, lexicalization, using syntactic constructions within sentences and for sentences, etc. Here, we will very briefly describe the G-TAG text generation, but in the further chapters, we will provide its detailed description.

G-TAG defines *lexical entries.* A lexical entry serves as a lexicalization of some concept. Each lexical entry is associated with a set of possible syntactic constructions with that lexical entry. The main process in G-TAG is lexicalization, that is, mapping of concepts to lexical entries. By lexicalizing all the concepts from the conceptual representation input, G-TAG produces a *g-derivation tree.* A g-derivation tree can be viewed as a semantic dependency tree, additionally decorated with syntactic information. It contains all the information needed for generating an output. A g-derivation tree specifies a unique *g-derived* tree, which can be seen as a syntactic analysis of a text. While a g-derived tree contains the morphological information for inflecting words, the words in it are not inflected. It is a post processing module of G-TAG that computes morphological information from the g-derived tree. Furthermore, it linearizes the tree and produces a text. In fact, the post processing module can do even more. It can produce a text and then modify it. The post processing module may issue the modified version of the original text as the output of the generation process, instead of the original text, which was directly obtained from the g-derived tree.

1.4 The Problems Considered in the Present Work

Discourse modeling is a complex problem, which one can study from various standpoints, with various motivations and goals. Ideally, one should take into account all aspects of a discourse to model it. However, it would be fair to say that this is one of the hardest problems in computational linguistics. Therefore, in this thesis, we only focus on the modeling of *the syntax-semantics interface for discourse*, which one also calls as the *syntax-discourse interface.*

The main goal of the present work is to design ACGs that enable one to consider the problem of discourse modeling. In particular, we aim at studying the ways one

can model the discourse-level phenomena in addition to sentence-level ones. Whether it is possible to integrate the sentence-level and the discourse-level phenomena at the same level of analysis is also a question that one has to answer. On the one hand, one knows that the border between the sentence-level and the discourse-level analyses is only conventional. For instance, in some cases, one can express the same meaning by a single sentence or with several sentences, i.e., by a discourse (text). This suggests that one already encounters discourse-level phenomena within sentences. On the other hand, there are certain phenomena that one does not observe in a single *clause* (idea unit). A clause itself does not need to be connected to something else in order to make sense. Therefore, one may still argue for the need of separation of the discourse-level analysis from the clause-level one. Thus, one can imagine (at least) the following scenarios of discourse modeling:

- Develop a unified framework for encoding the discourse-level and the clause-level phenomena.
- Develop a framework where the encodings of the discourse-level and the clause-level phenomena are provided by the separate modules. In this case, one has to give an account of how those modules interact with each other.

Although it seems more natural to have a unified framework for modeling both the discourse-level and clause-level phenomena, how plausible it is to build such a framework is yet another question that the present work has to answer.

In practical applications of computational linguistics, one aims to design tractable implementations, i.e., the ones that can be used to perform tasks by consuming a reasonable amount of resources. Thus, a golden mean between the linguistic adequacy and the computational cost of an approach is something that one has to establish.

Tasks such as content determination are not part of the problem of the syntax-semantics interface. In the present work, we do not deal with the tasks that are outside of the scope of the syntax-semantics interface problem. In addition, we do not study pragmatic problems, such as whether or why a given discourse is coherent/incoherent. As we have already mentioned, ideally, one aims to model all kinds of phenomena, and some theories even study some pragmatic phenomena within the syntax-discourse interface (Schlenker, 2011), but to our knowledge, no theory provides a fully functioning model that would enable one to encode pragmatic effects within the syntax-discourse interface.

Thus, in this thesis, we confine ourselves by studying the possibility of encoding the syntax-semantics interface for discourse with the help of ACGs. One of the goals of the present work is to construct such ACGs that allows one to consider problems of discourse parsing and generation. In other words, with the help of ACGs, we aim to encode the way text generation and parsing systems deal with discourses. At the same time, one of the objectives is to design tractable ACGs, i.e., such ACGs that one can implement in practical applications.

1.5 The Road Map of the Thesis

Part 1: Introduction

Chapter 1: We describe the subject matter of the thesis and some basic notions.

Chapter 2: We present some notions from formal language theory. We define notions related to formal grammars. We focus on Context-Free Grammars (CFGs) and Tree-Adjoining Grammars (TAGs). In addition, we briefly discuss Synchronous TAGs, which were introduced for modeling the syntax-semantics interface based on TAG grammars.

Chapter 3: We provide the definition of Abstract Categorial Grammars (ACGs), since ACGs serve as the main framework to the present work. As an example of ACGs, we illustrate how one encodes CFGs and TAGs as ACGs. In addition, we describe the ACG encoding of TAG with Montague semantics. The ACG encoding of TAG with Montague semantics enables one to address the problems of modeling of the syntax-semantics interface for sentences.

Chapter 4: We discuss the discourse theories. We focus on RST and SDRT. RST is a theory that studies organizational problems of texts. SDRT is a theory of dynamic semantics, that is, it interprets a discourse as a logical form. Unlike other dynamic theories, SDRT makes use of the pragmatic and semantic information encoded with the discourse structure.

Chapter 5: We present discourse formalisms that study discourse regularities with grammars. Namely, we explore the TAG-based formalisms, D-LTAG, G-TAG, and D-STAG. The grammars of these formalisms experience problems in dealing with certain kinds of texts. In particular, we focus on the problem that the grammars of these formalisms face in encoding discourses containing clause-medial connectives. We discuss the ways these formalisms choose in order to overcome the problem.

Part 2: Thesis Contributions

Chapter 1: We propose an encoding of G-TAG as ACGs. We show how one encodes the G-TAG grammar and its text generation process with the help of ACGs.

Chapter 2: We propose a method that one can make use of in TAG-based approaches to grammatically encode texts containing clause-medial connectives. In particular, we encode clause-medial connectives by extending the ACG encoding of G-TAG, which is proposed in the previous chapter. The resultant ACGs enable one to encode the texts containing clause-medial connectives in a purely grammatical manner.

Chapter 3: We encode D-STAG as ACGs. By adopting for D-STAG the method developed in the previous chapter for modeling clause-medial connectives, we construct the ACG encoding of D-STAG, which can model texts containing clause-medial connectives. In addition, we define another version of the ACG encoding of D-STAG where we define labeled semantic interpretations of discourses.

Chapter 4: We briefly discuss the related work to the presented one. We also present the questions with some solutions that one may investigate as a part of future research.

Chapter 5: We draw some conclusions about the work presented in the thesis.

Chapter 2

Formal Grammars

Contents

In this chapter, we discuss formal grammars. First, we present phrase-structure grammars (PSGs) and define the Chomsky hierarchy of grammars. In the Chomsky hierarchy, context-free grammars (CFGs) are able to capture various phenomena in natural languages. The parsing problem for them is of polynomial complexity. CFGs, however, cannot give a fully satisfactory account of certain phenomena in natural languages. To find the class of grammars adequate to describe the syntax of natural languages, a class of mildly-context sensitive grammar formalisms (MCSGs) was characterized. We focus on Tree-Adjoining Grammars (TAGs) from the class of MCSGs. TAGs overcome some of the problems that CFGs face in terms of expressive power. At the same time, the parsing problem for TAGs is of polynomial complexity. In addition, we present synchronous TAGs (STAGs), which were introduced with the goal of modeling the syntax-semantics interface with TAG grammars.

2.1 Overview

A *formal grammar* is a mathematical model that allows one to describe a set of admissible sentences of a language; under 'language' one may understand both an artificial (formal,

programming, etc.) and a natural language. In this thesis, we focus only on formal grammars that one employs to study natural languages. We may also refer to such formal grammars as *grammatical formalisms*.

A number of formal grammars have been proposed (e.g. (Ajdukiewicz, 1935; Bar-Hillel, 1953; Chomsky, 1956; A. K. Joshi, Levy, and Takahashi, 1975; Lambek, 1958)). One of them is known as the class of phrase-structure grammars (PSGs) (Chomsky, 1956). A phrase-structure grammar describes a set of sentences (strings) of a language. Four sub-classes are distinguished in the class of PSGs, called the *Chomsky hierarchy of grammars*. One of the sub-class of PSGs, called *context-free grammars* (CFGs), are expressive enough to encode a number of natural language phenomena. At the same time, for CFGs parsing algorithms of polynomial complexity are available.[2] Polynomial complexity is considered to be *feasible* for practical applications.[3] Although it is true that CFGs are capable of modeling a number of syntactic phenomena in natural languages, it has been argued that CFG cannot capture certain natural language phenomena (Shieber, 1985). In other words, CFGs are not *expressive* enough to describe all natural languages. Although the grammars from an upper class in the Chomsky hierarchy, called context-sensitive grammars (CSGs), can model natural languages, no algorithm of polynomial complexity can parse them.

A. K. Joshi (1985) proposed criteria to characterize grammatical formalisms that are necessary to describe all natural languages. He called the class of such formalisms *mildly context-sensitive grammars* (MCSGs). One of the criteria is that any formalism in the class has a polynomial parsing algorithm. Another criteria is that in this class some formalisms can describe certain kind of structures that CFGs cannot capture. The criteria for being a member of MCSGs are generalizations of the properties of the Tree-Adjoining Grammar (TAG) formalism (A. K. Joshi, Levy, and Takahashi, 1975). TAGs are more expressive than CFGs, but only *slightly* so that for them also exist parsing algorithms of polynomial complexity.

2.2 Preliminary Notions

We make use of standard mathematical notions[4] such as a function, relation, partial order, graph, Turing machine (algorithm), etc.

[2] A problem is *polynomial* (*of polynomial complexity*) if (1) an answer to it is either *yes* or *no*; (2) there exists a Turing machine (an algorithm) that requires the time that is *polynomial* to the length of the input in order to provide an answer to the problem (Goldreich, 2008).

[3] One refers to an algorithm as *feasible* if it is polynomial, as the following quote indicates:

> In most practical cases ...: polynomial-time algorithms are usually feasible, and non-polynomial-time algorithms are usually not feasible.
>
> Kreinovich, Lakeyev, Rohn, and Kahl (1998)

[4] For details, we refer readers to (Hopcroft, Motwani, and Ullman, 2006).

Alphabet, Word, Language

We denote by $\mathbb{N}$ the set of natural numbers ($\{0, 1, 2, \ldots\}$), whereas by $\mathbb{N}_+$, we denote the set of positive natural numbers ($\{1, 2, \ldots\}$). We fix the terminology and conventions by providing the definitions of the following notions:

Definition 2.2.1.

- *An* alphabet Σ *is a finite nonempty set of symbols.*
- *A* string *(*word*) is a finite sequence (possibly empty) of symbols of an alphabet* Σ.
- *The* empty string *(*empty word*), denoted with* ϵ*, is the string with zero occurrences of symbols (of any alphabet).*
- *Let* ω_1 *and* ω_2 *be two words over an alphabet* Σ*. Then* $\omega_1\omega_2$ *denotes the* concatenation *of* ω_1 *and* ω_2*, i.e.,* $\omega_1\omega_2$ *is the word over* Σ *obtained out of a copy of* ω_1 *followed by a copy of* ω_2*. One can check that* $\epsilon\omega = \omega\epsilon = \omega$ *for any string* ω*, where* ϵ *is the empty string.*
- *A number of* occurrences *of symbols in* ω *is called the* length *of* ω*; we denote the length of* ω *with* $\mathtt{len}(\omega)$.
- *If* Σ *is an alphabet,* Σ^k *denotes the set of strings over* Σ *of length* k*. Thus, the set of all strings over an alphabet* Σ*, denoted with* Σ^* *(Kleene star of* Σ*), can be defined as follows:*
$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cdots$$
We denote with Σ^+ *the set of non-empty strings of* Σ^**, hence:* $\Sigma^* = \{\epsilon\} \cup \Sigma^+$.
- *Given two languages* L_1 *and* L_2*, we define the* concatenation *of* L_1 *and* L_2*, denoted by* L_1L_2*, as follows:*
$$L_1L_2 = \{\omega_1\omega_2 \mid \omega_1 \in L_1 \text{ and } \omega_2 \in L_2\}$$

Convention: Lowercase letters of the Latin alphabet denote symbols; lowercase letters of the Greek alphabet denote strings (words); and capital letters of the Latin alphabet denote languages, unless otherwise stated.

Definition 2.2.2 (Prefix Order).
For an alphabet Σ*, one defines a partial order* $\leqslant$ *over* Σ^**, called the* prefix order*, as follows:*

$$\forall\, \omega_1, \omega_2 \in \Sigma^* : \omega_1 \leqslant \omega_2 \text{ if and only if } \exists \delta \in \Sigma^* :\ \omega_1\delta = \omega_2$$

If $\omega_1 \leqslant \omega_2$*, we say that* ω_1 *is a prefix of* ω_2.

It follows from Definition 2.2.2 that ϵ is a prefix for any string ω as $\epsilon\omega = \omega$.

Definition 2.2.3 (Lexicographic Order).
One defines the lexicographic order *over* $\mathbb{N}^*$*, denoted as* $\prec_l$*. Let* $\omega_1 = a_1 \ldots a_m$ *and* $\omega_2 = b_1 \ldots b_k$ *be two strings of* $\mathbb{N}^*$*, then* $\omega_1 \prec_l \omega_2$ *holds if and only if*

- *either* ω_1 *is a prefix of* ω_2 *or*
- $a_i < b_i$*, for some* $1 \leq i \leq \min\{\mathtt{len}(\omega_1), \mathtt{len}(\omega_2)\}$ *and* $a_j = b_j$ *for all* j *such that* $j < i$*, where* $<$ *is the standard linear order*[5] *over* $\mathbb{N}$.

[5]For $a, b \in \mathbb{N}$, $a < b$ if and only if $b = a + c$ for some $c \in \mathbb{N}_+$.

$\preceq_l$ denotes the reflexive closure of $\prec_l$.

Definition 2.2.4 (Tree Domain).
We call D a tree domain *if it is a finite subset of the set of strings over the set of positive natural numbers $\mathbb{N}_+$ (i.e., D is finite and $D \subseteq \mathbb{N}_+^*$) satisfying the following properties:*

1. *For any u and v, where $u, v \in \mathbb{N}_+^*$, if $uv \in D$, then $u \in D$.*
2. *For any string $u \in \mathbb{N}_+^*$ and a natural number $i \in \mathbb{N}_+$, if $ui \in D$, then for any $j \in \mathbb{N}_+$ such that $j < i$, $uj \in D$.*

We refer to elements of a tree domain as positions.

Labeled Trees

Definition 2.2.5 (Labeled Ordered Tree).
Let Σ be an alphabet. A labeled ordered tree is a pair $\gamma = \langle D, l \rangle$ where D is a tree domain and $l : D \longrightarrow \Sigma$ is a tree labeling function. Elements of Σ are called labels.

- *Given a labeled ordered tree $\gamma = \langle D, l \rangle$ and $d \in D$, the elements of D are called* nodes *in γ (or nodes of γ). Sometimes, they are also called as the (Gorn) addresses of γ. For a node d in γ, we say that $l(d)$ labels the node d. We call $l(d)$ the label of the node d.*
- *The* root *node r of the tree $\gamma = \langle D, l \rangle$ is the empty string ϵ.*
- *A node p is a frontier node if and only if $\forall j \in \mathbb{N}_+ : pj \notin D$. We also call frontier nodes as* leaves. *If a node in a tree is not a leaf, then we call it an* interior *or* internal *node.*
- *For an internal node n in the tree $\gamma = \langle D, l \rangle$, we say that n has k* daughters *(*children*) in γ if $\max_{ni \in D} i = k$. We call the nodes $n1, \ldots, nk$ in γ the* daughters *of n; we call the node n their mother (parent).*
  If $n1, \ldots, nk$ are daughters of n, we say that γ has k branches at the node n. If $k > 1$, the daughter nodes $n1, \ldots, nk$ of n are siblings.

$\mathbb{T}_\Sigma$ denotes the set of all trees whose nodes are labeled with symbols from Σ.

Hence, a node in a labeled ordered tree is a position of a tree domain. The label of a node is an element of an alphabet. Given the nodes $m_1, \ldots, m_l$ in a tree γ such that m_i is the mother of m_{i+1} for $i = 1, \ldots l-1$, we say that m_j is an *ancestor* of m_h if $1 \leq j \leq h \leq i$. In other words, if m and n are two nodes in a tree γ such that m is a prefix of n, then m is an ancestor of n in γ. We depict a tree as a graph where every node is connected only to its mother node (if any) and its daughter nodes (if any). We refer to the connection between a mother node and a daughter node as an *edge*. In the labeled tree, we decorate the nodes with their labels. Figure 2.1 shows an example of a labeled ordered tree.

Definition 2.2.6. *The yield of a labeled tree γ, denoted by $\texttt{yield}(\gamma)$, is a string over Σ^*, defined as follows:*

Let the set $\{\mathsf{n}_1, \cdots, \mathsf{n}_k\}$ be the set of frontier nodes of γ, where $\mathsf{n}_1, \cdots, \mathsf{n}_k$ occur in the lexicographic order, then the yield $\texttt{yield}(\gamma)$ is the string obtained by concatenating the labels of the frontier nodes, that is, $\texttt{yield}(\gamma) = l(\mathsf{n}_1) \cdots l(\mathsf{n}_l)$.

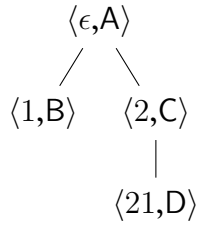


Figure 2.1: A pictorial representation of a labeled ordered tree

Tree Operations

We define two operations on trees, *substitution* and *adjunction* (A. K. Joshi, Levy, and Takahashi, 1975). To define them, it is useful to define *subtrees* and *supertrees* of a tree.

Definition 2.2.7 (Subtree and Supertree (A. K. Joshi, Levy, and Takahashi, 1975)). *For a tree $\gamma = \langle D, l\rangle$ and a node p, we define the* subtree *and the* supertree *at p as follows:*

Subtree $\gamma/p = \{\langle q, \mathsf{A}\rangle | \langle pq, \mathsf{A}\rangle \in \gamma, q \in \mathbb{N}^*\}$

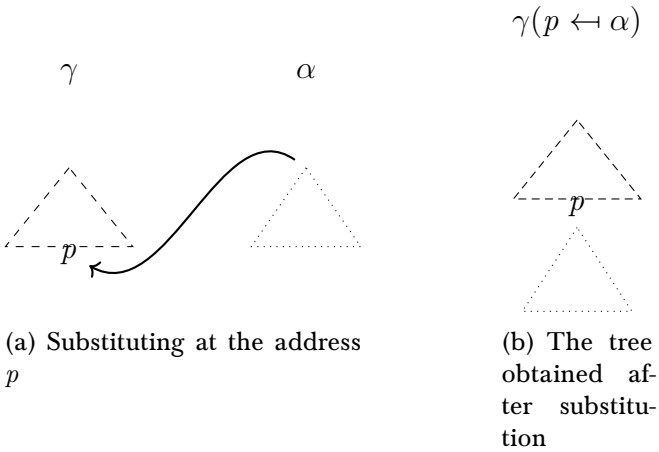
Supertree $\gamma\backslash p = \{\langle q, \mathsf{A}\rangle | \langle q, \mathsf{A}\rangle \in \gamma, p \not\leqslant q\}$

The supertree of a node is the set of its ancestors, its siblings and their ancestors. The subtree of a node is a set of nodes whose ancestor is the given node. For example, from Definition 2.2.7 follows that if p is the root node, i.e., $p = \epsilon$, then the subtree γ/ϵ is γ. The root node has no supertree, i.e., $\gamma\backslash\epsilon = \emptyset$. If p is a frontier node in γ, then the subtree at p is a tree consisting of p, i.e., $\gamma/p = \{p\}$.

Definition 2.2.8 (Substitution). *Let p be a frontier node in γ, and α be a tree. We define the substitution of α into γ at p, denoted by $\gamma(p \leftarrowtail \alpha)$, as follows:*

$$\gamma(p \leftarrowtail \alpha) \triangleq \gamma\backslash p \cup p\alpha$$

where $p\alpha = \{\langle pk, \mathsf{A}\rangle |\ \langle k, \mathsf{A}\rangle \in \alpha\}$.



(a) Substituting at the address p

(b) The tree obtained after substitution

Figure 2.2: The operation of substitution

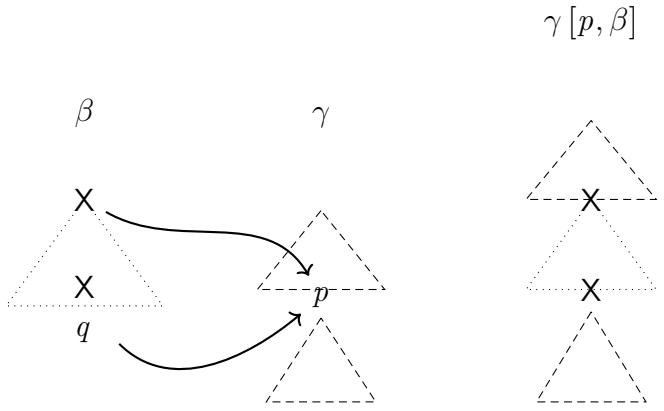
Figure 2.3 illustrates the substitution of a tree α in a tree γ at a node p, which, by definition, is a frontier node in γ.

Definition 2.2.9 (Adjunction). *Let β be a tree whose one of the frontier nodes is marked with the same label* X *as the root node. We call that frontier node the* foot *node of β. Let q be the foot node of β. Let γ be a tree and p be a node in γ. One defines the adjunction of the tree β into γ at p, denoted by $\gamma[p, \beta]$, as follows:*

$$\gamma[p, \beta] \triangleq \gamma \backslash p \cup p\beta \cup pq(\gamma/p)$$

where $p\beta = \{\langle pk, \mathsf{A}\rangle |\ \langle k, \mathsf{A}\rangle \in \beta\}$ *and* $pq(\gamma/p) = \{\langle pqk, \mathsf{A}\rangle | \langle k, \mathsf{A}\rangle \in \gamma/p\}$.

Figure 2.3 illustrates the adjunction of β into γ.[6]



(a) β adjoins on the node X into γ

(b) The tree obtained after adjunction

Figure 2.3: The operation of adjunction

Ranked Trees

We have defined labeled (ordered) trees by purely set-theoretic means. One can use another definition of a tree by representing it as a *term.*

Definition 2.2.10 (Ranked Alphabet).
A ranked *alphabet is a pair $\langle \Delta, \rho \rangle$, where Δ is a set and ρ is a function mapping each element of Δ to a natural number. We call Δ an alphabet. We say that $f \in \Delta$ is a symbol of the rank (or of arity) $\rho(f)$, or f is a $\rho(f)$-ary symbol.*

By convention, if f is of arity $n > 0$, we may write f_n; if f is of arity 0, we write f.

Usually, instead of $\langle \Delta, \rho \rangle$, we write Δ if it does not cause a confusion. We also define subsets of Δ as follows: $\Delta_n = \rho^{-1}(n) = \{f \in \Delta \,|\, \rho(f) = n\}$, where $n \geq 0$. Thus, we may write $f \in \Delta_n$ to express that f belongs to the alphabet Δ and the rank of f is n.

[6]One can consider a more generic notion of adjunction than the current one provided by Definition2.2.9. Namely, one can lift the restriction imposed on β about having one of the frontier nodes marked with the same label X as the root node. Nevertheless, we only consider the current version of adjunction as it fits the formalisms that we will discuss below.

Definition 2.2.11 (Ranked Tree).
Given a tree domain D and a ranked alphabet $\langle \Delta, \rho \rangle$*, we define a* ranked tree *as a pair $\langle D, l \rangle$ where $l : D \to \Delta$ is a labeling function so that the following holds:*
For any $p \in D$, if $\rho(l(p)) = m$, then $pm \in D$ and for any $k > m$, $km \notin D$.

By definition, a ranked tree is a labeled ordered tree. Conversely, one can transform a labeled ordered tree into a ranked tree. Indeed, by assigning to the labels of the nodes in a labeled tree the ranks so that the requirement of Definition 2.2.11 is full-filled, one obtains the ranked tree representation of that labeled tree. In particular, if p is a node of a labeled ordered tree, then we assign to the label $l(p)$ the rank $\max_{pi \in D} i$, i.e., the number of *branches* in the tree at the node p. However, notice that to one and the same label, we may have to assign several ranks. If a label f appears at several different tree addresses, then we may need to introduce several copies of f. For instance, assume that f appears at $m > 1$ tree addresses. Let at these tree addresses, the tree has branches $k_1, \ldots, k_m$ such that $k_i \neq k_j$ for any $1 \leq i, j \leq m$. We introduce m *copies* of the label f with the ranks $k_1, \ldots, k_m$. That is, we introduce the following symbols: $f_{k_1}, \cdots, f_{k_m}$.

It is convenient to use term representations of ranked trees. Since every node of a ranked tree $\langle D, l \rangle$ is coupled with a symbol of a ranked alphabet, one can equivalently define a ranked tree as a term.

Definition 2.2.12 (Term).
The set of terms over the ranked alphabet Δ, denoted by $\mathbb{T}_\Delta$, is the smallest set satisfying the following conditions:
1. *If f is of the rank 0, then $f \in \mathbb{T}_\Delta$;*
2. *if $f \in \Delta_n$, where $n \geq 1$, and $t^1, \ldots, t^n$ belong to $\mathbb{T}_\Delta$, then $f(t^1, \cdots, t^n) \in \mathbb{T}_\Delta$.*

Every term can be represented as a ranked tree whose frontier nodes are labeled with symbols of arity 0; each internal node is labeled with a symbol of positive arity that equals to the number of branches at that node. In the rest of this thesis, we will not specify whether we discuss ranked or unranked trees if it does not cause a confusion. We will also forget the difference between ranked trees and terms. While depicting a tree $\langle D, l \rangle$, usually, we will only depict labels of nodes, i.e., we will omit d in $\langle d, l \rangle$, unless otherwise stated.[7]

2.3 Phrase Structure Grammars

Phrase-structure grammars (PSGs) (Chomsky, 1956) is a class of formal grammars. PSGs were inspired by Bloomfield's (1933) linguistic notion of *constituents*, which allow one to analyze natural language expressions by determining their *constituent* structures.[8] Let us provide definitions of a phrase-structure grammar (PSG), a phrase-structure derivation, and a language defined by a phrase-structure grammar.

[7]Since from a pictorial representation of a tree, one can unequivocally reconstruct the tree addresses, one can omit them while depicting a tree.

[8]Because of this, PSGs are also known as *constituency grammars*.

Definition 2.3.1 (Phrase-Structure Grammars (Chomsky, 1956)).
A phrase-structure grammar (PSG) is a quadruple $G = \langle N, \Sigma, P, \mathsf{S} \rangle$*, where*

- N *is a finite set of symbols called the* non-terminal *symbols;*
- Σ *is a set of symbols called the* terminal *symbols such that* $\Sigma \cap N = \emptyset$*;*
- $\mathsf{S} \in N$ *is a symbol called the* start *(*initial, distinguished*) symbol;*
- $P \subseteq (\Sigma \cup N)^+ \times (\Sigma \cup N)^*$ *is a finite set of* production *(*rewrite*) rules.*
  By convention, for $p = \langle \gamma,\ \delta \rangle \in P$*, we write* $\gamma \to \delta$*.*

We use lower case symbols of the Latin alphabet ($a, b, \ldots$) to denote symbols of Σ (terminal symbols), where for non-terminals symbols, i.e., elements of N, we use capital letter symbols ($\mathsf{A}, \mathsf{B}, \ldots$). To denote a string of terminals and non-terminal symbols, i.e., a string over $\Sigma \cup N$, we use a lower case symbol of the Greek alphabet.

Definition 2.3.2 (One-step Derivation). *Given a PSG* $G = \langle N, \Sigma, P, \mathsf{S} \rangle$*, the one-step derivation* $\Longrightarrow_G$ *is the binary relation over* $(\Sigma \cup N)^*$*, defined as follows:* $\alpha \Longrightarrow_G \beta$ *holds if and only if there are* $\delta_1 \in (\Sigma \cup N)^*$*,* $\delta_2 \in (\Sigma \cup N)^*$*, and* $p \in P$*, where* $p = (\mu_1 \to \mu_2)$*, such that*

$$\alpha = \delta_1 \mu_1 \delta_2 \qquad \textit{and} \qquad \beta = \delta_1 \mu_2 \delta_2$$

Definition 2.3.3 (Derivation and Generated Language). *Given a PSG* $G = \langle N, \Sigma, P, S \rangle$*, the derivation relation* $\Longrightarrow_G^*$ *is the reflexive and transitive closure of* $\Longrightarrow_G$*.*

The language generated by G *is a set* L *defined as follows:*

$$L = \{\alpha \mid \mathsf{S} \Longrightarrow_G^* \alpha\}$$

2.3.1 The Chomsky Hierarchy of Grammars

By constraining rewriting rules of a PSG, one can define various classes of PSGs. Chomsky (1956) determines three proper sub-classes of PSGs, provided within Definition 2.3.4.

Definition 2.3.4 (Four Types of Grammars).

Type-0 *A PSG* $G = \langle N, \Sigma, P, \mathsf{S} \rangle$ *is called* type-0*, or* unrestricted*, if each of its production rules* $p \in P$ *has the form* $\alpha \to \beta$*, where* $\alpha \in (\Sigma \cup N)^+$ *and* $\beta \in (\Sigma \cup N)^*$*, or equivalently, if production rules of* G *are unrestricted, then* G *is a* type-0 *grammar.*

Type-1 *A PSG* $G = \langle N, \Sigma, P, \mathsf{S} \rangle$ *is called* type-1*, or* context-sensitive*, if each of its production rules* $p \in P$ *is either of the form* $\mathsf{S} \to \epsilon$*, or* $\alpha \mathsf{A} \beta \to \alpha \mu \beta$*, where* $\alpha, \beta \in (\Sigma \cup N)^*$*;* $\mu \in (\Sigma \cup N)^+$*; and* $\mathsf{A} \in N$*.*

Type-2 *A PSG* $G = \langle N, \Sigma, P, \mathsf{S} \rangle$ *is called* type-2*, or* context-free*, if each of its production rules* $p \in P$ *has the form* $\mathsf{A} \to \omega$*, where* $\mathsf{A} \in N$ *and* $\omega \in (\Sigma \cup N)^*$

Type-3 *A PSG* $G = \langle V, \Sigma, P, \mathsf{S} \rangle$ *is called* type-3*, or* regular*, if each of its production rules* $p \in P$ *is either of the following forms* $\mathsf{S} \to \epsilon$*,* $\mathsf{A} \to a$*, or* $\mathsf{A} \to a\mathsf{B}$*, where* $\mathsf{A}, \mathsf{B} \in N$*, and* $a \in \Sigma$*.*

An immediate consequence of Definition 2.3.4 are the following inclusions (Chomsky, 1956):

$$\{\textit{Type-0 Grammars}\} \supset \{\textit{Type-1 Grammars}\} \supset \{\textit{Type-2 Grammars}\} \supset \{\textit{Type-3 Grammars}\} \tag{2.1}$$

The inclusions in (2.1) are known as *the Chomsky hierarchy.* Type-0 grammars generate exactly *recursively enumerable* languages (the languages that Turing machines accept). Thus, one only knows that if a string ω belongs to a language L_G generated/accepted by a type-0 grammar G, then there is a Turing machine M that halts in a final state. If $\omega \notin L_G$, then M halts in a non-final state or does not halt at all, i.e., loops forever. Thus, the question whether a $\omega \in L_G$ holds is undecidable. That is why one does not make use of type-0 grammars in practical applications.

The next class in the Chomsky hierarchy is Type-1, also known as the class of context-sensitive grammars. The problem whether a given string belongs to the language generated/accepted by a context-sensitive grammar is PSPACE-complete.[9] The class of CFGs is the next class in the Chomsky hierarchy. Importantly, for a context-free language (CFL), i.e., for the language generated by a CFG, there are several polynomial parsing algorithms.

2.3.2 Context-Free Grammars

Given a CFG $G = \langle N, \Sigma, P, \mathsf{S}\rangle$, one defines a set of *derivation* trees associated with G.

Definition 2.3.5 (CFG Derivation Tree).
For a CFG grammar $G = \langle V, \Sigma, P, \mathsf{S}\rangle$, we define a derivation tree as follows:

1. *Every node of a derivation tree has a label (either a terminal or a non-terminal symbol).*
2. *Any interior node is labeled with a non-terminal symbol.*
3. *Each frontier node is labeled by either a non-terminal or a terminal symbol, or ϵ. If ϵ labels a frontier node, then it must be the only child of its mother.*
4. *If nodes $n_1, \ldots, n_m$ (ordered according to the lexicographic order) are mutually distinct daughters of a node n with labels $\mathsf{A}_1, \ldots, \mathsf{A}_m$ respectively (i.e., $l(n_i)) = \mathsf{A}_i$ for $i = 1, \ldots, n$), and the label of n is A, then $\mathsf{A} \rightarrow \mathsf{A}_1 \ldots \mathsf{A}_m$ is a production rule of G.*

With the help of derivation trees, we define *parse trees.*[10]

Definition 2.3.6 (Parse Tree). *We refer to a derivation tree t of a CFG G as a parse tree if each frontier node of t is labeled with either a terminal symbol or ϵ. We denote a set of parse trees of G with $\mathtt{PTR}(G)$, whereas we denote with $\mathtt{PTR}(G, \mathsf{A})$ the set of parse trees whose root is labeled with A.*

Theorem 2.3.1. *Let a string of terminals ω belongs to the language of a CFG G, then there is a parse tree with the root labeled by S whose yield is ω. That is, if $\omega \in L_G$, then there exists $t \in \mathtt{PTR}(G, \mathsf{S})$ such that $\mathtt{yield}(t) = \omega$. Conversely, if a parse tree of a grammar G has the root labeled by S and its yield is ω, then ω belongs to the language generated by G.*

[9] If a problem can be solved by an algorithm that uses an amount of space that is polynomial to the size of its input, the problem is said to be in the class PSPACE. A problem is PSPACE-complete if any other problem that can be solved in polynomial space can be transformed to it in polynomial time (for more details, we refer readers to (Goldreich, 2008)).

[10] While some authors do not make a distinction between derivation trees and parse trees, we follow (Nijholt, 1980), which defines a parse tree as a derivation tree of a specific kind.

Hence, Theorem 2.3.1 states that the set of parse trees of L_G, i.e., the parse trees whose yields are in L_G coincides with the set $\texttt{PTR}(G, \mathsf{S})$. We may refer to $\texttt{PTR}(G, \mathsf{S})$ as the set of parse trees generated by G.

Example 2.1.

(2) Fred is grumpy because he failed an exam.

Figure 2.4 shows a CFG generating/accepting the sentence (2). Figure 2.5 illustrates the derivation tree of the sentence (2).

S	$\rightarrow$ NP VP
S	$\rightarrow$ S Conjunction S
VP	$\rightarrow$ V NP
VP	$\rightarrow$ Aux Adjective
NP	$\rightarrow$ Propername
NP	$\rightarrow$ Pronoun
NP	$\rightarrow$ Det N
Propername	$\rightarrow$ *fred*
Pronoun	$\rightarrow$ *he*
Adjective	$\rightarrow$ *grumpy*
Aux	$\rightarrow$ *is*
Det	$\rightarrow$ *an*
N	$\rightarrow$ *exam*
V	$\rightarrow$ *failed*
Conjunction	$\rightarrow$ *because*

Figure 2.4: An example of a context-free grammar

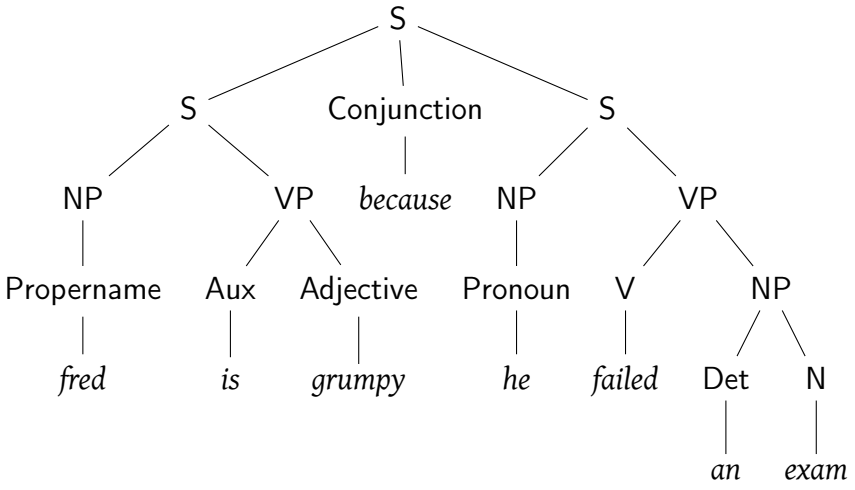


Figure 2.5: A CFG parse tree

For parsing purposes, it is beneficial to *simplify* CFG rules without weakening its expressive power. Given a CFG G, one aims to transform G to some $G^{'}$ so that $G^{'}$ generates the same language as G but the production rules of $G^{'}$ follow certain patterns. Definition 2.3.7 and Definition 2.3.8 present *Chomsky Normal Form* (CNF) and *Greibach Normal Form* (GNF), respectively.

Definition 2.3.7 (Chomsky Normal Form). *A CFG is in Chomsky Normal Form (CNF) if its production rules have either of the following forms:*

- $\mathsf{A} \rightarrow \mathsf{BC}$
- $\mathsf{A} \rightarrow a$
- $\mathsf{S} \rightarrow \epsilon$

Where $\mathsf{A}, \mathsf{B},$ *and* C *are nonterminals;* S *is the start symbol;* a *is a terminal symbol, and* ϵ *is the empty string.*

Theorem 2.3.2 (Chomsky, 1959). *Any context-free language is generated by some CFG in Chomsky Normal Form.*

Definition 2.3.8 (Greibach Normal Form). *A CFG is in Greibach Normal Form (CNF) if its production rules have either of the following forms:*

- $\mathsf{A} \rightarrow a\mathsf{B}_1 \cdots \mathsf{B}_k$
- $\mathsf{A} \rightarrow a$
- $\mathsf{S} \rightarrow \epsilon$

Where A *and* B_i*, for* $i = 1, \ldots, k$ $(k \geq 1)$*, are nonterminals;* S *is the start symbol;* a *is a terminal symbol; and* ϵ *is the empty string.*

Theorem 2.3.3 (Greibach, 1965). *Any context-free language is generated by some context-free grammar in Greibach Normal Form.*

If G is a CFG in CNF, then Cocke-Younger-Kasami (CYK) algorithm[11] parses an input of size n in $O(n^3)$ time. A CFG in GNF has a property that each production rule contains a terminal symbol. To parse a string of terminals with a CFG in GNF, one only selects those production rules that contain a symbol in the string. A normalization of a CFG has its trade-offs, however. In particular, converting a CFG in CNF and GNF forms may drastically increase the size of the original set of production rules.

2.4 Regular Tree Grammars

Regular Tree Grammars (RTGs) (Brainerd, 1969) is a *tree* generating formalism whose production rules can be seen as production rules of CFGs. In particular, one can obtain the set of parse trees of the language generated by a CFG as the *tree language* generated by some regular tree grammar.

Definition 2.4.1 (Regular Tree Grammar (Brainerd, 1969)).
A regular tree grammar (RTG) is a quadruple $G = \langle N, \Sigma, P, \mathsf{S} \rangle$*, where:*

[11]For more details, one can refer to (Jurafsky and Martin, 2000).

- N *is a finite set of non-terminal symbols of rank* 0*;*
- Σ *is a finite ranked alphabet;*
- P *is a finite set of productions* $A \to t$*,* $A \in N$ *and* $t \in \mathbb{T}_{N \cup \Sigma}$*;*
- $\mathsf{S} \in N$ *is a non-terminal symbol, called the distinguished symbol of the grammar.*

Definition 2.4.2. *A derivation step* $s \Longrightarrow s^{'}$ *with* $s, s^{'} \in \mathbb{T}_{N \cup \Sigma}$ *is obtained by selecting an occurrence of a non-terminal* A *(by definition of rank* 0*) in* s *and a production* $\mathsf{A} \to t$ *in* P *and constructing* $s^{'}$ *from* s *by replacing the selected occurrence of* A *with* t*. One defines the language* $L(G)$ *determined by an RTG* G *as follows:*

$$L(G) = \{s \in \mathbb{T}_{\Sigma} \mid \mathsf{S} \Longrightarrow^{*} s\}$$

Definition 2.4.3. *Given an RTG* G *with the tree language* $G(L)$*, we can define the string language determined by* G *as follows:*

$$\texttt{yield}(G) = \{\omega \mid \exists t \in L(G) : \texttt{yield}(t) = \omega\}$$

Theorem 2.4.1 (Brainerd, 1969).
- *If* G *is a context-free grammar, then the set of parse trees of* L_G *is a regular tree language.*
- *If* L *is a regular tree language then* $\texttt{yield}(L)$ *is a context-free language.*

Definition 2.4.4. *(Generative Capacity and Equivalence of Grammars)*
- *The* weak generative capacity (WGC) *of a grammar is the set of* strings *that a grammar generates, that is, WGC is the string language defined by the grammar.*
  We say that two grammars are weakly equivalent *if and only if they define the same set of strings.*
- *The* strong generative capacity (SGC) *of a grammar is the set of* structural descriptions *(*syntactic analyses*) that the grammar generates.*
  We say that two grammars are strongly equivalent *if and only if they define the same set of* structural descriptions.

In Definition 2.4.4, under 'structural description', one means syntactic trees. If two grammars are strongly equivalent, then they are weakly equivalent too because the set of structural descriptions defined by a grammar uniquely determines the string language defined by that grammar. In this terminology, a claim that Theorem 2.4.1 makes is that for any RTG, there exists its weakly equivalent CFG.

2.5 Mildly-Context Sensitivity

A. K. Joshi (1985) proposed the class of *mildly context-sensitive grammars* (MCSGs) with the aim to determine *the class of grammars that is necessary for describing natural languages.* We refer to a language as a *mildly context-sensitive language* (MCSL) if it is a language defined by an MCSG. The class of MCSGs (at least) possesses the following properties (A. K. Joshi, 1985):

1. MCSLs contain CFLs as a proper sub-class.
2. An MCSL has a polynomial parsing algorithm.

3. MCSGs are only slightly more powerful than CFGs. Namely, an MCSG can capture (at least) the following two kinds of dependencies:

   (a) Limited *nested* dependencies, illustrated by the following German sentence:

   (3) *Hans Peter Marie schwimmen lassen sah.*
   Hans Peter Marie swim make saw

   'Hans saw Peter make Marie swim.'

   (b) Limited *crossing* dependencies, illustrated by the following Swiss-German sentence:[12]

   (4) *... mer d'chind em Hans es huss lönd hälfe aastriiche.*
   ... we the children$_{\text{ACC}}$ Hans$_{\text{DAT}}$ the house$_{\text{ACC}}$ let help paint

   '... we let the children help Hans paint the house'

4. An MCSL has the *constant growth property*, that is, for a given MCSL L, there is a constant c such that for each $\omega \in L$, there is $\omega^{'} \in L$, such that $\texttt{len}(\omega^{'}) < \texttt{len}(\omega) < \texttt{len}(\omega^{'}) + c$.

2.6 Tree-Adjoining Grammars

In the tradition of phrase-structure grammars, A. K. Joshi, Levy, and Takahashi (1975) introduced the Tree-Adjoining Grammar (TAG) formalism. Like CFGs, TAGs also make use of rewriting in order to analyze/generate a sentence. However, while for a CFG the object of rewriting is a string, TAGs rewrite trees into trees. TAGs are capable of encoding both nested dependencies and Swiss-German cross-serial dependencies. Thus, TAGs are more expressive than CFGs. However, the expressive power of TAGs is only *slightly* greater than the one of CFGs so that the parsing complexity of a TAG is $O(n^6)$ (Schabes and A. K. Joshi, 1988). String languages determined by tree languages generated by TAGs have the constant growth property. Thus, TAGs belong the class of MCSGs.

2.6.1 Basic Notions and Properties

Tree-Adjoining Grammar (TAG) (A. K. Joshi, Levy, and Takahashi, 1975; A. K. Joshi and Schabes, 1997) is a tree generating formalism. It generates (derives) trees by rewriting trees into trees. In a TAG, *elementary trees* serve as a starting point for deriving a tree. Sometimes, one refers to elementary trees as *grammar entries.* Among elementary trees, one distinguishes two kinds of trees, *initial* and *auxiliary* trees. TAG defines two operations on trees, *substitution* and *adjunction.* While in a CFG derivation step, one rewrites a non-terminal symbol with the help of a production rule, in a TAG derivation step, one either substitutes or adjoins a tree into a tree. A resultant tree of a TAG

[12]Although CFGs can capture limited nested dependencies (A. K. Joshi, 1994), as Shieber (1985) shows, the pattern of Swiss-German limited crossing dependencies cannot be described by CFGs.

derivation step is called a *derived* tree. That is, a tree constructed by combining (either by substitution or adjunction) two trees is a *derived* tree. In TAG, one defines the notion of the *derivation* tree of a derived tree. The derivation tree of a given derived tree represents (records) the information *how the derived tree was built.* Thus, given a derivation tree, one can reconstruct the derived tree.

Definition 2.6.1 (Tree-Adjoining Grammar (A. K. Joshi and Schabes, 1997)).
A TAG is a quintuple $\langle N, \Sigma, I, A, \mathsf{S}\rangle$*, where*

1. N *is a finite set of non-terminal symbols.*
2. Σ *is a finite set of terminal symbols.*
3. S *is a distinguished non-terminal symbol.*
4. I *is a finite set of finite trees, called* initial *trees. One identifies an initial tree by the following properties:*
   - *its internal nodes are labeled with non-terminal symbols;*
   - *its frontier nodes are either labeled with terminal symbols, or non-terminal ones, which are marked for* substitution *with* $\downarrow$.
5. A *is a finite set of finite trees, called* auxiliary *trees. One identifies an auxiliary tree by the following properties:*
   - *its internal nodes are labeled by non-terminals symbols;*
   - *its frontier nodes are labeled by terminals, or by non-terminal nodes marked for substitution, except for exactly one non-terminal node that is labeled with the same label as the* root *node; this node is marked with* * *and referred to as the* foot *node of the auxiliary tree.*

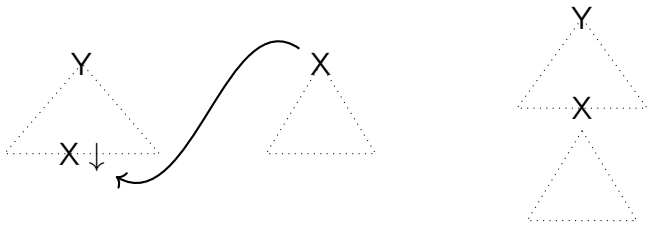
We define substitution of a tree into another one as in Definition 2.2.8 on page 29 but with an additional requirement that if α substitutes into γ at the node p, then the root of α and p should have the same label. Moreover, α must be derived from an initial tree, that is, (1) either α is an initial tree with no substitution sites; (2) or α is obtained from an initial tree by filling its substitution sites with some trees. Figure 2.6 illustrates the adjunction of a tree into another tree.

Adjunction is defined in the same way as in Definition 2.2.9 on page 30 but with the following additional requirement: β can adjoin into γ at the node p, where p is not marked for substitution, if and only if β is an auxiliary tree or derived from an auxiliary tree and the label of the root (and therefore of the foot) node of β coincides with label of p. Figure 2.7 illustrates the adjunction of a tree into another tree.

Convention: We call an initial (resp. auxiliary, derived) tree an X-initial (resp. auxiliary, derived) if its root is labeled with a non-terminal X. We refer to a node labeled with X as an X-substitution site if it is marked for substitution. If a node labeled with X is not marked for substitution, then it is an adjunction site, which we may refer to as X-adjunction site.

Definition 2.6.2 (Completed Tree (A. K. Joshi and Schabes, 1997)).
A tree (either initial or derived) is considered completed *if its frontier is made of nodes labeled by terminal symbols only.*

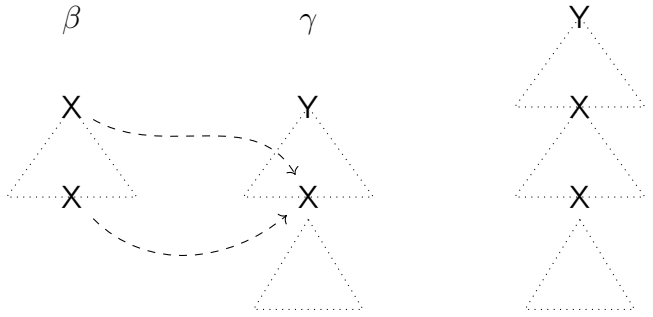
We may refer to a completed tree as a derived tree, unless we do not need to underline that it is a completed tree but not any other derived tree.



(a) Substituting at the node X

(b) The tree obtained after substitution

Figure 2.6: The TAG operation of substitution



(a) β adjoins on the node X into γ

(b) The tree obtained after adjunction

Figure 2.7: The TAG operation of adjunction

Definition 2.6.3 (TAG Tree and String Languages (A. K. Joshi and Schabes, 1997)). *One defines the tree and string languages determined by a TAG G as follows:*

- *T_G denotes the set of completed trees of G derived from S-initial trees, that is, we have:*

$$T_G = \{t \mid t \textit{ is completed tree derived from some S-initial tree}\}$$

- *One defines the string language of G, denoted by L_G, as follows:*

$$L_G = \{w \mid \exists\, t \in T_G \quad \textit{such that} \quad \texttt{yield}(t) = w\}$$

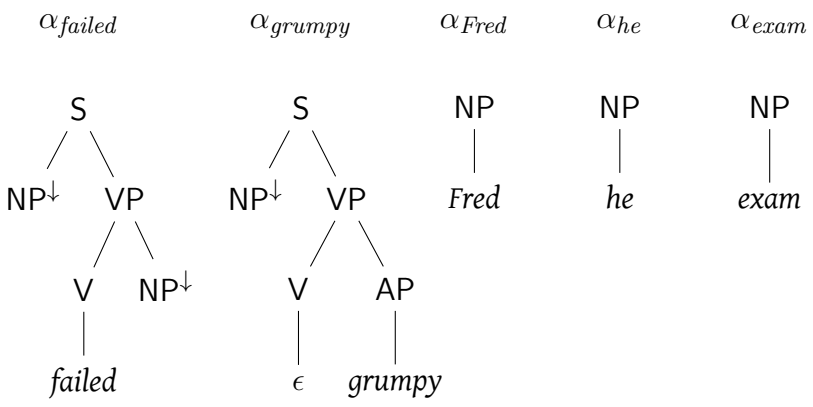
Sometimes, we may say a *derived (parse) tree of a sentence*, under which we mean a derived tree whose yield is the given sentence.

Example 2.2.

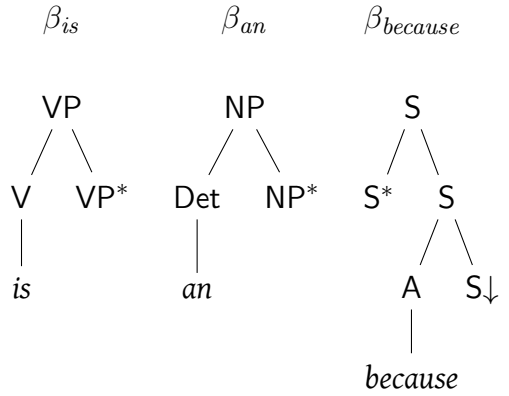
(5) Fred is grumpy because he failed an exam.

Let our TAG grammar consist of the elementary trees depicted in Figure 2.8.[13] To obtain the derived tree of the sentence (5), one combines these elementary trees as it

[13]The grammar follows the principles of X-TAG (XTAG-Group, 1998).



(a) Initial trees



(b) Auxiliary trees

Figure 2.8: TAG elementary trees

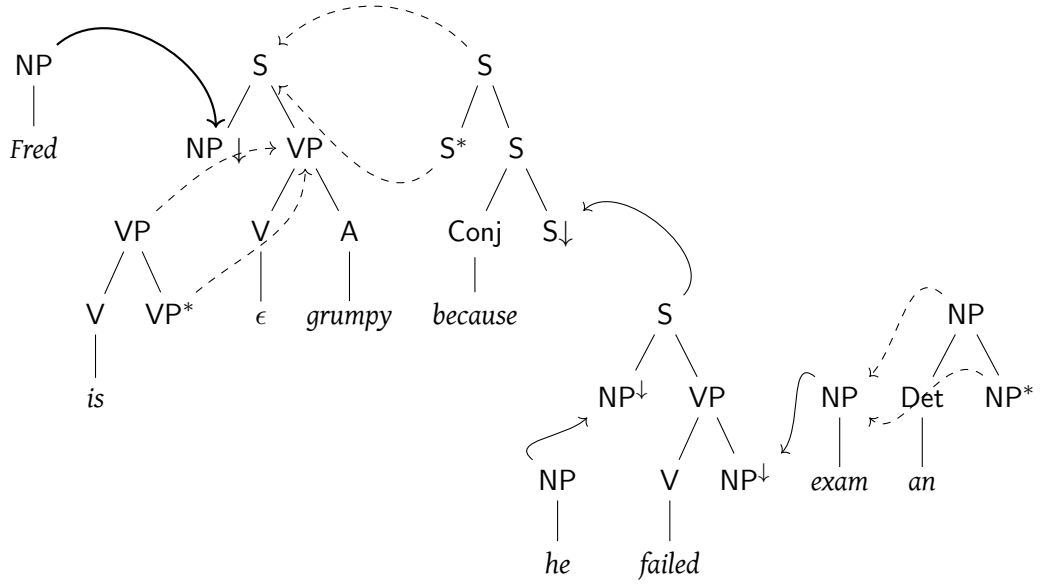
is shown in Figure 2.9(a) (we use dashed lines to illustrate adjunction, and solid lines for substitution). In result, one constructs the derived tree depicted in Figure 2.9(b).

Adjoining Constraints

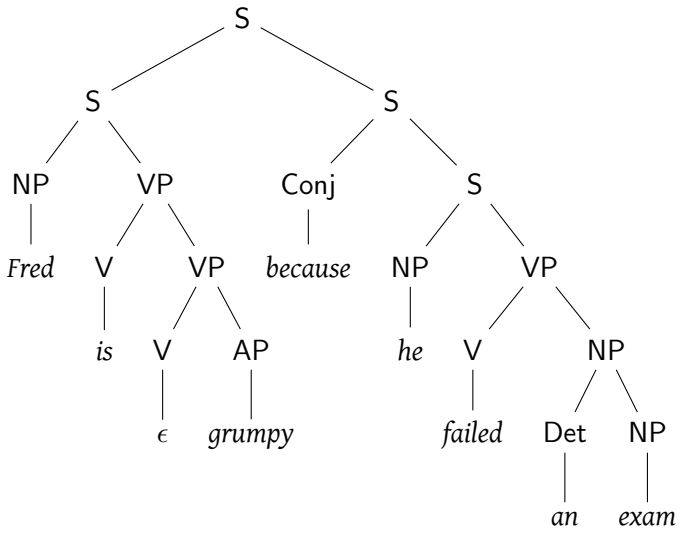
In a TAG, any non-terminal symbol that is not marked for substitution is an adjunction site. This may cause over-generation. For instance, using the trees in Figure 2.8, one can adjoin β_{an} into α_{Fred} so that one obtains an NP rooted tree with the yield *an Fred*. To control the derived structures of a TAG, one defines *adjoining constrains*. Namely, one makes use of the following adjoining constrains:

- Selective Adjunction (SA): Only auxiliary trees from a set of auxiliary trees $T \subseteq A$ can adjoin on a given node in a tree. Adjunction of these auxiliary trees is not mandatory.
- Null Adjunction (NA): No adjunction is allowed on a given node in a tree.
- Obligatory Adjunction (OA): An auxiliary tree from a set of auxiliary trees $T \subseteq A$ must be adjoined on a given node in a tree.

By convention, one indicates an adjunction constraint (if any) with the help of a subscript on a node label, $\mathsf{X}(T)_{NA}$, $\mathsf{X}(T)_{OA}$, and $\mathsf{X}(T)_{SA}$. If no set T is specified for



(a) TAG elementary trees



(b) A completed tree derived from S-rooted initial tree

Figure 2.9: Derivation of a (completed) derived tree

the adjunction site X, we assume that T is the set of X-auxiliary trees, unless otherwise stated.

Derivation Trees

By using Gorn addresses, one can indicate the site where an operation (either a substitution of adjunction) is applied in a derivation step. With this in mind, one can construct a *derivation* tree of a derived tree as follows:

- Labels of the nodes in the derivation tree stand for the employed elementary trees.

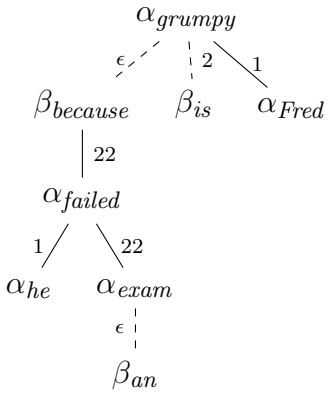


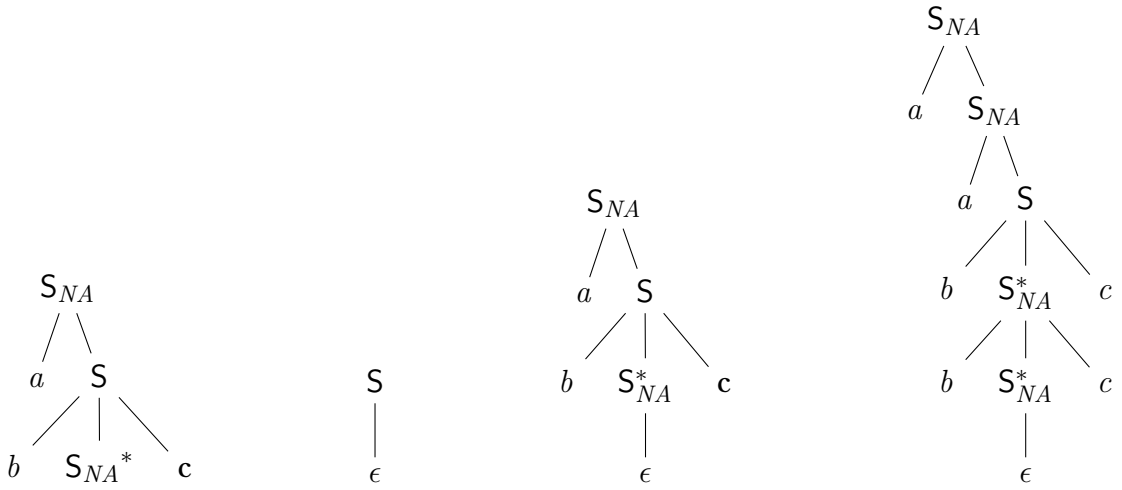
Figure 2.10: A derivation tree

- If γ substitutes (resp. adjoins) into δ, then in the derivation tree a node labeled with γ is a child of a node labeled with δ. We put a solid (resp. dashed) edge in the derivation tree between the nodes labeled with γ and δ. We label this edge with the Gorn address in δ that serves as the substitution (resp. adjunction) site at which γ substitutes (resp. adjoins) into δ.

For instance, the tree shown in Figure 2.10 records derivation of the derived tree shown in Figure 2.9(b) on the preceding page.

Example 2.3. Let us consider a TAG grammar that allows one to describe (limited) cross-serial dependencies. With the help of the adjunction constraints, it becomes possible to generate the tree language whose yield is the string language $\{a^n b^n c^n\}$, which no CGF can generate.

To generate $\{a^n b^n c^n\}$, one considers the grammar consisting of one auxiliary `A` and one initial tree `I` depicted in Figure 2.11 on the next page. The auxiliary tree `A` has only one available adjunction site (at the Gorn address 2). Since `A` is an S-auxiliary tree, `A` can adjoin into a tree with an S-adjunction site. The initial tree `I` has an S-adjunction site, which at the same time serves as its root node. If we adjoin the tree `A` on the S-node into the tree `I`, we obtain a derived tree shown in Figure 2.11(c) on the facing page, denoted with `AI`. The yield of the tree `AI` is $ab\epsilon c$. In order to obtain a derived tree whose yield is $aabb\epsilon cc$, one adjoins `A` into `AI`. Thanks to adjunction constraints, `AI` has only one adjunction site. By adjoining the auxiliary tree `A` at that adjunction site, one obtains the tree shown in Figure 2.11(d) on the next page, denoted by `AAI`. The yield of the tree `AAI` is the string $aabb\epsilon cc$. The tree `AAI` also has only one adjunction site. Assume that we derived a tree with the yield $\{a^k b^k \epsilon c^k\}$. To obtain a derived tree with the yield $\{a^{k+1} b^{k+1} \epsilon c^{k+1}\}$, one adjoins the auxiliary tree `A` into the derived tree with yield $\{a^k b^k \epsilon c^k\}$. In this way, the TAG grammar consisting of two elementary trees `I` and `A` generates the tree language whose yield is the string language $\{a^n b^n \epsilon c^n\}$. Since ϵ stands for the empty string, the language $\{a^n b^n \epsilon c^n\}$ is the same as $\{a^n b^n c^n\}$.



(a) A (b) I (c) AI (d) AAI

Figure 2.11: TAG derived trees for $\{a^n b^n c^n\}$

2.6.2 LTAG - Lexicalized TAG

Definition 2.6.4 (Lexicalized Grammar (A. K. Joshi and Schabes, 1997)).
A grammar is lexicalized *if it consists of:*

- *A finite set of structures each associated with a lexical item; each lexical item will be called the anchor of the corresponding structure.*
- *An operation or operations for composing the structures.*

One can apply the notion of lexicalized grammars to TAGs to define lexicalized TAGs (LTAGs). It has been claimed that LTAGs are more beneficial than non-lexicalized TAGs in terms of parsing. The advantages that the parsing with LTAGs has over the parsing with non-lexicalized TAGs is the possibility of *grammar filtering* (Schabes, Abeillé, and A. K. Joshi, 1988; Schabes and A. K. Joshi, 1988, 1991). To parse an input string with an LTAG, one *filters the grammar* by selecting only the trees whose anchors are present in the input string. Let us provide a definition of an LTAG.

Definition 2.6.5 (Lexicalized TAG (A. K. Joshi and Schabes, 1997)).
We say that a TAG is a lexicalized *TAG (LTAG) if at least one terminal symbol called* anchor *appears at the frontier of every elementary tree. If c is an anchor of a tree γ, then we say that c anchors γ or γ is anchored with c.*

For instance, Figure 2.8 on page 40 shows elementary trees anchored by the lexical items from the sentence (5); *Fred*, *he*, *exam*, *failed*, and *grumpy*, each of them anchors an initial tree; *because*, *is*, and *an* anchor auxiliary trees.

While in an LTAG, each elementary tree is associated with a lexical item, one lexical item might anchor finitely many elementary trees. For the sake of illustration, let us consider *like*.

(6) a. The boys $like_1$ apples.

b. Apples the boys $like_2$.

c. Apples are $liked_3$ by the boys.

d. The boys ate apples $like_4$ this one.

e. Nectarines look $like_5$ apples.

(6) provides five different usages of *like*. To cover all of the examples in (6), an LTAG grammar contains five distinct trees[14] anchored with *like* shown in Figure 2.12. The initial trees in Figure 2.12(a) and Figure 2.12(b) correspond to the usages of *like* as the predicate in the active (e.g. (6)(a)) and passive (e.g. (6)(c) constructions, respectively. Figure 2.12(c) shows an initial tree anchored with *like* modeling *like* in the topicalized constructions (e.g. (6)(b)). The auxiliary tree in Figure 2.12(d) is used in a case where *like* is the prepositional head of an NP post-modifier (e.g. (6)(d)). Figure 2.12(e) shows the auxiliary used in a case where *like* is a VP post-modifier (e.g. (6)(d)).

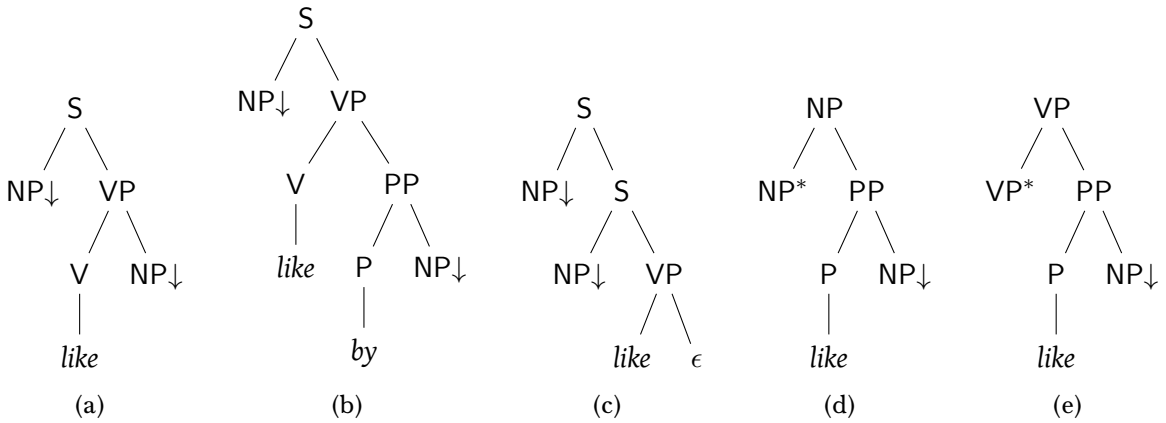


Figure 2.12: LTAG trees anchored with *like*

Note that elementary trees of a TAG can be of any size and shape. In a TAG (LTAG), a hypothesis is that locally occurring dependencies associated with a lexical item can be encoded with an elementary tree anchored by the lexical item. This is known as the property of *extended domain of locality*. For example, one can encode the arguments of a predicate within an elementary tree anchored with the predicate.

> The *argument structure* is not reduced to a list of arguments as the usual subcategorization frames. It is the syntactic structure constructed with the lexical value of the predicate and with all the nodes for its arguments. The argument structure for a predicate is its maximal structure. An argument is present in the argument structure even if it is optional and its optionality is stated in the structure. Schabes, Abeillé, and A. K. Joshi (1988)

[14]The example is adapted from (B. L. Webber, 2004).

Extended domain of locality allows TAG to give an account of long-distance dependencies. In particular, one *expands* locally occurring dependencies encoded with an elementary tree by recursively adjoining trees into the elementary tree. In the resultant tree, the dependents will be located at a greater distance from each other compared to the distance between them in the original elementary tree. The fact that one can distribute content among different trees that adjoin into a given tree is known as the property of *factoring recursion.*

Example 2.4.
Let us consider an example from the TAG literature that illustrates how one factors recursion over several trees in order to model *long-distance dependencies.*

(7) John Bill claims Mary seems to love.

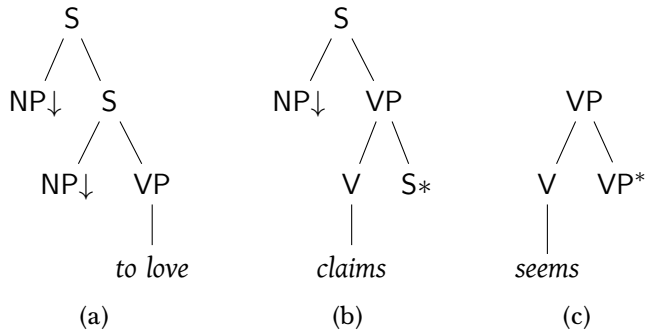


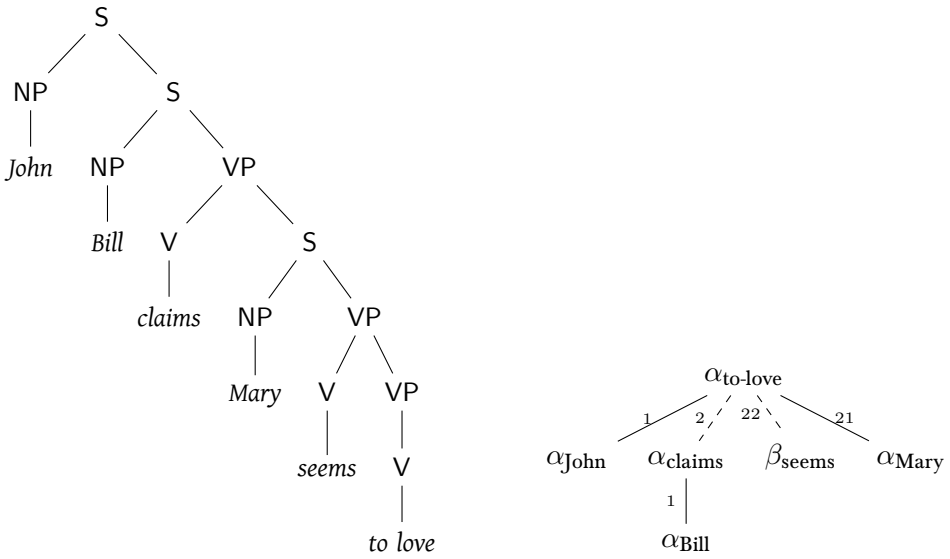
Figure 2.13: LTAG elementary trees

Figure 2.13 shows elementary trees anchored with the lexical items of the sentence (7). The locally occurring dependencies, the subject and object of *to love*, are encoded within an initial tree anchored with *to love.* One distributes (factors) the lexical entries *claims* and *seems* over two different auxiliary trees. Figure 2.14(a) depicts the derived of the sentence (7). As the derivation tree of this derived tree indicates (see Figure 2.14(b)), the auxiliary trees anchored with *claims* and *seems* adjoin into the initial tree anchored with *to love.*

Convention: In an LTAG derivation tree, we identify a node denoting an elementary tree by the lexeme *lex.entry* that anchors it together with the name of the tree (if there are several trees anchored with the same lexeme). We use $\alpha_{lex.entry}$ (resp. $\beta_{lex.entry}$) to denote an initial (resp. auxiliary) tree anchored with *lex.entry.* In order to denote either an initial, or auxiliary, or derived tree, we use γ, unless otherwise stated.

Some Extensions of TAGs

The operation of adjunction in TAG can be seen as follows: β adjoins on the node n into γ is equivalent to say that there are two trees γ_u and γ_b such that γ_u adjoins at the root of β and at the same time γ_b substitutes at the foot of β. Indeed, one can take as γ_u and γ_b the supertree and the subtree of γ at n, respectively. Thus, one can consider a derivation step where one adds a *set* of trees to a tree. Multicomponent



(a) The derived tree (b) The derivation tree

Figure 2.14: The derived and the derivation trees

TAGs (MCTAGs) are based on a generalization of this idea (Vijay-Shanker, David J. Weir, and A. K. Joshi, 1987). An MCTAG is a like a TAG, but MCTAG elementary structures are *sets* of elementary trees. In a derivation step, one must use all trees from one elementary tree set.

> MCTAGs are linguistically interesting because they extend the domain of locality since the contributions of single lexical elements are separated into different trees. Kallmeyer (2010)

One can define adjunction and substitution of one tree set into another one in various ways. Thus, the notion of *derivation* in an MCTAG can be defined in various ways, which gives rise to different variants of MCTAGs. In particular, one considers *tree-local*, *set-local* and *non-local* MCTAGs. An MCTAG is *tree-local* if, at each derivation step, all trees from the same tree set adjoin and/or substitute only at nodes belonging to a *single elementary tree.* An MCTAG is *set-local* if, at each derivation step, all trees from the same tree set adjoin and/or substitute only at nodes belonging to trees from *the same elementary tree set.* If an MCTAG is neither tree-local nor set-local, then it is called non-local. Tree-local and set-local MCTAGs belong to the class of MCSGs, whereas non-local MCTAGs do not.

Another formalism from the class of MCSGs is linear-context free rewriting system (LCFRS) (Vijay-Shanker, David J. Weir, and A. K. Joshi, 1987). LCFRS and set-local MCTAG are weakly equivalent formalisms (David Jeremy Weir, 1988). At the same time, set-Local MCTAGs and LCFRSs are more expressive than TAG. In contrast to set-local MCTAG and LCFRS, tree-local MCTAG and TAG generate the same tree languages. Indeed, one can encode a single derivation step in a tree-local MCTAG as a sequence of derivation steps of a corresponding TAG. However, the recognition problem of a tree-local MCTAG (that is, to determine whether a given string is in a

given language or not) is NP-complete (Nesson, Satta, and M. Shieber, 2010).

2.7 Synchronous Tree Adjoining Grammar

Synchronous Tree Adjoining Grammar (STAG) is a synchronous variant of TAG (Shieber and Schabes, 1990). An STAG defines *derived* structures with the help of *elementary* structures. A structure of an STAG is a *pair* of TAG elementary trees. The nodes of the two trees making a pair are linked. This makes possible to define simultaneous substitutions/adjunctions on the linked nodes of the trees that are components of a tree pair. At a derivation step of an STAG, one combines two tree pairs by either simultaneously *substituting* or *adjoining* one tree pair into another one.

One of the motivations for introducing STAGs is to model the syntax-semantics interface so that both syntactic and semantic analyses of natural language expressions are provided with the help of TAG grammars. Since the grammar is synchronous, the syntactic and semantic derived trees have the *isomorphic* derivation trees (see Remark 2.2 on page 49).

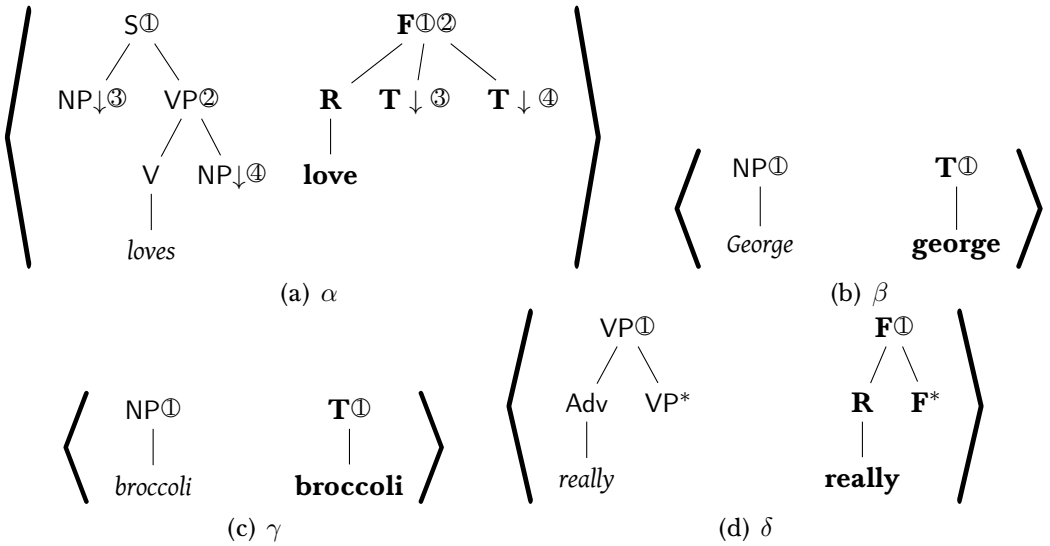


Figure 2.15: STAG elementary structures

More formally, an entry of an STAG is a triple $\langle \gamma_L, \gamma_R, \frown \rangle$, where γ_L and γ_R are TAG elementary trees; $\frown$ is the linking relation between the nodes of the trees γ_L and γ_R. To illustrate that two node are linked with, we annotate them with the same marker $\textcircled{i}$, where $i = 1, 2 \ldots$. We will call $\textcircled{i}$ ***a link*** between the nodes. Thus, we represent the non-empty linking relation $\frown$ as a set of links $\{\textcircled{$i_1$}, \ldots, \textcircled{$i_k$}\}$, where $i_1, \ldots, i_k$ are natural numbers. For instance, Figure 2.15(a) shows that both the S and VP nodes are linked with **F**. We denote the link between S and **F** with ①, whereas the link between VP and **F** is ②.

An STAG defines the tree language by tree pair rewriting. The rewriting process of an STAG involves adjoining/substituting in a derived tree pair another tree pair as follows:

1. We choose a link ⓘ between two nodes n_L and n_R in a given derived tree pair $\langle \gamma_L, \gamma_R, \frown' \rangle$;
2. We choose a tree pair $\langle \beta_L, \beta_R, \frown'' \rangle$ such that β_L can adjoin/substitute at n_L in γ_L and β_R can adjoin/substitute at n_R in γ_R.
3. We obtain a derived tree pair $\langle \gamma_L', \gamma_R', \frown''' \rangle$ by adjoining/substituting β_L at n_L in γ_L and β_R at n_R in γ_R. The linking relation $\frown'''$ is defined as follows: All links in $\frown'$ and $\frown''$ are also in $\frown'''$ except that the chosen link ⓘ.

The language defined by an STAG is a set of derived tree pairs $\langle \gamma_L, \gamma_R \rangle$. For instance, Figure 2.15 shows the elementary structures of an STAG. If one enriches the derivation tree of one of the trees with the information about the related nodes, it is possible to use the derivation tree of γ_L as the derivation tree for γ_R (as they are isomorphic). For instance, by using the grammar entries in Figure 2.15, one produces the pair of TAG derived trees shown in Figure 2.17. One of them is the syntactic derived tree, whose yield is *George really loves broccoli*, whereas the other one encodes the formula **really(love(george, broccoli))**.

Remark 2.1. *A tree rooted in **F** encodes a logical formula as follows:*

- *The predicate is the terminal symbol whose parent node is labeled with the non-terminal **R**;*
- *an argument of a predicate is (a) either a terminal symbol whose parent is labeled with a non-terminal **T** (b) or a sub-tree whose root is a non-terminal symbol **F**.*

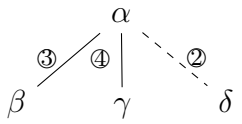
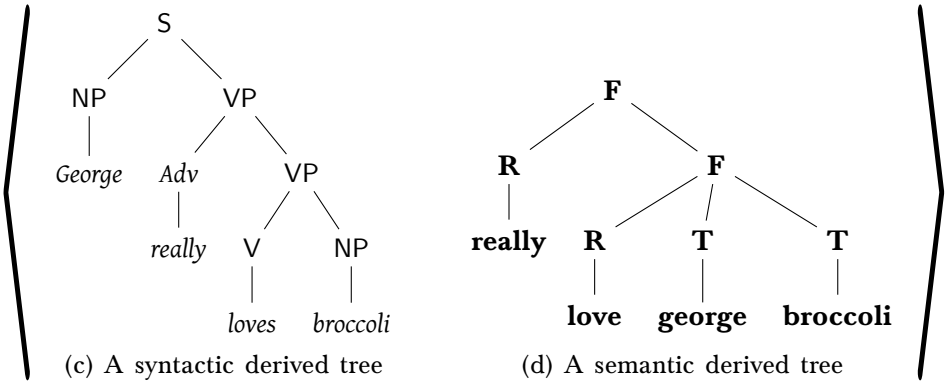


Figure 2.16: An STAG derivation tree



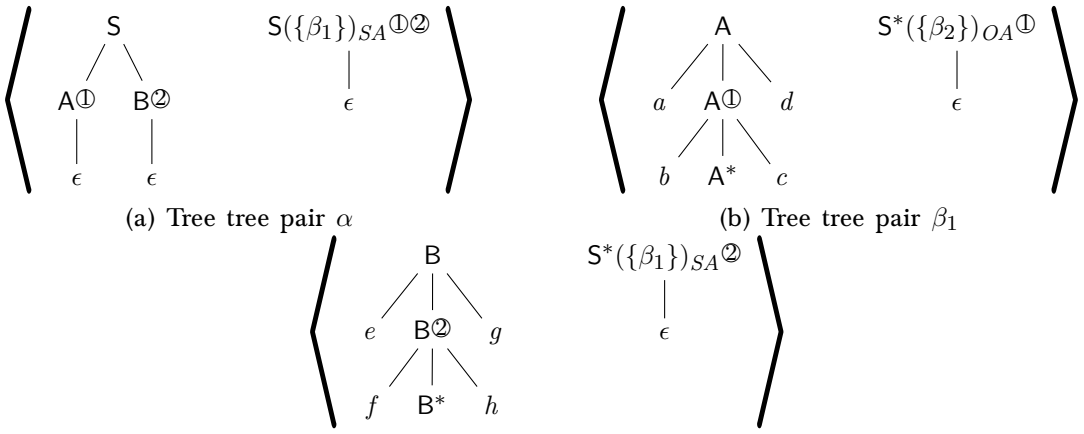
(c) A syntactic derived tree (d) A semantic derived tree

Figure 2.17: An STAG derived tree pair

To obtain the pair of derived trees illustrated in Figure 2.17, one makes use of the operations of substitution and adjunction on the elementary tree pairs as it is shown by the derivation tree in Figure 2.16. In this derivation tree, one records the substitution of

β and γ into α at the nodes with the links ③ and ④, respectively; δ adjoins into α on the node with the link ② (see α, β, γ and δ in Figure 2.15). In this way, one obtains both the syntactic and semantic derived trees. Thus, the derivation tree in Figure 2.16 can serve as the derivation tree for the derived syntactic tree (Figure 2.17(c)) as well as for the derived semantic tree (Figure 2.17(d)).

Remark 2.2. *We confine ourselves with the above provided informal definition of the STAG formalism. However, under this definition, in a pair $L(G) = \langle L_L, L_R \rangle$ of languages generated by an STAG G, one of the languages L_L or L_R may not be a TAG language (Shieber, 1994). For instance, Figure 2.18 shows a STAG grammar that gives rise to the pair of languages $\langle \{a^n b^n c^n d^n e^n f^n g^n h^n\}, \{\epsilon\} \rangle$. No TAG can generate the language $\{a^n b^n c^n d^n e^n f^n g^n h^n\}$. Shieber (1994) provides a solution to the problem. In particular, he redefines the notion of derivation in an STAG that reduces the expressive power of STAGs. With the new definition, the derivation trees of two trees in a tree pair are isomorphic. As it was already indicated above, we assume that the latter requirement holds for the trees in a derived tree pair.*



(a) Tree tree pair α

(b) Tree tree pair β_1

(c) Tree tree pair β_2

Figure 2.18: A grammar producing an STAG language $\langle \{a^n b^n c^n d^n e^n f^n g^n h^n\}, \{\epsilon\} \rangle$

Chapter 3

Abstract Categorial Grammars

Contents

In this chapter, we discuss the Abstract Categorial Grammar (ACG) framework. We show how to encode CFGs and TAGs as ACGs. Afterwards, we make a concise overview of the ACG hierarchy. Among various classes of ACGs, we focus on second-order ones because (1) they allow one to encode a number of formal grammars; (2) the problems of parsing and generation for them are polynomial. In addition, we define a more expressive version of second-order ACGs, called λ-CFGs. They enable us to encode Montague semantics within ACGs. Despite the increase in the expressive power, the parsing and generation problems for λ-CFGs are polynomial. Finally, we show how one encodes TAG with Montague semantics with the help of ACGs.

3.1 Introduction

In the tradition of type-logical approaches to the natural language modeling (Curry, 1960; Lambek, 1958; Montague, 1973; Moortgat, 1997; Morrill, 1994; D. Oehrle, 1995; R. T. Oehrle, 1988, 1994; Ranta, 1994), de Groote (2001) and Muskens (2001) independently proposed two very similar formalisms. The one by de Groote (2001) is referred to as the Abstract Categorial Grammar (ACG) framework, or sometimes as Abstract Categorial Grammars (ACGs), and the one by Muskens (2001) is known as Lambda Grammars. In this thesis, we follow de Groote's (2001) ACGs.

ACGs present a grammatical framework, which can be seen as a generalization of Categorial Grammars (Lambek, 1958). An ACG defines two levels of grammar, called the *abstract* and *object* vocabularies. The idea of a two-level structure that an ACG defines was inspired by Curry's (1960) view on a language, expressed by the following quote:

> . . . we may conceive of the *grammatical structure* of the language as something independent of the way it is represented in terms of expressions . . . This gives us two levels of grammar, the study of grammatical structure in itself, and a second level which has much the same relation to the first that morphophonemics does to morphology. In order to have terms for immediate use I shall call these two levels *tectogrammatics* and *phenogrammatics* respectively . . .
>
> Curry (1960)

Montague's works (Montague, 1970a,b, 1973) (called Montague Grammar) addressing the modeling of the syntax-semantics interface had served as an inspiration source for introducing ACGs. The ideological basis of Montague's work was his famous thesis, quoted as follows:

> There is in my opinion no important theoretical difference between natural languages and the artificial languages of logicians; indeed, I consider it possible to comprehend the syntax and semantics of both kinds of languages within a single natural and mathematically precise theory.
>
> Montague (1970b)

One of the primary goals of ACGs is to deal with the problems of the syntax-semantics interface within a computational framework, as the following quote indicates:

> The distinction between syntax and semantics is of course relevant from a linguistic point of view. This does not mean, however, that it must be wired into the computational model. On the contrary, a computational model based on a small set of primitives that combine via simple composition rules will be more flexible in practice and easier to implement. De Groote (2001)

Generalizing the idea of a functional translation from syntax to semantics proposed in Montague Grammar, an ACG defines the notion of a *lexicon.* The lexicon is a homomorphism from the abstract vocabulary to the structures built upon the object vocabulary.

ACGs are capable of encoding various grammatical formalisms. De Groote (2001, 2002) encodes CFGs and TAGs as ACGs. De Groote and Pogodalla (2004) construct ACGs that are (strongly) equivalent to linear[15] context-free tree grammars and linear context-free rewriting systems (LCFRSs). Since LCFRSs are weakly equivalent to set-local MCTAGs[16] and Minimalist Grammars,[17] ACGs provide a framework where one encodes a number of grammatical formalisms. Pogodalla (2004, 2009) presents a modeling of the syntax-semantics interface where one interprets TAG derivation trees into semantic formulas. To do so, one makes use of higher-order interpretations reminiscent of ones introduced by Montague (1973). However, to encode Montague's (1973) semantics with ACGs, it is necessary to extend the original version of ACGs. Kanazawa (2007) provides one of such extensions of ACGs, called λ-CFGs. It is noteworthy that for the ACGs encodings of all the above-mentioned formalisms (including the one for modeling TAG with Montague semantics), the problems of parsing and generation are polynomial (Kanazawa, 2007; Salvati, 2005).

3.2 Mathematical Preliminaries

We provide some mathematical notions that ACGs involve (de Groote, 2001).

Definition 3.2.1 (Linear Implicative Type).
Let A be a set of atomic types. We define two sets $\mathscr{T}^{\circ}(A)$ and $\mathscr{T}^{\rightarrow}(A)$, called the set of linear implicative types built upon A and the set of implicative types built upon A respectively.
Linear implicative types

1. $A \subset \mathscr{T}^{\circ}(A)$;
2. *if* $\alpha \in \mathscr{T}^{\circ}(A)$ *and* $\beta \in \mathscr{T}^{\circ}(A)$, *then* $(\alpha \multimap \beta) \in \mathscr{T}^{\circ}(A)$.

Implicative types

1. $A \subset \mathscr{T}^{\rightarrow}(A)$;
2. *if* $\alpha \in \mathscr{T}^{\rightarrow}(A)$ *and* $\beta \in \mathscr{T}^{\rightarrow}(A)$, *then* $(\alpha \rightarrow \beta) \in \mathscr{T}^{\rightarrow}(A)$.

[15] Here, *linear* means *non-duplicating and non-deleting.*
[16] (David Jeremy Weir, 1988).
[17] (Michaelis, 2001).

By convention, we write $\alpha_1 \multimap \ldots \multimap \alpha_n \multimap \beta$ for $(\alpha_1 \multimap (\ldots \multimap (\alpha_n \multimap \beta)\ldots))$; and $\alpha_1 \to \ldots \to \alpha_n \to \beta$ for $(\alpha_1 \to (\ldots \to (\alpha_n \to \beta)\ldots))$.

Definition 3.2.2 (Order of Type).
Given a set of atomic types A, the order $\mathfrak{ord}(\alpha)$ of a type belonging to either $\mathscr{T}^{\multimap}(A)$ or $\mathscr{T}^{\to}(A)$ is inductively defined as follows:

- $\mathfrak{ord}(\alpha) = 1$ *for any atomic α (i.e., for any $\alpha \in A$);*
- $\mathfrak{ord}(\alpha \multimap \beta) = \max\{\mathfrak{ord}(\alpha) + 1,\ \mathfrak{ord}(\beta)\}$ *for any $\alpha, \beta \in \mathscr{T}^{\multimap}(A)$.*
- $\mathfrak{ord}(\alpha \to \beta) = \max\{\mathfrak{ord}(\alpha) + 1,\ \mathfrak{ord}(\beta)\}$ *for any $\alpha, \beta \in \mathscr{T}^{\to}(A)$.*

Definition 3.2.3 (Higher-Order Signature and Higher-Order Linear Signature).

A higher-order linear signature *(*HOS*) is a triple $\Sigma = \langle A, C, \tau^{\multimap}\rangle$ defined as follows:*

- *A is a finite set of atomic types;*
- *C is a finite set of constants;*
- *$\tau^{\multimap} : C \longrightarrow \mathscr{T}^{\multimap}(A)$ is type assignment that maps each constant from C to a linear implicative type built upon A.*

A higher-order signature *is a tripe $\Sigma = \langle A, C, \tau^{\to}\rangle$ defined as follows:*

- *A is a finite set of atomic types;*
- *C is a finite set of constants;*
- *$\tau^{\to} : C \longrightarrow \mathscr{T}^{\to}(A)$ is type assignment that maps each constant from C to an implicative type built upon A.*

To express that a is of type α, i.e., that $\tau^{\multimap}(a) = \alpha$ or $\tau^{\to}(a) = \alpha$, we write $a : \alpha$. In $\tau^{\multimap}$ and $\tau^{\to}$, we drop the superscripts $\multimap$ and $\to$ if it does not cause a confusion.

Definition 3.2.4. *We define the order of a* HOS *$\Sigma = \langle A, C, \tau\rangle$, denoted by $\mathfrak{ord}(\Sigma)$, to be the maximum of the orders of the types of its constants, that is, $\mathfrak{ord}(\Sigma) = \max_{a \in C} \mathfrak{ord}(\tau(a))$.*

We also make use of standard notions and notations[18] from λ-calculus such as *free and bound variables* of a λ-term, where one denotes with $\mathsf{FV}(t)$ the set of free variables of a term t; *a closed term*, i.e., a term without any free variable; *a combinator*, i.e., a term with neither constants nor free variables; the notions of α-equivalence ($\leftrightarrow_\alpha$), a single step β-reduction ($\to_\beta$), a multi-step β-reduction ($\twoheadrightarrow_\beta$), β-equivalence ($\leftrightarrow_\beta$), η-conversion, $\beta\eta$-equivalence ($\leftrightarrow_{\beta\eta}$), β-normal form, $\beta\eta$-normal form, etc. By assuming these notions, we define *linear lambda terms* (abbreviated as λ^{o}-terms).

Definition 3.2.5 (Linear Lambda Terms).
Let X be a countably infinite set of variables and $\Sigma = \langle A, C, \tau\rangle$ be a higher-order linear signature, then $\Lambda(\Sigma)$ the set of λ^{o}-terms built on Σ is defined as follows:

- *$X \subset \Lambda(\Sigma)$;*
- *$C \subset \Lambda(\Sigma)$;*
- *if $x \in X$ and $t \in \Lambda(\Sigma)$ and if x has exactly one free occurrence in t, then $\lambda^{\text{o}} x.t \in \Lambda(\Sigma)$;*
- *if $t \in \Lambda(\Sigma)$, $u \in \Lambda(\Sigma)$ and t and u do not share any free variable, that is, $\mathsf{FV}(t) \cap \mathsf{FV}(u) = \emptyset$, then $(t\ u) \in \Lambda(\Sigma)$.*

[18] Following (Barendregt, 1992).

We follow the standard conventions:
- We write $t_1 \dots t_n$ for application $((\dots(t_1\, t_2)\dots)\, t_n)$;
- we write $\lambda^o x_1 \dots x_n.\ t$ for $(\lambda^o x_1.(\lambda^o x_2.\dots.(\lambda^o x_n.\ t)\dots))$.
- we write $\lambda^o \vec{x}.t$ for $(\lambda^o x_1.(\lambda^o x_2.\dots.(\lambda^o x_n.\ t)\dots))$, where $\vec{x}$ stands for the sequence $x_1, x_2, \dots, x_n$.

Definition 3.2.6 (Typing Judgments for λ^o-Terms).
Let $\Sigma = \langle A, C, \tau\rangle$ be a HOS. *The typing rules are given with an inference system whose judgments are of the form $\Gamma \vdash_\Sigma t : \alpha$, where*
- *Γ is a finite set of variable declarations of the form $x : \chi$, where $x \in X$ and $\chi \in \mathscr{T}^\circ(A)$, so that any variable declaration can occur in Γ at most once;*
- *$t \in \Lambda(\Sigma)$;*
- *$\alpha \in \mathscr{T}^\circ(A)$.*

One derives typing judgments according to the following inference system:
Axioms:

$$\frac{}{\vdash_\Sigma c : \tau(c),\ \textit{where } c \in C}\,(\textit{con}) \qquad \frac{x : \chi}{\vdash_\Sigma x : \chi\,,\ \textit{where } x \in X}\,(\textit{var})$$

Inference Rules:

$$\frac{\Gamma, x : \alpha \vdash_\Sigma t : \beta}{\Gamma \vdash_\Sigma \lambda^o x.t : \alpha \multimap \beta}\,(\textit{lin.abs}) \qquad \frac{\Gamma \vdash_\Sigma t : \alpha \multimap \beta \qquad \Delta \vdash_\Sigma u : \alpha}{\Gamma, \Delta \vdash_\Sigma (t\, u) : \beta}\,(\textit{lin.app})$$

We drop the subscript Σ in $\vdash_\Sigma t : \alpha$ whenever it does not cause a confusion.

3.2.1 Strings and Trees as λ^o-terms

We show the standard way of encoding trees and strings as λ^o-terms.

3.2.1.1 Strings

Definition 3.2.7 (String Signature).
Given a set Δ, we consider a HOS $\Sigma_\Delta^{string} = \langle A, C, \tau\rangle$ *defined as follows:*
1. *Σ_Δ^{string} has a single atomic type $*$, i.e., $A = \{*\}$.*
2. *$C = \Delta$.*
3. *For each $c \in C$, $\tau(c) = * \multimap *$. The type $* \multimap *$ is called the string type.*

We call Σ_Δ^{string} a string *signature over Δ.*

Given a finite alphabet Δ, we build a HOS Σ_Δ^{string} according to Definition 3.2.7. Constants of Σ_Δ model symbols in Δ. We have a single atomic type $*$ in Σ_Δ. One encodes a sting over Δ with a λ^o-term in $\Lambda(\Sigma_\Delta)$ of type $* \multimap *$ as follows:

$$a_1 \dots a_n \in \Delta^* \quad \text{is represented as a term} \quad \lambda^o z.\, a_1\,(\cdots(a_n\, z)\cdots) : * \multimap *.$$

We encode a string concatenation with a combinator $\lambda^o u_1\, u_2.\, \lambda^o z.\, u_1\,(u_2\, z)$. Indeed, if t_1 is a λ^o-term encoding a string θ_1 and t_2 is a λ^o-term encoding θ_2, then $\lambda^o z.\, t_1\,(t_2\, z)$

encodes the concatenation of the two strings $\theta_1\theta_2$. To obtain the term $\lambda^o z.\, t_1\,(t_2\,z)$, one applies the combinator $\lambda^o x_1\, x_2.\ \lambda^o\, z.\, x_1\,(x_2\,z)$ to t_1 and t_2. Let us illustrate this on an example.

Example 3.1. One computes the concatenation of a string ab and a string c as follows:

$$\underbrace{(\lambda^o u_1.\lambda^o u_2.\lambda^o\ z\,.u_1\,(u_2\,z))}_{\texttt{concatenation}}\ \underbrace{(\lambda^o\ x.\ a\,(b\,x))}_{ab}\ \underbrace{(\lambda^o\ y.\ c\,y)}_{c} \twoheadrightarrow_\beta \underbrace{\lambda^o\ z.a\,(b\,(c\,z))}_{abc}$$

We denote the combinator $\lambda^o u_1\, u_2.\, \lambda^o\, z.\, u_1\,(u_2\,z)$ encoding the string concatenation operation with $+$. With this notation, $(\lambda^o u_1\, u_2.\, \lambda^o\, z.\, u_1\,(u_2\,z))\,(t)\,(u)$ becomes $+t\,u$. We use an infix notation $t + u$ instead of the Polish notation $+t\,u$. The type of $+$ is $(* \multimap *) \multimap (* \multimap *) \multimap * \multimap *$. One can check that the string concatenation operator $+$ has the following properties:

1. There is a term ϵ that encodes the empty string, that is, for any λ^o-term t encoding a string, the following holds: $t + \epsilon = \epsilon + t = t$, where $=$ should be understood as $\leftrightarrow_{\beta\eta}$. Indeed, the identity function $\lambda^o z.z$ plays the role of ϵ.
2. $+$ is associative, i.e., for any λ^o-terms t, u, and v that encode strings, the following holds: $(t + u) + v = t + (u + v)$.

Since for each a in C, the type of a is $* \multimap *$, the signature Σ_Δ is *second-order*. As $\forall\ a \in C$, we have $\tau(a) = * \multimap *$, and $+$ is of type $(* \multimap *) \multimap (* \multimap *) \multimap * \multimap *$, by denoting $* \multimap *$ with σ, we get that $\forall\ a \in C$, $\tau(a) = \sigma$; the type of $+$ becomes $\sigma \multimap \sigma \multimap \sigma$. Thereafter, we use a convention that instead of $\Sigma_\Delta^{\textit{string}} = \langle\{*\}, \Delta, \tau\rangle$, we write $\Sigma_\Delta^{\textit{string}} = \langle\{\sigma\}, \Delta, \tau_\sigma\rangle$. We may call σ *the type* of a string signature $\Sigma_\Delta^{\textit{string}}$, though it abbreviates a functional type $* \multimap *$. Whenever it does not cause a confusion, we drop the subscript Δ in $\Sigma_\Delta^{\textit{string}}$ and the subscript σ in τ_σ.

Example 3.2. Let Δ be an alphabet consisting of three symbols *Fred*, *failed* and *exams*. To model strings over Δ, we introduce $\Sigma_\Delta^{\textit{string}}$, which is $\langle\{\sigma\}, \{\textit{Fred}, \textit{failed}, \textit{exams}\}, \tau_\sigma\rangle$ where $\tau_\sigma(\textit{Fred}) = \tau_\sigma(\textit{failed}) = \tau_\sigma(\textit{exams}) = \sigma$. We encode strings obtained by concatenating symbols of Δ as terms of type σ over $\Sigma_\Delta^{\textit{string}}$. For example, we model a string *Fred failed exams* over Δ as a term $\textit{Fred} + \textit{failed} + \textit{exams}$ of type σ.

3.2.1.2 Trees

Definition 3.2.8 (Tree Signature).
For a ranked alphabet $\langle\Delta, \rho\rangle$, we define a HOS $\Sigma_\Delta^{\textit{tree}} = \langle A, C, \tau\rangle$ *as follows:*

1. $\Sigma_\Delta^{\textit{tree}}$ *has a single atomic type τ, i.e., $A = \{\tau\}$.*
2. $C = \Delta$.
3. *For every $f \in \Delta$,* $\tau(f) = \underbrace{\tau \multimap \ldots \multimap \tau}_{\rho(f)\ \textit{times}} \multimap \tau$.

We call $\Sigma_\Delta^{\textit{tree}}$ a tree *signature (determined by the ranked alphabet Δ).*

One models the tree set $\mathbb{T}_\Delta$ as the subset of $\Lambda(\Sigma_\Delta)$ whose elements are closed terms of type τ. Whenever it does not cause a confusion, we drop the subscript Δ in $\Sigma_\Delta^{\textit{tree}}$.

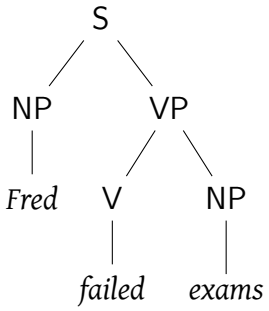


Figure 3.1: A syntactic tree

Example 3.3. Let us consider a tree depicted in Figure 3.1. To model this tree as a term over a tree signature, we follow Definition 3.2.8 and define the following tree signature Σ^{tree}:

$$\Sigma^{tree} = \langle \{\tau\},\ \{\mathit{Fred},\ \mathit{failed},\ \mathit{exams},\ \mathsf{NP}_1,\ \mathsf{V}_1,\ \mathsf{S}_2,\ \mathsf{VP}_2\},\ \tau_\tau \rangle$$

Where *Fred*, *failed*, and *exams* are of type τ; NP_1, and V_1 are of type $\tau \multimap \tau$; S_2 and VP_2 are of type $\tau \multimap \tau \multimap \tau$. We model the tree in Figure 3.1 with a term $u : \tau$ over Σ^{tree}, defined as follows:

$$u \ = \ \mathsf{S}_2\ (\mathsf{NP}_1\ \mathit{Fred})\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \mathit{failed})\ (\mathsf{NP}_1\ \mathit{exams})) : \ \tau \tag{3.1}$$

It follows from Definition 3.2.8 that, like string signatures, tree signatures are second-order ones.

3.2.2 Adjunction and Substitution as Functional Application

We saw how to encode trees as λ^{o}-terms of type τ. We encode the operations of substitution and adjunction over trees as the operation of functional application over λ^{o}-terms (de Groote, 2002).

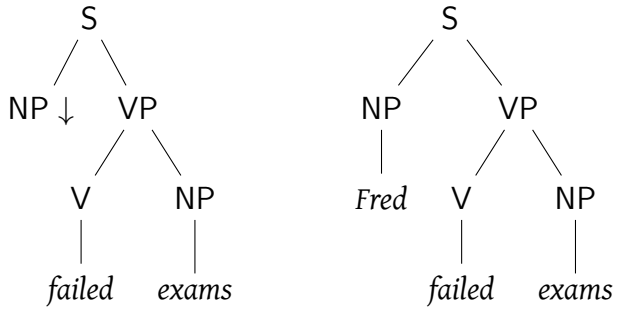
3.2.2.1 Substitution as Functional Application

If a tree α substitutes into a tree γ, then we encode γ as an ***operator***, and α as its ***operand***. The operator ***applies*** to the operand. The result of the application of the operator to the operand encodes the tree obtained by substituting α into γ. Let a λ^{o}-term a be the encoding of the tree α and a λ^{o}-term g be encoding of γ. We encode the node at which α substitutes into γ with the help of a λ^{o}-variable x in g. Indeed, the functional application of g to a will ***insert*** a in the place of x in g. For example let us consider $\overset{\mathsf{NP}}{\underset{\mathit{Fred}}{|}}$ in the role of α, whereas we take the tree in Figure 3.2(a) in the role of γ. The tree obtained by substituting α into γ at the node marked with $\downarrow$ is shown in Figure 3.2(b). We already saw how to encode trees with no substitution sites

as λ^o-terms. Thus, we encode $\overset{\mathsf{NP}}{\underset{\mathit{Fred}}{|}}$ as the following λ^o-term of type τ:

$$a = \mathsf{NP}_1\ \mathit{Fred} : \tau \tag{3.2}$$

An encoding of the tree in Figure 3.2(a) should be an operator with one operand of type τ. The output of the application of the operator to the operand is also a term of type τ. In other words, the encoding of the tree in Figure 3.2(a) transforms a term of type τ into a term of type τ. Hence, it is a term of type $\tau \multimap \tau$.



(a) A tree with a substitution site

(b) The obtained tree

Figure 3.2: Two trees

To encode the tree in Figure 3.2(a) as a λ^o-term, we denote the node marked for substitution $\downarrow$ with x. We encode the obtained tree as a λ^o-term, as follows:

$$\mathsf{S}_2\ x\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \mathit{failed})\ (\mathsf{NP}_1\ \mathit{exams}))$$

However, the latter term lacks the information that x is going to be substituted by a term of type τ (as it models a tree with no substitution sites). To do that, we bind x by $\lambda^o x$. Hence, one proposes the following encoding of the tree in Figure 3.2(a).

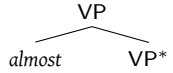
$$g = \lambda^o x.\ \mathsf{S}_2\ x\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \mathit{failed})\ (\mathsf{NP}_1\ \mathit{exams})) : \tau \multimap \tau \tag{3.3}$$

The following checks that the term $(g\ a)$ indeed models the tree Figure 3.2(b):

$$g\ a = (\lambda^o x.\ \mathsf{S}_2\ x\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \mathit{failed})\ (\mathsf{NP}_1\ \mathit{exams})))\ (\mathsf{NP}_1\ \mathit{Fred}) \rightarrow_\beta \mathsf{S}_2\ (\mathsf{NP}_1\ \mathit{exams})\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \mathit{failed})\ (\mathsf{NP}_1\ \mathit{exams})) \tag{3.4}$$

3.2.2.2 Adjunction as Functional Application

Similar to the case of substitution, we view a tree that adjoins into another tree as an *operand*, and the tree where it adjoins is an *operator*. However, while a substitution takes place at a frontier node in a tree, an adjunction takes place at an *internal* node in a tree. Hence, in the case of adjunction, the variable nodes, which we are going to substitute by auxiliary trees, are *internal* ones. To model adjunction as functional application, we cannot use the type τ used in the case of substitution, because adjunction is a more

complex operation than substitution. Indeed, when a tree β adjoins into a tree γ, β substitutes an internal node n of γ so that (1) the mother of n becomes the mother of the root node of β; (2) the daughters of n become daughters of the foot node of β. Thus, we encode the tree β as an argument of the tree γ such that β itself receives an argument (the subtree of γ at n). In other words, the auxiliary tree β is an *operator* whose operand is the subtree of γ at n. Hence, we model β with a *functional* type. For example, let us consider an adjunction of a tree $\beta =$  on the VP-adjunction site into the tree γ in Figure 3.2(b) on the preceding page (for now, let us assume that it is the only adjunction site in γ). The resultant tree is depicted in Figure 3.3.

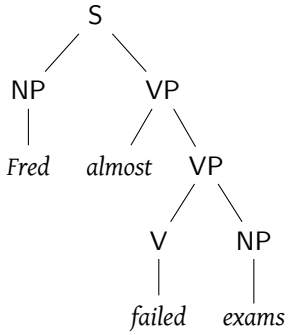


Figure 3.3: A tree obtained by adjoining β into γ

We model β with a term of type $\tau \multimap \tau$, as follows:

$$b = \lambda^{o} y.\ \mathsf{VP}_2\ \textit{almost}\ y : \tau \multimap \tau \tag{3.5}$$

The encoding of the tree γ should be able to apply to the term b. We propose the following encoding of γ:

$$g = \lambda^{o} z.\ \mathsf{S}_2\ (\mathsf{NP}_1\ \textit{Fred})\ (z\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \textit{failed})\ (\mathsf{NP}_1\ \textit{exams}))) : (\tau \multimap \tau) \multimap \tau \tag{3.6}$$

Let us check that the term $(g\ b)$ indeed models the tree in Figure 3.3:

$$\begin{aligned} g\ b = \lambda^{o} z.\ \mathsf{S}_2\ (\mathsf{NP}_1\ \textit{Fred})\ (z\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \textit{failed})\ (\mathsf{NP}_1\ \textit{exams})))\ (\lambda^{o} y.\ \mathsf{VP}_2\ \textit{almost}\ y) \rightarrow_{\beta} \\ \mathsf{S}_2\ (\mathsf{NP}_1\ \textit{Fred})\ ((\lambda^{o} y.\ \mathsf{VP}_2\ \textit{almost}\ y)\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \textit{failed})\ (\mathsf{NP}_1\ \textit{exams}))) \rightarrow_{\beta} \\ \mathsf{S}_2\ (\mathsf{NP}_1\ \textit{Fred})\ (\mathsf{VP}_2\ \textit{almost}\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \textit{failed})\ (\mathsf{NP}_1\ \textit{exams}))) \end{aligned} \tag{3.7}$$

In this way, we model adjunction as functional application where an operator applies to an operand of type $\tau \multimap \tau$.

Remark 3.1. *In a TAG, one can substitute α into γ, if α is derived from an initial tree. One can assume that α does not have unfilled substitution sites, like*  *. Indeed, if α has some unfilled substitution sites, we first fill these substitution sites and then we substitute the obtained tree into γ. Hence, one can model α with a term of type τ. This means that we model a substitution as a functional application where the operand is of type τ.*

If β adjoins in γ, we assume that β, which is either an auxiliary tree or is derived from an auxiliary tree, has no substitution sites. Indeed, if β has substitution sites, then we first fill the substitution sites and then we adjoin the obtained tree into γ. This means that we model an adjunction as a functional application where the operand is of type $\tau \multimap \tau$.

The two above made assumptions do not cause any restrictions on the language defined by the TAG. Indeed, by definition, trees of the TAG derived tree language are completed *ones, that is, they do not have unfilled substitution sites. Thus, in order to derive a completed tree γ, one has to fill the substitution sites of the elementary trees that are involved in the derivation of γ. The order in which one performs these substitutions does not have an impact on the derived tree (Schabes and Shieber, 1994). Thus, one generates the same tree language under these assumptions.*

3.3 Abstract Categorial Grammars

In this section, we present the definition of ACGs (de Groote, 2001).

Definition 3.3.1 (Lexicon (de Groote, 2001)).
A lexicon from a HOS $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ *to a* HOS $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ *is a pair $\mathscr{L} = \langle F, G \rangle$, defined as follows:*

1. *$F : A_1 \longrightarrow \mathscr{T}^{\circ}(A_2)$ maps atomic types in A_1 to linear implicative types built on A_2; we also denote by the same symbol F the homomorphic extension of F over $\mathscr{T}^{\circ}(A_1)$, i.e., we write $F : \mathscr{T}^{\circ}(A_1) \longrightarrow \mathscr{T}^{\circ}(A_2)$.*
2. *$G : C_1 \longrightarrow \Lambda(\Sigma_2)$ maps constants in C_1 to λ^{o}-terms built on Σ_2; we also denote by the same symbol G the homomorphic extension of G over $\Lambda(\Sigma_1)$, i.e., we write $G : \Lambda(\Sigma_1) \longrightarrow \Lambda(\Sigma_2)$.*
3. *F and G are such that for any $c \in C_1$, it holds that $\vdash_{\Sigma_2} G(c) : F(\tau_1(c))$ is a provable (derivable) typing judgment. The latter condition ensures that the mapping of types and the mapping of constants are in concordance with each other, that is, $\mathscr{L}$ is well-defined.*

By convention, we write $\mathscr{L}$ instead of F and G whenever it does not cause a confusion. We write $\mathscr{L} : \Sigma_1 \longrightarrow \Sigma_2$ to denote that $\mathscr{L}$ is a lexicon between two HOS*s Σ_1 and Σ_2.*

Given $\mathscr{L} : \Sigma_1 \longrightarrow \Sigma_2$, one can check that if $t \in \Lambda(\Sigma_1)$ is of type $\alpha \multimap \beta$ and $\mathscr{L}$ is a lexicon, then $\mathscr{L}(t)$ is of type $\mathscr{L}(\alpha) \multimap \mathscr{L}(\beta)$. That is, a lexicon interprets functional types into functional ones.

Definition 3.3.2 (Abstract Categorial Grammars (de Groote, 2001)).
An abstract categorial grammar (ACG) $\mathcal{G}$ is a quadruple $\langle \Sigma_1, \Sigma_2, \mathscr{L}, s \rangle$, where

1. *Σ_1 and Σ_2 are higher-order linear signatures, called the* abstract *vocabulary and the* object *vocabulary, respectively;*
2. *$\mathscr{L} : \Sigma_1 \longrightarrow \Sigma_2$ is a lexicon from Σ_1 to Σ_2;*
3. *s is a type of the abstract vocabulary (either atomic or built upon the atomic types in Σ_1), called the* distinguished *type of the grammar.*

Definition 3.3.3 (Abstract and Object Languages (de Groote, 2001)). *Let $\mathcal{G}$ be an ACG, where $\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, s \rangle$. Two sets, called the* abstract *language $\mathcal{A}(\mathcal{G})$ and the* object *language $\mathcal{O}(\mathcal{G})$, are associated with $\mathcal{G}$. They are defined as follows:*

The abstract language*:* $\mathcal{A}(\mathcal{G}) = \{t \in \Lambda(\Sigma_1) \mid \vdash_{\Sigma_1} t : s \quad \textit{is derivable}\}$
The object language*:* $\mathcal{O}(\mathcal{G}) = \{u \in \Lambda(\Sigma_2) \mid \exists t \in \mathcal{A}(\mathcal{G}) : u = \mathscr{L}(t)\}$
We say that an ACG $\mathcal{G}$ defines *(*generates*) the abstract and object languages* $\mathcal{A}(\mathcal{G})$ *and* $\mathcal{O}(\mathcal{G})$*, respectively.*

> It may be useful of thinking of the abstract language as a set of abstract grammatical structures, and of the object language as the set of concrete forms generated from these abstract structures. De Groote (2001)

Hence, the abstract language is a set of closed λ^{o}-terms over the abstract vocabulary of the distinguished type. The object language is an image of the abstract language under the lexicon. Thus, a term over the object vocabulary u is a member of the object language if and only if there is a closed term t over the abstract vocabulary of the distinguished type s such that $\mathscr{L}(t) = u$, where $=$ is $\leftrightarrow_{\beta\eta}$. We may write $\mathscr{L}(t) = u$ also as $t :=_{\mathscr{L}} u$, or just as $t := u$ if it does not cause a confusion. If we say that an ACG $\mathcal{G}$ generates a language L, without specifying whether L is the abstract language or the object one, then one assumes that L is the object language.

As Definition 3.3.3 indicates, the abstract and object vocabularies are the same kind of mathematical structures as both are HOSs. Instead of the abstract and object languages, we may rather consider only the abstract and object vocabularies, the distinguished type, and the lexicon of an ACG, because they unequivocally determine the abstract and object languages defined by the ACG. We also may refer to constants and terms over the abstract (resp. object) vocabulary as *abstract constants* (resp. *object constants*) and *abstract terms* (resp. *object terms*) whenever it does not cause a confusion.

Figure 3.4 illustrates an ACG, where the larger disks are pictorial representations of the involved vocabularies and the smaller ones stand for languages. The arrow linking the vocabularies, and subsequently the languages, stands for the lexicon.[19]

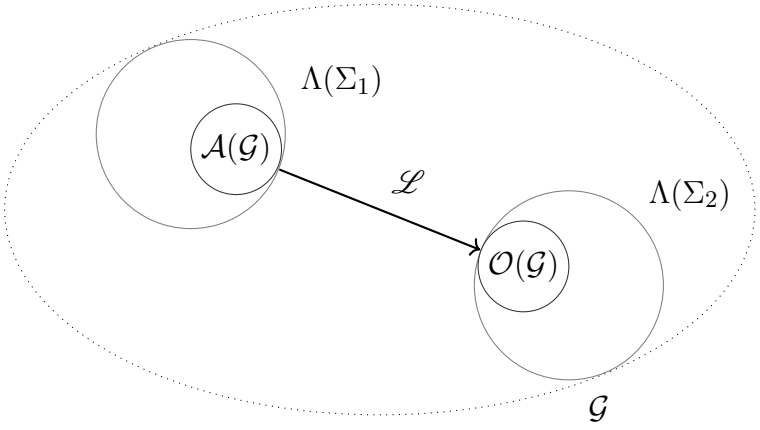


Figure 3.4: A picture of an ACG with its abstract and object languages

[19]The ellipse with the dotted borderline in Figure 3.4 indicates that this is an ACG whose building blocks (i.e. two signatures linked with a lexicon) are in the confines of the ellipse. We do not draw a borderline of an ACG whenever it does not create a confusion.

3.3.1 An Example of an ACG

As one of the examples of ACGs, we encode the correspondence between trees and their yields. We model trees as abstract terms and yields as their images under the lexicon. We assume that trees are ranked trees over some ranked alphabet $\Delta = N \cup \Sigma$, where $N \cap \Sigma = \emptyset$. Elements of Σ are terminal symbols and elements of N are non-terminal symbols. Terminal symbols are of rank 0, whereas the rank of a non-terminal symbol is a positive natural number.

For a given Δ, we consider two signatures, $\Sigma_{\Delta}^{\mathit{tree}}$ and $\Sigma_{\Sigma}^{\mathit{string}}$. $\Sigma_{\Delta}^{\mathit{tree}}$ is the abstract vocabulary and $\Sigma_{\Delta}^{\mathit{string}}$ is the object vocabulary of the ACG we are building. The distinguished type of the ACG is the atomic type of $\Sigma_{\Delta}^{\mathit{tree}}$, i.e., τ. We define a lexicon $\mathscr{L}_{\mathsf{yield}}$ from $\Sigma_{\Delta}^{\mathit{tree}}$ to the terms over $\Sigma_{\Delta}^{\mathit{string}}$ as follows:

- The image of the tree type τ of the $\Sigma_{\Delta}^{\mathit{tree}}$ under $\mathscr{L}_{\mathsf{yield}}$ is the string type σ, i.e., $\mathscr{L}_{\mathsf{yield}}(\tau) = \sigma$.
- If $\mathsf{X} \in \Sigma_{\Delta}^{\mathit{tree}}$ is of type τ (X encodes a 0-ary symbol of Δ, which means that X is a terminal symbol), then: $\mathscr{L}_{\mathsf{yield}}(\mathsf{X}) = \mathsf{X}$.
- If $\mathsf{X}_n \in \Sigma_{\Delta}^{\mathit{tree}}$ is of type $\underbrace{\tau \multimap \ldots \multimap \tau}_{n\text{–times}} \multimap \tau$ where $n \geq 1$ (X_n encodes an n-ary symbol of Δ), then:

$$\mathscr{L}_{\mathsf{yield}}(\mathsf{X}) = \lambda^{\circ} x_1 \ldots x_n.\ x_1 + \ldots + x_n \ : \underbrace{\sigma \multimap \ldots \multimap \sigma}_{n\text{–times}} \multimap \sigma$$

Closed terms over $\Sigma_{\Delta}^{\mathit{tree}}$ of type τ encode trees over Δ. Their images under $\mathscr{L}_{\mathsf{yield}}$ encode yields of these trees. For the sake of example, let us consider the tree shown in Figure 3.5. We encode this tree as a λ°-term u, where $u = \mathsf{S}_2\ (\mathsf{NP}_1\ \mathit{Fred})\ (\mathsf{VP}_2\ (\mathsf{V}_1\ \mathit{failed})\ (\mathsf{NP}_1\ \mathit{exams})) : \tau$. By interpreting u with $\mathscr{L}_{\mathsf{yield}}$, we obtain the following:

$$\begin{aligned}
&\mathscr{L}_{\mathsf{yield}}(u) = \\
&\mathscr{L}_{\mathsf{yield}}(\mathsf{S}_2)\ (\mathscr{L}_{\mathsf{yield}}(\mathsf{NP}_1)\ \mathscr{L}_{\mathsf{yield}}(\mathit{Fred}))\ (\mathscr{L}_{\mathsf{yield}}(\mathsf{VP}_2)\ (\mathscr{L}_{\mathsf{yield}}(\mathsf{V}_1)\ \mathscr{L}_{\mathsf{yield}}(\mathit{failed}))\\
&\qquad\qquad (\mathscr{L}_{\mathsf{yield}}(\mathsf{NP}_1)\ \mathscr{L}_{\mathsf{yield}}(\mathit{exams}))) \twoheadrightarrow_{\beta}\\
&\qquad\qquad \twoheadrightarrow_{\beta} \mathit{Fred} + \mathit{failed} + \mathit{exams} : \sigma
\end{aligned}$$

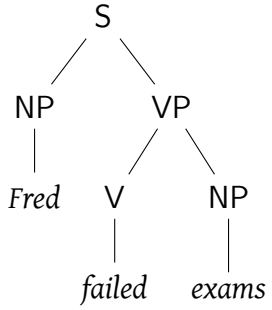


Figure 3.5: A syntactic tree

In the rest of the thesis, we will make use of the ACG $\langle \Sigma_\Delta^{tree}, \Sigma_\Delta^{string}, \mathscr{L}_{\mathsf{yield}}, \tau \rangle$ while discussing encodings of ACG encodings of grammatical formalisms. We may not specify Δ if it clear from the context. We refer to the ACG $\langle \Sigma_\Delta^{tree}, \Sigma_\Delta^{string}, \mathscr{L}_{\mathsf{yield}}, \tau \rangle$ as the *yield ACG for* Δ, or simply as *a yield ACG*.

3.3.2 ACGs with the Same Abstract Language

The abstract language of an ACG does not depend on a lexicon; it rather depends on the abstract signature and on the distinguished type. Thus, if several ACGs share the same abstract vocabulary and their distinguished types are the same, then these ACGs define the same abstract languages. In this case, we depict these ACGs with the help of only one abstract signature and several object signatures and lexicons, for example, as it is in Figure 3.6. This architecture is useful for modeling the syntax-semantics interface using the ACGs encoding of TAG. With the shared abstract vocabulary we encode TAG derivation trees. One of the lexicons maps TAG derivation trees to TAG derived trees (one of the ACGs). The other one maps TAG derivation trees to semantic interpretations (another ACG).

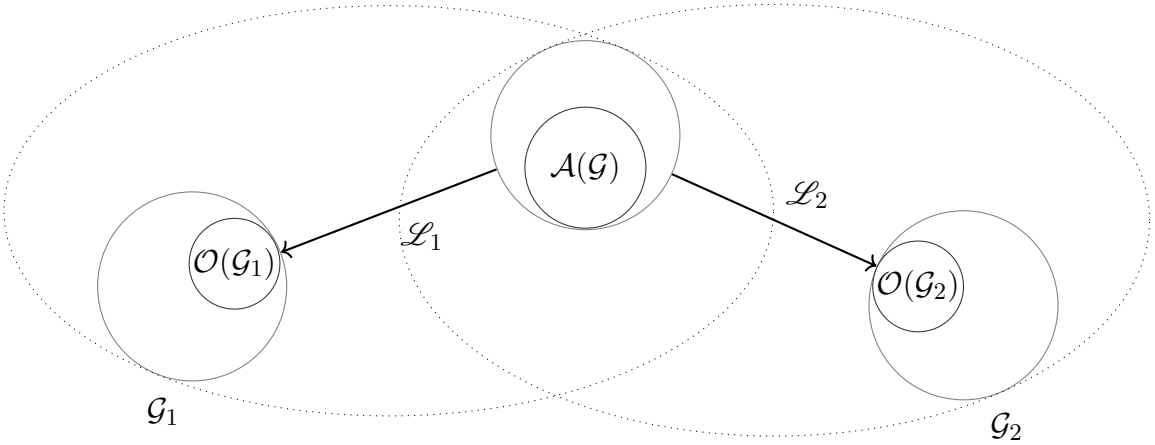


Figure 3.6: Two ACGs with the same abstract language

3.3.3 Composition of ACGs

The abstract and object vocabularies are both higher-order signatures. A lexicon is a function. One can compose functions. Thus, if we have two lexicons $\mathscr{L}_1 : \Sigma_1 \longrightarrow \Sigma_2$ and $\mathscr{L}_2 : \Sigma_2 \longrightarrow \Sigma_3$, then we can consider a lexicon $\mathscr{L}_3 : \Sigma_1 \longrightarrow \Sigma_3$ that maps a constant (resp. a type) a of Σ_1 to a term (resp. type) over Σ_3 as follows: $\mathscr{L}_3(a) = \mathscr{L}_2(\mathscr{L}_1(a))$. One can check that $\mathscr{L}_3$ meets the requirements imposed on a lexicon, namely, it is a homomorphism; and for every constant a of Σ_1, if $a : \kappa$, $\mathscr{L}_3(a) : \mathscr{L}_3(\kappa)$. We denote $\mathscr{L}_3$ with $\mathscr{L}_2 \circ \mathscr{L}_1$. Relying on this property of the lexicon composition, one defines the *composition* of ACGs.

Definition 3.3.4 (ACG Composition).
Let $\mathcal{G}_1 = \langle \Sigma_1, \Sigma_2, \mathscr{L}_1, s_1 \rangle$ *and* $\mathcal{G}_2 = \langle \Sigma_2, \Sigma_3, \mathscr{L}_2, s_2 \rangle$ *be two ACGs. We define the ACG* $\mathcal{G}_3 = \langle \Sigma_1, \Sigma_3, \mathscr{L}_3, s_3 \rangle$ *as follows:*

$$s_3 = s_1 \quad \textit{and} \quad \mathscr{L}_3 = \mathscr{L}_2 \circ \mathscr{L}_1.$$

Thus, we have:

$$\mathcal{A}(\mathcal{G}_3) = \mathcal{A}(\mathcal{G}_1) = \{t \in \Lambda(\Sigma_1) \mid \vdash_{\Sigma_1} t : s_1 \quad \textit{is derivable}\}$$
$$\mathcal{O}(\mathcal{G}_3) = \{t \mid \exists u \in \mathcal{O}(\mathcal{G}_1) : \mathscr{L}_2(u) = t\}$$

We call $\mathcal{G}_3$ *the composition of ACGs* $\mathcal{G}_1$ *and* $\mathcal{G}_2$ *and denote with* $\mathcal{G}_2 \circ \mathcal{G}_1$.

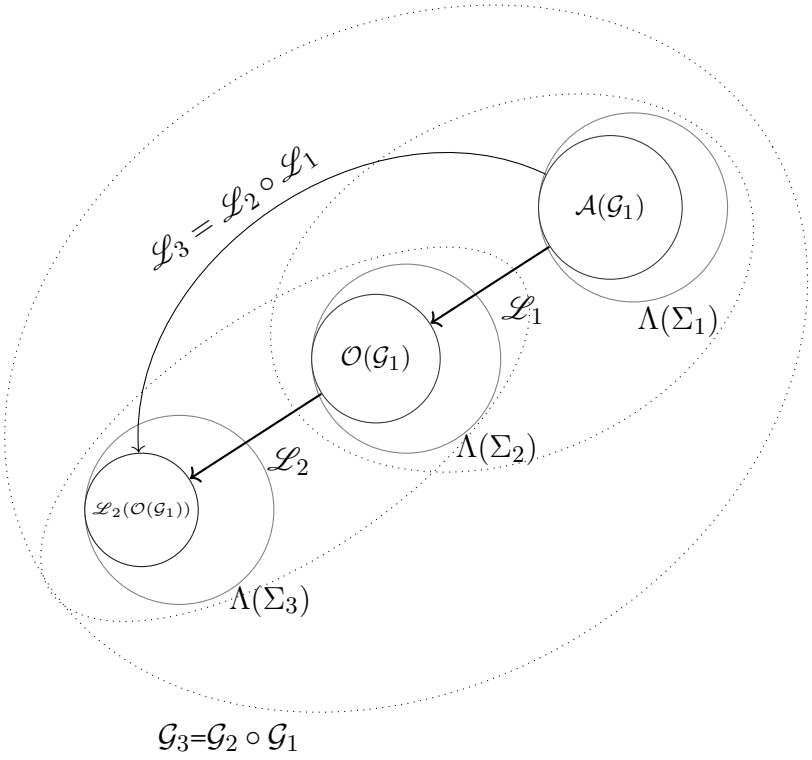


Figure 3.7: An ACG Composition

3.4 CFGs as ACGs

In this section, we discuss de Groote's (2001) encoding of CFGs[20] as ACGs. Let $\mathcal{G}$ be the ACG encoding of a CFG G. $\mathcal{G}$ has the following properties:

- The abstract language of $\mathcal{G}$ is the set derivations defined by G.
- The object language of $\mathcal{G}$ is the language of G.

Thus, the ACG encoding of a CFG G is strongly equivalent to G.

[20]We refer readers to See Section 2.3.2 on page 33 for the definitions and notations that we use for CFGs.

3.4.1 General Principles

We build two ACGs, $\mathcal{G}_1$ and $\mathcal{G}_2$. $\mathcal{G}_1$ establishes the correspondence between *derivations* of G and parse trees determined by G. That is, if the string of terminals ω is generated by the grammar G using some rules in some order, call it *a derivation* of ω, then $\mathcal{G}_1$ corresponds to this derivation a parse tree whose yield is ω. $\mathcal{G}_2$ interprets a parse tree as its yield. The composition of these two ACGs is the ACG encoding of G.

The ACG encoding of a context-free grammar relies on the following ideas:

- One identifies the *atomic types* of the abstract vocabulary with the *non-terminal symbols* of the grammar.
- One associates the *constants* of the abstract vocabulary with the *production rules* of the grammar.
- The *type* of a constant C_p associated with a production rule p is built with the help of the non-terminal symbols occurring in the production rule p.
- The lexicon of $\mathcal{G}_1$ maps an abstract constant C_p to a tree associated with a rule p.
- $\mathcal{G}_2$ is a yield ACG, defined in Section 3.3.1 on page 62.
- $\mathcal{G}_2 \circ \mathcal{G}_1$ is the ACG encoding of G.

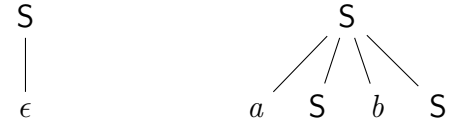
3.4.2 An Exemplifying Encoding

For the sake of illustration, we consider a CFG grammar $G = \langle N, \Sigma, P, \mathsf{S}\rangle$ generating the Dyck language. Figure 3.8 shows the production rules of this grammar. We construct the ACG encoding this grammar.

$$\begin{aligned} p_1 &: \quad \mathsf{S} \longrightarrow \epsilon \\ p_2 &: \quad \mathsf{S} \longrightarrow a\mathsf{S}b\mathsf{S} \end{aligned}$$

Figure 3.8: Production rules of the CFG G

We can associate with each production $\mathsf{X} \to \omega$ a tree whose root node is labeled with X. The children of the root node are the frontier nodes in the tree. The frontier nodes in the tree are labeled with the symbols from ω. The order of nodes follows the order of their labels in ω. For instance, the rules p_1 and p_2 can be represented as trees shown in Figure 3.9.



(a) p_1 (b) p_2

Figure 3.9: Trees representations of production rules

By definition, a string belongs to L_G if one can derive it from the start symbol S with the help of the production rules of G. Let $p^{'} : \mathsf{A}^{'} \rightarrow \omega^{'}$ and $p^{''} : \mathsf{A}^{''} \rightarrow \omega^{''}$ be production rules such that $\mathsf{A}^{'}$ has an occurrence in $\omega^{''}$. Let us select one of the occurrences of $\mathsf{A}^{'}$ into $\omega^{''}$. In a derivation step, we replace the selected occurrence of $\mathsf{A}^{'}$ in $\omega^{''}$ with $\omega^{'}$. This corresponds to the ***substitution*** of the tree associated with $p^{'}$ into the tree associated with $p^{''}$ at the node corresponding to the selected occurrence of $\mathsf{A}^{'}$ in $\omega^{''}$. For instance, let us consider a parse tree with the yield ab, shown in Figure 3.10(a). This parse tree corresponds to the following derivation:

$$\underbrace{\mathsf{S} \Longrightarrow_G a\mathsf{S}b\mathsf{S}}_{p_2} \underbrace{\Longrightarrow_G a\epsilon b\mathsf{S}}_{p_1} \underbrace{\Longrightarrow_G a\epsilon b\epsilon}_{p_1}$$

One can obtain the parse tree in Figure 3.10(a) by substituting the occurrences of S in the frontier of the tree associated with p_2 (see Figure 3.9(b)) by the tree associated with p_1 (see Figure 3.9(a)).

Thus, to each derivation corresponds a parse tree. We build an ACG $\mathcal{G}_1$ that allows one to interpret a derivation of a string as the corresponding parse tree. The object vocabulary is a tree signature so that terms over it model derivation trees of the CFG. Constants of the abstract vocabulary model production rules so that abstract terms model CFG derivations. To type a constant C_p modeling a production rule p, we refer to its tree representation. Let the non-terminals labeling the frontier of the tree representation of p be $\mathsf{X}_1, \ldots, \mathsf{X}_n$ (from left to right); and the root label be X. We type C_p with the type $\mathsf{X}_1 \multimap \ldots \multimap \mathsf{X}_n \multimap \mathsf{X}$. In other words, the non-terminal symbols $\mathsf{X}_1, \ldots, \mathsf{X}_n$ label ***substitution sites*** in the tree representation of p. At the node labeled with X_i, one can only substitute a tree whose root is labeled with X_i, for $i = 1, \ldots, n$. We model the substitution site labeled with X_i as an argument of C_p of type X_i, for $i = 1, \ldots, n$. If all the substitution sites are filled, one obtains a tree with the root labeled by X. This modeling is expressed by typing C_p with $\mathsf{X}_1 \multimap \ldots \multimap \mathsf{X}_n \multimap \mathsf{X}$.

Thus, to model the grammar in Figure 3.8, we introduce two constants C_{p_1} and C_{p_2} associated with the rules p_1 and p_2 respectively. We type C_{p_1} with S, whereas we type C_{p_2} with $\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$. A rewriting of an occurrence of S by ϵ in p_2 corresponds to an application of C_{p_2} to C_{p_1}. In derivation trees, the application of C_{p_2} to C_{p_1} is to substitute an occurrence of S at the frontier of the tree representation of p_2 with the tree representation of p_1. For instance, we model the parse trees in Figure 3.10 by the terms t_1 and t_2 respectively, defined as follows:

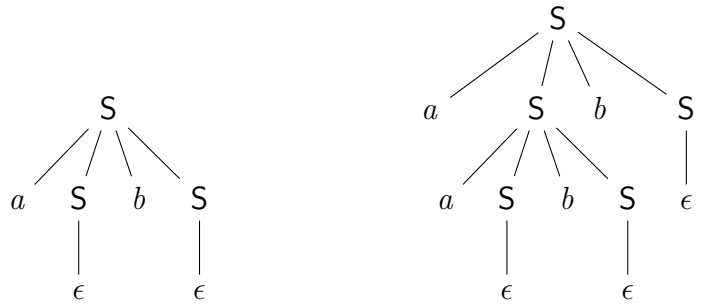
$$t_1 = C_{p_2}\ C_{p_1}\ C_{p_1} : \mathsf{S} \tag{3.8}$$

$$t_2 = C_{p_2}\ (C_{p_2}\ C_{p_1}\ C_{p_1})\ C_{p_1} : \mathsf{S} \tag{3.9}$$

Thus, we build the following abstract vocabulary:

$$\Sigma_1 = \{\{\mathsf{S}\},\ \{C_{p_1} : \mathsf{S}, C_{p_2} : \mathsf{S} \multimap \mathsf{S}\}\}$$

The abstract language is a set of closed terms of type S. The abstract language gives rise to $\mathtt{PTR}(G, \mathsf{S})$, i.e., the set of parse trees whose root is labeled with S.



(a) A parse tree with the yield ab

(b) A parse tree with the yield $aabb$

Figure 3.10: Two parse trees

$$\begin{array}{lll} C_{p_1} & :=_{\mathscr{L}_G} & \mathsf{S}_1\,\epsilon \\ C_{p_2} & :=_{\mathscr{L}_G} & \lambda^{o} xy.\,\mathsf{S}_4\,a\,x\,b\,y \end{array}$$

Figure 3.11: Interpretations of the constants modeling the production rules

To construct this set, we interpret constants of Σ_1 to the *trees* that they the give rise to. Thus, we build a lexicon $\mathscr{L}_G : \Sigma_1 \longrightarrow \Sigma^{\textit{tree}}_{N\cup\Sigma}$. Figure 3.11 shows the interpretations of C_{p_1} and C_{p_2}.

Where $\mathsf{S}_1 : \tau \multimap \tau$, $\mathsf{S}_4 : \tau \multimap \tau \multimap \tau \multimap \tau \multimap \tau$; a, b, and ϵ are of type τ. For instance, the terms t_1 and t_2 have the following interpretations under $\mathscr{L}_G$:

$$\mathscr{L}_G(t_1) \twoheadrightarrow_\beta \quad \mathsf{S}_4\,a\,(\mathsf{S}_1\,\epsilon)\,b\,(\mathsf{S}_1\,\epsilon) : \tau \tag{3.10}$$

$$\mathscr{L}_G(t_2) \twoheadrightarrow_\beta \quad \mathsf{S}_4\,a\,(\mathsf{S}_4\,a\,(\mathsf{S}_1\,\epsilon)\,b\,(\mathsf{S}_1\,\epsilon))\,b\,(\mathsf{S}_1\,\epsilon) : \tau \tag{3.11}$$

In this way, we build an ACG $\mathcal{G}_1 = \langle \Sigma_1, \Sigma^{\textit{tree}}_{N\cup\Sigma}, \mathscr{L}_G, \mathsf{S}\rangle$. The object language of $\mathcal{G}_1$ is the set $\mathtt{PTR}(G, \mathsf{S})$.

Afterwards, one can interpret terms modeling parse trees as their yields with the help of a yield ACG, in the same way as we did that in Section 3.3.1. Thus, we define $\mathcal{G}_2 = \langle \Sigma^{\textit{tree}}_{N\cup\Sigma}, \Sigma^{\textit{string}}_{\Sigma}, \mathscr{L}_{\mathsf{yield}}, \tau\rangle$. The composed ACG $\mathcal{G} = \mathcal{G}_2 \circ \mathcal{G}_1$ is the ACG encoding of G. The object language of the composed ACG is the string language defined by the original CFG G.

3.4.3 General Case

Given a CFG $G = \langle N, \Sigma, P, \mathsf{S}\rangle$, we use the following notations. One defines $[\![\omega]\!]_{\mathsf{X}}$ using a structural induction on ω (denoting a string of terminals and non-terminals) as follows:

1. $[\![\epsilon]\!]_{\mathsf{X}} = \mathsf{X}$;
2. $[\![\mathsf{Y}\omega]\!]_{\mathsf{X}} = \mathsf{Y} \multimap [\![\omega]\!]_{\mathsf{X}}$ if $\mathsf{Y} \in N$;
3. $[\![a\omega]\!]_{\mathsf{X}} = [\![\omega]\!]_{\mathsf{X}}$ if a is a terminal symbol, i.e., $a \in \Sigma$.

By $\vec{\sigma}$, we denote a sequence of variables and constants of some signature. We write $u, \vec{\sigma}$ to denote a sequence $u, u^1, \ldots, u^n$, where $\vec{\sigma}$ denotes the sequence $u^1, \ldots, u^n$.

We define $(\!|\omega|\!)$ as follows:

1. $(\!|\epsilon|\!) = (x; x)$.
2. $(\!|\mathsf{Y}\,\omega|\!) = (y, \vec{\sigma_1}; y, \vec{\sigma_2})$, where $(\!|\omega|\!) = (\!|\vec{\sigma_1}; \vec{\sigma_2}|\!)$, $\mathsf{Y} \in N$ and y is a fresh variable;
3. $(\!|a\,\omega|\!) = (\vec{\sigma_1}; a, \vec{\sigma_2})$, where $(\vec{\sigma_1}; \vec{\sigma_2}) = (\!|\omega|\!)$ and $a \in \Sigma$.

One builds an ACG $\mathcal{G}_1 = \langle \Sigma_1, \Sigma^{\mathit{tree}}_{N\cup\Sigma}, \mathscr{L}_G, \mathsf{S}\rangle$ strongly equivalent to $G = \langle N, \Sigma, P, \mathsf{S}\rangle$ as follows:

The Abstract Vocabulary $\Sigma_1 = \langle A_1, C_1, \tau_1\rangle$

- $\mathsf{X} \in A_1$ if and only if $\mathsf{X} \in N$, i.e., the set of the abstract atomic types of the ACG is exactly the same as the set of non-terminal symbols of the CFG.
- To each production rule $p : \mathsf{X} \to \omega$ corresponds a constant C_p of the abstract vocabulary Σ_1.
- If C_p is a constant of Σ_1 associated with a production rule $p : \mathsf{X} \to \omega$, then $\tau_1(\mathsf{C}_p) = [\![\omega]\!]_{\mathsf{X}}$.

The Object Vocabulary $\Sigma^{\mathit{tree}}_{N\cup\Sigma} = \langle \{\tau\}, C_2, \tau_\tau\rangle$

- For each $a \in \Sigma$, we have $a \in C_2$.
- For every $\mathsf{A} \in N$ and $\omega = u^1 \dots u^n$ such that there is a production rule in P of the form $\mathsf{A} \to \omega$, we have a constant $\mathsf{A}_n \in C_2$ (i.e., $\tau_\tau(A_n) = \underbrace{\tau \multimap \dots \multimap \tau}_{n\text{-}\mathit{times}} \multimap \tau$).

The distinguished symbol is the start symbol S.

The Lexicon $\mathscr{L}_G : \Sigma_1 \longrightarrow \Sigma^{\mathit{tree}}_{C_2}$

- $\mathscr{L}_G$ interprets every atomic type of Σ_1 as the tree type τ.
- $\mathscr{L}_G$ interprets each constant $\mathsf{C}_p \in C_1$, where $p : \mathsf{A} \to \omega$, as follows:

$$\mathscr{L}_G(\mathsf{C}_p) \;=\; \lambda^{o}\vec{\sigma_1}.\; \mathsf{A}_n \;\; u^1 \dots u^n$$

Where $(\!|\omega|\!) = (\vec{\sigma_1}; \vec{\sigma_2})$ and $\vec{\sigma_2}$ denotes the sequence $u^1, \dots, u^n$.

In order to obtain the string language, one defines a yield ACG $\mathcal{G}_2$.

The Abstract Vocabulary of $\mathcal{G}_2$ is $\Sigma^{\mathit{tree}}_{N\cup\Sigma} = \langle \{\tau\}, \{N\}, \tau_\tau\rangle$. That is, we declare the non-terminal symbols of G as the constants of the abstract vocabulary.

The Object Vocabulary of $\mathcal{G}_2$ is $\Sigma^{\mathit{string}}_{\Sigma} = \langle \{\sigma\}, \{\Sigma\}, \tau_\sigma\rangle$. That is, we declare the terminal symbols of G as the constants of the object vocabulary.

The distinguished type of $\mathcal{G}_2$ is τ.

The Lexicon $\mathscr{L}_{\mathsf{yield}} : \Sigma^{\mathit{tree}}_{C_2} \longrightarrow \Sigma^{\mathit{string}}_{\Sigma}$ is defined according Section 3.3.1.

The object vocabulary of $\mathcal{G}_1$ serves as the abstract vocabulary to $\mathcal{G}_2$. Thus, we can consider the composition of $\mathcal{G}_1$ and $\mathcal{G}_2$. The ACG composition of $\mathcal{G}_2 \circ \mathcal{G}_1$ establishes the correspondence between the derivations and the generated strings.

3.5 TAGs as ACGs

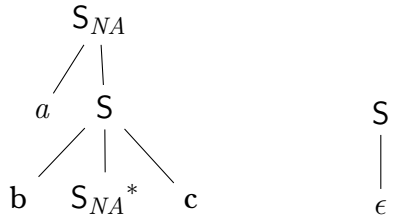
This section provides an encoding of a TAG as ACGs, proposed in (de Groote, 2001, 2002). Given a TAG G, the ACG encoding of G generates the object language isomorphic to the tree language generated by G.

3.5.1 General Principles

We already discussed how to model the operations of adjunction and substitution on trees as functional application of the λ°-terms modeling trees (see Section 3.2.2). In a TAG, a tree derived from an initial (resp. auxiliary) tree can substitute (resp. adjoin) at a given substitution (resp. adjunction) site in a tree if its root has the same label as the substitution (resp. adjunction) site does. However, the encoding of substitution and adjunction that we proposed does not have this property. To model the way TAG controls derived structures, for a substitution (resp. adjunction) site labeled with X, we introduce a type X (resp. $\mathsf{X_A}$). The substitution (resp. adjunction) of a tree δ at a node with the label X into a tree γ becomes the functional application where the operand, which models the tree δ, is a term of type X (resp. $\mathsf{X_A}$). In this way, we obtain terms modeling TAG *derivation* trees. To obtain TAG derived trees, we interpret terms modeling TAG derivation trees as terms modeling derived trees, i.e., terms over a tree signature. We interpret the types X and $\mathsf{X_A}$, modeling substitution and adjunction respectively, as τ and $\tau \multimap \tau$, respectively.

3.5.2 An Exemplifying Encoding

We consider a TAG G generating a non-context free language $\{a^n b^n c^n\}$ and model it as ACGs. Elementary trees of G are illustrated in Figure 3.12.



(a) A (b) I

Figure 3.12: Elementary trees of a TAG generating $\{a^n b^n c^n\}$

3.5.2.1 TAG Derivation Trees as Abstract Terms

To construct the abstract vocabulary encoding derivation trees of the TAG whose elementary trees are shown in Figure 3.12, in the abstract vocabulary we introduce two atomic types, $\mathsf{S_A}$ and S. The type S encodes S-substitution sites, whereas the type $\mathsf{S_A}$ encodes S-adjunction sites. The type S is the distinguished type of the ACG we are building, because the non-terminal S is the distinguished symbol of the grammar.

As the grammar consists of two elementary trees, we introduce two constants in the abstract vocabulary. We denote these constants by $\mathrm{C}_{\mathtt{A}}$ and $\mathrm{C}_{\mathtt{I}}$, where we associate $\mathrm{C}_{\mathtt{A}}$ with the auxiliary tree `A` (see Figure 3.12(a)) and $\mathrm{C}_{\mathtt{I}}$ with the initial tree `I` (see Figure 3.12(b)). These constants do not directly encode syntactic trees, but rather correspond to *the nodes in TAG derivation trees that stand for these elementary trees.* To put it another way, for each elementary tree, we introduce its representative constant in the abstract vocabulary, similar to a node representing that elementary tree in a TAG derivation tree. We propose the following abstract vocabulary:

$$\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}G = \{\mathrm{C}_{\mathtt{I}} : \mathsf{S}_{\mathsf{A}} \multimap \mathsf{S},\ \mathrm{C}_{\mathtt{A}} : \mathsf{S}_{\mathsf{A}} \multimap \mathsf{S}_{\mathsf{A}},\ \mathrm{I}_{\mathsf{S}_{\mathsf{A}}} : \mathsf{S}_{\mathsf{A}}\}$$

$\mathrm{C}_{\mathtt{I}}$ is of type $\mathsf{S}_{\mathsf{A}} \multimap \mathsf{S}$, which one constructs by the breadth first, left to right traversing the initial tree `I`. Indeed, the type $\mathsf{S}_{\mathsf{A}} \multimap \mathsf{S}$ encodes the fact that the initial tree `I` can receive an S-adjunction (a term of type S_{A}). By adjoining a tree into the initial tree `I`, one obtains an S-rooted completed TAG derived tree. The byproduct of this derived tree is a TAG derivation tree, which we encode with a term of type S.

$\mathrm{C}_{\mathtt{A}}$ is of type $\mathsf{S}_{\mathsf{A}} \multimap \mathsf{S}_{\mathsf{A}}$, which one constructs by the breadth first, left to right traversing the auxiliary tree `A`. The type $\mathsf{S}_{\mathsf{A}} \multimap \mathsf{S}_{\mathsf{A}}$ encodes the fact that the auxiliary tree `A` receives an adjunction on its S node and the resultant tree (a term of type S_{A}) can adjoin on an S-node of some tree.

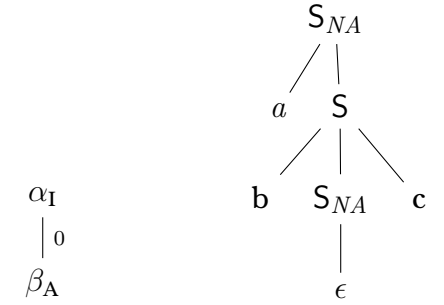
Since we use simple types, if a constant models a tree with an X-adjunction site, then this adjunction site is obligatory. For instance, $\mathrm{C}_{\mathtt{I}}$ is of type $\mathsf{S}_{\mathsf{A}} \multimap \mathsf{S}$, which models that the initial tree `I` has an S-adjunction site. This adjunction is not obligatory in `I`. Thus, one can use `I` as a derived tree of the TAG language. However, as $\mathrm{C}_{\mathtt{I}}$ is of type $\mathsf{S}_{\mathsf{A}} \multimap \mathsf{S}$, it *must* apply to some term of type S_{A} in order to produce a term of type S. To be able to handle cases where no adjunction takes place at an adjunction site modeled by the type X_{A}, de Groote (2002) introduces a constant $\mathrm{I}_{\mathsf{X}_{\mathsf{A}}}$ of type X_{A} in the abstract vocabulary. $\mathrm{I}_{\mathsf{X}_{\mathsf{A}}}$ can be seen as an X-rooted *empty* (*fake*) auxiliary tree, as it adjoins at an X-adjunction site in a tree but does not modify the tree. However, once it adjoins in some tree, no other adjunction can be made at that adjunction site of the tree. One can view $\mathrm{I}_{\mathsf{S}_{\mathsf{A}}}$ as an auxiliary tree X^*_{NA}. Indeed, X^*_{NA} consists of a single node, which is its root and foot node at the same time, and it accepts no adjunction. For instance, to model the case where `I` is a derived tree of the TAG tree language, we define a term $\mathrm{C}_{\mathtt{I}}\,\mathrm{I}_{\mathsf{S}_{\mathsf{A}}}$ whose type is indeed S.

Remark 3.2. *Figure 3.13 shows a representation of the term* $\mathrm{C}_{\mathtt{I}}\,(\mathrm{C}_{\mathtt{A}}\,\mathrm{I}_{\mathsf{S}_{\mathsf{A}}})$ *as a tree. This tree is reminiscent of the derivation tree in Figure 3.14(a) on the facing page. The main difference between the two is due to the occurrence of the empty adjunction* $\mathrm{I}_{\mathsf{S}_{\mathsf{A}}}$ *in the term. The presence of empty adjunctions is an explicit declaration that no adjunction takes place. In the case of a TAG derivation tree, however, it is implied that if there is no information about an adjunction on a node into a tree, then there is no adjunction on that node.*

3.5.2.2 Derived Trees as Object Terms

We encode TAG derived trees as terms over $\Sigma^{\mathit{tree}}_{N\cup\Sigma}$. Figure 3.12 shows the elementary trees `I` and `A`. To encode them, in the tree signature $\Sigma^{\mathit{tree}}_{N\cup\Sigma}$, we have constants,

Figure 3.13: A tree-representation of a term modeling a derivation tree



(a) A derivation tree

(b) A derived tree

Figure 3.14: A derivation and a derived tree

$\mathsf{S}_1 : \tau \multimap \tau$, $\mathsf{S}_2 : \tau \multimap \tau \multimap \tau$, and $\mathsf{S}_3 : \tau \multimap \tau \multimap \tau \multimap \tau$, which model symbols of arity 1, 2, and 3, respectively.

3.5.2.3 Interpretations as Derived Trees

We map the constants $\mathrm{C}_{\mathbf{I}}$ and $\mathrm{C}_{\mathbf{A}}$ to the terms over Σ_G^{tree} that serve as the encodings of the elementary trees `I` and `A`. Thus, we build a lexicon $\mathscr{L}_{\text{synt}}^{\text{TAG}}$ interpreting $\mathrm{C}_{\mathbf{I}}$ and $\mathrm{C}_{\mathbf{A}}$ (see Figure 3.15). It remains to interpret $\mathrm{I}_{\mathsf{S_A}} : \mathsf{S_A}$. Since $\mathrm{I}_{\mathsf{S_A}}$ models an empty adjunction, it should not change a tree into which it adjoins. We interpret it as an identity function $\lambda^{\mathrm{o}} x.x : \tau \multimap \tau$.

Types and Constant of Σ_1	Their interpretations under $\mathscr{L}_{\text{synt}}^{\text{TAG}}$
$\mathsf{S_A}$	$\tau \multimap \tau$
S	τ
$\mathrm{C}_{\mathbf{I}}$	$\lambda^{\mathrm{o}}\ P.\ P\ (\mathsf{S}_1\ \epsilon)$
$\mathrm{C}_{\mathbf{A}}$	$\lambda^{\mathrm{o}}\ P.\ \lambda^{\mathrm{o}}\ x.\ \mathsf{S}_2\ a\ (P\ (\mathsf{S}_3\ b\ x\ c))$
$\mathrm{I}_{\mathsf{S_A}}$	$\lambda^{\mathrm{o}} x.x$

Figure 3.15: The lexicon interpreting TAG derivation trees into TAG derived trees

In order to obtain a derived tree specified by a derivation tree, one interprets the term encoding the derivation tree by the lexicon. For instance, to obtain the term

modeling the derived tree depicted in Figure 3.14(b), the lexicon $\mathscr{L}_{\text{synt}}^{\text{TAG}}$ interprets the term $\mathrm{C}_{\mathbf{I}}\ (\mathrm{C}_{\mathbf{A}}\ \mathrm{I}_{\mathsf{S}_\mathsf{A}})$ as follows:

$$\mathscr{L}_{\text{synt}}^{\text{TAG}}(\mathrm{C}_{\mathbf{I}}\ (\mathrm{C}_{\mathbf{A}}\ \mathrm{I}_{\mathsf{S}_\mathsf{A}})) = \mathscr{L}_{\text{synt}}^{\text{TAG}}(\mathrm{C}_{\mathbf{I}})\ (\mathscr{L}_{\text{synt}}^{\text{TAG}}(\mathrm{C}_{\mathbf{A}})\ \mathscr{L}_{\text{synt}}^{\text{TAG}}(\mathrm{I}_{\mathsf{S}_\mathsf{A}})) =$$
$$(\lambda^{\mathrm{o}}\, P.\, P\,(\mathsf{S}_1\,\epsilon))((\lambda^{\mathrm{o}}\, P.\lambda^{\mathrm{o}} x.\mathsf{S}_2\ a\ (P\,(\mathsf{S}_3\ b\ x\ c)))\ (\lambda^{\mathrm{o}}\, x.\ x)) \twoheadrightarrow_\beta \mathsf{S}_2\ a\ (\mathsf{S}_3\ b\,(\mathsf{S}_1\,\epsilon)\ c)\ \ :\tau$$

3.5.2.4 Yields as Object Terms

In order to obtain yields of TAG derived trees, we build *a yield ACG*, defined in Section 3.3.1. By composing the latter ACG with the former, we obtain an ACG that provides interpretations of TAG derivation trees as yields of TAG derived trees.

3.5.3 General Case

Given a TAG $G = \langle N, \Sigma, I, A, \mathsf{S}\rangle$, we build an ACG $\mathcal{G}_G = \langle \Sigma_1, \Sigma^{\textit{tree}}_{\Sigma\cup N}, \mathscr{L}^{\text{TAG}}_{\text{synt}}, s\rangle$. The abstract language $\mathcal{A}(\mathcal{G}_G)$ models the derivation trees of G, wheres the object language $\mathcal{O}(\mathcal{G}_G)$ is isomorphic to the tree language of G.

The Abstract Vocabulary $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$

- Σ_1 has type a X (resp. X_A) for every non-terminal symbol in N labeling a substitution (resp. adjunction) site in some elementary tree of G.
- For each elementary tree γ of G, Σ_1 has a constant C_γ. If γ is an X-initial (resp. X-auxiliary) tree, the we type C_γ with the type $\vec{\alpha}$, where $\vec{\alpha}$ is of the form $\alpha_1 \multimap \ldots \multimap \mathsf{X}$ (resp. $\alpha_1 \multimap \ldots \multimap \mathsf{X}_\mathsf{A}$). To construct $\vec{\alpha}$, we make the breadth first left to right traversal of γ and record the substitution and adjunction sites in the order they appear in the traversal.
- For every X labeling a non-obligatory adjunction site, Σ_1 contains a constant $\mathrm{I}_{\mathsf{X}_\mathsf{A}} : \mathsf{X}_\mathsf{A}$ modeling an empty X-adjunction.

The Object Vocabulary $\Sigma^{\textit{tree}}_{\Sigma\cup N}$ is a tree signature over $\Sigma \cup N$.

The Lexicon $\mathscr{L}^{\text{TAG}}_{\text{synt}} : \Sigma_1 \longrightarrow \Sigma^{\textit{tree}}_{\Sigma\cup N}$ interprets types and constants as follows:

- $\mathscr{L}^{\text{TAG}}_{\text{synt}}(\mathsf{X}) = \tau$ and $\mathscr{L}^{\text{TAG}}_{\text{synt}}(\mathsf{X}_\mathsf{A}) = \tau \multimap \tau$.
- $\mathscr{L}^{\text{TAG}}_{\text{synt}}(\mathrm{C}_\gamma) = u_\gamma$, where C_γ is a constant associated with an elementary tree γ; and u_γ is a λ^{o}-term encoding γ.
- $\mathscr{L}^{\text{TAG}}_{\text{synt}}(\mathrm{I}_{\mathsf{X}_\mathsf{A}}) = \lambda^{\mathrm{o}} x.x$, where X_A is an abstract type modeling X-adjunction sites.

3.5.4 The ACG Encoding of an Exemplifying LTAG for a Fragment of English

We provide an exemplifying LTAG to analyze the following English sentences, which we use in further chapters as well:

(12) a. Fred is really grumpy.

b. Fred failed an important exam.

c. Fred is grumpy because he failed an exam.

d. John Mary seems to love.

As we will refer to this encoding in further chapters, we name signatures and lexicons used in this example as follows:

- $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ denotes the abstract vocabulary, where we encode derivation trees;
- $\Sigma^{\mathsf{Synt}}_{\mathsf{TAG}}$ denotes the object vocabulary, where we encode derived trees.

Table 3.1 on the next page provides constants in $\Sigma^{\mathsf{Synt}}_{\mathsf{TAG}}$ modeling derivation trees of TAG elementary trees anchored with lexical items from Example (12) and their translations in TAG derived trees. In particular, Table 3.1 shows the encodings of initial trees anchored with verbs, such as *failed*, *seems*, etc. To model their substitution and adjunction sites, we use types $\mathsf{np}, \mathsf{S}, \mathsf{S}_{\mathsf{A}}, \mathsf{V}_{\mathsf{A}}$ etc. In addition, Table 3.1 contains constants encoding nouns, adjectives, articles and quantifiers. We use the type np to model noun phrases. n_{A} and n_d are types encoding adjunctions on an initial tree anchored with a *common noun* (CN). For the types n_{A} and n_d, as for any type encoding a non-obligatory adjunction site, we have the abstract constants encoding empty adjunctions of these types, $\mathrm{I}_{\mathsf{n}_{\mathsf{A}}} : \mathsf{n}_{\mathsf{A}}$ and $\mathrm{I}_{\mathsf{n}_d} : \mathsf{n}_d$, respectively. Notice that we type $\mathrm{C}_{\mathbf{CN}}$ with $\mathsf{n}_d \multimap \mathsf{n}_{\mathsf{A}} \multimap \mathsf{np}$. That is, we have two different possible adjunctions at the node of an initial tree anchored by a common noun, which is not a standard interpretation. In other words, instead of the initial tree anchored by a CN, $\begin{smallmatrix}\mathsf{N}\\ |\\ \mathit{CN}\end{smallmatrix}$, we interpret $\mathrm{C}_{\mathbf{CN}}$ as a tree $\begin{smallmatrix}\mathsf{Nad}\\ |\\ \mathsf{Na}\\ |\\ \mathit{CN}\end{smallmatrix}$. The way we interpret CNs is motivated by the way (XTAG-Group, 1998) encodes them, which is expressed by the following quote:

> Common nouns do not require determiners in order to form grammatical NPs ... Common nouns have negative("−") values for determiner features in the lexicon in our analysis and can only acquire a positive("+") value for those features if determiners adjoin to them. XTAG-Group (1998)

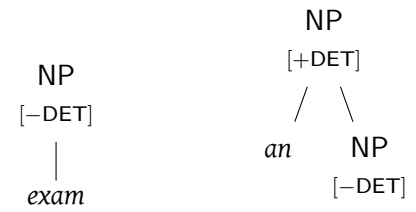
Thus, according to (XTAG-Group, 1998), a CN anchors an NP-initial tree whose root has a feature $-\mathsf{DET}$. The feature $-\mathsf{DET}$ of the root node becomes $+\mathsf{DET}$ if a determiner adjoins on it. Figure 3.16 shows the XTAG analyses of determiners and CNs. Since we do not use features but types, we use the type n_d in order to enable an adjunction of a determiner into an N-initial tree anchored with a CN. By adjoining a determiner in an N-initial tree anchored with a CN, we obtain an NP-derived tree. Hence, in our notations N is NP with the feature $-\mathsf{DET}$, whereas NP is NP with the feature $+\mathsf{DET}$.

Example 3.4. We model the derivation the trees of the sentences listed in Example (12) as the terms in $\Lambda(\Sigma^{\mathsf{Der}}_{\mathsf{TAG}})$ shown in Figure 3.17.

For instance, the terms $t_3 : \mathsf{S}$ and $t_4 : \mathsf{S}$ model the TAG derivation trees depicted in Figure 3.18(b) and Figure 3.20(b) on page 76, respectively.

Abstract constants of $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$	Their images by $\mathscr{L}^{\mathrm{TAG}}_{\mathrm{synt}}$	The corresponding TAG trees
$\mathrm{C}_{\mathrm{Fred}} : \mathsf{np}$	$: \tau$ $= \mathsf{NP}_1\ \mathit{Fred}$	$\gamma_{\mathit{Fred}} =$ NP | *Fred*
$\mathrm{C}_{\mathrm{grumpy}} : \mathsf{S_A} \multimap \mathsf{V_A} \multimap \mathsf{np} \multimap \mathsf{S}$	$: (\tau \multimap \tau) \multimap (\tau \multimap \tau) \multimap \tau \multimap \tau$ $= \lambda^{\mathrm{o}} avs.\ a\ (\mathsf{S}_2\ s\ (v\ (\mathsf{VP}_2\ (\mathsf{VP}_1\ \epsilon)\ \mathit{grumpy})))$	$\gamma_{\mathit{grumpy}} =$ S NP$_\downarrow$ VP ϵ grumpy
$\mathrm{C}_{\mathrm{failed}} : \mathsf{S_A} \multimap \mathsf{V_A} \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{S}$	$: (\tau \multimap \tau) \multimap (\tau \multimap \tau) \multimap \tau \multimap \tau \multimap \tau$ $= \lambda^{\mathrm{o}} avso.\ a\ (\mathsf{S}_2\ s\ (v\ (\mathsf{VP}_2\ \mathit{failed}\ o)))$	$\gamma_{\mathit{failed}} =$ S NP$_\downarrow$ VP *failed* NP$_\downarrow$
$\mathrm{C}_{\mathrm{is}} : \mathsf{V_A} \multimap \mathsf{V_A}$	$: (\tau \multimap \tau) \multimap (\tau \multimap \tau)$ $= \lambda^{\mathrm{o}} vx.v\ (\mathsf{VP}_2\ \mathit{is}\ x)$	$\gamma_{\mathit{is}} =$ VP *is* VP*
$\mathrm{C}_{\mathrm{seems}} : \mathsf{V_A} \multimap \mathsf{V_A}$	$: (\tau \multimap \tau) \multimap (\tau \multimap \tau)$ $= \lambda^{\mathrm{o}} vx.v\ (\mathsf{VP}_2\ \mathit{seems}\ x)$	$\gamma_{\mathit{seems}} =$ VP *seems* VP*
$\mathrm{C}_{\mathrm{to\ love}} : \mathsf{S_A} \multimap \mathsf{V_A} \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{S}$	$: (\tau \multimap \tau) \multimap (\tau \multimap \tau) \multimap \tau \multimap \tau \multimap \tau$ $= \lambda^{\mathrm{o}}\ avso.\mathsf{S}_2\ o\ (a_1\ (\mathsf{S}_2\ s\ (v\ (\mathsf{VP}_1\ \mathit{to\ love}))))$	$\gamma_{\mathit{to\ love}} =$ S NP$_\downarrow$ S NP$_\downarrow$ VP | *to love*
$\mathrm{C}_{\mathrm{because}} : \mathsf{S_A} \multimap \mathsf{S} \multimap \mathsf{S_A}$	$: (\tau \multimap \tau) \multimap \tau \multimap (\tau \multimap \tau)$ $= \lambda^{\mathrm{o}} asx.\ a\ (\mathsf{S}_2\ x\ (\mathsf{S}_2\ \mathit{because}\ s))$	$\gamma_{\mathit{because}} =$ S S* S *because* S$_\downarrow$
$\mathrm{C}_{\mathrm{really}} : \mathsf{V_A} \multimap \mathsf{V_A}$	$: (\tau \multimap \tau) \multimap (\tau \multimap \tau)$ $= \lambda^{\mathrm{o}} vx.v\ (\mathsf{VP}_2\ \mathit{really}\ x)$	$\gamma_{\mathit{really}} =$ VP *really* VP*
$\mathrm{C}_{\mathrm{exam}} : \mathsf{n}_d \multimap \mathsf{n_A} \multimap \mathsf{np}$	$: (\tau \multimap \tau) \multimap (\tau \multimap \tau) \multimap \tau$ $= \lambda^{\mathrm{o}}\ da.d\ (a\ (\mathsf{N}_1\ \mathit{exam}))$	$\gamma_{\mathit{exam}} =$ N | *exam*
$\mathrm{C}_{\mathrm{important}} : \mathsf{n_A} \multimap \mathsf{n_A}$	$: (\tau \multimap \tau) \multimap (\tau \multimap \tau) \multimap \tau$ $= \lambda^{\mathrm{o}}\ a\ n.\ a\ (\mathsf{N}_2\ \mathit{important}\ n)$	$\gamma_{\mathit{important}} =$ N *important* N*
$\mathrm{C}_{\mathrm{an}} : \mathsf{n}_d$	$: \tau \multimap \tau$ $= \lambda^{\mathrm{o}}\ n.\ \mathsf{NP}_2\ \mathit{a}\ n$	$\gamma_{\mathit{a}} =$ NP *a* N*
$\mathrm{C}_{\mathrm{each}} : \mathsf{n}_d$	$: \tau \multimap \tau$ $= \lambda^{\mathrm{o}}\ n.\ \mathsf{NP}_2\ \mathit{each}\ n$	$\gamma_{\mathit{each}} =$ NP *each* N*
$\mathrm{C}_{\mathrm{every}} : \mathsf{n}_d$	$: \tau \multimap \tau$ $= \lambda^{\mathrm{o}}\ n.\ \mathsf{NP}_2\ \mathit{every}\ n$	$\gamma_{\mathit{every}} =$ NP *every* N*
$\mathrm{I}_{\mathsf{X}} : \mathsf{X_A}$	$\lambda^{\mathrm{o}} x.x : \tau \multimap \tau$	

Table 3.1: TAG as ACG: the $\mathscr{L}^{\mathrm{TAG}}_{\mathrm{synt}}$ lexicon



(a) A tree anchored with an CN

(b) A tree anchored with an article

Figure 3.16: XTAG analyses of determiners and CNs

t_1	$\mathrm{C}_{\mathbf{grumpy}}\ \mathrm{I}_{\mathsf{S_A}}\ (\mathrm{C}_{\mathbf{is}}\ (\mathrm{C}_{\mathbf{very}}\ \mathrm{I}_{\mathsf{V_A}}))\ \mathrm{C}_{\mathbf{Fred}}\ :\ \mathsf{S}$
t_2	$\mathrm{C}_{\mathbf{failed}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{C}_{\mathbf{Fred}}\ (\mathrm{C}_{\mathbf{exam}}\ \mathrm{I}_{\mathsf{n_A}}\ \mathrm{C}_{\mathbf{an}})\ :\ \mathsf{S}$
t_3	$\mathrm{C}_{\mathbf{grumpy}}\ (\mathrm{C}_{\mathbf{because}}\ \mathrm{I}_{\mathsf{S_A}}\ (\mathrm{C}_{\mathbf{failed}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{C}_{\mathbf{he}}\ (\mathrm{C}_{\mathbf{exam}}\ \mathrm{I}_{\mathsf{n_A}}\ \mathrm{C}_{\mathbf{an}})))\ (\mathrm{C}_{\mathbf{is}}\ \mathrm{I}_{\mathsf{V_A}})\ \mathrm{C}_{\mathbf{Fred}}\ :\ \mathsf{S}$
t_4	$\mathrm{C}_{\mathbf{to\ love}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{S_A}}\ (\mathrm{C}_{\mathbf{seems}}\ \mathrm{I}_{\mathsf{V_A}})\ \mathrm{C}_{\mathbf{John}}\ \mathrm{C}_{\mathbf{Mary}}\ :\ \mathsf{S}$

Figure 3.17: Examples of terms over $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ modeling LTAG derivation trees

Example 3.5.

Let us provide some examples of the terms modeling TAG derivation trees. One obtains the corresponding TAG derived trees by interpreting those terms by the lexicon $\mathscr{L}^{\mathrm{TAG}}_{\mathrm{synt}}$ (defined in Table 3.1 on the preceding page). In particular, we consider the terms t_3 and t_4, defined in Table 3.17 . Table 3.19 shows the interpretations of the terms t_3 and t_4 under the lexicon $\mathscr{L}^{\mathrm{TAG}}_{\mathrm{synt}}$. As one can see, the terms $\mathscr{L}^{\mathrm{TAG}}_{\mathrm{synt}}(t_3)$ and $\mathscr{L}^{\mathrm{TAG}}_{\mathrm{synt}}(t_4)$ model the TAG derived trees shown in Figure 3.18(c) and Figure 3.20(c), respectively.[21]

3.6 The ACG Hierarchy of Languages

One defines the *order of an ACG* and the *order of a lexicon* as follows.

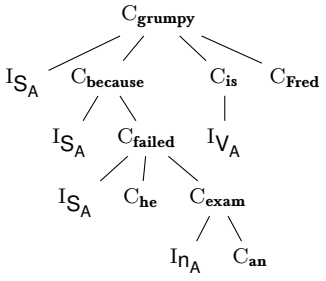
Definition 3.6.1 (Order of an ACG and a Lexicon).

- *The order of an ACG is the maximum of the orders of the types of the constants in the abstract vocabulary.*
- *The order of the lexicon $\mathscr{L} = \langle F, G\rangle$ of an ACG $\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, s\rangle$, where $\Sigma_1 = \langle A_1, C_1, \tau_1\rangle$, is defined as follows:*

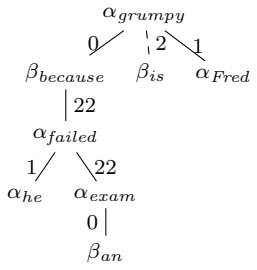
$$\mathfrak{ord}(\mathscr{L}) = \max_{\alpha \in A_1} \mathfrak{ord}(F(\alpha))$$

Thus, the order of an ACG is the order of the abstract vocabulary. The order of a lexicon is the maximum of the orders of types of images of the abstract constants. For

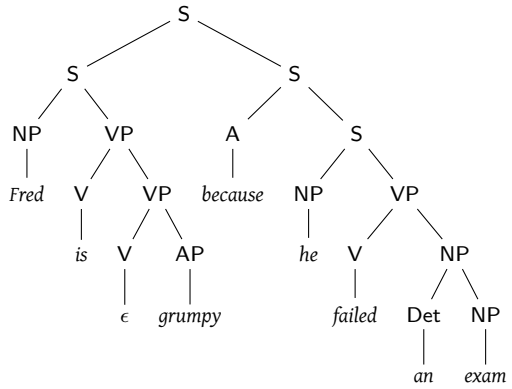
[21] In Appendix A, we provide ACG codes for these examples, which one can run using the ACG development software (the ACG toolkit).



(a) A term as a tree



(b) A TAG derivation tree

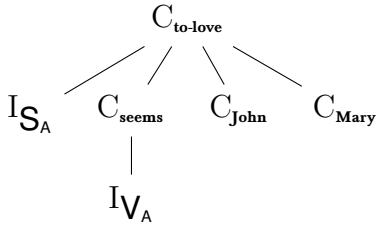


(c) A TAG derived tree

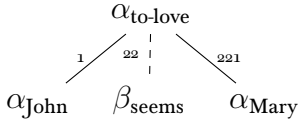
Figure 3.18: A derivation tree, a term modeling it, and a derived tree

Terms over $\Sigma^{\text{Der}}_{\text{TAG}}$	Their interpretations under $\mathscr{L}^{\text{TAG}}_{\text{synt}}$
t_3	S_2 (S_2 (NP_1 *Fred*) (VP_2 *is* (VP_2 (VP_1 ϵ) *grumpy*))) (S_2 *because* (S_2 (N_1 *Fred*)(VP_2 *failed* (NP_2 *an* (N_1 *exam*))))) : τ
t_4	S_2 (NP_1 *Mary*) (S_2 (NP_1 *John*) (VP_2 *seems* (VP_1 *to-love*))) : τ

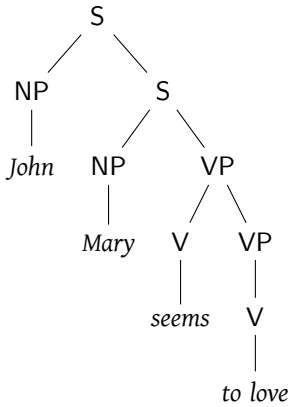
Figure 3.19: Interpretations of terms over $\Sigma^{\text{Der}}_{\text{TAG}}$ under the lexicon $\mathscr{L}^{\text{TAG}}_{\text{synt}}$



(a) A term as a tree



(b) A TAG derivation tree



(c) A TAG derived tree

Figure 3.20: A term as a tree, a TAG derivation tree, and a TAG derived tree

instance, in de Groote's (2002) encoding of TAGs as ACGs, the lexicon is second-order. Indeed, an abstract type X models either an adjunction site, or a substitution site, or the distinguished symbol of the grammar. In the case of adjunction, one maps X to $\tau \multimap \tau$ (a type of order 2); otherwise, one maps X to τ.

Definition 3.6.2 (Class of ACGs). *By $G(m, n)$ we denote the subclass of ACGs whose members have the following property: $\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, s \rangle$ is in this class if $\mathfrak{ord}(\Sigma_1) \leq m$ and $\mathfrak{ord}(\mathscr{L}) \leq n$.*

We call[22] a *string* (resp. *tree*) ACG an ACG whose *object* vocabulary is a *string* (resp. *tree*) signature and whose distinguished type is mapped to the type σ (resp. τ) of a string (resp. tree) signature defined in Definition 3.2.7 (resp. Definition 3.2.8). We denote a class of string (resp. tree) ACGs with G^{string} (resp. G^{tree}).

Definition 3.6.3 (Yoshinaka, 2006).
Let G be an ACG $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, s \rangle$. A constant $c \in \Sigma_1$ is called lexical *in G, if $\mathscr{L}(c)$ contains a constant of Σ_2. If each abstract constant of an ACG G is lexical, then G is called lexicalized. We denote the class of lexicalized ACGs by G_{lex}.*

With the help of these notions, one can define various classes of ACGs. For instance, $G^{tree}_{lex}(2, 2)$ denotes the class of second-order lexicalized tree ACGs whose lexicons are second-order.

3.6.1 Second-Order ACGs

We already discussed the ACG encodings of CFGs and TAGs (de Groote, 2001, 2002). Both of the encodings are second-order ones. It is noteworthy that second-order ACGs can encode a number of more powerful grammatical formalisms than CFGs and TAGs (de Groote and Pogodalla, 2003, 2004). Second-order ACGs are lexicalized by second-order ACGs (Kanazawa and Yoshinaka, 2005). At the same time, the problems of parsing and generation with a second-order ACG are polynomial (Kanazawa, 2007; Salvati, 2005).

Proposition 3.6.1 (Kanazawa and Yoshinaka, 2005). *For each second-order ACG $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, s \rangle$, there is an RTG G such that the tree language generated by G is isomorphic to the* abstract language *generated by the ACG $\mathscr{G}$.*

3.6.1.1 String Languages

De Groote and Pogodalla (2004) encode several formalisms as second-order ACGs. They show that the string languages generated by context-free string grammars are included in the class $G^{string}(2, 2)$. It is noteworthy that the ACGs in the class $G^{string}(2, 2)$ generate only context-free string languages, that it, the class of context-free string grammars and $G^{string}(2, 2)$ coincide (see Table 3.2[23]). The string languages generated

[22] Following the terminology of (Kanazawa, 2006; Yoshinaka, 2006).
[23] We borrow Table 3.2 from (Yoshinaka, 2006).

by linear context-free tree grammars[24] are included in the class $G^{string}(2,3)$. Any linear context free rewriting system (LCFRS) can be encoded with some ACG from the class $G^{string}(2,4)$.

Theorem 3.6.1 (Salvati, 2005). *Any second-order string ACG in $G^{string}(2,n)$ for $n \geq 1$ can be encoded with an equivalent LCFRS.*

Since for each LCFRS, there exists its equivalent ACG from the class $G^{string}(2,4)$, Salvati's (2005) theorem entails the following corollary:

Corollary 3.6.1.1 (Salvati, 2005). *For every second-order string ACG from the class $G^{string}(2,n)$, there is an equivalent ACG in the class $G^{string}(2,4)$.*

Context-Free Grammars	=	$G^{string}(2,2)$
Linear Context Free-Tree Grammars	$\subseteq$	$G^{string}(2,3)$
Linear Context Free Rewriting Systems	=	$G^{string}(2,n)$ for $n \geq 4$

Table 3.2: The ACG hierarchy of string languages

3.6.1.2 Tree Languages

To discuss the generative power of second-order tree ACGs, it is useful to define the sub-class of second-order ACGs, called *relabeling* second-order ACGs (Yoshinaka, 2006).

Definition 3.6.4 (Yoshinaka, 2006).
We say that the lexicon $\mathscr{L}$ of a second-order ACG is relabeling *if (1) $\mathscr{L}$ is first-order; and (2) $\mathscr{L}$ maps any abstract constant to some object constant.*

We define the class of second-order relabeling ACGs, denoted with $G(2,1(r))$, as the class of second-order ACGs whose lexicons are relabeling.

Since the lexicon of a second-order relabeling ACG $\mathcal{G} \in G(2,1(\mathrm{r}))$ is first-order, every abstract (atomic) type is mapped to an atomic type. Thus, the abstract and object languages defined by such an ACG are isomorphic.

The class of RTGs generate the same tree languages as $G^{tree}(2,1(r))$. Indeed, the abstract language of a second-order ACG is a regular tree language. Since the object language of any ACG in $G^{tree}(2,1(r))$ is isomorphic to the abstract language of that ACG, we conclude that the class of languages generated by $G^{tree}(2,1(r))$ is a subclass of the class of regular tree languages. On the other hand, given an RTG G, there is an ACG in $G^{tree}(2,1(r))$ whose object language is isomorphic to the tree language of G (Kanazawa and Yoshinaka, 2005).

The ACG encoding of TAG by de Groote (2001, 2002) (with the tree signature as the object vocabulary) falls into the class $G^{tree}(2,2)$. In addition, de Groote and

[24]Here, *linear* means *non-duplicating* (i.e non-copying) and *non-deleting*, whereas usually in the context of grammars, it means only *non-duplicating*. Some authors (e.g. (Maletti and Engelfriet, 2012)), instead of *linear*, use *simple* in order to refer to *non-duplicating* and *non-deleting* context-free tree grammars.

Pogodalla (2004) show that ACGs in $G^{tree}(2,2)$ generate exactly the same languages as linear context free tree grammars. We provide the ACG hierarchy of tree languages in Table 3.3.[25]

Regular Tree Grammars	=	$G^{tree}(2,1)$
Tree Adjoining Grammars	$\subseteq$	$G^{tree}(2,2)$
Linear Context-Free Tree Grammars	$=$	$G^{tree}(2,2)$

Table 3.3: The ACG hierarchy of tree languages

3.6.2 ACGs of Order $n \geq 3$

For the order of 3 or above, there is an ACG in $G^{tree}_{lex}(3,1)$ generating an NP-complete language (Salvati, 2005, 2010). Salvati (2005) showed that in general, the decision problem whether a term belongs to the (object) language generated by an ACG is equivalent to the decidability of Multiplicative-Exponential Linear Logic (MELL) (Girard, 1987a,b), which recently was shown to be decidable (Bimbó, 2015).

3.7 Second-Order Almost-Linear ACGs (λ-CFGs)

The original definition of ACGs employs the notion of *linearity* of λ-terms, which sometimes is a strong requirement. In particular, the linearity condition does not allow one to encode Montague semantics because terms encoding semantic interpretations are not linear. For instance, a term such as $\lambda x.\mathbf{man}(x) \wedge \mathbf{walk}(x)$ is not linear (x has two occurrences bound with the same λ-abstraction). To overcome the problem, Kanazawa (2007) introduces *almost-linear* λ-terms. With almost-linear λ-terms, one is able encode Montague semantics. At the same time, the parsing and generation problems of second-order ACGs where object terms are almost-linear are polynomial (Kanazawa, 2007).

Definition 3.7.1 (Kanazawa, 2007).
Let $\Sigma = \langle A, C, \vec{\tau} \rangle$ be a higher-order signature. One defines a set of almost-linear terms over Σ, denoted by $\vec{\Lambda}(\Sigma)$, and the type of a term $t \in \vec{\Lambda}(\Sigma)$, denoted by $\vec{\tau}(t)$. The set $\vec{\Lambda}(\Sigma)$ is the smallest set satisfying the following properties:

1. *If $t, u \in \vec{\Lambda}(\Sigma)$, $\vec{\tau}(t) = \alpha \to \beta$, $\vec{\tau}(u) = \alpha$, and if for any $x \in \mathsf{FV}(t) \cap \mathsf{FV}(u)$, $\vec{\tau}(x)$ is atomic, then $(t\ u) \in \vec{\Lambda}(\Sigma)$ and $\vec{\tau}(t\ u) = \beta$.*
2. *If $t \in \vec{\Lambda}(\Sigma)$, $\vec{\tau}(t) = \beta$, and $x \in \mathsf{FV}(t)$, where $\vec{\tau}(x) = \alpha$ and α is atomic, then $\lambda x.t \in \vec{\Lambda}(\Sigma)$ and $\vec{\tau}(\lambda x.t) = \alpha \to \beta$.*
3. *$\vec{\Lambda}(\Sigma)$ is closed under $\beta\eta$-equivalence.*

One obtains the definition of linearity by strengthening the requirement of the clause 2 of Definition 3.7.1 as follows: $\mathsf{FV}(t) \cap \mathsf{FV}(u) = \emptyset$. Thus, any linear term is also almost-linear. Although almost linear λ-terms generalize λ°-terms, we still would like

[25] We borrow Table 3.3 from (Yoshinaka, 2006).

to emphasize the fact that a given term is linear, or a given λ-abstraction is linear. In other words, if $x : \alpha$ is a variable with a single occurrence in $t : \beta \in \vec{\Lambda}(\Sigma)$, we write $\lambda^{o}x.t : \alpha \multimap \beta \in \vec{\Lambda}(\Sigma)$.

Definition 3.7.2 (λ-CFG (Kanazawa, 2007)).
A context-free λ-grammar (λ-CFG) G is a quadruple $\langle \Sigma_1, \Sigma_2, \mathscr{L}, s \rangle$, where

- $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ *is a linear higher-order signature and* $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ *is a higher-order signature, called the abstract and the object vocabularies of G, respectively.* Σ_1 *is second-order.*
- $\mathscr{L}$ *is a lexicon between* Σ_1 *and* Σ_2*, which is a homomorphism between* Σ_1 *and* Σ_2 *defined in Definition 3.3.1 with the difference that (a) a constant of* Σ_1 *may translate to an almost-linear λ-term over* Σ_2*; (b) an abstract atomic type translates to an almost-linear type.*
- $s \in A_1$ *is the distinguished type of the λ-CFG.*

Kanazawa (2007) named almost-linear second-order ACGs as *λ-CFGs* due to the fact that they can be seen as CFGs that rewrite λ-terms instead of strings. Indeed, the abstract language of any second-order ACG can be viewed as a regular tree language. The yield of a regular tree language is context-free language.

Below, while discussing semantic encodings within second-order ACGs, that is, the signatures with the help of which one encodes semantic interpretations, we will have in mind second-order almost-linear ACGs, i.e., λ-CFGs, unless otherwise stated; in rest of cases, we stick to de Groote's (2001) linear version of ACGs. We will write $\Lambda(\Sigma)$ for denoting both the sets of almost-linear λ-terms and λ^{o}-terms over Σ, unless otherwise stated. With HOS, we abbreviate both a *linear higher-order signature* and a *higher-order signature* whenever it does not cause a confusion.

3.8 TAG with Montague Semantics as ACGs

As we already discussed in Section 3.5, to model a TAG as an ACG, one represents TAG derivation trees as abstract terms and TAG derived trees as object ones.

To model the syntax-semantic interface using TAGs, Pogodalla (2004) proposes to compute the semantic representations from TAG derivation trees by interpreting them as underspecified semantic formulas. (Pogodalla, 2009) presents another version of this encoding, where one defines fully specified semantic representations, instead of underspecified ones. In both of these encodings, one makes use of higher-order interpretations reminiscent of ones provided in (Montague, 1973). Due to this, one refers to these encodings as the ACG encoding of TAG with Montague semantics. We follow (Pogodalla, 2009) while discussing the encoding of TAG with Montague semantics as ACGs.

3.8.1 Montague Semantics as Object Terms

We interpret TAG derivation trees as Montague's (1973) logical semantic formulas. Thus, we consider an ACG whose abstract terms encode TAG derivation trees, whereas

the object ones encode semantic interpretations. To encode Montague semantics, we introduce a signature Σ_{Log}. We interpret constants and types of the vocabulary $\Sigma_{\text{TAG}}^{\text{Der}}$ as terms and types over Σ_{Log}. The signature Σ_{Log} contains two types, t for propositions and e for individuals. It has constants encoding n-place predicates, logical connectives and quantifiers, as it is shown in Table 3.4.

fred, **john**, **mary**, **he**	$: e$	**because**	$: t \multimap t \multimap t$
exam, **important**, **grumpy**	$: e \multimap t$	**fail**, **love**	$: e \multimap e \multimap t$
really	$: t \multimap t$	**seem**	$: (e \multimap t) \multimap e \multimap t$
$\wedge$	$: t \multimap t \multimap t$	$\vee$	$: t \multimap t \multimap t$
$\Rightarrow$	$: t \multimap t \multimap t$	$\neg$	$: t \multimap t$
$\exists$	$: (e \to t) \multimap t$	$\forall$	$: (e \to t) \multimap t$

Table 3.4: Constants in the semantic vocabulary Σ_{Log}

3.8.2 Interpretations as Montague Semantics

We interpret a type np as a type $(e \multimap t) \multimap t$, which is the linear version of Montague's (1973) higher-order interpretation of noun phrases. The interpretation of the distinguished type S is t. As Table 3.5 shows, one interprets S_A as $t \multimap t$, since the terms of type S_A model S-modifiers, i.e., the ones that modify clauses (terms of type t). V_A translates to $(e \multimap t) \multimap e \multimap t$ as a term of type V_A models a modifier of a verb phrase (i.e., a term of type $e \multimap t$). Hence, by translating V_A as $(e \multimap t) \multimap e \multimap t$, one also makes a semantic difference between the types V_A and S_A. Nevertheless, to interpret both V_A and S_A as $t \multimap t$ is also possible.

Types in $\Sigma_{\text{TAG}}^{\text{Der}}$	Their semantic interpretations
np	$(e \multimap t) \multimap t$
S	t
S_A	$t \multimap t$
V_A	$(e \multimap t) \multimap (e \multimap t)$
n_A	$(e \to t) \multimap (e \to t)$
n_d	$(e \to t) \multimap (e \to t) \multimap t$

Table 3.5: The semantic interpretations of abstract types

Elementary Trees Anchored with Nouns, Adjectives, and Determiners

The interpretations of constants modeling initial and auxiliary trees that we use in order to encode noun phrases, that is, trees anchored with articles, quantifiers, and adjectives, are shown in Table 3.6. Table 3.5 provides interpretations of atomic types involved in the types of these constants.

Notice that in Table 3.6, the constants $\forall$ and $\exists$ used in the encodings of quantifier words (e.g. *each*, *every*) and articles (e.g. *an*) are of type $(e \to t) \multimap t$, which is not

Constants of $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$	Their interpretations by $\mathscr{L}^{\mathsf{sem}}_{\mathsf{TAG}}$
$\mathrm{C}_{\textbf{exam}} : \mathsf{n}_d \multimap \mathsf{n}_\mathsf{A} \multimap \mathsf{np}$	$\lambda^{\mathrm{o}}\ det\ adj\ .\ det\ (adj\ (\lambda x.\textbf{exam}\ x))$
$\mathrm{C}_{\textbf{important}} : \mathsf{n}_\mathsf{A} \multimap \mathsf{n}_\mathsf{A}$	$\lambda^{\mathrm{o}}\ adj\ n\ .\ adj\ (\lambda\, x.\, (\textbf{important}\ x) \wedge (n\ x))$
$\mathrm{C}_{\textbf{every}}, \mathrm{C}_{\textbf{each}} : \mathsf{n}_d$	$\lambda^{\mathrm{o}}\ P\ Q\ .\ \forall\, x.\, (P\, x)\ \Rightarrow\ (Q\, x)$
$\mathrm{C}_{\textbf{a}}, \mathrm{C}_{\textbf{an}} : \mathsf{n}_d$	$\lambda^{\mathrm{o}}\ P\ Q\ .\ \exists\, x.\, (P\, x)\ \wedge\ (Q\, x)$

Table 3.6: Semantic interpretations of constants

a linear type. This is due to the fact that the same variable x (of type e) has two occurrences in the same sub-term. Therefore, one has a non-linear abstraction over that variable. Hence, we interpret the type n_d with the help of a (non-linear) implicative type (see Table 3.5 on the previous page). Nevertheless, our semantic terms are almost-linear. Hence, in this case, Kanazawa's (2007) results apply, which guarantee that the parsing and generation problems are of polynomial complexity. Moreover, we could interpret np as $(e \to t) \multimap t$, which would make possible to use a variable corresponding to an individual (that is, a variable of type e) within a sub-term more than once. Still, we would obtain an almost-linear second-order ACG (λ-cfg). We translate the constants modeling empty adjunctions $\mathrm{I}_{\mathsf{S}_\mathsf{A}}$, $\mathrm{I}_{\mathsf{V}_\mathsf{A}}$, $\mathrm{I}_{\mathsf{n}_\mathsf{A}}$ to $\lambda^{\mathrm{o}}\, x.\, x$.

Remark 3.3. *In Montague Grammar (Montague, 1973), one translates neither articles nor quantifier words directly, but rather gives the recipes for translating noun phrases such as* 'every CN', 'a CN', *etc. In contrast with Montague Grammar, in the ACG encoding of TAG with Montague semantics, one provides interpretations of articles, quantifiers, plural markers, and common nouns separately. Out of these interpretations, one produces the interpretation of a noun phrase as the* composition *of the interpretations of an article/quantifier/plural marker, of adjectives, and of a CN.*

Elementary Trees Anchored with Predicative Adjectives, Verbs, Adverbs, and Copulas

To interpret the abstract constants modeling initial and auxiliary trees anchored with verbs, we interpret S-adjunctions and VP-adjunctions with the help of higher-order predicates. For instance, Table 3.7 shows the interpretation of a constant modeling an initial tree anchored by a transitive verb *failed*. The interpretations of adjunctions on S and VP nodes, which are denoted with the variables s_a and v_a, scope over $(\textbf{fail}\, x\, y)$ as they *modify* the content expressed by the predicate **fail** and its arguments. The type of **fail** is $e \multimap e \multimap t$. We interpret the substitution sites of the initial tree anchored with *failed* using the higher-order interpretations of noun phrases. In this way, one can encode the predicate-argument relations expressed by a verb anchoring an initial tree.

A predicative adjective plays the role of a predicate. In TAG elementary trees, predicative adjectives anchor initial trees. Thus, we interpret an initial tree anchored by a predicative adjective similar to what we do in the case of verbs (see Table 3.7).

We interpret elementary trees according to the semantic properties of their anchors. To illustrate this, let us consider auxiliary trees anchored with adverbs (e.g. *really*) and ones anchored with raising verbs (such as *seems*). Although these trees are VP-auxiliary

Constants of $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$	Their interpretations by $\mathscr{L}^{\mathsf{sem}}_{\mathsf{TAG}}$
$\mathrm{C}_{\textbf{grumpy}}$	$\lambda^{\mathrm{o}}\, s_a\, v_a\;\; subje.\; s_a\,(subje\,(v_a\,(\lambda^{\mathrm{o}} x.\,((\textbf{grumpy}\, x))))) :$ $(t \multimap t) \multimap ((e \multimap t) \multimap e \multimap t) \multimap ((e \multimap t) \multimap t) \multimap t$
$\mathrm{C}_{\textbf{failed}}$	$\lambda^{\mathrm{o}}\, s_a\, v_a\;\; subje\;\; obj.\; s_a\,(subje\,(\lambda^{\mathrm{o}} x.\,(v_a\,(obj\,(\lambda^{\mathrm{o}} y.\;\textbf{fail}\, xy))))) :$ $(t \multimap t) \multimap ((e \multimap t) \multimap e \multimap t) \multimap ((e \multimap t) \multimap e \multimap t) \multimap$ $\multimap ((e \multimap t) \multimap t) \multimap t$
$\mathrm{C}_{\textbf{to love}}$	$\lambda^{\mathrm{o}}\, s_{a1}\, s_{a2}\, v_a\;\; obj\;\; subje.\; s_{a1}\,(s_{a2}\,(subje\,(v_a\,(\lambda^{\mathrm{o}} x.\,(obj\,(\lambda^{\mathrm{o}} y.\;\textbf{love}\, xy)))))))) :$ $(t \multimap t) \multimap (t \multimap t) \multimap ((e \multimap t) \multimap e \multimap t) \multimap$ $\multimap ((e \multimap t) \multimap t) \multimap ((e \multimap t) \multimap t) \multimap t$
$\mathrm{C}_{\textbf{seems}}$	$\lambda^{\mathrm{o}}\, vp_a\, r.\; vp_a\;(\lambda^{\mathrm{o}}\, x.\textbf{seem}\; r\; x) :$ $((e \multimap t) \multimap e \multimap t) \multimap (e \multimap t) \multimap (e \multimap t)$
$\mathrm{C}_{\textbf{really}}$	$\lambda^{\mathrm{o}}\, vp_a\, r.\; vp_a\;(\lambda^{\mathrm{o}} x.\;\textbf{really}\;(rx)) :$ $((e \multimap t) \multimap e \multimap t) \multimap (e \multimap t) \multimap (e \multimap t)$
C_{is}	$\lambda^{\mathrm{o}}\, vp_a\, r.\; vp_a\;(\lambda^{\mathrm{o}} x.\;(rx)) :$ $((e \multimap t) \multimap e \multimap t) \multimap (e \multimap t) \multimap (e \multimap t)$

Table 3.7: Semantic interpretations of elementary trees anchored with predicative adjectives, verbs, adverbs, and copulas

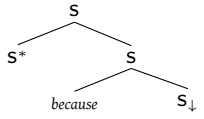
trees, we interpret them differently as they have contrasting semantic properties. For example, Table 3.7 provides interpretations of the constants encoding auxiliary trees anchored with *seems* and *really*. To illustrate that their semantic properties are different, we provide the interpretations of the sentences *John Mary seems to love* and *Mary really loves John* in Equation (3.13) and Equation (3.14), respectively.

$$i_{seems} = \textbf{seem}\;(\lambda^{\mathrm{o}}\, x.\;\textbf{love}\; x\;\textbf{john})\;\textbf{mary} \quad : t \tag{3.13}$$

$$i_{really} = \textbf{really}\;(\textbf{love mary john}) \quad : t \tag{3.14}$$

The difference between i_{seems} and i_{really} reflects that one models **seems** as a two-place (higher-order) predicate, whereas **really** is a one-place predicate.

As Table 3.7 shows, we interpret the constant $\mathrm{C}_{\textbf{is}} : \mathsf{V}_{\mathsf{A}}$ modeling the VP-auxiliary tree anchored with the copula *is* as a term without any constant, i.e., without any semantic material.

Furthermore, let us interpret $\mathrm{C}_{\textbf{because}} : \mathsf{S}_{\mathsf{A}} \multimap \mathsf{S} \multimap \mathsf{S}_{\mathsf{A}}$, which models an S-auxiliary tree 

. Let **because** denote the predicate signaled by *because*. The predicate **because** relates two clauses. In other words, **because** is a 2-place predicate whose arguments are propositions.

The predicate **because** receives one of the arguments by substituting an S-rooted (completed) derived tree in the S-substitution site of the auxiliary tree anchored with *because*. By adjoining the auxiliary anchored with *because* into an S-rooted derived tree, one provides the predicate **because** with the other argument. To model the way **because** obtains its arguments, we interpret $\mathrm{C}_{\textbf{because}}$ as it is shown in Table 3.8.

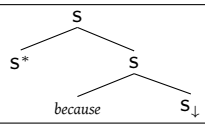
Constants of $\Sigma^{\text{Der}}_{\text{TAG}}$	LTAG trees	Their interpretations under $\mathscr{L}^{\text{sem}}_{\text{TAG}}$
$\mathrm{C}_{\textbf{because}} : \mathsf{S}_\mathsf{A} \multimap \mathsf{S} \multimap \mathsf{S}_\mathsf{A}$		$\lambda^{\text{o}}\, s_a\, s\, x.\, s_a\, (\textbf{because}\, s\, x) \;:\; (t \multimap t) \multimap t \multimap t$

Table 3.8: The semantic interpretation of the LTAG tree anchored with *because*

Example 3.6. [26]

((12)(d), repeated) John Mary seems to love.

Figure 3.20(c) on page 76 shows the derived tree whose yield is the sentence (12)(d). The derivation tree of this derived tree is shown in Figure 3.20(b). As one may notice, the derivation tree does not encode the correct semantic dependencies. Indeed, $\alpha_{\text{to-love}}$ dominates (scopes over) β_{seems}. However, from a semantic standpoint, *seems* scopes over *to love*, as we already saw in i_{seems} (see Equation (3.13)). We model this derivation tree by the term t_4 given in Table 3.17 on page 75. To obtain the semantic interpretation of the sentence (12)(d), we interpret the term t_4 by the lexicon $\mathscr{L}^{\text{sem}}_{\text{TAG}}$ as follows:

$$\mathscr{L}^{\text{sem}}_{\text{TAG}}(t_4) = \mathscr{L}^{\text{sem}}_{\text{TAG}}(\mathrm{C}_{\text{to love}}\ \mathrm{I}_{\mathsf{S}_\mathsf{A}}\ \mathrm{I}_{\mathsf{S}_\mathsf{A}}\ (\mathrm{C}_{\text{seems}}\ \mathrm{I}_{\mathsf{V}_\mathsf{A}})\ \mathrm{C}_{\text{John}}\ \mathrm{C}_{\text{Mary}}) \twoheadrightarrow_\beta$$
$$\twoheadrightarrow_\beta \textbf{seem}\ ((\lambda^{\text{o}}\, x.\ \textbf{love john}\ x)\ \textbf{mary}) \;:\; t \quad (3.15)$$

Thus, we obtain the term **seem** $(\lambda^{\text{o}}\, x.\ \textbf{love john}\ x)$ **mary** as the semantic interpretation of t_4. In contrast to the derivation tree, the obtained semantic interpretation encodes the correct semantic dependencies (**love** is indeed under the scope of **seem**).

Example 3.7.

((12)(c), repeated) Fred is grumpy because he failed an exam.

In the case of the sentence (12)(c), we model the derivation tree (in Figure 3.18(b) on page 76) with a term t_3 defined in Table 3.17 on page 75. We compute the image of t_3 under the lexicon $\mathscr{L}^{\text{sem}}_{\text{TAG}}$ as follows:

$$\mathscr{L}^{\text{sem}}_{\text{TAG}}(t_3) = \mathscr{L}^{\text{sem}}_{\text{TAG}}(\mathrm{C}_{\textbf{grumpy}}(\mathrm{C}_{\textbf{because}}\ \mathrm{I}_{\mathsf{S}_\mathsf{A}}\ (\mathrm{C}_{\textbf{failed}}\ \mathrm{I}_{\mathsf{S}_\mathsf{A}}\ \mathrm{I}_{\mathsf{V}_\mathsf{A}}\ \mathrm{C}_{\textbf{he}}\ (\mathrm{C}_{\textbf{exam}}\ \mathrm{I}_{\mathsf{n}_\mathsf{A}}\ \mathrm{C}_{\textbf{an}})))\ (\mathrm{C}_{\textbf{is}}\ \mathrm{I}_{\mathsf{V}_\mathsf{A}})\ \mathrm{C}_{\textbf{Fred}}) \twoheadrightarrow_\beta$$
$$\twoheadrightarrow_\beta \textbf{Because}\ (\exists\, x\ (\textbf{exam}\ x) \wedge (\textbf{fail Fred}\ x))\ (\textbf{grumpy Fred}) \;:\; t \quad (3.16)$$

Figure 3.21 on the facing page illustrates the overall architecture of the ACG encoding of TAG with Montague semantics.

[26] In Appendix A, we provide ACG codes for the examples provided below, which one can run on the ACG toolkit.

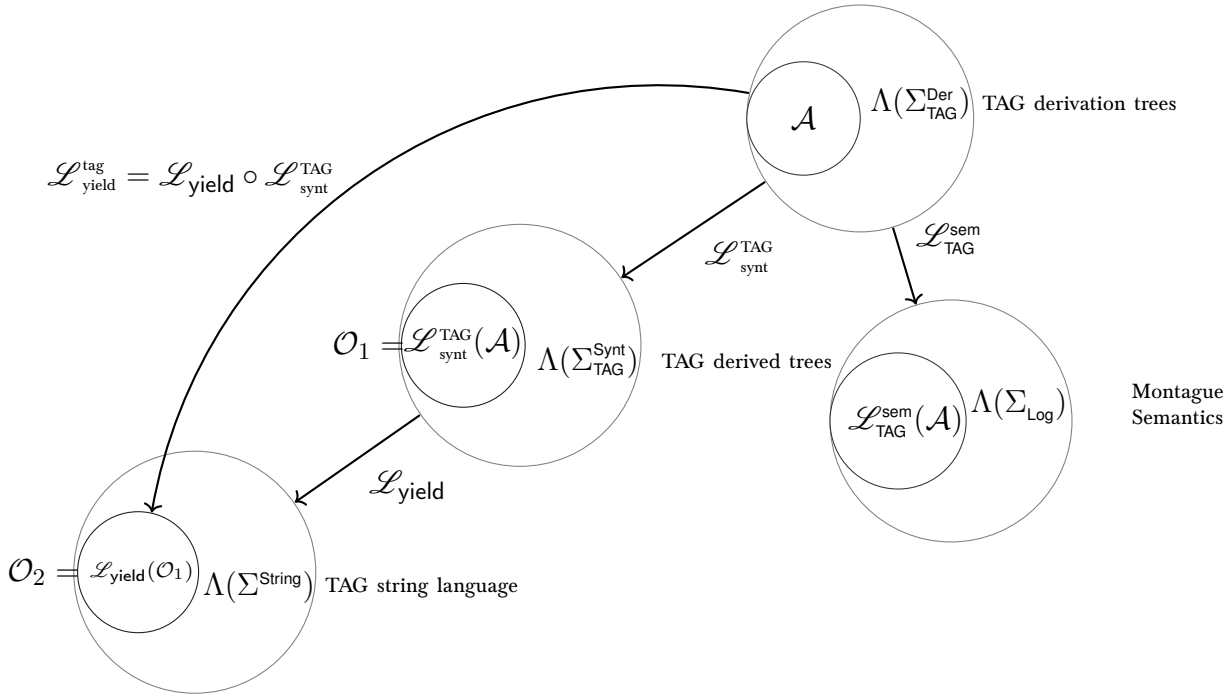


Figure 3.21: The ACG architecture of TAG with Montague semantics

Remark 3.4. *The ACG encoding of TAG has the following property: A lexicon interprets a linear abstraction always as a linear one. This implies that if a term over Σ^{Der}_{TAG} is of type $\alpha \multimap \beta$ and $\mathscr{L}^{sem}_{TAG}$ is a lexicon, then $\mathscr{L}^{sem}_{TAG}(\alpha \multimap \beta) : \mathscr{L}^{sem}_{TAG}(\alpha) \multimap \mathscr{L}^{sem}_{TAG}(\beta)$. On other other hand, as we already saw, in Kanazawa's (2007) λ-CFGs, one may interpret $\alpha \multimap \beta$ as $\mathscr{L}^{sem}_{TAG}(\alpha) \to \mathscr{L}^{sem}_{TAG}(\beta)$. Hence, Pogodalla (2009) uses a version of ACGs that falls between de Groote's (2001) linear and Kanazawa's (2007) almost-linear versions of the second-order ACGs. Thus, the problems of parsing and generation with the ACG encoding of TAG with Montague semantics are polynomial.*

Chapter 4

Discourse Theories

Contents

In this chapter, we discuss discourse theories. We focus on two paradigms in discourse studies, known as discourse structure theories and dynamic semantic theories. Discourse structure theories view a coherent discourse as a structured entity constructed by linking the sub-entities to each other. Usually, a sub-entity in the discourse structure is identified with a meaningful sub-piece (sub-part) of the original discourse. A meaningful sub-part of a discourse is called a discourse constituent, or a discourse unit. Special relations, called rhetorical (discourse) relations, provide connections between discourse units. In order to interpret a discourse, discourse structure theories largely rely on pragmatic knowledge. Unlike discourse structure theories, most dynamic theories do not pay significant attention to the discourse structure. They make use of the notions of a dynamic meaning of a proposition and a context in order to interpret a discourse. The dynamic meaning of a proposition is its potential to change (update) the context where it occurs. From discourse structure theories, we discuss Rhetorical Structure Theory (RST), which has been applied in a number of tasks in computational linguistics. From dynamic semantic theories, we focus on SDRT. While SDRT is a dynamic semantic theory, it incorporates the notions of a rhetorical relation

and a discourse structure in its dynamic setting. Thus, SDRT can be seen as a bridge between dynamic semantic theories and discourse structure theories. In addition, we provide a brief overview of linguistic aspects of discourse connectives. In particular, we highlight some problems of identifying the arguments of a discourse connective.

4.1 Linguistic Aspects of Discourse Connectives

A discourse connective relates two pieces of semantic content, called *arguments* of a discourse connective. A piece of discourse that gives rise to an argument of a discourse connective is referred to as *discourse unit.* We say that a discourse connective signals a *discourse (rhetorical) relation.* Usually, a discourse relation is a binary relation, or to put it another way, a discourse relation has two arguments.[27]

In this section, we mainly focus on overt (explicit) lexical markers of discourse connectives. They serve as basic linguistic means for expressing ideas in a coherent way. One refers to an explicit lexical marker of a discourse connective as a *cue word* or a *cue phrase*, depending on whether it consists of one or more lexical items (words). Besides overt lexical markers, discourse connectives can also be expressed by other means, such as textual adjacency. Since this section is concerned with the explicit lexical markers of discourse connectives, we may refer to them as discourse connectives if it does not cause a confusion.

In natural languages like English and French (the ones that we focus on within this thesis), three main classes of discourse connectives are distinguished. These classes consist of *subordinate conjunctions*, *coordinate conjunctions*, and *adverbial connectives.* For instance, because and although in Example (17) are subordinate conjunctions, whereas but and and in Example (18) are coordinate ones. We may use the term *conjunction* to refer both subordinate and coordinate conjunctions. In Examples (17) and (18), the conjunctions signal the discourse relations connecting two pieces of semantic content, denoted in bold and in italics.[28]

Adverbial connectives (discourse adverbials), like conjunctions, give rise to discourse relations that have two arguments. For instance, moreover in (19)(a) and then (19)(b) are adverbial connectives. In Example (19), each of these two discourse adverbials signals a discourse relation that relates the content of the first sentence (in italics) with the content of the second one (in bold).

(17) a. *Fred is grumpy* because **he lost his keys**.
b. Although **Fred is generous**, *he is hard to find.*

(18) a. *Fred is French* but **his wife is Spanish**.
b. *Fred is French* and **his wife is Spanish**.

[27]Discourse relations vary from theory to theory. A more or less agreed assumption about a discourse relation is that it has *two* arguments.

[28]Following the style of (Rashmi Prasad et al., 2008; Bonnie Webber and Rashmi Prasad, 2009).

(19) a. *Fred lost his keys.* Moreover, **he failed an exam**.

b. *Fred went to the cinema.* Then, **he went to the bar**.

To analyze a discourse, it is necessary to identify the *content* related by the discourse relations signaled by the discourse connectives in the discourse. In other words, interpreting a discourse incorporates finding the arguments of discourse relations. For now, we will call them arguments of discourse connectives. While in the examples presented so far, one straightforwardly identifies the arguments of a discourse connective, to define what are the arguments of a discourse connective is a problem in general.

> Arguments of explicit connectives are not constrained to be single clauses or single sentences: They can be associated with multiple clauses or sentences. However, a minimality principle requires an argument to contain the minimal amount of information needed to complete the interpretation of the relation.
>
> Rashmi Prasad et al. (2008)

Thus, even though a text may contain various kinds of information, only certain kinds of information can qualify as arguments of a discourse connective. For instance, let us consider the following example from (Miltsakaki, A. Joshi, R. Prasad, and B. Webber, 2004):

(20) Workers described "clouds of dust" *that hung over parts of the factory* even though **exhaust fans ventilated the air**.

In (20), even though has an argument which is a relative clause (denoted in italics), but not the entire clause – workers described "clouds of dust" *that hung over parts of the factory*.

One of the main problems of identifying arguments of a discourse connective is the mismatch between its syntactic and discourse-level (semantic) properties. In other words, a discourse connective may exhibit different behaviors at the sentence-level and at the discourse-level. It has been argued that the semantic arguments of a conjunction are obtained from *locally* available material (Bonnie Webber, Knott, Stone, and Aravind Joshi, 1999). A stronger assumption about conjunctions is that the semantic arguments of a conjunction are defined in part by syntax.[29] For instance, in (17)(b), although has two arguments, the clauses written in italics and in bold. According to the assumption about conjunctions, these clauses are *syntactically bound* arguments to although. One says that a conjunction obtains its arguments *structurally*, or that both of the arguments of a conjunction are *structural* because the arguments of a conjunction appear in *the parse tree of the discourse* (B. L. Webber, 2004; B. L. Webber and A. K. Joshi, 1998; Bonnie Webber, Stone, Aravind Joshi, and Knott, 2003). Sometimes, we may refer to conjunctions as *structural connectives*.

While for conjunctions one assumes that some syntactic rules govern the way they obtain their arguments, some authors argue that for *discourse adverbials* one cannot make such an assumption (B. L. Webber, 2004; B. L. Webber and A. K. Joshi, 1998;

[29]Similar to arguments of a verb, which are defined by syntax (e.g. the subcategorizing frame, or the domain of locality in LTAG).

Bonnie Webber, Stone, Aravind Joshi, and Knott, 2003). They argue that only one of two arguments is syntactically bound to an adverbial connective, that is, it has only one structural argument. For example, in (19)(a), the syntactically bound argument of moreover is **he failed an exam**. The other argument of an adverbial connective is called *anaphoric*. While the structural argument appears in the parse tree of a discourse, the anaphoric one does not. To put it another way, an adverbial connective may have only one argument defined by syntax. The other argument is *anaphoric* (inferred) since it is not defined by syntax but has to be either anaphorically retrieved in the discourse or inferred from the context (B. L. Webber, 2004; Bonnie Webber, Stone, Aravind Joshi, and Knott, 2003). Various researchers develop their approaches with contrasting views on the structural and anaphoric arguments. For instance, Danlos (2009) develops an approach where both arguments of an adverbial connective appear in a parse tree of a discourse, and thus there is no distinction between structural and anaphoric arguments, or in the terminology of (B. L. Webber, 2004; Bonnie Webber, Stone, Aravind Joshi, and Knott, 2003), both arguments are structural. Although how arguments of discourse connectives are provided is a subject of discussions, most researchers agree that every discourse connective has two arguments. Let us denote the arguments of a discourse connective (relation) with ARG1 and ARG2. In this section, ARG1 is in italics and ARG2 in bold; a discourse connective is underlined.

4.1.1 Arg1

As we already mentioned, according to some theories, ARG1 is the *anaphoric* argument of an adverbial connective, whereas ARG2 is *structural* (B. L. Webber, 2004; B. L. Webber and A. K. Joshi, 1998; Bonnie Webber, Stone, Aravind Joshi, and Knott, 2003). If the anaphoric argument of a discourse adverbial is explicitly present in a text, it can be retrieved in the discourse using some sort of mechanism, similar to anaphora resolution. Otherwise, ARG1 can be *inferred*. The discourse adverbial obtains the structural argument, ARG2, by means of syntax. In other words, the semantic content denoted by ARG2 is obtained by interpreting a piece of discourse that is syntactically bound to the adverbial connective, whereas the semantic content denoted by ARG1 may not be an interpretation of any discourse unit in a text.

While ARG1 is a content that is accessible to adverbial connectives, it may not be accessible to structural connectives, i.e., conjunctions. The capability of accessing the inferred (abstract) material in a discourse is considered to be one of the main characteristics that makes adverbials different from conjunctions. For the sake of illustration, let us consider the coordinate conjunction or and the adverbial otherwise. In a number of cases, one may use or and otherwise interchangeably, as it in the following discourses:

(21) a. If the light is red, *stop*. Otherwise **you'll get a ticket**.

b. If the light is red, *stop*, or **you'll get a ticket**.

(21)(a) and (21)(b) express the same meaning: If you do something else than stop, you'll get a ticket. However, in the examples such as (22), otherwise and or behave

differently. To illustrate that, let us replace otherwise by or in (22). One obtains the discourse (23), whose meaning differs from the meaning of (22). The difference in their meanings is due to the fact that ARG1 of otherwise is inferred in (22). In particular, in (22), otherwise has an access to *the interpretation of the condition*, whereas in (23), or does not have an access to that (Bonnie Webber, Stone, Aravind Joshi, and Knott, 2003).

(22) *If the light is red, stop.* Otherwise **go straight on**.

(23) If the light is red, *stop*, or **go straight on**.

According to recent studies strengthened with the evidence from corpus analyses (Bonnie Webber and Rashmi Prasad, 2009), sentence-initial (S-initial) coordinate conjunctions show some similarities to adverbial connectives. In particular, ARG1 of an S-initial coordinate conjunction might be a non-adjacent textual unit to the S-initial conjunction. For example, in the discourse (24), the S-initial coordinate conjunction but has ARG1 at the distance of one sentence from the location of the S-initial conjunction.

(24) *I'm not suggesting that the producers start putting together episodes about topics like the Catholic-Jewish dispute over the Carmelite convent at Auschwitz.* That issue, like racial tensions in New York City, will have to cool down, not heat up, before it can simmer. But **I am suggesting that they stop requiring Mr. Mason to interrupt his classic shtik with some line about "caring for other people" that would sound shmaltzy on the lips of Miss America**.

Thus, the new evidence suggests that the difference between structural and anaphoric discourse connectives may not lay into the locality or non-locality of arguments, contrary to what was claimed before. At the same time, it is noteworthy that in lines with the previous studies, the recent studies indicate that adverbials have access to the content that is inaccessible for conjunctions. Based on the corpus studies, (Bonnie Webber and Rashmi Prasad, 2009) claims that everything that can be an argument of an S-initial coordination conjunction can also serve as an argument of a discourse adverbial, whereas the reverse statement does not hold.

4.1.2 Arg2

Since ARG2 of an explicit marker of a discourse connective is defined in part by syntax, to identify ARG2 is considered to be relatively easier than to identify ARG1. Indeed, in certain cases, the syntactic argument of an adverbial connective coincides with the discourse unit that gives rise to ARG2, as it is in the following example from (Danlos, 2013):

(25) *Fred* *ira* *à* *Dax* *pour* *Noël.* Ensuite, **il** **ira** **à** **Pau**.
Fred go$_{\text{FUT;3P;SNG.}}$ to Dax for Christmas. Afterwards, he go$_{\text{FUT;3P;SNG.}}$ to Pau.

'Fred will go to Dax for Christmas. Afterwards, he will go to Pau.'

However, in certain cases, it is not clear what serves as ARG2 of a discourse connective.

4.1.2.1 Attitude Verbs

In Example (25), the adverbial ensuite (then, afterwards) gives rise to a temporal relation between the events of Fred going to Dax and Fred going to Pau. Thus, in this case, ARG2 is the content of the clause that serves as the syntactic argument to the adverbial connective (the piece of text in bold). In some cases, however, identifying ARG2 may also depend on various factors that make it a rather difficult task. For instance, identifying ARG2 becomes more problematic if one employs *attitude* verbs. Since one uses attitude verbs to express beliefs/assertions of the agents, this may complicate the task of determining the content that discourse connectives relate (Bernard, 2015; Danlos, 2013; Dines et al., 2005; Bonnie Webber, Egg, and Kordoni, 2012). To illustrate that, we consider the second clause in (25), [**il ira à Pau**]. By using this clause and the verb *croire* (believe), we can produce the discourses (26)(a) and (26)(b).[30] In both of the cases, ensuite (then) signals a temporal relation between [*Fred ira à Dax pour Noël*] and [**il ira à Pau**]. Thus, in the case of discourses (25), (26)(a), and (26)(b), ARG2 of ensuite is the same. With the same kind of reasoning, we modify (26)(a) by using the first sentence of (26)(a) with the verb *croire* (believe). We obtain the discourse (27). In this case, ARG2 is not the same as in the cases of (26)(a). In (27), ensuite establishes the relation between [*Jane a cru que Fred irait à Dax pour Noël*] and [**elle a cru qu'il irait à Pau**] (Jane thought that Fred would go to Dax and then she thought that he would go to Pau). Thus, if attitude verbs are involved in a discourse, to identify the arguments of a discourse adverbial becomes a non-trivial task, even in a case of a discourse with two sentences.

(26)

a. *Fred* *ira* *à* *Dax* *pour* *Noël.* Ensuite, Jane croit qu'**il** **ira** **à** **Pau**.
Fred *go*$_{\text{FUT;3P;SNG.}}$ *to* *Dax* *for* *Christmas.* Afterwards, Jane think$_{\text{PRS;3P;SNG.}}$ that he go$_{\text{FUT;3P;SNG.}}$ to Pau.

'Fred will go to Dax for Christmas. Afterwards, Jane thinks that he will go to Pau.'

[30]The boxed texts denote the parts of the text that contribute neither to arguments of discourse connectives nor to cue phrases.

b. *Fred* *ira* *à* *Dax* *pour* *Noël.* Ensuite, croit Jane,
Fred go$_{\text{FUT;3P;SNG.}}$ to Dax for Christmas. Afterward, think$_{\text{PRS;3P;SNG.}}$ Jane,
il **ira** **à** **Pau**.
he go$_{\text{FUT;3P;SNG}}$ to Pau.

'Fred will go to Dax for Christmas. Afterwards, Jane thinks, he will go to Pau.'

(27) *Jane* *a* *cru* *que* *Fred* *irait* *à* *Dax* *pour*
Jane have$_{\text{PRES.3P;SNG.}}$ think$_{\text{PAST.PART.}}$ that Fred go$_{\text{PRES.COND.3P;SNG.}}$ to Dax for
Noël. Ensuite, **elle** **a** **cru** **qu'il** **irait**
Christmas. Afterwards, she have$_{\text{PRES.3P;SNG.}}$ think$_{\text{PAST.PART.}}$ that he go$_{\text{PRES.COND.3P;SNG.}}$
à **Pau**.
to Pau.

'Jane thought that Fred would go to Dax for Christmas. Afterwards, she thought that he would go to Pau.'

In a discourse where attitude verbs are involved, to determine ARG2 becomes problematic not only for adverbial connectives, but even for subordinate conjunctions, which are considered as connectives whose arguments are defined by syntax. Let us consider the following examples from (Danlos, 2012):

(28)

a. *Fred* *est* *allé* *travailler* bien que Jane dise qu'**il**
Fred is go$_{\text{PAST.PART.}}$ work$_{\text{INDEF.}}$ although Jane say$_{\text{SUBJ.PRES;3P;SNG.}}$ that he
est **très** **malade**.
is very ill.

'Fred went to work even though Jane says that he is very ill. '

b. ∗Fred est fatigué parce que Jane dit qu'il a
Fred is tired because Jane says$_{\text{PRES;3P;SNG.}}$ that he have$_{\text{PRES.3P;SNG.}}$
mal dormi.
bad sleep$_{\text{PAST.PART.}}$.

'Fred is tired because Mary says that he slept poorly.'

While the discourse (28)(a) is felicitous, the discourse of the similar syntactic structure (28)(b) is infelicitous. Thus, a purely syntactic approach cannot determine whether a given discourse is felicitous, and if it is felicitous, then what is (are) its interpretation(s). Although Danlos (2013) proposes some principles/rules how to extract ARG2, they apply in a limited number of cases where one imposes certain requirements on both discourse connectives and syntactico-semantic properties of sentences.

4.1.2.2 Clause-medial Adverbials

In a case where an adverbial connective appears at an internal position in a clause, one refers to it as a *clause-medial* adverbial (clause-medial connective). For instance, in (29), ensuite is a clause-medial adverbial. Identifying ARG2 is problematic due to the

fact that syntactic properties of the adverbial do not associate it with the clause but rather with the VP. For instance, in TAG, the adverbial would anchor a VP-auxiliary tree, hence it modifies the VP of the clause. However, a VP cannot be a discourse argument, where a minimal (atomic) unit of discourse is a clause. Thus, in this case, there is a mismatch between syntax and discourse, that is, between the sentence-level and discourse-level analyses of a clause-medial adverbial.

(29) *Fred* *ira* *à* *Dax* *pour* *Noël.* **Il** **ira** ensuite **à**
Fred go$_{\text{FUT;3P;SNG.}}$ to Dax for Christmas. He go$_{\text{FUT;3P;SNG.}}$ afterwards to
Pau.
Pau.

'Fred will go to Dax for Christmas. He will then go to Pau.'

4.1.3 Constraints for Identifying Arguments of a Discourse Connective

Danlos (2009) studies the syntax-semantics interface for discourse by proposing a formalism called D-STAG. In D-STAG, any connective is structural, that is, both of the arguments of every connective in a discourse (if any) appear in the parse tree of a discourse. In order to identify arguments of a connective, D-STAG makes use of certain constraints.

> The arguments of a discourse relation/connective are the discursive semantic/syntactic representations of the same (continuous) discourse segments.
>
> Danlos (2011)

In D-STAG, one refers to a clause where a discourse connective appears as the *host clause* of the discourse connective. For instance, a subordinate conjunction always appears in front of its host clause (at a clause-initial position of a clause). As we already saw, an adverbial connective may either appear in front of its host clause or within its verb phrase (at a clause-medial position). The host clause of a subordinate conjunction is called an *adverbial clause*. The name adverbial clause is due to the fact that at the sentence level, an adverbial clause functions in the same way as an *adverb* as both of them modify a *matrix clause*. Regarding matrix clauses, D-STAG considers the following cases:

1. The matrix clause is on the right of the adverbial clause. In this case, the subordinate conjunction is called *postposed*.
2. The matrix clause is on the left of the adverbial clause, or inside the adverbial clause (before the VP of the adverbial clause). In this case, the subordinate conjunction is called *preposed*.

As we already saw, the host and matrix clauses of a discourse connective may not be the *arguments* of the discourse relation signaled by the discourse connective. D-STAG introduces terms the *host segment* and *mate segment* of a discourse connective in order to denote the arguments of a discourse connective/relation. The host segment is obtained

from the host clause and the mate segment from the mate clause. By comparing the notions of the host and mate segments with ARG1 and ARG2, we can see that the host segment is ARG2, whereas the mate segment is ARG1. Thus, below we may use these terms interchangeably.

In order to identify the host and mate segments of a discourse connective, D-STAG proposes the following constraints (Danlos, 2011):

Constraint 1: The host segment of a connective is identical to or starts at its host clause (possibly crossing a sentence boundary).

Constraint 2: The mate segment of an adverbial is anywhere on the left of its host segment (generally crossing a sentence boundary).

Constraint 3: The mate segment of a postposed conjunction is on the left of its host segment without crossing a sentence boundary.

Constraint 4: The mate segment of a preposed conjunction is identical to or starts at the matrix clause (possibly crossing a sentence boundary).

To illustrate the motivations behind Constraints 1-4, let us consider the following examples:

[(17)(a), repeated] *Fred is grumpy* because **he lost his keys**.

[(19)(b), repeated] *Fred went to the cinema.* Then, **he went to the bar**.

(30) When **he was in Paris**, *Fred went to the Eiffel Tower. Next, he visited the Louvre.*

In (17)(a), the connective because is a *postposed* conjunction. The mate segment of *because* coincides with the matrix clause of *because*; the host segment of because is the host clause of *because.*

In (19)(b), the mate segment of then is on its left, which is the first sentence in (19)(b) (with respect to the linear order in the surface level). The host segment of then is identical to the host clause of then, which is the second sentence.

In the case of (30), when is a *preposed* conjunction. The mate segment of when starts at the matrix clause of the first sentence in (30), but spans over the second sentence as well, i.e., crosses the sentence boundary. In this case, when is a *frame* adverbial (Charolles, 2005).

In addition to above mentioned theories, to identify arguments of discourse connectives, Wellner and Pustejovsky (2007) develop an approach based on machine learning. They train classifiers on ARG1 and ARG2. Candidates for ARG2 are selected only among those ones that lay within the same sentence as the connective. They argue that ARG1 and ARG2 for different discourse connectives behave differently. According to their results, the classifier trained on pairs of arguments performs significantly better than the one that is trained on each argument independently.

Soricut and Marcu (2003) develop a system for the sentence-level discourse parsing also based on machine learning. Their approach makes use of the lexical and syntactic information to train the system. In particular, the training set consists of triples: sentence, its syntactic tree(s), its discourse tree(s). The input for the discourse parser

is a lexicalized syntactic parse tree in which the discourse boundaries are marked. A lexicalized syntactic parse tree is associated with a set of *features*. A feature serves as a representation of the syntactic and lexical information of an attachment site of discourse units in the syntactic tree. According to the authors of the study, such features provide sufficiently rich information to enable the derivation of felicitous discourse trees (structures). All in all, to identify arguments of discourse connectives remains one of the problems in (computational) linguistics that requires further studies.

4.2 Rhetorical Structure Theory

Rhetorical Structure Theory (RST) (Mann and Thompson, 1987, 1988) is a discourse structure theory. One of the main motivations for introducing RST was to study the questions of text organization. Thus, the RST interpretation of a text is a structure that describes the way the text is organized. The RST analysis applies to a certain kind of written monologues, as the following quote indicates:

> Certain text types characteristically do not have RST analyses. These include laws, contracts, reports 'for the record' and various kinds of language-as-art, including some poetry. Mann and Thompson (1987)

One of the first applications of RST was found in the text generation task. The later developments of RST made it possible to employ RST in other tasks of computational linguistics, such as discourse parsing and summarization (Marcu, 1997).

4.2.1 Basic Principles

Given a text, its RST analysis is a hierarchically organized structure where rhetorical relations link sub-parts of a text, called *text spans.*

To analyze a text with RST, one assumes that the following hypotheses hold (Mann and Thompson, 1987):

1. Texts are not just strings of clauses. Instead, they consist of hierarchically organized clauses and groups of clauses that relate to one another in various ways.
2. These relations, which can be described functionally in terms of the purposes of the writer and the writer's assumptions about the reader, reflect the writer's options for organizing and presenting the concepts.
3. The most common type of text relation is a relation called the *nucleus-satellite* relation, in which one part of the text is auxiliary to the other, and therefore of less significance for the overall information conveyed by the discourse.

The first hypothesis suggests that any text is a hierarchically organized entity. The hierarchical organization is defined by *rhetorical connections.* A rhetorical connection consists of a two text *spans* connected by a *rhetorical relation.* One assumes that a rhetorical relation connects two non-overlapping text spans.

The second assumption provides a general purpose of rhetorical relations explained in terms of writer/reader's intention/expectations. In RST, there are two kinds of relations, *subject matter* and *presentational* ones.

- Subject matter relations make it easy for a reader to recognize that there is a *semantic* relation between the given text spans.
- Presentational relations increase some inclination in a reader. They bear some information that is beyond semantics, i.e., the information that can be attributed to pragmatics.[31]

[31]As it is noted in (Nicholas, 1994), both subject matter and presentational relations can be unified under a property of having a *perlocutionary* effect. However, since subject matter relations provide semantic information (some facts about a world), they give rise to perlocutions that are easier to perceive compared to ones that arise in the case of presentational relations.

The third assumption is characteristic to RST. According to it, a rhetorical relation may relate two text spans in such a way that one of them is more *central* to the writer's intention and/or is more informative for the reader compared to the other one. One refers to the more important text span as the *nucleus* of the rhetorical relation. The less important text span is called the *satellite* of the relation. To illustrate the difference between the nucleus and the satellite of a rhetorical relation, let us consider the following example from (Mann and Thompson, 1986):

(31) [Tempting as it be], [we shouldn't embrace every popular issue that comes along].

In Example (31), *tempting as it be* does not serve as the source of the main information conveyed by the text. The central idea that a writer communicates by means of the text (31) is expressed by [we shouldn't embrace every popular issue that comes along]. Thus, this text span is the nucleus, whereas the other is the satellite. The rhetorical relation between the nucleus and satellite is CONCESSION. In RST, one establishes CONCESSION if the nucleus expresses a situation affirmed by the writer. The satellite expresses a situation which is (apparently) inconsistent with the information expressed by the nucleus, but nevertheless it is also affirmed by the writer. Figure 4.1 shows a pictorial representation of the RST discourse structure for the text in Example (31). The horizontal segments stand for the text spans, whereas the rhetorical connection between them is depicted as the edge. The name of the rhetorical relation labels the edge. The edge is directed from the satellite to the nucleus. The vertical line indicates the position of the nucleus in the text.

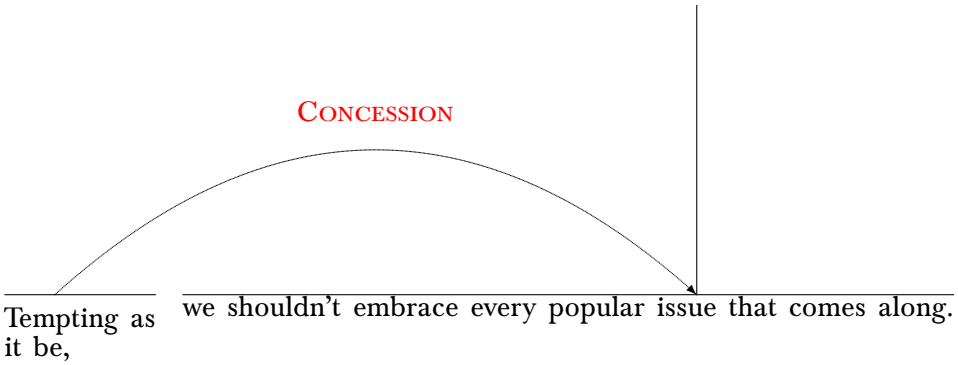


Figure 4.1: An RST structure of a discourse

It is not always straightforward to distinguish a nucleus from a satellite or vice versa. To illustrate that, let us consider the following example from (Carlson and Marcu, 2001):

(32) [Although the earnings were fine and above expectations]π_1^{32}, [Salomon's stock fell \$1.125 yesterday]π_2^{32}.

In Example (32), π_2^{32} is the nucleus and π_1^{32} is the satellite.[32] The rhetorical relation between the two is CONCESSION.

[32]We tag with π_k^m the k-th clause in the text of Example (m), unless otherwise stated.

(33) [The earnings were fine and above expectations]π_3^{33}. [Nevertheless, Salomon's stock fell \$1.125 yesterday]π_4^{33}.

The semantic content of the text in Example (33) is very similar to the one of the text in Example (32). Indeed, the clauses π_1^{32} and π_1^{33} have the same contents, and at the same time, the clauses π_2^{32} π_2^{33} have the same contents as well. Moreover, the discourse units π_2^{32} and π_2^{33} are ***in contrast*** to each other due to the presence of the cue word ***nevertheless***. Thus, the discourses (32) and (33) are semantically very similar to each other. In spite of that, the RST structures of the discourses (32) and (33) are different from each other. Indeed, in the case of the text in Example (33), both π_1^{33} and π_2^{33} have the same status in the text, because both of them convey the same kind of information of the same importance. Hence, both π_1^{33} and π_2^{33} are nuclei in Example (33). The RST relation between them is CONTRAST. Figure 4.2 depicts the RST structure corresponding to Example (33). Thus, while π_1^{32} plays the role of the satellite in the discourse (32), in a semantically very similar discourse, namely, in the discourse (33), π_1^{33} is a nucleus. This suggests that one may not be able to detect the distinction between a satellite and a nucleus by examining only semantic contents.

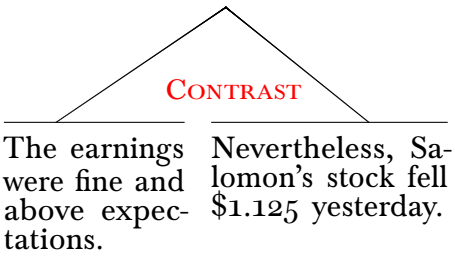


Figure 4.2: A multinuclear discourse structure

One of the guidelines to identify satellites and nuclei in a text suggests that by discarding the satellites, the remaining text (consisting of only nuclei) should convey the most essential information provided by the original text. Another suggestion is that replacing the satellite in the text (with something else) should not induce (significant) changes in the intended function of the text.

We say that a relation is *multinuclear* if both of the discourse units it connects are nuclei (e.g. CONTRAST). Otherwise, if only one of them is nucleus, then we say that a relation is *mononuclear* (e.g. CONCESSION).

Relation name	CONCESSION
Constrains on N	W has positive regard for the situation presented in N.
Constrains on S	W is not claiming that the situation presented in S does not hold.
Constrains on N+S	W acknowledges a potential or apparent incompatibility between N and S; W regards the situations presented in N and S as compatible; recognizing the compatibility between N and S increases R's positive regard for N.
Effect	R's readiness to accept W's right to present N is increased.
Locus of the effect	N and S.

Figure 4.3: The RST definition of the rhetorical relation CONCESSION

In RST, a rhetorical relation is defined in terms of the constraints on the text spans it connects. These constraints are formulated in terms of perlocutionary effects which the reader experiences/writer intends. The complete RST recipe of defining a relation is as follows (Mann and Thompson, 1988):

1. Constrains on the nucleus of the relation.
2. Constrains on the satellite of the relation.
3. Constrains on the combination of the nucleus and satellite of the relation.
4. Effect on the reader.

Figure 4.3 on the previous page provides an example of a definition of a rhetorical relation. Table 4.1 provides concise descriptions of the definitions of the rhetorical relations that we are going to use in further examples.

Relation Name	*Nucleus*	*Satellite*
ANTITHESIS	ideas favored by the author	ideas disfavored by the author
CIRCUMSTANCE	text expressing the events or ideas occurring in the interpretive context	an interpretive context of situation or time
ELABORATION	basic information	additional information
ENABLEMENT	an action	information intended to aid the reader in performing an action
EVIDENCE	a claim	information intended to increase the reader's belief in the claim
JUSTIFICATION	text	information supporting the writer's right to express the text
MOTIVATION	an action	information intended to increase the reader's desire to perform the action
NON-VOLITIONAL CAUSE	a situation	another situation which causes that one, but not by anyone's deliberate action
PURPOSE	an intended situation	the intent behind the situation
SOLUTIONHOOD	a situation or method supporting full or partial satisfaction of the need	a question, request, problem or other expressed need

Table 4.1: Rhetorical relations

4.2.2 Schemas

In RST, while rhetorical relations are the first-level objects, the second-level objects are *schemas.* A schema is defined in terms of either one or two relations. Each *schema* indicates the way a text span is built out of other text spans. By recursively applying schemas, one derives the RST analysis of a text. The simplest schema consists of a single relation and two text spans. For instance, Figure 4.4(a) shows the Circumstance schema, which is made up with the help of a single rhetorical relation CIRCUMSTANCE. While horizontal lines denote the text spans, the vertical line indicates the position of the nucleus of the relation. The edge linking the horizontal lines is labeled with the name of the rhetorical relation. However, a schema may consist of more than one rhetorical relation. One refers to such schemas as *multi-relation schemas.* Figure 4.4(b) illustrates an example of a multi-relation schema made out of two rhetorical relations, MOTIVATION and ENABLEMENT.

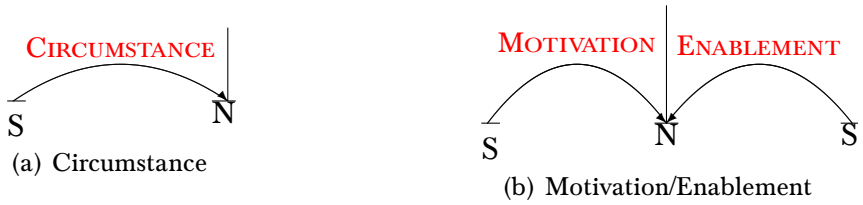
In using schemas, RST provides one with a certain degree of freedom. Namely, one is allowed to do the following manipulations with schemas:

Unordered Spans: The schemas do not put any restriction on the order of nuclei or satellites in the text span in which the schema is applied.

Optional Relations: In the case of multi-relation schemas, all relations are optional, but in the schema application at least one of the relations contained in that schema must hold.

Repeated Relations: All the relations that are part of a given schema can be applied any number of times in the application of that schema.

(4.34)



(a) Circumstance

(b) Motivation/Enablement

Figure 4.4: RST schemas

Despite permitting a certain degree of flexibility in schema application, according to (Mann and Thompson, 1988), *a set of schema applications* qualifies as a structural *analysis of a text* if the set of schema applications satisfies the following constraints:

Completedness: The set contains one schema application that contains a set of text spans that constitute the entire text.

Connectedness: Except for the entire text as a text span, each text span in the analysis is either a minimal unit or a constituent of another schema application of the analysis.

Uniqueness: Each schema application consists of a different set of text spans, and, within a multi-relation schema, each relation applies to a different set of text spans.

Adjacency: The text spans of each schema application constitute one text span.

Completedness, Connectedness, and Uniqueness ensure that an RST analysis results in a tree-like, hierarchically structured entity. Adjacency guarantees that the resultant structure contains no crossing edges. For instance, Figure 4.5 shows the RST structure of the following discourse:

(35) [No matter how much one wants to stay a nonsmoker]π_1^{35}, [the truth is that the pressure to smoke in junior high is greater than it will be any other time of one's life]π_2^{35}. [We know that 3,000 teens start smoking each day]π_3^{35}, [although it is a fact that 90% of them once thought that smoking was something that they'd never do]π_4^{35}.

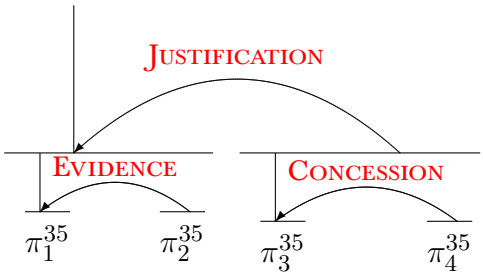


Figure 4.5: An RST discourse structure

As Figure 4.5 indicates, one obtains the RST structure of the discourse (35) by using the Justification schema, the Evidence schema, and the Concession schema.

4.2.3 A Formalization of RST

4.2.3.1 RST Structures as Trees

Although one may find RST structures to be close to trees, they are not trees by definition. The asymmetry between a nucleus and a satellite makes RST structures different from trees. Marcu (1997, 2000) proposes to transform RST structures into binary trees. He also presents a mathematical formalization of the notion of a rhetorical relation. In addition, he defines *valid text structures*. With the help of the notion of a valid text structure, Marcu (1997) gives a formal definition of the RST parsing problem: Parsing a text with RST amounts finding all valid text structures of that text.

Let REL be a rhetorical relation. Marcu (1997) encodes REL as a sorted predicate *rhet-rel*(*Rel*, u_i, u_j), whose meaning is that the relation REL holds between the text spans denoted by u_i and u_j. Here, the pair u_i and u_j stand either for a pair of a satellite and a nucleus or for a pair of a nucleus and a nucleus, respectively, depending on whether REL is a mononuclear or multinuclear relation. Once every relation is of the form *rhet-rel*(*Rel*, u_i, u_j), to represent an RST discourse structure as a binary tree, one decorates every node in a tree with the features *Status*, *Type*, and *Salience* or *Promotion*, where:

- *Status* indicates whether the node stands for the nucleus or for the satellite of a relation.
- *Type* indicates the rhetorical relation that holds between the text spans over which the node spans.
- *Promotion* is the set of units that constitute the most *important* part of the text spanned by the node.

For a leaf node, its type is *leaf*, and its promotion set is the textual unit that it corresponds to. For example, one encodes the RST structure shown in Figure 4.5 with the tree in Figure 4.6.

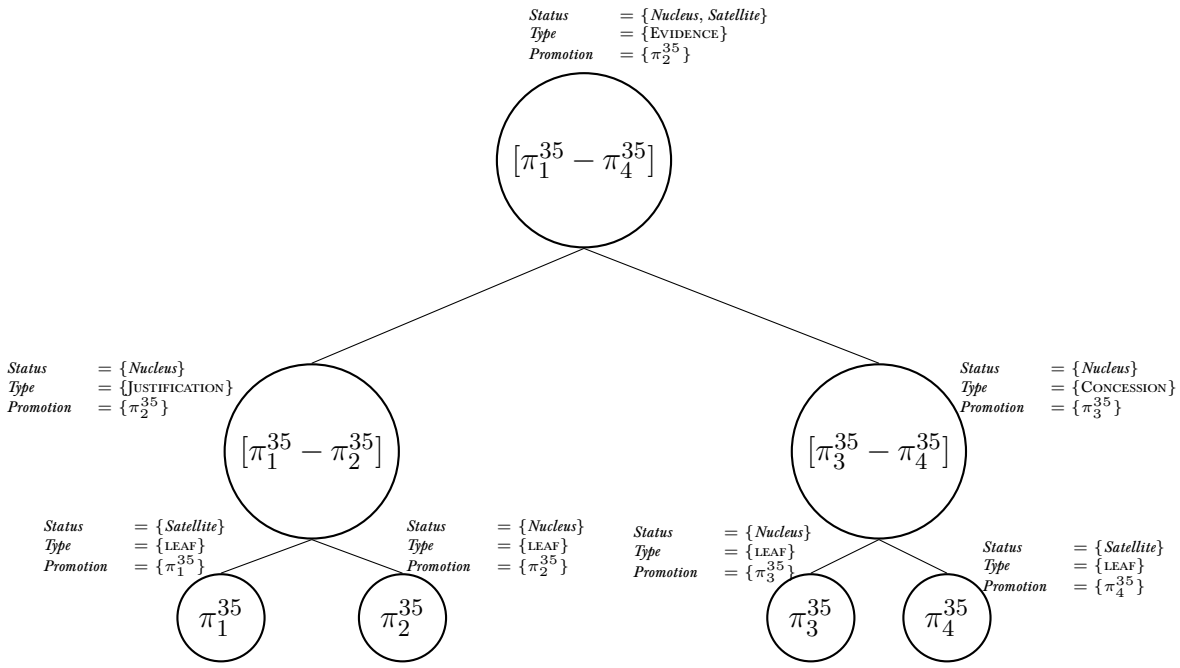


Figure 4.6: A binary tree corresponding to an RST Structure

4.2.3.2 An Extension of RST

To make use of RST in the discourse parsing task, one has to be able to distinguish well-formed RST structures from ill-formed ones. Although RST provides some instructions for building schemas (Completeness, Connectedness, Uniqueness, and Adjacency), RST does not define what a well-formed structure is. Furthermore, Marcu (1997) claims that since RST misses compositionality principles, it is impossible to formally define the notion of parsing with RST. His point could be rephrased as follows: If one has to continue building a partially built discourse, then what is a procedure that allows one to decide how to proceed? Let us assume that π_a and π_b are two adjacent text spans such that their discourse structures are already built. Assume also that some rhetorical relation holds between two minimal units each belonging to the spans π_a and π_b. Now, a question is whether π_a and π_b are also related by a rhetorical relation. Marcu (1997) answers this question by providing two compositionality criteria, which he proposed as a result of analysis of a number of texts.

Proposition 4.2.1 (Weak Compositionality Criterion (Marcu, 1997)).
If a relation $\mathcal{R}$ holds between two nodes of the tree structure of a text, then $\mathcal{R}$ can be explained in terms of a similar relation $\mathcal{R}$ that holds between two linguistic or nonlinguistic constructs that pertain to the most important constituents of those nodes.

Proposition 4.2.2 (Strong Compositionality Criterion (Marcu, 1997)).
If a rhetorical relation $\mathcal{R}$ holds between two textual spans of the tree structure of a text, then it can be explained by a similar relation $\mathcal{R}$ that holds between at least two of the most important textual units of the constituent spans.

> The intuitive notion behind the stronger criterion is that, after all, all the linguistic and nonlinguistic constructs that are used as arguments of rhetorical relations can be derived from the textual units and the relations that pertain to those units. Marcu (2000)

Although from a theoretical point of view, it is easier to satisfy the weak compositionality criterion than the strong one, from a practical point of view, to use the strong compositionality criterion could be more beneficial than the weak one. However, as Marcu (1997) notices, in certain cases, neither of these compositionality criteria can be applied. To illustrate that, he provides the following example:

(36) [He wanted to play squash with Janet]π_1^{36}, [but he also wanted to have dinner with Suzanne]π_2^{36}. [He went crazy]π_3^{36}.

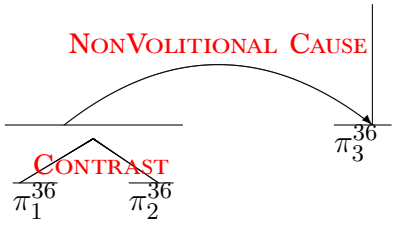


Figure 4.7: An RST structure of a text

Figure 4.7 depicts the discourse structure of the text (36). π_1^{36} and π_2^{36} are in the relation Contrast. It is hardly possible to imagine any rhetorical relation that could hold either between π_1^{36} and π_3^{36} or π_2^{36} and π_3^{36}. Nevertheless, the discourse (36) is a felicitous one. The rhetorical relation Non-Volitional Cause holds between π_3^{36} and the text span consisting of π_1^{36} and π_2^{36}. In other words, π_3^{36} is caused neither by π_1^{36} nor π_2^{36} but by the fact that π_1^{36} and π_2^{36} are in Contrast. Since Contrast is multinuclear, both π_1^{36} are π_2^{36} are of the same importance. Thus, one cannot argue that Non-Volitional Cause holds between π_3^{36} and the most important one among π_1^{36} are π_2^{36}. One concludes that, in this case, not only the strong compositionality criterion yields an incorrect analysis, but even the weak compositionality criterion cannot be satisfied.

4.2.3.2.1 Extended Relations

Marcu (1997) argues for introducing another kind of discourse relations in addition to rhetorical relations. He calls them *extended relations*. To illustrate the reasons for introducing extended relations, let us consider a text where paragraphs are listed using

keywords *first*, *second*, etc. In order to produce the precise analysis of such a text, one has to give an account of the fact that these paragraphs are in the relation LIST, where LIST is a relation that holds between paragraphs in a text rather than elementary units. If there is no notion that could take care of such relations like LIST, then according to the strong compositionality principle, there should be a relation (reminiscent of LIST) between some elementary units in these paragraphs. However, while the listing *first*, *second*, etc. may make sense for paragraphs, it could be the case that these paragraphs do not contain such textual units that can be considered as a list: *first, idea1*; *second, idea2*; etc. Thus, the strong compositionality principle would fail in that case. Hence, if one extends RST with the strong compositionality principle, one has to allow for relations like LIST. In Marcu's (1997) terminology, LIST is an extended relation. LIST does not qualify as a rhetorical relation because it does not connect elementary discourse units but paragraphs. In general, an extended relation is like a rhetorical relation but it holds between larger portions of text rather than elementary units.

4.2.3.2.2 Nondeterminism

In certain cases, one may associate several structures to a given discourse. This makes discourse parsing task ambiguous. As an instance of an ambiguous discourse, let us consider the following example from (Marcu, 1997):

(37) [John likes sweets]π_1^{37}. [Most of all, John likes ice cream and chocolate]π_2^{37}. [*In contrast*, Mary likes fruits]π_3^{37}. [*Especially* bananas and strawberries]π_4^{37}.

The discourse (37) consists of four minimal units. The discourse connective *in contrast* signals the rhetorical relation CONTRAST. However, one has to identify what are the spans that CONTRAST connects. One may consider the several possible cases: CONTRAST connects [π_1^{37}-π_2^{37}] and [π_3^{37}-π_4^{37}]; CONTRAST connects [π_1^{37}-π_2^{37}] and π_3^{37}; CONTRAST connects π_1^{37} and π_3^{37}; CONTRAST connects π_2^{37} and π_3^{37}, etc. In addition, the discourse connective *especially* gives rise to the ELABORATION relation, which may also hold between various text spans in the text. RST does not provide a formal definition that one could use to justify why their choice of a particular structure is more admissible than the other ones. In some cases, it is not even clear whether there is the best structure among various possible ones. If we had a text that is larger than the one in Example (37), then it is likely that it would even more ambiguous, because RST trees for larger texts would contain more nodes. Therefore, there would be more text spans and consequently, there would be more options for linking various text spans. To overcome the problems that arise to due to ambiguity in the parsing task, Marcu (1997) introduces the *exclusive disjunction* operator, $\bigoplus$. It enables one to encode a non-deterministic choice. For instance, in the case of the discourse (37), one makes

hypothesis such the following one:

$$\begin{aligned} h = &\textit{rhet-rel}(\textsc{contrast}, \pi_1^{37}, \pi_3^{37}) \bigoplus \textit{rhet-rel}(\textsc{contrast}, \pi_1^{37}, \pi_4^{37}) \bigoplus \\ &\textit{rhet-rel}(\textsc{contrast}, \pi_2^{37}, \pi_3^{37}) \bigoplus \textit{rhet-rel}(\textsc{contrast}, \pi_2^{37}, \pi_4^{37}) \bigoplus \\ &\textit{rhet-rel}(\textsc{Elaboration}, \pi_2^{37}, \pi_1^{37}) \bigoplus \textit{rhet-rel}(\textsc{Elaboration}, \pi_4^{37}, \pi_1^{37}) \bigoplus \\ &\textit{rhet-rel}(\textsc{Elaboration}, \pi_4^{37}, \pi_2^{37}) \bigoplus \textit{rhet-rel}(\textsc{Elaboration}, \pi_4^{37}, \pi_3^{37}) \end{aligned} \tag{4.38}$$

Note that h only contains relations between atomic text units, because according to the strong compositionality principle, if a relation between larger discourse units holds, then the same relation hold between smaller discourse units. One can check that h encodes all possible rhetorical connections that CONTRAST and ELABORATION can provide between the discourse units in the discourse (37).

However, using the exclusive disjunction may prove to be not productive if one relates most of the text spans with the exclusive disjunction. Given a relation, one has to have an idea what are the possible text spans that this relation can relate. With regard to this matter, Marcu (1997) proposes the *exclusively disjunctive hypothesis.* According to this hypothesis, all the disjuncts corresponding to a rhetorical relation should be *from the same area* of a text. More formally, one formulates the exclusively disjunctive hypothesis as follows:

Definition 4.2.1 (Exclusively Disjunctive Hypothesis (Marcu, 2000)).
An exclusively disjunctive hypothesis of rhetorical relations is well formed *if all textual spans that have as boundaries the units found in each disjunct overlap.*

Now, we can formulate Marcu's (2000) definition of the *text structure derivation.*

Definition 4.2.2 (Text Structure Derivation (Marcu, 2000)).
The problem of text structure derivation: Given a sequence of textual units $U = u_1, u_2, \ldots, u_n$ *and a set RR of simple, extended, and well-formed exclusively disjunctive rhetorical relations that hold among these units and among contiguous textual spans that are defined over U, find all valid text structures of the linear sequence* U.

In this way, Marcu (1997) formalizes RST by adding to it compositionality principles and extending its relations. This allows one to formally define the notion of a text structure derivation in RST. It is noteworthy that Marcu (1997, 2000) provides both a model-theoretic and a proof-theoretic account of valid text structures. He shows that the proof theory is both sound and complete, that is, one derives all and only valid text structures of RST using the proof theory.

4.3 Segmented Discourse Representation Theory

SDRT is a theory of dynamic semantics that studies the relations between discourse, pragmatics, and semantics (Asher and Lascarides, 2003). By formalizing pragmatic reasoning, SDRT builds the logical representation of discourse. With the help of the dynamic semantics, SDRT identifies formal properties of that logical form. SDRT incorporates the discourse structure within the logical form of a discourse. In the SDRT discourse structure, rhetorical relations provide connections between discourse constituents (units). A rhetorical relation connects two utterances. Although under the term utterance one may understand various kinds of information, for the sake of simplicity, in this thesis, we assume that a rhetorical relation links propositions, unless otherwise stated.

In order to formalize the notion of a discourse structure within a dynamic setting, SDRT starts with atomic discourse units, that is, discourse units whose representations do not involve rhetorical relations. To obtain the logical representation of a discourse unit, SDRT refers to dynamic semantic theories such as DRT (Kamp and Reyle, 1993) or DPL (Groenendijk and Stokhof, 1991). In this thesis, we will use DRT in order to represent interpretations of atomic discourse units.

To build the logical form of a discourse, SDRT deals with various tasks, including inference of rhetorical relations. For that, SDRT defines a logic, which is different from the dynamic logic. The reason behind having two distinct logics is to separate the different levels of *knowledge*. SDRT distinguishes the *semantic* information from the *pragmatic* one. In order to reason about the semantic properties of a discourse, one utilizes the dynamic logic of SDRT. That is why the dynamic logic of SDRT is also called the *logic of information content*. The logic that enables one to make pragmatically preferred decisions, i.e., that formalizes the pragmatic knowledge is referred to as the *logic of information packaging*. Below, we briefly describe motivations for introducing SDRT and its fundamental principles. Then, we discuss the language of SDRT and the logic of information content.

4.3.1 Basic Principles of SDRT

4.3.1.1 Discourse Coherence

One of the basic principles in SDRT is *discourse coherence*, which can be formulated as follows:

Definition 4.3.1 (Discourse Coherence (Asher and Lascarides, 2003)).
A discourse is coherent if:

(a) Every proposition (and question and request) that is introduced in the discourse is rhetorically connected to another bit of information in the discourse, resulting in a single *connected structure for the whole discourse;*

(b) and all anaphoric expressions can be resolved.

Thus, a coherent discourse has a single connected structure, which serves as an interpretation of the discourse. Definition 4.3.1 does not specify what happens if several

coherent interpretations of a discourse are simultaneously available. While a discourse may have several coherent interpretations, one can argue that some of them are *more coherent* than others. Let us illustrate this claim with the help of the following example:[33]

(39) a. Max moved from Nancy to Vandœuvre.

b. The rent was less expensive.

In the sentences (39)(a) and (39)(b), *the rent* may refer either to *the rent in Nancy* or to *the rent in Vandœuvre.* Thus, the following two interpretations of (39) can be considered:

Interpretation 1 *The rent = The rent in Nancy.* In this case, the discourse (39) is merely about Max's moving from Nancy to Vandœuvre, which is the information that (39)(a) provides. The sentence (39)(b) only serves as the background information to (39)(a), because its only contribution to the overall information in the discourse is to inform a reader about *the rent in Nancy* compared to *the rent in Vandœuvre.*

Interpretation 2 *The rent = The rent in Vandœuvre.* In this case, the discourse (39) has the following meaning: *The reason for Max's moving from Nancy to Vandœuvre is the cheaper rent in Vandœuvre compared to the one in Nancy.* Now, the information provided by the sentence (39)(b) is not only the background information in the discourse, but also explains the sentence (39)(a).

Although both of these interpretations are coherent, Interpretation 2 offers *a richer discourse structure* than Interpretation 1 does. In particular, in the case of Interpretation 2, the sentences (39)(a) and (39)(b) are linked with two rhetorical relations, BACKGROUND and EXPLANATION, whereas in the case of Interpretation 1, only BACKGROUND links the sentences (39)(a) and (39)(b). That is why Interpretation 2 is *pragmatically more preferable* than Interpretation 1. To establish the interpretations a discourse, SDRT selects the pragmatically most preferable interpretations. If there is only one such interpretation, then it is declared as the interpretation of the discourse. For instance, Interpretation 2 is the SDRT interpretation of the discourse (39). If there are several such interpretations, then either the discourse is ambiguous or one does not have enough information to identify *the* interpretation of it. To illustrate that, let us consider the following example:

(40) a. Max was released from hospital.

b. He recovered completely.

In the discourse (40), it is not clear whether *Max left hospital because he recovered completely*, or *Max was released from hospital* and *then he recovered completely.* Hence, we have two possibilities:

1. Either the sentence (40)(b) *explains* the sentence (40)(a),
2. or the sentence (40)(b) *follows* (in terms of the temporal order) the sentence (40)(a).

In the first case, the rhetorical relation EXPLANATION is the link between two propositions, while in the second one, the link between two propositions is NARRATION

[33]All the examples in this section are borrowed from (Asher and Lascarides, 2003) with slight simplifications, unless otherwise stated.

(from *narrative*). Both of these interpretations are equally possible as they are equally coherent for us because we do not have enough information to prefer one to the other. Now, let us consider the following two discourses, which are two possible extensions corresponding to the two interpretations.

(41) a. Max was released from hospital.
b. He recovered completely,
c. and they needed the bed.

(42) a. Max was released from hospital.
b. He recovered completely,
c. then, he resumed training.

In the discourse (41), *he recovered completely* serves as an explanation why *Max was released from hospital.* With the same success, in (42),[34] *he recovered completely* relates to *Max was released from hospital* with NARRATION. Thus, in two different situations, both of the interpretations are equally acceptable. In the cases such as (40), where we cannot specify what is the interpretation of a discourse, SDRT makes use of an *underspecified* representation of a discourse, which we will discuss in more details in the further sections.

Nevertheless, in some cases (like the discourse (39)), one is able to identify what is the interpretation of a discourse. In order to formally define what is the most coherent interpretation, SDRT makes use of the principle called *Maximise Discourse Coherence* (MDC).

Definition 4.3.2 (Maximise Discourse Coherence (Lascarides and Asher, 2007)).
The logical form of a discourse is always a logical form that is maximal in the partial order of the possible interpretations, where the ranking of interpretations is performed according to following principles:

- *All else being equal, the more rhetorical connections there are between two items in a discourse, the more coherent the interpretation.*
- *All else being equal, the more anaphoric expressions whose antecedents are resolved, the higher the quality of coherence of the interpretation.*
- *All else being equal, an interpretation which maximizes the quality of its rhetorical relations is more coherent than one that does not.*

The last principle of MDC involving the notion of a *quality* of a rhetorical relation is related to the fact that some relations are *scalar.* A relation is scalar if its quality may *vary* depending on a discourse where it appears. For instance, the following discourse relations are scalar: NARRATION, CONTRAST, and PARALLEL. The quality of each of these relations depends on different factors.

[34]Example (42) is from (Prévot and L. Vieu, 2008).

NARRATION

One links two propositions with NARRATION only if one of them temporarily precedes the other one. For instance, in each of the discourses in (43), the first sentence temporarily precedes the second one and therefore it makes sense to further check whether they are linked with NARRATION.

(43) a. Yesterday, Pedro noticed a lovely donkey near his farm. He gave it some carrots.

b. Yesterday, Pedro noticed a lovely donkey near his farm. He ate some carrots.

Actually, in both of the discourses (43)(a) and (43)(b), NARRATION connects the first and the second propositions. Although both of these texts are coherent, the *quality* of the discourse (43)(a) is better than the *quality* of (43)(b). The reason is that the quality of NARRATION gets higher as *the common topic* of the utterances it links gets more specific. Indeed, the common topic of the first and second propositions in (43)(a) includes *Pedro* and *a donkey*, while in the case of (43)(b), the common topic of the related propositions is only *Prado*. That is why the quality of NARRATION in (43)(a) is better than in (43)(b). Thus, NARRATION is a scalar relation.

CONTRAST

(44) a. John loves to collect classic cars. But his favorite car is a 2012 Ford Mondeo.

b. John loves to collect classic cars. But, he hates football.

The quality of CONTRAST depends on a degree of *dissimilarity* between the propositions it relates. For example, the quality of CONTRAST in the discourse (44)(a) is better than the quality of CONTRAST in the discourse (44)(b). Since *dissimilarity* can be graded in terms of *more dissimilar*/*less dissimilar*, one concludes that CONTRAST is a scalar relation.

PARALLEL

(45) a. John has brown hair and Bill has brown eyes.

b. John has brown hair and Bill likes brown eyes.

In (45)(a), two propositions, *John has brown hair* and *Bill has brown eyes* are in the PARALLEL relation. In the case of (45)(b), *John has brown hair* and *Bill likes brown eyes* are also in the PARALLEL relation. However, as one may notice, the quality of PARALLEL in (45)(a) is significantly better compared to the quality of PARALLEL in (45)(b). The greater similarity between the contents of the propositions related by PARALLEL provides the better quality of PARALLEL. Thus, PARALLEL is a scalar relation.

4.3.1.2 The Right Frontier Constraint

How to increment a discourse so that one maintains its coherence is one of the problems that SDRT studies extensively. As we saw in the definition of discourse coherence (see Definition 4.3.1 on page 107), a new piece must add by some rhetorical relation to the current one. However, a question is *where* a new piece can attach to the current one. To put it another way, SDRT aims to identify the possible candidates amongst the discourse constituents in a given discourse that can make a rhetorical connection with a new piece of discourse. For the sake of illustration, let us consider the following example:

(46) π_1. Max had a great evening last night.
π_2. He had a great meal.
π_3. He ate salmon.
π_4. He devoured lots of roquefort.
π_5. He then won a dancing competition.

Example (46) illustrates a coherent discourse. However, adding the following proposition to (46) yields an incoherent discourse:

(47) π_X. It was beautiful pink.

Adding π_X to (46) makes the resultant discourse infelicitous because the pronoun *it* from π_X does not find an antecedent in (46). At the same time, one knows that the pronoun *it* in π_X could only refer to *salmon* in π_3 as it is the only thing in the discourse that is *pink*. We can even substitute *it* by *the salmon* in π_X, i.e., we could try to increment (46) with *the salmon was beautiful pink* instead of π_X. Nevertheless, the result would be as incoherent as in the previous case. Hence, the incoherence of the discourse '(46)+π_X' is not due to a problem of ambiguity of the anaphora resolution task (since there is no ambiguity). To identify the problem, let us build the *rhetorical structure* of (46).

The propositions π_2 and π_5, both *elaborate* π_1 as each of them expresses a *sub-event* (sub-part) of the event corresponding to π_1. Indeed, to *have a meal* (π_2) and to *win a dancing competition* (π_5) are among the events that happened at *the evening last night* (π_1). In such cases, SDRT links two propositions with the rhetorical relation called ELABORATION. π_2 and π_5 are also related to each other. There is a temporal relation between them: first, π_2 took place and when it was over *then* π_5 began. The cue word *then* in π_5 makes this temporal relation explicit. One concludes that the link between π_2 and π_5 is NARRATION.

The propositions π_3 and π_4, both elaborate π_2 as they express sub-events of *having a meal* (π_2). Thus, ELABORATION links π_3 and π_2 and it also links π_4 and π_2. Let us check whether there is a temporal link between π_3 and π_4. There is no explicit (overt) marker in π_3 nor in π_4 to give us a hint about the temporal order between them. As we have no sentence-level linguistic clue for ordering the set $\{\pi_3, \pi_4\}$, we rely on the *discourse-level* information. In particular, one assumes that the textual order of these

clauses indicates the order of the events expressed in them. Therefore, we assume that the event of π_3 took place before the one of π_4, that it, NARRATION connects the propositions π_3 and π_4.

Thus, we obtain the discourse structure, which one can represent as a directed acyclic graph (DAG). Figure 4.8 on the next page illustrates this DAG. The edges of the DAG are labeled with the discourse relations. The nodes in the DAG stand for discourse constituents. As one may notice, the edges corresponding to NARRATION are displayed horizontally, while the ones for ELABORATION are vertical. This reflects the fact that NARRATION induces *coordination*, while ELABORATION induces *subordination* in the discourse structure (we say that NARRATION is coordinating, while ELABORATION is subordinating). A hypothesis is that no two nodes are connected by a subordinating relation and a coordinating relation simultaneously. As SDRT suggests, the presence of subordinating and coordinating relations in a discourse affects the *availability of antecedents of anaphoric expressions.* This relates to the notion of the *right frontier* (RF) of a discourse. In the definition of an RF, the notions of subordination and coordination play the central role. One can determine the RF of a DAG representing a rhetorical structure of a discourse. Indeed, since SDRT puts down edges either horizontally or vertically in a graph representing the structure of the discourse, one can visualize the RF of such a graph. For instance, the RF of the graph illustrating the discourse structure of (46) (see Figure 4.8) consists of π_1 and π_5. With the help the notion of an RF, one defines the *Right Frontier Constraint* (RFC), which originates from Polanyi's (1985) work on discourse. According to the RFC, two pieces of discourse must meet certain requirements in order to be possible to put them together. The original version only concerns anaphoric pronouns, but the SDRT one (whose exact formulation we provide in Definition 4.3.12 on page 123) is more general as it deals with all *anaphoric elements.* For now, we formulate the RFC without using formal notions.

Definition 4.3.3 (RFC (Asher and Lascarides, 2003)).
A new piece of discourse can add to the current one only in the case where the anaphoric expressions occurring in this new piece have antecedents in the clauses that belong to the RF of the current discourse.

Now, one can explain why adding the clause π_X (see (47)) to the discourse (46) turns a coherent discourse into an incoherent one. The RFC prohibits adding π_X to (46) as the antecedent of the anaphoric expression *it* from π_X is introduced in the discourse unit π_3, which does not belong to the RF of the discourse (46) ($\pi_3 \notin \{\pi_1, \pi_5\}$).

4.3.2 The Logical Form of Discourse

In SDRT, the logical form of a discourse involves representations of discourse units and rhetorical connections, which are realized by rhetorical relations as they connect utterances. In this way, the logical form of a discourse is a logical representation of the information content of a discourse. The language where SDRT defines the logical form of discourse is called the *SDRS* language. SDRT defines well-formed formulas of this language and provides them with dynamic semantic interpretations.

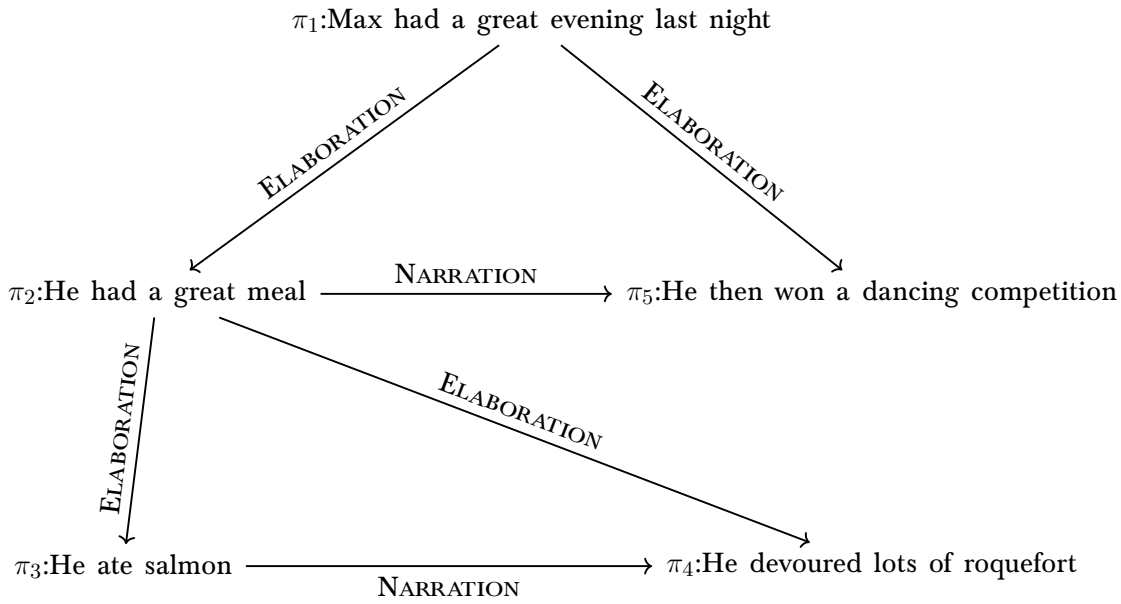


Figure 4.8: A rhetorical structure of a text

4.3.2.1 The Logical Form of Clauses

The SDRT discourse analysis starts from identifying meanings of sentences. A sentence might be ambiguous as it can have several meanings. The question is *how* to represent the meaning of an ambiguous sentence, or *what* is the meaning of an ambiguous sentence. Is that the set of possible meanings, or is that only one meaning? One may answer these questions by declaring the *pragmatically* most acceptable meaning among all the possible ones to be *the* meaning of the ambiguous sentence. Indeed, it would be a solution as the goal of SDRT is to interpret discourse by formalizing pragmatics. However, it is not always possible to select the pragmatically most preferred meaning. That is why SDRT chooses another solution, which enables one to obtain a *representation* of a sentence that does not require to select any of the particular meanings. This representation is called the *underspecified logical form* of a sentence; it represents all the possible meanings of an ambiguous sentence. To define underspecified logical forms, SDRT employs a *labeled* language. The labeled language of SDRT makes possible to encode an underspecified meaning of a sentence in a concise way. For the sake of illustration, let us consider the following example.

(48) Every boxer loves a rock-band.

$$\phi_1 = \quad \forall x(\textbf{boxer}(x) \rightarrow \exists y(\textbf{rock-band}(y) \wedge \textbf{love}(x, y))) \tag{4.49}$$

$$\phi_2 = \quad \exists y(\textbf{rock-band}(y) \wedge (\forall x(\textbf{boxer}(x) \rightarrow \textbf{love}(x, y)))) \tag{4.50}$$

The sentence (48) is ambiguous because the following two readings are available for it: either *for every boxer, there is a rock band that they love*, or *there is a rock-band such that every boxer loves it.* ϕ_1 and ϕ_2, defined in Equation (4.49) and Equation (4.50) respectively, encode the two readings of the sentence (48). These two readings of (48) correspond to the scope ambiguities of the quantifies for $\forall x$ and $\exists y$. To select one of the meanings of the sentence in (48) is to set that $\forall x$ outscopes (scopes over) $\exists y$, or vice versa. At the same time, in both ϕ_1 and ϕ_2, each of the quantifiers $\forall x$ and $\exists y$ outscopes $\mathbf{love}(x, y)$. An underspecified representation of the sentence (48) should encode these facts.

In order to consider scopes of expressions and how they are related, it is useful to *name* these expressions using *labels.* That is, we associate labels with expressions. A given label is associated with only one expression. One can translate scoping relations between expressions into scoping relations between their labels. For instance, if we label $\forall x$ with l_1 and $\exists y$ with l_2, instead of saying that $\forall x$ outscopes $\exists y$ or vice versa, we can express that by using l_1 and l_2. The original language is called the base language. One can translate the base language expressions into the label language ones (and vice versa, i.e., one can reconstruct the base language expressions from the label language ones).

In this thesis, we will use another approach to underspecified representations, called *Hole Semantics* (Bos, 1995). With this approach, one makes use of *holes* in order to represent *unassigned* scopes between labeled expressions. One encodes every scope bearing expression using holes. A hole can be seen as a *variable* over labels. Given an underspecified representation, by instantiating the holes in it with labels, one generates a disambiguated representation out of the underspecified one. *Scoping (domination) constraints* govern which labels can fill which holes. That is, to fill holes with labels, one has to obey scoping constraints. No label can be plugged in two different holes at the same time. Each hole should be filled by some label. To plug a label into a hole is to substitute a hole by a label. Domination constraints have the following form $a \leq b$, where a and b are formulas built with holes and labels. $\leq$ is a partial order. For instance, $l_1 \leq l_2$ encodes that the formula with label l_1 is a *subformula* of the one with label l_2. The constraint such as $l \leq h$ means that the formula with label l is in the scope of an operator with hole h. This means that l directly or indirectly is in the scope of h of that operator. If the operator with hole h directly scopes over the formula with label l, then one fills h by l.

Formally, one defines *underspecified representations for predicate logic.*[35] The language where one encodes expressions with holes and labels is called *Predicate Logic Unplugged* (PLU). One defines its syntax as follows:

Definition 4.3.4 (Syntax of PLU (Bos, 1995)).

1. *If h is a hole, then $\neg h$ is a PLU formula;*
2. *if h_i and h_j are holes, then $h_i \rightarrow h_j$, $h_i \wedge h_j$, $h_i \vee h_j$ are PLU formulas;*
3. *x is a variable of Predicate Logic and h is a hole, then $\exists x h$ and $\forall x h$ are PLU formulas;*
4. *if R is an n-place predicate of and $t_1, \ldots t_n$ are terms, then $R(t_1, \ldots, t_n)$ is a PLU formula.*
5. *Nothing else is a PLU formula.*

[35]It is possible to make use of Hole Semantics for various languages, not only for predicate logic.

In order to obtain an underspecified formula encapsulating the meaning of (48), consider again ϕ_1 and ϕ_2 defined in Equations 4.49 and Equations 4.50. We aim to build an underspecified formula that should not specify the right hand-side of the implication $\rightarrow$, nor the conjunct of $\wedge$, otherwise we obtain either ϕ_1 and ϕ_2, which are concrete interpretations of (48). From now, we write $\textbf{and}(a, b)$ instead of $a \wedge b$ (not to confuse the logical connective $\wedge$ of PLU with the logical connective $\wedge$ of predicate logic). In addition, the underspecified formula should encode what is shared by both of the interpretations ϕ_1 and ϕ_2. Otherwise we may build an underspecified formula that is more general than the one the one that we aim to construct. Thus, the underspecified formula should encode the following things: (a) $\forall x$ directly scopes over $\rightarrow$; (b) $\textbf{boxer}(x)$ is directly under the scope of $\rightarrow$ (i.e., $\textbf{boxer}(x)$ is the premise of $\rightarrow$); (c) $\forall x$ directly scopes over $\wedge$; (d) **and** directly scopes over $\textbf{rock-band}(y)$; (e) the conclusion of $\rightarrow$ scopes over $\textbf{love}(x, y)$; (f) the argument of **and** besides $\textbf{rock-band}(y)$ scopes over $\textbf{love}(x, y)$. Finally, the underspecified formula should encode that either $\forall x$ or $\exists y$ scopes over the rest of the operators. In order to construct the underspecified formula, we encode every operator with holes $(h_0, h_1, h_2, \ldots,)$ and attach to every expression a label $(l_0, l_1, l_2, \ldots,)$. By h_0, one denotes the hole such that the label plugged into h_0 will receive widest scope.

$$\begin{aligned}\phi_u = \exists h_0\ \exists h_1\ \exists h_2\ \exists h_3\ \exists h_4\ \exists h_5\ \exists h_6\ \exists l_1\ \exists l_2\ \exists l_3\ \exists l_4\ \exists l_5\ \exists l_6\ \exists l_7 \\ (l_1 : \forall x\, h_1\ \wedge\ l_2 : h_2 \rightarrow h_3\ \wedge\ l_3 : \textbf{boxer}(x)\ \wedge\ l_4 : \exists y\, h_4\ \wedge\ l_5 : \textbf{and}(h_5, h_6) \wedge \\ l_6 : \textbf{rock-band}(y)\ \wedge\ l_7 : \textbf{love}(x, y)\ \wedge\ l_2 \leq h_1\ \wedge\ h_1 \leq l_2\ \wedge\ l_3 \leq h_2\ \wedge\ h_2 \leq h_3 \\ \wedge\ l_5 \leq h_4\ \wedge\ h_4 \leq l_5\ \wedge\ l_7 \leq h_3\ \wedge\ l_7 \leq h_6 \wedge\ l_1 \leq h_0 \wedge\ l_4 \leq h_0) \quad (4.51)\end{aligned}$$

In Equation (4.51), the constraints encode the following:

- $l_2 \leq h_1 \wedge h_1 \leq l_2$ models that $\rightarrow$ directly scopes over $\textbf{boxer}(x)$;
- $l_3 \leq h_2 \wedge h_2 \leq l_3$ models that $\rightarrow$ directly scopes over $\textbf{boxer}(x)$;
- $l_5 \leq h_4 \wedge h_4 \leq l_5$ models that $\exists y$ directly scopes over **and**;
- $l_7 \leq h_6$ models that the h_6 hole of **and** scopes over $\textbf{rock-band}(y)$;
- $l_7 \leq h_3$ models that the conclusion of $\rightarrow$ scopes over $\textbf{rock-band}(y)$;
- $l_1 \leq h_0 \wedge l_4 \leq h_0$ that both $\forall x$ nor $\exists y$ are outscoped by h_0.

Thus, in every possible plugging, one has: $h_1 = l_2, h_2 = l_3, h_4 = l_5$. The rest of holes and labels can be plugged in two ways. Indeed, we can set $h_0 = l_1$, then we obtain that $h_6 = l_7$ and $h_3 = l_4$. In that case, ϕ_u (see Equation (4.51)) encodes the formula ϕ_1 (see Equation (4.49)). Another plugging can be obtained by setting $h_0 = l_4$, then $h_3 = l_7$ and $h_6 = l_1$. In that case, ϕ_u encodes the formula ϕ_2 (see Equation 4.49). In this way, two possible pluggings encode two particular meanings.

4.3.2.2 Discourse Representation

The logical form of a discourse involves the logical representation of content of propositions together with the *hierarchical structure* of the discourse imposed by rhetorical relations. In particular, if between two discourse constituents π_1 and π_2 holds a rhetorical relation R, then $R(\pi_1, \pi_2)$ is a discourse constituent that is ***higher*** in the hierarchy of discourse constituents than π_1 and π_2. In this way, one obtains a hierarchical structure

of discourse. While some rhetorical relations are subordinating (e.g. ELABORATION), others (e.g. NARRATION) are not. As we already saw, one may represent such a discourse structure as a two-dimensional directed acyclic graph as follows: A relation is displayed vertically if it is subordinating, otherwise it is depicted as a horizontal line (see e.g. Figure 4.14 on page 122). To encode the logical form of a discourse, SDRT defines the SDRS language.

4.3.2.3 DRT

One represents contents of discourse units and rhetorical connections within SDRSs, where the notion of an SDRS extends the notion of DRS of DRT (Kamp, 1981, 1988; Kamp and Reyle, 1993; Kamp, van Genabith, and Reyle, 2011). The definition of an SDRS incorporates the notion of a DRS.[36] In particular, to represent atomic discourse units, SDRT makes use of their DRS representations. Thus, at the clause-level, the SDRS language of SDRT coincides with the DRS language of DRT.

However, only having DRSs and their semantic interpretations is not sufficient for SDRT as DRT does not take into account the rhetorical structure of a discourse. SDRT extends DRT by incorporating the *discourse structure* in the logical form of a discourse.

We provide the definitions of the syntax and semantics of the basic fragment of DRT. Afterwards, we discuss the way one extends DRT to SDRT.

4.3.2.3.1 The DRS Syntax

One represents a DRS K as a pair $\langle U, C\rangle$, where U is a set of discourse referents of K and C is a set of conditions in K.

Definition 4.3.5 (The Syntax of DRSs (Lascarides and Asher, 2007)).
The set of DRSs is defined recursively as follows:

$$K := \langle U, \emptyset\rangle \mid K \oplus \langle \emptyset, \gamma\rangle$$

Where:

1. U *is a set of discourse referents.*
2. γ *is a DRS-condition, defined as follows: If* $x_1, \ldots, x_n$ *are discourse referents and* R *is an n-place predicate symbol, then* $\gamma := R(x_1, \ldots, x_n) \mid \neg K \mid K_1 \Rightarrow K_2$.
3. $\oplus$ *is an* append *operation*[37] *on DRSs, defined as follows: If* $K_1 = \langle U_1, C_1\rangle$ *and* $K_1 = \langle U_2, C_2\rangle$, *then* $K_1 \oplus K_2 = \langle U_1 \cup U_2, C_1 \cup C_2\rangle$.

An informal way of writing a DRS is a box divided into two different sub-boxes. One of the boxes contains discourse referents that a DRS introduces, whereas the other one contains conditions of a DRS.

[36] While the notion an SDRS originates from the notion a DRS of DRT, to define an SDRS, one can rely on some dynamic theory other than DRT. However, in this thesis, we follow the version of SDRT where the notion of an SDRS incorporates the notion of a DRS.

[37] Unlike the original DRT, the SDRT uses the operation *append*, which is non-commutative, and therefore the DRS-conditions make a list rather than a set.

Example 4.1.

Let us illustrate the notion of a DRS on the following example:

(52) John drives a car.

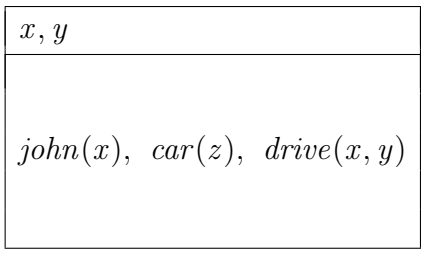


Figure 4.9: An example of a DRS

One represents the content of the sentence (52) as a DRS shown in Figure 4.9. The variables x and y stand for the discourse referents for ***John*** and ***a car***, respectively. Apart from the discourse referents, it contains three conditions, two of them encode that x is a discourse referent introduced by ***John*** and y is a one introduced by ***a car***. The remaining one encodes that x ***drives*** y.

Example 4.2.

Let us consider the following discourse, whose DRS representation is shown in Figure 4.10.

(53) a. Pedro owns a donkey.

b. He beats it.

K and N denote the DRSs corresponding to (53)(a) and (53)(b) respectively. By appending N to K, one obtains the DRS R (see Figure 4.10). However, this example of merging gives a felicitous result because we append ***compatible*** DRSs. Indeed, K and N describe the pieces of discourse whose combination yields a coherent discourse. The discourse referents z and u from N can ***resolve*** to the discourse referents in K, because K is ***accessible*** to N.

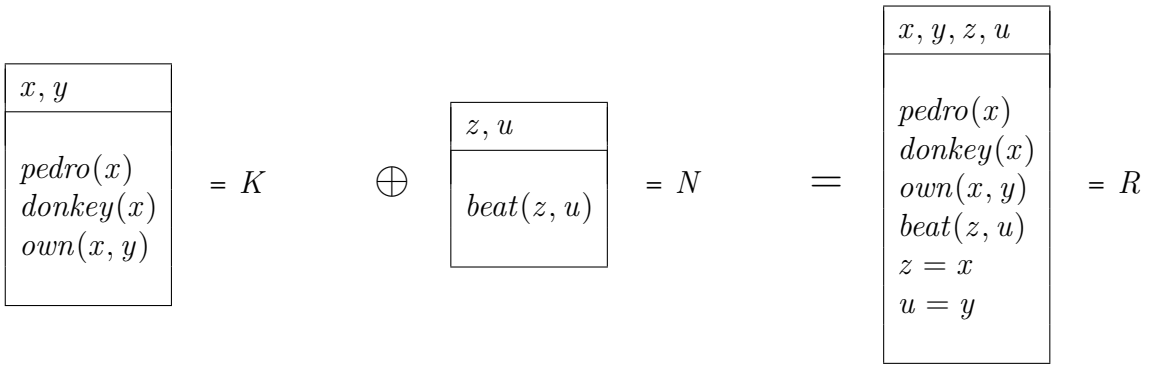
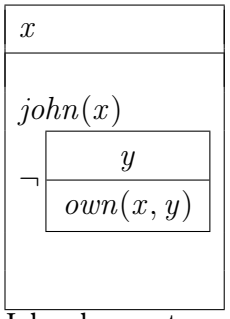
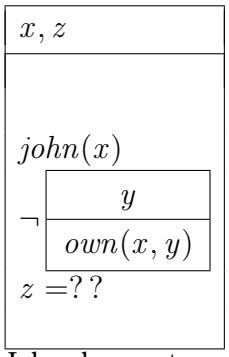


Figure 4.10: An example of the DRS merging

(54) a. John does not own a car.
b. ??It is red.



(a) John does not own a car



(b) John does not own a car. It is red

Figure 4.11: An example of the content inaccessible from the outside the box

Let us consider Example (54). The proposition (54)(b) is not compatible with the rest of discourse, which consists of a single clause (54)(a). The problem is that *it* from (54)(b) cannot resolve to *a car* in (54)(a). DRT gives an account of such cases by introducing the *accessibility constraints* on discourse referents. To formulate the accessibility constraints of DRT, one defines the *subordination* relation over DRSs.

Definition 4.3.6 (Immediate Subordination & Subordination).
We say that K_1 immediately subordinates K_2 if one of the following conditions is met:

1. *K_1 contains a condition of the form $\neg K_2$;*
2. *there is a DRS K_3 such that either $K_3 \Rightarrow K_2$ or $K_2 \Rightarrow K_3$ is contained as a condition in K_1.*

The subordination relation over DRSs is a reflexive, transitive closure of the relation of immediate subordination, and it is denoted by $\leqslant$.

Definition 4.3.7 (Accessibility).
A discourse referent x introduced in U_{K_1} is accessible to an anaphoric DRS condition in K_2 if and only if one of the following conditions holds:

1. $K_2 \leqslant K_1$;
2. *there exists such a DRS* K_3 *that* $K_2 \leqslant K_3$ *and* $K_1 \Rightarrow K_3$.

Now, we explain why in (54), a discourse referent z cannot resolve to a discourse referent y. The DRS introducing y is subordinated to the DRS introducing z. This means that z is accessible to y, but not vice versa. Therefore, z cannot resolve to y.

4.3.2.3.2 Dynamic Semantics of DRSs

A dynamic interpretation of a proposition is a *relation* between variable assignment functions. Under a *dynamic proposition*, we mean a dynamic interpretation of a proposition. The main insight of dynamic semantics is that dynamic propositions may change a *context* in which they appear. Here, the notion of a context is realized as a *variable assignment function* f, which is a mapping from discourse referents to domain entities in some model M. The intuition behind viewing a dynamic proposition as a relation over assignments is that a dynamic proposition may introduce new discourse referents in a context. In order to evaluate the new discourse referents introduced by the dynamic proposition, one *extends* the current variable assignment function. The extended variable assignment function is defined for those discourse referents as well on which the original one is not defined. In this way, the dynamic proposition *relates* the initial variable assignment function and the new one. To put it another way, a dynamic proposition *changes* (updates) the context. In a case where a dynamic proposition does not introduce any new referent in a context, it introduces conditions that have to be fulfilled. In that case, a dynamic proposition behaves as a *test* on a context. More formally, if f is a variable assignment function and U_K is a set of discourse referents of a DRS K, then f may be not defined on the elements of U_K. Therefore, one extends f to g (we write it as $f \subset g$) so that g is defined on U_K as well ($dom(g) = dom(f) \cup U_K$ and $\forall x \in dom(f) \ : f(x) = g(x)$). However, extending a variable assignment function does not define when a DRS is *true* in a model. In other words, the notion of truth of a DRS cannot be defined only by interpreting variables. By interpreting *conditions* of a DRS, one is able to define the notion of a true DRS in a model. Let us denote by C_K the set of conditions of a DRS K. The DRS K is verified by a pair of variable assignment functions f, g, where $f \subset g$, if and only if for every condition $\gamma \in C_K$, g satisfies γ in M (that is, the variable assignment function g maps the discourse referents of U_K to the domain elements in M so that the interpretation of γ in M becomes true). Hence, the DRS K is a relation on variable assignment functions such that f and g are related via K if and only if g extends f on U_K and g verifies all the conditions from C_K.

Let us formally define the semantics of a DRS K in a model M, denoted by $[\![K]\!]_M$, by following (Asher and Lascarides, 2003).

Definition 4.3.8 (Model).
M is a first order model $M = \langle A_M, I_M \rangle$, where:

- *A_M is a set of individuals.*
- *I_M is an interpretation function: I_M assigns to an n-ary predicate P_n, a set of n-tuples of the elements of A_M (we denote it as $I_M(P_n)$).*

Definition 4.3.9 (Interpretation of DRSs).

1. $f[\![\langle U, \emptyset\rangle]\!]_M g$ *if and only if* $f \subset g$ *and* $dom(g) = dom(f) \cup U$.
2. $f[\![K \oplus \langle \emptyset, \gamma\rangle]\!]_M g$ *if and only if there exists* h *such that* $f[\![K]\!]_M h$ *and* $h[\![\gamma]\!]_M g$. *We can write this as a relational composition* $f[\![K]\!]_M \circ [\![\gamma]\!]_M g$.
3. $f[\![R(x_1, \ldots, x_n)]\!]_M g$ *if and only if* $f = g$ *and* $\langle f(x_1), \ldots, f(x_n)\rangle \in I_M(R)$.
4. $f[\![\neg K]\!]_M g$ *if and only if* $f = g$ *and there is no* h *such that* $f[\![K]\!]_M h$.
5. $f[\![K_1 \Rightarrow K_2]\!]_M g$ *if and only if* $f = g$ *and for every* h *such that* $f[\![K_1]\!]_M h$, *there exists some function* k *such that* $h[\![K_2]\!]_M k$.

We drop the subscript M in $[\![K]\!]_M$ whenever it does not create a confusion.

4.3.2.4 The SDRS Language

In SDRT, one extends the language of DRSs in order to incorporate rhetorical connections between discourse constituents in the logical form of a discourse. For that, SDRT defines the notion of SDRS.

Definition 4.3.10 (SDRS and Well-formed SDRS (Asher and Lascarides, 2003)).
One constructs well-formed SDRS-formulas from the following vocabulary:

- *Microstructure: A set* Ψ *of the DRS representations of atomic natural language clauses.*
- *Labels:* $\pi, \pi_1, \pi_2, \cdots$.
- *A set of relational symbols for discourse relations (rhetorical relations):* $R, R_1, R_2, \cdots$.

The set of well-formed SDRS-formulas Φ *contains* Ψ *as a subset. Besides the elements of* Ψ, *it contains formulas defined as follows:*

1. *If* R *is an* n*-place relational symbol and* $\pi_1, \ldots, \pi_n$ *are labels, then* $R(\pi_1, \ldots, \pi_n) \in \Phi$.
2. *If* $\phi, \psi \in \Phi$, *then* $\phi \wedge \psi \in \Phi$ *and* $\neg\phi \in \Phi$, *where* $\wedge$ *and* $\neg$ *are interpreted dynamically, as it is in Definition 4.3.9 (*$f[\![\phi \wedge \psi]\!] g$ *if and only if* $f[\![\phi]\!] \circ [\![\psi]\!] g$*).*

Definition 4.3.11 (Discourse Structure (Asher and Lascarides, 2003)).
An SDRS or a discourse structure is represented by a triple $\langle \mathcal{A}, \mathcal{F}, LAST\rangle$, *where:*

- $\mathcal{A}$ *is a set of speech act discourse referents (labels);*
- $LAST$ *is a member of* $\mathcal{A}$ *(the label of the content of the last clause that was added to the logical form);*
- $\mathcal{F}$ *is a function which assigns each member of* $\mathcal{A}$ *a well-formed SDRS-formula.*

In addition, one defines a relation $i\_outscope$ *on* $\mathcal{A}$ *as follows:* $i\_outscope(\pi, \pi')$ *holds if there is some relation* R *such that either* $R(\pi', \pi'')$ *or* $R(\pi'', \pi')$ *is included as a conjunct in the formula* $\mathcal{F}(\pi)$. *The transitive closure of* $i\_outscope$, *denoted by* $outscope$, *is a partial order over* $\mathcal{A}$. *The requirement upon* $outscope$ *is that there exists the unique supremum* π_0 *of* $outscope$ *in* $\mathcal{A}$.

Example 4.3. Let us give an example of SDRSs by considering the structure of the discourse (46), repeated below.

(46, repeated) π_1. Max had a great evening last night.

π_2. He had a great meal.

π_3. He ate salmon.

π_4. He devoured lots of roquefort.

π_5. He then won a dancing competition

For π_i, where $i = 1, 2, 3, 4, 5$, K_{π_i} denotes a DRS that describes the content of the proposition π_i. One obtains the discourse structure corresponding to (46) by defining an SDRS $\langle \mathcal{A}, \mathcal{F}, LAST \rangle$ as it is shown in Figure 4.12.

$$
\begin{aligned}
&\mathcal{F}(\pi_1) = K_{\pi_1}\\
&\mathcal{F}(\pi_2) = K_{\pi_2}\\
&\mathcal{F}(\pi_3) = K_{\pi_3}\\
&\mathcal{F}(\pi_4) = K_{\pi_4}\\
&\mathcal{F}(\pi_5) = K_{\pi_5}\\
&\mathcal{F}(\pi_0) = \textsc{Elaboration}(\pi_1, \pi_6)\\
&\mathcal{F}(\pi_6) = \textsc{Narration}(\pi_2, \pi_5) \wedge \textsc{Elaboration}(\pi_2, \pi_7)\\
&\mathcal{F}(\pi_7) = \textsc{Elaboration}(\pi_3, \pi_4)\\
&LAST = \pi_5
\end{aligned}
$$

Figure 4.12: An SDRS

Figure 4.13 on the following page illustrates a box-style representation of SDRSs. This representation is a visualization of what the labeling function $\mathcal{F}$ and *ouscope* relation do:

- $\mathcal{F}$ assigns to every label π_i a formula $\mathcal{F}(\pi_i)$.
- The visualization of *ouscope* is the hierarchical embedding of boxes.

In addition, one can represent the SDRS describing (46) as a graph in Figure 4.14, which one builds as follows:

1. If $i\_outscope(\pi_1, \pi_2)$, then π_1 is π_2 are connected with a vertical line, where π_2 is above π_1;
2. if there exists some π for which $F_\pi = R(\pi_1, \pi_2)$ and R is subordinating, then a downward vertical arrow labeled with R comes from π_2 to π_1;
3. if $F_\pi = R(\pi_1, \pi_2)$ and R is not subordinating, then a horizontal arrow connects π_1 and π_2, where π_1 is on the left of π_2.

In the cases where $LAST$ is not necessary to specify, one omits it by writing $\langle \mathcal{A}, \mathcal{F} \rangle$ as a representation of a discourse structure. By convention, for any label π, one denotes $\mathcal{F}(\pi)$ with K_π. Thus, K_π is a condition with label π, which one also may write as $\pi : K_\pi$.

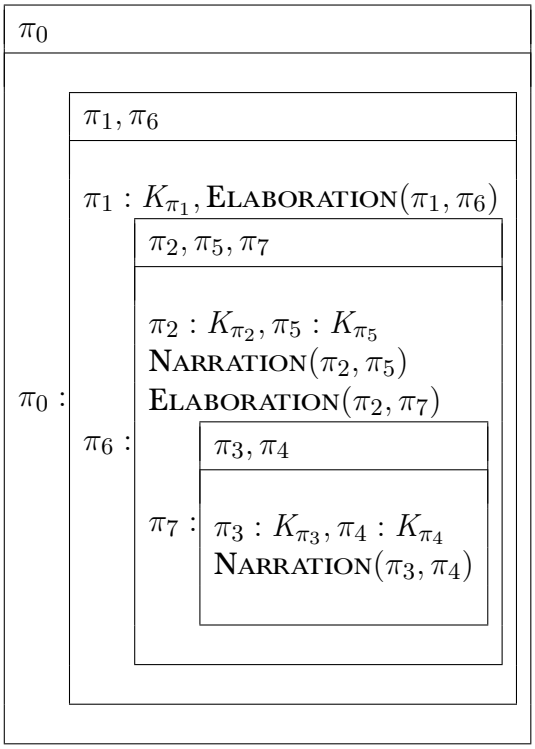


Figure 4.13: The box-style representation of an SDRS

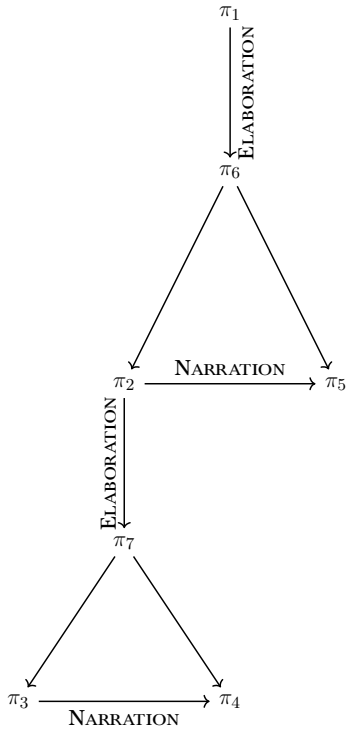


Figure 4.14: A DAG representation of an SDRS

4.3.2.5 Availability

To update a discourse with a new piece, its content must be related to some information in a current discourse. However, by requiring only that, one may obtain an incoherent discourse out of a coherent one. For instance, even though the discourse (46) on page (111) contains some information about *salmon*, incrementing (46) with a clause *the salmon was pink*, which has certainly something in common with *salmon*, results in an incoherent discourse. Hence, a new piece must attach to a current one at a point which is *available for attachment.* As we saw, the accessibility constraints of DRT can be useful only in the cases where one can establish the subordination between DRSs. In the cases such as (46), where the accessibility constraints of DRT do not apply, the RFC might be helpful to make a correct analysis. SDRT extends both the RFC and the DRT accessibility constraints by defining formally what are *available attachment points* and what are possible *antecedents to anaphoric conditions.*

In order to define what are available points for attachment, SDRT dichotomizes discourse relations by assuming that a discourse relation is either *subordinating* or *coordinating.* The following lists[38] provide examples of subordinating and coordinating relations:

Coordinating: Narration, Background, Result, Continuation, Parallel, Contrast, Correction.

Subordinating: Elaboration, Explanation, Precondition, Topic, Commentary, Consequence.

Definition 4.3.12 (Available Attachment Points).
If β is going to attach to a constituent[39] in the SDRS $\langle \mathcal{A}, \mathcal{F}, LAST \rangle$, then one identifies the available attachment points where β can attach to as follows:

1. *The label $\alpha = LAST$.*
2. *Any label γ for which:*
   (a) *There are such δ and R that either $R(\alpha, \delta)$ or $R(\delta, \alpha)$ is a conjunct of the formula labeled with $\mathcal{F}(\gamma)$ (that is, $i\_outscopes(\gamma, \alpha)$ holds).*
   (b) *There exist a subordinating relation R and a label λ such that $R(\gamma, \alpha)$ is a conjunct of the formula labeled with $\mathcal{F}(\lambda)$. This property is abbreviated as $\alpha < \gamma$.*
3. *Any label γ dominating α in the following sense: there exist $\gamma_1, \ldots, \gamma_n$ such that $\alpha < \gamma_1$, $\gamma_1 < \gamma_2$, $\cdots$, $\gamma_n < \gamma$.*

Besides subordinating and coordinating relations, SDRT defines *structural relations.* A relation is structural if it imposes constraints on the propositional structure of its arguments. Thus, given that $R(\pi_1, \pi_2)$ holds, where R is a structural relation, then the contents labeled by π_1 and π_2 have certain propositional structure. For instance, Contrast is a structural relation because if Contrast(π_1, π_2) holds then the pieces whose contents are labeled with π_1 and π_2 are structurally similar (and semantically dissimilar) to each other. An example of a relation that is not structural is Explanation.

[38] For a detailed discussion of the differences between subordinating and coordinating relations, we refer readers to (Asher and Lascarides, 2003; Asher and Laure Vieu, 2005).

[39] A constituent corresponds to a node in the graph representation of an SDRS.

> ··· structural discourse relations differ from other relations, in that they allow discourse referents that are introduced in embedded DRSS to be available, subject to certain constraints. Asher and Lascarides (2003)

Since structural relations demonstrate their distinctive nature from other relations when it comes to availability of discourse referents, the definition that concerns antecedents of anaphora does not apply in a case of structural relations.

Definition 4.3.13 (Antecedents to Anaphora).
If β is the label of a DRS K_β containing an anaphoric condition ϕ, then the available antecedents *for ϕ are the discourse referents that are:*

1. *In K_β and DRS-accessible to ϕ;*
2. *in K_α, DRS-accessible to any condition in K_α (in this case we say it is DRS-accessible in K_α), and there is a condition $R(\alpha, \gamma)$ in the SDRS such that either $\gamma = \beta$ or $outscope(\gamma, \beta)$ holds (R stands for any rhetorical relation except for structural ones).*

Let us illustrate on an example how Definition 4.3.13 allows/prohibits an anaphoric condition to resolve.

Example 4.4.

(55) π_1. John drives a car.
π_2. It is red.

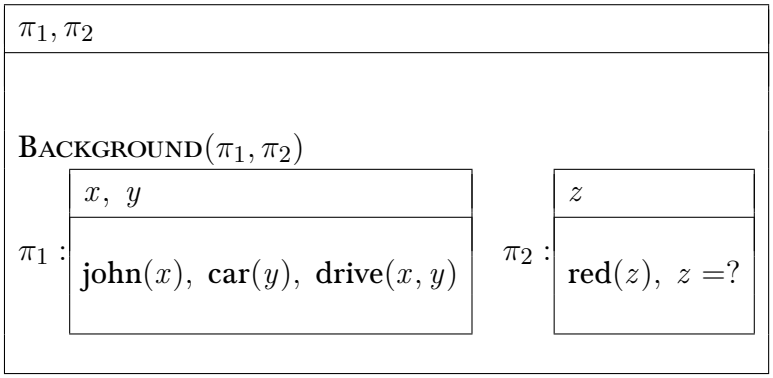


Figure 4.15: The SDRT analysis of *John drives a car. It is red.*

Figure 4.15 shows the SDRS of the discourse (55). Finding an antecedent of *it* from π_2 amounts resolving the anaphoric condition $z =?$, where one uses the symbol $?$ to encode that z is to be resolved to some value. Let us show that the underspecified condition $z =?$ can resolve to y. Indeed, π_2 attaches to π_1 with a discourse relation BACKGROUND. Thus, the condition 2 of Definition 4.3.13 is fulfilled as we have BACKGROUND(π_1, π_2) (π_1 is α, π_2 is β, i.e. $\gamma = \beta$). Since y is DRS-accessible in K_{π_1}, y is accessible to the condition $z =?$ as well. Thus, $z =?$ can resolve to y. In the SDRS shown in Figure 4.15, one can replace the underspecified condition $z =?$ with $z = y$, and thereby obtain a fully specified SDRS.

4.3.2.6 Dynamic Semantics of SDRSs

Since an SDRS describing a single discourse unit is a DRS by definition, we refer to the DRT interpretations of DRSs in order to interpret that SDRS. However, an SDRS can describe more than an atomic discourse unit. Therefore, it becomes necessary to define dynamic semantics for the SDRS language. In particular, since SDRT represents the discourse (rhetorical) connections inside SDRSs, to interpret an SDRS, it becomes necessary to provide dynamic interpretations of rhetorical connections, which involve rhetorical relations and discourse units. For example, if an SDRS is of the form $R(\pi_1, \pi_2)$, where R is a rhetorical relation, then the problem of its interpretation goes beyond the scopes of DRT. Although $R(\pi_1, \pi_2)$ is an atomic formula, its interpretation is more complex than the interpretations of atomic formulas representing meanings of clauses. Taking into account the distinctive properties that we encounter among rhetorical relations, the interpretation of $R(\pi_1, \pi_2)$ should reflect (be in concordance with) the properties of R. For instance, if R is EXPLANATION, then the SDRT satisfiability conditions of the formula EXPLANATION(π_1, π_2) is different from the one when R is CORRECTION.[40] Intuitively it is clear that if EXPLANATION(π_1, π_2) holds, then both K_{π_1} and K_{π_2} hold. In contrast, if CORRECTION(π_1, π_2) holds, then K_{π_1} and K_{π_2} both cannot hold, because they are incompatible. Below, we only provide interpretations of the discourse relations that share the property that EXPLANATION has and CORRECTION does not. We call them *veridical* relations. Veridical relations typically occur in narratives and expository texts. One formulates the property that defines veridicality of a relation as follows:

Definition 4.3.14 (Satisfaction Schema of Veridical Relations (Asher and Lascarides, 2003)).
For a veridical relation R the following holds:

$$f[\![R(\pi_1, \pi_2)]\!]g \qquad \textit{if and only if} \qquad f[\![K_{\pi_1} \wedge K_{\pi_2} \wedge \phi_{\mathsf{R}(\pi_1,\pi_2)}]\!]g$$

Where

- $\wedge$ *is the dynamic conjunction (*$f[\![\phi \wedge \psi]\!]g$ *if and only if* $f[\![\phi]\!] \circ [\![\psi]\!]g$*);*
- $\phi_{\mathsf{R}(\pi_1,\pi_2)}$ *expresses the semantic constraints characteristic to the particular discourse relation* $R(\pi_1, \pi_2)$.

Veridical relations include NARRATION, EXPLANATION, ELABORATION, BACKGROUND, CONTRAST, PARALLEL, etc. The rhetorical relation CORRECTION is not veridical because $f[\![R(\pi_1, \pi_2)]\!]f$ implies that $f[\![\neg K_{\pi_1}]\!]g$ (the dynamic negation of K_{π_1}).

Thus, $R(\pi_1, \pi_2)$ holds if and only if K_{π_1}, K_{π_2}, and $\phi_{\mathsf{R}(\pi_1,\pi_2)}$ hold, where $\phi_{\mathsf{R}(\pi_1,\pi_2)}$ depends on $R(\pi_1, \pi_2)$. As one see from the satisfaction schema for veridical rhetorical relations, they are complex discourse *updates*. That is, one can view $\phi_{\mathsf{R}(\pi_1,\pi_2)}$ as a

[40] For example, CORRECTION is a link between the first and second utterances in the following discourse:

π_1. John is a journalist.

π_2. No, he is a sailor.

condition that constrains the way the dynamic proposition $[\![R(\pi_1, \pi_2)]\!]$ updates a context. Thus, together with the satisfaction schema, one has predefined conditions, i.e., some sort of axioms for rhetorical relations. In SDRT, one refers to these axioms as *meaning postulates*. Let us illustrate meaning postulates of rhetorical relations by discussing the meaning postulates of EXPLANATION and ELABORATION.

ELABORATION & EXPLANATION

(56) Alexis did well in school this year. She got As in every subject.

(57) Max fell. John pushed him.

In (56), *she got As in every subject* elaborates *Alexis did well in school this year.* Hence, the rhetorical link between them is ELABORATION. In the case of (57), the second proposition explains the first one, that is, EXPLANATION connects them. These rhetorical relations give rise to different effects on the events of the propositions they link. Indeed, as (56) shows, we have the *temporal inclusion* between the events expressed by the first and second propositions, which is an effect of ELABORATION. In (57), the event of the explained proposition (*Max fell*) takes place after the event that explains it (*John pushed him*). Thus, in each of the discourse (56) and (57), the two events are linked with *temporal precedence.* These observations lead to the following meaning postulates of SDRT:

Temporal Consequence

EXPLANATION :

$$\phi_{\text{Explanation}(\alpha,\beta)} \Rightarrow (\neg e_\beta \prec e_\alpha) \tag{4.58}$$

ELABORATION :

$$\phi_{\text{Elaboration}(\alpha,\beta)} \Rightarrow \textit{Part-of}(e_\beta, e_\alpha) \tag{4.59}$$

Where $\prec$ encodes a relation on events in terms of the temporal order; *Part-of* is a sub-event relation.

It is noteworthy that to determine those meaning postulate that apply in a particular case depends not only on the involved rhetorical relation but on the propositions that are related with that rhetorical relation.

Example 4.5. Let us illustrate the way SDRT makes use of meaning postulates and dynamic semantics in order to establish truth conditions under which a discourse is felicitous. We consider the following discourse:

(60) a. Max fell.

b. John pushed him.

Figure 4.16 shows the SDRS representation of (6o). π_1 (resp. π_2) labels the DRS K_{π_1} (resp. K_{π_2}), which represents the logical form of the clause (6o)(a) (resp. (6o)(b)). We use predicate ***holds_at*** to express that an event e_π takes place at the t_π time and write it as ***holds-at***(e_π, t_π). The label π_0 is the ***highest*** label as it labels $\textsc{Explanation}(\pi_1, \pi_2)$, i.e., the rhetorical connection of K_{π_1} and K_{π_2} provided by the rhetorical relation Explanation. Since Explanation connecting π_1 and π_2 is veridical, to interpret the discourse (6o), that is, to interpret the SDRS in Figure 4.16, one refers to the satisfaction schema for veridical relations. According to it (see Definition 4.3.14), one obtains the following:

$$f[\![\textsc{Explanation}(\pi_1, \pi_2)]\!]_M\, g \quad \text{if and only if} \quad f[\![K_{\pi_1} \wedge K_{\pi_2} \wedge \phi_{\textsc{Explanation}(\pi_1,\pi_2)}]\!]_M\, g$$

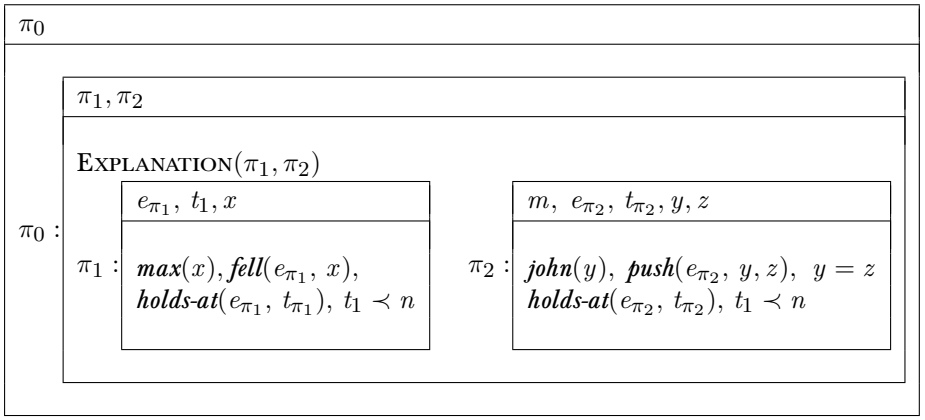


Figure 4.16: An SDRS

By interpreting the right-hand side of the last statement, that is, $f[\![K_{\pi_1} \wedge K_{\pi_2} \wedge \phi_{\textsc{Explanation}(\pi_1,\pi_2)}]\!]_M\, g$, one obtains the interpretation of the discourse. To interpret that, we refer to the interpretation of the dynamic conjunction $\wedge$. According to the interpretation of dynamic $\wedge$, there are two variable assignment functions h and k such that $f[\![K_{\pi_1}]\!]_M\, h$, $h[\![K_{\pi_2}]\!]_M\, k$, and $k[\![\phi_{\textsc{Explanation}(\pi_1,\pi_2)}]\!]_M\, g$ hold. Let us consider each of them separately.

- $f[\![K_{\pi_1}]\!]_M\, h$ holds if $dom(h) = dom(f) \cup \{e_{\pi_1}, x, t_{\pi_1}\}$ and $\langle h(x)\rangle \in I_M(\textbf{\textit{max}})$, $\langle h(e_{\pi_1}), h(x)\rangle \in I_M(\textbf{\textit{fall}})$, $\langle e_{\pi_1}, t_{\pi_1}\rangle \in I_M(\textbf{\textit{holds-at}})$.
- $h[\![K_{\pi_2}]\!]_M\, k$ holds if $dom(k) = dom(h) \cup \{e_{\pi_2}, y, z, t_{\pi_2}\}$ and $\langle h(y)\rangle \in I_M(\textbf{\textit{john}})$, $\langle k(e_{\pi_2}), k(y), k(x)\rangle \in I_M(\textbf{\textit{push}})$, $\langle e_{\pi_2}, t_{\pi_2}\rangle \in I_M(\textbf{\textit{holds-at}})$.
- According to the meaning postulate for Explanation, if $k[\![\phi_{\textsc{Explanation}(\pi_1,\pi_2)}]\!]_M\, g$ holds, then $k[\![\neg e_{\pi_2} \prec e_{\pi_2}]\!]_M\, g$ holds as well. $k[\![\neg e_{\pi_2} \prec e_{\pi_2}]\!]_M\, g$ holds if and only if $k \subset g$, and $\langle \neg e_{\pi_2},\ e_{\pi_1}\rangle \in I_M(\prec)$. Since $\prec$ represents the temporal relation, we obtain that $k[\![\phi_{\textsc{Explanation}(\pi_1,\pi_2)}]\!]_M\, g$ holds only in the case where e_{π_1} ***does not take place before*** e_{π_2}, that is, ***Max's fell*** had not happened before ***John pushed him.***

In this way, we have discussed the SDRS language and the dynamic logic. They allow one to represent a discourse in a logical form and to interpret that logical form, respectively. In the next chapter, we discuss formal grammars of discourse whose discourse structures are inspired by SDRT.

Chapter 5

Discourse Grammar Formalisms

Contents

In this chapter, we discuss formalisms that study discourse regularities with grammars. One refers to such formalisms as discourse grammar formalisms. We focus on D-LTAG, G-TAG, and D-STAG. D-LTAG and D-STAG were introduced for discourse parsing, whereas G-TAG is a formalism for discourse generation. Our choice of these formalisms is determined by the fact that each of them proposes its discourse grammar based on the TAG principles. Since the ACG encoding of TAG coupled with Montague semantics is already available, these formalisms are interesting for one who aims at the modeling of the syntax-discourse interface with ACGs. Although the formalisms D-LTAG, G-TAG, and D-STAG, each proposes a TAG grammar for discourse, each of these discourse grammars is designed under certain assumptions. In G-TAG and D-LTAG, the assumption is that a discourse structure is tree-shaped. D-STAG proposes a grammar that allows for a richer structure of discourse than just tree-shaped ones. Namely, in D-STAG, the discourse structure can be a directed acyclic graph. While the grammars of D-LTAG, G-TAG, and D-STAG prove to be capable of encoding various phenomena, they experience some problems when one considers a discourse where an adverbial connective occupies an internal (clause-medial) position in a clause. To overcome the problem, each of these formalisms develops a two-step approach to discourse processing. One of the steps is the grammatical step. During this step, one applies the grammar of a formalism to produce the derived (parse) tree of a discourse. The other step is an extra-grammatical step. In G-TAG, the extra-grammatical step involves moving discourse adverbials from clause-initial to clause-medial positions so that one generates a text where a discourse adverbial appears at a clause-medial position. In D-LTAG and in D-STAG, during the extra-grammatical step, one moves discourse adverbials from clause-medial positions to the clause-initial ones. This step is necessary in order to parse a discourse with either the D-LTAG grammar or the D-STAG one.

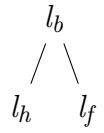
5.1 D-LTAG

Lexicalized Tree Adjoining Grammar for Discourse (D-LTAG) (Forbes et al., 2003; Forbes-Riley, Bonnie Webber, and Aravind Joshi, 2006; B. L. Webber, 2004; B. L. Webber and A. K. Joshi, 1998; Bonnie Webber, Knott, Stone, and Aravind Joshi, 1999) is a formalism for discourse parsing. D-LTAG is based on the (L)TAG principles. But while (L)TAG deals with sentence-level structures, D-LTAG focuses on discourse-level ones. In addition to the discourse parsing, with the help of D-LTAG, one can interpret a discourse as a formula of a labeled language. In particular, one constructs the labeled formula out of the derivation tree of the discourse. The discourse structure[41] encoded by the obtained labeled formula is a tree-shaped one. For instance, in the case of the sentence (61), D-LTAG produces the discourse interpretation shown in Figure 5.1(a).

(61) Sue is happy because she found a job.

$$\boxed{\begin{array}{l} l_b : \textbf{because}'(l_h,\ l_f), \\ l_h : \textbf{happy}(\textbf{sue}), \\ l_f : \textbf{found}(\textbf{sue},\ \textbf{job}). \end{array}}$$

(a) The D-LTAG interpretation of *Sue is happy because she found a job*



(b) The tree representation

Figure 5.1: A D-LTAG interpretation and its tree representation

According to the D-LTAG notations, in a labeled formula (e.g. in Figure 5.1(a)), a comma stands for the logical conjunction $\wedge$. For the sake of simplicity, one denotes a discourse relation (rhetorical relation) signaled by a discourse connective *lex.item* with **lex.item**$'$, unless otherwise stated. l_u denotes a label, where u is a natural number or a symbol of the Latin alphabet.

One can represent the formula in Figure5.1(a) as the tree in Figure 5.1(b), where nodes stand for labels, and the parent-child relation in a tree realizes a predicate-argument relation in a formula.

As we already discussed in Section 4.1 on page 88, according to (Bonnie Webber, Stone, Aravind Joshi, and Knott, 2003), both of the arguments of a subordinate and/or coordinate conjunction are structural, that is, they appear in the parse tree of a discourse. In contrast to them, only one of the arguments of an adverbial connective is structural. The D-LTAG grammar encodes this difference between conjunctions and adverbial connectives. Every conjunction is encoded with two arguments, whereas every adverbial connective is encoded only one argument. The other argument of the adverbial connective, called the *anaphoric* argument, can be found using an extra

[41] In D-LTAG, the term 'discourse structure' refers to a parse tree of a discourse. Nevertheless, we refer to the interpretation of a discourse as the 'discourse structure.'

grammatical mechanism (e.g. anaphora resolution). By retrieving/inferring the anaphoric arguments of the adverbial connectives in a discourse, one may obtain a non tree-shaped interpretation of the discourse out of its D-LTAG interpretation, which is tree-shaped. For instance, let us consider the discourse (62) containing the discourse adverbial *in this way*. Figure 5.2(a) shows the D-LTAG interpretation of the discourse (62) as a labeled formula. In the labeled formula, $[e_i]^{a_c}$ denotes the anaphoric argument; ϵ' models *the empty connective* as there is no overt structural relation between the first and second sentences.[42] The value of the anaphoric argument $[e_i]^{a_c}$ is not specified in the D-LTAG interpretation. As one can see, this labeled formula gives rise to the tree in Figure 5.2(b). However, one can resolve $[e_i]^{a_c}$ to l_i. Hence, one obtains the labeled formula shown in Figure 5.3(a). This formula gives rise to a directed acyclic graph (DAG) depicted in Figure 5.3(b), which is not a tree.

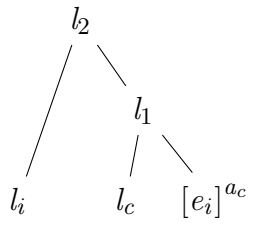
(62) a. The company interviewed everyone.

b. In this way, they considered all their options.

We first discuss D-LTAG elementary trees. Afterwards, we describe the D-LTAG parsing process. We discuss the way one interprets a discourse with the help of D-LTAG.

$$\begin{array}{ll} l_1 & : \textbf{in\_this\_way}'(l_c,\ [e_i]^{a_c}), \\ l_2 & : \epsilon'(l_i,\ l_1), \\ l_i & : \textbf{interview}(\mathbf{c},\ \mathbf{e}), \\ l_c & : \textbf{consider}(\mathbf{c},\ \mathbf{e}) \end{array}$$

(a) The D-LTAG interpretation

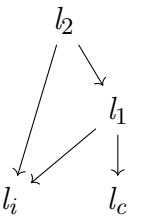


(b) The tree representation

Figure 5.2: The D-LTAG interpretation of discourse

$$\begin{array}{ll} l_1 & : \textbf{in\_this\_way}'(l_4,\ l_3), \\ l_2 & : \epsilon'(l_3,\ l_1), \\ l_3 & : \textbf{interview}(\mathbf{c},\ \mathbf{e}), \\ l_4 & : \textbf{consider}(\mathbf{c},\ \mathbf{o}) \end{array}$$

(a) The fully specified interpretation



(b) The DAG representation

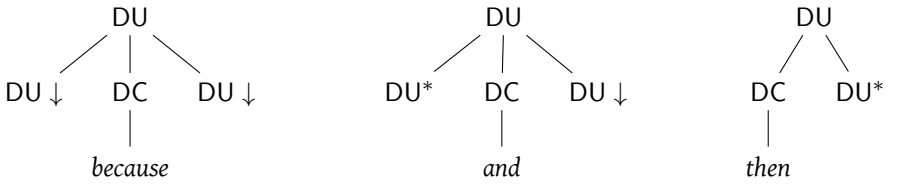
Figure 5.3: The interpretation obtained from the D-LTAG interpretation by resolving an anaphoric link

[42]We discuss this point in more derails in Remark5.2.

5.1.1 D-LTAG Elementary Trees

An elementary tree in D-LTAG is either anchored with a discourse connective or with a clause. A clause C is an atomic unit of discourse and it anchors an initial tree $\overset{\text{DU}}{|}\ C$ (the non-terminal symbol labeling nodes in D-LTAG trees is usually denoted by DU - *discourse unit*).

In D-LTAG, various discourse connectives anchor various kinds of trees. Contrasting encodings of different elementary trees are motivated by differences between the properties of their lexical anchors. One of the main differences that D-LTAG encodes within its grammar is the difference between *structural* and *anaphoric* discourse connectives. Figure 5.4(a) and Figure 5.4(b) show examples of D-LTAG elementary trees anchored with structural connectives, whereas Figure 5.4(c) shows a D-LTAG elementary tree anchored with an adverbial connective.



(a) An DU-initial tree with two DU-substitution sites

(b) A D-LTAG auxiliary tree anchored with a coordinate conjunction

(c) A DU-auxiliary with no substitution sites

Figure 5.4: D-LTAG elementary trees anchored with a subordinate conjunction, a coordinate conjunction, and an adverbial

5.1.2 Structural Connectives

Apart from distinguishing between structural and anaphoric connectives, D-LTAG makes differences among structural connectives. For instance, as Figure 5.4 shows, *because* and *and* anchor an initial tree and an auxiliary one, respectively. Contrasting encodings of different structural connectives is motivated by their *semantico-pragmatic* properties.

5.1.2.1 Initial Trees

A structural connective anchoring an *initial* tree obtains both of its arguments via substitution as it has two DU-substitution sites (see Figure 5.4(a)). By filling these substitution sites with trees, the discourse relation signaled by the discourse connective obtains its arguments.

Various lexical items may anchor D-LTAG initial trees. For instance, as Figure 5.5 shows, subordinate conjunctions, certain coordinate conjunctions, and some imperative verbs anchor DU-initial trees with two DU-substitution sites. Besides these categories, paired connectives (e.g. *on the one hand, on the other hand*), and subordinators (multi-word subordinate conjunctions, e.g. *in order to*) also anchor the DU-initial two DU-substitution

sites. All these linguistic constructions serve as *discourse predicates*, which define the *domain of locality at the discourse-level* (B. L. Webber, 2004). Therefore, they are encoded as initial trees.

Remark 5.1. *The D-LTAG trees anchored with the above listed linguistic constructions (i.e., subordinate conjunctions, certain coordinate conjunctions, multi-word expressions, paired connectives, etc.) differ from the LTAG trees anchored by them. Indeed, at the sentence-level, they are not the ones that define the domain of locality, but verbs, predictive adjective, etc. Since subordinate conjunctions, paired connectives, and other above listed constructions are beyond the domain of locality of verbs, predictive adjective, etc., they are encoded by LTAG auxiliary trees.*

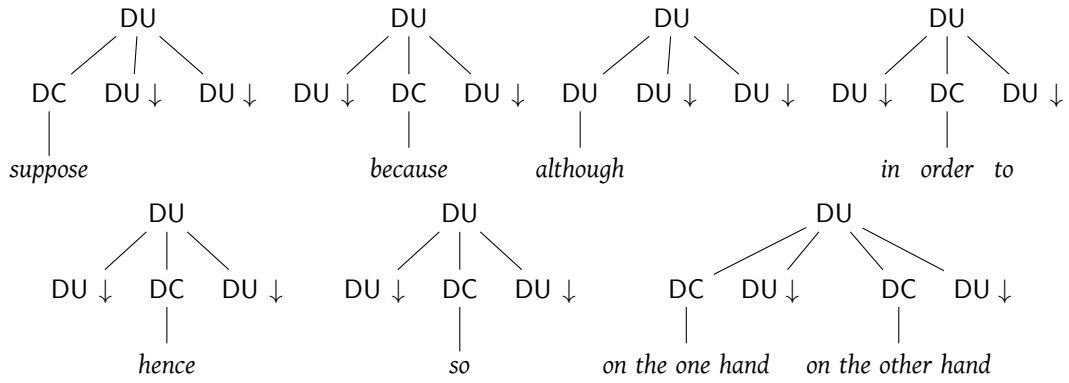


Figure 5.5: D-LTAG initial trees anchored with discourse connectives

5.1.2.2 Auxiliary Trees

While subordinate conjunctions and certain coordinate conjunctions anchor initial trees, some coordinate conjunctions anchor auxiliary ones. For example, *and* anchors an auxiliary tree shown in Figure 5.4(b) on the preceding page. The motivation behind encoding coordinate conjunctions, such as *and*, as anchors of auxiliary trees is that they *extend* the description of a situation or an entity conveyed in a discourse which they attach to.[43] Hence, they provide a *recursive* way of extending the information in a discourse (B. L. Webber, 2004). Therefore, they anchor D-LTAG auxiliary trees. In these auxiliary trees, there is a single substitution site where one substitutes a discourse unit that further elaborates the previous piece of discourse.

In addition to overt markers of discourse connectives, the *lexically unexpressed connective* ϵ, sometimes called the *empty* connective, anchors a D-LTAG auxiliary tree. If two discourse units are *structurally adjacent* to each other, but no lexically expressed discourse connective connects them, D-LTAG assumes that the lexically unexpressed connective ϵ relates these two discourse units. The empty connective ϵ anchors an auxiliary tree with a single DU-substitution site, illustrated in Figure 5.6.

[43]In contrast to *and*, the coordinate conjunction such as *so* and *hence* do not extend the previous piece of discourse, but rather express *result*.

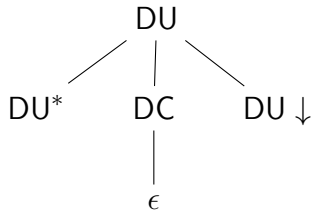


Figure 5.6: The D-LTAG tree anchored with ϵ

5.1.2.3 Anaphoric Connectives

As we already discussed in Section 4.1 on page 88, discourse adverbials are identified with anaphoric connectives. Unlike structural connectives, an anaphoric connective obtains only *one* of the two arguments *structurally*. The other argument is either retrieved anaphorically or inferred. An adverbial connective anchors a D-LTAG auxiliary tree, such as one shown in Figure 5.4(c). By adjoining an auxiliary tree anchored with an adverbial connective into a parse tree of a piece of discourse, the adverbial connective obtains its structural argument. Getting the anaphoric argument is beyond the scopes of D-LTAG.

Convention: We denote a D-LTAG initial (auxiliary) tree anchored with a ***lex.item*** by $\alpha^{D}_{lex.item}$ (resp. $\beta^{D}_{lex.item}$), where the superscript D indicates that it is a D-LTAG initial (res. auxiliary) tree. To denote an LTAG tree anchored with a ***lex.item***, we write $\alpha_{lex.item}$ or $\beta_{lex.item}$, unless otherwise stated.

5.1.3 Discourse Parsing with D-LTAG

Having described the grammar of D-LTAG, we discuss the way such a grammar is used to parse a text. We already mentioned that a clause C anchors an initial tree in D-LTAG. Actually, it is not a clause C that is the anchor of the tree, but rather the LTAG derived (parse) tree of the clause C. That is, instead of $\overset{\text{DU}}{|}{}_{C}$, one has $\overset{\text{DU}}{|}{}_{\gamma}$ where γ is the LTAG derivation tree of the clause C. Thus, the clause-level grammar underlying D-LTAG is an LTAG.

One can use the same TAG parser for both LTAG and D-LTAG (Forbes et al., 2003; B. L. Webber, 2004), as both of grammars are TAG grammars. The general description of the D-LTAG parsing process is the following:

1. An LTAG parser parses each sentence of the input text.
2. The Tree Extractor component extracts the following items from the parse tree of each sentence: LTAG parse trees of clauses and LTAG elementary trees anchored with discourse connectives.
3. The Tree Mapper component applies to the extracted LTAG elementary trees anchored with discourse connectives. It maps them to the corresponding D-LTAG elementary trees (an LTAG elementary tree anchored with a discourse connective may differ from the D-LTAG elementary tree anchored by the same discourse connective).

4. The Discourse Input Generator component generates *the discourse input*: The input text is represented as the sequence (string) of D-LTAG trees anchored by the clauses and the connectives obtained in the previous steps.
5. Using the D-LTAG (discourse) grammar, an LTAG parser parses the sequence produced in the previous step by constructing a derivation tree using the trees in the sequence.

Let us consider some of these steps in more details.

Tree Extractor

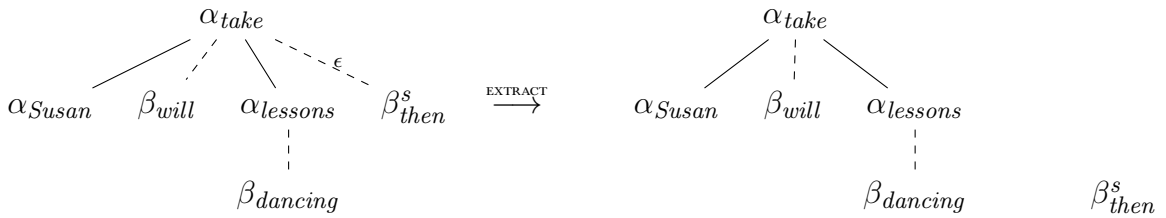
At the first step of parsing, one produces the derivation trees of sentences in the text using an LTAG grammar. The sequence of these derivation trees are the inputs of the Tree Extractor (TE). TE outputs:

1. Elementary trees anchored with *discourse connectives* that appear in the sentences.
2. The derivation tree for each clause in a sentence.

Sometimes, to extract a derivation tree of a clause and identify discourse connectives is relatively straightforward compared to some other cases. In particular, the cases where connectives appear at clause-medial positions require special treatments. Let us consider the following examples:

(63) a. Susan will *then* take dancing lessons.

b. *Then*, Susan will take dancing lessons.



(a) An LTAG derivation tree of a sentence

(b) An LTAG clausal derivation tree

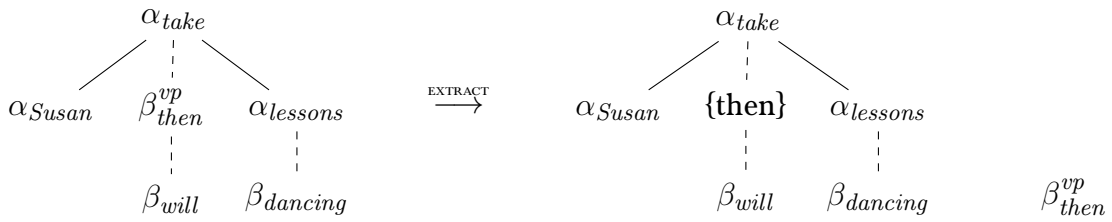
(c) An LTAG auxiliary tree anchored with *then* *then*

Figure 5.7: A case of a discourse with a clause-initial connective

In the sentence (63)(a), *then* is a clause-medial discourse adverbial, whereas in the case of (63)(b), *then* is a clause-initial one. TE extracts from the derivation trees of the sentences (63)(a) and (63)(b), the derivation trees of clauses and elementary trees anchored with discourse adverbials. In the case of (63)(b), the extracted trees (see Figure 5.7) are the derivation tree of the clause *Susan will take dancing lessons* and the single node tree denoting the auxiliary tree anchored with *then* (β^s_{then} - the superscript

s denotes that its is an S-auxiliary tree). It is straightforward to extract these trees as β^{s}_{then} adjoins on the root node of the clausal derivation tree.

In the case of the sentence (63)(a), β_{will} adjoins into β^{vp}_{then} (β^{vp}_{then} - the superscript vp denotes that its is a VP-auxiliary tree). The resultant tree adjoins into α_{take} (see Figure 5.8(a)). We cannot extract the derivation trees of the clause and an elementary tree of the adverbial as straightforwardly as in the case of (63)(b), because if we removed an edge connecting β_{then} and α_{take}, then the result would not be a tree as the node β_{will} would become disconnected from the rest of the nodes. Therefore, both the edge connecting the node α_{take} with β^{vp}_{then} and the node β^{vp}_{then} are removed, the edge connecting the β_{will} and α_{take} has to be reintroduced in the tree. Thus, in the obtained tree, β_{will} and α_{take} are still connected, but the node β^{vp}_{then} is not there anymore. To maintain the information about the original position of a connective,[44] in an extracted tree, one leaves a trace of the connective (the node {then} of the tree in Figure 5.8(b)).



(a) An LTAG derivation tree of a sentence

(b) An LTAG clausal derivation tree

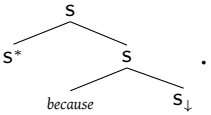
(c) An LTAG auxiliary tree anchored with ***then***

Figure 5.8: A case of a discourse with a clause-medial connective

The cases with subordinate and coordinate conjunctions may involve two clausal derivation trees.[45] For the sake of illustration, let us consider the sentence (61), repeated as follows:

(61, repeated)

Sue is happy because she found a job.

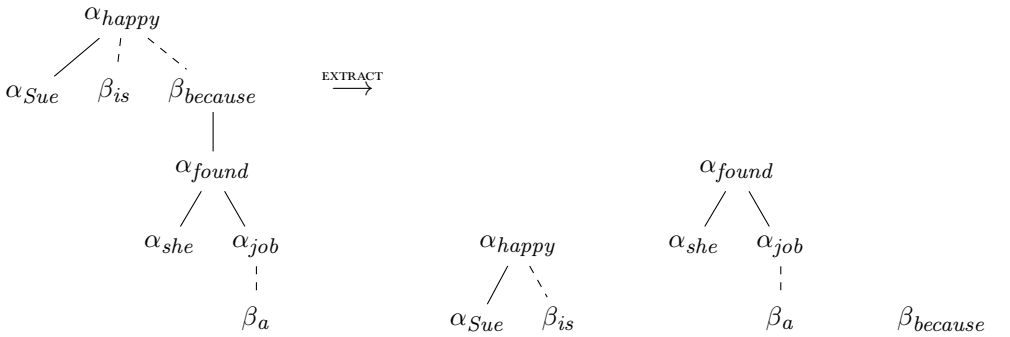
The LTAG derivation tree of the sentence (61) is shown in Figure 5.9(a). One extracts out of it three trees, two clausal derivation trees and a single node tree $\beta_{because}$, which denotes anchored with the subordinate conjunction ***because***, 

. To extract these trees is easier than in the case of clause-medial adverbials. Indeed, by

[44]The clause-medial position of discourse connective is considered to be important from the information structure points of view (Steedman, 2000).

[45]Since we only focus on English and French, one can claim subordinate and coordinate conjunctions can only occupy clause-initial positions.

extracting $\beta_{because}$, one obtains the clausal derivation trees. On one of them adjoins $\beta_{because}$, and the other one substitutes into $\beta_{because}$. Hence, it is not problematic to identify these clausal derivation trees.



(a) An LTAG derivation tree of a sentence

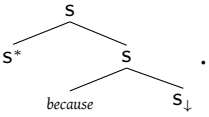
(b) An LTAG clausal derivation tree

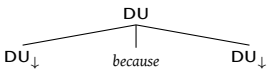
(c) An LTAG clausal derivation tree

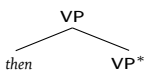
(d) An LTAG auxiliary tree anchored with *because*

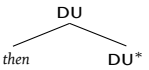
Figure 5.9: A case of a discourse with a subordinate conjunction

Tree Mapping

The Tree Mapping module (TM) applies to the output of TE. TM maps the LTAG elementary trees anchored with the discourse connectives to the D-LTAG elementary trees anchored by these discourse connective. Let us assume that TE extracts an LTAG elementary tree anchored with a subordinate conjunction, for example, the tree anchored by ***because***, $\beta_{because}$ denoting the auxiliary tree 

. TM maps this auxiliary tree to its D-LTAG correspondent, i.e., to $\alpha^{D}_{because} =$ 

. As another example, let us consider β^{vp}_{then} extracted from the derivation tree shown in Figure 5.8(a). β^{vp}_{then} stands for the VP-auxiliary tree 

in LTAG. However, in D-LTAG, ***then*** anchors the following tree $\beta^{D}_{then} =$ 

. TM transforms β^{VP}_{then} into β^{D}_{then}, that is, TM transforms *a clause-medial adverbial* into *a clause-initial one.* This is due to the fact that the D-LTAG encoding of an adverbial connective is only applicable when it appears at a clause-initial position in a discourse.

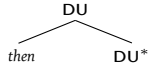
Discourse Input Generation

The component called Discourse Input Generation (DIG) produces an input to a parser out of a *sequence of lexicalized trees.* Each tree in the sequence is either a D-LTAG

elementary tree anchored with a connective (being produced by TM), or a D-LTAG clausal elementary tree, i.e., DU-initial tree anchored with a clausal derivation tree extracted by TE. The trees are ordered according to the surface order of their lexical anchors, except for the clause-medial connectives. They are placed in front of their host clauses. In such a sequence, it may occur that there is no *structural* discourse connective between two clausal elementary trees. In that case, DIG places the D-LTAG auxiliary tree with an empty lexical anchor ϵ between these clausal elementary trees. Thus, one obtains the sequence of D-LTAG elementary trees. One uses a TAG parser to build *a derivation tree of the text* out of this sequence.

While building a derivation tree out the given sequence of D-LTAG trees, one faces *derivational* ambiguity issues. Indeed, there might be various possible sites in a derived tree for a discourse where a derived tree for a new piece can attach. Thus, there are a number of derivation trees which give rise to the same surface forms. D-LTAG reduces the amount of possible attachment points by introducing the following heuristic constraints:

- In an initial tree, it is only allowed to adjoin at the root node.
- In other trees, only the lowest adjunction is allowed.

Remark 5.2. *It may occur that an adverbial discourse connective occurs between two clauses but no structural one. Despite that, DIG inserts ϵ between the two clauses. The need for structural connective in such cases is due to the structure of an auxiliary tree anchored with an adverbial connective. In particular, it cannot connect two discourse units. For example, consider* $\beta^D_{then} =$ 

. The tree β^D_{then} can adjoin into a tree anchored with a clause. However, given two initial trees anchored with clauses, β^D_{then} is not capable to build a new tree which would incorporate the two trees. To be able to combine two derived trees of discourse units, one has to use a structural connective.

5.1.4 Computing Discourse Semantics

To interpret a discourse, one refers to its derivation tree. D-LTAG couples elementary trees with their semantic interpretations so that one can interpret the operations over elementary trees as operations over their interpretations. In this way, D-LTAG develops a compositional approach to the syntax-semantics interface for discourse. We assume that the clausal derived and derivation trees are provided by an LTAG grammar.

D-LTAG employs a labeled language to define interpretations. As we already mentioned, one denotes with **because**$'$, **and**$'$, and ϵ', the discourse relations signaled by *because*, *and*, and ϵ respectively. This notation is useful because the lexically unexpressed connective ϵ a priori does not signal any particular discourse relation. To interpret ϵ' as a particular rhetorical relation, one needs the context where ϵ appears.

We first discuss interpretations of elementary trees (anchored with clauses, structural connectives, and adverbials). Afterwards, we show how one composes the interpretations of the elementary trees in order to obtain the interpretation of a tree derived from them.

Convention: One denotes the LTAG derived tree of a clause whose main predicate is

v with γ_v; by τ_v, we denote the derivation tree of the D-LTAG initial tree $\overset{\text{DU}}{\underset{\gamma_v}{|}}$, unless otherwise stated.

Elementary Trees Anchored with Clauses

Figure 5.10 shows initial trees anchored with the derived trees of the following clauses:

(64) C_1. Sue is happy.
C_2. Sue found a job.
C_3. Sue likes her job.

Under each tree in Figure 5.10, we depict its interpretation. The interpretation of a tree with a clause is the interpretation of that clause. More precisely, given a clause C and its LTAG derived tree γ, the interpretation of the D-LTAG tree $\overset{\text{DU}}{\underset{\gamma}{|}}$ is the interpretation of C. Interpretations of clauses play roles of arguments (operands) to discourse relations. They themselves do not have arguments. To express that, in their interpretations, one declares $\mathit{arg} : -$. One labels formulas with l_i, where i is a natural number or a letter of the Latin alphabet. We may refer to l_i as a ***propositional label.***

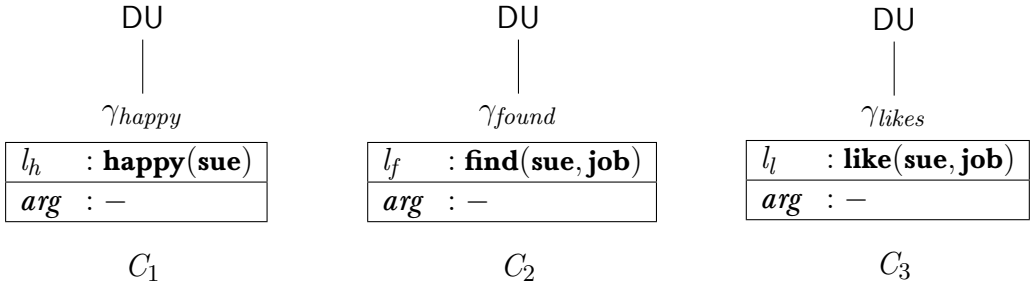


Figure 5.10: Initial trees anchored with clauses coupled with their semantic interpretations

Elementary Trees Anchored with Structural Connectives

If ***lex.item*** is a structural connective, then **lex.item**′ obtains both of the arguments structurally. We mark an argument of **lex.item**′ that corresponds to a substitution site in $\alpha^D_{lex.item}$ with the Gorn address of that substitution site. For instance, as Figure 5.11(a) shows, for $\alpha^D_{because}$, we have the predicate **because**′ whose arguments are s_1 and s_2 marked with (1) and (3), respectively (the latter fact is encoded by the labeled formula $\mathit{arg} : \langle s_1, (1)\rangle, \langle s_2, (3)\rangle$). Thus, **because**′ receives one of its arguments, denoted by s_1, as a result of substitution of a tree at the Gorn address 1 into $\alpha^D_{because}$. **because**′ obtains the other argument, denoted by s_2, as a result of substitution of a tree at the address 3 into $\alpha^D_{because}$. The semantic interpretations of the substituted trees serve as arguments of **because**′.

In the case of a structural connective anchoring an auxiliary tree, such as β^D_{and} (see Figure 5.11(b)), we specify the Gorn address of only one of the arguments - the one that comes from substitution. For example, by substituting a tree into β^D_{and} at the Gorn address 3, **and**$'$ obtains the value of the argument denoted by s_4. The other structural argument of **and**$'$, denoted by s_3, comes from the interpretation of the derived tree into which β^D_{and} adjoins. s_i, where i is a natural number or a symbol of the Latin alphabet, denotes a *label variable*, sometimes called a *propositional label variable*.

Elementary Trees Anchored with Adverbial connectives

Adverbial connectives receive only one of their arguments structurally. Figure 5.11(c) shows a D-LTAG auxiliary tree anchored with the discourse adverbial *otherwise*. Its semantic interpretation is **otherwise**$'(s_s, [e_i]^{a_c})$, which is encoded with two arguments, s_s and $[e_i]^{a_c}$. However, among these arguments, only s_s is a structural one (obtained via adjunction). The argument $[e_i]^{a_c}$ denotes an anaphoric argument. Finding the value of $[e_i]^{a_c}$ is beyond the *compositional* account of discourse semantics in D-LTAG.

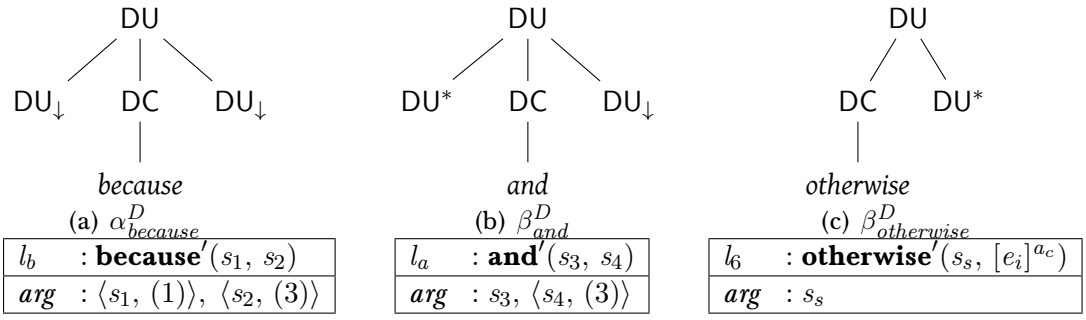


Figure 5.11: D-LTAG semantic interpretations of discourse connectives

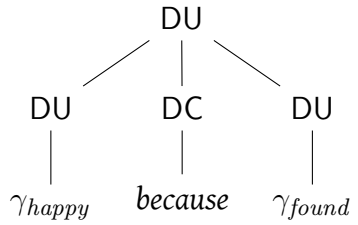
5.1.4.1 Subordinate Conjunctions

We consider the following example of a discourse with a single subordinate conjunction.

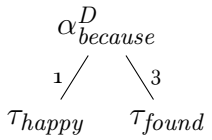
(61, repeated)
Sue is happy because she found a job.

Figure 5.10 and Figure 5.11 provide the D-LTAG grammar for (61). Figure 5.13(a) shows the D-LTAG derived tree of the discourse (61).

In order to interpret the discourse (61), we refer to its D-LTAG derivation tree depicted in Figure 5.12(b) on the following page. As it shows, τ_{happy} and τ_{found} substitute into $\alpha^D_{because}$. The operations on these trees give rise to the operations on their interpretations (shown in Figure 5.10 and Figure 5.11(a)). In particular, since τ_{happy} substitutes at the address 1 into $\alpha^D_{because}$, in the interpretation of $\alpha^D_{because}$, the label of the interpretation of τ_{happy}, denoted by l_h, substitutes the label variable s_1 associated with the address 1. Analogously, l_f labeling the interpretation of τ_{found} substitutes the label variable s_2 in **because**$'(s_1, s_2)$. Thus, one obtains the interpretation shown in Figure 5.12(c), which indeed is a coherent interpretation of (61).

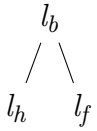


(a) The derived tree



(b) The derivation tree

l_b : **because**$'(l_h,\ l_f)$,
l_h : **happy**(**sue**),
l_f : **found**(**sue**, **job**).



(c) The interpretation of the discourse: the formula and its graphical representation

Figure 5.12: The D-LTAG derived tree, derivation tree, and the interpretation of the discourse

5.1.4.2 Coordinate Conjunctions

(65) Sue found a job and she likes it.

Figure 5.13(a) on the facing page depicts the derived tree of the two clause sentence (65). Figure 5.13(b) on the next page shows the corresponding derivation tree. To interpret (65), we refer to the derivation tree. It consists of three nodes, τ_{found}, τ_{likes}, and β^D_{and}. Their interpretations are provided in Figure 5.10 and Figure 5.11(b) on the preceding page. By substituting τ_{likes} into β^D_{and}, **and**$'$ obtains one of the arguments. In particular, the label variable s_4 obtains as its value the label of τ_{likes}, which is l_l.

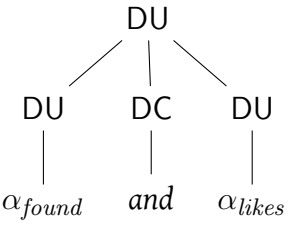
By adjoining the tree β^D_{and} into the tree τ_{found}, the predicate **and**$'$ receives the value for the other argument, which is s_3. The value of s_3 becomes the label of the interpretation of τ_{found}, namely, l_f. Hence, we obtain the semantic interpretation shown in Figure 5.13(c) on the next page.

5.1.4.3 Interaction between Subordinate and Coordinate Conjunctions

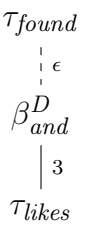
Let us consider the following discourse where the subordinate and coordinate conjunctions interact.

(66) Sue is happy because she found a job and she likes it.

The derived tree for the discourse (66) is the one in Figure 5.14(a). Its derivation tree is shown in Figure 5.14(b). If one builds an interpretation of the discourse according to this derivation tree in the same way we did in the previous two examples, one obtains the interpretation shown in Figure 5.14(c), which is an incoherent interpretation of

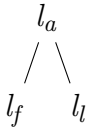


(a) The derived tree



(b) The derivation tree

l_a : **and**′$(l_f,\ l_l)$,
l_f : **find**(**sue**, **job**),
l_l : **like**(**sue**, **job**).



(c) The interpretation of the discourse: the formula and its graphical representation

Figure 5.13: The derived and derivation trees, and the interpretation of the discourse

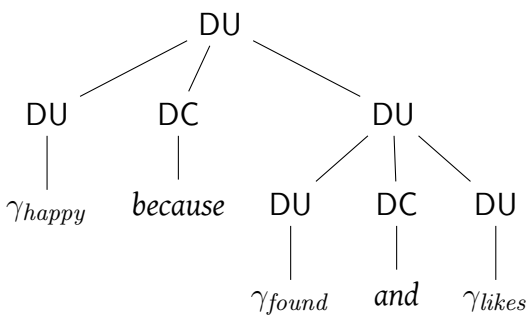
(66). Indeed, according to the derivation tree, one substitutes τ_{happy} in $\alpha^D_{because}$. Thus, one of the arguments of **because**′ becomes the label of the interpretation of τ_{happy}, l_h. The tree τ_{found} substitutes into the other substitution site of $\alpha^D_{because}$ and therefore the second argument of **because**′ becomes the label of the interpretation of τ_{found}. Hence, we obtain **because**′(l_h, l_f). By substituting τ_{likes} in the tree β^D_{and}, the second argument of **and**′ becomes l_l. By adjoining β^D_{and} into τ_{found}, the first argument of **and**′ obtains the value l_f. The interpretation shown in Figure 5.14(a) is incoherent. One can formulate it as follows: *Sue is happy that she found a job. She likes it.* This does not have the same meaning as *Sue is happy because she found a job and she like the job she found.* To overcome this problem, D-LTAG makes use of a *flexible direction of composition*, previously proposed for LTAG in (Aravind Joshi, Kallmeyer, and Romero, 2007). It allows one to traverse a derivation tree so that it can start at any node. However, one needs to *restrict* the principle of flexible composition in order to equate the generative capacity with the one of the original D-LTAG grammar.

The bottom up traversal of the derivation tree of the discourse (66) (see Figure 5.14(b) on the following page) yields the interpretation shown in Figure 5.15, which is the coherent interpretation of (66). Indeed, we have:

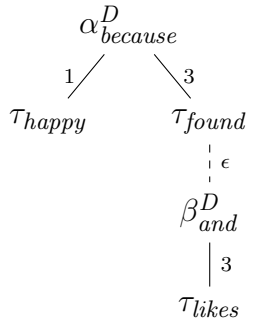
step1 τ_{likes} substitutes into β^D_{and}.
In result, the semantic interpretation of β^D_{and} receives the value of its argument denoted by s_4 (see Figure 5.11). The value of s_4 becomes the label of τ_{likes}, i.e., l_l.

step2 The resultant tree of **step1** adjoins into the tree τ^D_{found}.
As a result, the semantic interpretation of β_{and} obtains a value of its other argument, denoted by s_3. Since the label of interpretation of τ_{found} is l_f, we assign l_l to s_3.

step3 The tree produced as a result of **step2** substitutes into $\alpha^D_{because}$ at the Gorn address 3.
Thus, the interpretation of $\alpha^D_{because}$ obtains the value of its s_2 argument (as s_2 is marked with (3), it indicates that the label of the interpretation of a tree substituted at address 3 becomes the value of s_2, see Figure 5.11 on page 141). The label that becomes the value of s_2 is l_a (as it is the label of the formula with the predicate **and**, see Figure 5.11).

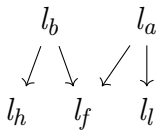


(a) The derived tree



(b) The derivation tree

$$
\begin{array}{|l|}
\hline
l_b : \textbf{because}'(l_h,\ l_f), \\
l_a : \textbf{and}'(l_f,\ l_l), \\
l_h : \textbf{happy}(\textbf{sue}), \\
l_f : \textbf{find}(\textbf{sue},\ \textbf{job}), \\
l_l : \textbf{like}(\textbf{sue},\ \textbf{job}). \\
\hline
\end{array}
$$



(c) An incoherent interpretation: the formula and its graph representation

Figure 5.14: D-LTAG derived and derivation trees

step4 τ_{happy} substitutes in the tree obtained in **step3**.

In this way, $\alpha^D_{because}$ fills both of its substitution sites. Thus, the s_1 argument of *because* obtains the value l_3 as it is the label denoting the interpretation of τ_{happy}.

As a result of **step4**, we obtain the interpretation shown in Figure 5.15, which is the coherent interpretation of (66).

As (Forbes-Riley, Bonnie Webber, and Aravind Joshi, 2006) indicates, only using a bottom-up traversal of a derivation tree may turn out to be insufficient. However, by allowing both the bottom up and top down traversals, the ambiguity of discourse parsing increases (as the number of possible traversals increases). To avoid the ambiguity increase, D-LTAG only allows one to consider the bottom-up traversal of a derivation tree.

$$
\begin{array}{|l|}
\hline
l_b : \textbf{because}'(l_h,\ l_a), \\
l_a : \textbf{and}'(l_f,\ l_l), \\
l_h : \textbf{happy}(\textbf{sue}), \\
l_f : \textbf{found}(\textbf{sue},\ \textbf{job}), \\
l_l : \textbf{likes}(\textbf{sue},\ \textbf{job}). \\
\hline
\end{array}
$$

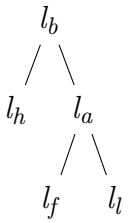


Figure 5.15: The coherent interpretation of discourse obtained by a bottom-up traversal of the derivation tree

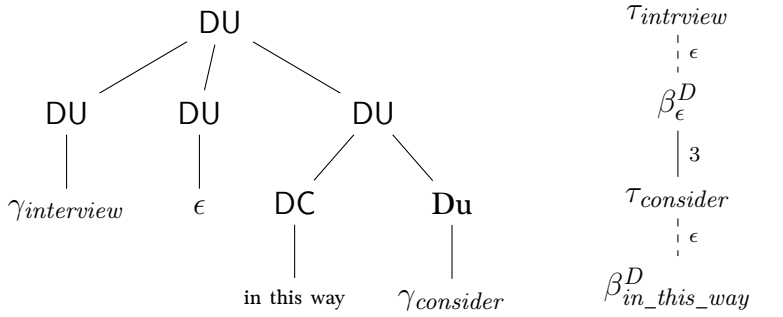
5.1.4.4 Adverbial Connectives

An adverbial connective gives rise to a discourse relation with two arguments, but only one of them can be obtained compositionally, namely, the one that is expressed by the structural argument of an adverbial connective. To illustrate that, let us consider the following discourse consisting of two sentences:

(62, repeated) a. The company interviewed everyone.

b. In this way, they considered all their options.

Figure 5.16 illustrates the derived and derivation of the discourse (62). Since no structural connective relates the first and second clauses, D-LTAG inserts the lexically unexpressed connective ϵ between them (see Remark 5.2 on page 139).



(a) A derived tree

(b) A derivation tree

Figure 5.16: D-LTAG derived and derivation trees

Each elementary tree is associated with a semantic recipe (see Figure 5.17 and Figure 5.18 on the following page).[46] The semantic interpretation of an adverbial connective encodes that one of its arguments is structural (denoted by s_s), whereas the other one, denoted by $[e_i]^{ac}$, has to be retrieved ***anaphorically***. To obtain the coherent interpretation of the discourse (62), one makes use of a bottom-up traversal of the derivation tree. D-LTAG traverses the derivation tree in Figure 5.16(b) as follows:

step1 $\beta^D_{in\_this\_way}$ adjoins into the tree $\tau_{consider}$.

In result, the semantic interpretation of $\beta^D_{in\_this\_way}$, i.e., $\textbf{in\_this\_way}'(s_s, [e_i]^{a_c})$ obtains the value of its structural argument. That is, s_s becomes l_c, which is the label of the interpretation of $\tau_{consider}$.

step2 β^D_ϵ adjoins into $\alpha_{interview}$.

As a result, the interpretation of β^D_ϵ, that is, $\epsilon'(s_2, s_3)$ receives l_i (the label of $\alpha_{interview}$) as the value for the argument variable s_2 (see Figure 5.17).

[46]It is noteworthy that D-LTAG computes the semantic interpretation of the compound adverbials compositionally. In particular, D-LTAG derives the semantic interpretation of ***in this way*** by composing the semantic interpretations of ***in***, ***this***, and ***way***. Nevertheless, for the sake of simplicity, we assume that the semantic interpretation of the adverbial connective ***in this way*** is already provided as an entry.

step3 The resultant tree of **step1**, i.e., the tree obtained by adjoining $\beta^{D}_{in\_this\_way}$ into $\alpha_{consider}$, substitutes at the address 3 into the tree obtained as a result of **step2**. As a consequence, the vacant argument of $\epsilon'(l_i,\ s_3)$, that is, s_3 receives as its value the label of the interpretation of the tree produced in **step1**, which is the label of l_1 : **in_this_way**$'(l_c,\ [e_i]^{a_c})$, i.e., l_1.

l_1	: **in_this_way**$'(s_1,\ [e_i]^{a_c})$
arg	: s_1

(c) The adverbial connective *in this way*

l_2	: $\epsilon'(s_2,\ s_3)$
arg	: $s_2, \langle s_3, (3)\rangle$

(d) The structural connective ϵ

Figure 5.17: D-LTAG semantic interpretations of discourse connectives

l_i	: **interview**(**c**, **e**)
arg	: –

l_c	: **consider**(**c**, **o**)
arg	: –

Figure 5.18: D-LTAG semantic interpretations of clauses

Thus, we obtain the labeled formula shown in Figure 5.19. The value of the anaphoric argument of **in_this_way**, which is $[e_i]^{ac}$, is not specified. In order to obtain the fully specified interpretation of the discourse, one may use the anaphora resolution to identify the label that can be value of $[e_i]^{ac}$. The possible values of $[e_i]^{ac}$ can be labels of the interpretations of discourse units belonging the piece of discourse that an adverbial connective attaches to. However, the value of $[e_i]^{ac}$ can also be a label of some proposition that is *inferred* in the discourse. In the case of the discourse (62), $[e_i]^{ac}$ resolves to l_i, which is the label of l_i : **interview**$'$(**c**, **e**). Thus, one would obtain the fully specified interpretation by instantiating $[e_i]^{ac}$ with the label l_i.

5.1.4.5 Computing Semantics of a Discourse with a Parasitic Connective

As (Bonnie Webber, Stone, Aravind Joshi, and Knott, 2003) suggests, some lexical items that could be *classified* as discourse connectives do not follow the pattern of discourse connectives. They differ from both the structural and adverbial connectives by their semantic properties. One of such adverbials is a *parasitic adverbial connective.*

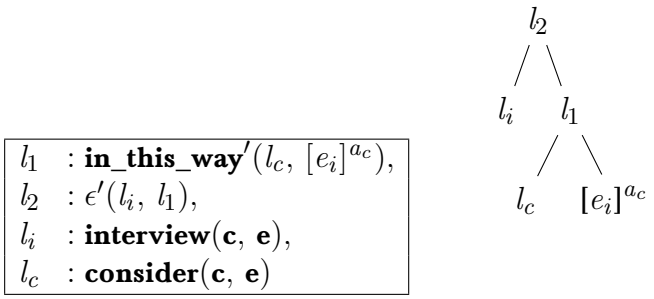


Figure 5.19: The D-LTAG interpretation of discourse and its tree representation

5.1.4.5.1 The Interpretation of a Parasitic Adverbial Connective

Let us consider the adverbial connective *for example*. It exhibits a *parasitic* behavior on a structural connective. In particular, the adverbial connective *for example* does not signal a discourse relation with two arguments, but rather *modifies* a discourse relation signaled by a structural connective. For the sake of illustration, we consider the following example:

(67) John just broke his arm. So, for example, he can't cycle to work.

To interpret the discourse (67), we first discuss the behavior of *for example* at the sentence-level. Let us consider the following sentence with the adverbial *for example*:

(68) The collection includes, *for example*, a piece of hematite.

In order to see the semantic contribution of *for example* in the sentence (68), we remove *for example* from it. We obtain the following sentence:

(69) The collection includes a piece of hematite.

One interprets the sentence (69) as **includes**(**collection**, **hematite**). The difference between the sentences (68) and (69) is that in (68) *for example* stresses the point that a *hematite is one of the things that the collection contains*. Therefore, if we represent *the things that are included in the collection* as a set, we can interpret *for example* with two arguments, *an object* and *a set* that are related as follows:

$$I_1 = \textbf{exemplification}' (\textbf{hematite}, \{x \mid \textbf{include}(\textbf{collection}, x)\}) \tag{5.70}$$

$$I_2 = \textbf{exemplification}' (\textbf{hematite}, \lambda x. \textbf{include}(\textbf{collection}, x)) \tag{5.71}$$

Note that I_2 defined in Equation (5.71) is a λ-notational version of I_1 from Equation (5.70). In Equation (5.71), $\lambda x. \textbf{include}(\textbf{collection}, x)$ encodes the characteristic function of the set. Since I_2 is a sentence-level interpretation, but not a discourse-level one, we cannot use I_2 as the interpretation of *for example* in the case of (67), where *for example* modifies a discourse connective. Nevertheless, one assumes that the semantic properties of the discourse-level *for example* are close to the properties of *for example*

at the sentence-level. In other words, the discourse-level interpretation of *for example* can be constructed with the help of I_2. With I_2 in mind, to establish the interpretation of an adverbial connective *for example*. We consider the piece of discourse that is on the left of *for example* (let us denote it with D and let its interpretation be δ), and the clause in which *for example* occurs (let us denote it by S and the interpretation of S with σ). Now, let us consider again the two clause sentence (67). In (67), the fact that *he (John) can't cycle to work* (S with interpretation σ) serves as one of the *examples* of the *results* which are outcomes of *John just broke his arm* (D with interpretation δ). *John just broke his arm* (σ) is the first argument of the structural connective *so*. In general, one interprets S_1 *so,* S_2 as $\mathbf{so}'\ (d_1, d_2)$, where d_i is interpretation of S_i. Thus, one proposes the following interpretation of the (67) sentence:

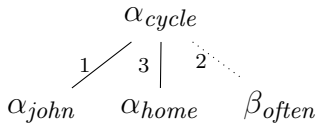
$$I_p = \mathbf{exemplification}'(\sigma,\ \lambda x.\mathbf{so}'\ (\delta, x)) \tag{5.72}$$

Indeed, I_p is similar to I_1 (see Equation (5.71)): σ exemplifies *the results of John braking his arm*, like *a hematite* exemplifies *what is contained in the collection.* Thus, we obtain a semantic representation in a case where the adverbial connective *for example* is parasitic on a structural connective.

5.1.4.5.2 D-LTAG and Hole Semantics

In order to derive a semantic interpretation such as I_p in defined Equation (5.72), D-LTAG (Bonnie Webber, Stone, Aravind Joshi, and Knott, 2003) makes use of *Hole Semantics* (Bos, 1995), discussed in Section 4.3.2.1. To illustrate the way D-LTAG makes use of Hole Semantics, let us consider the following sentence:

(73) John often cycles home.



(a) The LTAG derivation tree

l_1 : **cycle**(**john**, **home**)
$l_1 \leq h_1$
arg : $-$

(b) The interpretation of a clause

l_2 : **often**(h_2)
$l_2 \leq g_1$
$s_1 \leq h_2$
arg : $g_1,\ s_1$

(c) The interpretation of the adverbial

Figure 5.20: The LTAG derivation tree and semantic recipes

We use h_i for holes, and g_i for hole variables, whereas we use l_i and s_i to denote labels and label variables as before. The LTAG derivation tree for (73) is in Figure 5.20(a). Figure 5.20(b) shows the interpretation of the clause *John cycles home* in Hole Semantics.

Figure 5.20(c) illustrates the semantic recipe for β_{often}. It introduces the label l_2, the hole h_2, the hole variable g_1, and the propositional variable s_1. According to the derivation tree, β_{often} adjoins into α_{cycles}. In result, the label variable s_1 in the

interpretation of β_{often} obtains l_1 as its value, which is the propositional label of the interpretation of α_{cycles}. The hole variable of g_1 in the interpretation of β_{often} becomes the hole h_1, which is a hole of the interpretation of α_{cycles}. Thus, we obtain the following interpretation of the discourse:

$$\boxed{l_1 : \textbf{cycle}(\textbf{john}, \textbf{home}),\ l_2 : \textbf{often}(h_2),\ l_1 \leq h_1,\ l_1 \leq h_2,\ l_2 \leq h_1} \tag{5.74}$$

From the interpretation in (5.74), we find that $h_2 < l_2$ (as h_2 appears in the formula labeled with l_2). Since $l_1 \leq h_2$ and $h_2 < l_2$, we obtain $l_1 < l_2$. Since $h_2 \leq l_1$, we conclude that $l_1 < h_1$, and thereby $h_1 \neq l_1$. The only possible disambiguation is $h_1 = l_2$ and $h_2 = l_1$. Hence, we obtain the following interpretation out of (5.74):

$$\boxed{l_1 : \textbf{cycle}(\textbf{john}, \textbf{home}),\ l_2 : \textbf{often}(l_1)} \tag{5.75}$$

5.1.4.5.3 Computing Interpretation of a Discourse with a Parasitic Adverbial

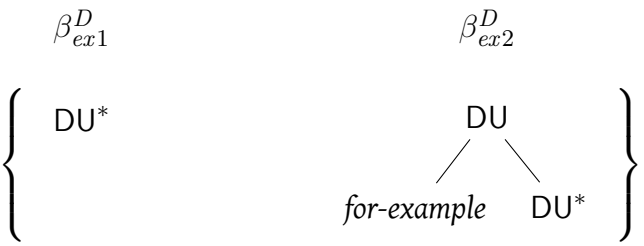
In order to develop a compositional approach to the parasitic discourse connective *for example*, D-LTAG makes use of an MCTAG approach.

(67, repeated) John just broke his arm. So, for example, he can't cycle to work.

Figure 5.21(a) shows the elementary structure of *for example* in an MCTAG. It is a set of two TAG auxiliary trees, denoted with β^D_{ex1} and β^D_{ex2}. The tree β^D_{ex2} adjoins in the discourse unit it modifies. In the case of (67), β^D_{ex2} adjoins in the initial tree anchored with *he can't cycle*. The auxiliary tree β^D_{ex1}, which has a single node tree, adjoins on the root of the higher discourse unit. In the case of (67), β^D_{ex1} adjoins on the root node of the DU-derived tree obtained out of the initial tree anchored with *so* by filling its DU-substitution sites. Figure 5.22 shows the derivation tree for (67). However, our derivation tree indicates that it is not *tree-local*, because β^D_{ex1} and β^D_{ex2} do not adjoin into the same tree, but in two different trees. Nonetheless, as β^D_{ex1} does not have any syntactic material, one can allow β^D_{ex1} and β^D_{ex2} to adjoin into different trees without increasing the generative power of a tree-local MCTAG (A. K. Joshi, Kallmeyer, and Romero, 2003).

Figure 5.21(b) and Figure 5.21(c) show interpretations associated with the trees β^D_{ex1} and β^D_{ex2} respectively. The interpretation of β^D_{ex1} encodes that $\textbf{exemplify}'$ has two holes. One of them must be filled by something that outscops s_1, where s_1 is a (propositional) argument variable. s_1 obtains its value by adjoining of β^D_{ex1} into a tree. The abstracted propositional variable s in $\textbf{exemplify}'(h_1, \lambda s.h_2)$ should have an occurrence in the formula labeled with a value of h_2, otherwise an obtained interpretation would be incoherent.

In the interpretation of β^D_{ex2}, s is the same variable as the abstracted one in the interpretation of β^D_{ex1}. In this way, adjoining β^D_{ex1} and β^D_{ex2} gives rise to $\lambda s.\cdots s\cdots$. Furthermore, in the interpretation of β^D_{ex2}, the argument variable s_2 is outscoped by h_1, which is a hole in $\textbf{exemplify}'$. The interpretation of β^D_{ex2} has a constraint that g_1, which denotes a variable for holes, outscopes l_2. The variables s_2 and g_1 obtain their values as β^D_{ex2} adjoins into a tree.



(a) An MC-TAG tree set

l_1 : **exemplify**$'(h_1,\ \lambda s.h_2)$
$s_1 \leq h_2$
arg : s_1

(b) The interpretation of β^D_{ex1}

$l_2 : s$
$l_2 \leq g_1$
$s_2 \leq h_1$
arg : s_2

(c) The interpretation of β^D_{ex2}

Figure 5.21: The elementary tree set for *for example* and its interpretation

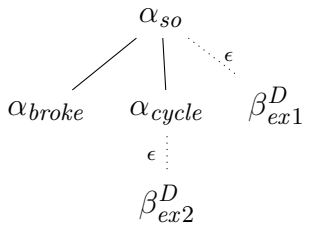


Figure 5.22: The MCTAG derivation tree

l_3 : **so**$'(s_3, s_4)$
$l_3 \leq h_3$
arg : $\langle s_3(1), s_4(3)\rangle$

l_4 : **break**(**john**, **arm**)
$l_4 \leq h_4$
arg : $-$

l_5 : $\neg$**cycle**(**john**, **work**)
$l_5 \leq h_5$
arg : $-$

Figure 5.23: Hole semantics for clauses and connectives

Thus, the derivation tree is traversed as follows: β^D_{ex2} adjoins into α_{cycle}. In result, s_2 becomes l_5 (as it labels $\neg$**cycle**(**john**, **work**)), wheres g_1 obtains the value of the hole that outscopes the label of $\neg$**cycle**(**john**, **work**), which is h_5 (see Figure 5.23). The resultant derived tree substitutes into α^D_{so} at the Gorn address 3. Hence, the variable s_4 from the interpretation of *so* becomes l_2 (the label of s). At the same time, β^D_{ex1} adjoins into the root node of α_{so}. Thus, the variable s_1 obtains the value l_3 (the label of **so**$'(s_3, s_4)$, see Figure 5.23). In this way, one obtains an interpretation shown in Figure 5.24.

One disambiguate the holes having occurrences in the interpretation shown in Figure 5.24 as follows: $h_2 = l_3,\ h_3 = l_1,\ h_1 = l_5,\ h_4 = l_4,$ and $h_5 = l_2$. Using these values for the holes in Figure 5.24, one obtains the following interpretation of the discourse:

$$\textbf{exemplify}'(\neg\textbf{cycle}'(\textbf{john}, \textbf{work}),\ \lambda s.\textbf{so}'(\textbf{break}'(\textbf{john}, \textbf{arm}), s))$$

$$
\begin{array}{l}
l_1 : \textbf{exemplify}'(h_1, \lambda s.h_2) \\
l_2 : s \\
l_3 : \textbf{so}'(l_4, l_2) \\
l_4 : \textbf{break}'(\textbf{john}, \textbf{arm}) \\
l_5 : \neg\textbf{cycle}'(\textbf{john}, \textbf{work}) \\
l_3 \leq h_2,\ l_2 \leq h_5,\ l_5 \leq h_1,\ l_3 \leq h_3,\ l_4 \leq h_4,\ l_5 \leq h_5
\end{array}
$$

Figure 5.24: An interpretation of a discourse in Hole Semantics

5.1.5 Discourse Structure

We are interested in the properties of *discourse structures* that D-LTAG semantic interpretations give rise to, that it, in the properties of formulas standing for D-LTAG discourse interpretations. As we already discussed, D-LTAG interprets a discourse as a labeled formula that gives rise to a tree-shaped structure. If the discourse contains adverbial connectives, then its D-LTAG interpretation does not specify the anaphoric arguments of the adverbial connectives. By finding the anaphoric arguments of the adverbial connectives, the D-LTAG interpretation may be turned into a DAG.

It is noteworthy that in D-LTAG, by using only the trees anchored with *subordinate* and *coordinate conjunctions*, it is possible to produce non-tree shaped structures. In other words, without using anaphora resolution, one can produce *compositionally* a non-tree shaped structure. Indeed, let us consider (66) discourse given on page 142, repeated as follows:

(66, repeated) Sue is happy because she found a job and she likes it.

As its derivation tree in Figure 5.14(b) on page 144 illustrates, β_{and} adjoins in the argument of $\alpha_{because}$. The top-down traversal of the derivation tree yields an interpretation of the discourse shown in Figure 5.14(c) on page 144. In this interpretation, we have two sub-formulas $l_1 : \textbf{because}'(l_3,\ l_4)$ and $l_2 : \textbf{and}'(l_4,\ l_5)$ sharing the label l_4, which stands for the interpretation of *she found a job*. The structure that this interpretation determines is not tree-shaped, but rather a DAG. However, this interpretation is incoherent. To avoid argument sharing, one only allows for the bottom up traversal of a derivation tree.

On the other hand, some discourse interpretations are not possible to produce using D-LTAG. For instance, the coherent interpretation of the following discourse is rather a DAG than a tree:

(76) a. John loves Barolo.

b. He first tasted it in 1992.

c. According to Hugh Johnson, it's one of Italy's supreme reds.

The clause (76)(a) is elaborated by the clause (76)(b). At the same time, the clause (76)(a) is elaborated by the clause (76)(c), which is not a continuation of the previous elaboration. Thus, the clause (76)(a) is independently elaborated by the clauses (76)(b) and (76)(c). That is why the interpretation of the (76) discourse should be a multi-parent DAG shown in Figure 5.25. As (Bonnie Webber, Stone, Aravind Joshi, and

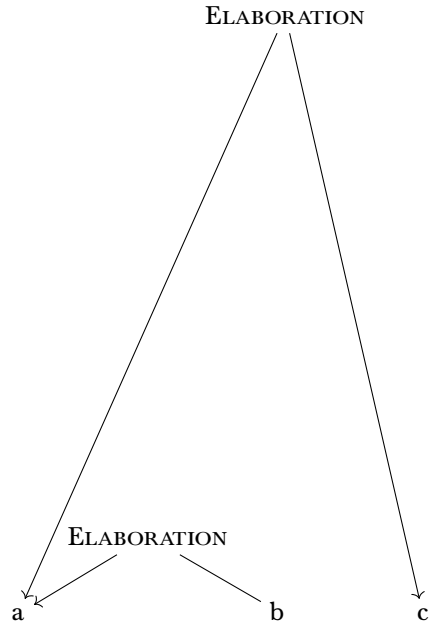


Figure 5.25: A multi-parent DAG

Knott, 2003) notes, one cannot obtain such DAGs with the help of D-LTAG "*because the adjoining and substitution operators in TAG do not let us produce them.*"

5.2 G-TAG

G-TAG is a text generating system (Danlos, 1998, 2000). G-TAG was designed with an aim to implement it in practical applications, to generate technical, domain specific texts (Danlos, Frédéric Meunier, and Combet, 2011; Frédéric Meunier, 1997). G-TAG develops a grammatical approach to text generation based on (L)TAG principles. It generates a text out of a conceptual representation input. Texts that G-TAG generates may consist of one or more sentences and each sentence may consist of one or more clauses. A conceptual representation input of G-TAG incorporates both sentence-level and discourse-level information. The structure of a discourse encoded within a conceptual representation input is tree-shaped. Hence, G-TAG generates texts of tree-shaped discourse structures.

Below, we first overview the general architecture of G-TAG. Then, we discuss the grammar that G-TAG offers and the way G-TAG generates a text.

5.2.1 Architecture

To discuss the G-TAG architecture, we briefly describe the notions that it involves. G-TAG defines *g-derivation* and *g-derived* trees. One can think of a g-derivation tree as the G-TAG counterpart of a (L)TAG derivation tree. A g-derived tree is like a TAG derived tree but its terminal nodes are labeled by lemmas. The morphological information for inflecting lemmas is provided by the labels of their mother nodes. A g-derivation tree gives rise to a unique g-derived tree.

One can divide the text generation of G-TAG into two steps. The one of them is a grammatical step. During this step, G-TAG refers to its grammar. The other one is a post processing step. Figure 5.26 on the next page illustrates the architecture of G-TAG.

5.2.1.1 Grammatical Step

The grammatical step consists of two stages.

Stage 1 One constructs a *g-derivation tree* out of the conceptual representation input. For that G-TAG has a *lexical database*. In the database, each concept points to a set of lexical entries that can be used as a lexicalization of that concept. One constructs a g-derivation tree by selecting lexicalizations of the concepts from the conceptual input.

Stage 2 One computes the *g-derived* tree out of the g-derivation tree produced during the first stage.

5.2.1.2 Post Processing Step

G-TAG employs a *post processing module* in order to produce a text from a g-derived tree. The post processing module computes inflected forms of lemmas out of the morphological information provided in a g-derived tree. Afterwards, by concatenating the inflected words, it produces a text. The post processing module may *modify* the produced text. The original text is called *canonical*. A text obtained by modifying the

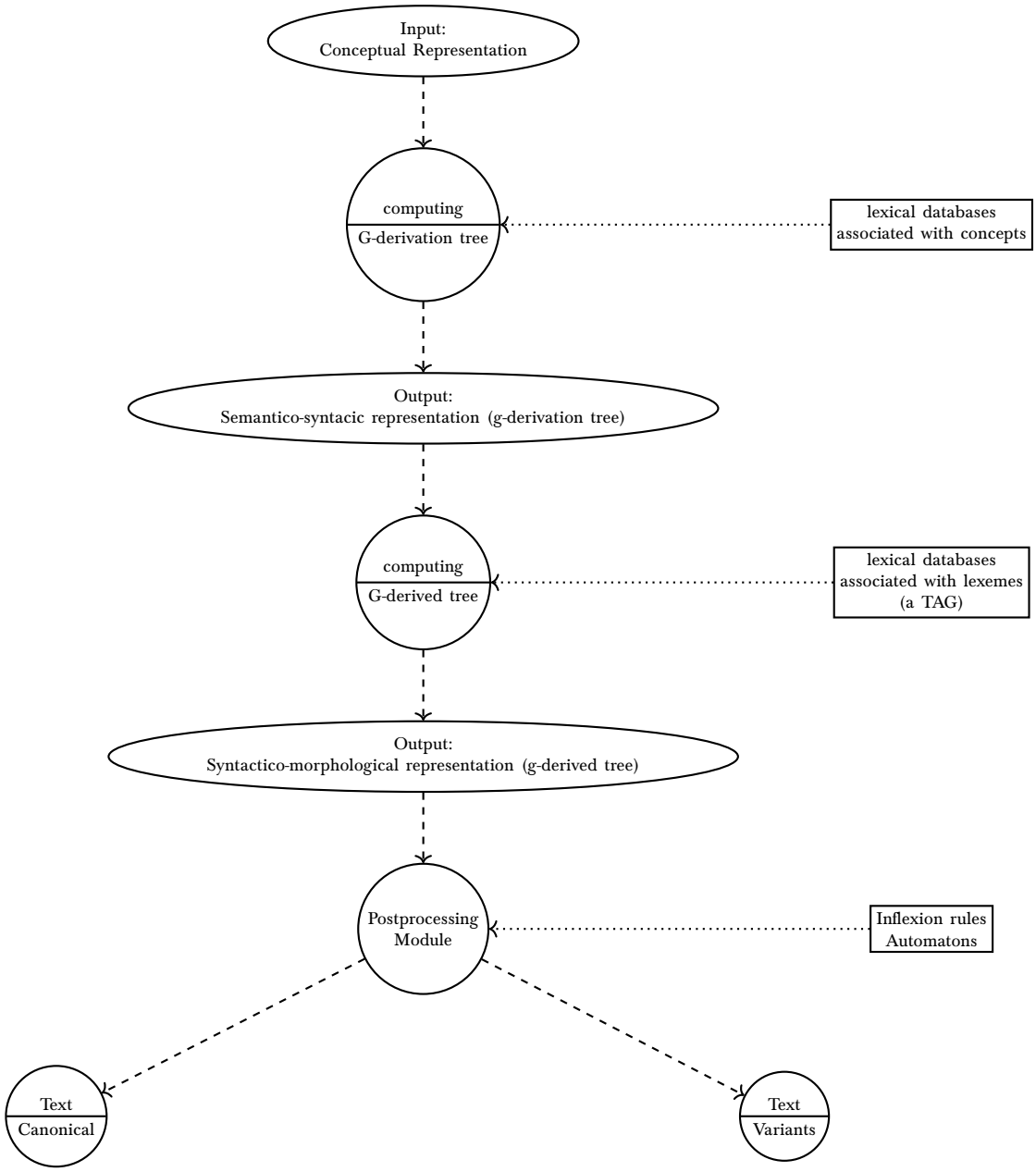


Figure 5.26: The G-TAG architecture

canonical one is referred to as a *variant* of the canonical text. One of the possible modifications the post processing module carries out concerns discourse connectives. In a canonical text, a discourse connective can only occupy a clause-initial position. To support stylistic diversity, the post processing module may move an adverbial connective from the clause-initial position to a clause-medial one. For instance, the post processing module produces the following text as the canonical one:

(77) *Jean a passé l'aspirateur pour être récompensé par Marie.*
John have$_{\text{3PS. SG. PRS.}}$ pass$_{\text{PAST PART.}}$ vaccumer$_{\text{DEF.}}$ for to-be$_{\text{INF.}}$ reward$_{\text{PAST PART.}}$ by Mary.
Ensuite, il a fait une sieste.
Afterwards, he have$_{\text{3PS. SG. PRS.}}$ make$_{\text{PAST PART.}}$ a nap.

'John vacuumed in order to be rewarded by Mary. Afterwards, he took a nap.'

Although (77) is the canonical text, the post processing module may decide not to use it as the output of the system. It may output the following *variant* of (77):

(78) *Jean a passé l'aspirateur pour être récompensé par Marie. Il*
John have$_{\text{3PS. SG. PRS.}}$ pass$_{\text{PAST PART.}}$ vaccumer$_{\text{DEF.}}$ for to-be$_{\text{INF.}}$ reward$_{\text{PAST PART.}}$ by Mary. He
a ensuite *fait une sieste.*
have$_{\text{3PS. SG. PRS.}}$ *afterwards* make$_{\text{PAST PART.}}$ a nap.

'John vacuumed in order to be rewarded by Mary. He afterwards took a nap.'

Thus, the post processing module transforms the canonical text (77) into the text (78) by moving an adverbial connective *ensuite* (afterwards) from the clause-initial position to a clause-medial one.

5.2.2 Conceptual Representation Language

As Figure 5.26 on the facing page illustrates, the starting point in G-TAG text generation is a conceptual representation. In order to encode conceptual representations, G-TAG makes use of a sub-language of LOGIN (Aït-Kaci and Nasr, 1986).

5.2.2.1 LOGIN

LOGIN is an extension of the language of PROLOG. However, instead of the first-order terms used in PROLOG (terms of the form $p(t_1, \ldots, t_n)$), LOGIN defines a *ψ-term.* A *ψ-term* is a record-like typed structure, which can be defined as follows:

Definition 5.2.1 (Aït-Kaci and Nasr (1986)).
A ψ-term has:

1. A root symbol, *which is a type constructor and denotes a class of objects.*
2. Attribute labels, *which are record field symbols, associated with ψ-terms. Each label denotes a function* in intenso *from the root type to the type denoted by its associated sub-ψ-term.* Concatenation *of labels denotes function composition.*
3. Coreference constraints *among paths of labels, which indicate that the corresponding attribute compositions denote the same functions.*

For instance, $\mathtt{M}_1$ defined in Equation (5.79) is a ψ-term. In $\mathtt{M}_1$, `person` is the *root*

symbol. The *attribute labels* of the term $\mathtt{M}_1$ are `id`, `born`, and `father`.

$$\begin{aligned}\mathtt{M}_1 = \mathtt{person}\,(\mathtt{id} \Rightarrow \mathtt{name};\\ \mathtt{born} \Rightarrow \mathtt{date}\,(\mathtt{day} \Rightarrow \mathtt{integer};\\ \mathtt{month} \Rightarrow \mathtt{monthname};\\ \mathtt{year} \Rightarrow \mathtt{integer});\\ \mathtt{father} \Rightarrow \mathtt{person})\end{aligned} \tag{5.79}$$

In a ψ-term, each attribute label has a ψ-term as a value. For instance, in the term $\mathtt{M}_1$, the value of `born` is a ψ-term with the root symbol `date`. Let us denote this term by $\mathtt{M}_2$, defined in Equation (5.80). Thus, under the label `born`, the term $\mathtt{M}_1$ has a sub-ψ-term, which is the term $\mathtt{M}_2$.

$$\begin{aligned}\mathtt{M}_2 = \mathtt{date}\,(\mathtt{day} \Rightarrow \mathtt{integer};\\ \mathtt{month} \Rightarrow \mathtt{monthname};\\ \mathtt{year} \Rightarrow \mathtt{integer})\end{aligned} \tag{5.80}$$

Furthermore, in $\mathtt{M}_2$, the attribute label `year` is associated with a sub-ψ-term `integer`. Consequently, in the ψ-term $\mathtt{M}_1$ (see Equation (5.79)), one can associate the concatenation of two attribute labels `born.year` with a ψ-term `integer`. Thus, an attribute label l is a *function* such that for a given ψ-term `M`, it produces a ψ-term $\mathtt{M}_l$ that is the sub-ψ-term of `M` under the label l. By considering attribute label as functions, *function composition* corresponds to *label concatenation.* For instance, `born` is a function, whose value on the term $\mathtt{M}_1$ is the ψ-term $\mathtt{M}_2$. The label `year` is a function that maps the ψ-term $\mathtt{M}_2$ to the ψ-term `integer`. One identifies with the label concatenation `born.year` a function that maps the ψ-term $\mathtt{M}_1$ to the ψ-term `integer`.

$$\begin{aligned}\mathtt{M}_4 = \mathtt{person}\,(\mathtt{id} \Rightarrow \mathtt{name}\,(\mathtt{first} \Rightarrow \mathtt{string};\\ \mathtt{last} \Rightarrow X : \mathtt{string});\\ \mathtt{father} \Rightarrow \mathtt{person}\,(\mathtt{id} \Rightarrow \mathtt{name}\,(\mathtt{last} \Rightarrow X)))\end{aligned} \tag{5.81}$$

The ψ-term $\mathtt{M}_4$ (see Equation (5.81)) illustrates an example of *coreference*: The symbol X occurs under `id.last` and `father.id.last`, which indicates that these two attribute compositions denote the same functions. The possibility of coreferring enables one to describe looping, infinite structures with the help of ψ-terms.

5.2.2.2 The Language of G-TAG

G-TAG does not make use of the full-fledged LOGIN language, but of its limited fragment. G-TAG limits the symbols that one can use as labels in a ψ-term. In

particular, only the symbols belonging to the following categories can serve as a root symbol of a ψ-term of the G-TAG conceptual representation language: SECOND ORDER RELATION, FIRST ORDER RELATION, and THING. The SECOND ORDER RELATION and FIRST ORDER RELATION categories are unified under the RELATION category. Each *concept* from a G-TAG input is realized as one of the symbols from these categories. A concept represents either a clause-level or a discourse-level phenomenon. For instance, the concept SUCCESSION is from the SECOND ORDER RELATION category and the concept HUMAN belongs to the THING category. While HUMAN encodes a clause-level phenomenon, SUCCESSION gives rise to a discourse-level one, namely, to a temporal relation between two events.

A concept is associated with the set of its *arguments* (possible empty). To model that in ψ-terms, one defines the set of attribute labels associated with a given symbol. A symbol encodes a concept and the set of attribute labels associated with it encodes the set of arguments of the concept. The requirement is that if a given symbol appears as the root in a ψ-term, then the ψ-term must have sub-ψ-terms under the attribute labels associated with the root. For example, SUCCESSION is associated with two arguments, denoted with 1^{st}EVENT and 2^{nd}EVENT. As we declare 1^{st}EVENT and 2^{nd}EVENT to be the arguments of SUCCESSION, every ψ-term whose root symbol is SUCCESSION must have sub-ψ-terms under the labels 1^{st}EVENT and 2^{nd}EVENT.

Attribute labels of a G-TAG concept, i.e., the arguments of a concept are *conceptually* restricted. To illustrate this, let us consider VACUUMING, which is a FIRST ORDER RELATION. It has a single argument denoted with VACUUMER. G-TAG puts the following constraint on the concept VACUUMING and its argument VACUUMER: Any ψ-term with VACUUMING as its root symbol should have under the label VACUUMER only a ψ-term whose root is HUMAN.

A SECOND ORDER RELATION has two arguments each of which is a RELATION. A FIRST ORDER RELATION has (several) arguments that are either THINGS or FIRST ORDER RELATIONS. One encodes individual *entities* with the help of THING, whereas one uses a FIRST ORDER RELATION in order to express that some individual entities are in a *relation* (clause-level information). With SECOND ORDER RELATIONS, one encodes *relations between relations* (discourse-level information).

$$
\begin{aligned}
\mathtt{E}_0 &=: \mathtt{SUCCESSION}[1^{st}\mathtt{EVENT} \Rightarrow \mathtt{E}_1,\ 2^{nd}\mathtt{EVENT} \Rightarrow \mathtt{E}_2] \\
\mathtt{E}_1 &=: \mathtt{GOAL}[\mathtt{Action} \Rightarrow \mathtt{E}_{11},\ \mathtt{Purpose} \Rightarrow \mathtt{E}_{12}] \\
\mathtt{E}_2 &=: \mathtt{NAPPING}[\mathtt{NAPPER} \Rightarrow \mathtt{H}_1] \\
\mathtt{E}_{11} &=: \mathtt{VACUUMING}[\mathtt{VACUUMER} \Rightarrow \mathtt{H}_1] \\
\mathtt{E}_{12} &=: \mathtt{REWARDING}[\mathtt{REWARDER} \Rightarrow \mathtt{H}_2,\ \mathtt{REWARDEE} \Rightarrow \mathtt{H}_1] \\
\mathtt{H}_1 &=: \mathtt{HUMAN}[\mathtt{NAME} \Rightarrow \mathtt{Jean},\ \mathtt{gender} \Rightarrow \mathtt{masc}] \\
\mathtt{H}_2 &=: \mathtt{HUMAN}[\mathtt{NAME} \Rightarrow \mathtt{Marie},\ \mathtt{gender} \Rightarrow \mathtt{fem}]
\end{aligned}
$$

Figure 5.27: An input of G-TAG

Figure 5.27 illustrates an example of a G-TAG input. It consists of ψ-terms. Let us consider one of them, for instance, a ψ-term $\mathtt{E}_0$. Its root symbol is a SECOND ORDER RELATION, SUCCESSION. Since SUCCESSION has two arguments, namely 1^{st}EVENT and

2^{nd}EVENT, the ψ-term E_0 has two non-empty ψ-terms under the labels 1^{st}EVENT and 2^{nd}EVENT, E_1 and E_2, respectively. The ψ-terms E_1 and E_2 are also specified within the given input. E_1 is a ψ-term with root symbol GOAL, which is a SECOND ORDER RELATION. In G-TAG, the arguments associated with GOAL are Action and Purpose. The ψ-term E_2 has NAPPING as its root symbol, which is a FIRST ORDER RELATION and which has only one argument NAPPER from the THING category.

5.2.2.3 Conceptual Representation Inputs as Trees

As we already mentioned, the *discourse structure* encoded within a conceptual representation input is tree-shaped. We show how one can represent a G-TAG input as a tree. Our transformation is valid only for the ψ-terms that appear in G-TAG conceptual representation inputs.

- Given a ψ-term, we use its root symbol as the label of the root of the tree that we are building. Under the attribute labels associated with the root, we have ψ-terms, which we use as the daughter nodes of the root. We label the edges connecting the root with the daughters by the attribute labels. To the ψ-term that are the daughters of the root, we apply the same procedure. At the end of this recursive process, we obtain a tree out of a given ψ-term.
- Within the input, we find the ψ-term that does not appear as a proper sub-ψ-term to any other ψ-term in the conceptual representation input. In G-TAG conceptual representation inputs, one can always find such a ψ-term. We transform that ψ-term into a tree. In this way, we obtain the tree representation of the discourse structure of the G-TAG input.

For instance, to transform the input in Figure 5.27, we find the ψ-term E_0 (it is not a proper sub-ψ-term to any other ψ-term in the conceptual representation input). We represent E_0 as a tree in Figure 5.28.

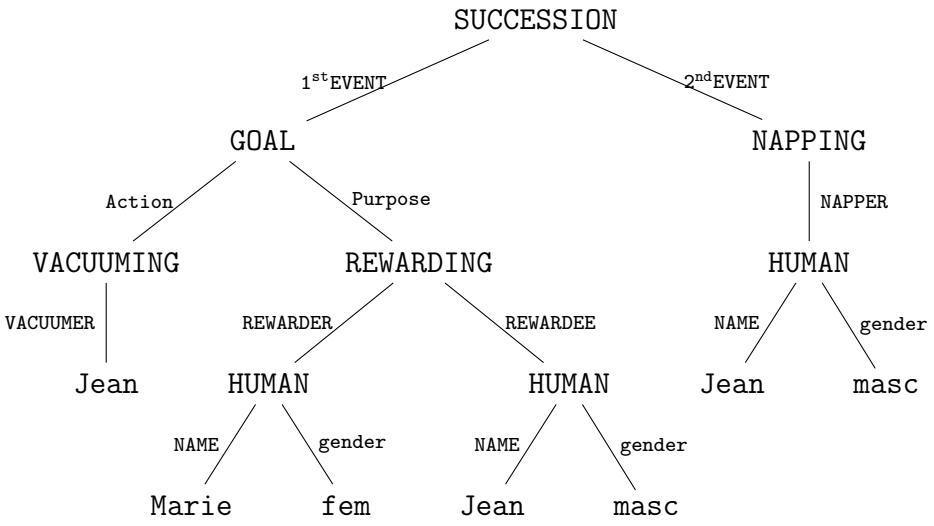


Figure 5.28: The tree representation of a G-TAG conceptual representation input

Remark 5.3. *The claim that the G-TAG conceptual representation input can be represented as a tree concerns only representations of the* discourse-level *phenomena. That is, if one records the predicate-argument relations of the* SECOND ORDER RELATIONS *and their augments, one obtains a tree. Otherwise, if one considers the clause-level representations, then one obtains a DAG rather than a tree. For instance, in the conceptual representation in Figure 5.27,* H_1 *is shared by two relations belonging to* FIRST ORDER RELATION*s. However,* FIRST ORDER RELATION*s encode clause-level phenomena rather than discourse-level ones.*

5.2.3 Lexical Databases

To generate a text out of a conceptual representation input, G-TAG makes use of lexical databases. One of them records correspondence between concepts and their lexicalizations, which are *lexical entries* of G-TAG. The other database pairs lexical entries and their lexico-syntactic realizations with elementary trees. Thus, both databases share lexical entries, which are mediators between concepts and elementary trees. We first describe lexical entries of G-TAG and the way they are linked with concepts. Then, we discuss how one associates elementary trees with a lexical entry.

5.2.3.1 Lexical Entries

To each concept, one associates *a set of lexical entries.* Each of these lexical entries may serve as a lexicalization of that concept. One denotes a lexical entry by a lemma. For example, récompenser (to reward)[47] is a lexical entry of G-TAG.

Concepts may have arguments, and so do lexical entries. The arguments of a lexical entry are the *thematic roles* associated with the lexical entry. G-TAG records the correspondence between the arguments of the concept and the arguments of a lexical entry that serves as the lexicalization of that concept. For instance, let us consider REWARD, which is a FIRST ORDER RELATION. It has two arguments from the category THING, denoted with RewardER and RewardEE. A lexical entry récompenser (to reward) is a lexicalization of the concept REWARD. To lexicalize the arguments of REWARD, récompenser also has two arguments, namely, the arg1 and arg2 thematic roles. The database records that the thematic role arg1 (resp. arg2) récompenser corresponds to the RewardER (resp. RewardEE) argument of REWARD.

G-TAG encodes lexical entries as trees, called *underspecified g-derivation* trees. In an underspecified g-derivation tree, one distinguishes between the *constant* node and the *variable* ones. The constant node is the name of the lexical entry. The variable nodes correspond to the lexicalizations of the arguments of the concept that is lexicalized by the lexical entry.

For example, the underspecified g-derivation tree in Figure 5.29 on the next page stands for a lexical entry récompenser (to reward). This underspecified g-derivation tree has the constant node *récompenser*, whereas the other nodes, RewardER and RewardEE, are variable ones. The variable nodes RewardER and RewardEE are linked with *récompenser* by the thematic roles arg1 and arg2, respectively. Since récompenser is a lexicalization

[47] By underscore, we denote lexical entries.

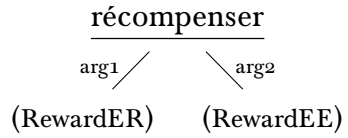
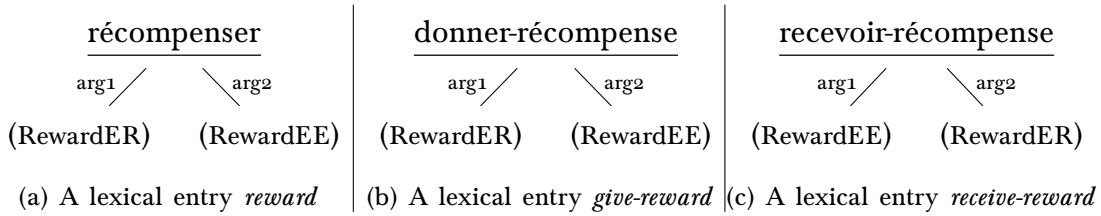


Figure 5.29: The GTAG lexical entry récompenser

of REWARD, the variable nodes RewardER (linked to *récompenser* with arg1) and RewardEE (linked to *récompenser* with arg2) stand for *lexicalizations* of RewardER and RewardEE, respectively.

A concept may have several lexicalizations. For instance, in addition to récompenser, REWARD has the lexicalizations shown in Figure 5.30.



(a) A lexical entry *reward* (b) A lexical entry *give-reward* (c) A lexical entry *receive-reward*

Figure 5.30: The lexical entries linked with REWARD

5.2.3.2 Morpho-Syntactic Realizations of a Lexical Entry

A lexical entry can be used in various syntactic constructions. For instance, one can use the transitive verb récompenser (to reward) in both the passive and active voice constructions. To distinguish the various syntactic uses of a lexical entry, G-TAG employs T-features. For example, to denote the passive voice usage of récompenser, one attaches +[T-passive] to récompenser. By default, no T-feature means the active voice construction with a verb and it is considered to be the *canonical* construction with the verb. One can combine T-features to define other syntactic constructions. For instance, Figure 5.31 depicts the canonical construction with récompenser, one with the feature +[T-passive], and one with the combination of +[T-passive] and +[T-reduced-conj]. By convention, one attaches T-features to the root node (the constant node) of the tree representation of a lexical entry. G-TAG, being inspired by TAG, encodes syntactic constructions with *elementary trees.* That is, each usage of a lexical entry is realized with the help of an elementary tree. Hence, every combination (set) of T-features (including the empty combination) that defines a syntactic usage of a lexical entry is associated with an elementary tree.

However, a set of T-features only defines a syntactic construction with the lexical entry. They do not to concern morphological information. G-TAG encodes the morphological information in *morphological features.* They are also attached to the constant node of the underspecified g-derivation tree representation of a lexical entry. For instance, as Figure 5.31 illustrates, the feature {tense=pass.comp} decorates the constant node of the

trees. It indicates that the lexical unit is used in the *passé composé* tense.[48]

More formally, one associates with a lexical entry $\underline{e}$ a set of elementary trees, $\{e_0, \ldots, e_k\}$. The first of the elementary trees $e_0, \ldots, e_k$, i.e., e_0 is the canonical representative of $\underline{e}$. Each of the trees e_i, for $i = 1, \ldots, k$, is obtained by adding a unique set of T-features to the canonical representative e_0. Each e_i, for $i = 0, \ldots, k$, apart from syntactic information encoded as T-features, is annotated with morphological features. For instance, trees in Figure 5.31(a), Figure 5.31(b), and Figure 5.31(c) stand for $\underline{\text{récompenser}}_0$, $\underline{\text{récompenser}}_1$, and $\underline{\text{récompenser}}_2$, respectively. They denote the elementary trees shown in Figure 5.32. Although each of these elementary trees is anchored with the lexical entry, which is a lemma, the mother node of the anchor bears the morphological information that indicates what the inflected from of the anchor is. The post processing module uses this morphological information to compute the inflected form of the anchor.

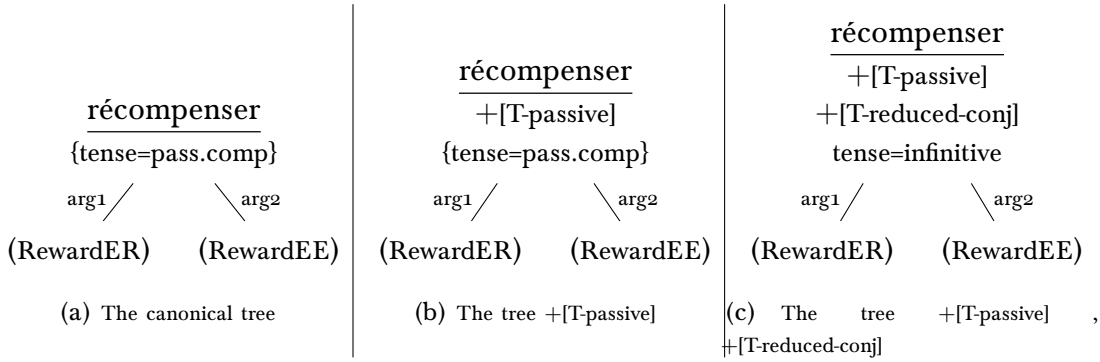
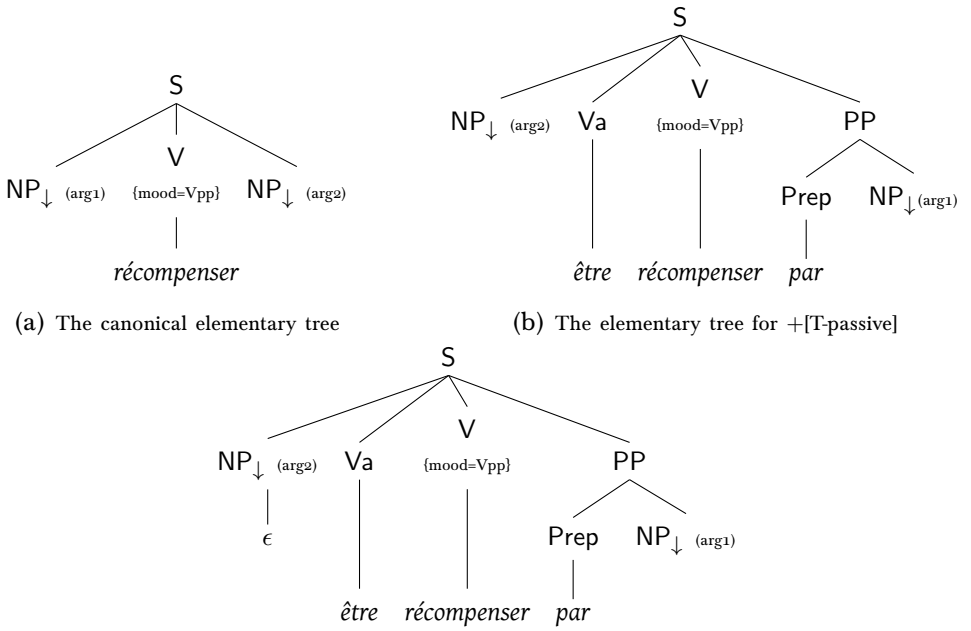


Figure 5.31: The underspecified g-derivation tree $\underline{\text{récompenser}}$ decorated with various sets of T-features and morphological features

As we already mentioned, a lexical entry represented as an underspecified g-derivation tree may have variable nodes, which are connected with the constant node with the thematic roles. Adding T-features and/or morphological features to the constant node affect neither variable nodes nor thematic roles. For instance, in Figure 5.31, the correspondence between variable nodes and thematic roles is the same in all three trees. Let $\underline{e}$ be a lexical entry, and e_i be one of the trees obtained out of $\underline{e}$ by adding T-features and/or morphological features. In the elementary tree corresponding to e_i, the variable nodes of e_i (i.e., of $\underline{e}$) correspond to the *substitution* sites. We mark a substitution site corresponding to a variable node with the same thematic role that connects that variable node to the constant node in the underspecified g-derivation tree.

Remark 5.4. *Instead of a single underspecified g-derivation tree associated with a lexical entry, we prefer to have as many underspecified g-derivation trees as the elementary trees that serve as the possible syntactic realizations of the lexical entry. Indeed, each elementary tree is defined by a unique set of T-features, which decorates the constant node of the underspecified g-derivation*

[48]Syntactically, the passé composé tense of French corresponds to the present perfect tense of English. However, from the linguistic usage point of view, passé composé is rather reminiscent of the past simple tense of English.



(a) The canonical elementary tree

(b) The elementary tree for +[T-passive]

(c) The elementary tree for +[T-passive, T-reduced-conj]

Figure 5.32: G-TAG elementary tree corresponding to the underspecified g-derivation trees of récompenser

tree. If we assume that by adding a set of T-features to the underspecified g-derivation tree, we produce a new *underspecified g-derivation tree, each of the elementary trees will have its own, unique underspecified g-derivation tree. With this assumption, we do not change anything in G-TAG, but now instead of listing T-features in order to signify an elementary tree, we have an underspecified tree denoting that elementary tree. In this way, each of the trees in Figure 5.31 is an underspecified g-derivation tree denoting a single elementary tree.*

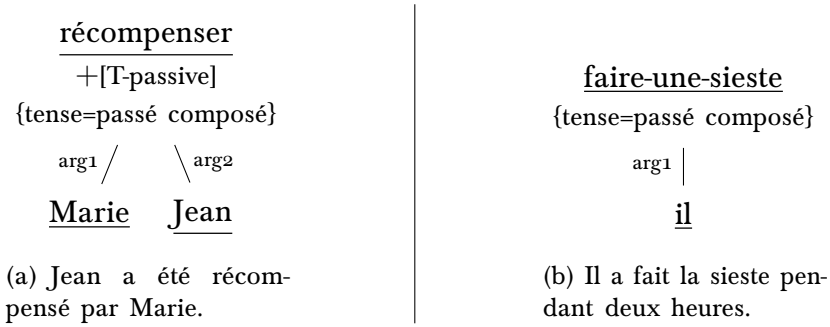
5.2.4 G-derivation and G-derived Trees

One constructs a *g-derivation* tree by lexicalizing concepts from a conceptual representation input. A lexicalization of a concept is an underspecified g-derivation tree (with T-features and morphological features), which may have variable nodes. One instantiates the variable nodes with the lexicalizations of the arguments of a concept. By recursively instantiating variable nodes, one obtains a g-derivation tree whose nodes are lexical entries decorated with T-features and morphological features.

(82) *Jean a été récompensé par Marie.*
John have$_{\text{3PS. SG. PRS.}}$ to-be$_{\text{PAST PART.}}$ reward$_{\text{PAST PART.}}$ by Mary.
'John was rewarded by Mary.'

(83) *Il a fait une sieste.*
John have$_{\text{3PS. SG. PRS.}}$ make$_{\text{PAST PART.}}$ a nap.
'John took a nap.'

For instance, Figure 5.33(a) and Figure 5.33(b) show the g-derivation trees of the sentences (82) and (83), respectively.



(a) Jean a été récompensé par Marie.

(b) Il a fait la sieste pendant deux heures.

Figure 5.33: G-derivation trees

As one may notice, the representation of a g-derivation tree differs from the representation of TAG derivation trees. In particular, while in a TAG derivation tree, a node represents an elementary tree whose anchor is inflected, in a g-derivation tree, a node represents a lexical entry annotated with T-features and morphological features. In TAG, one uses Gorn addresses in order to represent the information where a substitution or adjunction takes place. Instead of using Gorn addresses, G-TAG uses thematic roles. As Danlos (1998) suggests, one can view a g-derivation tree as a semantic dependency tree whose nodes are annotated with features encoding the morpho-syntactic information. On the other hand, g-derivation trees and TAG derivation trees are conceptually very close to each other. Both are trees. Both record how to combine elementary trees. Both give rise to a unique derived tree. For instance, the g-derivation tree shown in Figure 5.33(a) determines a g-derived tree of a passive construction with by-agent. Figure 5.34 shows that g-derived tree.

In a g-derived tree lemmas label frontier nodes; the morphological information how to inflect these lemmas are provided by their mother nodes. As we already mentioned, the post processing module computes inflected forms of lemmas.

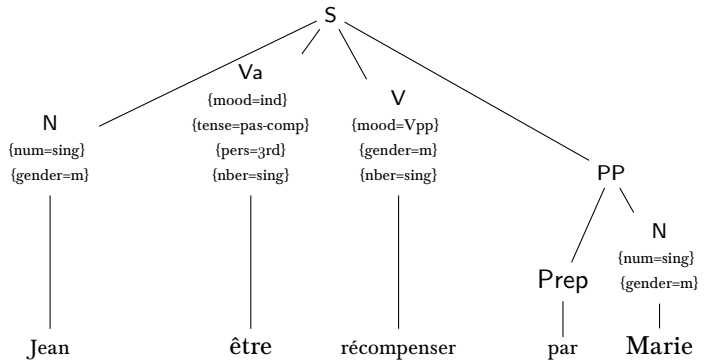


Figure 5.34: A g-derived tree

Remark 5.5. *The main difference between g-derivation and TAG derivation trees is that TAG derivation trees do not make use of variable nodes. That is, TAG derivation trees are* complete, *whereas underspecified g-derivation trees are incomplete by definition.*

5.2.5 Discourse Grammar

The G-TAG discourse grammar consists of entries for adverbial connectives and subordinate conjunctions. *Any discourse connective*, either an adverbial or a subordinate conjunction, anchors an *S-initial tree with two S-substitution sites.*

The G-TAG discourse grammar enables one to generate multi-sentential texts. For instance, one can generate the texts given in Examples (84)-(87). In each of these examples, the texts are generated from the same conceptual representation input.

(84) a. *Jean a passé l'aspirateur. Ensuite, il a fait une sieste.*
John have$_{\text{3PS. SG. PRS.}}$ pass$_{\text{PAST PART.}}$ vacuumer$_{\text{DEF.}}$. Afterwards, he have$_{\text{3PS. SG. PRS.}}$ make$_{\text{PAST PART.}}$ a nap.

'John vacuumed. Afterwards, he took a nap.'

b. *Jean a fait une sieste. Auparavant, il avait passé l'aspirateur.*
John have$_{\text{3PS. SG. PRS.}}$ make$_{\text{PAST PART.}}$ a nap. Beforehand, he have$_{\text{3PS. SG. IMPERF.}}$ pass$_{\text{PAST PART.}}$ vacuumer$_{\text{DEF.}}$.

'John took a nap. Beforehand, he had vacuumed.'

(85) a. *Jean a passé l'aspirateur avant que Marie fasse une sieste.*
John have$_{\text{3PS. SG. PRS.}}$ pass$_{\text{PAST PART.}}$ vacuumer$_{\text{DEF.}}$ before that Mary make$_{\text{SUBJUNCTIVE}}$ a nap

'John vacuumed before Mary took a nap.'

b. *Marie a fait une sieste après que Jean a passé l'aspirateur.*
Mary have$_{\text{3PS. SG. PRS.}}$ make$_{\text{PAST PART.}}$ a nap after that John have$_{\text{3PS. SG. PRS.}}$ pass$_{\text{PAST PART.}}$ vacuumer$_{\text{DEF.}}$

'Mary took a nap after that John vacuumed.'

(86) a. *Jean a passé l'aspirateur avant de faire une sieste.*
John have$_{\text{3PS. SG. PRS.}}$ pass$_{\text{PAST PART.}}$ vacuumer$_{\text{DEF.}}$ before of make$_{\text{INF.}}$ a nap

'John vacuumed before taking a nap.'

b. *Jean a fait une sieste après avoir passé l'aspirateur.*
John have$_{\text{3PS. SG. PRS.}}$ make$_{\text{PAST PART.}}$ a nap after have$_{\text{INF.}}$ pass$_{\text{PAST PART.}}$ vacuumer$_{\text{DEF.}}$.

'John took a nap after vacuuming.'

(87) a. *Jean a passé l'aspirateur pour que Marie le recompense.*
John have$_{\text{3PS. SG. PRS.}}$ pass$_{\text{PAST PART.}}$ vaccumer$_{\text{DEF.}}$ for that Marie him reward$_{\text{SUBJUNCTIVE}}$
'John vacuumed in order to be rewarded by Mary.'

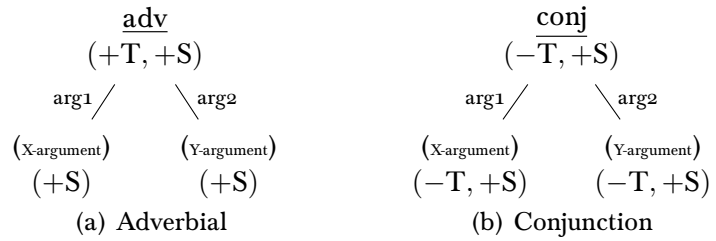
b. *Jean a passé l'aspirateur pour être récompensé par Marie.*
John have$_{\text{3PS. SG. PRS.}}$ pass$_{\text{PAST PART.}}$ vaccumer$_{\text{DEF.}}$ for to-be$_{\text{INF.}}$ reward$_{\text{PAST PART.}}$ by Mary.
'John vacuumed in order to be rewarded by Mary.'

To distinguish texts, sentences, and noun phrases from each other, G-TAG employs an additional set of features to decorate nodes of g-derivation trees. Table 5.1 provides intended meanings of these features.

Features	Intended Meanings
$(+\text{T}, +\text{S})$	a text - two or more sentences
$(-\text{T}, +\text{S})$	a sentence
$(+\text{S})$	either a sentence or a text
$(-\text{T}, -\text{S})$	an NP
$(-\text{T})$	either a sentence or an NP

Table 5.1: G-TAG features denoting a text, a sentence, either a text or a sentence, etc.

Figure 5.35 depicts the lexical entries of an adverbial and a subordinate conjunction. The root of the lexical entry of the adverbial has a feature $(+\text{T}, +\text{S})$ to indicate that it gives rise to a text (two or more sentences). The root of the lexical entry of the subordinate conjunction has a feature $(-\text{T}, -\text{S})$ to indicate that it gives rise to a single sentence. Since the lexical entries of adverbials and conjunctions differ, we discuss them separately.



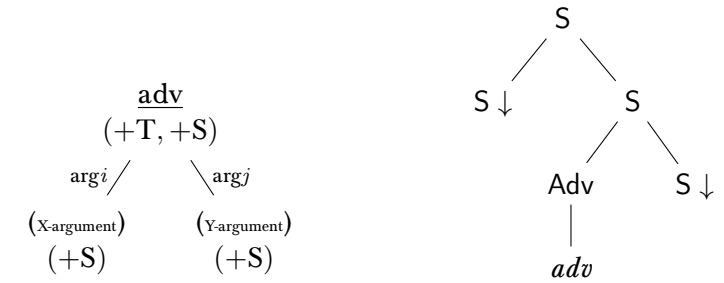
(a) Adverbial (b) Conjunction

Figure 5.35: Lexical entries of adverbials and conjunctions

5.2.5.1 Adverbials

G-TAG employs lexical entries of adverbials (underspecified g-derivation trees whose roots are adverbials) in order to generate multi-sentential texts.

As Figure 5.36(a) shows, the root of the underspecified g-derivation tree of an adverbial has the feature $(+\text{T}, +\text{S})$ and its variable nodes have the feature $(+\text{S})$, denoting either a text or a sentence. In the corresponding elementary tree in Figure 5.36(b), the variable nodes are realized as S-substitution sites. For instance, Figure 5.37 illustrates the

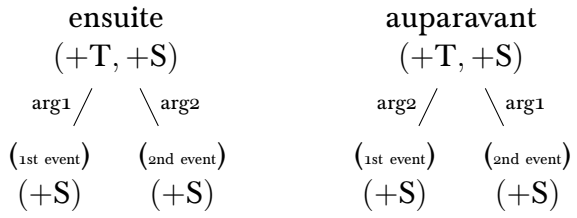


(a) An underspecified g-derivation tree for adv

(b) An S-initial tree anchored with an adverbial

Figure 5.36: A lexical entry of an adverbial and a corresponding elementary tree

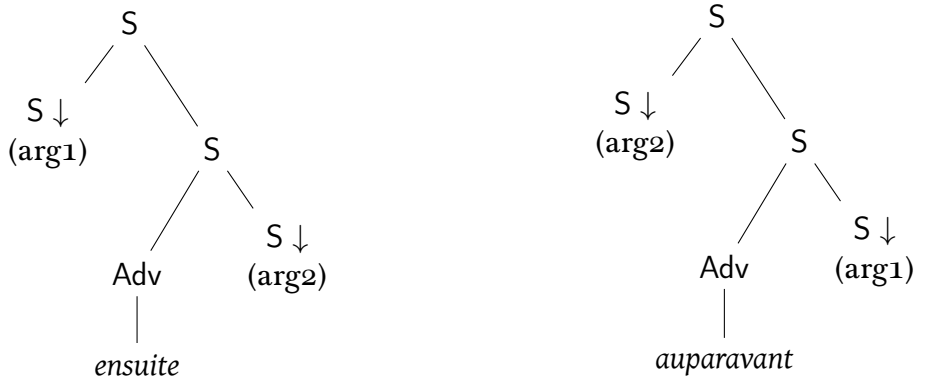
underspecified g-derivation trees of the lexical entries of the adverbials *ensuite* (afterward) and *auparavant* (beforehand). Both of them are lexicalizations of SUCCESSION. Both have the same feature sets. The difference between them is that their thematic roles arg1 and arg2 are reversed, i.e., arg1 of ensuite (afterward) is arg2 of auparavant (beforehand); and arg2 of ensuite (afterward) is arg1 of auparavant (beforehand). Figure 5.38 shows the elementary trees anchored with *ensuite* and *auparavant*.



(a) *ensuite*

(b) *auparavant*

Figure 5.37: Underspecified g-derivation trees for adverbials *ensuite* and *auparavant*



(a) An elementary tree anchored with *ensuite*

(b) An elementary tree anchored with *auparavant*

Figure 5.38: Elementary trees anchored with adverbials *ensuite* and *auparavant*

5.2.5.2 Subordinate Conjunctions

A lexical entry of a subordinate conjunction consists of several underspecified g-derivation trees. These underspecified g-derivation trees define various syntactic constructions involving a subordinate conjunction. There are two essentially different syntactic uses of a subordinate conjunction that G-TAG offers. We call these two cases the *canonical* use and the *reduced* one. To show the differences between them, let us consider the sentences (85)(a) and (86)(a) on page 164, repeated as follows:

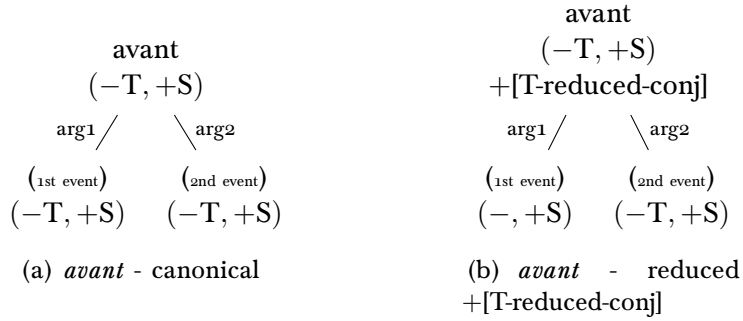
[(85)(a), repeated] *Jean a passé l'aspirateur avant que Marie fasse une sieste.*
John have$_{\text{3PS. SG. PRS.}}$ pass$_{\text{PAST PART.}}$ vacuumer$_{\text{DEF.}}$ before that Mary make$_{\text{SUBJUNCTIVE}}$ a nap

'John vacuumed before Mary took a nap.'

[(86)(a), repeated] *Jean a passé l'aspirateur avant de faire une sieste.*
John have$_{\text{3PS. SG. PRS.}}$ pass$_{\text{PAST PART.}}$ vacuumer$_{\text{DEF.}}$ before of make$_{\text{INF.}}$ a nap

'John vacuumed before taking a nap.'

Each of the sentences (85)(a) and (86)(a) consists of two clauses connected by the subordinate conjunction *avant* (before). To be able to generate sentences with the syntactic constructions similar to ones in (85)(a) and (86)(a), G-TAG uses two underspecified g-derivation trees, shown in Figure 5.39.



(a) *avant* - canonical

(b) *avant* - reduced +[T-reduced-conj]

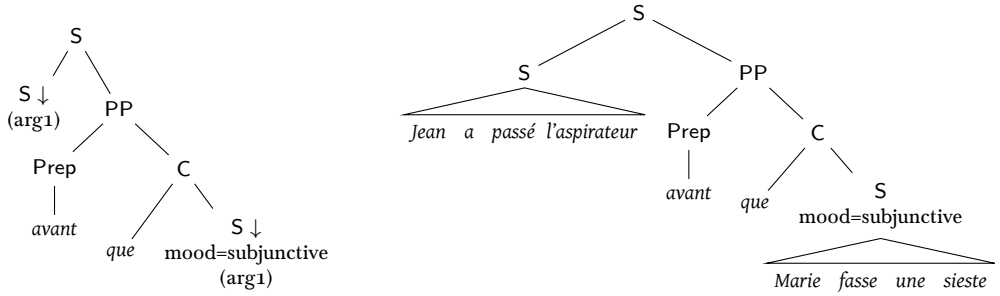
Figure 5.39: Underspecified g-derivation trees of a conjunction

5.2.5.2.1 Canonical

Figure 5.40 on the next page presents the G-TAG canonical initial tree anchored with *avant* and its usage in the sentence (85)(a). It is considered as a canonical construction with *avant* as it connects two *finite* clauses, that is, clauses whose main predicates are finite verb forms. In other words, in the canonical construction with the subordinate conjunction, both the matrix clause and the subordinated one are finite clauses.

5.2.5.2.2 Reduced

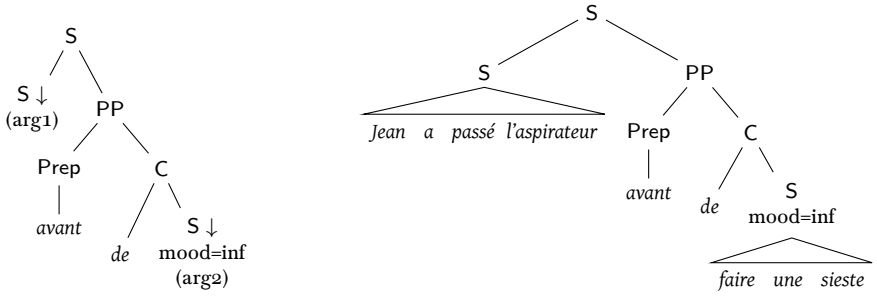
Figure 5.41 shows another initial tree anchored with *avant*, which one uses in order to generate sentences such as (86)(a). In this case, the subordinate conjunction *avant* connects a *finite* clause, which is the matrix clause, and a *reduced* clause, which is the subordinate one. In French, the reduced clause is expressed as an infinitive clause. Since an infinitive clause does not have an overt subject, it shares the subject with the matrix clause. We call this fact *argument-sharing*. Thus, to generate a text with a reduced conjunction, the conceptual representation input has to meet certain requirements. In particular, given some ψ-term $\texttt{RConj}[\texttt{l}_1 \Rightarrow \texttt{A},\ \texttt{l}_2 \Rightarrow \texttt{B}]$, where RConj is a SECOND ORDER RELATION, to lexicalize RConj with a reduced subordinate conjunction, A and B should have the same *argument*, i.e., A and B should have the *argument-sharing* property.



(a) The canonical elementary tree of *avant*

(b) The G-TAG analysis

Figure 5.40: The G-TAG analysis of a sentence with the canonical conjunction



(a) A non-canonical elementary tree of *avant*, defined by +[T-reduced-conj]

(b) The G-TAG analysis

Figure 5.41: The G-TAG analysis of a sentence with a reduced conjunction

5.2.6 An Example of Text Generation

G-TAG generates a g-derivation tree from a conceptual representation input by lexicalizing concepts in the following order: SECOND ORDER RELATIONS, then the FIRST

ORDER RELATIONS, and finally, THINGS. The hierarchal order of lexicalization from SECOND ORDER RELATIONS to THINGS is motivated by a hypothesis that SECOND ORDER RELATIONS define the rhetorical (discourse) structure of a text. This hypothesis relates to an argument that by selecting a lexicalization of a SECOND ORDER RELATION, one imposes certain restrictions on the possible lexicalizations of its arguments, which are either SECOND ORDER RELATION and/or FIRST ORDER RELATIONS. By lexicalizing the FIRST ORDER RELATIONS, one has fewer options for selecting lexicalizations of THINGS.[49]

$$
\begin{aligned}
\mathtt{E}_0 &=: \mathtt{SUCCESSION}[1^{\mathtt{st}}\mathtt{EVENT} \Rightarrow \mathtt{E}_1,\ 2^{\mathtt{nd}}\mathtt{EVENT} \Rightarrow \mathtt{E}_2] \\
\mathtt{E}_1 &=: \mathtt{GOAL}[\mathtt{Action} \Rightarrow \mathtt{E}_{11},\ \mathtt{Purpose} \Rightarrow \mathtt{E}_{12}] \\
\mathtt{E}_2 &=: \mathtt{NAPPING}[\mathtt{NAPPER} \Rightarrow \mathtt{H}_1] \\
\mathtt{E}_{11} &=: \mathtt{VACUUMING}[\mathtt{VACUUMER} \Rightarrow \mathtt{H}_1] \\
\mathtt{E}_{12} &=: \mathtt{REWARDING}[\mathtt{REWARDER} \Rightarrow \mathtt{H}_2,\ \mathtt{REWARDEE} \Rightarrow \mathtt{H}_1] \\
\mathtt{H}_1 &=: \mathtt{HUMAN}[\mathtt{NAME} \Rightarrow \mathtt{Jean},\ \mathtt{gender} \Rightarrow \mathtt{masc}] \\
\mathtt{H}_2 &=: \mathtt{HUMAN}[\mathtt{NAME} \Rightarrow \mathtt{Marie},\ \mathtt{gender} \Rightarrow \mathtt{fem}]
\end{aligned}
$$

Figure 5.42: conceptual representation input

Let us build a g-derivation tree from the conceptual input described in Figure 5.42. One starts to construct the g-derivation tree by lexicalizing the ψ-term $\mathtt{E}_0$ since no other ψ-term from the conceptual input contains $\mathtt{E}_0$ as a sub-ψ-term (see Section 5.2.2.3). To lexicalize SUCCESSION concept, which is the root symbol of $\mathtt{E}_0$, one has to choose among several lexical entries that are linked with SUCCESSION. They are as follows: ensuite (afterwards), auparavant (beforehand), après (after) and avant (before).

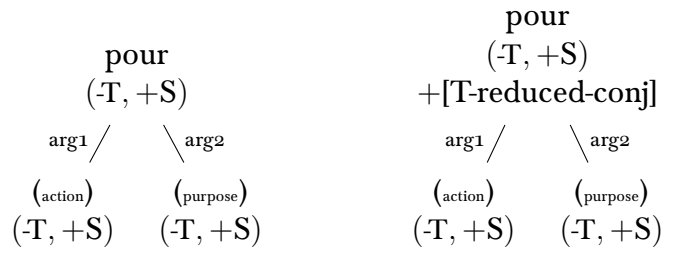
Let us assume that G-TAG selects ensuite (afterwards), which has only one underspecified g-derivation tree shown in Figure 5.38(a) on page 166. One has to lexicalize the arguments of SUCCESSION in $\mathtt{E}_0$, i.e., the sub-ψ-terms of $\mathtt{E}_0$ under the labels $1^{\mathtt{st}}$EVENT and $2^{\mathtt{nd}}$EVENT. $\mathtt{E}_0$ has the sub-ψ-term $\mathtt{E}_1$ under the label $1^{\mathtt{st}}$EVENT and $\mathtt{E}_2$ under the label $2^{\mathtt{nd}}$EVENT. To decide which one to lexicalize first, $\mathtt{E}_1$ or $\mathtt{E}_2$, G-TAG refers to their order in the text that G-TAG aims to produce. $\mathtt{E}_1$ will be lexicalized as arg1 of *ensuite* and $\mathtt{E}_2$ as arg2. As the elementary tree anchored with *ensuite* indicates, arg1 precedes arg2 in the surface order. Hence, G-TAG lexicalize $\mathtt{E}_1$ before $\mathtt{E}_2$.

To lexicalize $\mathtt{E}_1 =: \mathtt{GOAL}[\mathtt{Action} \Rightarrow \mathtt{E}_{11},\ \mathtt{Purpose} \Rightarrow \mathtt{E}_{12}]$, G-TAG selects the lexical entry pour (for) as a lexicalization its root symbol, which is GOAL. The lexical entry pour has two underspecified g-derivation trees depicted in Figure 5.43 on the following page. One in Figure 5.43(a) on the next page is the canonical tree of pour, whereas the other one (in Figure 5.43(b)) is non-canonical, namely, the *reduced* one. We can assume that G-TAG opts for the reduced underspecified g-derivation of *pour*. It is possible to do so[50] because $\mathtt{E}_{11}$ and $\mathtt{E}_{12}$ share an argument ($\mathtt{H}_1$).

To continue the lexicalization of $\mathtt{E}_1$, we should lexicalize its arguments as well. Thus, we lexicalize $\mathtt{E}_{11}$ and $\mathtt{E}_{12}$.

[49]Until all RELATIONS are not lexicalized, G-TAG does not start lexicalizing THINGS, due to the pronominalization issues, which we do not discuss for the sake of simplicity.

[50]In French, from stylistic points of view, the reduced construction (when possible) is preferred over the canonical one (Danlos, 2000).



(a) *pour* - canonical

(b) *pour* - reduced +[T-reduced-conj]

Figure 5.43: Underspecified g-derivation trees of pour

To lexicalize E_{11}, we lexicalize its root symbol, which is VACUUMING. In G-TAG, the VACUUMING concept is linked with the lexical entry passer-l'aspirateur (to vacuum), whose underspecified g-derivation tree is shown in Figure 5.44. H_1, which is the value of E_{11} under VACUUMER, is going to be lexicalized as arg1 of passer-l'aspirateur. In the elementary tree of passer-l'aspirateur, arg1 occurs at the subject position. Thus, the lexicalization of H_1 will be a lexicalization of the subject of the elementary tree of passer-l'aspirateur.

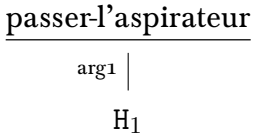


Figure 5.44: The lexicalization of E_{12}.

In order to lexicalize $E_{12} :=$ REWARDING[REWARDER $\Rightarrow H_2$, REWARDEE $\Rightarrow H_1$], we have a constraint to obey. In particular, the lexicalization of E_{12} has to be an infinitive clause, because we use it as the subordinate clause of the reduced construction with *pour*, which requires an infinitive clause (reduced clause). G-TAG cannot lexicalize E_{12} just as an infinitive clause but under the constraint that E_{12} and E_{11} share H_1. Since H_1 is the subject of the lexicalization of E_{11} (of passer-l'aspirateur), the thematic role of H_1 also must be *subject* for the lexicalization of E_{12}. The root symbol of E_{12} is REWARDING. To lexicalize E_{12}, G-TAG selects lexical entries linked with REWARDING:

- récompense (reward),
- donner-récompense (give-reward),
- recevoir-récompense (receive-reward).

Let us examine if these lexical entries, illustrated in Figure 5.45 on the facing page, can have H_1 as the subject.

- In the case of donner-récompense (see Figure 5.45(b)), REWARDEE is arg2, i.e., H_1 will be lexicalized as arg2. arg2 could have been the subject if one could use donner-récompense in a passive construction, but it is not possible as a passive construction for donner-récompense does not exist in French.

- For recevoir-récompense shown in Figure 5.45(c), REWARDEE, i.e., H_1 is arg1. arg1 is the subject of the elementary canonical of recevoir-récompense. Thus, recevoir-récompense is an option for lexicalizing E_{12}.
- In the case of récompenser (see Figure 5.45(a)), the canonical elementary has arg2 (corresponding to REWARDEE) as the object, but not as the subject. arg2 becomes the subject in the elementary trees of récompenser corresponding to the passive voice constructions.

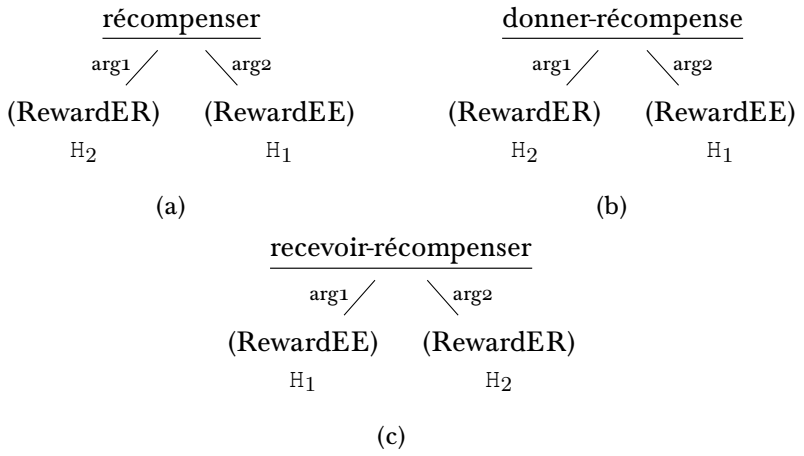


Figure 5.45: The candidates of lexicalization of REWARDING

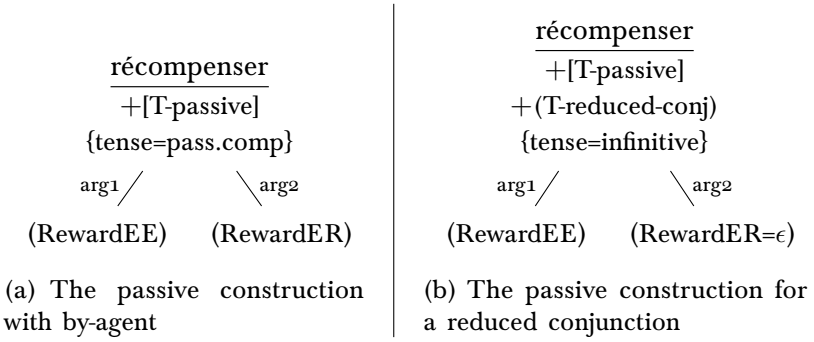
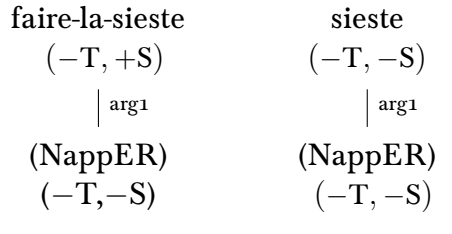


Figure 5.46: Underspecified g-derivation trees for passive constructions

Thus, there are two options to choose from: (1) recevoir-récompenser and (2) a passive construction with récompenser. For the sake of illustration of an usage of a passive voice construction, we assume that G-TAG chooses a passive construction with récompenser. As we already saw, there are several underspecified g-derivation trees for passive constructions (see Figure 5.46). Among them G-TAG selects the one with feature +[T-reduced-conj] (Figure 5.46(b)) since G-TAG has to construct a reduced (infinitive) clause. This underspecified g-derivation tree defines the elementary tree shown in Figure 5.32(c) on page 162.

Now, G-TAG moves to the lexicalization of E_2, whose root symbol is NAPPING. Figure 5.47 on the following page shows two underspecified g-derivation-trees serving as

lexicalizations of NAPPING. Between the two, G-TAG selects the one with the feature $(-\text{T}, +\text{S})$ (see Figure 5.47(a)) because the underspecified tree for ensuite requires its arguments to be either a text or a sentence.



(a) A sentence (b) An NP

Figure 5.47: Underspecified g-derivation-trees serving as lexicalizations of NAPPING

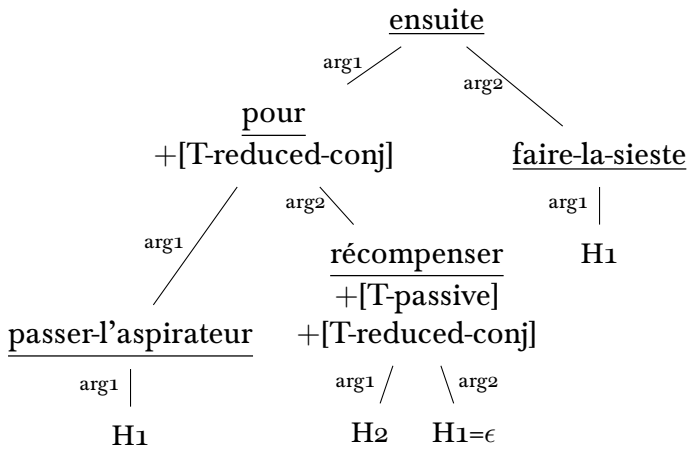


Figure 5.48: A g-derivation tree

Since all the concepts from the RELATION category are lexicalized, G-TAG outputs an incomplete g-derivation tree as it misses lexicalizations of THINGs (see Figure 5.49). To lexicalize THINGs, G-TAG employs the *pronominalization* module. The pronominalization module lexicalizes concepts from THINGs either as noun phrases or as pronouns. The exact modus operandi of the G-TAG pronominalization is not relevant for the current purposes. We can assume that the pronominalization module lexicalizes the first (from left to right in the g-derivation tree in Figure 5.49) occurrence of H_1 as Jean, whereas the second occurrence is lexicalized as ϵ due to the feature +[T-reduced-conj] in the underspecified g-derivation tree of récompenser. The third occurrence of H_1 is lexicalized as il (heNOM). The single occurrence of H_2 is lexicalized as Marie. As a result, we obtain the final g-derivation tree depicted in Figure 5.49 on the next page.

G-TAG maps the constructed g-derivation tree its g-derived tree. The post processing module computes the inflected forms of the leaves of the g-derived tree. As a result, one can transform the g-derived tree into a tree such as the one in Figure 5.50, i.e., into a syntactic tree whose leaves are inflected words.

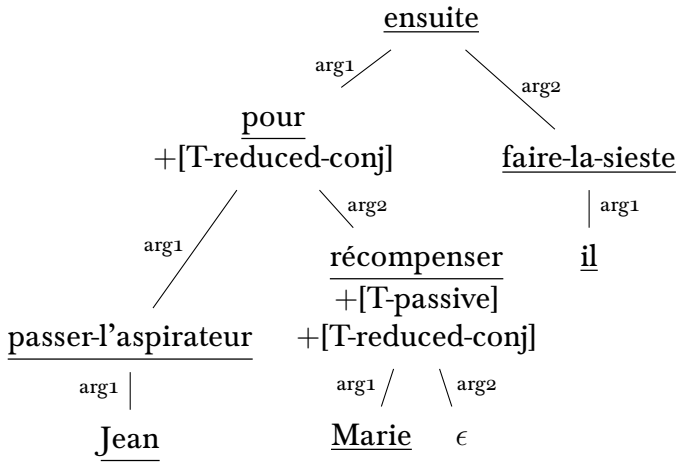


Figure 5.49: The final g-derivation tree

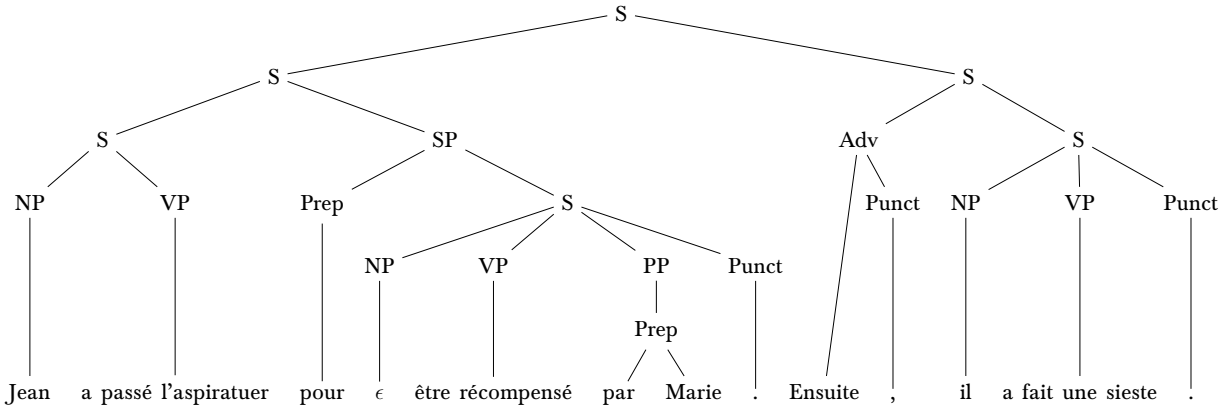


Figure 5.50: A (post-processed) derived tree

The post processing module outputs the following *canonical* surface form, which is the linearization of the tree in Figure 5.50:

(88) *Jean a passé l'aspirateur pour être récompensé par Marie.*
John have$_{\text{3PS. SG. PRS.}}$ pass$_{\text{PAST PART.}}$ vaccumer$_{\text{DEF.}}$ for to-be$_{\text{INF.}}$ reward$_{\text{PAST PART.}}$ by Mary.
Ensuite, il a fait une sieste.
Afterwards, he have$_{\text{3PS. SG. PRS.}}$ make$_{\text{PAST PART.}}$ a nap.

'John vacuumed in order to be rewarded by Mary. Afterwards, he took a nap.'

5.3 D-STAG

D-STAG was proposed by Danlos (2009, 2011) in order to address the problem of the syntax-semantic interface for discourse (the syntax-discourse interface). D-STAG is based on the principles of Synchronous TAG (STAG) (Shieber and Schabes, 1990) and SDRT (Asher and Lascarides, 2003). Being motivated by the SDRT discourse analysis, D-STAG offers a discourse grammar capable of producing discourse structures that cannot be represented as trees but directed acyclic graphs (DAGs). For example, Figure 5.51 shows the D-STAG interpretations of the following discourses:

(89) [Fred is grumpy]$_0$ because [he lost his keys]$_1$. Moreover, [he failed his exam]$_2$.

(90) [Fred is grumpy]$_0$ because [he didn't sleep well]$_3$. [He had nightmares]$_4$.

(91) [Fred went to the supermarket]$_5$ because [his fridge was empty]$_6$. Then, [he went to the movies]$_7$.

(92) [Fred is grumpy]$_0$ because [his wife is away this week]$_8$. [This shows how much he loves her]$_9$.

Figure 5.51(a) and Figure 5.51(d) show the trees that serve as the D-STAG interpretations of the discourses (89) and (92), respectively. Unlike them, Figure 5.51(b) and Figure 5.51(c) illustrate the D-STAG interpretations of the discourses (90) and (91), respectively. As one can see, the latter interpretations are *multi-parent*, i.e., non-tree shaped DAGs.

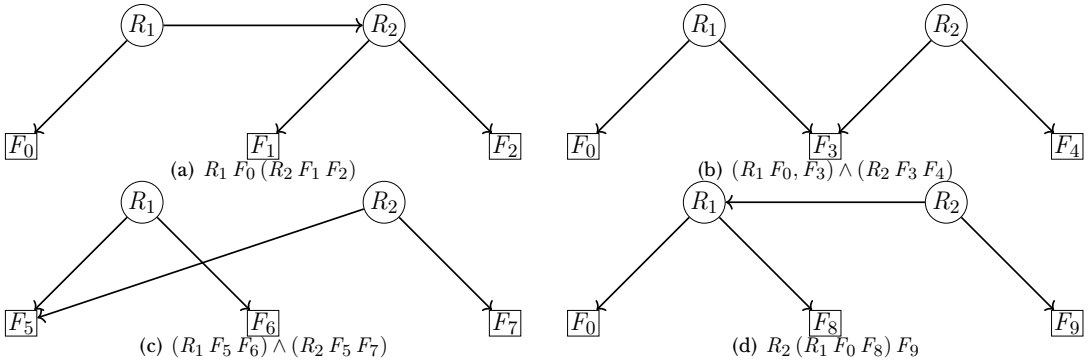


Figure 5.51: DAGs as discourse structures

We already provided terminology and constraints that D-STAG makes use of in Section 4.1.3. For the sake of convenience, we repeat them here.

1. The clause where a discourse connective appears is called the *host clause* of the discourse connective.
   - A subordinate conjunction always appears in front of its host clause. The host clause of the subordinate conjunction is called an *adverbial clause.*
   - An adverbial connective may either appear in front of its host clause or within its verb phrase (i.e., at a clause-medial position).
2. The matrix clause of a subordinate conjunction is on the right of the adverbial clause. In this case, the subordinate conjunction is called *postposed.*
3. The matrix clause is on the left of the adverbial clause, or inside the adverbial clause (before the VP of the adverbial clause). In this case, the subordinate conjunction is called *preposed.*

The arguments of a discourse connective/relation are the syntactic/semantic representations of the *host* and the *mate* segments of the discourse connective/relation, which obey the following constraints (Danlos, 2011):

Constraint 1: The host segment of a connective is identical to or starts at its host clause (possibly crossing a sentence boundary).

Constraint 2: The mate segment of an adverbial is anywhere on the left of its host segment (generally crossing a sentence boundary).

Constraint 3: The mate segment of a postposed conjunction is on the left of its host segment without crossing a sentence boundary.

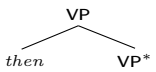
Constraint 4: The mate segment of a preposed conjunction is identical to or starts at the matrix clause (possibly crossing a sentence boundary).

5.3.1 Discourse Normalized Form

D-STAG defines the notion of a Discourse Normalized Form (DNF). DNF is a sequence of *discourse words*, where a discourse word is either a connective, or an identifier of a clause (without a connective), or a punctuation sign. For instance, the DNF of (93) is C_0 because C_1.

(93) [Fred is grumpy]$_0$ because [he has failed an exam]$_1$.

(94) [Fred went to the movies]$_2$. [He *then* went to a bar]$_3$.

One of the main motivations for introducing DNF is to identify the host clause, and moreover to compute the host segment of a discourse connective. As we saw in Section 4.1, to compute the host segments of a connective is a nontrivial task if attitude (report) verbs are involved. However, even if no attitude verb is involved in a discourse but some clause-medial adverbial connective appears in the discourse, by means of a pure syntactic analysis, one may not be able to identify the host segment of the adverbial connective. For instance, in (94), the LTAG analysis of the adverbial *then* is 

, i.e., it serves as a VP-modifier. Since at the discourse-level, an argument of *then* cannot be an VP, *went to a bar* cannot serve as an argument of the discourse relation signaled by *then.* To declare that an argument of *then* is *he went*

to a bar, D-STAG explicitly encodes it in the DNF. To achieve that, D-STAG moves *then* in front of the clause. Thus, one obtains the following DNF: $C_2 . \mathsf{then}^{internal}\ C_3$, where the superscript *internal* indicates that the connective *then* was moved from its original position. C_3 denotes the host clause of *then*, which, in this case, serves as the host segment to the discourse connective *then*. Moving of an adverbial connective from a clause-medial to a clause-initial position is called *normalization* of a clause. By normalizing the clauses where connectives appear at clause-medial positions, one can construct the DNF of the text. Consequently, in a DNF, each connective appears in front of its host clause.

In a case of a discourse where no connective heads a normalized clause C, one places ϵ (the lexically unexpressed connective) in the DNF in front of the clause C. For instance, $C_3\, \epsilon\, C_4$ serves as the DNF of the following discourse:

(95) [Max fell]$_3$. [Fred pushed him]$_4$.

A DNF of a discourse without a preposed conjunction, i.e., only with adverbial connectives and/or postposed conjunctions follows the following pattern:

$$C\ (\mathsf{Punct}\ \mathsf{Conn}\ C)^*$$

Where the pair Punct Conn can be either (a) fullstop Adverbial, or (b) Comma Conjunction with Comma being optional. In general, we discard commas and fullstops and write $C_0 \cdots \mathsf{Conn}_n\ C_n$.

(96) When [he was in Paris]$_5$, [Fred went to the Eiffel Tower]$_6$. Next, [he visited the Louvre]$_7$.

The DNF of a discourse with a preposed conjunction follows another pattern. For example, let us consider the discourse (96). The preposed conjunction *when* in (96) is a *framing adverbial*, i.e., it sets a frame for a piece of discourse consisting of several sentences (Charolles, 2005). The discourse starts with the preposed conjunction *when* and its host clause, which also serves as the host segment of *when*. The mate segment of *when* crosses the sentence boundary. In the case of (96), the mate segment of *when* is a piece of discourse consisting of two clauses connected by the discourse connective *next*. The DNF of (96) is $\mathsf{When}\ C_5\ ,\ C_6\ .\ \mathsf{Next}\ C_7$, which, as one cane see, does not follow the pattern $C_0 \cdots \mathsf{Conn}_n\ C_n$.

Remark 5.6. *The notion of a DNF of D-STAG is inspired by the treatment of the discourse with clause-medial connectives in D-LTAG. Indeed, in D-LTAG, to parse a discourse, the clause-medial connectives are mapped to the clause-initial ones. Inserting ϵ as a connective between two structurally adjacent clauses where no lexically expressed connective connects them is also reminiscent of what Discourse Input Generation component of D-LTAG does.*

5.3.2 D-STAG: Synchronous Tree Adjoining Grammar for Discourse

D-STAG is based on STAG (Shieber and Schabes, 1990).[51] Thus, an elementary D-STAG

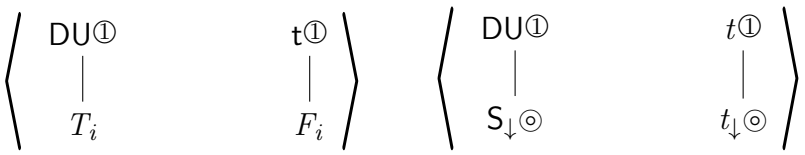
[51]We discussed STAG in Section 2.7 on page 47.

structure α is a tree pair of TAG elementary trees $\langle \alpha_1, \alpha_2 \rangle$, where the substitution and adjunction sites in the trees α_1 and α_2 are linked. The tree α_1 is an elementary tree anchored by either an LTAG derived tree of a clause or a discourse connective; α_2 is an elementary tree anchored with a *semantic* tree, which models the interpretation of either a clause or a discourse connective.

5.3.2.1 Trees Anchored by Clauses

In D-STAG, a minimal (atomic) discourse unit is a clause. One analyzes a discourse consisting of a single clause as a pair whose first component is a DU-rooted tree anchored with a TAG derived tree of the clause and the second component is a tree anchored with a semantic interpretation of the clause. Figure 5.52(a) illustrates such a tree pair, where T_i denotes the S-rooted derived tree of a clause C_i and F_i denotes a t-rooted semantic tree of the clause, where i is a natural number. T_i and F_i trees stand for the sentence-level syntactic and semantic interpretations of the clause C_i. In order to obtain the tree pair in Figure 5.52(a) out of T_i and F_i, D-STAG uses the pair of initial trees shown in Figure 5.52(b).[52] In the first component of the pair in Figure 5.52(b), one substitutes T_i, and in the second component, one substitutes F_i. One denotes a derivation tree of a derived pair shown in Figure 5.52(b) by τ_i.

Convention: Although a clause C does not anchor a tree in D-STAG but rather its derived tree T, we may still write $\overset{\mathsf{DU}①}{\underset{C}{|}}$ instead of $\overset{\mathsf{DU}①}{\underset{T}{|}}$. We will refer to $\overset{\mathsf{DU}①}{\underset{C}{|}}$ as *the tree anchored by the clause* C.



(a) A D-STAG tree pair of a single clause discourse

(b) A D-STAG initial tree pair

Figure 5.52: The D-STAG representation of a clause

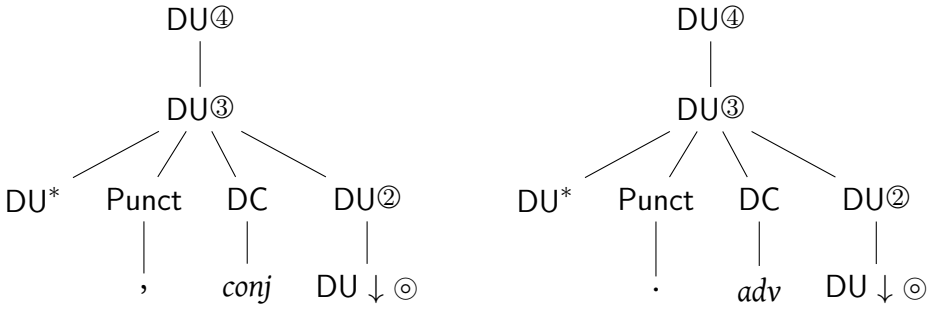
5.3.2.2 Adverbial Connectives and Postposed Conjunctions

Each connective (either an adverbial connective, or a postposed conjunction, or a preposed one) anchors a DU-auxiliary tree with one DU-substitution site. At this substitution site substitutes the tree anchored by the host clause of the connective.

Elementary trees anchored with adverbial connectives and postposed conjunctions have the same structure. Figure 5.53 shows two auxiliary trees, anchored with an adverbial connective and a postposed conjunction. Apart from a single DU-substitution site (marked with ◎), each of these auxiliary trees has three DU-adjunction sites (marked with ②, ③, and ④). Below, we refer to both adverbial connectives and

[52] We remind readers that by convention, we use ⓝ, where n is a positive natural number, to mark adjunction sites. By ◎, we mark a substitution site.

postposed conjunctions as *connectives*, unless otherwise stated. A tree anchored with a clause, $\overset{\text{DU}①}{\underset{C}{|}}$, can substitute at the substitution site of a DU-auxiliary tree anchored with a connective. We obtain a derived tree with four DU-adjunction sites (marked with ①, ②, ③, and ④). Let θ be a tree that adjoins on any of these adjunction sites. The resultant tree would have the same yield as the one that one obtains by adjoining θ at any of the rest of these adjunction sites. In other words, we could have only a single DU-adjunction site instead of four, but we could still generate the same string language. The reason for having different adjunction sites lays in the semantic trees. We first focus on the syntactic part of D-STAG and after that we discuss the semantic one.



(a) A D-STAG tree anchored with a postposed conjunction

(b) A D-STAG tree anchored with an adverbial

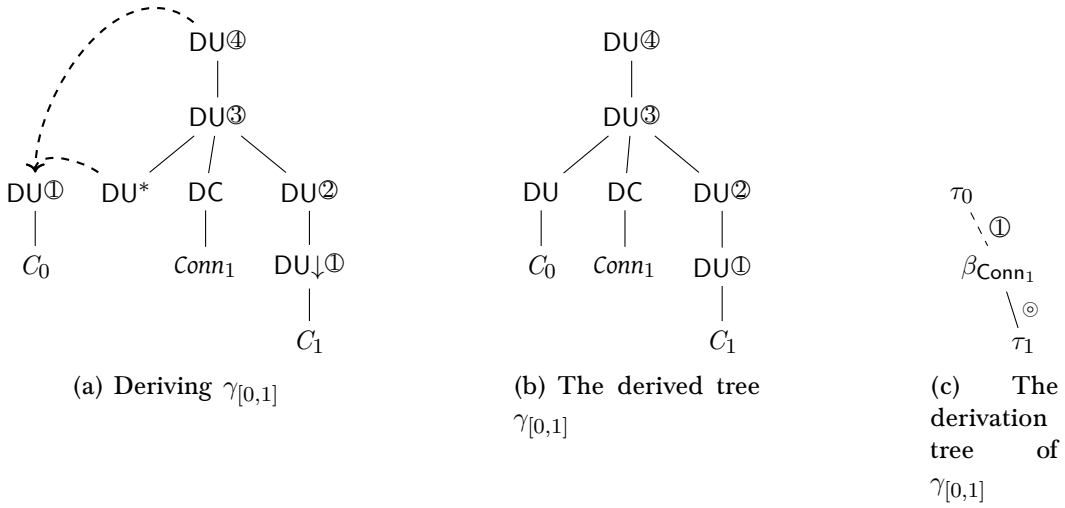
Figure 5.53: D-STAG elementary trees anchored with adverbial & postposed conjunction

5.3.3 The D-STAG Discourse Update and the Right Frontier of a Discourse

A discourse where a discourse connective is ether an adverbial connective or a postposed conjunction has a DNF of the form $C_0\,\mathsf{Conn}_1 \ldots \mathsf{Conn}_n\,C_n$. As D-STAG follows SDRT, to update (extend) the current discourse with a new piece C_{n+1}, one must add C_{n+1} to the current discourse with some discourse (rhetorical) relation R_{n+1}. One assumes that a discourse relation is either expressed by an overt discourse connective or by the empty discourse connective ϵ. Hence, to update a DNF of a discourse, one adds to it C_{n+1} headed by Conn_{n+1} signaling the discourse relation R_{n+1}. To update the current discourse with DNF $C_0\,\mathsf{Conn}_1 \ldots \mathsf{Conn}_n\,C_n$ with a connective-clause pair $\mathsf{Conn}_{n+1}\,C_{n+1}$, D-STAG substitutes $\overset{\text{DU}①}{\underset{C_{n+1}}{|}}$ into the auxiliary tree anchored with Conn_{n+1}, denoted by $\beta_{\mathsf{Conn}_{n+1}}$. The resultant derived tree, denoted by γ_{n+1}, adjoins into the derived tree of the discourse with DNF $C_0\,\mathsf{Conn}_1 \ldots \mathsf{Conn}_n\,C_n$, which we denote by $\gamma_{[0,n]}$. In this way, we derive the tree $\gamma_{[0,n+1]}$, i.e., the derived tree of the extended discourse with DNF $C_0\,\mathsf{Conn}_1 \ldots \mathsf{Conn}_n\,C_n\,\mathsf{Conn}_{n+1}\,C_{n+1}$.

By substituting the tree anchored with C_{n+1} into the auxiliary tree anchored by Conn_{n+1}, R_{n+1} receives the host argument. The discourse relation R_{n+1} obtains the mate argument as a result of adjoining γ_{n+1} in $\gamma_{[0,n]}$. The mate segment is on the left

of the host segment as Constraint 2 and Constraint 3 require (formulated on page 175). In addition, to fully satisfy Constraint 3, which allows the mate segment of an *adverbial* connective to cross the sentence boundary (while it is prohibited for a conjunction), D-STAG employs features on the foot node (i.e., the place where the mate segment comes from) in order to distinguish the case with an adverbial connective from the case with a conjunction.



(a) Deriving $\gamma_{[0,1]}$

(b) The derived tree $\gamma_{[0,1]}$

(c) The derivation tree of $\gamma_{[0,1]}$

Figure 5.54: The derived and derivation trees for a discourse with DNF $C_0\ \mathsf{Conn}_1\ C_1$

One should note that one cannot use any DU-node as a possible adjunction site in a derived tree of discourse. To update a discourse with DNF $C_0\ \ldots\ \mathsf{Conn}_n\ C_n$ with a new piece $\mathsf{Conn}_{n+1}\ C_{n+1}$, one can only adjoin the tree γ_{n+1} on an adjunction site in the tree $\gamma_{[0,n]}$ that belongs to *the right frontier of* $\gamma_{[0,n]}$. Let us illustrate that with the example where $n = 1$. Figure 5.54(a) illustrates the way one derives $\gamma_{[0,1]}$, whereas Figure 5.54(b) and Figure 5.54(c) show the derived tree $\gamma_{[0,1]}$ and its derivation tree, respectively. To extend the current discourse with DNF $C_0\,\mathsf{Conn}_1\,C_1$ with a new piece whose DNF is $\mathsf{Conn}_2\,C_2$, one adjoins γ_2 into $\gamma_{[0,1]}$. The requirement for a node where γ_2 can adjoin is that the node must be on the right frontier of the discourse with DNF $C_0\,\mathsf{Conn}_1\,C_1$. As Figure 5.55 on the next page shows, four sites marked with labels ①, ②, ③ and ④, all are on the right frontier of the derived tree of the discourse with DNF $C_0\,\mathsf{Conn}_1\,C_1$. Consequently, there are four possibilities of attaching the new piece to the current discourse. As Figure 5.54(b) illustrates, there is also one more DU-node, the one that serves as the root of $\begin{smallmatrix}\mathsf{DU}①\\ |\\ C_0\end{smallmatrix}$. However, since the latter node does not belong to the right frontier of the derived tree $\gamma_{[0,1]}$, it does not qualify as an adjunction site in $\gamma_{[0,1]}$. Figure 5.56 shows the derivation tree for a discourse with DNF $C_0\ \mathsf{Conn}_1\ C_1\ \mathsf{Conn}_2\ C_2$, where ⓧ can be either ①, or ②, or ③, or ④. Thus, a discourse with DNF $C_0\ \mathsf{Conn}_1\ C_1\ \mathsf{Conn}_2\ C_2$ may have four derivation trees depending on the value of ⓧ, depending on which link γ_2 adjoins into the derived tree $\gamma_{[0,1]}$ (see Figure 5.55). These derivation trees give rise to the syntactic derived trees with the same yields,

whereas the corresponding semantic trees encode different interpretations of a discourse.

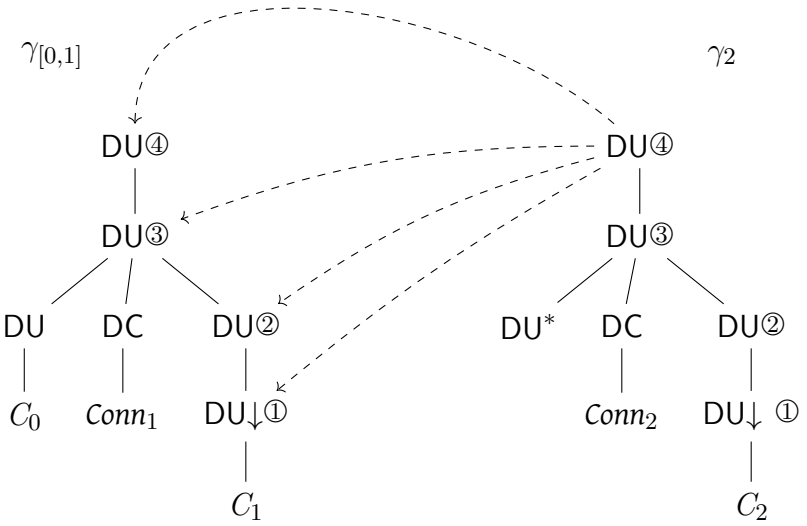


Figure 5.55: The four possibilities of adjoining γ_2 into $\gamma[0,1]$

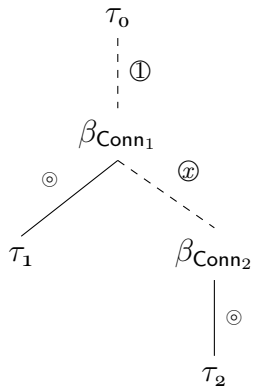


Figure 5.56: The possible D-STAG derivation trees for C_0 Conn_1 C_1 Conn_2 C_2 where $x = 1, 2, 3, 4$

While it is not hard to determine the right frontier of the derived tree of a discourse containing few clauses, in general, it could be a tedious job to define what is a right frontier of a derived tree. Instead, Danlos (2011) suggest to define *the right frontier of a derivation tree.* For that, one needs to order the nodes in a derivation tree, which are intrinsically unordered. To order nodes in a derivation tree, one projects all nodes denoting derivation trees of clauses (τ_i, for $i = 0, 1, 2, \ldots, n$) on some line, and then orders them according to their order in the DNF. In this way, one defines an order $\prec$ on the nodes of a derivation tree. Consequently, one can identify the right frontier of a derivation tree with respect to the order $\prec$. The nodes which appear on the right frontier can be used as adjunction sites. We have two kinds of nodes, some of them

denote derivation trees (τ_i) of clauses and the others denote auxiliary trees (β_{Conn_i}). If τ_k for some k appears at the right frontier, one can adjoin a tree on it as it has single adjunction site marked with ①. If β_{Conn_k} appears at the right frontier, it has three adjunction sites (marked with ②, ③, and ④). The above defined ordering and the right frontier concerns trees but not nodes within the same tree. For instance, if some tree already was adjoined on the node with the link ③ into β_{Conn_k}, then we cannot adjoin a new tree on the node linked with ② in β_{Conn_k}, but on the node with the link ④, because ② is not any more on the right frontier, while ④ is still on the right frontier. To encode this way of building derivation trees, D-STAG defines Constraint 5, formulated as follows:

Constraint 5 (Danlos, 2011)

If β_{Conn_j}, in which τ_j is substituted, adjoins at the link ⓝ of a node β_{Conn_i} (on the DU-adjunction site marked with ⓝ in the auxiliary tree anchored with a connective Conn_i), then β_{Conn_k}, in which τ_k is substituted, can adjoin at the link ⓜ of the node β_{Conn_i} (on the link ⓜ in the auxiliary of a discourse connective tree denoted by β_{Conn_i}) if and only if the following condition holds:
$\tau_j \prec \tau_k$ implies $n < m$, where $n, m \in \{2, 3, 4\}$.

To illustrate the shape of the derivation trees Constraint 5 allows us to build, let us consider the derivation tree depicted in Figure 5.57,[53] which is a derivation tree of discourse whose DNF is $C_0\,\mathsf{Conn}_1\,C_1\,\mathsf{Conn}_2\,C_2\,\mathsf{Conn}_3\,C_3$. In the derivation tree of the discourse, the derivation trees of the clauses C_0, C_1, C_2, and C_3 are ordered as follows: $\tau_0 \prec \tau_1 \prec \tau_2 \prec \tau_3$. As Figure 5.57 shows, the auxiliary trees β_{Conn_2} and β_{Conn_3}, both adjoin into β_{Conn_1}, on the links ⓜ and ⓝ respectively. Since we have that $\tau_2 \prec \tau_3$, according to Constraint 5, one should have $m < n$.

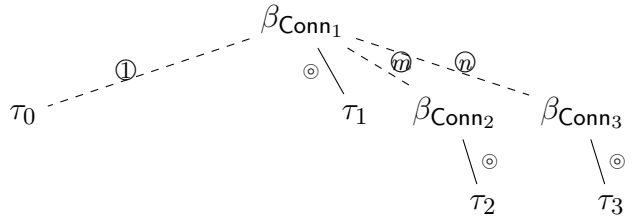


Figure 5.57: A D-STAG derivation tree obeying Constraint 5

5.3.4 Semantic Interpretation

Each syntactic tree is paired with a semantic one. The semantic trees encode semantic interpretations, which are modeled by Higher Order Logic (HOL) formulas. As we already saw, even though a syntactic tree corresponding to a connective-clause pair

[53]Note that the derivation tree in Figure 5.57 is not depicted in the usual way. Indeed, since τ_0 is the root of the tree, one would expect that it to take the highest position among the nodes, in the pictorial representation of the derivation tree. D-STAG uses such an illustration of a tree in order to pictorially represent the right frontier of the tree. Below, we will use the usual representation of trees, where a mother node gets a higher position than its daughters. Nevertheless, we will have in mind that one can always define the right frontier of a derivation tree.

(a tree obtained by substituting a tree anchored by a clause into an auxiliary tree anchored with a connective) has four sites of adjunction (DU①, DU②, DU③, and DU④), only having one of them would suffice to generate the same string language. The reason behind introducing these adjunction sites lays in semantic trees. In particular, D-STAG designs semantic trees in a way that adjoining on different sites gives rise to the semantic trees encoding *different interpretations.*

5.3.4.1 D-STAG Semantic Trees Encoding λ-terms

In D-STAG, semantic trees encode λ-terms. In particular, D-STAG semantic trees serve as tree representations of λ-terms. One represents a λ-term as a tree as follows:[54]

- If a term u is a variable or constant, then one represents it as a tree consisting of a single node labeled with u.
- If a term u is represented as an abstraction $\lambda x.s$, then one represents it as a ternary branching tree whose root node is u. The first child of the root node is (the node at the Gorn address 1) labeled with λ; the second child of the root node is labeled with x (the node at the Gorn address 2); and at the Gorn address 3 is the subtree that is the tree representation of the term s.
- Given a term u where $u = st$, one represents it as a tree whose root node is labeled with u and has two subtrees at the root node: the first subtree is the tree representation of the term s and the second subtree is the tree representation tree of t.

D-STAG uses typed λ-terms. We may decorate interior nodes of the tree representation of a λ-term with types as well, or sometimes, only with types, as for the trees illustrated in Figure 5.58 on the facing page.

5.3.4.2 Two Kinds of Semantic Trees Anchoring Discourse Relations

D-STAG couples an elementary tree anchored with a discourse connective with the semantic trees shown Figure 5.58. Each of these two trees is anchored with the relation R signaled by the discourse connective. We refer to these trees as *the semantic tree A* and *the semantic tree B.*

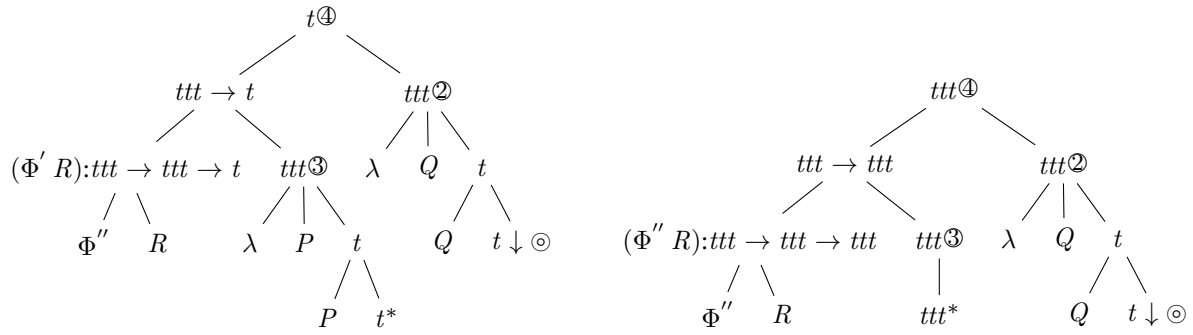
$$\Phi^{'} = \lambda R.\,\lambda X.\,\lambda Y.\,X\,(\lambda\, x.\,(\,Y\,(\lambda\, y.\,R\,x\,y)))\;:(t \to t \to t) \to ttt \to ttt \to t \tag{5.97}$$

$$\Phi^{''} = \lambda R.\,\lambda X.\,\lambda Y.\,\lambda P.\,X\,(\lambda\, x.\,(\,Y\,(\lambda\, y.\,(R\,x\,y) \wedge (P\,x))))\;:(t \to t \to t) \to ttt \to ttt \to ttt \tag{5.98}$$

R is of type $t \to t \to t,$ and ttt is an abbreviation of $(t \to t) \to t$

$\Phi^{'}$ and $\Phi^{''}$, defined in Equation (5.97) and Equation (5.98), enable one to use a discourse relation as an anchor of the semantic trees A and B. If a Conn_α points to a

[54] One can also propose the equivalent definitions (Ker, 2009).



(a) The semantic tree A (b) The semantic tree B

Figure 5.58: Semantic trees A and B
ttt denotes $(t \to t) \to t$

discourse relation R_α, then in certain cases, R_α is used as $\Phi^{'}\ R_\alpha$, and in some cases as $\Phi^{''}\ R_\alpha$. Such possibilities together with the various sites for adjunction in trees enables D-STAG to produce semantic trees encoding various kinds of semantic dependencies, including non-tree shaped DAGs.

By $\beta_{\mathsf{Conn}/R}$, we denote an auxiliary tree pair of D-STAG where the syntactic elementary tree is anchored with Conn and the semantic tree is anchored with the relation R. To point out that we use the semantic tree A (resp. B) paired with an elementary tree anchored with a connective Conn, we attach the superscript A (resp. B) to a node denoting an elementary tree pair $\beta_{\mathsf{Conn}/R}$, i.e., we write $\beta^A_{\mathsf{Conn}/R}$ (resp. $\beta^B_{\mathsf{Conn}/R}$).

5.3.5 Parsing Ambiguity

To parse a discourse with a D-STAG grammar, one faces ambiguity issues. Indeed, in a derived tree of a discourse with several clauses, there are a number of DU-adjunction sites on which one can adjoin a tree so that the resultant derived trees will have the same yields. Thus, one may have a number of derivation trees whose syntactic derived trees have the same yields. However, these derivation trees may give rise to different semantic interpretations. The problem of identifying among these derivation trees the ones that give rise to coherent interpretations, we call the ***ambiguity problem*** of the D-STAG parsing.

To illustrate the parsing ambiguity of D-STAG, let us compare the D-STAG parsing with the one of D-LTAG. In D-LTAG, one interprets a discourse as a tree-shaped structure by using compositional means only. The tree-shaped discourse interpretation can be further expanded to a DAG by retrieving anaphoric arguments of adverbial connectives (if any). In contrast to D-LTAG, D-STAG encodes possibility of obtaining a DAG as a discourse structure within its grammar. In other words, D-STAG produces DAGs compositionally. In D-LTAG, the number of the possible derivation trees for a discourse with three or more clauses is less than the number of possible D-STAG derivation trees for the same discourse. This is due to the fact D-LTAG trees have fewer attachment points (adjunction sites) than D-STAG ones have. That is why the parsing task in D-STAG is more ambiguous compared to the one in D-LTAG. One may consider

the higher ambiguity of the D-STAG parsing compared to the D-LTAG one as the trade-off of having various attachment points. However, the same kind of ambiguity problems arise in D-LTAG as well if one takes into account the resolution of the anaphoric links in the D-LTAG interpretations of discourse. Thus, the differences between the parsing ambiguities in D-STAG and D-LTAG are due to the differences in their goals: while D-LTAG (compositionally) interprets a discourse as a tree, D-STAG (compositionally) interprets a discourse as a DAG. From the perspective of natural language generation, D-STAG has an advantage over D-LTAG as the semantic interpretations already provide information about all the arguments of discourse connectives.

5.3.6 D-STAG Examples

To illustrate the shape of D-STAG derivation trees, we provide the D-STAG derivation trees for the discourses (89)-(92) on page 174, and the syntactic and semantic interpretations they give rise to. How to construct the right derivation tree for a given discourse is the problem that is directly related to the D-STAG parsing ambiguity. Below, we do not deal with the ambiguity problems but directly provide the *right* derivation trees for the examples (89)-(92).

Example 5.1.

(89, repeated)
[*Fred is grumpy*]$_0$ *because* [*he lost his keys*]$_1$. *Moreover,* [*he failed an exam*]$_2$.

DNF: C_0 *because* C_1 *moreover* C_2
Interpretation: EXPLANATION F_0 (CONTINUATION F_1 F_2)

To interpret (89), one identifies rhetorical relations signaled by the discourse connectives *because* and *moreover.* The relation signaled by *because* is EXPLANATION, and the one signaled by *moreover* is CONTINUATION. These relations can anchor the semantic tree A or B. That is, for because/EXPLANATION, we have two tree pairs $\beta^A_{because/explanation}$, and $\beta^B_{because/explanation}$. We have also two tree pairs for $\beta^A_{moreover/continuation}$, and $\beta^B_{moreover/continuation}$. Thus, to build the derivation tree of the discourse, we have the following options:

1. We choose $\beta^A_{because/explanation}$ and $\beta^A_{moreover/continuation}$.
2. We choose $\beta^A_{because/explanation}$ and $\beta^B_{moreover/continuation}$.
3. We choose $\beta^B_{because/explanation}$ and $\beta^A_{moreover/continuation}$.
4. We choose $\beta^B_{because/explanation}$ and $\beta^B_{moreover/continuation}$.

Moreover, we have various adjunction sites where trees can be adjoined. To obtain the interpretation of (89), which is EXPLANATION F_0 (CONTINUATION F_1 F_2), one chooses $\beta^A_{because/explanation}$, and $\beta^A_{moreover/continuation}$. The tree $\beta^A_{moreover/continuation}$ adjoins on the DU① adjunction site in the tree obtained by substituting τ_0 into $\beta^A_{because/explanation}$. Figure 5.59(a) shows the D-STAG derivation tree of the discourse. It gives rise to the pair of

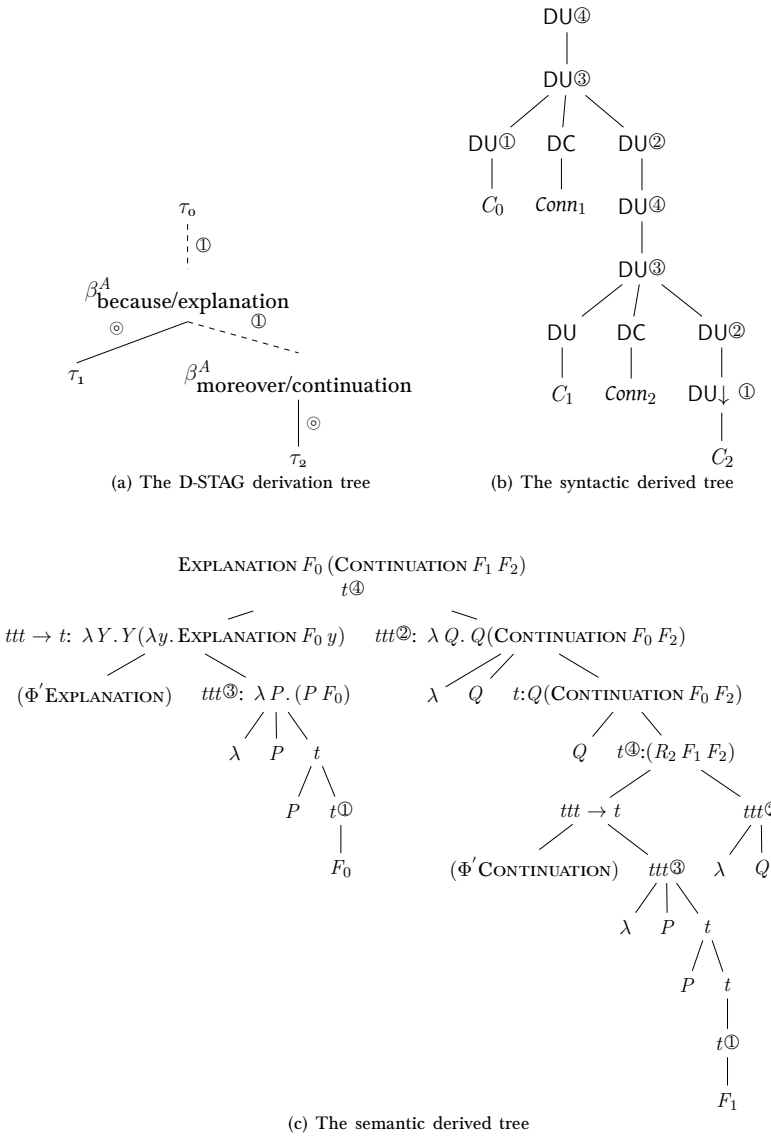


Figure 5.59: The D-STAG derivation tree, and the syntactic and semantic derived trees

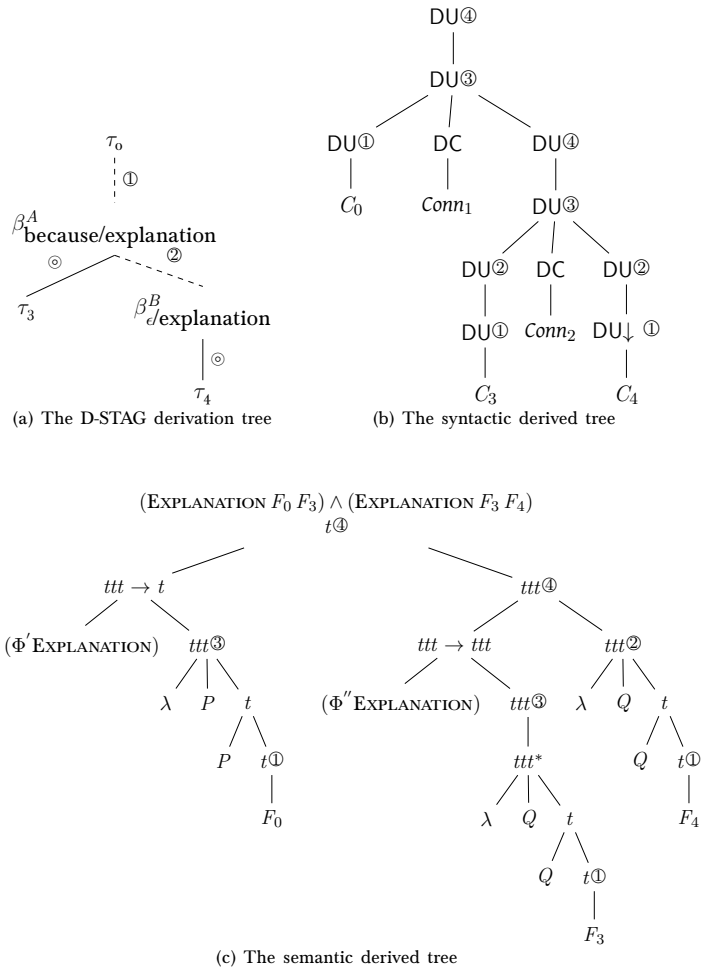
the semantic and the syntactic trees, depicted in Figure 5.59(c) and Figure 5.59(b), respectively. The semantic derived tree encodes the term EXPLANATION F_0 (CONTINUATION F_1 F_2).

Example 5.2.

(90, repeated) [*Fred is grumpy*]$_0$ *because* [*he didn't sleep well*]$_3$. [*He had nightmares*]$_4$.

DNF: C_0 ***because*** C_3 ϵ C_4
Interpretation: (EXPLANATION F_0 F_3) $\wedge$ (EXPLANATION F_3 F_4)

In the case of discourse (90), the discourse marker *because* signals EXPLANATION. The other discourse connective in (90) is the empty connective ϵ, which also signals EXPLANATION.



(a) The D-STAG derivation tree

(b) The syntactic derived tree

(c) The semantic derived tree

Figure 5.60: The D-STAG derivation tree, and the syntactic and semantic derived trees

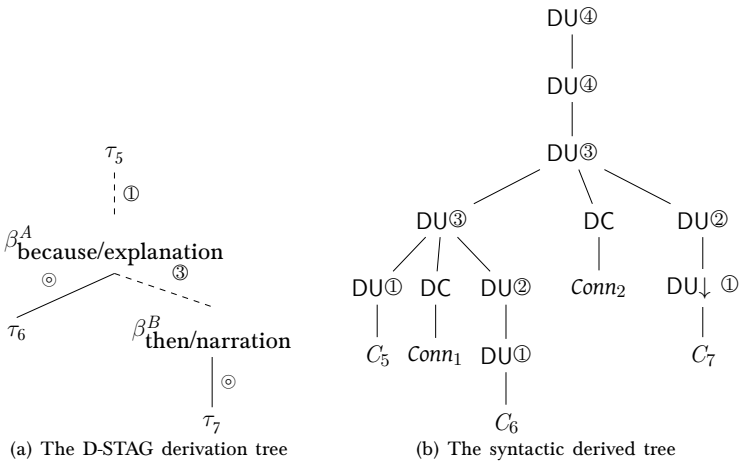
Like in the previous case, one has several choices for selecting trees anchored by connectives. To obtain (EXPLANATION F_0 F_3) $\wedge$ (EXPLANATION F_3 F_4) as the interpretation of the discourse (90), D-STAG selects $\beta^A_{because/explanation}$ and $\beta^B_{\epsilon/explanation}$. The tree $\beta^B_{\epsilon/explanation}$ adjoins on the DU② adjunction site in $\beta^A_{because/explanation}$. Figure 5.60 shows the D-STAG derivation tree and the corresponding syntactic and semantic derived trees. As we can see, the semantic tree indeed encodes (EXPLANATION F_0 F_3) $\wedge$ (EXPLANATION F_3 F_4), which is the interpretation of (90).

Example 5.3.

(91, repeated) [*Fred went to the supermarket*]$_0$ *because* [*his fridge was empty*]$_1$. *Then,* [*he went to the movies*]$_2$.

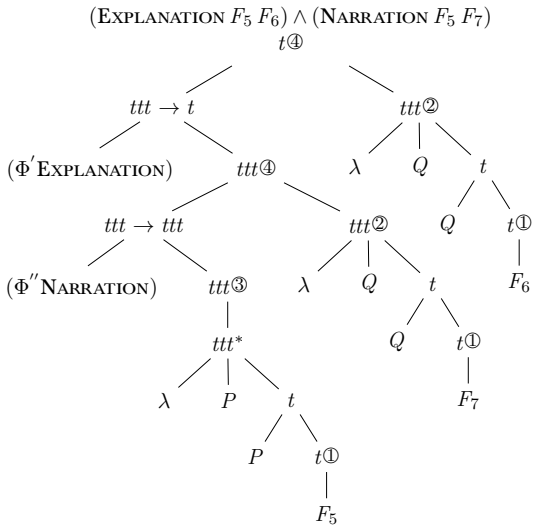
DNF: C_5 *because* C_6 *then* C_7.
Interpretation: (EXPLANATION F_5 F_6) $\wedge$ (NARRATION F_5 F_7)

In the discourse (91), *because* gives rise to the EXPLANATION relation, whereas *then*



(a) The D-STAG derivation tree

(b) The syntactic derived tree



(c) A semantic tree

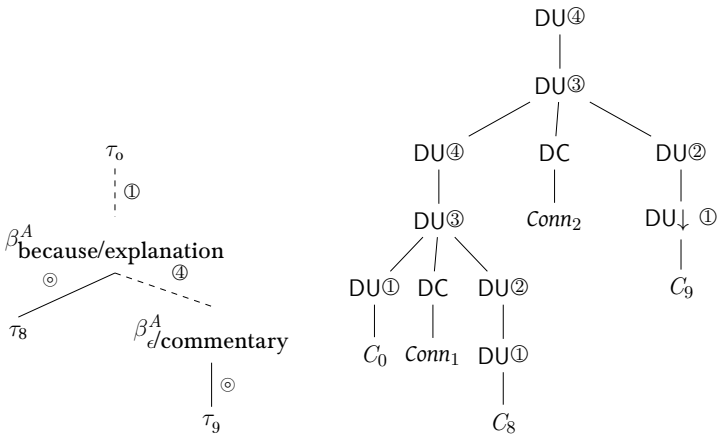
Figure 5.61: The D-STAG derivation tree, and the syntactic and semantic derived trees

signals NARRATION. Like in the previous cases, one has several possibilities for selecting trees anchored by connectives. One interprets the discourse (91) by selecting the trees $\beta^A_{because/explanation}$ and $\beta^B_{then/narration}$. Figure 5.61 shows the derivation tree along with the semantic and syntactic derived trees for (91). The semantic derived tree encodes the term (EXPLANATION F_5 F_6) ∧ (NARRATION F_5 F_7), which is indeed the interpretation of (91).

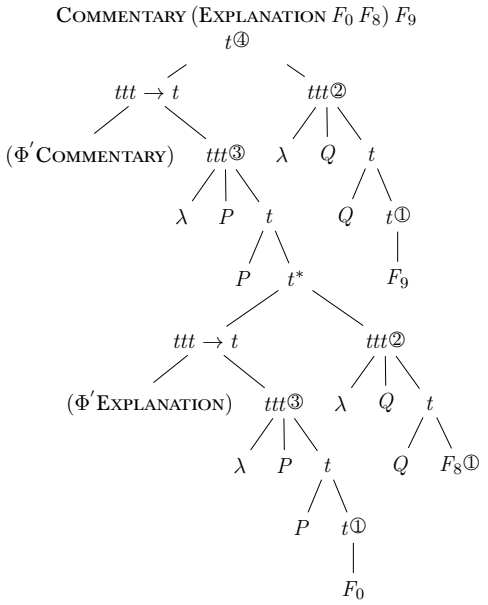
Example 5.4.

(92, repeated)
[*Fred is grumpy*]$_0$ *because* [*his wife is away this week*]$_8$. [*This shows how much he loves her*]$_9$.

DNF: C_0 *because* $C_8 \in C_9$
Interpretation: COMMENTARY (EXPLANATION F_0 F_8) F_9



(a) The D-STAG derivation tree (b) The derived tree



(c) The semantic derived tree

Figure 5.62: The D-STAG derivation tree, and the syntactic and semantic derived trees

In (92), *because* signals EXPLANATION. The other connective is the empty connective ϵ. In the case of (92), ϵ is interpreted as COMMENTARY. To obtain the interpretation of (92), D-STAG selects the trees $\beta^A_{because/explanation}$ and $\beta^A_{\epsilon/commentary}$. Figure 5.62 illustrates the D-STAG derivation tree along with the corresponding syntactic and semantic derived trees. As we can see, the semantic tree encodes the term COMMENTARY (EXPLANATION F_0 F_8) F_9, which is indeed the interpretation of (92).

5.3.7 Preposed Conjunctions

In D-STAG, one considers the case where a preposed conjunction plays the role of a *framing adverbial* in a discourse (Charolles, 2005). Such a preposed conjunction can include in its scope several sentences. For the sake of illustration, let us consider Example (96). The preposed conjunction *when* is a *frame builder* in (96). The mate segment of *when* is *Fred went to the Eiffel Tower. Next, he visited The Louvre.* Hence, the mate segment *when* crosses a sentence boundary. Figure 5.63 shows a pictorial representation of the discourse structure of (96).

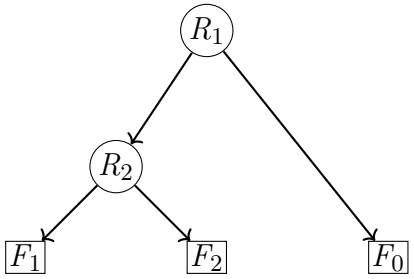


Figure 5.63: CIRCUMSTANCE (NARRATION$_2$ F_1 F_2) F_0

(96, repeated) *When* [*he was in Paris*]$_0$, [*Fred went to the Eiffel Tower*]$_1$. *Next,* [*he visited the Louvre*]$_2$.

DNF: When C_0, C_1. Next C_2.
Interpretation: CIRCUMSTANCE (NARRATION F_1 F_2) F_0

To give an account of a preposed conjunction such as *when*, D-STAG uses an auxiliary tree, such as one in Figure 5.64. This auxiliary tree has one more DU-adjunction site (marked with link ⑤) compared to one anchored with a postposed conjunction or an adverbial connective. To obtain the interpretation CIRCUMSTANCE (NARRATION F_1 F_2) F_0, D-STAG adjoins the tree pair of the piece of discourse $C_1 \mathsf{Conn}_2 C_2$ on this additional adjunction site, as the derivation tree in Figure 5.65(a) indicates. Figure 5.65(b) and Figure 5.65(c) illustrate the syntactic and semantic derived trees for (96) specified by the derivation tree in Figure 5.65(a).

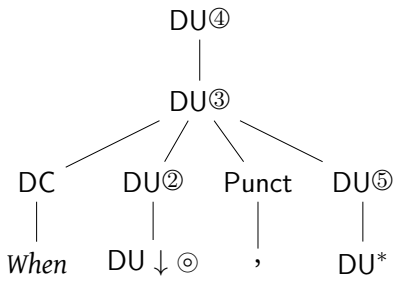
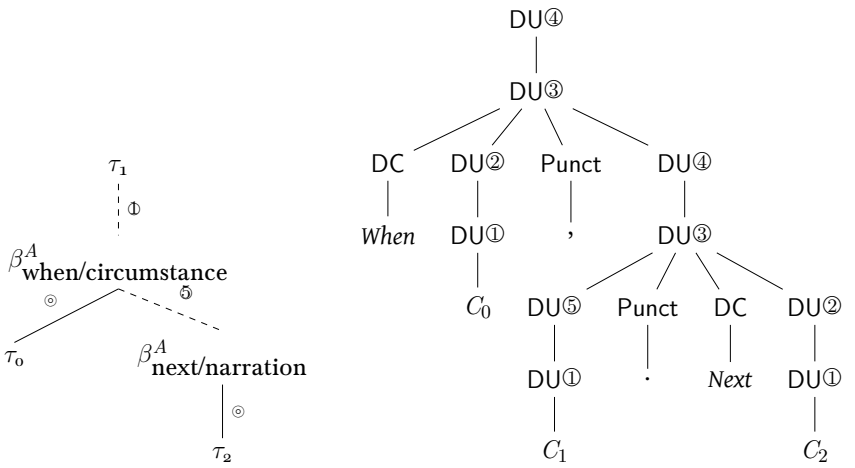
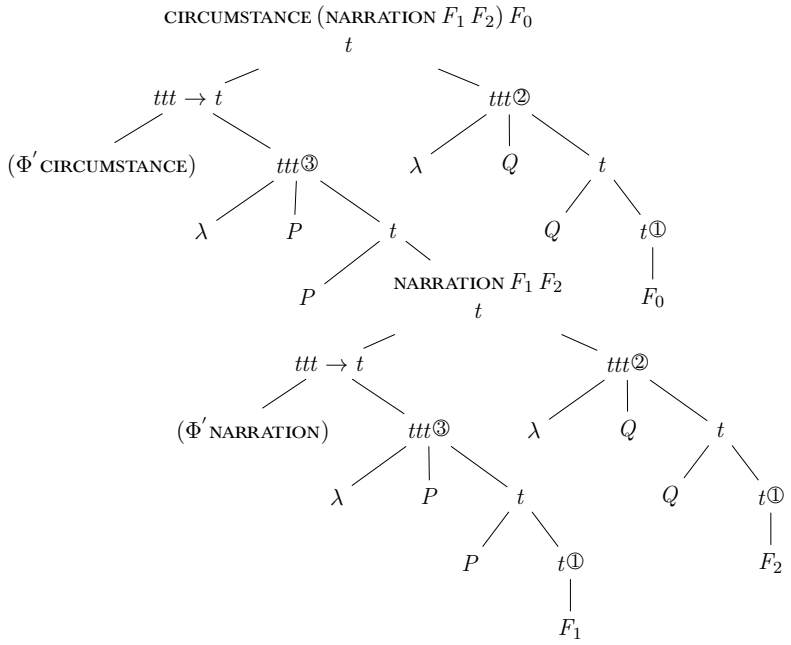


Figure 5.64: The D-STAG syntactic tree anchored by a preposed conjunction



(a) The derivation tree (b) The syntactic derived tree D-STAG derivation tree



(c) The semantic derived tree

Figure 5.65: The D-STAG derivation tree of a discourse, and its syntactic and semantic derived trees

5.3.8 Modifiers of Discourse Connectives in D-STAG

As we already discussed in D-LTAG (see Section 5.1.4.5.1), some adverbials contribute to a discourse structure by being parasitic on a discourse connective, like it is in the sentence (99). Following D-LTAG, D-STAG also considers the adverbial *for example* in (99) as a modifier of the discourse connective *because*.

(99) You shouldn't trust Jack because, for example, he never returns what he borrows.

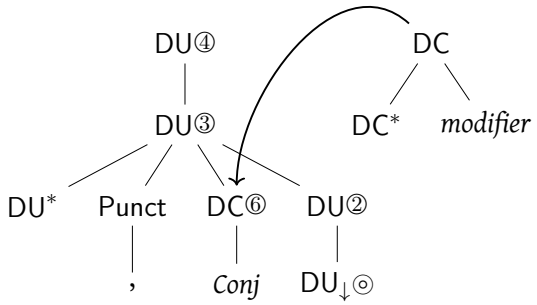
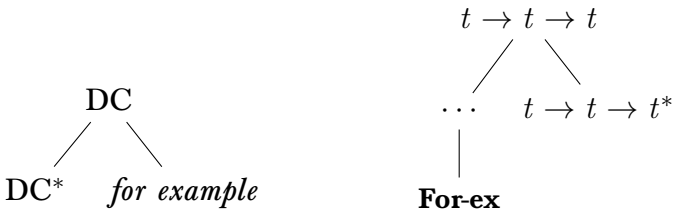


Figure 5.66: An auxiliary tree anchored with a connective modifier adjoins on the DC node into the auxiliary tree anchored by a discourse connective

One calls a connective that modifies discourse connectives a *connective modifier*. D-STAG encodes a connective modifier as an auxiliary tree pair. The syntactic tree of the pair is a DC-rooted auxiliary tree. To modify a discourse connective, it adjoins on the DC node of an auxiliary tree anchored with the connective. Figure 5.66 shows that an elementary tree anchored with a modifier adjoins on an elementary tree anchored with a preposed conjunction. To mark the new adjunction site where a tree anchored with a modifier adjoins, one attaches the new link ⑥ to the DC node of the tree anchored with a connective. Regarding the semantic tree of the pair, the exact shape of the tree depends on a modifier itself. In the case of the modifier *for example*, Figure 5.67 illustrates the semantic tree (together with the syntactic one) of *for example*. The anchor of the semantic tree is **For-ex**. D-STAG defines **For-ex** according to the semantic analysis of *for example* provided in D-LTAG, which is as follows:

$$\textbf{For-ex} \triangleq \lambda\, R\, p\, q.\, \textsc{Exemplification}\ q\ (\lambda\, r.\, R\, p\, r) \tag{5.100}$$



(a) The syntactic tree (b) The semantic tree

Figure 5.67: The D-STAG tree pair of *for-example*

Part II

Thesis Contributions

Chapter 1

G-TAG as ACGs

Contents

In this chapter, we present the ACG encoding of G-TAG. Since G-TAG is based on TAG principles, the ACG encoding of G-TAG relies on the ACG encoding of TAG with Montague semantics. On the other hand, while TAG is only concerned with sentence-level structures, the objective of G-TAG is to generate a text rather than a sentence. For that, G-TAG defines its own discourse grammar. By encoding the G-TAG grammar, we design ACGs suitable for discourse modeling. The ACGs that we design are second-order ones. This ensures that the tasks of parsing and generation with the ACG encoding of G-TAG are polynomial.

1.1 Motivations

By encoding (L)TAG with Montague semantics as ACGs (Pogodalla, 2009),[55] one models the syntax-semantics interface at the sentence-level. To model the syntax-semantics interface for discourse, one can extend the ACG encoding of TAG by encoding a discourse-level grammar in addition to the sentence-level one. In Section 5.2 on page 153, we discussed G-TAG (Danlos, 1998). With the help of G-TAG, one generates a text from a conceptual representation input. As the sentence-level grammar encoded in (Pogodalla, 2009) is a TAG grammar, and at the same time G-TAG offers a discourse grammar based on the TAG principles, we expect the study of G-TAG to help us to design ACGs suitable for discourse modeling.

1.2 The ACG Architecture for G-TAG

In G-TAG, the pivot for a g-derived tree is its g-derivation tree. Indeed, G-TAG first constructs a g-derivation tree out of a conceptual representation input. Afterwards, the g-derivation is mapped to the g-derived tree. Thus, the architecture of G-TAG is similar to the one of the ACG encoding of TAG with Montague semantics:

- In G-TAG, one builds a g-derivation tree (output) from a conceptual representation (input). In the ACG encoding of TAG with Montague semantics, one translates a TAG derivation tree (input) to a semantic formula (output).
- A g-derivation tree signifies a (unique) g-derived tree. Using the ACG encoding of TAG, we translate a TAG derivation tree to a (unique) derived tree.

Thus, in both G-TAG and the ACG encoding of TAG with Montague semantics, one establishes the correspondence between conceptual representations (semantic interpretations) and derived trees through derivation trees. In the ACG encoding of TAG with Montague semantics, derivation trees are abstract terms. In order to encode G-TAG as ACGs, we develop a similar approach to the one of the ACG encoding of TAG with Montague semantics. That is, we encode g-derivation trees as abstract terms. To model that G-TAG builds a g-derivation tree out of a conceptual representation input, we define a lexicon that interprets g-derivation trees to conceptual representations.

We can also define another lexicon for interpreting the abstract terms modeling g-derivation trees as derived trees. However, we develop a more modular approach than that by interpreting g-derivation trees as TAG derivation trees. Then, with the help of the ACG encoding of TAG, one can interpret TAG derivation trees as TAG derived trees. In this way, one establishes the correspondence between g-derivation trees and TAG derived trees via TAG derivation trees. By interpreting g-derivation trees as TAG derivation trees, one makes explicit (a) the motivations why g-derivation trees have various kinds of features; (b) how g-derivation trees relate to TAG derivation trees. That is why we opt for the modular interpretation of g-derivation trees as derived trees. However, in this case, one uses TAG derived trees instead of g-derived trees. Although g-derived trees differ from TAG derived trees, they are conceptually close to each other so that TAG derived trees can model g-derived trees. In other words, for a g-derived

[55] See Section 3.8 on page 80.

tree, we can always find its equivalent TAG derived tree. Hence, instead of using g-derived trees, we use TAG derived trees. Figure 1.1 depicts the ACG architecture of G-TAG.

We aim at designing such ACG architecture for G-TAG where the abstract vocabulary is second-order and the lexicons are almost-linear, because, in this case, the problems of parsing and generation are polynomial (Kanazawa, 2007).

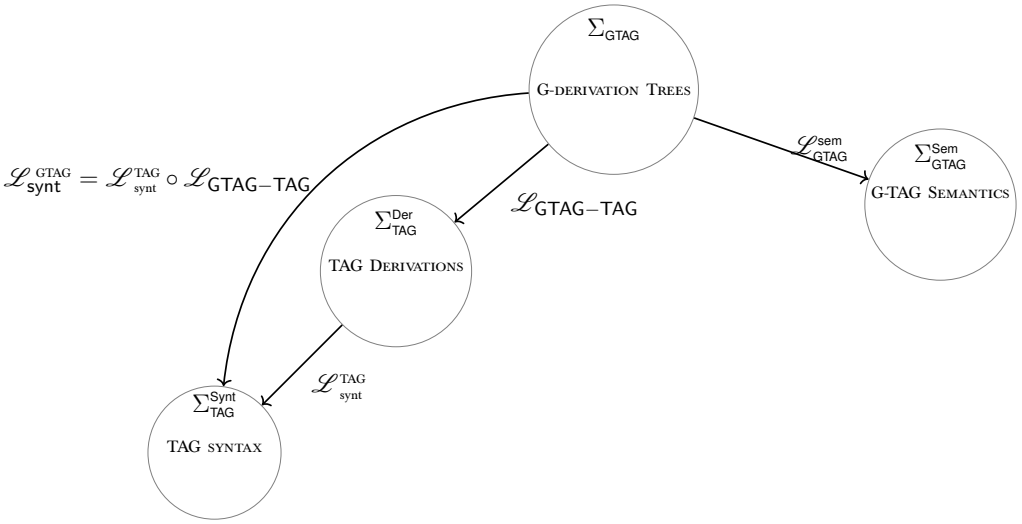


Figure 1.1: The ACG architecture for G-TAG

1.3 G-derivation Trees as Abstract Terms

To encode G-TAG as ACGs, we construct the abstract vocabulary Σ_{GTAG}. Terms over Σ_{GTAG} model g-derivation trees. Since the notion of a g-derivation tree is close to the notion of a TAG derivation tree, in order to build the signature Σ_{GTAG}, one may refer to the signature $\Sigma^{\text{Der}}_{\text{TAG}}$, which we use in order to encode TAG derivation trees of sentences in the ACG encoding of TAG with Montague semantics.

Although TAG derivation trees and g-derivation trees are conceptually alike, they show some differences. These differences prevent one from directly using the ACG encoding of TAG derivation trees for the purposes of encoding g-derivation trees. In a TAG derivation tree, a node represents an elementary tree with an inflected anchor, whereas in a g-derivation tree, T-features and morphological features decorate nodes standing for the lexical entries. Out of these features, one computes the structural description of a tree and the inflected version of the lexical entry anchoring the tree. One could try to represent g-derivation trees as abstract terms by encoding the lemmas and morphological features of G-TAG as the terms over the abstract vocabulary. In that case, one has to be able to build an abstract term encoding an inflected version of a word with the help of the terms encoding the lemma and features. Furthermore, one needs to interpret the terms encoding features and lemmas to derived (syntactic) trees and to the surface representations. This requires to develop a compositional

approach to computational-morphology within the ACG framework: A derived tree with inflected anchors has to be derivable from (decomposable into) the trees standing for the interpretations of lemmas and features. In addition, to interpret the terms modeling g-derivation trees as semantic (conceptual) representations, one should give a compositional account of the morphology-semantic interface problem within ACGs. Indeed, given T-features and morphological features of a lexical entry, one has to be able to obtain its semantic interpretation.[56] However, we do not develop these approaches in the present work, as they go beyond the scope of this thesis. We choose another way of encoding g-derivation trees. Instead of encoding lemmas and features in the abstract vocabulary, we encode inflected forms of words. Furthermore, instead of encoding T-features and their combinations, we directly encode in the abstract vocabulary elementary trees that (sets of) T-features give rise to. Thus, we deal morpho-syntactic questions in the same way as it is done in TAG (and consequently, in the ACG encoding of TAG).[57]

In Section 5.2 on page 153, we saw that one defines a g-derivation tree with the help of an *underspecified g-derivation tree*. Underspecified g-derivation trees differ from TAG derivation trees. In particular, underspecified g-derivation have variable nodes, which is not the case of TAG derivation trees. By instantiating variable nodes of an underspecified g-derivation tree, one obtains a g-derivation tree. Thus, the notion of an underspecified g-derivation tree is reminiscent of the notion of an abstract term of the ACG encoding of TAG. To illustrate that, let us consider the lexical entry *récompenser* (reward) and the canonical underspecified g-derivation tree associated with it. Figure 1.2(a) shows this underspecified g-derivation tree. This underspecified g-derivation tree denotes an initial tree encoding an active voice construction with a verb. To compute the anchor of the tree, the G-TAG post processing module uses the morphological features. Namely, the G-TAG post processing module computes that the anchor of the tree is the past participle form of *récompenser* (to reward), which is *récompensé*. One can model this underspecified g-derivation tree by a term over $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$, that is, one can encode this tree using the ACG encoding of TAG as follows:

$$t_1 = \lambda^{\circ} arg_1.\ \lambda^{\circ} arg_2.\ \mathrm{C}_{\textbf{récompensé}}\ \mathrm{I}_{\mathsf{S_A}}\ (\mathrm{C}_{\mathbf{a}}\ \mathrm{I}_{\mathsf{V_A}})\ arg_1\ arg_2 : \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{S} \tag{1.1}$$

Where $\mathrm{C}_{\textbf{récompensé}} \in \Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ models the TAG initial tree anchored with the past participle *récompensé*. It is of type $\mathsf{S_A} \multimap \mathsf{V_A} \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{S}$.

$\mathrm{C}_{\mathbf{a}} \in \Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ models a French auxiliary verb *a* ($\text{have}_{\text{3P.SGL.PRS.}}$). It is of type $\mathsf{V_A} \multimap \mathsf{V_A}$

$\mathrm{I}_{\mathsf{V_A}} : \mathsf{V_A}$ is a constant modeling an empty adjunction.

Terms can model both partially instantiated and fully instantiated underspecified g-derivation trees. For instance, Figure 1.2(b) shows a partially instantiated underspecified g-derivation tree. One encodes this tree as follows:

$$t_2 = \lambda^{\circ} arg_2.\ \mathrm{C}_{\textbf{récompensé}}\ \mathrm{I}_{\mathsf{S_A}}\ (\mathrm{C}_{\mathbf{a}}\ \mathrm{I}_{\mathsf{V_A}})\ \mathrm{C}_{\textbf{marie}}\ arg_2 : \mathsf{np} \multimap \mathsf{S} \tag{1.2}$$

[56] To achieve that, one may try to use a richer type system for ACGs (Pompigne, 2013).

[57] For a detailed discussion about the possibilities of encoding features within ACGs, we refer readers to (Kanazawa, 2015).

In the term t_2, we encode the g-derivation tree for Marie with the constant $\mathrm{C}_{\textbf{marie}} \in \Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$. Furthermore, we encode the g-derivation tree in Figure 1.2(c) as the term t_3 defined as follows:

$$t_3 = \mathrm{C}_{\textbf{récompensé}}\ \mathrm{I}_{\mathsf{S_A}}\ (\mathrm{C}_{\mathbf{a}}\ \mathrm{I}_{\mathsf{V_A}})\ \mathrm{C}_{\textbf{marie}}\ \mathrm{C}_{\textbf{jean}} : \mathsf{S} \tag{1.3}$$

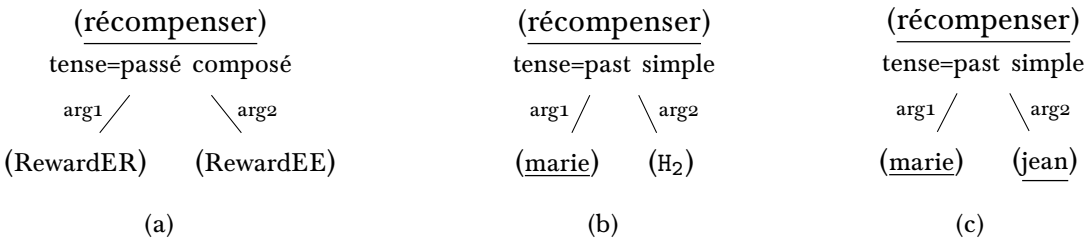
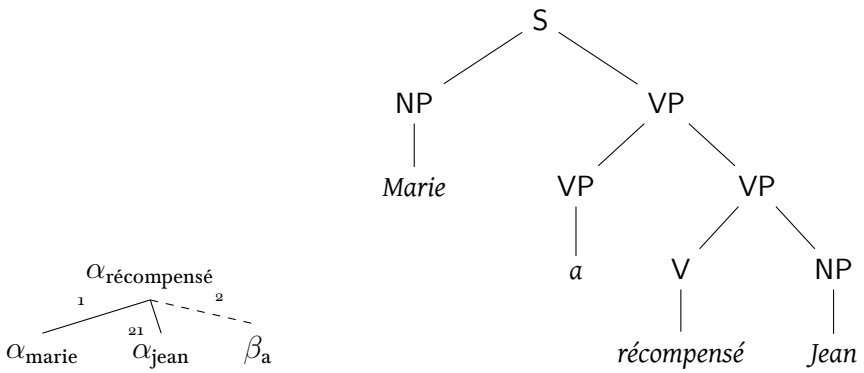


Figure 1.2: The underspecified g-derivation tree associated with the lexical entry récompenser and the trees obtained out of it by specifying its variable nodes



(a) A TAG derivation tree

(b) A TAG derived tree

Figure 1.3: The TAG derivation and derived trees for *Marie a récompensé Jean*

As one can see, the term t_3 serves as the ACG encoding of the TAG derivation tree shown in Figure 1.3(a), whose derived tree is given in Figure 1.3(b). Hence, we can encode underspecified g-derivation trees, and consequently, g-derivation trees, with the help of the constants of $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$. That is why we adopt the constants and types of $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ in Σ_{GTAG} (the vocabulary where we encode g-derivation trees). This allows us to define the same terms over Σ_{GTAG} as the ones over $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$. However, the terms over $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ encode only derivation trees of *sentences.* In Σ_{GTAG}, by only having constants and types adopted from $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$, one cannot model g-derivation trees of multi-sentential texts. In addition, notice that the G-TAG analysis of discourse connectives differs from the TAG one (XTAG-Group, 1998). While a discourse connective anchors an initial tree in G-TAG, it anchors an auxiliary one in TAG. Hence, in order to encode a g-derivation tree of a discourse, one should encode discourse connectives differently from the way they are encoded in $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$. Thus, one cannot adopt in Σ_{GTAG} the constants in $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ representing

TAG trees anchored with discourse connectives. Moreover, let us recall that G-TAG offers a special treatment of *reduced* conjunctions, where the *argument sharing* between the matrix clause and the subordinated one takes place. As we saw,[58] to generate a text with a reduced conjunction, one has to obey certain requirements. In ACGs, we do not have the same kind of decision mechanisms as G-TAG uses in its text generation process. What we have are only types and constants, or to put it another way, in ACGs, everything is grammaticalized. Thus, to model g-derivation trees of discourses, we introduce the constants and types in Σ_{GTAG}, besides those ones that we adopt from $\Sigma^{\text{Der}}_{\text{TAG}}$.

1.3.1 Types

In G-TAG, in order to distinguish the g-derivation trees of texts and sentences, one employs the features $(+\text{T}, +\text{S})$ and $(-\text{T}, +\text{S})$, respectively.

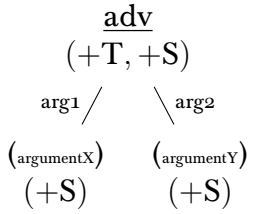


Figure 1.4: A g-derivation tree for the adv lexical entry

Figure 1.4 shows the lexical entry adv for the adverbial *adv*. It consists of an underspecified g-derivation tree whose root has the feature $(+\text{T}, +\text{S})$. Both of the daughter nodes of the root node have the feature $(+\text{S})$, which stands for either a text or a sentence. Thus, there are several cases to consider:

1. Both of the daughter nodes of the root node are g-derivation trees of texts;
2. both of the daughter nodes of the root node are g-derivation trees of sentences;
3. the first daughter node (arg1) is a g-derivation tree of a sentence, while the second one (arg2) is a g-derivation tree of a text;
4. the first daughter node (arg1) is a g-derivation tree of a text, and the second one (arg2) is a g-derivation tree of a sentence.

To define terms modeling g-derivation trees, one may encode the features $(-\text{T}, +\text{S})$ and $(+\text{T}, +\text{S})$ in the abstract vocabulary, i.e., in Σ_{GTAG}. We encode the feature $(-\text{T}, +\text{S})$ with the type S. To encode the feature $(+\text{T}, +\text{S})$, we introduce a new atomic type T in Σ_{GTAG}. Depending on whether a g-derivation tree gives rise to a sentence or a text (which is indicated by the feature of its rootnode), we model it by a term of type S or T, respectively.

Remark 1.1. *G-TAG introduces morphological features that the post-processing module uses in order to generate texts where clauses have to be of certain tenses, moods etc. For each combination of morphological features, we can introduce in the abstract vocabulary a new type and the constants that will enable us to define the terms of that type. In a sentence consisting*

[58] See Section 5.2.6 on page 168.

of two clauses connected by a conjunction, one has to encode the agreement between the mood and tense of two clauses. In order to model such constraints, one can use ACGs with richer type systems (de Groote and Maarek, 2007; de Groote, Maarek, and Yoshinaka, 2007; Pompigne, 2013) than the version of ACGs with simple types, which we use in the present work. Although we could also encode certain kinds of agreements using simple types, in this thesis, we do not deal the morphological features and agreements, but leave them for future work.

1.3.2 Constants

In G-TAG, one uses T-features in order to identify/designate the syntactic trees encoding different uses of a lexical entry. An underspecified g-derivation tree defined by T-features signifies a particular syntactic use of a lexical entry, i.e., a particular syntactic construction with the lexical entry. We model lexical entries with the help of abstract terms, i.e., terms over Σ_{GTAG}.

Convention: we denote the abstract constant modeling a tree anchored with α as a G_α constant in Σ_{GTAG} (so that we do not confuse the constants used in the current encoding of G-TAG as ACGs with the ones used in the ACG encoding of TAG, where the abstract constants are denoted with C_α).

1.3.2.1 Discourse Connectives

In G-TAG, both adverbials and conjunctions anchor initial trees, whereas in TAG, they anchor auxiliary trees. We do not adopt in Σ_{GTAG} the constants of $\Sigma^{\text{Der}}_{\text{TAG}}$ modeling subordinate conjunctions and adverbials because otherwise the ACG encoding of G-TAG would diverge from G-TAG. Instead, we introduce new constants in Σ_{GTAG} to model underspecified g-derivation trees of subordinate conjunctions and adverbials.

Convention: We say *conjunction* instead of *subordinate conjunction* whenever it does not cause a confusion.

1.3.2.1.1 Adverbials

With the help of underspecified g-derivation trees for discourse adverbials, G-TAG generates texts. For instance, one obtains the g-derivation tree of the discourse (4) by instantiating the variable nodes of the underspecified g-derivation tree of the adverbial *ensuite* (afterward).

(4) *Jean a passé l'aspirateur. Ensuite, il a fait une sieste.*
Jean have$_{\text{PRES. 3PS. SG.}}$ pass$_{\text{PAST PART.}}$ vacuum-cleaner$_{\text{DEF.}}$. Afterwards, he have$_{\text{PRES. 3PS. SG.}}$ make$_{\text{PAST PART.}}$ a nap.

John vacuumed. Afterwards, he took a nap.

In order to encode an underspecified g-derivation tree of an adverbial *adv* (see Figure 1.4 on the facing page), we encode the possible cases of the values of the features decorating its nodes. The feature $(+\mathrm{T}, +\mathrm{S})$ decorates the constant node (the root node).

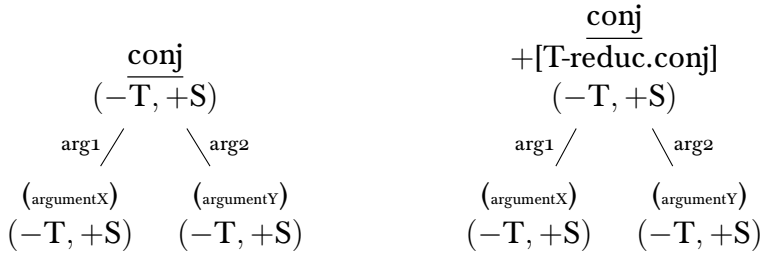
The feature $(+\mathrm{T}, +\mathrm{S})$ only applies to a *text*, which we encode with a term of type T. The constant node has two daughter nodes, which are the variable nodes. Each of the daughter nodes has the feature $(+\mathrm{S})$, which can denote both a *text* and a *sentence*. Hence, the possible values of the daughter nodes are the following four pairs: (sentence, sentence), (sentence, text), (text, sentence), and (text, text). To encode each of the four cases, in Σ_{GTAG}, we introduce the four constants shown in Table 1.1. Each of these four constants receives as its arguments terms of type T and/or S. The resultant term encodes a g-derivation tree specifying a text, i.e., a term of type T.

Constants in Σ_{GTAG}	Their Types
$\mathrm{G}_{\mathbf{adv}_{\mathrm{S}}}^{\ \mathrm{S}}$	$\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{T}$
$\mathrm{G}_{\mathbf{adv}_{\mathrm{T}}}^{\ \mathrm{S}}$	$\mathsf{S} \multimap \mathsf{T} \multimap \mathsf{T}$
$\mathrm{G}_{\mathbf{adv}_{\mathrm{S}}}^{\ \mathrm{T}}$	$\mathsf{T} \multimap \mathsf{S} \multimap \mathsf{T}$
$\mathrm{G}_{\mathbf{adv}_{\mathrm{T}}}^{\ \mathrm{T}}$	$\mathsf{T} \multimap \mathsf{T} \multimap \mathsf{T}$

Table 1.1: Constants in Σ_{GTAG} modeling the G-TAG lexical entry of an adverbial

1.3.2.1.2 Subordinate Conjunctions

G-TAG employs an underspecified g-derivation tree of a subordinate conjunction in order to generate a sentence with two clauses. This is expressed using features on the nodes of the underspecified g-derivation tree of a conjunction. Figure 1.5 illustrates underspecified g-derivation trees of a conjunction. Each node in these trees has the feature $(-\mathrm{T}, +\mathrm{S})$ denoting a sentence.



(a) The canonical case

(b) The canonical case with +[T-reduc.conj]

Figure 1.5: Two underspecified g-derivation trees for conj

The underspecified g-derivation trees in Figure 1.5 give rise to stylistically different constructions. The underspecified g-derivation tree shown in Figure 1.5(a) is the canonical one, which gives rise to the sentences such as the following one:

(5) *Jean fait une sieste après que Marie passe l'aspirateur.*
John make$_{\text{PRES. 3PS. SG.}}$ a nap after that Mary pass$_{\text{3PS. SG. SUBJ.}}$ vacuumer$_{\text{DEF.}}$.

John takes a nap after that Mary vacuums.

The underspecified g-derivation tree with the feature +[T-reduc.conj] (see Figure 1.5(b)) gives rise to the sentences such as the following one:

(6) *Jean fait une sieste après avoir passé l'aspirateur.*
John make$_{\text{PRES. 3PS. SG.}}$ a nap after to-have$_{\text{PRES. INF.}}$ pass$_{\text{PAST PART.}}$ vacuumer$_{\text{DEF.}}$.
John takes a nap after vacuuming.

The difference between the canonical g-derivation tree and the one with the feature +[T-reduc.conj] is that in the canonical g-derivation tree, the variable nodes must be instantiated with g-derivation trees of *complete*[59] clauses (see e.g. (5)), whereas in the case of the underspecified g-derivation tree with the feature +[T-reduc.conj], only one of the variable nodes must be instantiated with a g-derivation tree of a complete clause. Namely, the matrix clause should be a complete clause, whereas the subordinated one should be a *reduced* clause, i.e., an *infinitive clause* introduced by the subordinated conjunction (Danlos, 2000). We discuss the canonical case and the one with reduced conjunction separately.

The Canonical G-Derivation Tree

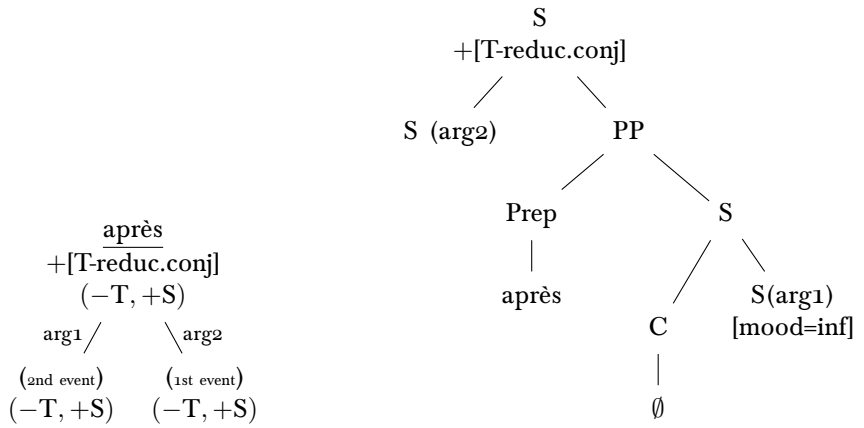
As we encode the feature $(-\mathrm{T}, +\mathrm{S})$ with the type S, we type an abstract constant encoding a canonical g-derivation tree of a conjunction with the type $\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$. Indeed, the canonical g-derivation tree of a conjunction gives rise to the g-derived tree of a sentence connecting two complete clauses. Since we encode a derivation tree of a complete clause with a term of type S, we introduce the constant $\mathrm{G}^{\text{canonical}}_{\textbf{conj}}$ of type $\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$ in the abstract vocabulary Σ_{GTAG} as the encoding of the canonical g-derivation tree of the conjunction *conj.*

The G-Derivation Tree of a Reduced Conjunction

Since we encode the feature $(-\mathrm{T}, +\mathrm{S})$ with the type S, one could type a constant encoding the g-derivation tree with the feature +[T-reduc.conj] (see Figure 1.5(b) on the preceding page) with the type $\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$. Indeed, one could claim that since the variable nodes have only the feature $(-\mathrm{T}, +\mathrm{S})$, we should model them with the type S.

However, such an encoding is not suitable in this case. To illustrate that, let us consider the conjunction *après* (after). Assume that one models the underspecified g-derivation tree of the après in the reduced case (see Figure 1.6(a)) with the constant $\mathrm{G}^{\text{red.}}_{\textbf{après}}$ of type $\mathsf{S} \multimap \mathsf{S} \multimap \overline{\mathsf{S}}$. With the help of the constant $\mathrm{G}^{\text{red.}}_{\textbf{après}}$, we should be able to analyze/produce sentences such as (6), where the subordinated clause is an infinitive (reduced) clause. This implies that we model a derivation tree of an infinitive clause by a term of type S, i.e., by a term of the same type as a term modeling a derivation tree of a complete clause. One may claim that an infinitive clause is *close to* a complete one so that we can model both with terms of the same type. Indeed, according to the LTAG grammar for French (Abeillé, 1988), the syntactic tree shown in Figure 1.7 is a syntactic analysis of an infinitive clause such as *avoir passé l'aspirateur* (have$_{\text{PRES. INF.}}$ pass$_{\text{PAST}}$

[59] We call a clause *complete* if its predicate is built with the help of a finite verb form and it contains a lexically expressed subject.



(a) The reduced underspecified g-derivation tree for après

(b) The elementary tree in the reduced case of après

Figure 1.6: The underspecified g-derivation tree of a conjunction and its corresponding elementary tree

PART. vacuumer DEF.). As this tree indicates, the null pronoun PRO occupies the position of the *syntactic* subject of the infinitive clause. Hence, we could model the derivation tree of an infinitive clause by a term of type S. The syntactic interpretation of such a term would be a derived tree, such as one in Figure 1.7.

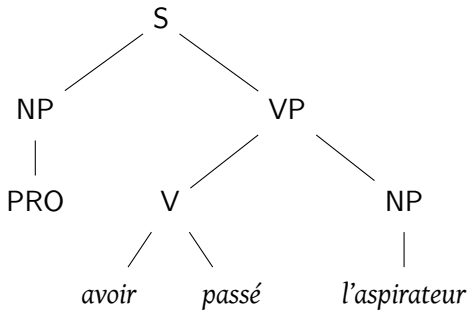
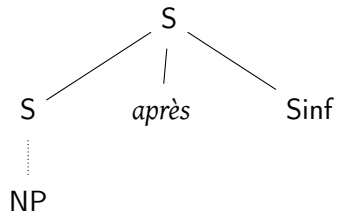


Figure 1.7: The LTAG analysis of infinitive phrases: PRO + infinitive verb form

However, apart from syntactic interpretations, we aim at defining *semantic* interpretations of the sentences such as (6). If one encodes the constant $\mathrm{G}^{\text{red.}}_{\textbf{après}}$ with the type $\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$, then a complete clause and a reduced (infinitive) one, both are encoded by terms of type S. Consequently, one has to interpret the terms encoding derivation trees of a complete clause and a reduced one as the semantic terms of the same type. While we can interpret a complete clause as a term of type t (as a proposition), we cannot interpret a term encoding a derivation tree of a reduced clause as a term of type t. Indeed, the null pronoun PRO is without any phonological content (see Figure 1.7). PRO does not serve as a lexicalization of any concept and thus PRO cannot provide an argument for a concept (predicate) expressed by an infinitive verb form (phrase), whose

arguments should be concepts.[60] Hence, we cannot encode the g-derivation trees of a complete clause and a reduced one in the same way. To distinguish them, we introduce a new type Sinf in Σ_{GTAG}. We model derivation trees of infinitive (reduced) clauses with terms of type Sinf. Now, we can translate a term of type Sinf as a term that does not contain a semantic subject, but receives it from the matrix clause, which is a complete clause. Since we use the terms of type Sinf for modeling the derivation trees of reduced clauses, one may propose to encode the constant $\mathrm{G}^{\text{red.}}_{\textbf{après}}$ with the type $\mathsf{S} \multimap \mathsf{Sinf} \multimap \mathsf{S}$. However, in that case, one has to express that the subject-sharing takes place between the complete clause (a term of type S) and the reduced one (a term of type Sinf). We propose our solution to this by extending the original G-TAG analysis of reduced conjunctions. Namely, we type the constant $\mathrm{G}^{\text{red.}}_{\textbf{après}}$ with the type $\mathsf{np} \multimap \mathsf{Sws} \multimap \mathsf{Sinf} \multimap \mathsf{S}$, where Sws denotes *a clause missing a subject* obtained by removing the subject from a matrix clause. In this way, we explicitly encode the subject np that is shared by the matrix clause and the infinitive one. Figure 1.8 provides a pictorial representation of this analysis. We annotate each part with the type assigned to the term encoding the corresponding derivation tree. This new analysis can be viewed as an extension of the G-TAG analysis of a sentence with a reduced conjunction (cf. Figure 1.6(b) on the facing page).



(a) $\mathsf{np} \multimap \mathsf{Sws} \multimap \mathsf{Sinf} \multimap \mathsf{S}$

(b) $\mathrm{G}_{\textbf{après}}{}^{\text{red}}_{\text{conj}} : \mathsf{np} \multimap \mathsf{Sws} \multimap \mathsf{Sinf} \multimap \mathsf{S}$

Figure 1.8: The extended G-TAG analysis of a sentence with a reduced conjunction

Thus, to encode g-derivation trees of conjunctions, we introduce constants such as the ones shown in Table 1.2, where the constants model the canonical and reduced underspecified g-derivation trees of après.

Remark 1.2. *By the constant* $\mathrm{G}^{\textit{red.}}_{\textit{conj}}$ *of type* $\mathsf{np} \multimap \mathsf{Sws} \multimap \mathsf{Sinf} \multimap \mathsf{S}$*, one encodes the fact that* Sws *and* Sinf *share an argument of type* np*. Hence, the sharing of a subject is rather a*

[60] We cannot interpret a reduced (infinite) clause as a proposition because we aim at having the semantic interpretations that are similar to conceptual representation inputs of G-TAG. In a G-TAG conceptual representation input, a reduced clause may correspond to an expression that contains a concept standing for a subject.

Constants in Σ_{GTAG}	Their Types
$\mathrm{G}_{\textbf{après}}^{\text{canonical}}$	$\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$
$\mathrm{G}_{\textbf{après}}^{\text{red.}}$	$\mathsf{np} \multimap \mathsf{Sws} \multimap \mathsf{Sinf} \multimap \mathsf{S}$

Table 1.2: The abstract constants encoding après

*particular case of the argument-sharing (*np-sharing*). Below, one may say* subject-sharing *but one has in mind a more generic* argument-sharing.

1.3.2.2 Introducing First Order Predicates in the Abstract Vocabulary

Σ_{GTAG} contains the constants adopted from the signature $\Sigma_{\mathsf{TAG}}^{\mathsf{Der}}$ (from the ACG encoding of TAG derivation trees). They enable us to build terms modeling derivation trees of single clauses. In particular, with the help of the constants adopted from $\Sigma_{\mathsf{TAG}}^{\mathsf{Der}}$, one can produce the terms over Σ_{GTAG} of type S, which model derivation trees of clauses. However, the terms of type S are not the only ones that interact with the constants modeling the discourse connectives. By introducing the abstract constants encoding adverbials and conjunctions, we subsequently introduced types T, Sws and Sinf. In order to make use of these constants, one should be able to produce terms over Σ_{GTAG} of types T, Sws and Sinf.

In order to obtain terms of type T, we use the constants $\mathrm{G}_{\textbf{adv}_{\mathrm{S}}}^{\ \mathrm{S}}$, $\mathrm{G}_{\textbf{adv}_{\mathrm{T}}}^{\ \mathrm{S}}$, $\mathrm{G}_{\textbf{adv}_{\mathrm{S}}}^{\ \mathrm{T}}$, $\mathrm{G}_{\textbf{adv}_{\mathrm{T}}}^{\ \mathrm{S}}$ (see Figure 1.1). With the help of these constants and the terms of type S, we are able to produce terms of type T.

We refer as a *g-derivation tree of a first order predicate*, or simply as a *first order predicate*, to an underspecified g-derivation tree whose variable nodes are the lexicalizations of THINGs. In other words, we refer as a *first order predicate* to an underspecified g-derivation that one uses to generate an *atomic* discourse unit, i.e., clause. Thus, it remains to introduce the constants in Σ_{GTAG} encoding g-derivation trees for the first order predicates of G-TAG that enable producing the terms of types Sws and Sinf. We refer also to these constants as *first order predicates*.

1.3.2.2.1 A Clause Missing a Subject - Sws

We introduced the type Sws in order to encode the type of an argument of a (reduced) conjunction. The difference between the Sws and S types is that a term of type S stands for the derivation tree of a complete clause, i.e., a clause with its own subject and the predicate, whereas a term of type Sws models a clause that misses a subject but contains everything else that a complete clause does. Therefore, an abstract constant encoding an initial tree from which one derives a clause missing a subject should have one less argument than the one that encodes an initial tree from which one derives a complete clause. To model that, we extend Σ_{GTAG} by introducing new constants such as $\mathrm{G}_{\mathsf{v}}^{\mathsf{sws}}$ of type $\vec{\alpha}_n$, whereas the constant G_{v} of type $\vec{\gamma}_{n+1}$ encodes an initial tree form which we derive a complete clause. The types $\vec{\gamma}_{n+1}$ and $\vec{\alpha}_n$ are defined as follows:

- $\vec{\gamma}_{n+1} = \mathsf{a}_1 \multimap \cdots \multimap \mathsf{a}_k \multimap \mathsf{a}_0$, where $\mathsf{a}_0 = \mathsf{S}$ and for some $1 \leq i \leq n$, the argument a_i of G_{v} stands for the subject (thus $\mathsf{a}_i = \mathsf{np}$).

- $\vec{\alpha}_n = \mathsf{a}_1 \multimap \cdots \multimap \mathsf{a}_{i-1} \multimap \mathsf{a}_{i+1} \multimap \cdots \multimap \mathsf{a}_k \multimap \mathsf{Sws}$.

In words, each argument of G_{v} is an argument of $\mathrm{G}_{\mathrm{v}}^{\mathrm{sws}}$ and vice versa except for the argument of G_{v} modeling a subject. Since we encode subjects with NPs, that atomic type is np. For example, $\mathrm{G}_{\textbf{fait-une-sieste}}$ is a constant modeling an initial tree anchored by *fait une sieste*[61] (takes a nap). The type of $\mathrm{G}_{\textbf{fait-une-sieste}}$ is $\mathsf{S}_{\mathsf{A}} \multimap \mathsf{V}_{\mathsf{A}} \multimap \mathsf{np} \multimap \mathsf{S}$. In Σ_{GTAG}, we introduce a constant $\mathrm{G}_{\textbf{fait-une-sieste}}^{\mathrm{sws}}$ of type $\mathsf{S}_{\mathsf{A}} \multimap \mathsf{V}_{\mathsf{A}} \multimap \mathsf{Sws}$.

1.3.2.2.2 Reduced (Infinitive) Clauses - Sinf

We encode a derivation tree of a reduced (infinitive) clause by a term of type Sinf. An infinitive clause has PRO in the syntactic position of a subject (see e.g. Figure 1.7). Therefore, it cannot receive another syntactic subject at that position. Consequently, in order to construct an infinitive clause, one needs to use one less NP compared to the case of a finite, complete clause. Hence, we encode the constants that enable us to build the terms of type Sinf similarly to what we did in the case of Sws. We extend the abstract vocabulary Σ_{GTAG} by adding the new constants such as the constant $\mathrm{G}_{\textbf{vinf}}^{\mathrm{inf}}$ of type $\vec{\beta}_n$, whereas the constant G_{v} of type $\vec{\gamma}_{n+1}$ encodes an initial tree form which one derives a complete (finite) clause; we define the type $\vec{\beta}_n$ as follows:

- $\vec{\gamma}_{n+1} = \mathsf{a}_1 \multimap \cdots \multimap \mathsf{a}_k \multimap \mathsf{a}_0$, where $\mathsf{a}_0 = \mathsf{S}$ and for some $1 \leq i \leq n$, the argument a_i of G_{v} stands for the subject (thus $\mathsf{a}_i = \mathsf{np}$).
- $\vec{\beta}_n = \mathsf{a}_1 \multimap \cdots \multimap \mathsf{a}_{i-1} \multimap \mathsf{a}_{i+1} \multimap \cdots \multimap \mathsf{a}_k \multimap \mathsf{Sinf}$.

In words, the types of the constants G_{v} and $\mathrm{G}_{\textbf{vinf}}^{\mathrm{inf}}$ have the same arguments except that G_{v} has an argument modeling a subject, whereas $\mathrm{G}_{\textbf{vinf}}^{\mathrm{inf}}$ does not. Since we encode subjects with NPs, that argument is of type np. For instance, we introduce in Σ_{GTAG} the constant $\mathrm{G}_{\textbf{faire-une-sieste}}^{\mathrm{inf}}$ of type $\mathsf{S}_{\mathsf{A}} \multimap \mathsf{V}_{\mathsf{A}} \multimap \mathsf{Sinf}$.

Convention: We denote with *vinf* the infinitive form of *v*, where *v* can be a finite verb, verb phrase etc. anchoring an initial tree from which one derives a derived tree of a clause.

1.3.3 Declaring the Abstract Signature Σ_{GTAG} and the Abstract Language

We have constructed the abstract vocabulary Σ_{GTAG} where we encode g-derivation trees of G-TAG. In Σ_{GTAG}, we have the constants (see Table 1.3) with the help of which one builds terms modeling g-derivation trees.

In order to define the abstract language, it remains to specify the distinguished type. We have two candidates, S and T. A term of type S models either a g-derivation tree of a (complete) clause, or a g-derivation tree of a sentence built with a subordinate conjunction. In the rest of the cases, we have terms of type T encoding g-derivation trees of texts. We declare T as the distinguished type. We propose to *transform* a term of type S into a term of type T. For that, we introduce a constant $\mathsf{AnchorT} : \mathsf{S} \multimap \mathsf{T}$. If a term t_{s} is of type S, then the term $t_{\mathrm{T}} = (\mathsf{AnchorT}\ t_{\mathrm{s}})$ is of type T. In words, we

[61] Since *fait une sieste* (take a nap) is an idiom in French, we model it as an abstract constant $\mathrm{G}_{\textbf{fait-une-sieste}}^{\mathrm{inf}}$ (Kobele, 2012).

Constants in Σ_{GTAG}	Their Types
$\mathrm{G}_{\textbf{ensuite}}{}^{\mathsf{S}}_{\mathsf{S}}$	$\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{T}$
$\mathrm{G}_{\textbf{ensuite}}{}^{\mathsf{S}}_{\mathsf{T}}$	$\mathsf{S} \multimap \mathsf{T} \multimap \mathsf{T}$
$\mathrm{G}_{\textbf{ensuite}}{}^{\mathsf{T}}_{\mathsf{S}}$	$\mathsf{T} \multimap \mathsf{S} \multimap \mathsf{T}$
$\mathrm{G}_{\textbf{ensuite}}{}^{\mathsf{T}}_{\mathsf{T}}$	$\mathsf{T} \multimap \mathsf{T} \multimap \mathsf{T}$
$\mathrm{G}_{\textbf{auparavant}}{}^{\mathsf{S}}_{\mathsf{S}}$	$\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{T}$
$\vdots$	$\vdots$
$\mathrm{G}_{\textbf{après}}{}^{\text{canonical}}$	$\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$
$\mathrm{G}_{\textbf{après}}{}^{\text{red.}}$	$\mathsf{np} \multimap \mathsf{Sws} \multimap \mathsf{Sinf} \multimap \mathsf{S}$
$\mathrm{G}_{\textbf{avant}}{}^{\text{canonical}}$	$\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$
$\mathrm{G}_{\textbf{avant}}{}^{\text{red.}}$	$\mathsf{np} \multimap \mathsf{Sws} \multimap \mathsf{Sinf} \multimap \mathsf{S}$
$\mathrm{G}_{\textbf{pour}}{}^{\text{canonical}}$	$\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$
$\mathrm{G}_{\textbf{pour}}{}^{\text{red.}}$	$\mathsf{np} \multimap \mathsf{Sws} \multimap \mathsf{Sinf} \multimap \mathsf{S}$
$\ldots$	$\ldots$
$\mathrm{G}_{\textbf{fait-une-sieste}}$	$\mathsf{S}_{\mathsf{A}} \multimap \mathsf{V}_{\mathsf{A}} \multimap \mathsf{np} \multimap \mathsf{S}$
$\mathrm{G}_{\textbf{fait-une-sieste}}{}^{\text{sws}}$	$\mathsf{S}_{\mathsf{A}} \multimap \mathsf{V}_{\mathsf{A}} \multimap \mathsf{Sws}$
$\mathrm{G}_{\textbf{faire-une-sieste}}{}^{\text{inf}}$	$\mathsf{S}_{\mathsf{A}} \multimap \mathsf{V}_{\mathsf{A}} \multimap \mathsf{Sinf}$
$\ldots$	$\ldots$
$\mathrm{G}_{\textbf{passe-l-aspirateur}}$	$\mathsf{S}_{\mathsf{A}} \multimap \mathsf{V}_{\mathsf{A}} \multimap \mathsf{np} \multimap \mathsf{Sws}$
$\mathrm{G}_{\textbf{passe-l-aspirateur}}{}^{\text{sws}}$	$\mathsf{S}_{\mathsf{A}} \multimap \mathsf{V}_{\mathsf{A}} \multimap \mathsf{Sws}$
$\mathrm{G}_{\textbf{passer-l-aspirateur}}{}^{\text{sws}}$	$\mathsf{S}_{\mathsf{A}} \multimap \mathsf{V}_{\mathsf{A}} \multimap \mathsf{Sws}$
$\ldots$	$\ldots$
$\mathrm{G}_{\textbf{récompense}}$	$\mathsf{S}_{\mathsf{A}} \multimap \mathsf{V}_{\mathsf{A}} \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{S}$
$\mathrm{G}_{\textbf{récompense}}{}^{\text{sws}}$	$\mathsf{S}_{\mathsf{A}} \multimap \mathsf{V}_{\mathsf{A}} \multimap \mathsf{np} \multimap \mathsf{Sws}$
$\mathrm{G}_{\textbf{récompenser}}{}^{\text{inf}}$	$\mathsf{S}_{\mathsf{A}} \multimap \mathsf{V}_{\mathsf{A}} \multimap \mathsf{np} \multimap \mathsf{Sinf}$
$\vdots$	$\vdots$
$\mathsf{AnchorT}$	$\mathsf{S} \multimap \mathsf{T}$

Table 1.3: Constants encoding underspecified g-derivation trees of G-TAG

view a derivation tree of a sentence as a derivation tree of a text consisting of a single sentence. Thus, we declare the abstract language $\mathcal{A}$ as follows:

$$\mathcal{A} = \{t \mid t \in \Lambda(\Sigma_{\mathsf{GTAG}}) \;\&\; t : \mathsf{T}\}$$

1.4 Interpretations as TAG Derivation Trees

G-TAG builds a g-derived tree out of a g-derivation tree. To model that in ACGs, we interpret g-derivation trees as derived trees. As we already mentioned, in order to interpret g-derivation trees as derived trees, we first interpret them as TAG derivation trees. Since TAG derivation trees are already interpreted as TAG derived trees, by composing these two interpretations, one obtains interpretations of g-derivation trees as derived trees. One of the main motivations for this modular approach is that one can see similarities and dissimilarities between TAG and the G-TAG grammar by comparing their derivation trees. Moreover, due to the extended analysis that we propose for reduced conjunctions (see Figure 1.8 on page 205), the terms encoding g-derivation trees of sentences with reduced conjunctions have different structure from g-derivation

trees. However, by interpreting these terms into TAG derivation trees, we obtain terms that have more similar structure to g-derivation trees than the terms over Σ_{GTAG} do.

Thus, we are building a lexicon $\mathscr{L}_{\mathsf{GTAG-TAG}}$ from Σ_{GTAG} to $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ (the signature where one encodes TAG derivation trees) and thereby the ACG $\langle \Sigma_{\mathsf{GTAG}}, \Sigma^{\mathsf{Der}}_{\mathsf{TAG}}, \mathscr{L}_{\mathsf{GTAG-TAG}}, \mathsf{T} \rangle$. Figure 1.9 shows the part of the ACG encoding of G-TAG that we are building now.

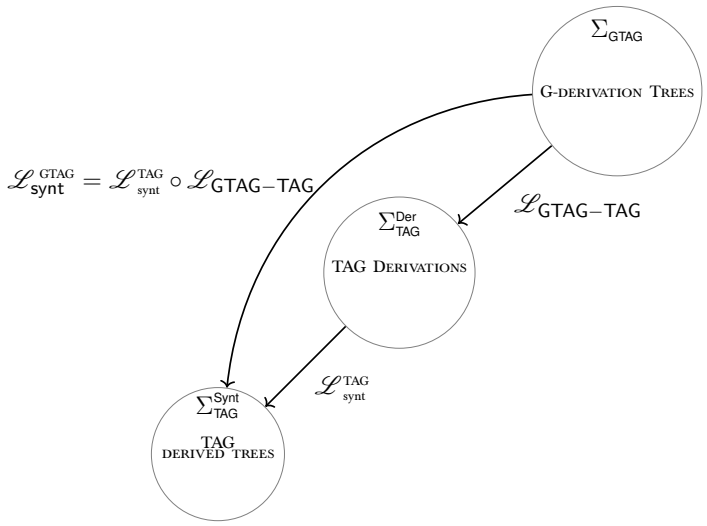


Figure 1.9: Interpretations of g-derivation trees as TAG derivation trees and as TAG derived trees

1.4.1 Interpretations of Types

The type Sws of Σ_{GTAG} encodes the same idea as $\mathsf{np} \multimap \mathsf{S}$ type over $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$, both encode *a clause missing a subject.* The only reason for using Sws in Σ_{GTAG} instead of $\mathsf{np} \multimap \mathsf{S}$ is that we aim at building a second-order abstract vocabulary (because in this case the problems of parsing and generation are polynomial (Kanazawa, 2007)). If we used $\mathsf{np} \multimap \mathsf{S}$ instead of Sws, then the type of a constant $\mathrm{G}^{\text{red.}}_{\mathbf{conj}}$ modeling a reduced conjunction would be $\mathsf{np} \multimap (\mathsf{np} \multimap \mathsf{S}) \multimap \mathsf{Sinf} \multimap \mathsf{S}$. The type $\mathsf{np} \multimap (\mathsf{np} \multimap \mathsf{S}) \multimap \mathsf{Sinf} \multimap \mathsf{S}$ is of order three. This would make the abstract vocabulary of order three as well.

Since Sws and $\mathsf{np} \multimap \mathsf{S}$ encode the same idea, one interprets Sws form g-derivation trees as $\mathsf{np} \multimap \mathsf{S}$ in TAG derivation trees. By interpreting the type Sws from Σ_{GTAG} as $\mathsf{np} \multimap \mathsf{S}$, interpretations of terms encoding g-derivation trees of reduced conjunctions as terms over $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ approximate the structure of the original g-derivation trees more than the terms over Σ_{GTAG} do.

The types T and S, both translate to S in TAG derivation trees, because in G-TAG at the level of derived trees, there is no difference between the features T and S (both are represented by S in g-derived trees).

In addition, we introduced the type Sinf in Σ_{GTAG} in order to type a term encoding a g-derivation tree of an infinitive clause. An infinitive clause is a (syntactically) complete clause whose predicate is an infinite verb form and whose subject is the null pronoun

PRO. Thus, one can represent its derivation tree as a term of type S in TAG derivation trees. Consequently, we interpret Sinf to S under the lexicon $\mathscr{L}_{\mathsf{GTAG}-\mathsf{TAG}}$.

The rest of the types in Σ_{GTAG} were adopted form $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$. If X is one of the adopted types in Σ_{GTAG}, X models the same phenomenon in Σ_{GTAG} as it does in $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$. Thus, we translate X from Σ_{GTAG} to X in $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ (e.g. np to np, n_{A} to n_{A} etc.).

1.4.2 Interpretations of Constants

In addition to types adopted from $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ in Σ_{GTAG}, we adopted constants. They enable us to build terms modeling g-derivation trees of clauses. If a constant $\mathrm{G}_{\mathbf{x}}$ is adopted from $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$, in $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ we have the constant $\mathrm{C}_{\mathbf{x}}$. The constants $\mathrm{G}_{\mathbf{x}}$ and $\mathrm{C}_{\mathbf{x}}$ model the same elementary trees in Σ_{GTAG} and $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ respectively. Therefore, we interpret $\mathrm{G}_{\mathbf{x}}$ from Σ_{GTAG} as $\mathrm{C}_{\mathbf{x}}$ into $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$.

It remains to interpret the constants of Σ_{GTAG} that enable us to encode g-derivation trees of texts. To encode the g-derivation trees of texts, we introduced the constants modeling the underspecified g-derivation trees of the lexical entries of adverbials and conjunctions. In Σ_{GTAG}, we also introduced constants enabling us to build terms of types Sws (a clause missing a subject) and Sinf (an infinitive clause).

1.4.2.1 Adverbials

To interpret constants encoding adverbials from Σ_{GTAG} into $\Lambda(\Sigma^{\mathsf{Der}}_{\mathsf{TAG}})$, we refer to the interpretations of their types. Since the types T and S translate to S, we obtain that the types of the constants $\mathrm{G}_{\mathbf{adv}_{\mathrm{S}}}{}^{\mathrm{S}}$, $\mathrm{G}_{\mathbf{adv}_{\mathrm{T}}}{}^{\mathrm{S}}$, $\mathrm{G}_{\mathbf{adv}_{\mathrm{S}}}{}^{\mathrm{T}}$ and $\mathrm{G}_{\mathbf{adv}_{\mathrm{T}}}{}^{\mathrm{T}}$ translate to $\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$. We introduce a new constant in $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$, namely, the constant $\mathrm{C}^{\mathrm{disc}}_{\mathbf{adv}}$ of type $\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$. We interpret each of the constants $\mathrm{G}_{\mathbf{adv}_{\mathrm{S}}}{}^{\mathrm{S}}$, $\mathrm{G}_{\mathbf{adv}_{\mathrm{T}}}{}^{\mathrm{S}}$, $\mathrm{G}_{\mathbf{adv}_{\mathrm{S}}}{}^{\mathrm{T}}$ and $\mathrm{G}_{\mathbf{adv}_{\mathrm{T}}}{}^{\mathrm{T}}$ as $\mathrm{C}^{\mathrm{disc}}_{\mathbf{adv}}$.

$$\mathscr{L}_{\mathsf{GTAG}-\mathsf{TAG}}(\mathrm{G}_{\mathbf{adv}_{\mathrm{S}}}{}^{\mathrm{S}}) = \mathscr{L}_{\mathsf{GTAG}-\mathsf{TAG}}(\mathrm{G}_{\mathbf{adv}_{\mathrm{T}}}{}^{\mathrm{S}}) = \\ = \mathscr{L}_{\mathsf{GTAG}-\mathsf{TAG}}(\mathrm{G}_{\mathbf{adv}_{\mathrm{S}}}{}^{\mathrm{T}}) = \mathscr{L}_{\mathsf{GTAG}-\mathsf{TAG}}(\mathrm{G}_{\mathbf{adv}_{\mathrm{T}}}{}^{\mathrm{T}}) = \mathrm{C}^{\mathrm{disc}}_{\mathbf{adv}} \quad (1.7)$$

Furthermore, we have to interpret the constant $\mathrm{C}^{\mathrm{disc}}_{\mathbf{adv}}$ of $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ into TAG derived trees, i.e., as a term over $\Sigma^{\mathsf{Synt}}_{\mathsf{TAG}}$. To do so, we refer to the G-TAG analysis of an adverbial (see Figure 1.10). Thus, we define the interpretation of $\mathrm{C}^{\mathrm{disc}}_{\mathbf{adv}}$ as it is shown in Equation (1.8).

$$\mathscr{L}^{\mathrm{TAG}}_{\mathrm{synt}}(\mathrm{C}^{\mathrm{disc}}_{\mathbf{adv}}) = \lambda^{\mathrm{o}} s_1\ s_2.\, \mathsf{S}_3\ s_1\ (\mathsf{Punct}_1\ \textit{dot})\ (\mathsf{S}_2\ (\mathsf{Adv}_2\ \textit{adv}(\mathsf{Punct}_1\ \textit{comma}))\ s_2) : \tau \multimap \tau \multimap \tau \quad (1.8)$$

1.4.2.2 Conjunctions

A constant encoding a conjunction can be either of the following types:

1. $\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$ (the canonical conjunction);
2. $\mathsf{np} \multimap \mathsf{Sws} \multimap \mathsf{Sinf} \multimap \mathsf{S}$ (the reduced conjunction).

We discuss each of these cases separately.

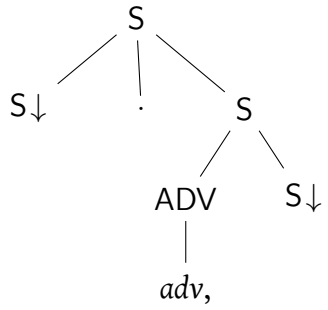


Figure 1.10: The G-TAG elementary tree anchored with an adverbial

1.4.2.2.1 The Canonical Conjunction

If a constant $\mathrm{G}_{\mathbf{conj}}^{\text{canonical}}$ in Σ_{GTAG} is of type $\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$, then we introduce a constant $\mathrm{C}_{\mathbf{conj}}^{\text{canonical}}$ in $\Sigma_{\mathsf{TAG}}^{\mathsf{Der}}$ of type $\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$. We translate the constant $\mathrm{G}_{\mathbf{conj}}^{\text{canonical}}$ to $\mathrm{C}_{\mathbf{conj}}^{\text{canonical}}$:

$$\mathscr{L}_{\mathsf{GTAG}-\mathsf{TAG}}(\mathrm{G}_{\mathbf{conj}}^{\text{canonical}}) = \mathrm{C}_{\mathbf{conj}}^{\text{canonical}} : \mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S} \tag{1.9}$$

We interpret the new constant $\mathrm{C}_{\mathbf{conj}}^{\text{canonical}}$ of $\Sigma_{\mathsf{TAG}}^{\mathsf{Der}}$ to $\Lambda(\Sigma_{\mathsf{TAG}}^{\mathsf{Synt}})$ as the term encoding the G-TAG elementary tree of the canonical underspecified g-derivation tree of the $\underline{\text{conj}}$ lexical entry (see Figure 1.11). Thus, we propose the following interpretation of $\mathrm{C}_{\mathbf{conj}}^{\overline{\text{canonical}}}$ into TAG derived trees:

$$\mathscr{L}_{\text{synt}}^{\text{TAG}}(\mathrm{C}_{\mathbf{conj}}^{\text{canonical}}) = \lambda^{\text{o}}\, s_1\, s_2.\, \mathsf{S}_2\, s_1\, (\mathsf{PP}_2\, (\mathsf{Prep}_1\, \textit{conj})\, s_2) : \tau \multimap \tau \multimap \tau \tag{1.10}$$

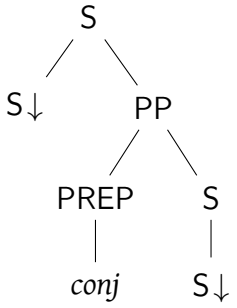


Figure 1.11: The G-TAG elementary tree anchored with a conjunction

1.4.2.2.2 The Reduced Conjunction

The interpretation of the type $\mathsf{np} \multimap \mathsf{Sws} \multimap \mathsf{Sinf} \multimap \mathsf{S}$ from Σ_{GTAG} into $\Sigma_{\mathsf{TAG}}^{\mathsf{Der}}$ is $\Sigma_{\mathsf{TAG}}^{\mathsf{Der}}$ is $\mathsf{np} \multimap (\mathsf{np} \multimap \mathsf{S}) \multimap \mathsf{S} \multimap \mathsf{S}$ (Sws translates to $\mathsf{np} \multimap \mathsf{S}$, and Sinf to S). Figure 1.12 illustrates the intended analysis behind the type $\mathsf{np} \multimap (\mathsf{np} \multimap \mathsf{S}) \multimap \mathsf{S} \multimap \mathsf{S}$. Thus, the interpretation of a constant $\mathrm{G}_{\mathbf{conj}}^{\text{red.}}$ of type $\mathsf{np} \multimap \mathsf{Sws} \multimap \mathsf{Sinf} \multimap \mathsf{S}$ from Σ_{GTAG} into TAG derivation trees can be viewed as follows: It receives an NP (the argument of

type np), a clause missing a subject (the argument of type $\mathsf{np} \multimap \mathsf{S}$), and an infinitive clause (the argument of type S). By combining the NP and the clause missing a subject, one obtains a complete clause. This clause serves as the matrix clause of the sentence, whereas the infinitive clause (Sinf) serves as the subordinate one. To encode this analysis, we propose the following interpretation of the constant $\mathrm{G}^{\text{red.}}_{\textbf{conj}}$:

$$\mathscr{L}^{\text{TAG}}_{\text{synt}}(\mathrm{G}^{\text{red.}}_{\textbf{conj}}) = \lambda^{\text{o}}\ subj.\lambda^{\text{o}}\ s_1.\ \lambda^{\text{o}}\ s_2.\ \mathrm{C}^{\text{red.}}_{\textbf{conj}}\ (s_1\ subj)\ s_2 : \mathsf{np} \multimap (\mathsf{np} \multimap \mathsf{S}) \multimap \mathsf{S} \multimap \mathsf{S} \quad (1.11)$$

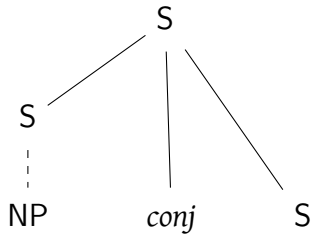


Figure 1.12: The intended meaning behind the type $\mathsf{np} \multimap (\mathsf{np} \multimap \mathsf{S}) \multimap \mathsf{S} \multimap \mathsf{S}$

In the interpretation shown in Equation (1.11), the constant $\mathrm{C}^{\text{red.}}_{\textbf{conj}}$ of type $\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$ is a new constant in $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$. We interpret $\mathrm{C}^{\text{red.}}_{\textbf{conj}}$ as a term over $\Sigma^{\mathsf{Synt}}_{\mathsf{TAG}}$ of type $\tau \multimap \tau \multimap \tau$ modeling the G-TAG elementary tree anchored with *conj.* For instance, in the case of the conjunction *après*, to interpret the constant $\mathrm{C}^{\text{red.}}_{\textbf{après}}$, we refer to the elementary tree in the reduced case (see Figure 1.6(b) on page 204). Thus, we propose the following interpretation:

$$\mathscr{L}^{\text{TAG}}_{\text{synt}}(\mathrm{C}^{\text{red.}}_{\textbf{après}}) = \lambda^{\text{o}}\ s_1\ s_2\ .\ \mathsf{S}_3\ s_1\ (\mathsf{PP}_2\ (\mathsf{Prep}_1\ \textit{après})\ (\mathsf{S}_2\ (\mathsf{C}_1\ \epsilon)\ s_2)) \quad (1.12)$$

1.4.2.3 First Order Predicates

It remains to interpret constants of Σ_{GTAG} that enable us to build terms of types Sinf and Sws into TAG derivation trees.

1.4.2.3.1 A Reduced (Infinitive) Clause

The constant $\mathrm{G}^{\text{inf}}_{\textbf{vinf}} \in \Sigma_{\mathsf{GTAG}}$ encodes the initial tree that gives rise to an infinitive clause. We model the same initial tree with a constant $\mathrm{C}_{\textbf{vinf}}$ in TAG derivation trees. Thus, we interpret $\mathrm{G}^{\text{inf}}_{\textbf{vinf}}$ as $\mathrm{C}_{\textbf{vinf}}$.

1.4.2.3.2 A Clause Missing a Subject

A term of type Sws encodes a derivation tree of a clause whose subject position is unfilled. For instance, let us consider the abstract constant $\mathrm{G}^{\text{sws}}_{\textbf{récompense}}$ of type $\mathsf{S}_{\mathsf{A}} \multimap \mathsf{V}_{\mathsf{A}} \multimap \mathsf{np} \multimap \mathsf{Sws}$. In Σ_{GTAG}, together with $\mathrm{G}^{\text{sws}}_{\textbf{récompense}}$, we have $\mathrm{G}_{\textbf{récompense}}$ of type $\mathsf{S}_{\mathsf{A}} \multimap \mathsf{V}_{\mathsf{A}} \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{S}$. The interpretation of $\mathrm{G}_{\textbf{récompense}}$ in TAG derivation trees is $\mathrm{C}_{\textbf{récompense}}$

as both of them denote an initial tree anchored by *récompense* (reward$_{\text{PRES.3P.SING.}}$). The interpretation of $\mathrm{G}_{\textbf{récompense}}^{\text{sws}}$ in TAG derivation trees should be the same as of the constant $\mathrm{G}_{\textbf{récompense}}$ but with one difference: The interpretation of $\mathrm{G}_{\textbf{récompense}}^{\text{sws}}$ should encode that after that every argument of $\mathrm{G}_{\textbf{récompense}}^{\text{sws}}$ is interpreted, there is still a slot for a subject. To model that, we propose the following interpretation of the constant $\mathrm{G}_{\textbf{récompense}}^{\text{sws}}$:

$$\mathscr{L}(\mathrm{G}_{\textbf{récompense}}^{\text{sws}}) = \lambda^{\text{o}}\, s_a\; v_a\; obj.\, \lambda^{\text{o}} subj.\, \mathrm{C}_{\textbf{récompense}}\; s_a\; v_a\; subj\; obj \tag{1.13}$$

In general, to interpret the constant $\mathrm{G}_{\text{v}}^{\text{sws}}$ of type $\vec{\alpha}_n$,[62] we refer to the interpretation of the constant G_{v} of type $\vec{\gamma}_{n+1}$. The constant G_{v} is of type $\mathsf{a}_1 \multimap \cdots \multimap \mathsf{a}_k \multimap \mathsf{S}$ and $\mathrm{G}_{\text{v}}^{\text{sws}}$ is of type $\mathsf{a}_1 \multimap \cdots \multimap \mathsf{a}_{i-1} \multimap \mathsf{a}_{i+1} \multimap \cdots \multimap \mathsf{a}_k \multimap \mathsf{Sws}$. We propose the interpretation of $\mathrm{G}_{\text{v}}^{\text{sws}}$ as it is shown in Equation (1.14).

$$\mathscr{L}_{\mathsf{GTAG-TAG}}(\mathrm{G}_{\text{v}}^{\text{sws}}) = \lambda^{\text{o}}\, x_1 \cdots x_{i-1}\, x_{i+1} \cdots x_n.\, \lambda^{\text{o}} x_i.\, \mathscr{L}_{\mathsf{GTAG-TAG}}(\mathrm{G}_{\text{v}})\; x_1 \cdots\; x_i\; \cdots x_n \tag{1.14}$$

In Equation (1.14), the variable x_i is of type np. It models the argument standing for a subject. The other variables $x_1 \cdots x_{i-1}\, x_{i+1} \cdots x_n$ model the adjunctions and substitutions in the initial tree modeled by $\mathrm{G}_{\text{v}}^{\text{sws}}$.

Finally, since there is no distinction in TAG derivation trees between features T and S, we translate the constant $\mathsf{AnchorT} : \mathsf{S} \multimap \mathsf{T}$ from Σ_{GTAG} as the identity function $\lambda^{\text{o}} x.x : \mathsf{S} \multimap \mathsf{S}$ in $\Sigma_{\mathsf{TAG}}^{\mathsf{Der}}$.

Example 1.1.

Let us illustrate that the terms over Σ_{GTAG} encoding g-derivation trees of the reduced conjunctions translate to the terms in $\Lambda(\Sigma_{\mathsf{TAG}}^{\mathsf{Der}})$ which resemble more the original g-derivation trees than the terms over Σ_{GTAG} do. For the sake of example, we consider a term $t_{1_{\text{ex}}}^{\text{GTAG}}$ over Σ_{GTAG}, defined as follows:

$$t_{1_{\text{ex}}}^{\text{GTAG}} = \mathsf{AnchorT}(\mathrm{G}_{\textbf{conj}}^{\text{red.}}\; t_{\textbf{subj}}^{\text{NP}}\; t_{\textbf{cms}}^{\text{sws}}\; t_{\textbf{reduced}}^{\text{sinf}}) : \mathsf{T} \tag{1.15}$$

The term $t_{1_{\text{ex}}}^{\text{GTAG}}$ encodes the g-derivation tree shown in Figure 1.13. $t_{\textbf{subj}}^{\text{NP}}$ encodes the shared subject by the clause misting a subject ($t_{\textbf{cms}}^{\text{sws}}$) and the infinitive clause ($t_{\textbf{reduced}}^{\text{sinf}}$). According to the interpretation provided in Equation (1.11) on page 212, we interpret the term $t_{1_{\text{ex}}}^{\text{GTAG}}$ to a term $t_{1_{\text{ex}}}^{\text{TAG}} \in \Lambda(\Sigma_{\mathsf{TAG}}^{\mathsf{Der}})$ as follows:

$$\begin{aligned} t_{1_{\text{ex}}}^{\text{TAG}} = \mathscr{L}_{\mathsf{GTAG-TAG}}(t_{1_{\text{ex}}}^{\text{GTAG}}) = \mathscr{L}_{\mathsf{GTAG-TAG}}(\mathsf{AnchorT}\; \mathrm{G}_{\textbf{conj}}^{\text{red.}}\; t_{\textbf{subj}}^{\text{NP}}\; t_{\textbf{cms}}^{\text{sws}}\; t_{\textbf{reduced}}^{\text{sinf}}) = \\ \mathrm{C}_{\textbf{conj}}^{\text{red.}}\; \underbrace{(\mathscr{L}_{\mathsf{GTAG-TAG}}(t_{\textbf{cms}}^{\text{sws}})\; \mathscr{L}_{\mathsf{GTAG-TAG}}(t_{\textbf{subj}}^{\text{NP}}))}_{t_1^{\text{TAG}}:\mathsf{S}}\; \underbrace{\mathscr{L}_{\mathsf{GTAG-TAG}}(t_{\textbf{reduced}}^{\text{sinf}})}_{t_2^{\text{TAG}}:\mathsf{S}} \end{aligned} \tag{1.16}$$

The structure of the term $t_{1_{\text{ex}}}^{\text{TAG}}$ is closer to the g-derivation tree shown in Figure 1.13 on the next page than the structure of the term $t_{1_{\text{ex}}}^{\text{GTAG}}$ because of the following reasons:

[62] For the definitions of the types $\vec{\alpha}_n$, $\vec{\beta}_n$, and $\vec{\gamma}_n$, we refer readers to Section 1.3.2.2.

- The g-derivation tree of the matrix clause (denoted by g-der-matrix in Figure 1.13) is encoded by the term t_1^{TAG} of type S (see Equation (1.16)).
- The g-derivation tree of the subordinated clause (denoted by g-der-subordinated Figure 1.13) is represented by the term t_2^{TAG} of type S, which stands for the interpretation of a term of type Sinf.

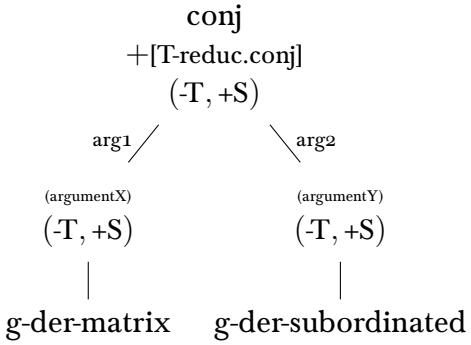


Figure 1.13: A g-derivation tree of a sentence with reduced conjunction

Example 1.2.

To illustrate how one makes use of the ACG encoding of G-TAG in order to generate a derived tree out of a g-derivation one, let us consider a g-derivation tree shown in Figure 1.25 on page 224. It gives rise to a g-derived tree, which can be represented as the syntactic tree (after computing the morphological information) shown in Figure 1.15 on the next page. We encode the g-derivation tree with the term $t_{2_{\text{ex}}}^{\text{GTAG}}$ defined in Equation (1.17). The lexicon $\mathscr{L}_{\mathsf{GTAG}-\mathsf{TAG}}$ interprets the term $t_{2_{\text{ex}}}^{\text{GTAG}}$ as a term $t_{2_{\text{ex}}}^{\text{TAG}} \in \Lambda(\Sigma_{\mathsf{TAG}}^{\mathsf{Der}})$, defined in Equation (1.18).

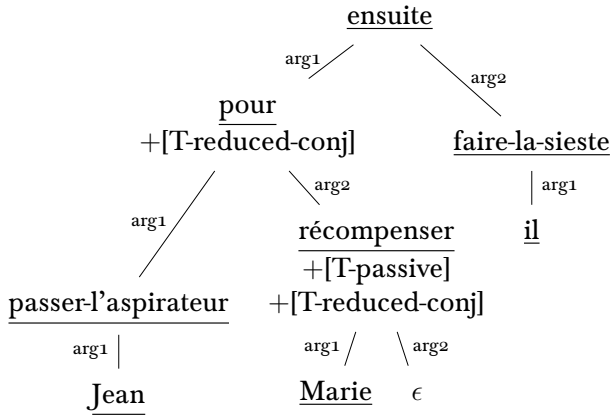


Figure 1.14: A g-derivation tree of a text

$$t_{2_{\text{ex}}}^{\text{GTAG}} = \mathrm{G}_{\mathbf{ensuite}_{\text{s}}}^{\text{s}}\,(\mathrm{G}_{\mathbf{pour}}^{\text{red.}}\ \mathrm{G}_{\mathbf{jean}}(\mathrm{G}_{\mathbf{passe\text{-}laspirateur}}^{\text{sws}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}})\ (\mathrm{G}_{\mathbf{etre\text{-}recomepnse\text{-}par}}^{\text{inf}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{G}_{\mathbf{marie}}))$$
$$(\mathrm{G}_{\mathbf{fait\text{-}une\text{-}sieste}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{G}_{\mathbf{jean}}) : \mathsf{T} \quad (1.17)$$

$$t_{2_{\text{ex}}}^{\text{TAG}} = \mathscr{L}_{\text{GTAG}-\text{TAG}}(t_{2_{\text{ex}}}^{\text{GTAG}}) =$$
$$\mathrm{C}_{\textbf{ensuite}}^{\text{disc}}\ (\mathrm{C}_{\textbf{pour}}^{\text{red.}}(\mathrm{C}_{\textbf{passe-laspirateur}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{C}_{\textbf{jean}})\ (\mathrm{C}_{\textbf{etre-recomepnse-par}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{G}_{\textbf{marie}}))\ (\mathrm{C}_{\textbf{fait-une-sieste}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{C}_{\textbf{jean}})$$
$$: \mathsf{T} \quad (1.18)$$

By interpreting the term $t_{2_{\text{ex}}}^{\text{TAG}}$ under the lexicon $\mathscr{L}_{\text{synt}}^{\text{TAG}}$, we obtain a term of type τ (see Equation 1.19) encoding the derived tree shown in Figure 1.15.

$$\begin{aligned}
&\mathscr{L}_{\text{synt}}^{\text{TAG}}(t_{2_{\text{ex}}}^{\text{TAG}}) = \\
&\mathsf{S}_3 \\
&\quad (\mathsf{S}_2 \\
&\qquad (\mathsf{S}_2\ (\mathsf{NP}_1\ \textit{Jean})(\mathsf{VP}_2\ (\mathsf{AUX}\ \textit{a})(\mathsf{VP}_2\ (\mathsf{V}_1\ \textit{passé})(\mathsf{NP}_2\ (\mathsf{DET}_1\ \textit{l})(\mathsf{N}_1\ \textit{aspirateur}))))) \\
&\qquad (\mathsf{SP}_2 \\
&\qquad\quad (\mathsf{PP}_2\ (\mathsf{PREP}_1\ \textit{pour})\ (C\ \epsilon)) \\
&\qquad\quad (\mathsf{S}_2\ (\mathsf{NP}_1\ \mathsf{PRO})\ (\mathsf{VP}_2\ (\mathsf{V}_2\ \textit{être}\ (\mathsf{V}_1\ \textit{récompensé}))\ (\mathsf{PP}_2\ (\mathsf{P}_1\ \textit{par})\ (\mathsf{NP}_1\ \textit{Marie}))) \\
&\qquad) \\
&\quad) \\
&\quad) \\
&\quad (\mathsf{PUNCT}_1\ \textit{dot}) \\
&\quad (\mathsf{S}_2 \\
&\qquad (\mathsf{ADV}_2\ \textit{Ensuite}\ (\mathsf{PUNCT}_1\ \textit{comma})) \\
&\qquad (\mathsf{S}_2\ (\mathsf{NP}_1\ \textit{Jean})\ (\mathsf{VP}_2\ (\mathsf{AUX}_1\ \textit{a})(\mathsf{VP}_2\ (\mathsf{V}_1\ \textit{fait})\ (\mathsf{NP}_2\ (\mathsf{DET}_1\ \textit{une})\ (\mathsf{N}_1\ \textit{sieste})))))) \\
&\quad)
\end{aligned} \quad (1.19)$$

The lexicon $\mathscr{L}_{\text{yield}}$ interprets the term $\mathscr{L}_{\text{synt}}^{\text{TAG}}(t_{2_{\text{ex}}}^{\text{TAG}})$ as follows:

$$\begin{aligned}
&\mathscr{L}_{\text{yield}}(\mathscr{L}_{\text{synt}}^{\text{TAG}}(t_{\textbf{ex2}}{}^{\text{TAG}})) = \\
&\quad \textit{Jean} + \textit{a} + \textit{passé} + \textit{l} + \textit{aspirateur} + \textit{pour} + \textit{être} + \textit{récompensé} + \textit{par} + \textit{marie} + \textit{dot} \\
&\qquad + \textit{Ensuite} + \textit{comma} + \textit{Jean} + \textit{a} + \textit{fait} + \textit{une} + \textit{sieste}
\end{aligned} \quad (1.20)$$

In Appendix B.1, we provide the code that one can use in order to run this example on the ACG toolkit.

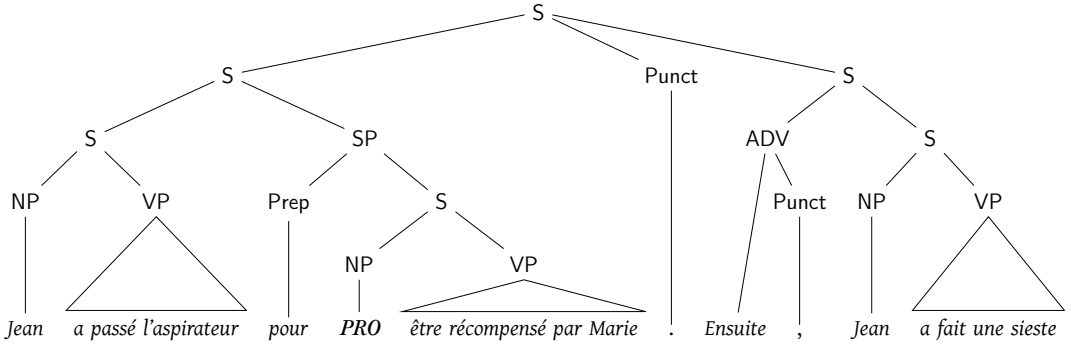


Figure 1.15: A derived tree

1.5 Interpretations as Conceptual Representations

G-TAG builds a g-derivation tree out of a conceptual representation input. We introduce a signature $\Sigma^{\text{Sem}}_{\text{GTAG}}$ to define terms modeling conceptual representation inputs. One refers to $\Sigma^{\text{Sem}}_{\text{GTAG}}$ either as *the semantic signature* or as *conceptual representations.* We call types, constants, and terms built over $\Sigma^{\text{Sem}}_{\text{GTAG}}$ as semantic types, semantic constants, and semantic terms, respectively. Thus, we are building the ACG $\langle \Sigma_{\text{GTAG}}, \Sigma^{\text{Sem}}_{\text{GTAG}}, \mathscr{L}^{\text{sem}}_{\text{GTAG}}, \mathsf{T} \rangle$, denoted with the dotted eclipse in Figure 1.16.

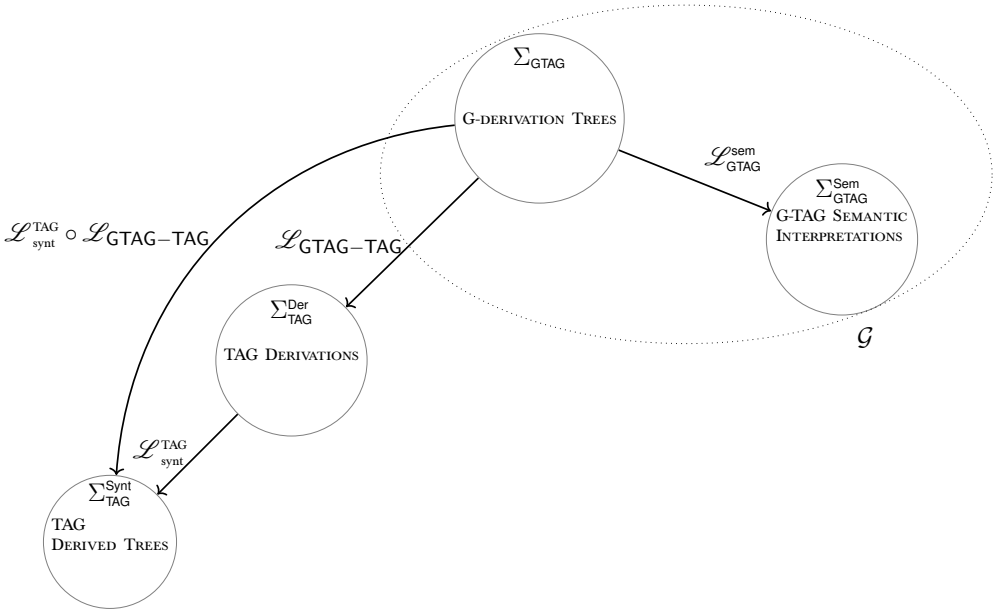


Figure 1.16: An ACG architecture for G-TAG

1.5.1 Encoding Conceptual Representations

Since the conceptual representation language of G-TAG is a sub-language of the language LOGIN (Aït-Kaci and Nasr, 1986), one may try to isomorphically encode ψ-terms of LOGIN as terms over $\Sigma^{\text{Sem}}_{\text{GTAG}}$. However, since the version of ACGs we use relies on simple types, we are not able do that. Instead, we encode a conceptual representation input of G-TAG as a term of the higher-order logic (HOL). Thus, the signature $\Sigma^{\text{Sem}}_{\text{GTAG}}$ is similar to the one defined in the ACG encoding of TAG with Montague Semantics (Pogodalla, 2009). To represent the G-TAG conceptual representations as HOL terms, in the signature $\Sigma^{\text{Sem}}_{\text{GTAG}}$, we introduce the constants and types shown in Figure 1.17.

In general, HOL terms are not isomorphic to ψ-terms because ψ-terms can encode more information than HOL ones. One can think of the ψ-terms as graphs where edges are labeled. However, recall that G-TAG makes use of only specific kinds of ψ-terms. As we have already discussed in the section about G-TAG, the shape of a G-TAG conceptual representation input is a tree.[63] We can encode tree-shaped terms using HOL terms.

[63]In Section 5.2.2.3 on page 158 we described the way a conceptual representation input of G-TAG

Atomic Types :	$e,\ t$
Logical constants :	
$\neg$	$: t \multimap t$
$\wedge,\quad \Rightarrow,\quad \vee$	$: t \multimap t \multimap t$
$\exists,\quad \forall$	$: (e \to t) \multimap t$
Non-logical Constants :	
SUCCESSION, **GOAL**, *etc*	$: t \multimap t \multimap t$
reward, **love**, *etc.*	$: e \multimap e \multimap t$
nap, **vacuum**, *etc.*	$: e \multimap t$
john, **mary**, *etc.*	$: e$

Figure 1.17: The types and constants in $\Sigma^{\mathsf{Sem}}_{\mathsf{GTAG}}$

To illustrate the way we encode conceptual representations as terms over $\Sigma^{\mathsf{Sem}}_{\mathsf{GTAG}}$, let us consider the conceptual representation input of G-TAG given in Figure 1.18. We encode it as the following term over $\Sigma^{\mathsf{Sem}}_{\mathsf{GTAG}}$:

$$s_0 = (\textbf{SUCCESSION}\ (\textbf{GOAL}\ (\textbf{vacuum john})\ (\textbf{reward mary john}))\ \textbf{nap}) \tag{1.21}$$

$$\begin{array}{ll}
\mathtt{E_0} & := \mathtt{SUCCESSION}[1^{\mathtt{st}}\mathtt{EVENT} \Rightarrow \mathtt{E_1},\ 2^{\mathtt{nd}}\mathtt{EVENT} \Rightarrow \mathtt{E_2}] \\
\mathtt{E_1} & := \mathtt{GOAL}[\mathtt{Action} \Rightarrow \mathtt{E_{11}},\ \mathtt{Purpose} \Rightarrow \mathtt{E_{12}}] \\
\mathtt{E_2} & := \mathtt{NAPPING}[\mathtt{NAPPER} \Rightarrow \mathtt{H_1}] \\
\mathtt{E_{11}} & := \mathtt{VACUUMING}[\mathtt{VACUUMER} \Rightarrow \mathtt{H_1}] \\
\mathtt{E_{12}} & := \mathtt{REWARDING}[\mathtt{REWARDER} \Rightarrow \mathtt{H_2},\ \mathtt{REWARDEE} \Rightarrow \mathtt{H_1}] \\
\mathtt{H_1} & := \mathtt{HUMAN}[\mathtt{NAME} \Rightarrow \mathtt{Jean},\ \mathtt{gender} \Rightarrow \mathtt{masc}] \\
\mathtt{H_2} & := \mathtt{HUMAN}[\mathtt{NAME} \Rightarrow \mathtt{Marie},\ \mathtt{gender} \Rightarrow \mathtt{fem}]
\end{array}$$

Figure 1.18: An example of a conceptual input of G-TAG

Our way of transformation is valid for the ψ-terms used in G-TAG as they have a specific form on which our transformation is based.

First, we transform a G-TAG input such as one in Table 1.18 by substituting every occurrence of a label denoting a ψ-term by that ψ-term. For instance, let us consider $\mathtt{H_1}$ in Table 1.18. It denotes the ψ-term $\mathtt{HUMAN}[\mathtt{NAME} \Rightarrow \mathtt{Jean},\ \mathtt{gender} \Rightarrow \mathtt{masc}]$. In any term where $\mathtt{H_1}$ has occurrences, we substitute those occurrences of $\mathtt{H_1}$ with $\mathtt{HUMAN}[\mathtt{NAME} \Rightarrow \mathtt{Jean},\ \mathtt{gender} \Rightarrow \mathtt{masc}]$.

Each concept defined in G-TAG is either a SECOND ORDER RELATION, or a FIRST ORDER RELATION, or a THING. We further process a conceptual input of G-TAG as follows:

can be represented a tree-shaped structure, where the parent-child dependency corresponds to the predicate-argument relation in the conceptual representation input.

- If u is a ψ-term whose root symbol belongs to THINGS (e.g. $\mathtt{H}_1$ in Table 1.18), we encode u by a constant h of type e.
- If u is a ψ-term whose root symbol P belongs to FIRST ORDER RELATIONS and P has n arguments, then u has n sub-ψ-terms under the labels associated with the arguments of P. Let us assume that those sub-ψ-terms are $u_1 \dots u_n$. We encode u by the term s_1 of type t defined as follows:

$$s_1 = \mathbf{p}\,\mathbf{p}_1 \dots \mathbf{p}_n \ : t \tag{1.22}$$

  Where $\mathbf{p}_i$ encodes the sub-ψ-term u_i of the ψ-term u, for $i = 1, \dots, n$. $\mathbf{p}$ encodes the root symbol P of u. We type $\mathbf{p}$ with the type $a_1 \multimap \dots \multimap a_n \multimap t$, where a_i is either e or t depending on the type of $\mathbf{p}_i$ in Equation (1.22), for $i = 1, \dots, n$. The type of $\mathbf{p}_i$ is either e if the root symbol of u_i belongs to THING, or it is t if the root symbol of u_i belongs to FIRST ORDER RELATION, for $i = 1, \dots, n$.
- If u is a ψ-term whose root P is a SECOND ORDER RELATION and P has n arguments,[64] then u has n sub-ψ-terms, denoted by $u_1, \dots, u_n$. Assume that one encodes the ψ-terms $u_1, \dots u_n$ with the terms $\mathbf{p}_1, \dots \mathbf{p}_n$ respectively. Then, we encode u by the term s_2, defined as follows:

$$s_2 = \mathbf{p}\,\mathbf{p}_1 \dots \mathbf{p}_n \ : t \tag{1.23}$$

  Where $\mathbf{p}$ is of type $\underbrace{t \multimap \dots \multimap t}_{n-times} \multimap t$ encoding the root symbol of u, i.e., P.

In order to obtain a HOL term modeling a given conceptual representation input, we select the term that corresponds to the root node in the tree representation of the conceptual representation input. By transforming the selected term into a HOL term using the above-described method, we obtain the HOL term encoding the given conceptual representation input. For example, in the case of the G-TAG input given in Table 1.18, we model it with the HOL term s_0 defined in Equation (1.21). One can check that the HOL term s_0 is obtained by transforming the ψ-term $\mathtt{E}_0$, which serves as the root node in the tree representation of the conceptual representation input shown in Table 1.18.

Remark 1.3. *Here, we have not considered a case where conceptual representations may give rise to noun phrases other than proper names, such as* une grande maison *(a big house). Nevertheless, it is possible to encode them as HOL terms. Indeed, using the ACG encoding of TAG with Montague semantics, we can encode semantic interpretations of various kinds of noun phrases.*

1.5.2 Interpretations of Types

In the signature Σ_{GTAG}, we introduced three types T, Sws, and Sinf in addition to the ones that we adopted from $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ (e.g. S, np, etc.). To interpret the types adopted from $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$, we refer to their semantic interpretations in the ACG encoding of TAG with

[64] In fact, n is 2 because a SECOND ORDER RELATION encodes a discourse relation, which has two arguments.

Montague semantics.[65] Thus, it only remains to interpret T, Sws, and Sinf as types built on the set $\{e, t\}$.

Type T

We interpret the type S as t, following the ACG encoding of TAG.[66] In G-TAG, a text (T) and a sentence (S) both may stand for a surface realization of the same conceptual representation input. In other words, T and S are not distinguished at the semantic-level. Thus, S and T should be interpreted in the same way. Since the interpretation of S is t, we interpret T as t.

Types Sws and Sinf

The lexicon $\mathscr{L}^{\mathsf{sem}}_{\mathsf{GTAG}}$ interprets the types Sinf and Sws as $\mathsf{qnp} \multimap t$, where qnp abbreviates the type $(e \to t) \multimap t$, which is the type standing for the interpretation of np. The type $\mathsf{qnp} \multimap t$ encodes the fact that the interpretations of the terms of the Sinf and Sws types must apply to a subject (a term of type qnp) in order to become terms encoding propositions (terms of type t). Table 1.4 provides the semantic interpretations of the types T, Sinf, and Sws.

Abstract Types in Σ_{GTAG}	Their translations by the lexicon $\mathscr{L}^{\mathsf{sem}}_{\mathsf{GTAG}}$
S, T	t
Sinf, Sws	$\underbrace{((e \to t) \multimap t)}_{\mathsf{qnp}} \multimap t$

Table 1.4: Semantic interpretations of the abstract types

Remark 1.4. *We interpret* np *as* $((e \to t) \multimap t)$*, where a variable of type* e *is non-linearly abstracted, as we use the same variable of type* e *twice in the translation shown in Equation (1.25) (to model the argument-sharing). Nevertheless, the interpretations are almost-linear and thereby the problems of parsing and generation are polynomial (Kanazawa, 2007).*

1.5.3 Interpretations of Constants

Given a concept C and a lexical entry lex.entry that serves as a lexicalization of C, G-TAG records the correspondence between the arguments of C (if any) and the arguments of the lexical entry lex.entry. In order to interpret the abstract constants modeling lex.entry into semantics, we make use of the concept C and the correspondence between the arguments of the concept C and lex.entry.

Σ_{GTAG} contains some constants adopted from $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$. To interpret them into semantics, we refer to the ACG encoding of TAG with Montague semantics. Thus, it remains to interpret the rest of the constants in Σ_{GTAG}, which can be divided into two parts:

[65] See Table 3.5 on page 81.

[66] By translating S to t, the ACG encoding of TAG with Montague semantics (Pogodalla, 2009) follows Montague's (1973) way of identifying types of sentences and propositions.

- Constants encoding the discourse connectives, i.e., the constants encoding lexical entries of adverbials and subordinate conjunctions;
- constants enabling us to construct terms of types Sinf (an infinitive clause) and Sws (a clause missing a subject).

1.5.3.1 Adverbials

We model underspecified g-derivation trees of an adverbial adv with the following four abstract constants:

$$\mathrm{G}_{\mathbf{adv_S}}{}^{\mathrm{S}} : \mathsf{S} \multimap \mathsf{S} \multimap \mathsf{T}$$
$$\mathrm{G}_{\mathbf{adv_T}}{}^{\mathrm{S}} : \mathsf{S} \multimap \mathsf{T} \multimap \mathsf{T}$$
$$\mathrm{G}_{\mathbf{adv_S}}{}^{\mathrm{T}} : \mathsf{T} \multimap \mathsf{S} \multimap \mathsf{T}$$
$$\mathrm{G}_{\mathbf{adv_T}}{}^{\mathrm{T}} : \mathsf{T} \multimap \mathsf{T} \multimap \mathsf{T}$$

The order of the arguments of an abstract constant matches the syntactic order of the arguments of the adverbial. Indeed, the first (resp. second) argument of each of the constants $\mathrm{G}_{\mathbf{adv_S}}{}^{\mathrm{S}}$, $\mathrm{G}_{\mathbf{adv_T}}{}^{\mathrm{S}}$, $\mathrm{G}_{\mathbf{adv_S}}{}^{\mathrm{T}}$, $\mathrm{G}_{\mathbf{adv_T}}{}^{\mathrm{T}}$ models the argument of the adverbial *adv* under the label arg1 (resp. arg2) (see Figure 1.19).

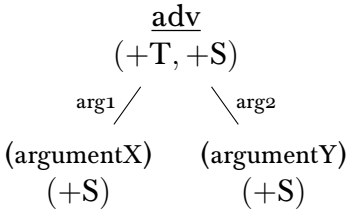
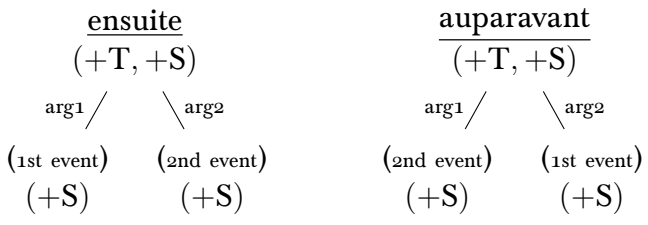


Figure 1.19: A g-derivation tree for the adverbial *adv*

For example, Table 1.5 shows the interpretations of the constants $\mathrm{G}_{\mathbf{ensuite}}$ and $\mathrm{G}_{\mathbf{auparavant}}$ encoding the g-derivation trees of ensuite (afterwards) and auparavant (beforehand) respectively (see Figure 1.20).



(a) (b)

Two lexical entries, ensuite and auparavant

Constants in Σ_{GTAG}	Their semantic interpretation
$\mathrm{G}_{\mathbf{ensuite}_{\mathrm{S}}}^{\ \mathrm{S}}, \mathrm{G}_{\mathbf{ensuite}_{\mathrm{T}}}^{\ \mathrm{S}}, \mathrm{G}_{\mathbf{ensuite}_{\mathrm{S}}}^{\ \mathrm{T}}, \mathrm{G}_{\mathbf{ensuite}_{\mathrm{T}}}^{\ \mathrm{T}}$	$\lambda^{\mathrm{o}} s_1\ s_2.\ \mathbf{SUCCESSION}\ s_1\ s_2 : t \multimap t \multimap t$
$\mathrm{G}_{\mathbf{auparavant}_{\mathrm{S}}}^{\ \mathrm{S}}, \mathrm{G}_{\mathbf{auparavant}_{\mathrm{T}}}^{\ \mathrm{S}}, \mathrm{G}_{\mathbf{auparavant}_{\mathrm{S}}}^{\ \mathrm{T}}, \mathrm{G}_{\mathbf{auparavant}_{\mathrm{T}}}^{\ \mathrm{T}}$	$\lambda^{\mathrm{o}} s_1\ s_2.\ \mathbf{SUCCESSION}\ s_2\ s_1 : t \multimap t \multimap t$

Table 1.5: Semantic interpretations of the constants encoding adverbials

1.5.3.2 Conjunctions

Σ_{GTAG} contains two kinds of constants encoding subordinate conjunctions:

1. The constants of type $\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$, which we use in order to encode canonical g-derivation trees of conjunctions.
2. The constants of type $\mathsf{np} \multimap \mathsf{Sws} \multimap \mathsf{Sinf} \multimap \mathsf{S}$, which we use in order to encode reduced conjunctions, that is, sentences with conjunctions where the *argument-sharing* takes place between the matrix clause and the subordinated one.

Let us discuss these two cases separately.

1.5.3.2.1 Canonical Conjunctions

Since the type S translates to t, the type $\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$ translates to $t \multimap t \multimap t$.

$$\mathrm{G}_{\mathbf{aprés}}^{\text{canonical}} \ = \ \lambda^{\mathrm{o}} s_1\ s_2.\ \mathbf{SUCCESSION}\ s_2\ s_1 :\ t \multimap t \multimap t \tag{1.24}$$

Thus, the types of constants modeling canonical conjunctions and adverbials translate to the same type, i.e., to $t \multimap t \multimap t$. We build the semantic term to which a constant encoding a canonical conjunction translates in the same way as we did in the case of adverbials. For example, Equation (1.24) shows the way one interprets the constant $\mathrm{G}_{\mathbf{aprés}}^{\text{canonical}}$ encoding the canonical g-derivation tree of a subordinate conjunction aprés (after). The interpretation of $\mathrm{G}_{\mathbf{aprés}}^{\text{canonical}}$ encodes that the argument arg1 (resp. arg2) of aprés corresponds to the argument 2nd event (resp. 1st event) of the concept SUCCESSION.

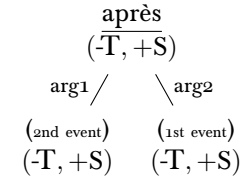


Figure 1.21: The canonical underspecified g-derivation tree of après

1.5.3.2.2 Reduced Conjunctions

We introduced constants of type $\mathsf{np} \multimap \mathsf{Sws} \multimap \mathsf{Sinf} \multimap \mathsf{S}$ in Σ_{GTAG} to model sentences with reduced conjunctions. They encode that the subject of a matrix clause is shared with a subordinated clause, which is a reduced clause. For instance, in the case of the

underspecified derivation tree pour with the feature +[T-reduc.conj] (see Figure 1.23), we use the analysis shown in Figure 1.22. The semantic interpretation of the constant $\mathrm{G}^{\text{red.}}_{\textbf{pour}}$ should pass the subject (the interpretation of the term of type np) to both the interpretation of the term of type Sws and the term of type Sinf. To model this, we propose the interpretation of the constant $\mathrm{G}^{\text{red.}}_{\textbf{pour}}$ shown in Equation (1.25).

$$\underbrace{\underset{\text{Jean}}{\mathsf{np}}}_{\text{shared subject}} \quad \underbrace{\underset{\text{a passé l'aspirateur}}{\mathsf{Sws}}}_{\text{clause without a subject}} \; \textit{pour} \; \underbrace{\underset{\text{être recompense par Marie.}}{\mathsf{Sinf}}}_{\text{infinitve clause}}$$

Jean	*a*	*passé*	*l'aspirateur*	*pour*	*être*	*recompensé*	*par*	*Marie.*
John	have$_{\text{3PS. SG.}}$	pass$_{\text{PAST PART.}}$	vacuumer$_{\text{DEF.}}$	for	to-be$_{\text{PRES. INF.}}$	reward$_{\text{PAST PART.}}$	by	Mary.

John vacuumed in order to be rewarded by Mary.

Figure 1.22: An analysis of a case with a reduced conjunction - the shared subject

$$\begin{aligned}\mathscr{L}^{\mathsf{sem}}_{\text{GTAG}}(\mathrm{G}_{\textbf{pour}}{}^{\text{red.}}) \;&= \lambda^{o}\, subj.\, \lambda^{o} s_1.\, \lambda^{o} s_2.\, subj\ (\lambda x.\, (\textbf{GOAL}\ (s_1\ (\lambda^{o} P.\, P(x)))\ (s_2\ (\lambda^{o} P.\, P(x))))) : \\ &\qquad \mathsf{qnp} \multimap (\mathsf{qnp} \multimap t) \multimap (\mathsf{qnp} \multimap t) \multimap t \\ &\qquad\qquad \text{where: } \mathsf{qnp} \triangleq (e \to t) \multimap t\end{aligned} \tag{1.25}$$

$$\begin{array}{c} \underline{\text{pour}} \\ +\text{[T-reduc.conj]} \\ (-\text{T}, +\text{S}) \\ {}^{\text{arg1}}\!/ \qquad \backslash^{\text{arg2}} \\ (\text{Action}) \qquad (\text{Purpose}) \\ (-\text{T}, +\text{S}) \quad (-\text{T}, +\text{S}) \end{array}$$

Figure 1.23: Pour with +[T-reduc.conj]

The interpretation provided by Equation (1.25) shows that both s_1 and s_2, which are the interpretations of the arguments of type Sws and Sinf, apply to the term $\lambda^{o} P.\, P(x)$ of type qnp, which stands for the interpretation of a term of type np.

1.5.3.3 Reduced (Infinitive) Clauses and Clauses Missing Subjects

We encode a derivation tree of an infinitive clause and a clause missing the subject with the help of the constants $\mathrm{G}^{\text{sws}}_{\textbf{v}}$ and $\mathrm{G}^{\text{inf}}_{\textbf{vinf}}$ of types $\vec{\alpha}_n$ and $\vec{\beta}_n$ respectively.[67] In order to provide their semantic interpretations, we refer to the semantic interpretation of the

[67]We defined $\vec{\alpha}_n$, $\vec{\beta}_n$, and $\vec{\gamma}_n$ within Section 1.3.2.2.1 and Section 1.3.2.2.2 on page 207.

constant $\mathrm{G}_{\mathbf{v}}$ of type $\vec{\gamma}_{n+1}$ encoding the initial tree form which one derives a complete clause. The semantic interpretation of $\mathrm{G}_{\mathbf{v}}$ is already available to us from the ACG encoding of TAG with Montague semantics, as its is one of the constants that we have adopted from $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$.

The interpretation of $\mathrm{G}_{\mathbf{v}}^{\mathrm{sws}}$ and $\mathrm{G}_{\mathbf{vinf}}^{\mathrm{inf}}$ should differ from the interpretation of $\mathrm{G}_{\mathbf{v}}$ by the property that $\mathrm{G}_{\mathbf{v}}^{\mathrm{sws}}$ and $\mathrm{G}_{\mathbf{vinf}}^{\mathrm{inf}}$ obtain subjects at the last place. Thus, one interprets the constants $\mathrm{G}_{\mathbf{v}}^{\mathrm{sws}}$ and $\mathrm{G}_{\mathbf{vinf}}^{\mathrm{inf}}$ as follows:

$$\begin{aligned}\mathscr{L}^{\mathsf{sem}}_{\mathsf{GTAG}}(\mathrm{G}_{\mathbf{v}}^{\mathrm{sws}}) = \mathscr{L}^{\mathsf{sem}}_{\mathsf{GTAG}}(\mathrm{G}_{\mathbf{vinf}}^{\mathrm{inf}}) = \\ \lambda^{\mathrm{o}}\ x_1 \cdots x_{i-1}\ x_{i+1} \cdots x_{n+1}.\ \lambda^{\mathrm{o}}\ subj.\ \mathscr{L}^{\mathsf{sem}}_{\mathsf{GTAG}}(\mathrm{G}_{\mathbf{v}})\ x_1 \cdots x_{i-1}\ subj\ x_{i+1} \cdots x_{n+1}\end{aligned} \tag{1.26}$$

For instance, let us consider the constants $\mathrm{G}_{\mathbf{récompense}}^{\mathrm{sws}} : \mathsf{S}_{\mathsf{A}} \multimap \mathsf{V}_{\mathsf{A}} \multimap \mathsf{np} \multimap \mathsf{Sws}$ and $\mathrm{G}_{\mathbf{récompenser}}^{\mathrm{inf}} : \mathsf{S}_{\mathsf{A}} \multimap \mathsf{V}_{\mathsf{A}} \multimap \mathsf{np} \multimap \mathsf{S}$. We propose the following semantic translations of these constants:

Constants in Σ_{GTAG}	Their interpretations under $\mathscr{L}^{\mathsf{sem}}_{\mathsf{GTAG}}$
$\mathrm{G}_{\mathbf{récompense}}^{\mathrm{sws}}$, $\mathrm{G}_{\mathbf{récompense}}^{\mathrm{inf}}$	$\lambda^{\mathrm{o}}\ s^a\ v^a\ obj.\ \lambda^{\mathrm{o}}\ subj.\ s^a(subj\ (v^a(\lambda\, x.\ (obj\ (\lambda\, y.\ (\mathbf{reward}\ x\ y))))))) :$ $(t \multimap t) \multimap ((e \to t) \multimap (e \to t)) \multimap \mathsf{qnp} \multimap \mathsf{qnp} \multimap t$

Table 1.6: Semantic interpretations of constants of Σ_{GTAG}

Finally, the constant $\mathsf{AnchorT} : \mathsf{S} \multimap \mathsf{T}$ translates to the identity function $\lambda^{\mathrm{o}} x.\, x : t \multimap t$. Indeed, in our encoding, there is no semantic difference between texts and sentences.

Table 1.24 provides the semantic interpretations of the constants and types of Σ_{GTAG}.

Example 1.3.

Let us consider the g-derivation tree in Figure 1.25 on the next page. We encode[68] it by the term $t_{2_{\mathrm{ex}}}^{\mathrm{GTAG}} \in \Lambda(\Sigma_{\mathsf{GTAG}})$ defined in Example 1.2 on page 214 and repeated in Equation (1.27).

$$\begin{aligned}t_{2_{\mathrm{ex}}}^{\mathrm{GTAG}} = \mathrm{G}_{\mathbf{ensuite}_{\mathrm{s}}}^{\ \mathrm{s}}\ (\mathrm{G}_{\mathbf{pour}}^{\mathrm{red.}}\mathrm{G}_{\mathbf{jean}}(\mathrm{G}_{\mathbf{passe\text{-}laspirateur}}^{\mathrm{sws}}\ \mathrm{I}_{\mathsf{S}_{\mathsf{A}}}\ \mathrm{I}_{\mathsf{V}_{\mathsf{A}}}\ )\ (\mathrm{G}_{\mathbf{etre\text{-}recomepnse\text{-}par}}^{\mathrm{inf}}\ \mathrm{I}_{\mathsf{S}_{\mathsf{A}}}\ \mathrm{I}_{\mathsf{V}_{\mathsf{A}}}\ \mathrm{G}_{\mathbf{marie}})) \\ (\mathrm{G}_{\mathbf{fait\text{-}une\text{-}sieste}}\ \mathrm{I}_{\mathsf{S}_{\mathsf{A}}}\ \mathrm{I}_{\mathsf{V}_{\mathsf{A}}}\ \mathrm{G}_{\mathbf{jean}}) : \mathsf{T}\end{aligned} \tag{1.27}$$

The lexicon $\mathscr{L}^{\mathsf{sem}}_{\mathsf{GTAG}}$ interprets the term $t_{2_{\mathrm{ex}}}^{\mathrm{GTAG}}$ as the following term of type t:

$$\begin{aligned}\mathscr{L}^{\mathsf{sem}}_{\mathsf{GTAG}}(t_{2_{\mathrm{ex}}}^{\mathrm{GTAG}}) = \ & \mathbf{SUCCESSION} \\ & (\mathbf{GOAL} \\ & \quad (\mathbf{vacuum\ john}) \\ & \quad (\mathbf{reward\ mary\ john}) \\ &) \\ & (\mathbf{nap\ john}) \\ & : t\end{aligned} \tag{1.28}$$

[68] We provide the ACG code for this example in Appendix B.1 on page 331.

Constants & Types in Σ_{GTAG}	Their interpretations under the lexicon $\mathscr{L}^{\text{sem}}_{\text{GTAG}}$
np	qnp
T, S	t
Sws, Sinf	qnp $\multimap t$
$\text{G}_{\textbf{ensuite}}{}^{\text{S}}_{\text{S}}$	$\lambda^{\text{o}} s_1\ s_2.\ \textbf{SUCCESSION}\ s_1\ s_2$
...	...
$\text{G}_{\textbf{ensuite}}{}^{\text{T}}_{\text{T}}$	$\lambda^{\text{o}} s_1\ s_2.\ \textbf{SUCCESSION}\ s_1\ s_2$
$\text{G}_{\textbf{auparavant}}{}^{\text{S}}_{\text{S}}$	$\lambda^{\text{o}} s_1\ s_2.\ \textbf{SUCCESSION}\ s_2\ s_1$
...	...
⋮	⋮
$\text{G}^{\text{canonical}}_{\textbf{après}}$	$\lambda^{\text{o}} s_1\ s_2.\ \textbf{SUCCESSION}\ s_2\ s_1$
$\text{G}^{\text{red.}}_{\textbf{après}}$	$\lambda^{\text{o}}\ subj.\ \lambda^{\text{o}} s_1.\ \lambda^{\text{o}} s_2.\ subj\ (\lambda x.(\textbf{SUCCESSION}\ (s_2\ (\lambda^{\text{o}} P.\ P(x)))\ (s_1\ (\lambda^{\text{o}} P.\ P(x)))))$
$\text{G}^{\text{canonical}}_{\textbf{avant}}$	$\lambda^{\text{o}} s_1\ s_2.\ \textbf{SUCCESSION}\ s_1\ s_2$
$\text{G}^{\text{red.}}_{\textbf{avant}}$	$\lambda^{\text{o}}\ subj.\ \lambda^{\text{o}} s_1.\ \lambda^{\text{o}} s_2.\ subj\ (\lambda x.(\textbf{SUCCESSION}\ (s_1\ (\lambda^{\text{o}} P.\ P(x)))\ (s_2\ (\lambda^{\text{o}} P.\ P(x)))))$
$\text{G}^{\text{canonical}}_{\textbf{pour}}$	$\lambda^{\text{o}} s_1\ s_2.\ \textbf{GOAL}\ s_1\ s_2$
$\text{G}^{\text{red.}}_{\textbf{pour}}$	$\lambda^{\text{o}}\ subj.\ \lambda^{\text{o}} s_1.\ \lambda^{\text{o}} s_2.\ subj\ (\lambda x.(\textbf{GOAL}\ (s_1\ (\lambda^{\text{o}} P.\ P(x)))\ (s_2\ (\lambda^{\text{o}} P.\ P(x)))))$
...	...
$\text{G}_{\textbf{fait-une-sieste}}$	$\lambda^{\text{o}}\ s^a\ v^a\ subj.\ s^a(subj\ (v^a(\lambda\, x.\ (\textbf{sleep}\ x))))$
$\text{G}^{\text{sws}}_{\textbf{fait-une-sieste}}$	$\lambda^{\text{o}}\ s^a\ v^a.\ \lambda^{\text{o}}\ subj.\ s^a(subj\ (v^a(\lambda\, x.\ (\textbf{sleep}\ x))))$
$\text{G}^{\text{inf}}_{\textbf{faire-une-sieste}}$	$\lambda^{\text{o}}\ s^a\ v^a.\ \lambda^{\text{o}}\ subj.\ s^a(subj\ (v^a(\lambda\, x.\ (\textbf{sleep}\ x))))$
...	...
$\text{G}_{\textbf{passe-l-aspirateur}}$	$\lambda^{\text{o}}\ s^a\ v^a\ subj.\ s^a(subj\ (v^a(\lambda\, x.\ (\textbf{vacuum}\ x))))$
$\text{G}^{\text{sws}}_{\textbf{passe-l-aspirateur}}$	$\lambda^{\text{o}}\ s^a\ v^a.\ \lambda^{\text{o}}\ subj.\ s^a(subj\ (v^a(\lambda\, x.\ (\textbf{vacuum}\ x))))$
$\text{G}^{\text{sws}}_{\textbf{passer-l-aspirateur}}$	$\lambda^{\text{o}}\ s^a\ v^a.\ \lambda^{\text{o}}\ subj.\ s^a(subj\ (v^a(\lambda\, x.\ (\textbf{vacuum}\ x))))$
...	...
$\text{G}_{\textbf{récompense}}$	$\lambda^{\text{o}}\ s^a\ v^a\ subj.\ \lambda^{\text{o}}\ obj.\ s^a(subj\ (v^a(\lambda\, x.\ (obj\ (\lambda\, y.\ (\textbf{reward}\ x\ y))))))$
$\text{G}^{\text{sws}}_{\textbf{récompense}}$	$\lambda^{\text{o}}\ s^a\ v^a\ obj.\ \lambda^{\text{o}}\ subj.\ s^a(subj\ (v^a(\lambda\, x.\ (obj\ (\lambda\, y.\ (\textbf{reward}\ x\ y))))))$
$\text{G}^{\text{inf}}_{\textbf{récompenser}}$	$\lambda^{\text{o}}\ s^a\ v^a\ obj.\ \lambda^{\text{o}}\ subj.\ s^a(subj\ (v^a(\lambda\, x.\ (obj\ (\lambda\, y.\ (\textbf{reward}\ x\ y))))))$
⋮	⋮

Figure 1.24: Semantic translations of the abstract constants and types

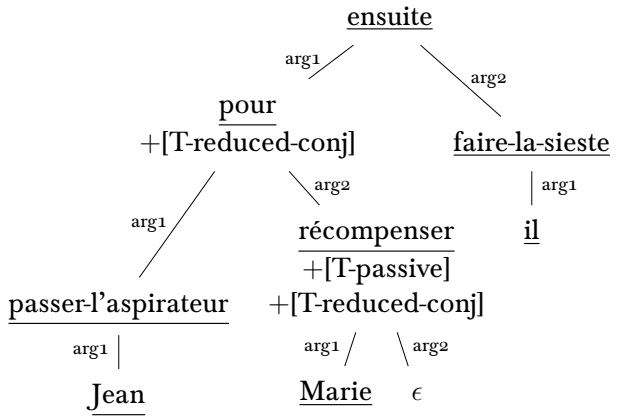


Figure 1.25: A g-derivation tree of a text

1.6 Parsing and Generation Using the ACG encoding of G-TAG

The abstract vocabulary of the ACG architecture for G-TAG is a second-order one and the interpretations are almost-linear. Therefore, the complexities of the problems of the parsing and generation with the ACG encoding of G-TAG are polynomial (Kanazawa, 2007). In order to simulate the G-TAG text generation process, for a given semantic formula u, we compute the set T_u of terms over Σ_{GTAG} such that the lexicon $\mathscr{L}^{\mathsf{sem}}_{\mathsf{GTAG}}$ interprets each element of T_u as u. In order to obtain the derived trees corresponding to the input u, one applies the $\mathscr{L}^{\mathsf{GTAG}}_{\mathsf{synt}}$ lexicon to the elements of T_u. To obtain surface realizations, one interprets the elements of T_u under the lexicon $\mathscr{L}_{\mathsf{yield}} \circ \mathscr{L}^{\mathsf{TAG}}_{\mathsf{synt}} \circ \mathscr{L}_{\mathsf{GTAG-TAG}}$.

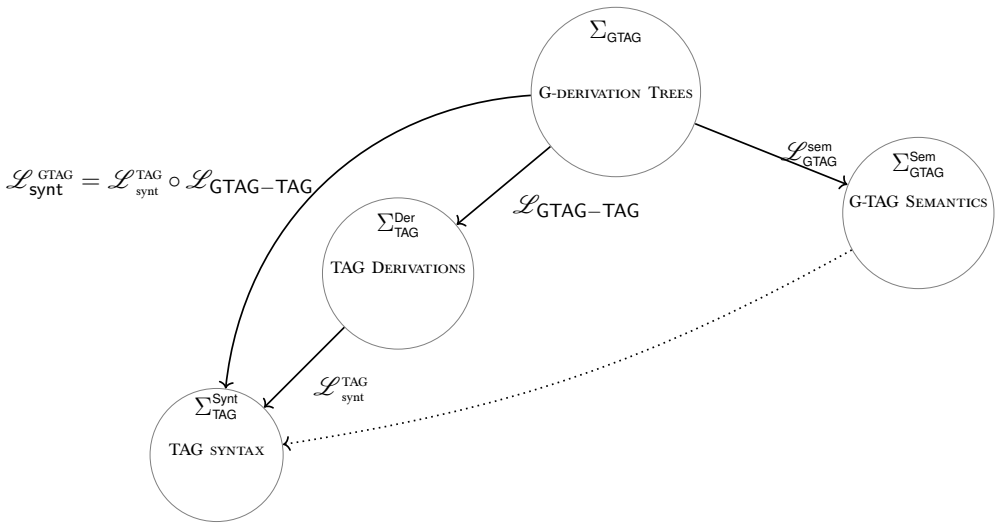


Figure 1.26: The ACG architecture for G-TAG

Notice that G-TAG selects only one surface form as the final output of its generation process. In this respect, our approach differs from G-TAG. Nevertheless, it is possible to apply a ranking strategy on the generated surface forms and select only one of them as a final result. However, in this thesis, we do not use any ranking strategy. Thus, the final output may consist of several surface forms.

Example 1.4.

Let us consider the following example which demonstrates the text generation process within the ACG architecture that we propose.

Let $u = \textbf{SUCCESSION}\ (\textbf{vacuum j})(\textbf{nap j}) : t$. We employ a sample ACG signatures encoding G-TAG grammar shown in Appendix B.3. As a result of parsing u with the lexicon $\mathscr{L}^{\mathsf{sem}}_{\mathsf{GTAG}}$, we obtain the terms listed in Figure 1.27. These terms encode various g-derivation trees that G-TAG could built out of the G-TAG conceptual representation input modeled by u.

We map the terms shown in Figure 1.27 using the lexicon $\mathscr{L}^{\mathsf{TAG}}_{\mathsf{synt}} \circ \mathscr{L}_{\mathsf{GTAG-TAG}}$ in order to produce the terms encoding their derived trees. We obtain the terms defined in

Equations (1.29),(1.30), (1.31), (1.32), (1.33), and (1.34). These terms encode derived trees depicted in Figure 1.28, Figure 1.29, Figure 1.30, Figure 1.31, Figure 1.32, and Figure 1.33, respectively. By mapping these derived trees under the lexicon $\mathscr{L}_{\mathsf{yield}}$, one obtains the strings, i.e., surface retaliations shown in Figure 1.34 on page 230.

$t_{1.1}$	$\mathrm{G}_{\textbf{ensuite}_{\mathrm{S}}}^{\mathrm{S}}\ (\mathrm{G}_{\textbf{passé-laspirateur}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{G}_{\textbf{a}}\ \mathrm{G}_{\textbf{jean}})\ (\mathrm{G}_{\textbf{fait-une-sieste}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{G}_{\textbf{a}}\ \mathrm{G}_{\textbf{il}})\ :\ \mathsf{T}$
$t_{1.2}$	$\mathrm{G}_{\textbf{ensuite}_{\mathrm{T}}}^{\mathrm{T}}\ (\mathsf{AnchorT}(\mathrm{G}_{\textbf{passé-laspirateur}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{G}_{\textbf{a}}\ \mathrm{G}_{\textbf{jean}}))\ (\mathsf{AnchorT}(\mathrm{G}_{\textbf{fait-une-sieste}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{G}_{\textbf{a}}\ \mathrm{G}_{\textbf{il}}))\ :\ \mathsf{T}$
$t_{1.3}$	$\mathrm{G}_{\textbf{ensuite}_{\mathrm{S}}}^{\mathrm{T}}\ (\mathsf{AnchorT}\ (\mathrm{G}_{\textbf{passé-laspirateur}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{G}_{\textbf{a}}\ \mathrm{G}_{\textbf{jean}}))\ (\mathrm{G}_{\textbf{fait-une-sieste}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{G}_{\textbf{a}}\ \mathrm{G}_{\textbf{il}})\ :\ \mathsf{T}$
$t_{1.4}$	$\mathrm{G}_{\textbf{ensuite}_{\mathrm{T}}}^{\mathrm{S}}\ (\mathrm{G}_{\textbf{passé-laspirateur}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{G}_{\textbf{a}}\ \mathrm{G}_{\textbf{jean}})\ (\mathsf{AnchorT}\ (\mathrm{G}_{\textbf{fait-une-sieste}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{G}_{\textbf{a}}\ \mathrm{G}_{\textbf{il}}))\ :\ \mathsf{T}$
$t_{2.1}$	$\mathrm{G}_{\textbf{auparavant}_{\mathrm{S}}}^{\mathrm{S}}\ (\mathrm{G}_{\textbf{fait-une-sieste}_{\mathrm{S}}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{G}_{\textbf{a}}\ \mathrm{G}_{\textbf{jean}})\ (\mathrm{G}_{\textbf{passé-laspirateur}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{G}_{\textbf{a}}\ \mathrm{G}_{\textbf{il}})\ :\ \mathsf{T}$
$t_{2.2}$	$\mathrm{G}_{\textbf{auparavant}_{\mathrm{T}}}^{\mathrm{T}}\ (\mathsf{AnchorT}\ (\mathrm{G}_{\textbf{fait-une-sieste}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{G}_{\textbf{a}}\ \mathrm{G}_{\textbf{jean}}))\ (\mathsf{AnchorT}\ (\mathrm{G}_{\textbf{passé-laspirateur}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{G}_{\textbf{a}}\ \mathrm{G}_{\textbf{il}}))\ :\ \mathsf{T}$
$t_{2.3}$	$\mathrm{G}_{\textbf{auparavant}_{\mathrm{S}}}^{\mathrm{T}}\ (\mathrm{G}_{\textbf{fait-une-sieste}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{G}_{\textbf{a}}\ \mathrm{G}_{\textbf{jean}})\ (\mathsf{AnchorT}\ (\mathrm{G}_{\textbf{passé-laspirateur}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{G}_{\textbf{a}}\ \mathrm{G}_{\textbf{il}}))\ :\ \mathsf{T}$
$t_{2.4}$	$\mathrm{G}_{\textbf{auparavant}_{\mathrm{T}}}^{\mathrm{S}}\ (\mathsf{AnchorT}\ (\mathrm{G}_{\textbf{fait-une-sieste}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{G}_{\textbf{a}}\ \mathrm{G}_{\textbf{jean}}))\ (\mathrm{G}_{\textbf{passé-laspirateur}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{G}_{\textbf{a}}\ \mathrm{G}_{\textbf{il}})\ :\ \mathsf{T}$
$t_{3.1}$	$\mathsf{AnchorT}(\mathrm{G}_{\textbf{avant}}^{\text{canonical}}\ (\mathrm{G}_{\textbf{passé-laspirateur}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{G}_{\textbf{a}}\ \mathrm{G}_{\textbf{jean}})\ (\mathrm{G}_{\textbf{fasse-une-sieste}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{G}_{\textbf{il}}))\ :\ \mathsf{T}$
$t_{3.2}$	$\mathsf{AnchorT}(\mathrm{G}_{\textbf{avant}}^{\text{red.}}\ \mathrm{G}_{\textbf{jean}}\ (\mathrm{G}_{\textbf{passé-laspiratuer}}^{\text{sws}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{G}_{\textbf{a}})\ (\mathrm{G}_{\textbf{faire-une-sieste}}^{\text{sinf}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}))\ :\ \mathsf{T}$
$t_{4.1}$	$\mathsf{AnchorT}(\mathrm{G}_{\textbf{après}}^{\text{canonical}}\ (\mathrm{G}_{\textbf{fait-une-sieste}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{G}_{\textbf{a}}\ \mathrm{G}_{\textbf{jean}})\ (\mathrm{G}_{\textbf{passé-laspirateur}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{G}_{\textbf{a}}\ \mathrm{G}_{\textbf{il}}))\ :\ \mathsf{T}$
$t_{4.2}$	$\mathsf{AnchorT}\ (\mathrm{G}_{\textbf{après}}^{\text{red.}}\ \mathrm{G}_{\textbf{jean}}\ (\mathrm{G}_{\textbf{fait-une-sieste}}^{\text{sws}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{G}_{\textbf{a}})\ (\mathrm{G}_{\textbf{passer-laspiratuer}}^{\text{sinf}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}))\ :\ \mathsf{T}$

Figure 1.27: Terms modeling g-derivation trees

$$\mathscr{L}_{\mathsf{synt}}^{\text{GTAG}}(t_{1.1}) = \mathscr{L}_{\mathsf{synt}}^{\text{GTAG}}(t_{1.2}) = \mathscr{L}_{\mathsf{synt}}^{\text{GTAG}}(t_{1.3}) = \mathscr{L}_{\mathsf{synt}}^{\text{GTAG}}(t_{1.4}) = \tag{1.29}$$

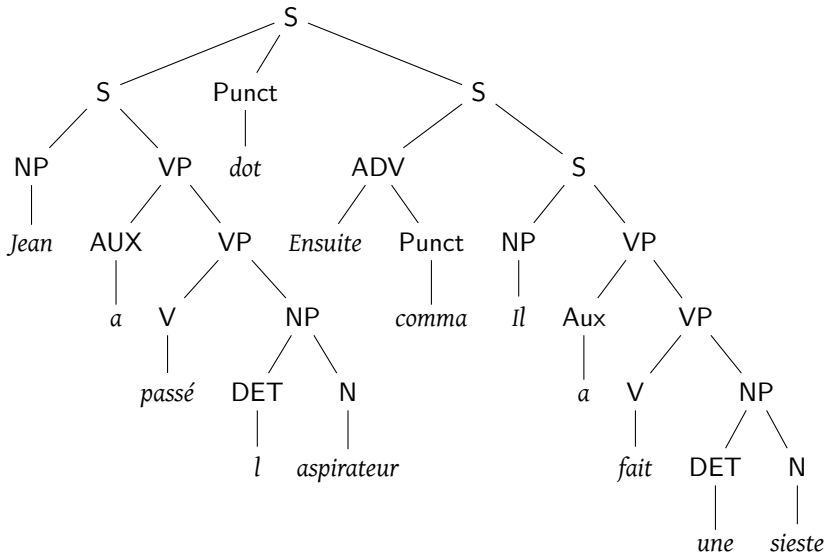


Figure 1.28: A derived tree obtained as the interpretation of the terms encoding g-derivation trees

$$\mathscr{L}_{\text{synt}}^{\text{GTAG}}(t_{2.1}) = \mathscr{L}_{\text{synt}}^{\text{GTAG}}(t_{2.2}) = \mathscr{L}_{\text{synt}}^{\text{GTAG}}(t_{2.3}) = \mathscr{L}_{\text{synt}}^{\text{GTAG}}(t_{2.4}) = \tag{1.30}$$

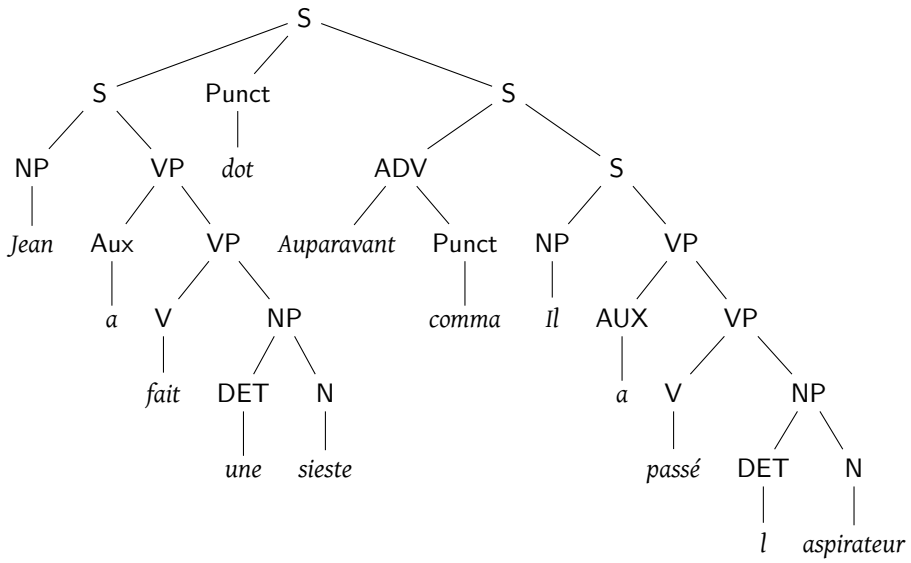


Figure 1.29: A derived tree obtained as the interpretation of the terms encoding g-derivation trees

$$\mathscr{L}_{\text{synt}}^{\text{GTAG}}(t_{3.1}) \quad = \tag{1.31}$$

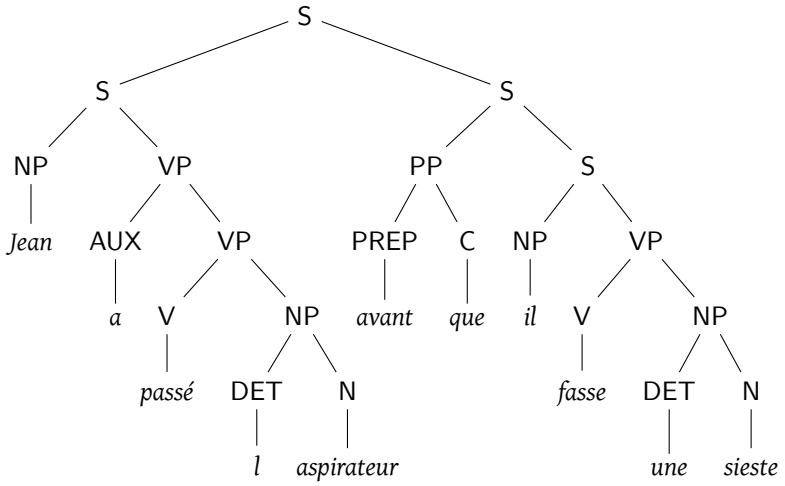


Figure 1.30: A derived tree obtained as the interpretation of a term encoding a g-derivation tree

$$\mathscr{L}^{\text{GTAG}}_{\text{synt}}(t_{3.2}) \quad = \tag{1.32}$$

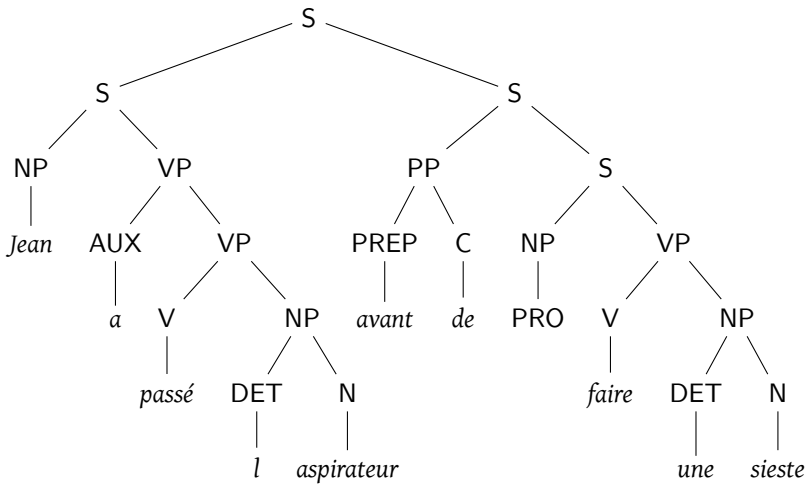


Figure 1.31: A derived tree obtained as the interpretation of a term encoding a g-derivation tree

$$\mathscr{L}^{\text{GTAG}}_{\text{synt}}(t_{4.1}) \quad = \tag{1.33}$$

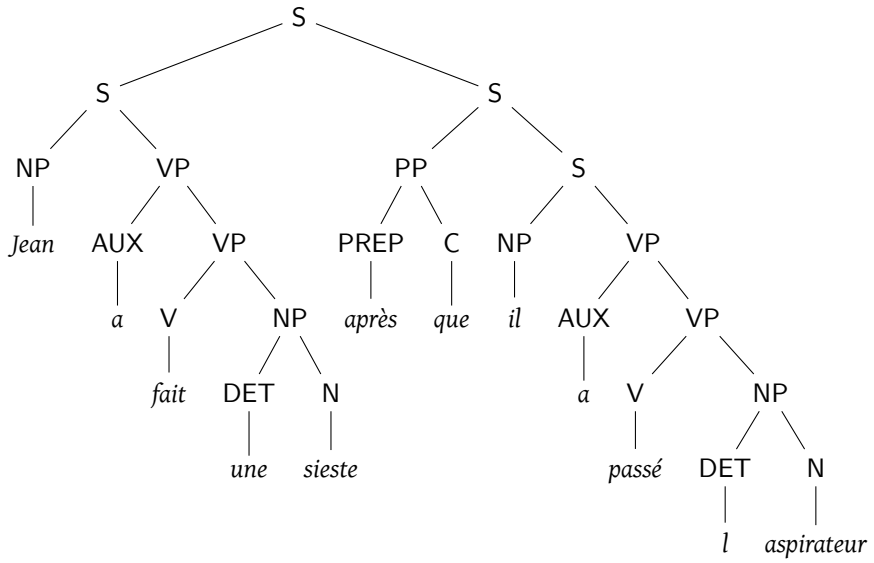


Figure 1.32: A derived tree obtained as the interpretation of a term encoding a g-derivation tree

$$\mathscr{L}_{\mathsf{synt}}^{\mathrm{GTAG}}(t_{4.2}) \quad = \tag{1.34}$$

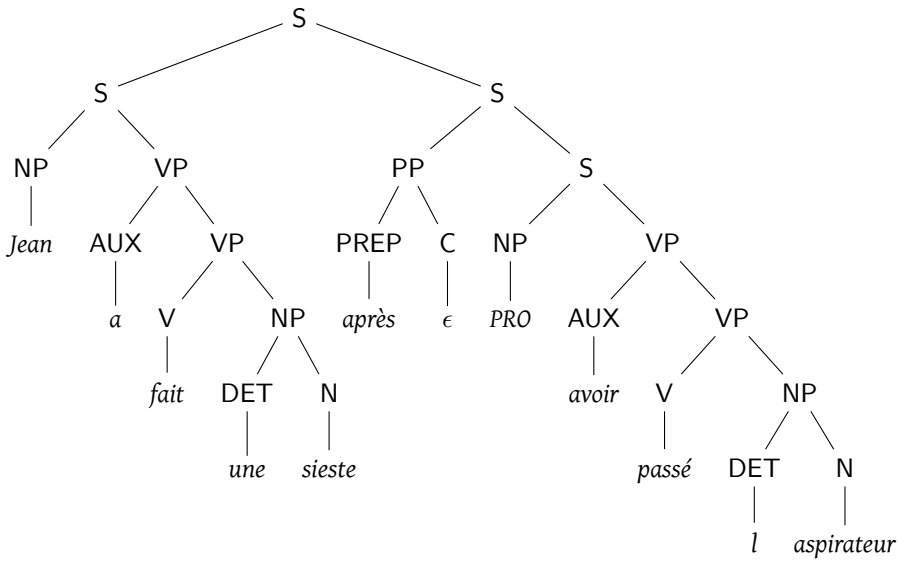


Figure 1.33: A derived tree obtained as the interpretation of a term encoding a g-derivation tree

Remark 1.5. *The terms* $\mathscr{L}_{\mathsf{synt}}^{GTAG}(t_{1.1})$, $\mathscr{L}_{\mathsf{synt}}^{GTAG}(t_{1.2})$, $\mathscr{L}_{\mathsf{synt}}^{GTAG}(t_{1.3})$ *and* $\mathscr{L}_{\mathsf{synt}}^{GTAG}(t_{1.4})$ *are the same (see Equation (1.29)), that is, the syntactic interpretations of the terms* $t_{1.1}$, $t_{1.2}$, $t_{1.3}$, *and* $t_{1.4}$ *are the same (the same is true in the case of* $t_{2.1}$, $t_{2.2}$, $t_{2.3}$, *and* $t_{2.4}$*). This is due to the fact that in* Σ_{GTAG}, *we introduced four constants for an adverbial all of which have the same syntactic interpretations, and at the same time,* $\mathsf{AnchorT} \in \Sigma_{GTAG}$ *translates to the identity function under the lexicon* $\mathscr{L}_{\mathsf{synt}}^{GTAG}$. *Instead of having four constants for an adverbial, we can have only one* $\mathrm{G}_{adv_T}^{\ T} : T \multimap T \multimap T$ *but we can still produce the same set of syntactic interpretations. For example, if we have* $\mathrm{G}_{adv_S}^{\ S}\ t_1\ t_2$, *where* t_1 *and* t_2 *are of type* S, *we define the term* $\mathrm{G}_{adv_T}^{\ T}(\mathsf{AnchorT}\ t_1)\,(\mathsf{AnchorT}\ t_2)$. *The terms* $\mathrm{G}_{adv_T}^{\ T}(\mathsf{AnchorT}\ t_1)\,(\mathsf{AnchorT}\ t_2)$ *and* $\mathrm{G}_{adv_S}^{\ S}\ t_1\ t_2$ *have the same translations under any lexicon defined by us. Thus, it is sufficient to have instead of the four constants* $\mathrm{G}_{adv_S}^{\ S}$, $\mathrm{G}_{adv_T}^{\ S}$, $\mathrm{G}_{adv_S}^{\ T}$, *and* $\mathrm{G}_{adv_T}^{\ T}$, *only one constant* $\mathrm{G}_{adv_T}^{\ T}$ *of type* $T \multimap T \multimap T$.

In the rest of this thesis, we assume that in Σ_{GTAG} *we have only one constant* $\mathrm{G}_{adv_T}^{\ T} : T \multimap T \multimap T$ *encoding the G-TAG lexical entry of the adverbial* adv.

As the generated surface realizations illustrate (see Figure 1.34 on the next page), the adverbials *ensuite* and *auparavant* occupy only clause-initial positions. However, G-TAG could output a text where an adverbial appears at a clause-medial position. G-TAG does so by first generating a text where all the adverbials are at the clause-initial positions, and then, the post-processing module may move an adverbial from a clause-initial position to a clause-medial one. The G-TAG way of producing texts with clause-medial adverbials is not an option for us because we do not employ any kind of extra processing step. In the next chapter, we provide a solution that allows one to

$\mathscr{L}_{\mathsf{yield}}(\mathscr{L}_{\mathsf{synt}}^{\text{GTAG}}(t_{1.1})) = \mathscr{L}_{\mathsf{yield}}(\mathscr{L}_{\mathsf{synt}}^{\text{GTAG}}(t_{1.2})) = \mathscr{L}_{\mathsf{yield}}(\mathscr{L}_{\mathsf{synt}}^{\text{GTAG}}(t_{1.3})) = \mathscr{L}_{\mathsf{yield}}(\mathscr{L}_{\mathsf{synt}}^{\text{GTAG}}(t_{1.4})) =$
Jean+a+passé+l+aspirateur+dot+Ensuite+comma+Il+a+fait+une+sieste

$\mathscr{L}_{\mathsf{yield}}(\mathscr{L}_{\mathsf{synt}}^{\text{GTAG}}(t_{2.1})) = \mathscr{L}_{\mathsf{yield}}(\mathscr{L}_{\mathsf{synt}}^{\text{GTAG}}(t_{2.2})) = \mathscr{L}_{\mathsf{yield}}(\mathscr{L}_{\mathsf{synt}}^{\text{GTAG}}(t_{2.3})) = \mathscr{L}_{\mathsf{yield}}(\mathscr{L}_{\mathsf{synt}}^{\text{GTAG}}(t_{2.4})) =$
Jean+a+fait+une+sieste+dot+Auparavant+comma+Il+a+passé+l+aspirateur

$\mathscr{L}_{\mathsf{yield}}(\mathscr{L}_{\mathsf{synt}}^{\text{GTAG}}(t_{3.1})) =$
Jean+a+passé+l+aspirateur+avant+que+il+fasse+une+sieste

$\mathscr{L}_{\mathsf{yield}}(\mathscr{L}_{\mathsf{synt}}^{\text{GTAG}}(t_{3.2})) =$
Jean+a+passé+l+aspirateur+avant+de+faire+une+sieste

$\mathscr{L}_{\mathsf{yield}}(\mathscr{L}_{\mathsf{synt}}^{\text{GTAG}}(t_{4.1})) =$
Jean+a+fait+une+sieste+après+que+il+a+passé+l+aspirateur

$\mathscr{L}_{\mathsf{yield}}(\mathscr{L}_{\mathsf{synt}}^{\text{GTAG}}(t_{4.2})) =$
Jean+a+fait+une+sieste+après+avoir+passé+l+aspirateur

Figure 1.34: Surface realizations

encode texts with clause-medial connectives without making use of an extra-processing step.

Chapter 2

Encoding Clause-Medial Connectives

Contents

We extend the ACG encoding of G-TAG with the aim of modeling texts containing clause-medial connectives. In contrast to G-TAG, where one makes use of a post-processing module in order to transform a text containing clause-initial connectives into a text containing clause-medial connectives, we propose an encoding that makes it possible to generate the texts containing clause-medial connectives without applying a post processing step. Our approach to texts containing with clause-medial connectives is purely grammatical, like it is in the case of texts containing no clause-medial but only clause-initial connectives, in the ACG encoding of G-TAG. In particular, we introduce a constant in the abstract vocabulary to model a clause-medial connective. This enables us to define an abstract term modeling a g-derivation tree of a text containing the clause-medial connective. By interpreting this abstract term, one obtains the text containing the clause-medial connective.

2.1 Encoding Clause-Medial Connectives

In G-TAG, to generate a text where a discourse connective appears at the clause-medial position, G-TAG first generates the canonical text with the help of its discourse grammar.

In the canonical text each connective occupies a clause-initial position. Afterwards, the G-TAG post processing module modifies the canonical text by moving some discourse adverbials from clause-initial positions to clause-medial ones. For example, to generate the discourse (30), the post-processing module of G-TAG modifies the canonical text (29) by moving the discourse adverbial *ensuite* (then) from the clause-initial position to the clause-medial one.

Since we make use of no extra processing step, we aim at the modeling of texts containing clause-medial connectives in *one step*.

(29) *Jean a passé l'aspirateur pour être récompensé par Marie.*
John have$_{3PS.\ SG.}$ pass$_{PAST\ PART.}$ vacuumer$_{DEF.}$ for to-be$_{PRES.\ INF.}$ reward$_{PAST\ PART.}$ by Mary.
Ensuite, *il a fait une sieste.*
Afterward, he have$_{3PS.\ SG.}$ make$_{3PS.\ SG.}$ a nap.

John vaccumed in order to be rewarded by Mary. Then, he took a nap.

(30) *Jean a passé l'aspirateur pour être récompensé par Marie. Il*
John have$_{3PS.\ SG.}$ pass$_{PAST\ PART.}$ vacuumer$_{DEF.}$ for to-be$_{PRES.\ INF.}$ reward$_{PAST\ PART.}$ by Mary. He
a ensuite *fait une sieste.*
have$_{3PS.\ SG.}$ *afterward* make$_{3PS.\ SG.}$ a nap.

John vacuumed in order to be rewarded by Mary. He then took a nap.

2.1.1 A New Analysis of Clause-Medial Connectives

The objective is to encode an adverbial connective at a clause-medial position without deviating significantly from the general principles of G-TAG regarding discourse connectives. In G-TAG, a connective anchors an initial tree with two substitution sites, as it is illustrated in Figure 2.1. By filling the substitution sites in this tree, one obtains a derived tree of a discourse. The yield of the derived tree is a discourse (text) where the adverbial *adv* occupies a *clause-initial* position.

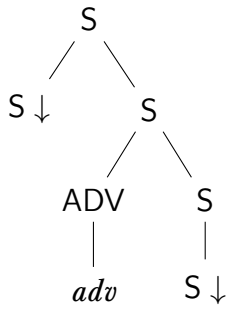
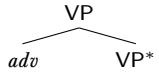
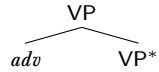


Figure 2.1: The G-TAG initial tree anchored with an adverbial

In LTAG (Abeillé, 1988; XTAG-Group, 1998), a discourse adverbial *adv* at the clause-medial position anchors a VP-rooted auxiliary tree 

. We cannot

adopt the LTAG analysis of clause-medial adverbial connectives because (a) in G-TAG, an elementary tree of an adverbial is an initial tree, whereas in LTAG, an adverbial anchors an auxiliary tree; (b) the LTAG analysis of an adverbial associates it only with one argument, whereas in G-TAG it has two arguments (obtained by substituting two trees into the initial tree anchored with the adverbial).

We propose to combine the properties of the LTAG auxiliary tree and the G-TAG initial tree anchored with an adverbial. In particular, a *construction* corresponding to the adverbial *adv* should have two substitution sites (in the style of G-TAG); and the auxiliary tree anchored with the adverbial *adv* should adjoin on the VP node into a tree (in the style of LTAG) that substitutes at one of these substitution sites. Figure 2.2 on the following page illustrates our analysis: In a tree anchored with the full stop, there are two S-substitution sites; an auxiliary tree 

adjoins on the VP-node of a substituted tree that gives rise to the rightmost piece of the discourse (the one that substitutes at the address 3).

Remark 2.1. *As Figure 2.2 indicates, we do not analyze a clause-medial adverbial as a tree, but as a set of trees* $\{\alpha, \beta\}$. *The tree* α *has two substitution sites. Two trees,* γ_1 *and* γ_2*, fill the substitution sites of* α*, at the addresses* 1 *and* 3*, respectively. The tree* β *adjoins on the VP node into* γ_2*. In spite of the fact that we use sets of trees* $\{\alpha, \beta\}$ *and* $\{\gamma_1, \gamma_2\}$*, like it is in a set-local MC-TAG (Vijay-Shanker, David J. Weir, and A. K. Joshi, 1987), our approach cannot be directly expressed in a set-local MC-TAG. In particular, the notion of a derivation step in a set-local MC-TAG only allows one to substitute/adjoin trees from one set* A *into the trees belonging to the same tree set* B*. The analysis proposed by us does more than that: The trees from the set* $\{\gamma_1, \gamma_2\}$ *substitute into the tree* α*, whereas the tree* β *adjoins into* γ_2*. To model this analysis using a set-local MC-TAG, one would need to make use of at least two derivation steps and some features. For instance, one can do it as follows: At the first step, the trees from the set* $\{\gamma_1, \gamma_2\}$ *substitute into* α*. At the second step,* β *adjoins in the resultant tree. In particular,* β *should adjoin on the VP node of the tree* γ_2*. This adjunction should be obligatory. One can model that using features on* β *and on* γ_2.

2.1.2 Encoding Clause-medial Connectives in the Abstract Vocabulary

We extend the ACG encoding of G-TAG proposed in Chapter 1 in order to encode texts with clause-medial connectives. To encode the case where an adverbial *adv* appears at a clause-medial position, we encode the analysis illustrated in Figure 2.2 with the help of ACGs. In particular, we introduce a new abstract constant $\mathrm{G}_{\mathbf{adv}}^{\mathrm{medial}}$ in the abstract vocabulary Σ_{GTAG}. In Σ_{GTAG}, the constant $\mathrm{G}_{\mathbf{adv}}^{\mathrm{medial}}$ has the similar characteristics to the constant $\mathrm{G}_{\mathbf{adv_T}}^{\mathrm{T}}$ encoding the adverbial *adv* at the clause-initial position. Indeed, like $\mathrm{G}_{\mathbf{adv_T}}^{\mathrm{T}}$, the constant $\mathrm{G}_{\mathbf{adv}}^{\mathrm{medial}}$ has two arguments representing the derivation trees of clauses. Let us denote them with t_1 and t_2. By applying $\mathrm{G}_{\mathbf{adv}}^{\mathrm{medial}}$ to t_1 and t_2, one obtains a term $\mathrm{G}_{\mathbf{adv}}^{\mathrm{medial}}\ t_1\ t_2$. The syntactic interpretation of the term $\mathrm{G}_{\mathbf{adv}}^{\mathrm{medial}}\ t_1\ t_2$ should be a term encoding a derived tree of a text where *adv* occurs at a clause-medial position in the syntactic interpretation of the term t_2.

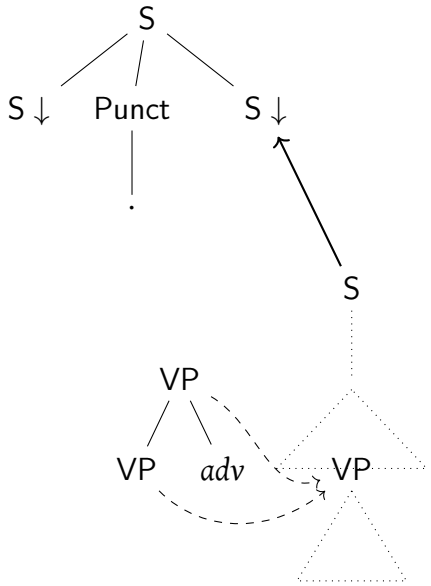
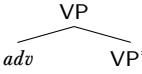


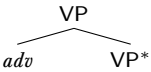
Figure 2.2: An analysis of a text containing the adverbial connective *adv* at a clause-medial position

In order to obtain the syntactic interpretation of the constant $\mathrm{G}_{\mathbf{adv}}^{\mathrm{medial}}$, we first interpret it into TAG derivation trees ($\Lambda(\Sigma_{\mathsf{TAG}}^{\mathsf{Der}})$) with the help of the lexicon $\mathscr{L}_{\mathsf{GTAG}-\mathsf{TAG}}$. However, before interpreting $\mathrm{G}_{\mathbf{adv}}^{\mathrm{medial}}$, we must type it. For that, let us consider the analysis presented in Figure 2.2. According to it, the interpretation of the second argument of $\mathrm{G}_{\mathbf{adv}}^{\mathrm{medial}}$ should be able to receive an adjunction on its VP-node, because otherwise, the auxiliary tree  would not be able to adjoin into it. By receiving a VP-adjunction, the interpretation of second argument of $\mathrm{G}_{\mathbf{adv}}^{\mathrm{medial}}$ should become a derived tree of a sentence. Something that is looking for a VP adjunction in order to become a sentence can be encoded by a term of type $\mathsf{V_A} \multimap \mathsf{S}$. Thus, the type $\mathsf{V_A} \multimap \mathsf{S}$ can serve as a type of the second argument of the constant $\mathrm{G}_{\mathbf{adv}}^{\mathrm{medial}}$. The first argument of $\mathrm{G}_{\mathbf{adv}}^{\mathrm{medial}}$ could be a term of type S, because neither adjunction nor substitution is going to be performed on the first sentence. In fact, since no operation is performed on the the first argument of the constant $\mathrm{G}_{\mathbf{adv}}^{\mathrm{medial}}$, we can model it as a text (a term of type T). Hence, we can type the constant $\mathrm{G}_{\mathbf{adv}}^{\mathrm{medial}}$ with the type $\mathsf{T} \multimap (\mathsf{V_A} \multimap \mathsf{S}) \multimap \mathsf{T}$.

Notice that the type $\mathsf{T} \multimap (\mathsf{V_A} \multimap \mathsf{S}) \multimap \mathsf{T}$ of the $\mathrm{G}_{\mathbf{adv}}^{\mathrm{medial}}$ constant is third-order (due to $(\mathsf{V_A} \multimap \mathsf{S})$). This makes the abstract signature Σ_{GTAG} third-order as well. In this case, Kanazawa's (2007) results do not apply. Consequently, one cannot guarantee that the parsing and generation tasks would remain polynomial as it is in the case of second-order ACGs. To maintain the second-order abstract vocabulary, we propose to type the constant $\mathrm{G}_{\mathbf{adv}}^{\mathrm{medial}}$ with a second-order type, namely, with $\mathsf{T} \multimap \mathsf{S} \multimap \mathsf{T}$. However, by typing the constant $\mathrm{G}_{\mathbf{adv}}^{\mathrm{medial}}$ with the type $\mathsf{T} \multimap \mathsf{S} \multimap \mathsf{T}$, one needs to express that a term over Σ_{GTAG} of type S (the second argument of an the constant $\mathrm{G}_{\mathbf{adv}}^{\mathrm{medial}}$) can receive an adjunction, i.e., an argument of type $\mathsf{V_A}$. To be more precise, the interpretation of a term over Σ_{GTAG} of type S should be a term over $\Sigma_{\mathsf{TAG}}^{\mathsf{Der}}$ of type $\mathsf{V_A} \multimap \mathsf{S}$. To achieve

that, we redefine the lexicon $\mathscr{L}_{\mathsf{GTAG}-\mathsf{TAG}} : \Sigma_{\mathsf{GTAG}} \longrightarrow \Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$. Now, $\mathscr{L}_{\mathsf{GTAG}-\mathsf{TAG}}$ interprets S from Σ_{GTAG} as $\mathsf{V_A} \multimap \mathsf{S}$ into $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$. Like before, $\mathscr{L}_{\mathsf{GTAG}-\mathsf{TAG}}$ interprets the type T from Σ_{GTAG} as S into $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$. We interpret the new constant $\mathrm{G}^{\text{medial}}_{\mathbf{adv}} : \mathsf{T} \multimap \mathsf{S} \multimap \mathsf{T}$ to $\Lambda(\Sigma^{\mathsf{Der}}_{\mathsf{TAG}})$ as it is shown in Equation (2.31).

$$\mathscr{L}_{\mathsf{GTAG}-\mathsf{TAG}}(\mathrm{G}^{\text{medial}}_{\mathbf{adv}}) = \lambda^{\mathrm{o}}\, s_1.\, \lambda^{\mathrm{o}}\, s_2.\, \mathrm{C}_{\mathbf{Concat}}\ s_1\ (s_2\ \mathrm{C}^{\mathrm{v}}_{\mathbf{adv}}) : \mathsf{S} \multimap (\mathsf{V_A} \multimap \mathsf{S}) \multimap \mathsf{S} \tag{2.31}$$

In Equation (2.31), the constant $\mathrm{C}^{\mathrm{v}}_{\mathbf{adv}}$ of type $\mathsf{V_A}$ stands for the auxiliary tree 

. The constant $\mathrm{C}_{\mathbf{Concat}}$ of type $\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$ represents an initial tree with two substitution sites anchored with the full stop, shown in Figure 2.3. Thus, the term $\lambda^{\mathrm{o}}\, s_1.\, \lambda^{\mathrm{o}}\, s_2.\, \mathrm{C}_{\mathbf{Concat}}\ s_1\ (s_2\ \mathrm{C}^{\mathrm{v}}_{\mathbf{adv}})$ encodes the fact that is pictorially shown in Figure 2.2 on the facing page: s_2 receives a VP-adjunction ($\mathrm{C}^{\mathrm{v}}_{\mathbf{adv}}$) inserting *adv* at a clause-medial position. The terms s_1 and $(s_2\ \mathrm{C}^{\mathrm{v}}_{\mathbf{adv}})$, which represent the derivation trees of the first and second pieces of a text respectively, are the arguments of $\mathrm{C}_{\mathbf{Concat}}$. The constant $\mathrm{C}_{\mathbf{Concat}}$ receives the first tree (s_1) and the second one $(s_2\ \mathrm{C}^{\mathrm{v}}_{\mathbf{adv}})$ and puts the full stop between them. In this way, one obtains a text where an adverbial *adv* appears at a clause-medial position in the second sentence $(s_2\ \mathrm{C}^{\mathrm{v}}_{\mathbf{adv}})$.

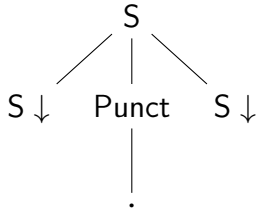


Figure 2.3: The tree modeled by $\mathrm{C}_{\mathbf{Concat}}$

2.2 Interpretations of G-derivation Trees as TAG Derivation Trees

To interpret g-derivation trees as derived trees, we first interpret g-derivation trees as TAG derivation ones. Since TAG derivation trees are already interpreted as derived trees, by composition of the two interpretations, one obtains the interpretation of g-derivation trees as derived trees.

Since we extend the ACG encoding of G-TAG, the translation of the type S from Σ_{GTAG} to $\mathsf{V_A} \multimap \mathsf{S}$ in $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ requires to make changes in the translations of the abstract constants whose types involve S. There are two cases:

1. G is a first order predicate (a clause missing a subject, an infinitive clause, an initial tree anchored with a verb, predicative adjective, etc.);

2. G is a constant encoding an initial tree anchored with a conjunction (e.g. $\mathrm{G}_{\textbf{avant}}^{\text{canonical}} : \mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$ or $\mathrm{G}_{\textbf{avant}}^{\text{red.}} : \mathsf{np} \multimap \mathsf{Sws} \multimap \mathsf{Sinf} \multimap \mathsf{S}$).

Let us consider a constant $\mathrm{G}_{\textbf{récompense}} : \mathsf{S}_\mathsf{A} \multimap \mathsf{V}_\mathsf{A} \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{S}$. Since the type $\mathsf{S} \in \Sigma_{\mathsf{GTAG}}$ translates to $\mathsf{S} \multimap \mathsf{V}_\mathsf{A}$ into $\Sigma_{\mathsf{TAG}}^{\mathsf{Der}}$, if we do not make any further changes in its translation by the $\mathscr{L}_{\mathsf{GTAG}-\mathsf{TAG}}$ lexicon, then the constant $\mathrm{G}_{\textbf{récompense}}$ would translate to a term over $\Lambda(\Sigma_{\mathsf{TAG}}^{\mathsf{Der}})$ of the following type:

$$\mathscr{L}_{\mathsf{GTAG}-\mathsf{TAG}}(\mathrm{G}_{\textbf{récompense}}) : \mathsf{S}_\mathsf{A} \multimap \mathsf{V}_\mathsf{A} \multimap \mathsf{np} \multimap \mathsf{np} \multimap (\mathsf{V}_\mathsf{A} \multimap \mathsf{S}) \tag{2.32}$$

Thus, the interpretation of the constant $\mathrm{G}_{\textbf{récompense}}$ in the TAG derivation trees requires an additional VP-adjunction in order to become a term of type S.

Let us denote with $t_{\textbf{récompense}}^{\text{GTAG}}$ the following abstract term over Σ_{GTAG}:

$$t_{\textbf{récompense}}^{\text{GTAG}} = \mathrm{G}_{\textbf{récompense}}\ \mathrm{I}_{\mathsf{S}_\mathsf{A}}\ \mathrm{I}_{\mathsf{V}_\mathsf{A}}\ \mathrm{G}_{\textbf{jean}}\ \mathrm{G}_{\textbf{marie}}\ : \mathsf{S} \tag{2.33}$$

The interpretation of the term $t_{\textbf{récompense}}^{\text{GTAG}}$ into $\Lambda(\Sigma_{\mathsf{TAG}}^{\mathsf{Der}})$ should be of type $\mathsf{V}_\mathsf{A} \multimap \mathsf{S}$ as the type S form Σ_{GTAG} translates to the $\mathsf{V}_\mathsf{A} \multimap \mathsf{S}$ type in $\Sigma_{\mathsf{TAG}}^{\mathsf{Der}}$. Let us interpret the term $t_{\textbf{récompense}}^{\text{GTAG}}$ into $\Lambda(\Sigma_{\mathsf{TAG}}^{\mathsf{Der}})$. We obtain the following:

$$\begin{aligned}\mathscr{L}_{\mathsf{GTAG}-\mathsf{TAG}}(t_{\textbf{récompense}}^{\text{GTAG}}) = \\ &\mathscr{L}_{\mathsf{GTAG}-\mathsf{TAG}}(\mathrm{G}_{\textbf{récompense}})\ \mathscr{L}_{\mathsf{GTAG}-\mathsf{TAG}}(\mathrm{I}_{\mathsf{S}_\mathsf{A}})\ \mathscr{L}_{\mathsf{GTAG}-\mathsf{TAG}}(\mathrm{I}_{\mathsf{V}_\mathsf{A}}) \\ &\mathscr{L}_{\mathsf{GTAG}-\mathsf{TAG}}(\mathrm{G}_{\textbf{jean}})\ \mathscr{L}_{\mathsf{GTAG}-\mathsf{TAG}}(\mathrm{G}_{\textbf{marie}}) : \mathsf{V}_\mathsf{A} \multimap \mathsf{S}\end{aligned} \tag{2.34}$$

By computing the value of the $\mathscr{L}_{\mathsf{GTAG}-\mathsf{TAG}}(t_{\textbf{récompense}}^{\text{GTAG}})$ term in the right-hand side of Equation (2.34), we must obtain the following:

$$\mathscr{L}_{\mathsf{GTAG}-\mathsf{TAG}}(t_{\textbf{récompense}}^{\text{GTAG}}) \twoheadrightarrow_\beta \lambda^{\mathrm{o}}\, mod.\, t[mod]\ : \mathsf{V}_\mathsf{A} \multimap \mathsf{S} \tag{2.35}$$

Where t is a term; $t[mod]$ denotes the term t but expresses that in t the variable mod has a free occurrence. The variable $mod : \mathsf{V}_\mathsf{A}$ should appear in the sub-term encoding the VP-adjunction site (the clause-medial position). Indeed, by applying $\mathscr{L}_{\mathsf{GTAG}-\mathsf{TAG}}(t_{\textbf{récompense}}^{\text{GTAG}})$ to the term $\mathrm{C}_{\textbf{adv}}^{\mathrm{V}} : \mathsf{V}_\mathsf{A}$, we should obtain a term modeling TAG derivation tree of a clause where the adverbial *adv* occupies a clause-medial position. To achieve that, we propose the following interpretation of the constant $\mathrm{G}_{\textbf{récompense}}$ to $\Lambda(\Sigma_{\mathsf{TAG}}^{\mathsf{Der}})$:

$$\begin{aligned}&\mathscr{L}_{\mathsf{GTAG}-\mathsf{TAG}}(\mathrm{G}_{\textbf{récompense}}) = \\ &\lambda^{\mathrm{o}}\, s_a\ v_a\ subj\ obj.\ \lambda^{\mathrm{o}}\, mod.\ \overbrace{\mathrm{C}_{\textbf{récompense}}}^{\mathsf{S}_\mathsf{A} \multimap \mathsf{V}_\mathsf{A} \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{S}}\ \overbrace{s_a}^{\mathsf{S}_\mathsf{A}}\ \overbrace{(v_a\ \underbrace{mod}_{\mathsf{V}_\mathsf{A}})}^{\mathsf{V}_\mathsf{A}}\ \overbrace{subj}^{\mathsf{np}}\ \overbrace{obj}^{\mathsf{np}} : \\ &\mathsf{S}_\mathsf{A} \multimap (\mathsf{V}_\mathsf{A} \multimap \mathsf{V}_\mathsf{A}) \multimap \mathsf{np} \multimap \mathsf{np} \multimap (\mathsf{V}_\mathsf{A} \multimap \mathsf{S})\end{aligned} \tag{2.36}$$

As Equation (2.36) indicates, we use the constant $\mathrm{C}_{\textbf{récompense}} \in \Sigma_{\mathsf{TAG}}^{\mathsf{Der}}$ of type $\mathsf{S}_\mathsf{A} \multimap \mathsf{V}_\mathsf{A} \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{S}$, which is a standard type of a constant encoding a transitive verb

in $\Sigma^{\text{Der}}_{\text{TAG}}$. Thus, we do not make any changes in the constants of $\Sigma^{\text{Der}}_{\text{TAG}}$. In the translation shown in Equation (2.36), the sub-term $(v_a\ mod)$ of type $\mathsf{V_A}$ provides a needed slot for a VP-adjunction that one can use in order to place an adverbial at a clause-medial position. The term $(v_a\ mod)$ is of type $\mathsf{V_A}$ because we use the constant $\mathrm{C}_{\textbf{récompense}}$ of $\Sigma^{\text{Der}}_{\text{TAG}}$ of type $\mathsf{S_A} \multimap \mathsf{V_A} \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{S}$. The variable mod is of type $\mathsf{V_A}$. Hence, the variable v_a is of type $\mathsf{V_A} \multimap \mathsf{V_A}$. This implies that the type $\mathsf{V_A}$ of Σ_{GTAG} translates to $\mathsf{V_A} \multimap \mathsf{V_A}$ into $\Sigma^{\text{Der}}_{\text{TAG}}$.

Types in Σ_{GTAG}	Their translations by $\mathscr{L}_{\text{GTAG}-\text{TAG}}$
S	$\mathsf{V_A} \multimap \mathsf{S}$
$\mathsf{V_A}$	$\mathsf{V_A} \multimap \mathsf{V_A}$

Table 2.1: Interpretations of the types S and $\mathsf{V_A}$ into $\Sigma^{\text{Der}}_{\text{TAG}}$

Hence, the lexicon $\mathscr{L}_{\text{GTAG}-\text{TAG}}$ interprets a constant of type $\mathsf{S_A} \multimap \mathsf{V_A} \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{S}$ from Σ_{GTAG} as a term of type $\mathsf{S_A} \multimap (\mathsf{V_A} \multimap \mathsf{V_A}) \multimap \mathsf{np} \multimap \mathsf{np} \multimap (\mathsf{V_A} \multimap \mathsf{S})$ into $\Lambda(\Sigma^{\text{Der}}_{\text{TAG}})$.

Since now $\mathscr{L}_{\text{GTAG}-\text{TAG}}$ interprets $\mathsf{V_A}$ from Σ_{GTAG} as $\mathsf{V_A} \multimap \mathsf{V_A}$ into $\Sigma^{\text{Der}}_{\text{TAG}}$, one must redefine $\mathscr{L}_{\text{GTAG}-\text{TAG}}$ on the constants of Σ_{GTAG} whose types involve $\mathsf{V_A}$.

Let us recall that in the ACG encoding of G-TAG, we interpreted the type Sws (a clause missing a subject) as $\mathsf{np} \multimap \mathsf{S}$ into $\Sigma^{\text{Der}}_{\text{TAG}}$. Now, a question is what should be the interpretation of Sws in $\Sigma^{\text{Der}}_{\text{TAG}}$, should it be $\mathsf{np} \multimap \mathsf{S}$ or $\mathsf{np} \multimap \mathsf{V_A} \multimap \mathsf{S}$? Interpreting Sws as $\mathsf{np} \multimap \mathsf{V_A} \multimap \mathsf{S}$ would allow us to analyze sentences with reduced conjunction whose matrix clause contains an adverbial at the clause-medial position.[69] Consequently, we have to make changes into the interpretations of the constants encoding reduced conjunctions into $\Lambda(\Sigma^{\text{Der}}_{\text{TAG}})$ because (a) we changed the interpretation of the type S; (b) we change the interpretation of the type Sws. Thus, we have to redefine the lexicon $\mathscr{L}_{\text{GTAG}-\text{TAG}}$ on various constants simultaneously.

While it is possible to redefine interpretation of all the above mentioned types and constants at the same time, it is also possible to do that sequentially. In the next section, we develop an approach that offers a modular view on the relationship between G-TAG derivation trees and TAG derivation ones.

2.3 A Modular Interpretation of Σ_{GTAG} to TAG Derivation Trees

Instead of simultaneously interpreting the types S and Sws from Σ_{GTAG} as $\mathsf{V_A} \multimap \mathsf{S}$ and $\mathsf{np} \multimap \mathsf{V_A} \multimap \mathsf{S}$ in $\Sigma^{\text{Der}}_{\text{TAG}}$ respectively, we propose to do it in a sequential order. For that, we introduce a new object vocabulary $\Sigma_{\text{g-der}}$. Figure 2.4 on the following page shows the new ACG architecture. We define the following ACGs:

1. $\mathcal{G}_1 = \langle \Sigma_{\text{GTAG}}, \Sigma_{\text{g-der}}, \mathscr{L}_{\mathsf{G}}, \mathsf{T} \rangle$

[69] For more details, see Section 2.3.2.2.2 on page 243.

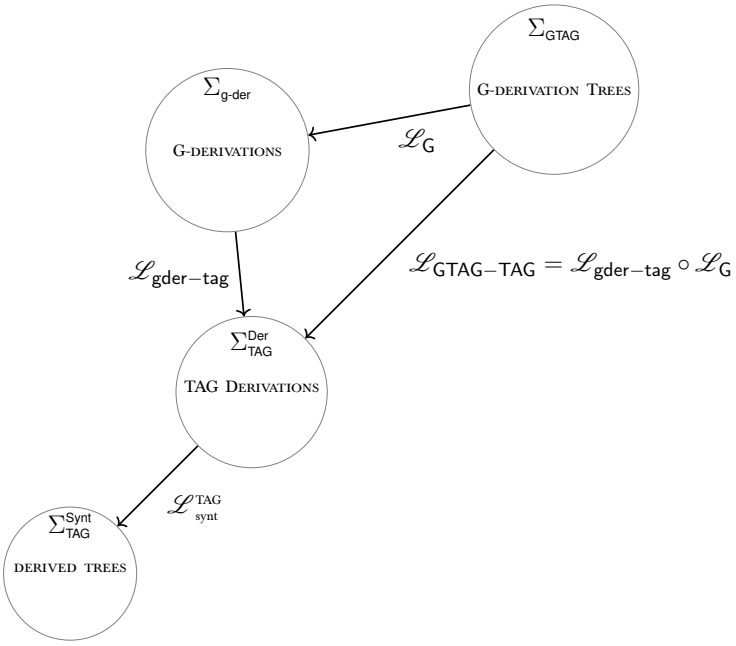


Figure 2.4: An ACG architecture for G-TAG

2. $\mathcal{G}_2 = \langle \Sigma_{\text{g-der}}, \Sigma^{\text{Der}}_{\text{TAG}}, \mathscr{L}_{\text{gder}-\text{tag}}, \mathsf{T} \rangle$

We split the tasks of encoding the phenomena of reduced conjunctions and clause-medial connectives into two. We encode reduced conjunctions with the help of the ACG $\mathcal{G}_1$, whereas we encode clause-medial connectives with the help of the ACG $\mathcal{G}_2$. The composed ACG $\mathcal{G}_2 \circ \mathcal{G}_1$ enables us to encode both clause-medial connectives and reduced conjunctions. In this ACG architecture, we have the following lexicons:

1. The lexicon $\mathscr{L}_{\mathsf{G}} : \Sigma_{\text{GTAG}} \longrightarrow \Sigma_{\text{g-der}}$ interprets Sws from Σ_{GTAG} as $\mathsf{np} \multimap \mathsf{S}$ in $\Sigma_{\text{g-der}}$;
2. The lexicon $\mathscr{L}_{\text{gder}-\text{tag}} : \Sigma_{\text{g-der}} \longrightarrow \Sigma^{\text{Der}}_{\text{TAG}}$ interprets S from $\Sigma_{\text{g-der}}$ as $\mathsf{V_A} \multimap \mathsf{S}$ in $\Sigma^{\text{Der}}_{\text{TAG}}$.

Thus, one obtains the following:

$$\mathscr{L}_{\text{gder}-\text{tag}}(\mathscr{L}_{\mathsf{G}}(\mathsf{Sws})) = \mathscr{L}_{\text{gder}-\text{tag}}(\mathsf{np} \multimap \mathsf{S}) = \mathscr{L}_{\text{gder}-\text{tag}}(\mathsf{np}) \multimap \mathscr{L}_{\text{gder}-\text{tag}}(\mathsf{S}) = \\ = \mathscr{L}_{\text{gder}-\text{tag}}(\mathsf{np}) \multimap \mathsf{V_A} \multimap \mathsf{S} = \mathsf{np} \multimap \mathsf{V_A} \multimap \mathsf{S} \quad (2.37)$$

As Equation (2.37) shows, the lexicon $\mathscr{L}_{\text{gder}-\text{tag}} \circ \mathscr{L}_{\mathsf{G}}$ of the composed ACG $\mathcal{G}_2 \circ \mathcal{G}_1$ interprets Sws from Σ_{GTAG} as $\mathsf{np} \multimap \mathsf{V_A} \multimap \mathsf{S}$ in $\Sigma^{\text{Der}}_{\text{TAG}}$.

The new object vocabulary $\Sigma_{\text{g-der}}$ is similar to the abstract vocabulary Σ_{GTAG}. In particular, in $\Sigma_{\text{g-der}}$, one has all the types from Σ_{GTAG} except Sws and Sinf. Besides types, in $\Sigma_{\text{g-der}}$, we adopt constants from Σ_{GTAG} so that if a constant G_α is in Σ_{GTAG}, then the constant g_α is in $\Sigma_{\text{g-der}}$, except from the constants such as $\mathrm{G}_{\mathrm{v}}^{\text{sws}}$. By convention, we denote constants of $\Sigma_{\text{g-der}}$ by g_α.

Thus, it remains to define the lexicons $\mathscr{L}_{\mathsf{G}} : \Sigma_{\text{GTAG}} \longrightarrow \Sigma_{\text{g-der}}$ and $\mathscr{L}_{\text{gder}-\text{tag}} : \Sigma_{\text{g-der}} \longrightarrow \Sigma^{\text{Der}}_{\text{TAG}}$.

2.3.1 The Lexicon from Σ_{GTAG} to $\Sigma_{\text{g-der}}$

2.3.1.1 Interpretations of Types

The lexicon $\mathscr{L}_{\mathsf{G}} : \Sigma_{\text{GTAG}} \longrightarrow \Sigma_{\text{g-der}}$ interprets the type Sws from Σ_{GTAG} as $\mathsf{np} \multimap \mathsf{S}$ in $\Sigma_{\text{g-der}}$. This is similar to what we did in the ACG encoding of G-TAG, where we interpreted Sws from Σ_{GTAG} as $\mathsf{np} \multimap \mathsf{S}$ in $\Sigma^{\text{Der}}_{\text{TAG}}$. In addition, we interpret Sinf from Σ_{GTAG} as S in $\Sigma_{\text{g-der}}$.

The rest of the types are interpreted as themselves: For any X atomic type in Σ_{GTAG} that is different from Sws and Sinf, the lexicon $\mathscr{L}_{\mathsf{G}}$ translates the type X from Σ_{GTAG} to X in $\Sigma_{\text{g-der}}$. For instance, the type T in Σ_{GTAG} translates to T in $\Sigma_{\text{g-der}}$.

2.3.1.2 Interpretations of Constants

Any constant $\mathrm{G}_{\mathbf{a}}$ from Σ_{GTAG} translates to $\mathrm{g}_{\mathbf{a}}$ in $\Sigma_{\text{g-der}}$ provided that the type of the constant $\mathrm{G}_{\mathbf{a}}$ contains neither Sws nor Sinf. For example, the lexicon $\mathscr{L}_{\mathsf{G}}$ translates constants $\mathrm{G}_{\mathbf{adv}_{\mathrm{T}}}^{\mathrm{T}} : \mathsf{T} \multimap \mathsf{T} \multimap \mathsf{T}$ and $\mathrm{G}_{\mathbf{adv}}^{\text{medial}} : \mathsf{T} \multimap \mathsf{S} \multimap \mathsf{T}$ as the constants $\mathrm{g}_{\mathbf{adv}_{\mathrm{T}}}^{\mathrm{T}} : \mathsf{T} \multimap \mathsf{T} \multimap \mathsf{T}$ and $\mathrm{g}_{\mathbf{adv}}^{\text{medial}} : \mathsf{T} \multimap \mathsf{S} \multimap \mathsf{T}$, respectively. It remains to interpret constants encoding subordinate conjunctions and the first order predicates (the ones that enable us to encode derivation trees of clauses).

2.3.1.2.1 Conjunctions

Canonical Conjunctions

$\mathscr{L}_{\mathsf{G}}$ interprets the constant $\mathrm{G}_{\mathbf{conj}}^{\text{canonical}}$ to $\mathrm{g}_{\mathbf{conj}}^{\text{canonical}}$, as they both represent the same elementary tree anchored by *conj*.

Reduced Conjunctions

Since the lexicon $\mathscr{L}_{\mathsf{G}}$ interprets the type $\mathsf{Sws} \in \Sigma_{\text{GTAG}}$ as $\mathsf{np} \multimap \mathsf{S}$ in $\Sigma_{\text{g-der}}$, one encodes the *subject-sharing* with the help of $\mathscr{L}_{\mathsf{G}}$. Namely, we interpret the constants modeling reduced conjunctions by $\mathscr{L}_{\mathsf{G}}$ so that the interpretations encode that the subject syntactically belongs to the clause missing a subject. That is, we interpret the constant $\mathrm{G}_{\mathbf{conj}}^{\text{red.}}$ of type $\mathsf{np} \multimap \mathsf{Sws} \multimap \mathsf{Sinf} \multimap \mathsf{S}$ encoding the reduced g-derivation tree of the conjunction *conj* as follows:

$$\mathscr{L}_{\mathsf{G}}(\mathrm{G}_{\mathbf{conj}}^{\text{red.}}) = \lambda^{\mathrm{o}}\, np\; s_1\; s_2.\, \mathrm{g}_{\mathbf{conj}}^{\text{red.}}\; (s_1\; np)\; s_2 \tag{2.38}$$

Where in Equation (2.38) the constant $\mathrm{g}_{\mathbf{conj}}^{\text{red.}}$ is of type $\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$. As this interpretation indicates, we combine the subject and the clause missing a subject $(s_1\; np)$. The constant $\mathrm{g}_{\mathbf{conj}}^{\text{red.}}$ resembles the underspecified g-derivation tree of $\underline{\text{conj}}$: $\mathrm{g}_{\mathbf{conj}}^{\text{red.}}$ has two variable nodes which can be instantiated with g-derivation trees of $\overline{\text{clauses}}$ (terms of type S).[70]

[70] This interpretation is similar to the interpretations of constants encoding reduced conjunctions from Σ_{GTAG} into $\Lambda(\Sigma^{\text{Der}}_{\text{TAG}})$.

2.3.1.2.2 First Order Predicates

To illustrate the way the lexicon $\mathscr{L}_{\mathsf{G}}$ interprets the constants enabling us to produce terms of type Sinf and Sws, we provide Table 2.2. In general, given the constants $\mathrm{G}_{\mathbf{v}}^{\text{sws}} : \vec{\alpha}_n$ and $\mathrm{G}_{\mathbf{vinf}}^{\text{inf}} : \vec{\beta}_n$ enabling us to produce the terms of types[71] Sws and Sinf respectively, we interpret them into $\Lambda(\Sigma_{\text{g-der}})$ as follows:

$$\mathscr{L}_{\mathsf{G}}(\mathrm{G}_{\mathbf{v}}^{\text{sws}}) = \lambda^{\text{o}} x_1 \ldots x_n . \lambda^{\text{o}} subj . \, \mathrm{g}_{\mathbf{v}} \; x_1 \ldots subj \ldots x_n \tag{2.39}$$

$$\mathscr{L}_{\mathsf{G}}(\mathrm{G}_{\mathbf{vinf}}^{\text{inf}}) = \mathrm{g}_{\mathbf{vinf}}^{\text{inf}} : \vec{\delta}_n \tag{2.40}$$

Where $\mathrm{g}_{\mathbf{v}}$ denotes an initial tree anchored by v. The interpretation shown in Equation (2.39) encodes that a clause missing the subject differs from a finite clause only by the fact that it receives the subject (the variable $subj$) at the last place.[72]

Constants in Σ_{GTAG}	Their translations by the lexicon $\mathscr{L}_{\mathsf{G}}$ into $\Lambda(\Sigma_{\text{g-der}})$
$\mathrm{G}_{\textbf{récompense}}^{\text{sws}} : \mathsf{S_A} \multimap \mathsf{V_A} \multimap \mathsf{np} \multimap \mathsf{Sws}$	$\lambda^{\text{o}} s^a \; v^a \; obj . \lambda^{\text{o}} subj . \mathrm{g}_{\textbf{récompense}} \; s^a \; v^a \; subj \; obj :$ $\mathsf{S_A} \multimap \mathsf{V_A} \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{S}$
$\mathrm{G}_{\textbf{récompenser}}^{\text{inf}} : \mathsf{S_A} \multimap \mathsf{V_A} \multimap \mathsf{np} \multimap \mathsf{Sinf}$	$\mathrm{g}_{\textbf{récompenser}}^{\text{inf}} : \mathsf{S_A} \multimap \mathsf{V_A} \multimap \mathsf{np} \multimap \mathsf{S}$

Table 2.2: Interpretations of constants enabling to produce terms of types Sws and Sinf

2.3.2 The Lexicon from $\Sigma_{\text{g-der}}$ to $\Sigma_{\text{TAG}}^{\text{Der}}$

As we already mentioned, to model the clause-medial adverbials, the lexicon $\mathscr{L}_{\text{gder}-\text{tag}} : \Sigma_{\text{g-der}} \longrightarrow \Sigma_{\text{TAG}}^{\text{Der}}$ interprets S as $\mathsf{V_A} \multimap \mathsf{S}$. As we saw in Section 2.2, in addition to interpreting S as $\mathsf{V_A} \multimap \mathsf{S}$, the modeling of clause-medial adverbials requires interpreting of $\mathsf{V_A}$ from $\Sigma_{\text{g-der}}$ as $\mathsf{V_A} \multimap \mathsf{V_A}$ into $\Sigma_{\text{TAG}}^{\text{Der}}$. In concordance with these interpretations we should interpret the constants whose types involve $\mathsf{V_A}$ and/or S from $\Sigma_{\text{g-der}}$ into $\Lambda(\Sigma_{\text{TAG}}^{\text{Der}})$. The constants whose types involve $\mathsf{V_A}$ and/or S are as follows:

1. Constants modeling clause-medial adverbials (i.e., $\mathrm{g}_{\textbf{adv}}^{\text{medial}} : \mathsf{T} \multimap \mathsf{S} \multimap \mathsf{T}$);
2. constants modeling conjunctions (i.e., $\mathrm{g}_{\textbf{conj}}^{\text{canonical}} : \mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$ and $\mathrm{g}_{\textbf{conj}}^{\text{red.}} : \mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$);
3. first order predicates adopted from $\Sigma_{\text{TAG}}^{\text{Der}}$ (e.g. $\mathrm{g}_{\textbf{récompense}}$).

To interpret the rest of the constants of $\Sigma_{\text{g-der}}$, we refer to the interpretations of g-derivation trees as TAG derivation trees in the ACG encoding of G-TAG. In particular, if g_{α} is a constant whose type is not built using $\mathsf{V_A}$ and/or S, then we consider the

[71] In Section 1.3.2.2 on page 206, we defined the types $\vec{\alpha}_n$, $\vec{\beta}_n$, and $\vec{\gamma}_n$.

[72] The interpretations provided in Equation (2.39) and Equation (2.40) are analogous to the interpretations of the constants $\mathrm{G}_{\mathbf{v}}^{\text{sws}}$ and $\mathrm{G}_{\mathbf{vinf}}^{\text{inf}}$ into TAG derivation trees proposed in the ACG encoding of G-TAG (see Section 1.4.2.3 on page 212 for a detailed discussion.)

constant G_α of Σ_{GTAG} in the ACG encoding of G-TAG. Since G_α and g_α model the same constraint, we interpret g_α into $\Lambda(\Sigma^{\mathsf{Der}}_{\mathsf{TAG}})$ to the same term to which we interpreted G_α into $\Lambda(\Sigma^{\mathsf{Der}}_{\mathsf{TAG}})$ in the ACG encoding of G-TAG.

2.3.2.1 Interpretations of Types

In $\Sigma_{\mathsf{g\text{-}der}}$, besides the types S and $\mathsf{V_A}$, we have types that are adopted from $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$. In addition, we have the type T, introduced for modeling g-derivation trees of texts. We interpret these types into $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ as follows:

- For any type X in $\Sigma_{\mathsf{g\text{-}der}}$ that differs from S, $\mathsf{V_A}$, and T, we have the type X in $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ modeling the same constraint. Thus, the lexicon $\mathscr{L}_{\mathsf{gder-tag}}$ interprets X from $\Sigma_{\mathsf{g\text{-}der}}$ as X in $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$.
- $\mathscr{L}_{\mathsf{gder-tag}}(\mathsf{T}) = \mathsf{S}$.

2.3.2.2 Interpretations of Constants

2.3.2.2.1 Clause-medial Adverbials

In $\Sigma_{\mathsf{g\text{-}der}}$, one has a constant $\mathrm{g}^{\mathrm{medial}}_{\mathbf{adv}} : \mathsf{T} \multimap \mathsf{S} \multimap \mathsf{T}$ modeling the clause-medial adverbial *adv*. Section 2.1 discusses the way we interpret the constant $\mathrm{G}^{\mathrm{medial}}_{\mathbf{adv}} : \mathsf{T} \multimap \mathsf{S} \multimap \mathsf{T}$ of Σ_{GTAG} modeling the clause-medial adverbial adv into $\Lambda(\Sigma^{\mathsf{Der}}_{\mathsf{TAG}})$. We employ the constants $\mathrm{G}^{\mathrm{medial}}_{\mathbf{adv}}$ and $\mathrm{g}^{\mathrm{medial}}_{\mathbf{adv}}$ to encode the same analysis, shown in Figure 2.2 on page 236. Thus, we can refer to the interpretation of $\mathrm{G}^{\mathrm{medial}}_{\mathbf{adv}}$ into $\Lambda(\Sigma^{\mathsf{Der}}_{\mathsf{TAG}})$ for interpreting $\mathrm{g}^{\mathrm{medial}}_{\mathbf{adv}}$ into $\Lambda(\Sigma^{\mathsf{Der}}_{\mathsf{TAG}})$, which is as follows:

$$\mathscr{L}_{\mathsf{gder-tag}}(\mathrm{g}^{\mathrm{medial}}_{\mathbf{adv}}) = \lambda^{\mathrm{o}}\, s_1.\ \lambda^{\mathrm{o}}\, s_2.\ \mathrm{C}_{\mathbf{Concat}}\ s_1\ (s_2\ \mathrm{C}^{\mathrm{v}}_{\mathbf{adv}}) : \mathsf{S} \multimap (\mathsf{V_A} \multimap \mathsf{S}) \multimap \mathsf{S} \tag{2.41}$$

2.3.2.2.2 Subordinate Conjunctions

In order to encode constants modeling subordinate conjunctions, notice that an adverbial connective may appear at the clause-medial position in a sentence built by a subordinate conjunction. In a case of a sentence with either a canonical or a reduced conjunction, an adverbial connective appears at the clause-medial position only in the matrix clause of a sentence. As Examples in (43) and (44) indicate, the adverbial *ensuite* occupies a clause-medial position in a matrix clause of a sentence with a conjunction.[73] Figure 2.5 illustrates the analysis of the discourses in Examples (43) and (44): The auxiliary tree anchored with the adverbial adjoins on the VP node into the derived tree of the matrix clause of the sentence with the conjunction.

[73] In a case where an adverbial connective is *inter-sentential*, it cannot appear within the subordinate clause of a sentence, but only in the matrix clause. However, if an adverbial is *intra-sentential*, it may appear within the subordinated clause, as it is in the following sentence:

(43) *Jean a* *préparé* *le petite déjeuner.* *Jean a* ensuite
John have$_{\text{3PS. SG. PRES.}}$ prepare$_{\text{PAST PART.}}$ the breakfast. John then have$_{\text{3PS. SG. PRES.}}$
passé l'aspirateur avant *que Marie fasse* *une sieste.*
vacuum$_{\text{PAST PART.}}$ before that Mary make$_{\text{3PS. SG. PRES. SUBJUNCTIVE}}$ a nap.

John prepared the breakfast. John then vacuumed before Mary took a nap.

(44) *Jean a* *préparé* *le petite déjeuner.* *Jean a* ensuite
John have$_{\text{3PS. SG. PRES.}}$ prepare$_{\text{PAST PART.}}$ the breakfast. Jean then have$_{\text{3PS. SG. PRES.}}$
passé l'aspirateur avant *de faire* *une sieste.*
vacuum$_{\text{PAST PART.}}$ before of make$_{\text{infinitive pres.}}$ a nap.

John prepared the breakfast. Jean then vacuumed before taking a nap.

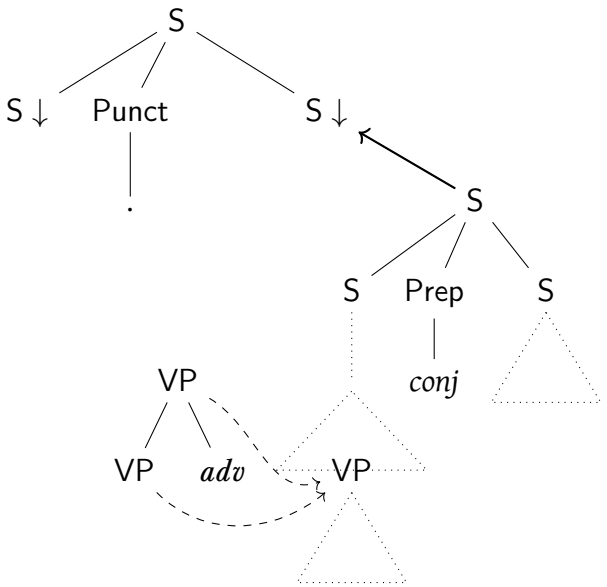


Figure 2.5: An analysis of a case with an adverbial at a clause-medial position of a sentence with a conjunction

In $\Sigma_{\text{g-der}}$, we have two constants $g^{\text{canonical}}_{\textbf{conj}}$ and $g^{\text{red.}}_{\textbf{conj}}$ of type $\mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$ modeling the canonical and reduced g-derivation trees of a conjunction *conj*, respectively. Since we interpret S as $\mathsf{V}_{\mathsf{A}} \multimap \mathsf{S}$ into $\Sigma^{\text{Der}}_{\text{TAG}}$, the arguments of these constants encoding the matrix clause and the subordinate clause can receive a VP-adjunction. However, as we

(42) *Jean a* *passé* *l'aspirateur* *dans la matinée* *pour* ensuite *être* *libre pendent toute*
John have$_{\text{3PS. SG.}}$ pass$_{\text{PAST PART.}}$ vacuumer$_{\text{DEF.}}$ in the morning for *then* to be$_{\text{INF.}}$ free during all
le *journée.*
the day.

Jean vacuumed in the morning in order to *then* be free during the whole day.

have discussed above, only the matrix clause can host an adverbial. Thus, we should disable adjunction on the subordinate clause. We do it with the help of the empty VP-adjunction ($\mathrm{I}_{\mathsf{V_A}}$). We propose the interpretations of the constants $\mathrm{g}_{\mathbf{conj}}^{\text{canonical}}$ and $\mathrm{g}_{\mathbf{conj}}^{\text{red.}}$ shown in Table 2.3.

Constants in $\Sigma_{\text{g-der}}$	Their interpretations into $\Lambda(\Sigma_{\mathsf{TAG}}^{\mathsf{Der}})$
$\mathrm{g}_{\mathbf{conj}}^{\text{canonical}} : \mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$	$\lambda^{\mathrm{o}}\, s_1.\, \lambda^{\mathrm{o}}\, s_2.\, \lambda^{\mathrm{o}} mod.\, \mathrm{C}_{\mathbf{conj}}^{\text{canonical}}\ (s_1\ mod)\ (s_2\ \mathrm{I}_{\mathsf{V_A}})$
$\mathrm{g}_{\mathbf{conj}}^{\text{red.}} : \mathsf{S} \multimap \mathsf{S} \multimap \mathsf{S}$	$\lambda^{\mathrm{o}}\, s_1.\, \lambda^{\mathrm{o}}\, s_2.\, \lambda^{\mathrm{o}} mod.\, \mathrm{C}_{\mathbf{conj}}^{\text{red.}}\ (s_1\ mod)\ (s_2\ \mathrm{I}_{\mathsf{V_A}})$

Table 2.3: Interpretations of the constants encoding conjunctions by the lexicon $\mathscr{L}_{\mathsf{gder-tag}}$

2.3.2.2.3 First Order Predicates

It remains to interpret constants of $\Sigma_{\text{g-der}}$ that originate from $\Sigma_{\mathsf{TAG}}^{\mathsf{Der}}$. In Section 2.2, we discussed the way we interpret constants of Σ_{GTAG} standing for initial trees anchoring verbs, predictive adjectives, etc. (the ones that give rise to clauses) into TAG derivation trees. Namely, we provided an example interpreting $\mathrm{G}_{\textbf{récompense}}$ into TAG derivation trees (see Equation (2.2) on page 237). We make use of the same principles in order to interpret first order predicates of $\Sigma_{\text{g-der}}$ into TAG derivation trees. Table 2.4 illustrates the way one interprets the constants encoding initial trees anchored by finite verb forms, infinite ones, and adverbs.

Constants in $\Sigma_{\text{g-der}}$	Their interpretations into $\Lambda(\Sigma_{\mathsf{TAG}}^{\mathsf{Der}})$
$\mathrm{g}_{\textbf{récompense}}$ $: \mathsf{S_A} \multimap \mathsf{V_A} \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{S}$	$\lambda^{\mathrm{o}}\, s^a\ v^a\ subj\ obj.\, \lambda^{\mathrm{o}}\ mod.\, \mathrm{C}_{\textbf{récompense}}\ s^a\ (v^a\ mod)\ \ subj\ obj$ $: \mathsf{S_A} \multimap (\mathsf{V_A} \multimap \mathsf{V_A}) \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{V_A} \multimap \mathsf{S}$
$\mathrm{g}_{\textbf{récompenser}}^{\text{inf}}$ $: \mathsf{S_A} \multimap \mathsf{V_A} \multimap \mathsf{np} \multimap \mathsf{S}$	$\lambda^{\mathrm{o}}\, s^a\ v^a\ obj.\, \lambda^{\mathrm{o}}\ mod.\, \mathrm{C}_{\textbf{récompenser}}\ s^a\ (v^a\ mod)\ obj$ $: \mathsf{S_A} \multimap (\mathsf{V_A} \multimap \mathsf{V_A}) \multimap \mathsf{np} \multimap \mathsf{V_A} \multimap \mathsf{S}$
$\mathrm{g}_{\textbf{vraiment}} : \mathsf{V_A} \multimap \mathsf{V_A}$	$\lambda^{\mathrm{o}}\ a\ mod\ .\ a\ (\mathrm{C}_{\textbf{vraiment}}\ mod) : (\mathsf{V_A} \multimap \mathsf{V_A}) \multimap \mathsf{V_A} \multimap \mathsf{V_A}$

Table 2.4: Interpretations of constants under the lexicon $\mathscr{L}_{\mathsf{gder-tag}}$

Finally, we interpret $\mathsf{AnchorT} : \mathsf{S} \multimap \mathsf{T}$ as $\lambda^{\mathrm{o}}\, s.\, s\, \mathrm{I}_{\mathsf{V_A}} : (\mathsf{V_A} \multimap \mathsf{S}) \multimap \mathsf{S}$.

Remark 2.2. *We do not modify anything in the abstract vocabulary Σ_{GTAG} of the ACG encoding of TAG but add constants encoding clause-medial adverbials. Thus, in order to define semantic interpretations of the constants and types in Σ_{GTAG}, we only need to interpret the constants encoding clause-medial adverbials. The semantic interpretation of a constant $\mathrm{G}_{\textbf{adv}}^{\textit{medial}} : \mathsf{T} \multimap \mathsf{S} \multimap \mathsf{T}$ encoding the adverbial* adv *at the clause-medial position is identical to the semantic interpretation of the constant $\mathrm{G}_{\textbf{adv}_T}^{\ T} : \mathsf{T} \multimap \mathsf{T} \multimap \mathsf{T}$ encoding the adverbial* adv *at the clause-initial position. Indeed, there is only a syntactic difference between the two, otherwise, they have the same semantic interpretations as both of them are the lexicalizations of the same concept.*

Example 2.1.

Figure 2.6 shows a g-derivation tree, which gives rise to the text (29). To obtain the text (30), which is the variant of the text (29), we build the term $t_{\mathbf{GTAG}}^{\text{medial}}$ defined in Equation (2.45).

$$t_{\mathbf{GTAG}}^{\text{medial}} = \mathrm{G}_{\mathbf{ensuite}}^{\ \text{medial}}\ (\mathsf{AnchorT}\ (\mathrm{G}_{\mathbf{pour}}^{\ \text{red.}}\ \mathrm{G}_{\mathbf{jean}}\ (\mathrm{G}_{\mathbf{passé\text{-}laspirateur}}^{\ \text{sws}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ ) \\ (\mathrm{G}_{\mathbf{être\text{-}récomepnsé\text{-}par}}^{\ \text{inf}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{G}_{\mathbf{marie}}))\ (\mathrm{G}_{\mathbf{fait\text{-}une\text{-}sieste}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{G}_{\mathbf{a}}\ \mathrm{G}_{\mathbf{il}}) : \mathsf{T} \quad (2.45)$$

(29, repeated)
Jean a passé l'aspirateur pour être récompensé par
John have$_{\text{3PS. SG.}}$ pass$_{\text{PAST PART.}}$ vacuumer$_{\text{DEF.}}$ for to-be$_{\text{PRES. INF.}}$ reward$_{\text{PAST PART.}}$ by
Marie. Ensuite, *il a fait une sieste.*
Mary. *Afterward*, he have$_{\text{3PS. SG.}}$ make$_{\text{3PS. SG.}}$ a nap.

John vaccumed in order to be rewarded by Mary. Then, he took a nap.

(30, repeated)
Jean a passé l'aspirateur pour être récompensé par
John have$_{\text{3PS. SG.}}$ pass$_{\text{PAST PART.}}$ vacuumer$_{\text{DEF.}}$ for to-be$_{\text{PRES. INF.}}$ reward$_{\text{PAST PART.}}$ by
Marie. Il a ensuite *fait une sieste.*
Mary. He have$_{\text{3PS. SG.}}$ *afterward* make$_{\text{3PS. SG.}}$ a nap.

John vaccumed in order to be rewarded by Mary. He then took a nap.

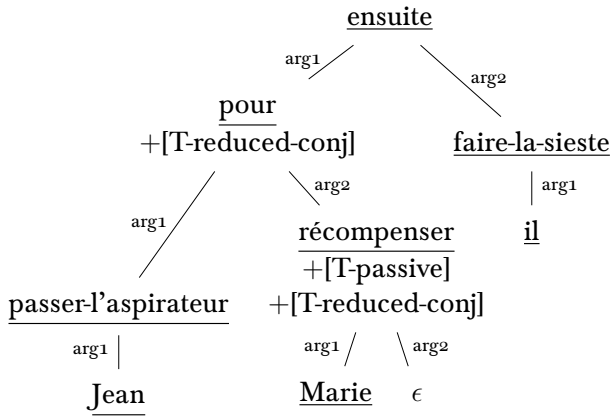


Figure 2.6: A g-derivation tree

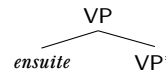
By interpreting the term $t_{\mathbf{GTAG}}^{\text{medial}}$ with the help of the lexicon $\mathscr{L}_{\mathsf{G}}$, we obtain the following term:

$$t_{\textbf{g-der}}^{\text{medial}} = \mathrm{g}_{\textbf{ensuite}}^{\text{medial}}\ (\mathsf{AnchorT}\ (\mathrm{g}_{\textbf{pour}}^{\text{red.}}\ (\mathrm{g}_{\textbf{passé-laspirateur}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{g}_{\textbf{jean}})\ (\mathrm{g}_{\textbf{être-récomepnsé-par}}^{\text{inf}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{g}_{\textbf{marie}})))$$
$$(\mathrm{g}_{\textbf{fait-une-sieste}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{g}_{\textbf{a}}\ \mathrm{g}_{\textbf{il}}) : \mathsf{T} \quad (2.46)$$

The structure of the term $t_{\textbf{g-der}}^{\text{medial}}$ in Equation (2.46) is closer to the g-derivation tree shown in Figure 2.6 on the facing page compared to the structure of the term $t_{\textbf{Gtag}}^{\text{medial medial}}$ (defined within Equation 2.45). Indeed, $\mathrm{g}_{\textbf{pour}}^{\text{red.}}$ has two arguments similar to an underspecified g-derivation tree of the conjunction *pour* (for).

The interpretation of the term $t_{\textbf{g-der}}^{\text{medial}}$ under the lexicon $\mathscr{L}_{\mathsf{gder-tag}}$ is the term $t_{\textbf{tag}}^{\text{medial}}$, which is as follows:

$$t_{\textbf{tag}}^{\text{medial}} = \mathrm{C}_{\textbf{concat}}\ (\mathrm{C}_{\textbf{pour}}^{\text{red.}}(\mathrm{C}_{\textbf{passé-laspirateur}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{C}_{\textbf{jean}})\ (\mathrm{C}_{\textbf{être-récomepnsé-par}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{C}_{\textbf{marie}}))$$
$$(\mathrm{C}_{\textbf{fait-une-sieste}}\ \mathrm{I}_{\mathsf{S_A}}\ \boxed{\mathrm{C}_{\textbf{ensuite}}^{\text{V}}}\ \mathrm{C}_{\textbf{il}}) : \mathsf{S} \quad (2.47)$$

In $t_{\textbf{tag}}^{\text{medial}}$, the constant $\mathrm{C}_{\textbf{ensuite}}^{\text{V}}$ (drawn within a box) stands for the tree 

. $\mathrm{C}_{\textbf{ensuite}}^{\text{V}}$ has an occurence in the sub-term of $t_{\textbf{tag}}^{\text{medial}}$ encoding a derivation tree of a clause. In particular, $\mathrm{C}_{\textbf{ensuite}}^{\text{V}}$ appears at a position where one models a VP-adjunction. By further interpreting the term $t_{\textbf{tag}}^{\text{medial}}$ under the lexicon $\mathscr{L}_{\text{synt}}^{\text{TAG}}$, we obtain the tree shown in Figure 2.7 on the next page. To obtain the yield of the tree representation, one interprets the term $t_{\textbf{tag}}^{\text{medial}}$ by the lexicon $\mathscr{L}_{\mathsf{yield}} \circ \mathscr{L}_{\text{synt}}^{\text{TAG}}$. We obtain the following yield:

$$\mathscr{L}_{\mathsf{yield}}(\mathscr{L}_{\text{synt}}^{\text{TAG}}(t_{\textbf{tag}}^{\text{der}})) =$$
$$\textit{Jean} + \textit{a} + \textit{passé} + \textit{l} + \textit{aspirateur} + \textit{pour} + \textit{être} + \textit{récompensé} + \textit{par} + \textit{marie} + \textit{dot} +$$
$$\textit{Il} + \textit{a} + \boxed{\textit{ensuite}} + \textit{fait} + \textit{une} + \textit{sieste}$$

As the term $\mathscr{L}_{\mathsf{yield}}(\mathscr{L}_{\text{synt}}^{\text{TAG}}(t_{\textbf{tag}}^{\text{medial}}))$ shows, the adverbial *ensuite* indeed occupies a clause-medial position.

In Appendix C on page 339, we provide the ACG codes that one can use in order to run this example with the ACG toolkit.

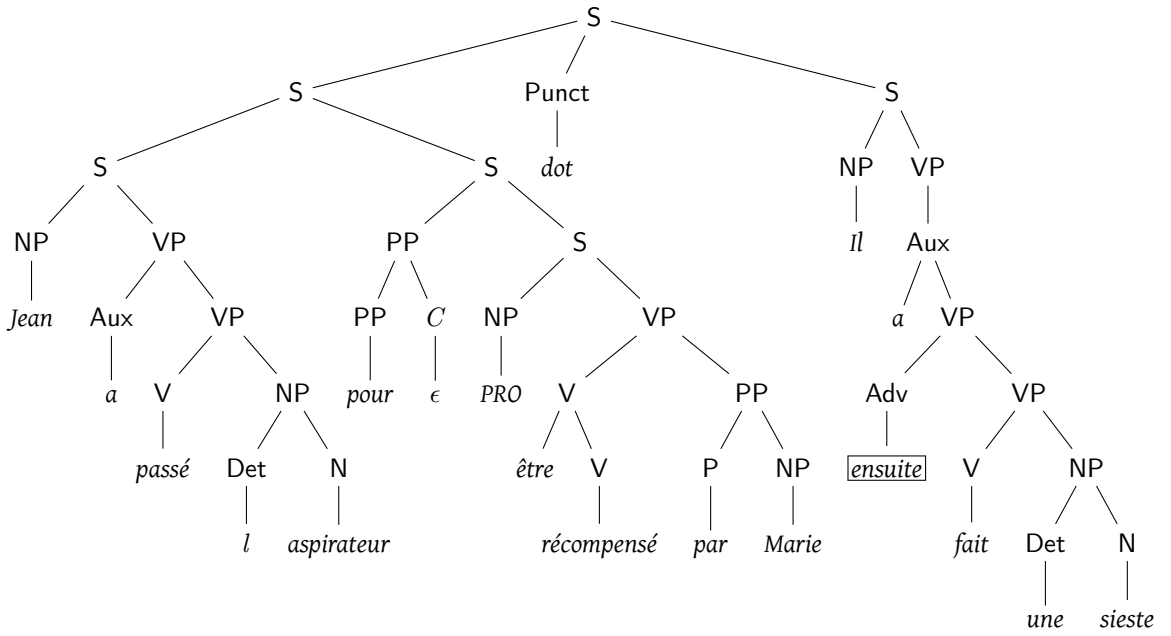


Figure 2.7: The tree obtained by interpreting the term $t^{\text{medial}}_{\mathbf{tag}}$ under the lexicon $\mathscr{L}^{\text{TAG}}_{\text{synt}}$

Chapter 3

D-STAG as ACGs

Contents

In this chapter, we encode D-STAG in the ACG framework. The architecture of the ACG encoding of D-STAG is similar to the ACG encoding of G-TAG and the encoding of TAG. That is, the D-STAG derivation trees are encoded as abstract terms. The D-STAG syntactic and semantic derived trees are obtained by interpreting the abstract terms under the corresponding lexicons. In this encoding, we propose a uniform modeling of clause-initial and clause-medial connectives. In addition, we provide semantic interpretations of terms encoding D-STAG derivation trees as labeled formulas instead of HOL formulas used in D-STAG.

3.1 Motivations

In Section 5.3, we discussed D-STAG (Danlos, 2011), which was proposed to address the problem of the syntax-semantics interface for discourse. With the help of D-STAG, one can interpret a discourse as a DAG. For instance, the DAGs depicted in Figure 3.1(a) and Figure 3.1(b) can serve as interpretations of discourses. In Chapter 1, we encoded G-TAG as ACGs. The ACG encoding of G-TAG allows one to model the syntax-semantics interface for discourse. Since the G-TAG grammar is designed for generating only the texts that have the tree-shaped discourse structures, with the help of the ACG encoding of G-TAG, one can only model texts with tree-shaped discourse structures. To design the ACGs that enable one to model texts whose discourse structures can be DAGs, we encode D-STAG as ACGs.

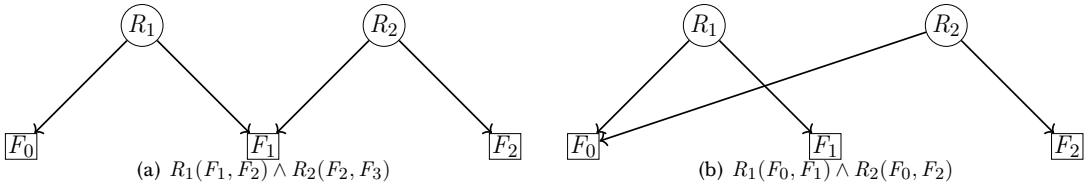


Figure 3.1: D-STAG semantic interpretations of discourse

3.2 The ACG Architecture of D-STAG

Parsing a discourse with D-STAG amounts to building its D-STAG derivation tree. Since D-STAG is based on Synchronous TAG (STAG) (Shieber and Schabes, 1990),[74] the derivation tree gives rise to both the syntactic and semantic interpretations of the discourse. Thus, a derivation tree is the pivot in a D-STAG analysis of a discourse.

[74]See Section 2.7 on page 47.

This is reminiscent of the ACG encoding of TAG with Montague semantics (Pogodalla, 2009)[75] and to the ACG encoding of G-TAG: The abstract terms are pivots for syntactic and semantic interpretations. Therefore, to design ACGs for D-STAG, we follow the same principles as in the case of the ACG encoding of G-TAG. That is, we model D-STAG derivation trees as abstract terms. One obtains the syntactic and semantic interpretations by interpreting the abstract terms under the corresponding lexicons.

In order to obtain the syntactic interpretations (trees), we first interpret the terms modeling D-STAG derivation trees as TAG derivation trees. Since the interpretations of TAG derivation trees as derived trees are already available, we compose the two interpretations. In this way, we obtain interpretations of D-STAG derivation trees as derived syntactic trees.

We interpret D-STAG derivation trees as HOL terms in order to model the D-STAG semantic interpretations. In addition, we define semantic interpretations of D-STAG derivation trees as labeled formulas. That is, like SDRT (Asher and Lascarides, 2003),[76] we encode discourse structures as labeled formulas. Figure 3.2 illustrates the ACG architecture that we are building to encode D-STAG.

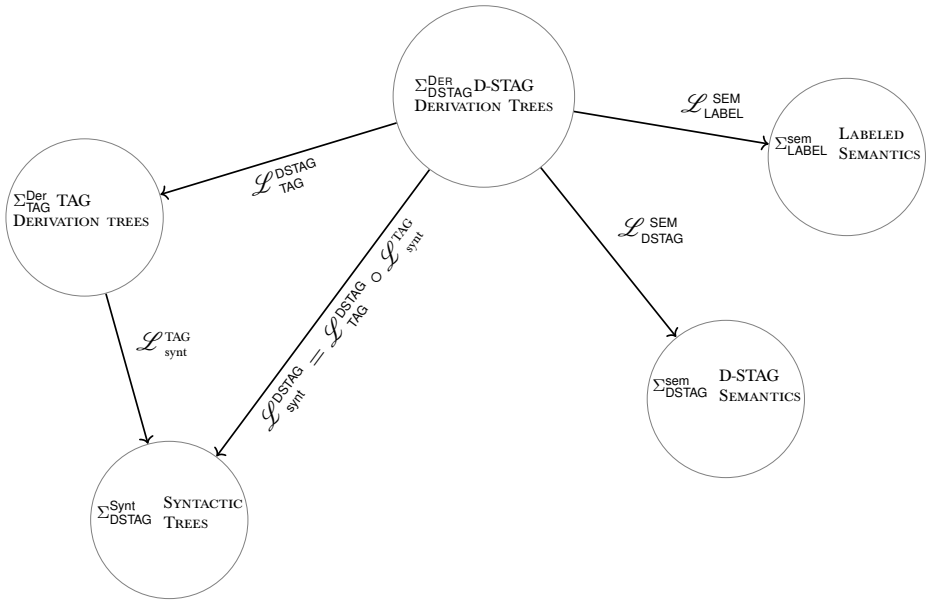


Figure 3.2: The ACG signatures and lexicons for encoding D-STAG

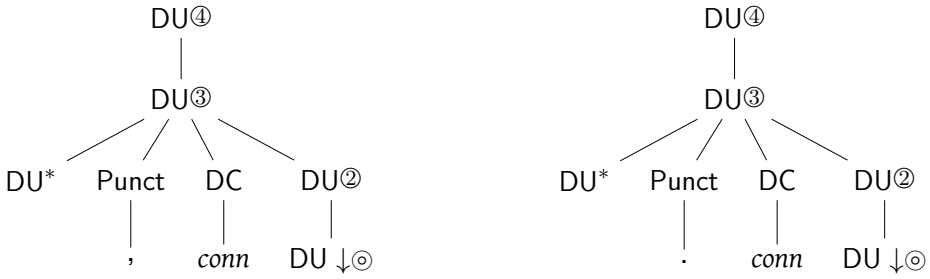
3.3 D-STAG Derivation Trees as Abstract Terms

To encode D-STAG derivation tree as abstract terms, we introduce constants and types in the abstract vocabulary Σ^{Der}_{DSTAG}. The elements in the set $\Lambda(\Sigma^{Der}_{DSTAG})$ model the D-STAG derivation trees. In D-STAG, the discourse connectives (such as discourse adverbials, subordinate conjunctions, empty connectives) account for the discourse structure. In

[75] See Section 3.8 on page 80.
[76] See Section 4.3 on page 107.

contrast to G-TAG where a discourse connective anchors an initial tree, any connective anchors an auxiliary tree in D-STAG. For now, we focus on the discourse connectives that are either postposed conjunctions or discourse adverbials. Figure 3.3 illustrates D-STAG elementary tree anchored with discourse connectives. An elementary tree anchored with a discourse connective is a DU-rooted auxiliary tree with three DU-adjunction sites (with the links ②, ③ and ④) and a single DU-substitution site (with the link ◎).[77]

Convention: For the sake of convenience, while discussing auxiliary trees anchored with discourse connectives, we may use the tree shown in Figure 3.4. In this tree, by *pmark*, we denote either the full stop, or the comma, or no punctuation mark.



(a) conn is a postposed conjunction

(b) conn is an adverbial or an empty connective

Figure 3.3: The auxiliary trees anchored with the *conn* discourse connective, where *conn* is either a preposed conjunction or a discourse adverbial

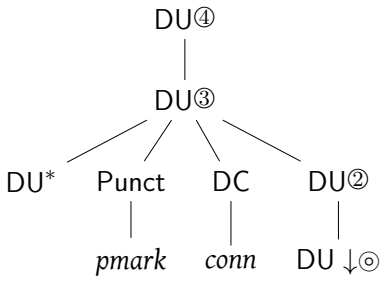


Figure 3.4: The auxiliary tree anchored with *conn*, where *conn* is either a preposed conjunction or a discourse adverbial

For a D-STAG auxiliary tree anchored with a *conn* connective, we introduce a constant $\mathrm{D}_{\mathbf{conn}}$ in the abstract vocabulary $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$. We model a DU-adjunction (resp. DU-substitution) site by introducing the type DU_{A} (resp. DU) in $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$. Since the D-STAG auxiliary tree anchored with the *conn* connective can receive three DU-adjunctions and one DU-substitution, we type[78] the constant $\mathrm{D}_{\mathbf{conn}}$ with the type $\mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU} \multimap \mathsf{DU}_{\mathsf{A}}$. We introduce the constant $\mathrm{I}_{\mathsf{DU}_{\mathsf{A}}}$ modeling the DU-adjunction with no content in order to model a case where no tree adjoins at a DU-adjunction site.

[77]We refer readers to Section 5.3 on page 174 for the notations used in D-STAG.

[78]We refer readers to Section 3.5 on page 69 for a detailed discussion about encoding adjunction and

Constants in $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$	Their Types
$\mathrm{D}_{\textbf{because}}$	$\mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU} \multimap \mathsf{DU}_{\mathsf{A}}$
$\mathrm{D}_{\textbf{then}}$	$\mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU} \multimap \mathsf{DU}_{\mathsf{A}}$
$\mathrm{D}_{\textbf{moreover}}$	$\mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU} \multimap \mathsf{DU}_{\mathsf{A}}$
$\mathrm{D}_{\textbf{and}}$	$\mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU} \multimap \mathsf{DU}_{\mathsf{A}}$
$\mathrm{D}_{\textbf{while}}$	$\mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU} \multimap \mathsf{DU}_{\mathsf{A}}$
$\mathrm{D}_{\textbf{afterwards}}$	$\mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU} \multimap \mathsf{DU}_{\mathsf{A}}$
D_{ϵ}	$\mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU} \multimap \mathsf{DU}_{\mathsf{A}}$
...	... .

Table 3.1: Constants in $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ encoding the D-STAG elementary trees anchored with postposed conjunctions, discourse adverbial, and the empty connective

Table 3.1 shows the encoding of the D-STAG auxiliary trees anchored with postposed conjunctions and adverbial connectives as constants of $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$. We encode the D-STAG auxiliary tree anchored with the lexically unexpressed connective ϵ with a constant D_{ϵ}.

In D-STAG, a clause C anchors a DU-initial tree $\overset{\mathsf{DU}^{\textcircled{1}}}{\underset{C}{|}}$. Thus, we encode it by a term of type DU. To model the anchor of the tree, i.e., the clause C, we introduce *the first order predicates*, that is, the constants that enable us to encode derivation trees of clauses. In order to introduce the first order predicates in $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$, we may adopt the abstract constants from the abstract vocabulary $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ defined in the ACG encoding of TAG with Montague semantics. The adopted constants from $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ enable us to build terms over $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ of types S, np, etc.

Thus, we model the derivation trees of clauses as terms in $\Lambda(\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}})$ of type S. However, to model the DU-rooted tree $\overset{\mathsf{DU}^{\textcircled{1}}}{\underset{C}{|}}$ anchored with the clause C, we need to transform a term of type S encoding the derivation tree of C clause to a term of type DU. In addition, since the tree $\overset{\mathsf{DU}^{\textcircled{1}}}{\underset{C}{|}}$ has a DU-adjunction site, we need to transform a term of type S to a term of the type $\mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU}$. For that, we introduce the constant $\mathsf{AnchorS}$ of type $\mathsf{S} \multimap \mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU}$ in the abstract vocabulary. If a term $\mathrm{t}_C : \mathsf{S}$ encodes the derivation tree of the clause C in $\Lambda(\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}})$, then we model the tree $\overset{\mathsf{DU}^{\textcircled{1}}}{\underset{C}{|}}$ as the following term over $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$:

$$\mathsf{AnchorS}\,\mathrm{t}_C \;:\; \mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU} \tag{3.1}$$

By adjoining a tree (a term of type DU_{A}) into the initial tree $\overset{\mathsf{DU}^{\textcircled{1}}}{\underset{C}{|}}$ (a term of type $\mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU}$), one obtains a derived tree of a discourse (a term of type DU). The resultant derived tree can be further used in order to build the discourse. Thus, a term of type DU does not model a *completed* discourse, i.e., a discourse that is not

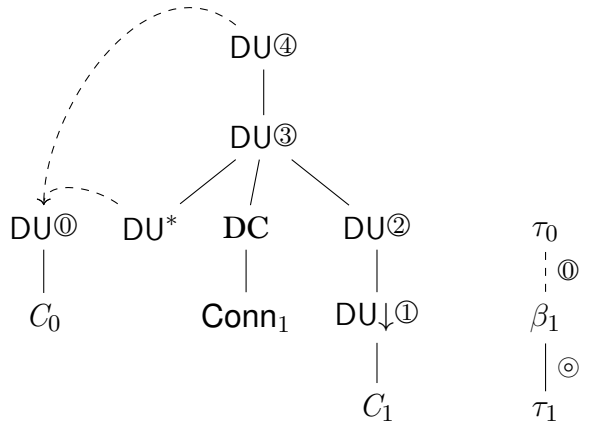
substitution sites.

going to be updated. To model the derivation tree of a discourse that is completed, we introduce one more type T in $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$. Terms of type T model derivation trees of completed discourses.

A discourse consisting of a single clause C is analyzed as $\overset{\mathsf{DU}①}{\underset{C}{\vdots}}$. We can model this tree as a term $\mathsf{AnchorS}\ \mathrm{t}_C\ \mathrm{I}_{\mathsf{DU_A}}$ of type DU. However, our goal is to obtain a term of type T instead of DU. To be able to do that, we introduce a constant $\mathsf{AnchorI}$ of type $\mathsf{S} \multimap \mathsf{DU_A} \multimap \mathsf{T}$. Thus, we model the derivation tree of the discourse consisting of the single clause C as a term $\mathsf{AnchorI}\ \mathrm{t}_C\ \mathrm{I}_{\mathsf{DU_A}}$ of type T, where $\mathrm{t}_C : \mathsf{S}$ is the derivation tree of the clause C.

Furthermore, with the help of the constants $\mathsf{AnchorI}$ and $\mathsf{AnchorS}$, we can model the larger discourses than single clause discourse. To model the first clause in a discourse, we employ the constant $\mathsf{AnchorI}$, whereas, in the rest of the cases, to model the initial trees anchored by clauses, we employ $\mathsf{AnchorS}$. For instance, let us consider a discourse $C_0\ \mathsf{Conn}_1\ C_1$. The way one obtains its derived tree is illustrated in Figure 3.5. Let t_{C_0} and t_{C_1} be the terms modeling the derivation trees of the clauses C_0 and C_1, respectively. We encode the derivation tree of the discourse by the following term:

$$\mathrm{t}_{C_0 \mathsf{Conn}_1 C_1} = \mathsf{AnchorI}\ \mathrm{t}_{C_0}\ (\mathrm{D}_{\mathsf{Conn}_1}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ (\mathsf{AnchorS}\ \mathrm{t}_{C_0}\ \mathrm{I}_{\mathsf{DU_A}})\ : \mathsf{T} \tag{3.2}$$



(a) The deriving the derived tree of the discourse

(b) The derivation tree

Figure 3.5: The D-STAG derivation tree of $C_0\ \mathsf{Conn}_1\ C_1$

In this way, we build the abstract vocabulary $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$. In order to define the abstract language, it remains to specify the distinguished type. Since we encode derivation trees of completed discourses with terms of type T, we declare T to be the distinguished type.

Remark 3.1. *In Section 5.3.5 on page 183, we discussed the ambiguity issues of D-STAG. The ambiguity in the D-STAG parsing is due to the number of possible derivation trees that a given*

discourse may have. Since we encode all the possible derivation trees with the abstract terms over Σ^{Der}_{DSTAG}, the ACG encoding of D-STAG inherits the D-STAG ambiguity in parsing.

3.3.1 Interpretations as TAG Derivation Trees

As we already mentioned, we define interpretations of D-STAG derivation trees as TAG derivation trees. By composing these interpretations with the interpretations of TAG derivation trees as derived trees, one obtains interpretations of D-STAG derivation trees as derived trees. Thus, we are building the ACG $\langle \Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}, \Sigma^{\mathsf{Der}}_{\mathsf{TAG}}, \mathscr{L}^{\mathsf{DSTAG}}_{\mathsf{TAG}}, \mathsf{T} \rangle$, shown in Figure 3.6.

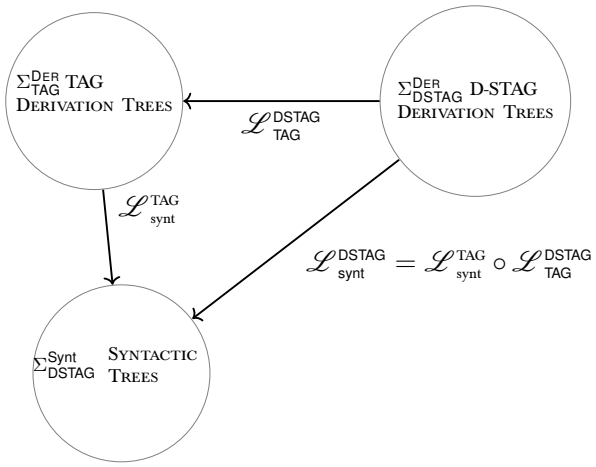


Figure 3.6: Interpretations of D-STAG trees as TAG derivations trees and derived trees

3.3.2 Connectives at the Clause-Medial & the Clause-Initial Positions

In D-STAG, in order to parse a discourse where an adverbial connective appears at a clause medial position (e.g. the discourse (3)), one preprocesses it by moving the adverbials occupying the clause-medial positions to the clause-initial ones. In that way, one obtains the discourse where every connective appears in front of its host clause. Thus, the D-STAG way of analyzing a discourse with clause-medial connectives consists of two steps (preprocessing and parsing).

(3) Fred went to the supermarket. He *then* went to the movies.

Unlike D-STAG, we do not develop a two-step approach, but rather analyze a discourse in a single step. We propose to encode a connective at a clause-medial position as a constant of the abstract vocabulary $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$. Indeed, by only using the constant $\mathrm{D}_{\mathbf{conn}} : \mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU}_{\mathsf{A}} \multimap \mathsf{DU} \multimap \mathsf{DU}_{\mathsf{A}}$, which we introduced in order to encode the D-STAG auxiliary tree anchored by a connective (see Figure 3.4 on page 252), one cannot give an account of a discourse where the connective *conn* occupies a clause-medial position. As the D-STAG elementary tree anchored by the

connective *conn* illustrates, the connective *conn* occupies the clause-initial position since it appears in front of the DU-substitution site. That is, the position of *conn* (in the surface form) is the position where the clause starts (the clause-initial position).

Thus, together with the constant $\mathrm{D}_{\mathbf{conn}}$ encoding a connective at a clause-initial position, we introduce another constant $\mathrm{D}^{\text{medial}}_{\mathbf{conn}}$ in $\Sigma^{\text{Der}}_{\text{DSTAG}}$ encoding the same connective at a *clause-medial position* (inside a verb-phrase of a clause). Here, the generic method of encoding constants modeling clause-medial connectives is the same as in the ACG encoding of clause-medial connectives in G-TAG. Figure 3.7(a) on the next page shows the way we analyze the clause-medial connectives in D-STAG: *The auxiliary tree anchored by the connective adjoins on the VP node in the derived tree of the host clause of the connective. In the resultant derived tree, the connective appears within the VP of the host clause (a clause-medial position.)*

To use the constant $\mathrm{D}^{\text{medial}}_{\mathbf{conn}}$, one must type it. There are several options for typing $\mathrm{D}^{\text{medial}}_{\mathbf{conn}}$.

Option 1: Introduce a New Type DU_v

In Chapter 2 on page 233, we encoded the constants modeling the clause-medial connectives with a different type from ones modeling the clause-initial connectives of G-TAG. In particular, we typed the constant $\mathrm{G}_{\mathbf{adv}}$ modeling the adverbial *adv* at a clause-initial position with the type $\mathsf{T} \multimap \mathsf{T} \multimap \mathsf{T}$, whereas we typed the constant $\mathrm{G}^{\text{medial}}_{\mathbf{adv}}$ modeling the adverbial *adv* at a clause-initial position with the type $\mathsf{T} \multimap \mathsf{S} \multimap \mathsf{T}$.

In the case of D-STAG, we may propose a similar solution to what we did in the case of G-TAG. That is, to distinguish the clause-medial and clause-initial usages of a connective, we can introduce a new type DU_v in the abstract vocabulary. We can associate the terms over $\Sigma^{\text{Der}}_{\text{DSTAG}}$ of type DU_v with the derivation trees of clauses that receive an adjunction at a clause-medial position. We can associate the terms of type DU to model the derivation trees of the clauses that do not receive any adjunction. Thus, to encode the *conn* discourse connective, we can introduce two constants shown in Table 3.2.

Constants in $\Sigma^{\text{Der}}_{\text{DSTAG}}$	Their Types
$\mathrm{D}^{\text{medial}}_{\mathbf{conn}}$	$\mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}_v \multimap \mathsf{DU}_\mathsf{A}$
$\mathrm{D}_{\mathbf{conn}}$	$\mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU} \multimap \mathsf{DU}_\mathsf{A}$

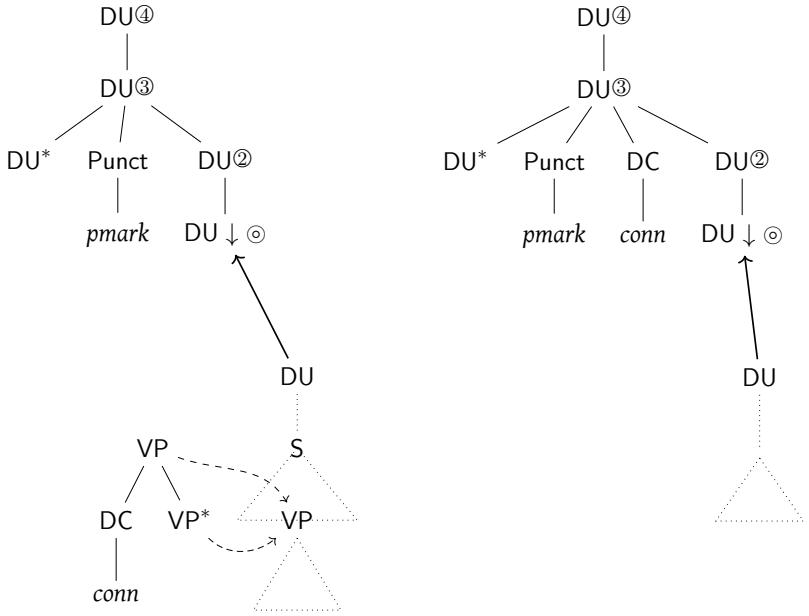
Table 3.2: Two constants encoding the discourse connective *conn*

The constant $\mathrm{D}^{\text{medial}}_{\mathbf{conn}}$ of type $\mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}_v \multimap \mathsf{DU}_\mathsf{A}$ encodes a clause-medial usage of the *conn* connective.

We can translate the constant $\mathrm{D}^{\text{medial}}_{\mathbf{conn}}$ to TAG derivation trees by encoding the derivation shown in Figure 3.7(a). Indeed, as Figure 3.7(a) shows, the substituted clause receives a VP-adjunction that inserts *conn* at a clause-medial position.

In order to interpret the constant $\mathrm{D}_{\mathbf{conn}}$ of type $\mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU} \multimap \mathsf{DU}_\mathsf{A}$ encoding the *conn* connective at the clause-initial position, we build the term encoding the derivation shown in Figure 3.7(b).

To interpret the types of $\mathrm{D}_{\mathbf{conn}}^{\text{medial}}$ and $\mathrm{D}_{\mathbf{conn}}$, we interpret DU as S and DU_A as S_A. The DU_v type translates to $\mathsf{V}_\mathsf{A} \multimap \mathsf{S}$, which encodes the fact that a term of type DU_v translates to a TAG derivation tree of a clause that can receieve a VP-adjunction (inserting a connective inside its VP).



(a) The clause-medial case (b) The clause-initial case

Figure 3.7: Syntactic trees of D-STAG discourse connectives

Option 2: Only One DU Type Encoding Discourse Units

Another option is to type the constants $\mathrm{D}_{\mathbf{conn}}^{\text{medial}}$ and $\mathrm{D}_{\mathbf{conn}}$ with same type, $\mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU} \multimap \mathsf{DU}_\mathsf{A}$. Thus, we do not introduce a new type. In this case, the encoding of constants modeling discourse connectives is more uniform compared to the other one, because now all of these constants are of the same type.

$$\begin{aligned} \mathrm{D}_{\mathbf{conn}} &: \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU} \multimap \mathsf{DU}_\mathsf{A} \\ \mathrm{D}_{\mathbf{conn}}^{\text{medial}} &: \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU} \multimap \mathsf{DU}_\mathsf{A} \end{aligned}$$

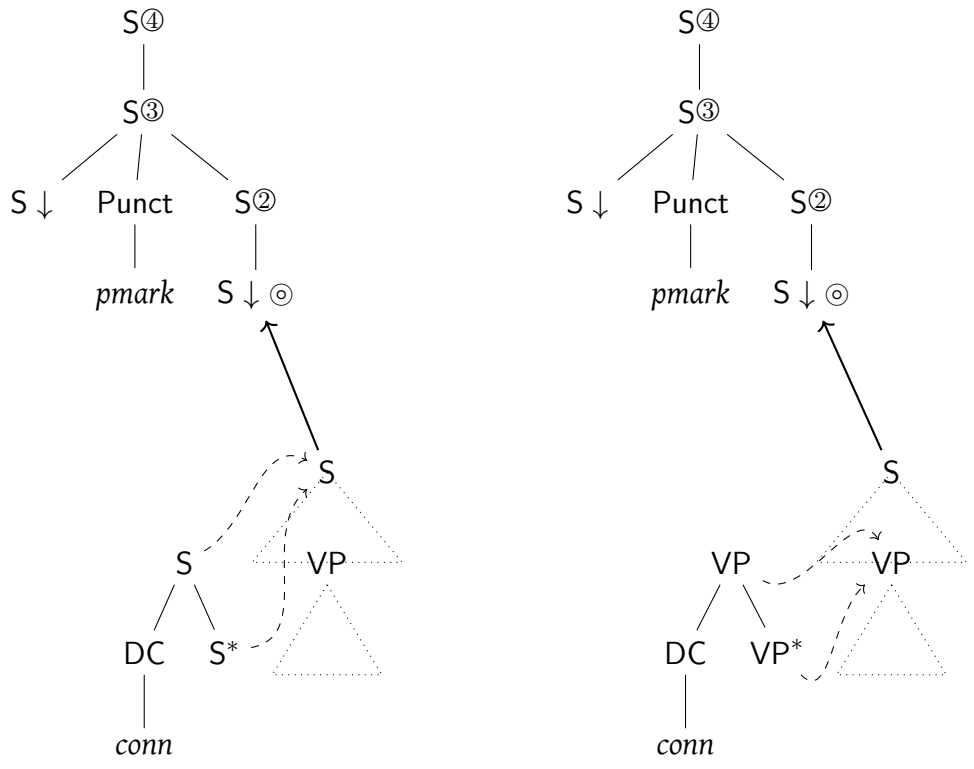
Table 3.3: two constants encoding the *conn* discourse connective

To be able to model the connectives at the clause-medial positions, we translate DU from $\Sigma_{\mathsf{DSTAG}}^{\mathsf{Der}}$ to $\mathsf{V}_\mathsf{A} \multimap \mathsf{S}$ in TAG derivation trees. This makes us able to adjoin on the host clause of $\Sigma_{\mathsf{DSTAG}}^{\mathsf{Der}}$ a VP-auxiliary tree anchored with a connective. However, in a case where one has a connective at the clause-initial position, the host clause of the connective should not receive a VP-adjunction inserting a connective at a clause-medial position. On the other hand, the type $\mathsf{V}_\mathsf{A} \multimap \mathsf{S}$ indicates that one has to perform a VP-adjunction in order to obtain a term of type S. To overcome this issue, we employ an empty VP-adjunction, i.e., the constant $\mathrm{I}_{\mathsf{V}_\mathsf{A}} : \mathsf{V}_\mathsf{A}$, which does not insert any content.

We choose this way of encoding clause-medial connectives, because the abstract vocabulary is more uniform and has less atomic types than in the previous case.

3.3.3 Clause-Initial and Clause-Medial Connectives as Adjunctions

We can make our approach to clause-initial and clause-medial connectives more uniform. Since we interpret a clause-medial connective with the help a VP-adjunction into a derived tree of clause, we propose the same kind of analysis of clause-initial connectives, but to use an S-adjunction instead of a VP-adjunction. Figure 3.8 illustrates these two analyses. Hence, a derived tree of a clause should be able to receive both an S-adjunction and a VP-adjunction that can insert a connective either at the clause-initial position (S-adjunction) or at the clause-medial one (VP-adjunction). Therefore, the interpretation of a term of type DU from $\Lambda(\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}})$ into TAG derivation trees should be able to receive both S-auxiliary and VP-auxiliary trees anchored with a connective, depending on whether one encodes the clause-initial or clause-medial connectives. To model that, one can interpret the type DU from $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ as $\mathsf{S}_\mathsf{A} \multimap \mathsf{V}_\mathsf{A} \multimap \mathsf{S}$ into $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$.



(a) The clause-initial case

(b) The clause-medial case

Figure 3.8: Analyses of the cases where connectives appear at the clause-medial and the clause-initial positions

3.3.4 A Clause-Medial Connective Between Two Adverbs

Although by interpreting DU from $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ as $\mathsf{S_A} \multimap \mathsf{V_A} \multimap \mathsf{S}$ into $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$, we achieve the goals defined so far, we propose to slightly change the interpretation of DU. Namely, we propose to interpret DU as $\mathsf{S_A} \multimap (\mathsf{V_A} \multimap \mathsf{V_A}) \multimap \mathsf{S}$. The reason behind interpreting DU as $\mathsf{S_A} \multimap (\mathsf{V_A} \multimap \mathsf{V_A}) \multimap \mathsf{S}$ is to model a syntactic phenomenon where a discourse connective appears between two VP adverbs, for instance, as it in the following example:

(4) a. Fred was desperate
b. because he was lost in the suburb of Paris for several hours.
c. Fred *fortunately*, *then*, *clearly* saw the Eiffel Tower.

In the clause (4)(c), the discourse connective *then* appears between two VP adverbs, *fortunately* and *clearly*. However, by interpreting DU as $\mathsf{S_A} \multimap \mathsf{V_A} \multimap \mathsf{S}$, the tree anchored with a clause-medial connective can adjoin on the VP of the clause. The connective will be placed either above or below the VP of the clause, the position where all the VP adverbs are adjoined. Thus, it would be impossible for a connective to appear between two adverbs. To overcome this problem, we explicitly encode the location inside the VP of the clause where an auxiliary tree anchored with the connective adjoins. We do that by encoding the adjunction sites *above* and *below* the place where a connective is going to be inserted within a derived tree of a clause. For that, in $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$, we encode the constants from which one obtains terms encoding derivation trees of clauses with two VP adjunction sites. For instance, in the case of (4)(c), we encode the initial tree anchored with *saw* with two distinct VP adjunction sites, as it is shown in Figure 3.9. Thus, we type the constant $\mathrm{D}_{\textbf{saw}}$ with the type $\mathsf{S} \multimap \mathsf{V_A} \multimap \mathsf{V_A} \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{S}$, where two $\mathsf{V_A}$ types encode two VP-adjunction sites in the initial tree shown in Figure 3.9. This enables us to define a term $\mathrm{t}^{\mathrm{DSTAG}}_{\textbf{saw}} \in \Lambda(\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}})$ (see Equation (3.5))[79] encoding the derivation tree of the clause (4)(c). Then, we can use it as an argument of $\mathrm{D}^{\mathrm{medial}}_{\textbf{then}}$.

$$\mathrm{t}^{\mathrm{DSTAG}}_{\textbf{saw}} = \mathsf{AnchorS}\ \mathrm{I}_{\mathsf{DU_A}}\ (\mathrm{D}_{\textbf{saw}}\ \mathrm{I}_{\mathsf{S_A}}\ \overbrace{(\mathrm{D}_{\textbf{clearly}}\ \mathrm{I}_{\mathsf{V_A}})}^{\mathsf{V_A}}\ \overbrace{(\mathrm{D}_{\textbf{fortunately}}\ \mathrm{I}_{\mathsf{V_A}})}^{\mathsf{V_A}}\ \mathrm{D}_{\textbf{Fred}}\ \mathrm{D}_{\textbf{the-Eiffel-tower}}) : \mathsf{DU} \quad (3.5)$$

The term $\mathrm{t}^{\mathrm{DSTAG}}_{\textbf{saw}} : \mathsf{DU}$ translates to a term that encodes that it can receive an S-adjunction (for inserting a clause-initial connective) and a VP adjunction (for inserting a clause-initial connective). Namely, we are aiming at interpreting the term $\mathrm{t}^{\mathrm{DSTAG}}_{\textbf{saw}}$ into $\Lambda(\Sigma^{\mathsf{Der}}_{\mathsf{TAG}})$ as follows:

$$\begin{aligned}\mathscr{L}^{\mathsf{DSTAG}}_{\mathsf{TAG}}(\mathrm{t}^{\mathrm{DSTAG}}_{\textbf{saw}}) = t^{\mathrm{TAG}}_{\textbf{saw}} = \\ = \lambda^{o}\, dc_s.\, \lambda^{o}\, dc_v.\, \mathrm{C}_{\textbf{saw}}\ \overbrace{dc_s}^{\mathsf{S_A}}\ \overbrace{(\mathrm{C}_{\textbf{clearly}}\ (dc_v\ (\mathrm{C}_{\textbf{fortunately}}\ \mathrm{I}_{\mathsf{V_A}})))}^{\mathsf{V_A}}\ \mathrm{C}_{\textbf{Fred}}\ \mathrm{C}_{\textbf{the-Eiffel-tower}} \\ : \mathsf{S_A} \multimap (\mathsf{V_A} \multimap \mathsf{V_A}) \multimap \mathsf{S}\end{aligned} \quad (3.6)$$

[79] We use the constant $\mathrm{D}_{\textbf{the-Eiffel-tower}}$ to model *the Eiffel Tower* since it does not have a compositional meaning (Kobele, 2012).

Where $\mathrm{C}_{\textbf{saw}} \in \Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ is the constant encoding a transitive verb *saw* and therefore it is of type $\mathsf{S_A} \multimap \mathsf{V_A} \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{S}$. Thus, we do not change encoding of initial trees in $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ but in $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$. Figure 3.10 illustrates the idea behind encoding initial trees with two VP adjunction sites on the example of $\mathrm{D}_{\textbf{saw}}$.

In the term $t^{\mathrm{TAG}}_{\textbf{saw}}$, by the variable $dc_v : \mathsf{V_A} \multimap \mathsf{V_A}$, we encode the position where a clause-medial connective can be inserted. Using the variable $dc_s : \mathsf{S_A}$, we encode the position where one can insert a clause-initial connective. As Equation (3.6) shows, the lexicon $\mathscr{L}^{\mathsf{DSTAG}}_{\mathsf{TAG}}$ interprets the type DU as $\mathsf{S_A} \multimap (\mathsf{V_A} \multimap \mathsf{V_A}) \multimap \mathsf{S}$.

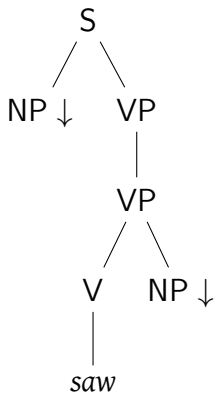


Figure 3.9: A visualization of the initial tree anchored with *saw* encoded by the constant $\mathrm{D}_{\textbf{saw}}$

Thus, in $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$, we modify the types of the constants that give rise to terms encoding derivation trees of clauses (constants standing for the initial trees anchored with verbs, predicative adjectives, etc.). In particular, we encode them with one more argument of type $\mathsf{V_A}$.

Remark 3.2. *By symmetry, one could also encode the same property for clause-initial connectives, as it is in the following example:*

(7) a. Fred went to Southern France for a week.

b. He planned to do nothing but sunbathe.

c. During the week, however, *every day, he was visiting either a museum or a theater.*

To do so, one could type constants encoding verbs as follows: $\cdots \multimap \mathsf{S_A} \multimap \mathsf{S_A} \multimap \cdots \multimap \mathsf{V_A} \multimap \mathsf{V_A} \multimap \cdots \multimap \mathsf{S}$.

However, we leave it for the future work to check the linguistic adequacy of the phenomenon of a fronted adverbial connective occupying a position between two fronted adverbs.

3.3.5 Interpretations of Types

We define the lexicon $\mathscr{L}^{\mathsf{DSTAG}}_{\mathsf{TAG}} : \Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}} \longrightarrow \Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ in order to interpret constants and types from $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ as terms and types built over $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ respectively.

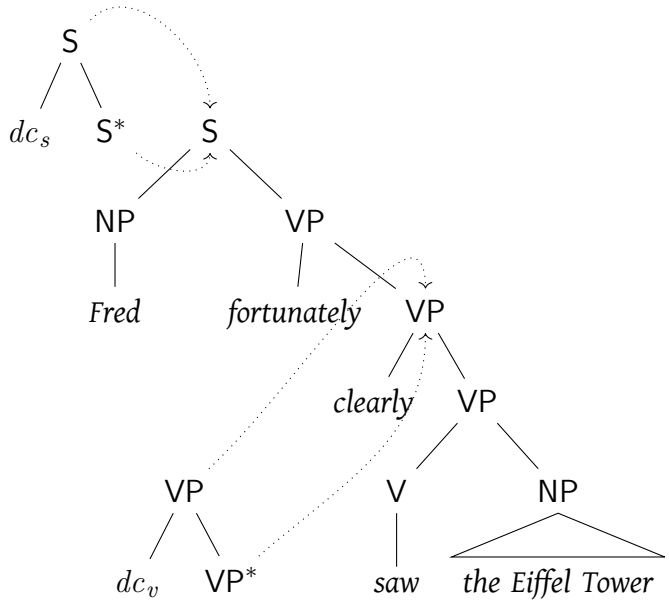


Figure 3.10: The illustration of a derived tree of a clause

The lexicon $\mathscr{L}^{\mathsf{DSTAG}}_{\mathsf{TAG}}$ interprets the type DU_A as S_A since DU_A is the type of a term modeling a DU-adjunction, which corresponds to an S-adjunction (S_A) in TAG derivation trees. We interpret the type S into TAG derivation trees as $\mathsf{S}_\mathsf{A} \multimap (\mathsf{V}_\mathsf{A} \multimap \mathsf{V}_\mathsf{A}) \multimap \mathsf{S}$. Thus, the interpretations of the types S and DU are the same. Table 3.4 shows the interpretations of these types together with the interpretations of the types S_A and V_A. In Section 3.3.6.2 on page 263, we justify our choices for interpreting S_A and V_A under the lexicon $\mathscr{L}^{\mathsf{DSTAG}}_{\mathsf{TAG}}$ as $\mathsf{S}_\mathsf{A} \multimap \mathsf{S}_\mathsf{A}$ and $\mathsf{V}_\mathsf{A} \multimap \mathsf{V}_\mathsf{A}$, respectively.[80]

The rest of the types translate to themselves as they encode noun phrases, common nouns, adjective etc., which model the same phenomena in $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ and $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$. That is, we interpret $\mathsf{X} \in \Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ as $\mathsf{X} \in \Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ if X is not among the types shown in Table 3.4.

Types in $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$	Their interpretations by $\mathscr{L}^{\mathsf{DSTAG}}_{\mathsf{TAG}} : \Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}} \longrightarrow \Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$
DU, S	$\mathsf{S}_\mathsf{A} \multimap (\mathsf{V}_\mathsf{A} \multimap \mathsf{V}_\mathsf{A}) \multimap \mathsf{S}$
DU_A	S_A
S_A	$\mathsf{S}_\mathsf{A} \multimap \mathsf{S}_\mathsf{A}$
V_A	$\mathsf{V}_\mathsf{A} \multimap \mathsf{V}_\mathsf{A}$
T	S

Table 3.4: Interpretations of types

[80]The reason for interpreting V_A (resp. S_A) as $\mathsf{V}_\mathsf{A} \multimap \mathsf{V}_\mathsf{A}$ (resp. $\mathsf{S}_\mathsf{A} \multimap \mathsf{S}_\mathsf{A}$) is the same as in the case of modeling clause-medial connectives in G-TAG: *It is done so in order to enable an additional VP-adjunction (resp. S-adjunction) on the derived tree of a clause.*

3.3.6 Interpretations of Constants

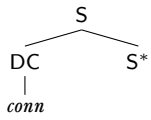
3.3.6.1 Discourse Connectives

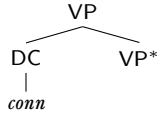
In order to interpret the constants modeling discourse connectives into TAG derivation trees, we refer to the analysis shown in Figure 3.8 on page 258. Table 3.5 provides the interpretations of the constants $\mathrm{D}_{\mathbf{conn}}$ and $\mathrm{D}_{\mathbf{conn}}^{\text{medial}}$ modeling the connective *conn* at a clause-initial and a clause-medial position, respectively.

Constants in $\Sigma^{\text{Der}}_{\text{DSTAG}}$	Their translations by $\mathscr{L}^{\text{DSTAG}}_{\text{TAG}} : \Sigma^{\text{Der}}_{\text{DSTAG}} \longrightarrow \Sigma^{\text{Der}}_{\text{TAG}}$
$\mathrm{D}_{\mathbf{conn}}$ $:\mathsf{DU_A}\multimap\mathsf{DU_A}\multimap\mathsf{DU_A}\multimap\mathsf{DU}\multimap\mathsf{DU_A}$	$\lambda^{o}\ d_4\ d_3\ d_2\ d_{subst}\ .\ \mathrm{C}_{\mathbf{concat}}\ d_4\ d_3\ d_2\ (d_{subst}\ \mathrm{C}_{\mathbf{conn}}^{\ \mathrm{S}}\ (\lambda^{o}x.x))$ $: \mathsf{S_A}\multimap\mathsf{S_A}\multimap\mathsf{S_A}\multimap(\mathsf{S_A}\multimap(\mathsf{V_A}\multimap\mathsf{V_A})\multimap\mathsf{S})\multimap\mathsf{S_A}$
$\mathrm{D}_{\mathbf{conn}}^{\text{medial}}$ $:\mathsf{DU_A}\multimap\mathsf{DU_A}\multimap\mathsf{DU_A}\multimap\mathsf{DU}\multimap\mathsf{DU_A}$	$\lambda^{o}\ d_4\ d_3\ d_2\ d_{subst}\ .\ \mathrm{C}_{\mathbf{concat}}\ d_4\ d_3\ d_2\ (d_{subst}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{C}_{\mathbf{conn}}^{\ \mathrm{VP}})$ $: \mathsf{S_A}\multimap\mathsf{S_A}\multimap\mathsf{S_A}\multimap(\mathsf{S_A}\multimap(\mathsf{V_A}\multimap\mathsf{V_A})\multimap\mathsf{S})\multimap\mathsf{S_A}$

Table 3.5: The interpretations of the constants in $\Sigma^{\text{Der}}_{\text{DSTAG}}$ encoding discourse connectives

In the interpretations given in Table 3.5, the variables d_4, d_3, and d_2 encode the interpretations of the DU-adjunction sites (the arguments of type $\mathsf{DU_A}$). These DU-adjunction sites are interpreted as S-adjunction in the tree anchored with *pmark*. We model this tree with the help of the constant $\mathrm{C}_{\mathbf{concat}} \in \Sigma^{\text{Der}}_{\text{TAG}}$ of type $\mathsf{S_A} \multimap \mathsf{S_A} \multimap \mathsf{S_A} \multimap \mathsf{S} \multimap \mathsf{S_A}$.[81] The variable d_{subst} encodes the interpretation of the host clause (the arguments of type DU) and therefore d_{subst} is of type $\mathsf{S_A} \multimap (\mathsf{V_A} \multimap \mathsf{V_A}) \multimap \mathsf{S}$. The type $\mathsf{S_A} \multimap (\mathsf{V_A} \multimap \mathsf{V_A}) \multimap \mathsf{S}$ encodes that d_{subst} can receive an S-adjunction ($\mathsf{S_A}$) and a VP-adjunction modification ($\mathsf{V_A} \multimap \mathsf{V_A}$).

In the case of the clause-initial connective $\mathrm{D}_{\mathbf{conn}}$, d_{subst} receives the S-adjunction inserting the connective *conn*, i.e., the S-auxiliary tree anchored with *conn*. We model this tree with the constant $\mathrm{C}_{\mathbf{conn}}^{\ \mathrm{S}}$, which is interpreted as the tree  into TAG derived trees. Since in this case nothing adjoins on the VP node of the host clause (d_{subst}), it receives the empty adjunction modification of type $\mathsf{V_A} \multimap \mathsf{V_A}$ as its second argument. We model that by applying d_{subst} to the identity function $\lambda^{o}x.x : (\mathsf{V_A} \multimap \mathsf{V_A})$.

In the case of the clause-medial connective $\mathrm{D}_{\mathbf{conn}}^{\text{medial}}$, the host clause ($d_{subst}$) receives no S-adjunction but the VP-adjunction modification inserting the connective *conn* at a clause-medial position. We model this fact with the help of the constants $\mathrm{I}_{\mathsf{S_A}}$ (no S-adjunction) and the constant $\mathrm{C}_{\mathbf{conn}}^{\ \mathrm{VP}}$, interpreted as  into TAG derived trees.

[81] *pmark* can have three values, either the full stop, or the comma, or the empty sign. Thus, we have tree constants $\mathrm{C}_{\mathbf{concat}^{.}}$, $\mathrm{C}_{\mathbf{concat}^{,}}$, and $\mathrm{C}_{\mathbf{concat}^{\epsilon}}$ modeling the trees anchored by the full stop, the comma and the empty punctuation sign, respectively.

3.3.6.2 First Order Predicates

In the previous sections, we have modified types of the constants that give rise to terms of type S (the ones modeling derivation trees of clauses). Namely, we encoded them with one more VP adjunction site. Nevertheless, in $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$, we have the same constants with the same types as before. For instance, if the type of a constant $\mathrm{D}_{\mathbf{v}} \in \Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ is $\mathsf{S_A} \multimap \mathsf{V_A} \multimap \mathsf{V_A} \multimap \underbrace{\mathsf{np} \multimap \cdots \multimap \mathsf{np}}_{k-\textit{times}} \multimap \mathsf{S}$, then the type of $\mathrm{C}_{\mathbf{v}} \in \Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ is $\mathsf{S_A} \multimap \mathsf{V_A} \multimap \underbrace{\mathsf{np} \multimap \cdots \multimap \mathsf{np}}_{k-\textit{times}} \multimap \mathsf{S}$. We interpret the constant $\mathrm{D}_{\mathbf{v}}$ into $\Lambda(\Sigma^{\mathsf{Der}}_{\mathsf{TAG}})$ as follows:

$$\mathscr{L}^{\mathsf{DSTAG}}_{\mathsf{TAG}}(\mathrm{D}_{\mathbf{v}}) =$$

$$\lambda^{o}\, sa\, va_1\, va_2\, np_1 \cdots np_k.\ \lambda^{o}\, dc_s . \lambda^{o}\, dc_v .\ \mathrm{C}_{\mathbf{v}}\ \overbrace{(sa\ dc_s)}^{\mathsf{S_A}}\ \overbrace{(va_2\ (dc_v\ (va_1\ \mathrm{I}_{\mathsf{V_A}})))}^{\mathsf{V_A}}\ np_1 \cdots np_k :$$
$$(\mathsf{S_A} \multimap \mathsf{S_A}) \multimap (\mathsf{V_A} \multimap \mathsf{V_A}) \multimap (\mathsf{V_A} \multimap \mathsf{V_A}) \multimap \underbrace{\mathsf{np} \multimap \cdots \multimap \mathsf{np}}_{k-\textit{times}} \multimap \mathsf{S_A} \multimap (\mathsf{V_A} \multimap \mathsf{V_A}) \multimap \mathsf{S} \tag{3.8}$$

In Equation (3.8), the sub-term $(sa\ dc_s) : \mathsf{S_A}$ encodes the possibility of inserting a connective into the clause-initial position. The variable dc_s is of type $\mathsf{S_A}$ as it is going to be substituted by a term modeling the S-auxiliary tree anchored with a connective. Hence, the type of the variable sa must be $\mathsf{S_A} \multimap \mathsf{S_A}$. The variable sa encodes the interpretation of the argument of type $\mathsf{S_A}$ of $\mathrm{D}_{\mathbf{v}} \in \Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$. Thus, we must interpret $\mathsf{S_A}$ from $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ as $\mathsf{S_A} \multimap \mathsf{S_A}$ in $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$.

In Equation (3.8), the sub-term $(va_2\ (dc_v\ (va_1\ \mathrm{I}_{\mathsf{V_A}}))) : \mathsf{V_A}$ encodes the adjunction that inserts a connective in the clause-medial position so that it can occupy a place between two VP adverbs. The interpretation of the type $\mathsf{V_A}$ from $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ must be $\mathsf{V_A} \multimap \mathsf{V_A}$, due to the same reason why we interpret $\mathsf{S_A}$ from $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ as $\mathsf{S_A} \multimap \mathsf{S_A}$ in $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$.

There are other first order predicates in $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ adopted from $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ whose types involve the atomic types $\mathsf{V_A}$ and/or $\mathsf{S_A}$. Since we interpret $\mathsf{V_A}$ (resp. $\mathsf{S_A}$) from $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ as $\mathsf{V_A} \multimap \mathsf{V_A}$ (resp. $\mathsf{S_A} \multimap \mathsf{S_A}$) in $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$, we must interpret the constants of $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ whose types involve $\mathsf{V_A}$ (resp. $\mathsf{S_A}$) accordingly. For the sake of illustration, Table 3.6 provides interpretations of the constants $\mathrm{D}_{\mathbf{really}} : \mathsf{V_A} \multimap \mathsf{V_A}$ and $\mathrm{D}_{\mathbf{indeed}} : \mathsf{S_A} \multimap \mathsf{S_A}$.

Constants in $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$	Their translations by $\mathscr{L}^{\mathsf{DSTAG}}_{\mathsf{TAG}}$
$\mathrm{D}_{\mathbf{really}} : \mathsf{V_A} \multimap \mathsf{V_A}$	$\lambda^{o}\, va\ v\ .\ \mathrm{C}_{\mathbf{really}}\ (va\ v) : (\mathsf{V_A} \multimap \mathsf{V_A}) \multimap (\mathsf{V_A} \multimap \mathsf{V_A})$
$\mathrm{D}_{\mathbf{indeed}} : \mathsf{S_A} \multimap \mathsf{S_A}$	$\lambda^{o}\, sa\ v\ .\ \mathrm{C}_{\mathbf{indeed}}\ (sa\ v) : (\mathsf{S_A} \multimap \mathsf{S_A}) \multimap (\mathsf{S_A} \multimap \mathsf{S_A})$

Table 3.6: Interpretations of first order predicates from $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ into $\Lambda(\Sigma^{\mathsf{Der}}_{\mathsf{TAG}})$

In $\Lambda(\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}})$, a term of type T encodes a derivation tree of a discourse derived from an initial tree anchored by a clause. In TAG derivation trees, one encodes derivation trees of clauses with terms of type S. Hence, we interpret T from $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ as S into $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$.

It remains to interpret the constants that enable us to build terms of type DU and T, namely, the constants $\mathsf{AnchorS} : \mathsf{S} \multimap \mathsf{DU_A} \multimap \mathsf{DU}$ and $\mathsf{AnchorI} : \mathsf{S} \multimap \mathsf{DU_A} \multimap \mathsf{T}$.

Constants in $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$	Their translations by $\mathscr{L}^{\mathsf{DSTAG}}_{\mathsf{TAG}}$
$\mathsf{AnchorI} : \mathsf{S} \multimap \mathsf{DU_A} \multimap \mathsf{T}$	$\lambda^o s\ m.\, \mathsf{Mod}\ (s\ \mathrm{I}_{\mathsf{S_A}}\ (\lambda^o x.x))\ m$
$\mathsf{AnchorS} : \mathsf{S} \multimap \mathsf{DU_A} \multimap \mathsf{DU}$	$\lambda^o s\ m\ dc_s\ dc_v.\, \mathsf{Mod}\ (s\ dc_s\ dc_v)\ m$

Table 3.7: Interpretations of AnchorI and AnchorS into $\Lambda(\Sigma^{\mathsf{Der}}_{\mathsf{TAG}})$

In Table 3.7, we propose the interpretations of the constants AnchorS and AnchorI. The variable s of type $\mathsf{S_A} \multimap (\mathsf{V_A} \multimap \mathsf{V_A}) \multimap \mathsf{S}$ models the interpretation into $\Lambda(\Sigma^{\mathsf{Der}}_{\mathsf{TAG}})$ of the D-STAG derivation tree of a clause (a term of type S in $\Lambda(\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}})$). The type $\mathsf{S_A} \multimap (\mathsf{V_A} \multimap \mathsf{V_A}) \multimap \mathsf{S}$ encodes the fact that one can insert a clause-initial ($\mathsf{S_A}$) or a clause-medial connective ($\mathsf{V_A} \multimap \mathsf{V_A}$) in a clause. However, we apply AnchorI to a term modeling the first clause in a discourse. Hence, no connective has to appear inside that clause. Thus, we have neither an S nor VP adjunction on the clause. To model that, we apply s to $\mathrm{I}_{\mathsf{S_A}}$ (the empty S-adjunction) and $\lambda^o x.x$ (the empty $\mathsf{V_A} \multimap \mathsf{V_A}$ adjunction modification). Thus, we obtain a term $(s\ \mathrm{I}_{\mathsf{S_A}}\ (\lambda^o x.x))$ of type S. The variable m models a tree adjoining in the initial tree anchored with the clause $\overset{\mathsf{DU}^{\textcircled{1}}}{\underset{C}{|}}$. Therefore, the type of m is the interpretation of $\mathsf{DU_A}$ into $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$, i.e., $\mathsf{S_A}$. One models an adjunction as a functional application. However, one cannot apply $m : \mathsf{S_A}$ to $(s\ \mathrm{I}_{\mathsf{S_A}}\ (\lambda^o x.x)) : \mathsf{S}$ since both are of atomic types. To express that a term of type $\mathsf{S_A}$ applies to a term of type S, we introduce a constant Mod in $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ of type $\mathsf{S} \multimap \mathsf{S_A} \multimap \mathsf{S}$. The interpretation of Mod in TAG derived trees will be application of the second argument (whose type will be $\tau \multimap \tau$ as it is interpretation of $\mathsf{S_A}$) to the first one (whose type will be τ as it is interpretation of S). The resultant term will stand for the derived tree of the discourse.

We employ the constant AnchorS to model derivation trees of the clauses into which connective may appear, either at the clause-initial or at the clause-medial position. To encode that, we define a sub-term $(s\ dc_s\ dc_v)$ (see Table 3.7). In the interpretation of AnchorS, the variable m encodes an adjoined tree, like in the case of interpreting AnchorI. We use Mod for the same reasons as in the case of interpreting AnchorI.

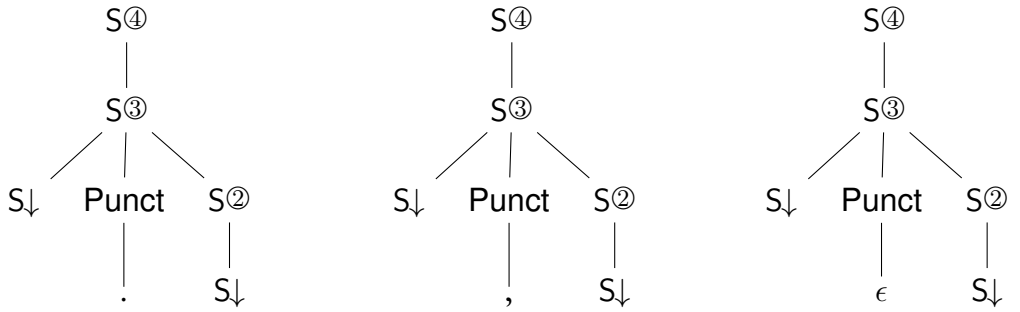
The rest of the constants in $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ are the ones whose types do not involve S, $\mathsf{V_A}$, and $\mathsf{S_A}$. We interpret a constant $\mathrm{D}_{\mathbf{u}}$ modeling ***u*** in $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ (where ***u*** can be a noun, determiner, adjective etc.) as the constant $\mathrm{C}_{\mathbf{u}} \in \Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ modeling the same ***u*** in $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ if the type of $\mathrm{D}_{\mathbf{u}}$ is not built using any of the types DU, $\mathsf{DU_A}$ S, $\mathsf{V_A}$, and $\mathsf{S_A}$.

3.3.7 Interpretations of Newly Introduced Constants $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ as Derived Trees

In Section 3.3.6.1, we introduced a constant $\mathrm{C}_{\mathbf{concat}} : \mathsf{S_A} \multimap \mathsf{S_A} \multimap \mathsf{S_A} \multimap \mathsf{S} \multimap \mathsf{S_A}$ in order to interpret the constants modeling discourse connectives in TAG derivation trees (see Table 3.5). The constant $\mathrm{C}_{\mathbf{concat}}$ stands for the tree anchored with *pmark*. However, *pmark* can have three values, the full stop, the comma, and the empty string. To model that, one needs three constants instead of one $\mathrm{C}_{\mathbf{concat}}$. We denote them by $\mathrm{C}_{\mathbf{concat}^{.}}$, $\mathrm{C}_{\mathbf{concat}^{,}}$, and $\mathrm{C}_{\mathbf{concat}^{\epsilon}}$. They model the elementary trees shown in Figure 3.11. To interpret them into derived trees, we encode these elementary trees into derived trees. Table 3.8 provides

the interpretations of these constants.

To define the interpretations given in Table 3.7, we introduced the constant Mod : $\mathsf{S} \multimap \mathsf{S}_\mathsf{A} \multimap \mathsf{S}$ in $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$. Mod express an adjunction of two trees. In the ACG encoding of TAG, one models an adjunction as a functional application. Table 3.8 shows the interpretation of Mod.



(a) The tree anchored by the full stop - $\mathrm{C}_{\textbf{concat}^{\cdot}}$

(b) The tree anchored by the comma - $\mathrm{C}_{\textbf{concat}^{,}}$

(c) The tree with the empty anchor - $\mathrm{C}_{\textbf{concat}^{\epsilon}}$

Figure 3.11: The elementary tree anchored by $\mathrm{C}_{\textbf{concat}^{\cdot}}$, $\mathrm{C}_{\textbf{concat}^{,}}$, and $\mathrm{C}_{\textbf{concat}}$

Constants in $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$	Their translations by $\mathscr{L}^{\mathsf{TAG}}_{\mathsf{synt}} : \Sigma^{\mathsf{Der}}_{\mathsf{TAG}} \longrightarrow \Sigma^{\mathsf{Synt}}_{\mathsf{DSTAG}}$
$\mathrm{C}_{\textbf{concat}^{\cdot}} : \mathsf{S}_\mathsf{A} \multimap \mathsf{S}_\mathsf{A} \multimap \mathsf{S}_\mathsf{A} \multimap \mathsf{S} \multimap \mathsf{S}_\mathsf{A}$	$\lambda^{o}\ sa_4\ sa_3\ sa_2\ s\ x.\ sa_4\ (sa_3\ (\mathsf{S}_3\ x\ \textbf{\textit{dot}}\ (sa_2\ s)))$
$\mathrm{C}_{\textbf{concat}^{,}} : \mathsf{S}_\mathsf{A} \multimap \mathsf{S}_\mathsf{A} \multimap \mathsf{S}_\mathsf{A} \multimap \mathsf{S} \multimap \mathsf{S}_\mathsf{A}$	$\lambda^{o}\ sa_4\ sa_3\ sa_2\ s\ x.\ sa_4\ (sa_3\ (\mathsf{S}_3\ x\ \textbf{\textit{comma}}\ (sa_2\ s)))$
$\mathrm{C}_{\textbf{concat}^{\epsilon}} : \mathsf{S}_\mathsf{A} \multimap \mathsf{S}_\mathsf{A} \multimap \mathsf{S}_\mathsf{A} \multimap \mathsf{S} \multimap \mathsf{S}_\mathsf{A}$	$\lambda^{o}\ sa_4\ sa_3\ sa_2\ s\ x.\ sa_4\ (sa_3\ (\mathsf{S}_3\ x\ \epsilon\ (sa_2\ s)))$
$\mathsf{Mod} : \mathsf{S} \multimap \mathsf{S}_\mathsf{A} \multimap \mathsf{S}$	$\lambda^{o}\ s.\ \lambda^{o}\ m.\ m\ s :\ \tau \multimap (\tau \multimap \tau) \multimap \tau$

Table 3.8: Interpretations of the constants introduced in $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ into derived trees

3.3.8 The Examples of Deriving D-STAG Syntactic Trees

We show the derivation of the syntactic interpretations of the following examples:[82]

(9) a. [Fred is grumpy]$_0$ because [he lost his keys]$_1$. Moreover, [he failed his exam]$_2$.

b. [Fred is grumpy]$_0$ because [he didn't sleep well]$_3$. [He had nightmares]$_4$.

c. [Fred went to the supermarket]$_5$ because [his fridge was empty]$_6$. Then, [he went to the movies]$_7$.

d. [Fred went to the supermarket]$_5$ because [his fridge was empty]$_6$. [He *then* went to the movies]$_{7^m}$.

[82]We discussed the way D-STAG encodes these examples in Section 5.3.

e. [Fred is grumpy]$_0$ because [his wife is away this week]$_8$. [This shows how much he loves her]$_9$.

The ACG signatures and lexicons together with the examples are provided in Appendix D.1.

The terms encoding the derivation trees of the clauses are as follows:[83]

$$\mathrm{t}_{C_0} = \mathrm{D}_{\textbf{is}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{D}_{\textbf{fred}}\ (\mathrm{D}_{\textbf{grumpy}}\ \mathrm{I}_{\mathsf{adj}})\ :\mathsf{S} \tag{3.10}$$
$$\mathrm{t}_{C_1} = \mathrm{D}_{\textbf{lost}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{D}_{\textbf{he}}\ (\mathrm{D}_{\textbf{keys}}\ \mathrm{D}_{\textbf{his}}\ \mathrm{I}_{\mathsf{n_A}})\ :\mathsf{S} \tag{3.11}$$
$$\mathrm{t}_{C_2} = \mathrm{D}_{\textbf{failed}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{D}_{\textbf{he}}\ (\mathrm{D}_{\textbf{exam}}\ \mathrm{D}_{\textbf{his}}\ \mathrm{I}_{\mathsf{n_A}})\ :\mathsf{S} \tag{3.12}$$
$$\mathrm{t}_{C_3} = \mathrm{D}_{\textbf{sleep}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ (\mathrm{D}_{\textbf{didn't}}\ \mathrm{I}_{\mathsf{V_A}})\ \mathrm{D}_{\textbf{he}}\ :\mathsf{S} \tag{3.13}$$
$$\mathrm{t}_{C_4} = \mathrm{D}_{\textbf{had}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{D}_{\textbf{he}}\ (\mathrm{D}_{\textbf{nightmare}}\ \mathrm{D}_{\textbf{plur}}\ \mathrm{I}_{\mathsf{n_A}})\ :\mathsf{S} \tag{3.14}$$
$$\mathrm{t}_{C_5} = \mathrm{D}_{\textbf{went-to}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{D}_{\textbf{fred}}\ (\mathrm{D}_{\textbf{supermarket}}\ \mathrm{D}_{\textbf{the}}\ \mathrm{I}_{\mathsf{n_A}})\ :\mathsf{S} \tag{3.15}$$
$$\mathrm{t}_{C_6} = \mathrm{D}_{\textbf{was}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{I}_{\mathsf{V_A}}\ (\mathrm{D}_{\textbf{fridge}}\ \mathrm{D}_{\textbf{the}}\ \mathrm{I}_{\mathsf{n_A}})\ (\mathrm{D}_{\textbf{empty}}\ \mathrm{I}_{\mathsf{adj}})\ :\mathsf{S} \tag{3.16}$$
$$\mathrm{t}_{C_7} = \mathrm{D}_{\textbf{went-to}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{D}_{\textbf{fred}}\ (\mathrm{D}_{\textbf{movies}}\ \mathrm{D}_{\textbf{the}}\ \mathrm{I}_{\mathsf{n_A}})\ :\mathsf{S} \tag{3.17}$$
$$\mathrm{t}_{C_8} = \mathrm{D}_{\textbf{is}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{I}_{\mathsf{V_A}}\ (\mathrm{D}_{\textbf{wife}}\ \mathrm{D}_{\textbf{his}}\ \mathrm{I}_{\mathsf{n_A}})(\mathrm{D}_{\textbf{away}}\ \mathrm{I}_{\mathsf{adj}}\ (\mathrm{D}_{\textbf{the}}\ (\mathrm{D}_{\textbf{week}}\ \mathrm{I}_{\mathsf{n_A}})))\ :\mathsf{S} \tag{3.18}$$
$$\mathrm{t}_{C_9} = \mathrm{D}_{\textbf{shows}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{D}_{\textbf{this}}\ (\mathrm{D}_{\textbf{loves}}\ \mathrm{D}_{\textbf{how-much}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{D}_{\textbf{he}}\ \mathrm{D}_{\textbf{her}})\ :\mathsf{S} \tag{3.19}$$

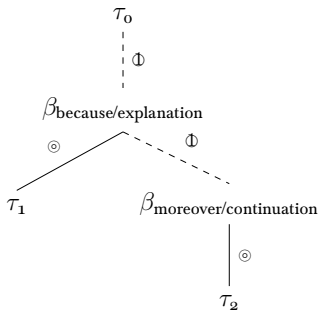
Example 3.1.

(9)(a) [Fred is grumpy]$_0$ because [he lost his keys]$_1$. Moreover, [he failed his exam]$_2$.

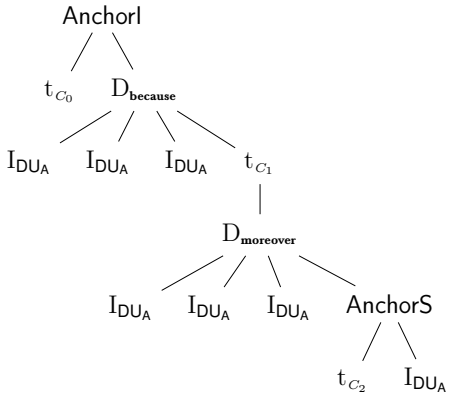
Figure 3.12(a) illustrates the D-STAG derivation tree of (9)(a). We encode it in the abstract language with the term t_1, defined in Equation (3.20), whose tree representation is shown in Figure 3.12(b). By interpreting the term t_{C_0} under the lexicon $\mathscr{L}^{\mathsf{TAG}}_{\mathsf{synt}} \circ \mathscr{L}^{\mathsf{DSTAG}}_{\mathsf{TAG}}$, we obtain the derived syntactic tree of the discourse depicted in Figure 3.12(c).

$$\mathrm{t}_1 = \tag{3.20}$$
$$\mathsf{AnchorI}\ \mathrm{t}_{C_0}\ (\mathrm{D}_{\textbf{because}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ (\mathrm{t}_{C_1}\ (\mathrm{D}_{\textbf{moreover}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ (\mathsf{AnchorS}\ \mathrm{t}_{C_2}\ \mathrm{I}_{\mathsf{DU_A}}))))\ :\mathsf{T}$$

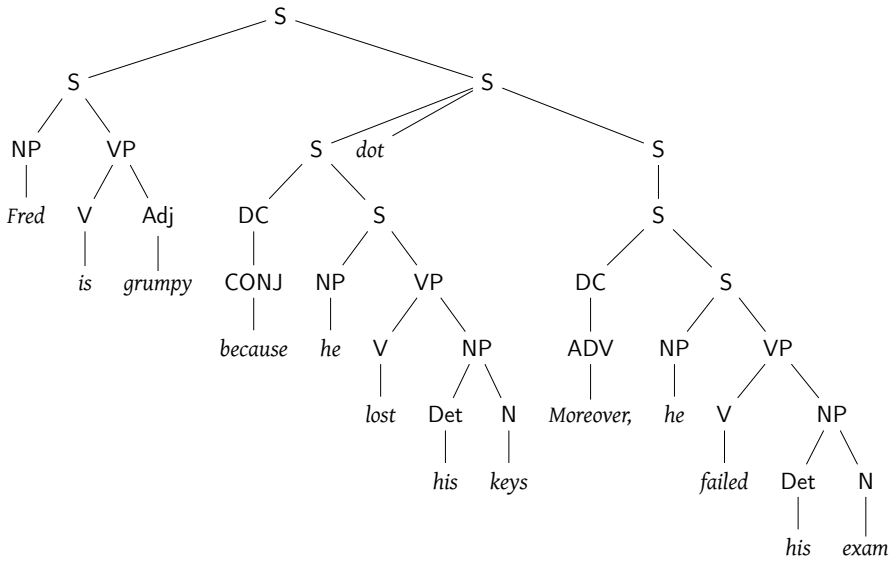
[83]The term encoding the derivation tree of a clause whose boundaries are indicated using a subscript i is denoted by t_{C_i}.



(a) The D-STAG derivation tree



(b) The ACG encoding of the D-STAG derivation tree



(c) The derived syntactic tree

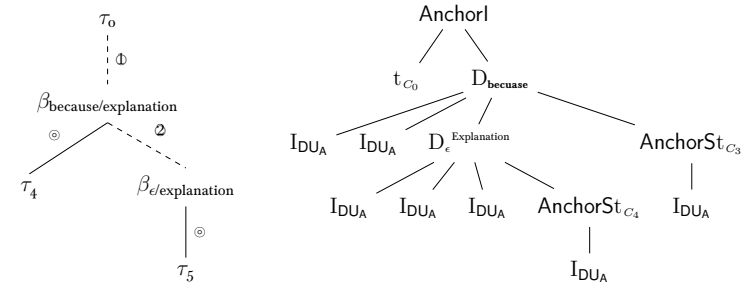
Figure 3.12: The D-STAG derivation tree, its ACG encoding, and the derived syntactic tree

Example 3.2.

(9)(b) [Fred is grumpy]$_0$ because [he didn't sleep well]$_3$. [He had nightmares]$_4$.

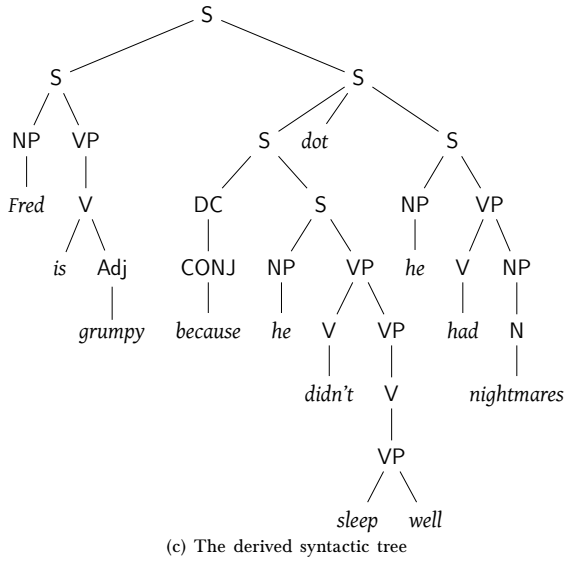
Figure 3.13(a) illustrates the D-STAG derivation tree of (9)(b). We encode it in the abstract language with the term t_2, defined in Equation (3.21), whose tree representation is shown in Figure 3.13(b). By translating the term t_2 with the lexicon $\mathscr{L}^{\mathrm{TAG}}_{\mathrm{synt}} \circ \mathscr{L}^{\mathsf{DSTAG}}_{\mathsf{TAG}}$, we obtain the derived tree of the discourse depicted in Figure 3.13(c).

$$\mathrm{t}_2 = \mathsf{AnchorI}\ \mathrm{t}_{C_0} \qquad (3.21)$$
$$(\mathrm{D}_{\textbf{becuase}}\ \mathsf{I_{DU_A}}\ \mathsf{I_{DU_A}}\ (\mathrm{D}_{\epsilon}{}^{\text{Explanation}}\ \mathsf{I_{DU_A}}\ \mathsf{I_{DU_A}}\ \mathsf{I_{DU_A}}\ (\mathsf{AnchorS}\ \mathrm{t}_{C_4}\ \mathsf{I_{DU_A}}))(\mathsf{AnchorS}\ \mathrm{t}_{C_3}\ \mathsf{I_{DU_A}})\ ) : \mathsf{T}$$



(a) The D-STAG derivation tree

(b) The ACG encoding of the D-STAG derivation tree



(c) The derived syntactic tree

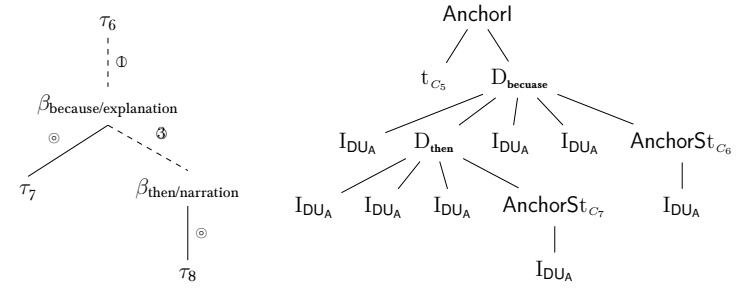
Figure 3.13: The D-STAG derivation tree, its ACG encoding, and the derived syntactic tree

Example 3.3.

(9)(c) [Fred went to the supermarket]$_5$ because [his fridge was empty]$_6$. Then, [he went to the movies]$_7$.

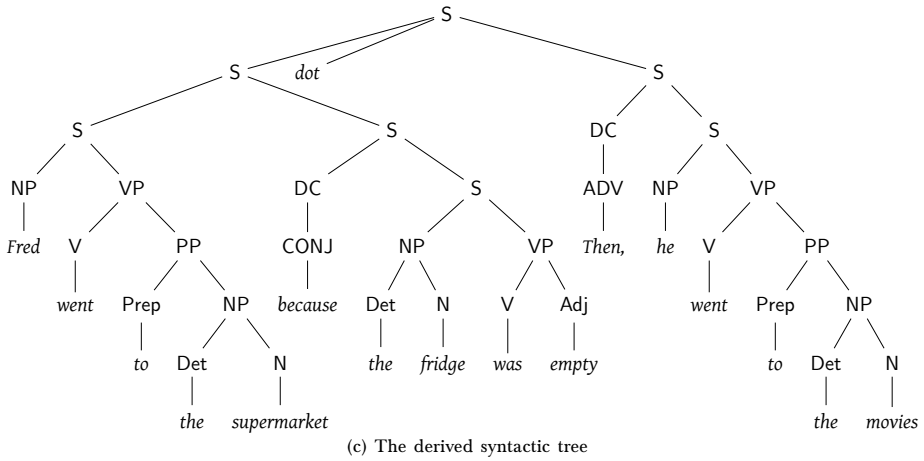
Figure 3.14(a) illustrates the D-STAG derivation tree of (9)(c). We encode it in the abstract language with the term t_3, defined in Equation (3.22). The tree representation of the term t_3 is provided in Figure 3.14(b). The interpretation of the term t_3 by the lexicon $\mathscr{L}_{\text{synt}}^{\text{TAG}} \circ \mathscr{L}_{\text{TAG}}^{\text{DSTAG}}$ is the derived tree of the discourse depicted in Figure 3.14(c).

$$\begin{aligned} \mathrm{t}_3 = \mathsf{AnchorI}\ \mathrm{t}_{C_5}\ (\mathrm{D}_{\textbf{becuase}}\ \mathrm{I}_{\mathsf{DU_A}} \\ (\mathrm{D}_{\textbf{then}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ (\mathsf{AnchorS}\ \mathrm{t}_{C_7}\ \mathrm{I}_{\mathsf{DU_A}}))\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ (\mathsf{AnchorS}\ \mathrm{t}_{C_6}\ \mathrm{I}_{\mathsf{DU_A}}) \end{aligned} \tag{3.22}$$



(a) The D-STAG derivation tree

(b) The ACG encoding of the D-STAG derivation tree



(c) The derived syntactic tree

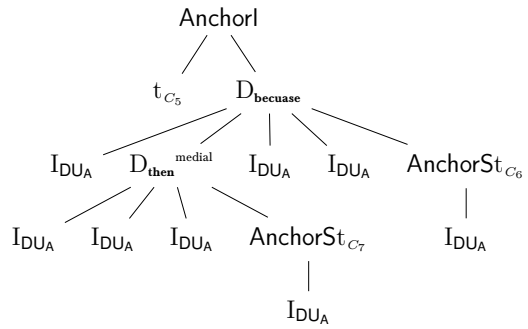
Figure 3.14: The D-STAG derivation tree, its ACG encoding, and the derived syntactic tree

Example 3.4.

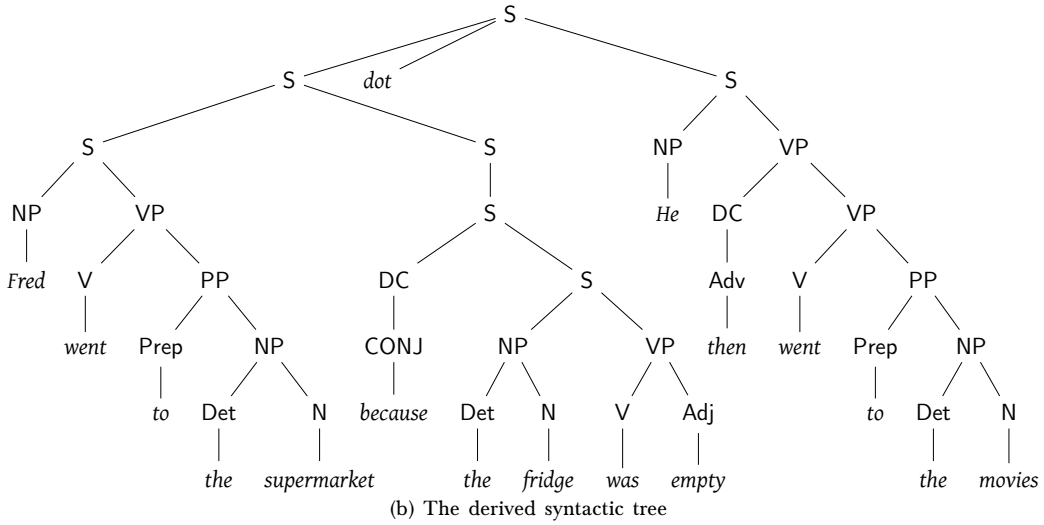
(9)(d) [Fred went to the supermarket]$_5$ because [his fridge was empty]$_6$. [He *then* went to the movies]$_{7^m}$.

The only difference between the discourses (9)(d) and (9)(c) (see Example 3.3) is that the discourse adverbial *then* is at the clause-medial position in the case of (9)(d), whereas *then* appears at the clause-initial position in the case of (9)(c). The D-STAG derivation tree of the discourses (9)(d) and (9)(c) are the same as D-STAG turns the (9)(d) into (9)(c) by moving the adverbial at the clause-initial position and considers the derivation tree of the normalized discourse. We do not that but directly encode the discourse (9)(d) by constructing the term $\mathrm{t}_3^{\text{medial}}$, defined in Equation (3.23). The tree representation of $\mathrm{t}_3^{\text{medial}}$ is shown in Figure 3.15(a). By interpreting the term $\mathrm{t}_3^{\text{medial}}$ with the lexicon $\mathscr{L}_{\text{synt}}^{\text{TAG}} \circ \mathscr{L}_{\text{TAG}}^{\text{DSTAG}}$, we obtain the derived syntactic tree of the discourse depicted in Figure 3.15(b). As the syntactic tree indicates, the adverbial *then* appears inside the VP of the clause (clause-medial position).

$$\begin{aligned}&\mathrm{t}_3^{\text{medial}} = \mathsf{AnchorI}\ \mathrm{t}_{C_5}\ (\mathrm{D}_{\textbf{becuase}}\ \mathrm{I}_{\mathsf{DU_A}}\\ &(\mathrm{D}_{\textbf{then}}^{\text{medial}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ (\mathsf{AnchorS}\ \mathrm{t}_{C_7}\ \mathrm{I}_{\mathsf{DU_A}}))\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ (\mathsf{AnchorS}\ \mathrm{t}_{C_6}\ \mathrm{I}_{\mathsf{DU_A}})\end{aligned} \tag{3.23}$$



(a) The abstract term a discourse with a clause-medial adverbial



(b) The derived syntactic tree

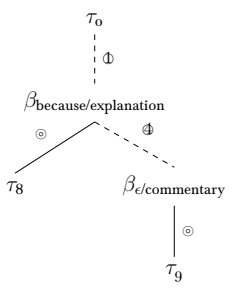
Figure 3.15: The ACG encoding of a discourse with a clause-medial adverbial

Example 3.5.

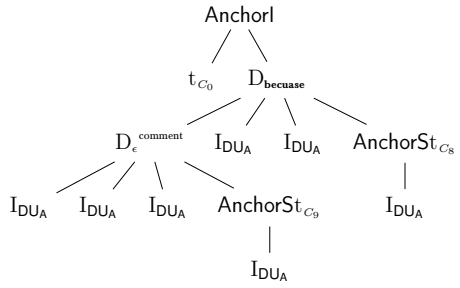
(9)(e) [Fred is grumpy]$_0$ because [his wife is away this week]$_8$. [This shows how much he loves her]$_9$.

Figure 3.16(a) illustrates the D-STAG derivation tree of (9)(e). We encode it in the abstract language with the term t_4, defined in Equation (3.24), whose tree representation is shown in Figure 3.16(b). By translating the term t_4 with the lexicon $\mathscr{L}^{\mathrm{TAG}}_{\mathrm{synt}} \circ \mathscr{L}^{\mathrm{DSTAG}}_{\mathrm{TAG}}$, we obtain the derived tree of the discourse depicted in Figure 3.16(c).

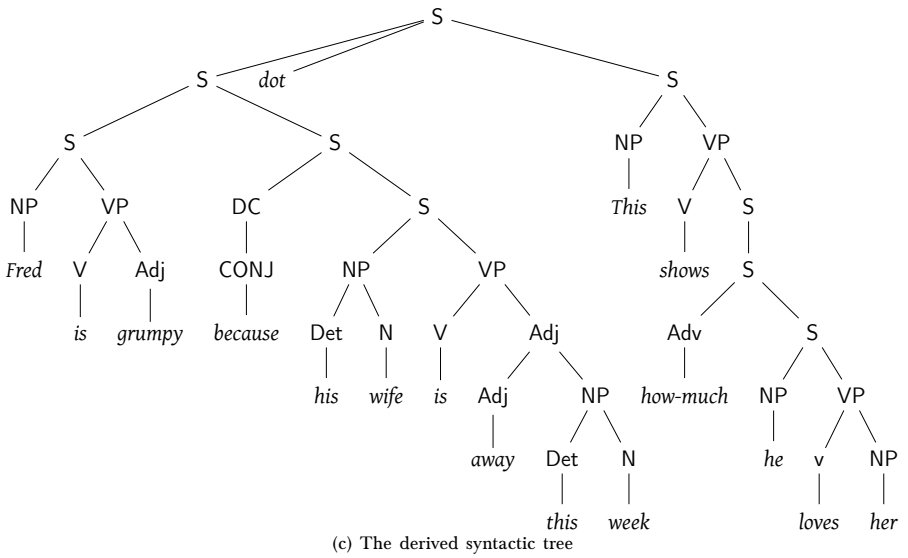
$$\begin{aligned}&\mathrm{t}_4 = \mathsf{AnchorI}\ \mathrm{t}_{C_0}\\ &(\mathrm{D}_{\textbf{becuase}}\ (\mathrm{D}_{\epsilon}{}^{\mathrm{comment}}\ \mathsf{I_{DU_A}}\ \mathsf{I_{DU_A}}\ \mathsf{I_{DU_A}}\ (\mathsf{AnchorS}\ \mathrm{t}_{C_9}\ \mathsf{I_{DU_A}}))\ \mathsf{I_{DU_A}}\ \mathsf{I_{DU_A}}\ (\mathsf{AnchorS}\ \mathrm{t}_{C_8}\ \mathsf{I_{DU_A}})\end{aligned} \tag{3.24}$$



(a) D-STAG derivation tree



(b) The ACG encoding of the D-STAG derivation tree



(c) The derived syntactic tree

Figure 3.16: The D-STAG derivation tree, its ACG encoding, and the derived syntactic tree

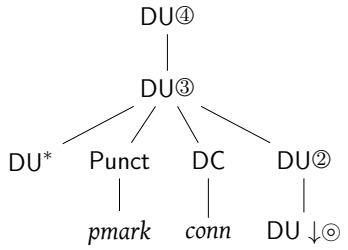
3.4 Encoding D-STAG Semantic Trees

In D-STAG, the derivation tree of a discourse specifies its semantic interpretation. In this section, we define an ACG to encode the correspondence between D-STAG derivation trees and their semantic interpretations.

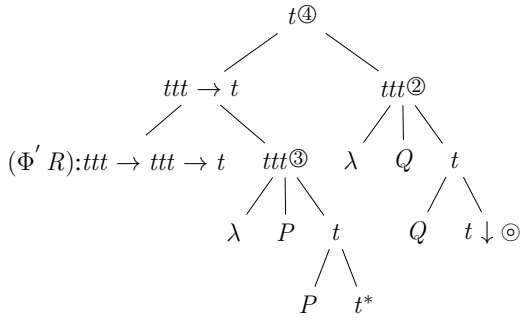
We already encoded the D-STAG derivation trees. To obtain the D-STAG semantic interpretations, it amounts interpreting D-STAG derivation trees as D-STAG semantic trees.

3.4.1 Extending the Abstract Vocabulary $\Sigma^{\text{Der}}_{\text{DSTAG}}$

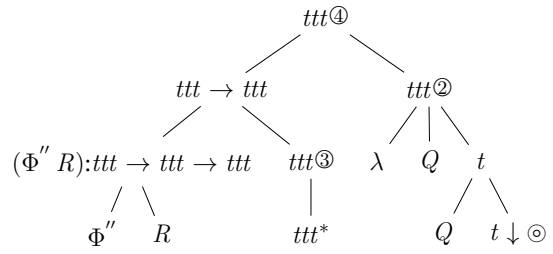
As we already saw in D-STAG,[84] an elementary tree anchored by a discourse connective *conn* is paired with two trees, the semantic trees A and B shown in Figure 3.17. The anchors of the trees A and B are $\Phi^{'} : R$ and $\Phi^{''} : R$ defined in Equation (3.25) and Equation (3.26) respectively, where R is a discourse relation signaled by the discourse connective *conn*.



(a) The syntactic tree anchored with a connective



(b) The D-STAG semantic tree A



(c) The D-STAG semantic tree B

Figure 3.17: D-STAG syntactic and semantic trees

$$\Phi^{'}\ R = \ \lambda X.\,\lambda Y.\,X\,(\lambda\, x.\,(Y\,(\lambda\, y.\,R\,x\,y)))\ :\ ttt \to ttt \to t \tag{3.25}$$

$$\Phi^{''}\ R = \ \lambda X.\,\lambda Y.\,\lambda P.\,X\,(\lambda\, x.\,(Y\,(\lambda\, y.\,(R\,x\,y) \wedge P(x))))\ :\ ttt \to ttt \to ttt \tag{3.26}$$

ttt is an abbreviation of $(t \to t) \to t$ and R is of type $t \to t \to t$.

[84] See Section 5.3 on page 174.

Thus, the constant $\mathrm{D}_{\mathbf{conn}}$ encoding an elementary tree anchored with the *conn* connective has to be paired with both the semantic tree A and semantic tree B. However, a lexicon cannot interpret $\mathrm{D}_{\mathbf{conn}}$ as the semantic tree A and the same time as the semantic tree B (since a lexicon is a function). The solution we propose is to introduce another version of the constant $\mathrm{D}_{\mathbf{conn}}$ the abstract vocabulary $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$. That is, instead of $\mathrm{D}_{\mathbf{conn}}$, we have two constants $\mathrm{D}_{\mathbf{conn}_{\mathrm{A}}}$ and $\mathrm{D}_{\mathbf{conn}_{\mathrm{B}}}$ in the abstract vocabulary $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$. We interpret both of them as the syntactic elementary tree anchored with the connective *conn* (see Figure 3.17(a)). The difference between the constants $\mathrm{D}_{\mathbf{conn}_{\mathrm{A}}}$ and $\mathrm{D}_{\mathbf{conn}_{\mathrm{B}}}$ is that we interpret $\mathrm{D}_{\mathbf{conn}_{\mathrm{A}}}$ as the term encoding the semantic tree A (Figure 3.17(b)), whereas we interpret $\mathrm{D}_{\mathbf{conn}_{\mathrm{B}}}$ as the term encoding the semantic tree B (Figure 3.17(c)).

3.4.2 The Signature $\Sigma^{\mathsf{sem}}_{\mathsf{DSTAG}}$

We introduce the signature $\Sigma^{\mathsf{sem}}_{\mathsf{DSTAG}}$ in order to build terms encoding D-STAG semantic interpretations. In D-STAG, the semantic interpretations are HOL terms.

To be able to build HOL terms encoding semantic interpretations, we introduce in $\Sigma^{\mathsf{sem}}_{\mathsf{DSTAG}}$ two atomic types, e and t. In order to interpret first order predicates of $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$, i.e., the clause-level grammar, we introduce the same predicates that one uses in the ACG encoding of TAG (see Section 3.8 on page 80).

To encode the discourse-level interpretations, we introduce in $\Sigma^{\mathsf{sem}}_{\mathsf{DSTAG}}$ constants encoding discourse relations **EXPLANATION**, **NARRATION**, etc. of type $t \to t \to t$. Table 3.9 shows the constants in $\Sigma^{\mathsf{sem}}_{\mathsf{DSTAG}}$.

Note that the types of the constants in $\Sigma^{\mathsf{sem}}_{\mathsf{DSTAG}}$ are non-linear. The is due to the fact that the D-STAG semantic terms are non-linear (but almost-linear). Indeed, the $\Phi''(R)$ term (see Equation (3.26) on page 273) is a non-linear one, which compels us to use non-linear types.

Constants in $\Sigma^{\mathsf{sem}}_{\mathsf{DSTAG}}$	Their types
fred, **paris**, **eiffel**, **louvre**, $\cdots$	e
grumpy, **sleep**, **empty**, **nightmare**, **fridge**, $\cdots$	$e \to e \to t$
love, **miss**, **fail**, **visit**, **go-to**, $\cdots$	$e \to e \to t$
badly, **clearly**, **a-lot**, **fortunately**, $\cdots$	$t \to t$
$\forall$, $\exists$, $\exists!$	$(e \to t) \to t$
EXPLANATION, **CONTINUATION**, **NARRATION**, **COMMENTARY**, **CONTRAST**, $\cdots$	$t \to t \to t$

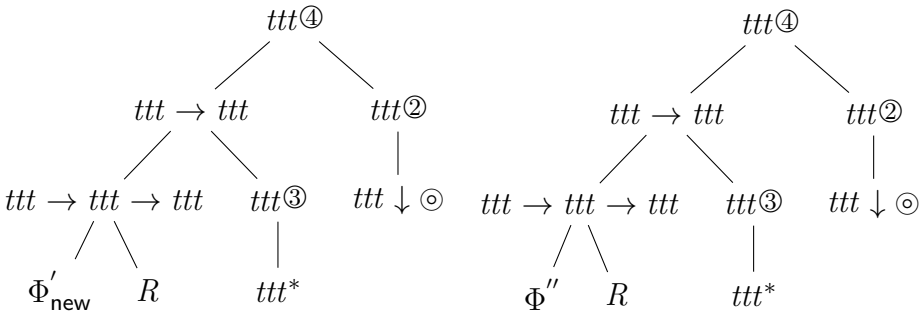
Table 3.9: Constants in the signature $\Sigma^{\mathsf{sem}}_{\mathsf{DSTAG}}$

3.4.3 Interpretations of Types

In D-STAG, an initial tree anchored by a C clause $\begin{smallmatrix}\mathsf{DU}^{\textcircled{\scriptsize I}}\\ |\\ C\end{smallmatrix}$ is paired with the t-rooted tree $\begin{smallmatrix}t^{\textcircled{\scriptsize I}}\\ |\\ F\end{smallmatrix}$, where F is the semantic representation of the clause C. The root node of

the syntactic tree $\mathsf{DU}①$ is paired with the root node of the semantic tree $t①$. Thus, one may interpret the type DU as t. On the other hand, as the semantic trees A and B indicate (see Figure 3.17), some DU nodes are also paired with the ttt nodes. Therefore, one may consider as well to interpret DU as ttt. Since one cannot interpret DU both as t and ttt, we have to choose to interpret DU either as t or as ttt. We choose to interpret DU as ttt. However, in the semantic tree A (see Figure 3.17(b)), we have the rootnode $t④$ (and therefore the footnode t^*) paired with a DU node. This makes us to modify the semantic tree A. We propose the new semantic tree A illustrated in Figure 3.18(a), where each of the nodes is ttt.

Since the only type we use in new semantic tree A is ttt, we cannot use the original $\Phi^{'}$ any more as its type is $(t \to t \to t) \to ttt \to ttt \to t$. In Equation (3.27), we define $\Phi^{'}_{\mathsf{new}}$ of type $(t \to t \to t) \to ttt \to ttt \to ttt$ that we use in the new semantic tree A instead of $\Phi^{'}$. The difference between the original $\Phi^{'}$ and $\Phi^{'}_{\mathsf{new}}$ is that, with the help of $\Phi^{'}$, we obtain formulas such as $R\,x\,y$ of type t, whereas by using $\Phi^{'}_{\mathsf{new}}$ instead of $\Phi^{'}$, we obtain $\lambda\,P.\,P\,(R\,x\,y)$ of type ttt, which is a type-raised version of $R\,x\,y$.



(a) The new semantic tree A (b) The Semantic tree B

Figure 3.18: semantic trees A and B

$$\Phi^{'}_{\mathsf{new}} = \lambda\,R\,.\,\lambda\,X\;Y\;P\,.\,X\;(\,\lambda\,x.Y(\,\lambda\,y\,.\,P\,(R\;x\;y))) : (t \to t \to t) \to ttt \to ttt \to ttt \quad (3.27)$$

Remark 3.3. *There is one more option for encoding semantic interpretations of the terms modeling D-STAG derivation trees. By introducing a new type* DU^2 *in* Σ_{GTAG}*, we can interpret* DU^2 *as* ttt*, and* DU *as* t*. However, in that case, the constants* D_{conn_A} *and* D_{conn_B} *would be of different types. The type of the constant* D_{conn_A} *would become* $\mathsf{DU}_A \multimap {\mathsf{DU}_A}^2 \multimap {\mathsf{DU}_A}^2 \multimap \mathsf{DU}_A$ *and the type of* D_{conn_B} *would become* ${\mathsf{DU}_A}^2 \multimap {\mathsf{DU}_A}^2 \multimap {\mathsf{DU}_A}^2 \multimap {\mathsf{DU}_A}^2$*. Since we prefer to have a uniform modeling of the discourse connectives in the abstract vocabulary, we do not develop this approach.*

Since the interpretation of DU is ttt, we interpret DU_A as $ttt \to ttt$. Thus, we pair the tree $\overset{\mathsf{DU}①}{|}\;{\scriptstyle C}$ with $\overset{ttt①}{|}\;{\scriptstyle \lambda P.PF}$, where the term $\lambda Q.QF$ of type ttt is a typed-raised version of F. Notice that the type-raising of F to $\lambda P.PF$ is what one does in D-STAG

as well. Indeed, as the original semantic tree A (see Figure 3.17(b)) and the semantic tree B (see Figure 3.17(c)) illustrate, the substitution site undergoes the type-raising, that is, if F is substituted in these trees, then it will be type-raised as $\lambda Q.QF$. Hence, we directly substitute in these trees a type-raised version of F, instead of first substituting it and then type-raising it.

The D-STAG interpretation of the *completed* discourse is of type t. To obtain a term of type t out of a term of type $ttt = (t \to t) \to t$, we apply the term of type ttt to the identity function $\lambda x.x$ of type $t \to t$. In the abstract language, a completed discourse is modeled by a term of type T. Therefore, we interpret the type T to the type t. In addition, we translate the type S from $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ to the ttt type.[85]

The rest of the types are the ones that we adopted from $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$. They are interpreted similar to the way they are interpreted in the ACG encoding of TAG but with the difference that we use non-linear types. Table 3.10 illustrates interpretations of types.

Types in $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$	Their interpretations by $\mathscr{L}^{\mathsf{SEM}}_{\mathsf{DSTAG}}$
DU, S	ttt
$\mathsf{DU_A}$	$ttt \to ttt$
T	t
np	$(e \to t) \to t$
$\mathsf{S_A}$	$t \to t$
$\mathsf{V_A}$	$t \to t$

Table 3.10: Semantic interpretations of the abstract types

3.4.4 Interpretations of Constants

3.4.4.1 Discourse Connectives

In the abstract signature $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$, we have two constants modeling the *conn* connective, $\mathrm{D}_{\mathbf{conn}_\mathrm{A}}$ and $\mathrm{D}_{\mathbf{conn}_\mathrm{B}}$. They stand for the D-STAG elementary tree of anchored with the *conn* discourse connective at the clause-initial position. Together with them, $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ contains two constants, $\mathrm{D}^{\mathrm{medial}}_{\mathbf{conn}_\mathrm{A}}$ and $\mathrm{D}^{\mathrm{medial}}_{\mathbf{conn}_\mathrm{B}}$, which model the cases where *conn* occupies the clause-medial positions. Hence, the constants $\mathrm{D}_{\mathbf{conn}_\mathrm{A}}$ and $\mathrm{D}^{\mathrm{medial}}_{\mathbf{conn}_\mathrm{A}}$ model the cases where one uses the semantic tree A as a semantic tree encoding *conn*, while the constants $\mathrm{D}_{\mathbf{conn}_\mathrm{B}}$ and $\mathrm{D}^{\mathrm{medial}}_{\mathbf{conn}_\mathrm{B}}$ model the cases where one uses the semantic tree B as a semantic tree encoding *conn*. Thus, there is no *semantic difference* between $\mathrm{D}_{\mathbf{conn}_\mathrm{A}}$ and $\mathrm{D}^{\mathrm{medial}}_{\mathbf{conn}_\mathrm{A}}$, nor between $\mathrm{D}_{\mathbf{conn}_\mathrm{B}}$ and $\mathrm{D}^{\mathrm{medial}}_{\mathbf{conn}_\mathrm{B}}$. Consequently, we translate the constant $\mathrm{D}^{\mathrm{medial}}_{\mathbf{conn}_\mathrm{A}}$ (resp. $\mathrm{D}^{\mathrm{medial}}_{\mathbf{conn}_\mathrm{B}}$) to the same term to which the constant $\mathrm{D}_{\mathbf{conn}_\mathrm{A}}$ (resp. $\mathrm{D}_{\mathbf{conn}_\mathrm{B}}$) translates.

The type of the constants $\mathrm{D}_{\mathbf{conn}_\mathrm{A}}$ and $\mathrm{D}_{\mathbf{conn}_\mathrm{B}}$ is $\mathsf{DU_A} \multimap \mathsf{DU_A} \multimap \mathsf{DU_A} \multimap \mathsf{DU} \multimap \mathsf{DU_A}$, where $\mathsf{DU_A}$ translates to $ttt \to ttt$ and DU to ttt. In order to interpret the constants $\mathrm{D}_{\mathbf{conn}_\mathrm{A}}$ and $\mathrm{D}_{\mathbf{conn}_\mathrm{B}}$, we encode the semantic tree A and the semantic tree B (see Figure 3.18 on the previous page) as terms over $\Sigma^{\mathsf{sem}}_{\mathsf{DSTAG}}$. Table 3.11 shows the results we obtain.

[85] We could translate S to t instead of ttt. However, we translate it to ttt because it does not create any issues since one can transform a term of type ttt into a term of type t by applying it to the term $\lambda x.x \ : t \to t$.

Constants in Σ_{GTAG}	Their translations by $\mathscr{L}^{\text{SEM}}_{\text{DSTAG}} : \Sigma^{\text{Der}}_{\text{DSTAG}} \longrightarrow \Sigma^{\text{sem}}_{\text{DSTAG}}$
$\text{D}_{\textbf{conn}_\text{A}}, \text{D}^{\text{medial}}_{\textbf{conn}_\text{A}}$	$\lambda\, d_4\, d_3\, d_2.\, \lambda\, d_{subst}.\lambda\, d_{foot}.\; d_4\, ((\Phi^{'}_{\textsf{new}}\, \textsc{Rconn})\, (d_3\, d_{foot})\, (d_2\, d_{subst}))$
$\text{D}_{\textbf{conn}_\text{B}}, \text{D}^{\text{medial}}_{\textbf{conn}_\text{B}}$	$\lambda\, d_4\, d_3\, d_2.\, \lambda\, d_{subst}.\, \lambda\, d_{foot}.\; d_4\, ((\Phi^{''}\, \textsc{Rconn})\, (d_3\, d_{foot})\, (d_2\, d_{subst}))$

Table 3.11: Semantic interpretations of the constants encoding discourse connectives

In Table 3.12, we provide semantic interpretations of the constants $\mathsf{AnchorS} : \mathsf{S} \multimap \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}$ and $\mathsf{AnchorI} : \mathsf{S} \multimap \mathsf{DU}_\mathsf{A} \multimap \mathsf{T}$.

Constants in $\Sigma^{\text{Der}}_{\text{DSTAG}}$	Their translations by $\mathscr{L}^{\text{SEM}}_{\text{DSTAG}}$
AnchorS	$\lambda\, s\; mod.\, \lambda P.\; mod\, (\lambda\, Q.\, Q\, (s\, (\lambda\, x.\, x)))\, P$
AnchorI	$\lambda\, s\; mod.\; mod\, (\lambda\, Q.\, Q\;\; (s(\lambda\, x.x)))\, (\lambda\, x.x)$

Table 3.12: Semantic interpretations of the constants AnchorI and AnchorS

Remark 3.4. *Sometimes we may write* D_{conn}*, but we mean two different constants* D_{conn_A} *and* D_{conn_B}*. The point is that the difference between the* D_{conn_A} *and* D_{conn_B} *constants is in their semantic interpretations. Therefore, in a case where we are not concerned with the semantic interpretations, we write* D_{conn}*.*

3.4.4.2 First Order Predicates

For interpreting the first order predicates of $\Sigma^{\text{Der}}_{\text{DSTAG}}$ into semantics, i.e., the ones that we use for modeling derivation trees of clauses, we cannot use the semantic translations provided in the ACG encoding of TAG. Indeed, in $\Sigma^{\text{Der}}_{\text{DSTAG}}$, an initial tree anchored with a verb (verb phrase, predicative adjective etc.) is modeled with two VP adjunction sites, whereas in $\Sigma^{\text{Der}}_{\text{TAG}}$, it has a single VP-adjunction site. Let us consider one of these constants, $\text{D}_{\textbf{walks}}$. It has two arguments encoding VP-adjunction sites. The semantic interpretations of each of these two arguments should scope over the predicate expressed by *walks*. In particular, we interpret $\text{D}_{\textbf{walks}}$ as follows:

$$\begin{aligned}&\mathscr{L}^{\text{SEM}}_{\text{DSTAG}}(\text{D}_{\textbf{walks}}) = \\ &\qquad \lambda\, sa\; va_1\; va_2\; subj.\, \lambda mod.\; sa\, (subj\, (\lambda x.\; va_1\, (mod\, (va_2\, (\textbf{walk}\; x))))) : \\ &\qquad\qquad (t \to t) \to (t \to t) \to (t \to t) \to \mathsf{qnp} \to \underbrace{(t \to t) \to t}_{ttt}\end{aligned}$$

Where qnp abbreviates $(e \to t) \to t$, which is the interpretation of the type np (3.28)

In Equation (3.28), the variables va_1 and va_2 scope over (**walks** x). Thus, we interpret $\mathsf{V_A}$ as $t \to t$. In Equation (3.28), we use the abstraction over the variable mod in order to type-raise t to ttt.

Similarly to the translation of the $\mathrm{D}_{\mathbf{walks}}$ constant, we translate the other abstract constants of $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ encoding the initial trees anchored with verbs, verb phrases, predicative adjectives etc.

We interpret the rest of the first order predicates (the ones we adopted from $\Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$) in the same way as it is done in the ACG encoding of TAG with the difference that we use almost-linear types instead of linear ones.

Remark 3.5. *The ACG encoding of D-STAG and the original D-STAG approach show some differences. One of the main differences is that we model clause-medial connectives as part of the grammar, whereas in D-STAG, in order to analyze a discourse containing them, one uses a prepossessing step. Another difference is that our encoding of D-STAG makes the semantic trees anchored with a discourse relation more similar to each other than their D-STAG counterparts are.*

3.5 The Examples of Semantic Interpretations

In order to obtain the semantic interpretation of a discourse, we translate a term over $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ encoding its derivation tree under the lexicon $\mathscr{L}^{\mathsf{SEM}}_{\mathsf{DSTAG}}$. We list the ACG signatures and commands in Appendix D.1 that are used in order to obtain the results given below.

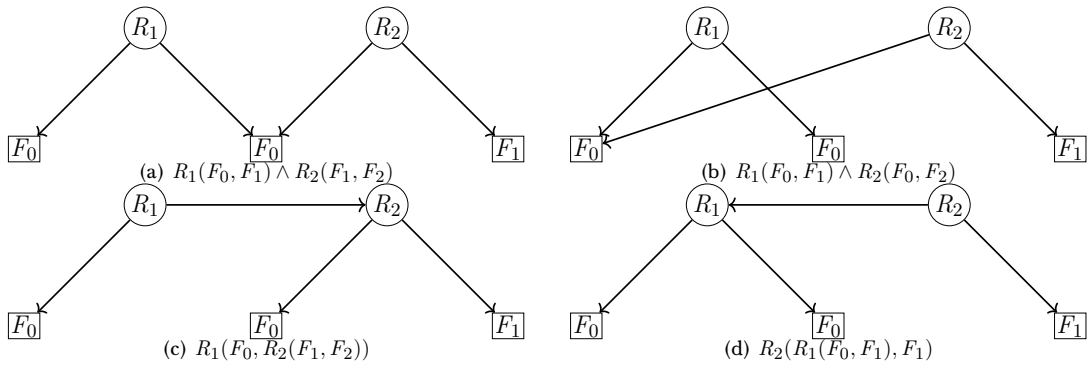


Figure 3.19: semantic interpretations of discourses

We use the same examples of discourses as the ones we used before (see Section 3.3.8 on page 265).

(29) a. [Fred is grumpy]$_0$ because [he lost his keys]$_1$. Moreover, [he failed his exam]$_2$.

b. [Fred is grumpy]$_0$ because [he didn't sleep well]$_3$. [He had nightmares]$_4$.

c. [Fred went to the supermarket]$_5$ because [his fridge was empty]$_6$. Then, [he went to the movies]$_7$.

d. [Fred went to the supermarket]$_5$ because [his fridge was empty]$_6$. [He *then* went

to the movies]$_{7^m}$.

e. [Fred is grumpy]$_0$ because [his wife is away this week]$_8$. [This shows how much he loves her]$_9$.

The D-STAG discourse structures of the examples (29)(a)-(29)(e) are depicted in Figure 3.19.

Example 3.6.

(29)(a) [Fred is grumpy]$_0$ because [he lost his keys]$_1$. Moreover, [he failed his exam]$_2$.

Interpretation: EXPLANATION F_0 (CONTINUATION F_1 F_2)

In order to interpret the discourse (29)(a), one uses two semantic trees A.

- The first semantic tree A is anchored with EXPLANATION. This tree is paired with the syntactic tree anchored with *because*.
- The second semantic tree A is anchored with CONTINUATION. This tree is paired with the syntactic tree anchored with *moreover*.

Figure 3.20(a) shows the D-STAG derivation tree.[86] We encode it as the term t_1 defined in Equation (3.30), whose tree representation is shown in Figure 3.20(b).

$$\begin{array}{l}\mathrm{t}_1 = \mathsf{AnchorI}\ \mathrm{t}_{C_0} \\ (\mathrm{D}_{\mathbf{because}_\mathrm{A}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ (\mathsf{AnchorS}\ \mathrm{t}_{C_0}\ (\mathrm{D}_{\mathbf{moreover}_\mathrm{A}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ (\mathsf{AnchorS}\ \mathrm{t}_{C_1}\ \mathrm{I}_{\mathsf{DU_A}})))) : \mathsf{T}\end{array} \tag{3.30}$$

In order to obtain the semantic interpretation of the discourse, we translate the term t_1 under the lexicon $\mathscr{L}^{\mathsf{SEM}}_{\mathsf{DSTAG}}$. We obtain the following semantic interpretation:

$$\begin{array}{rl}\mathscr{L}^{\mathsf{SEM}}_{\mathsf{DSTAG}}(\mathrm{t}_1) = & (\mathbf{EXPLANATION} \\ & \quad (\mathbf{grumpy\ fred}) \\ & \quad (\mathbf{CONTINUATION} \\ & \quad\quad (\exists!\, x.\, (\mathbf{key}\ x)\ \wedge\ (\mathbf{lose\ fred}\ x)) \\ & \quad\quad (\exists!\, y.\, (\mathbf{exam}\ y)\ \wedge\ (\mathbf{fail\ fred}\ y)) \\ & \quad) \\ & : t\end{array} \tag{3.31}$$

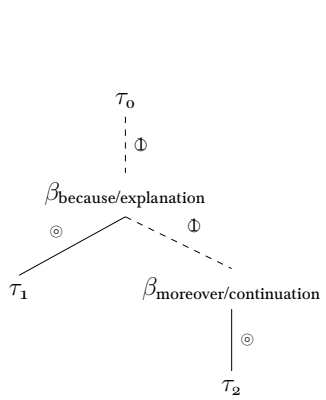
Example 3.7.

(29)(b) [Fred is grumpy]$_0$ because [he didn't sleep well]$_3$. [He had nightmares]$_4$.

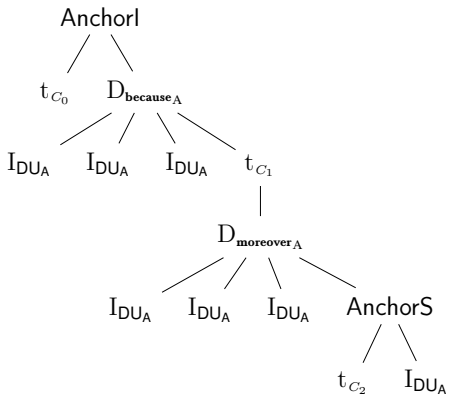
Interpretation: (EXPLANATION F_0 F_3) $\wedge$ (EXPLANATION F_3 F_4)

Figure 3.21(a) illustrates the derivation tree of the (29)(b) discourse. To interpret it, D-STAG uses one semantic tree A and one semantic tree B.

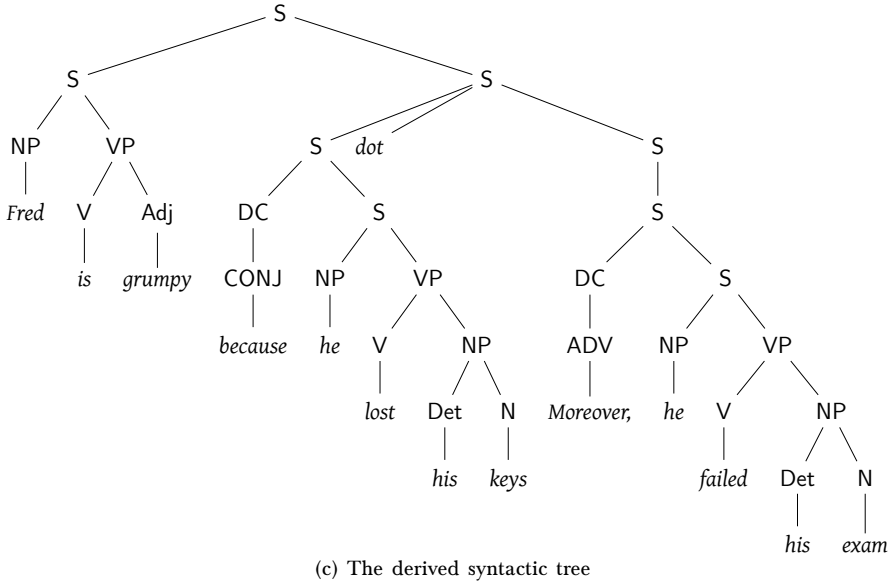
[86] The terms encoding the derivation trees of the clauses used in these examples can be found on page 266, Equations (3.10)-(3.19).



(a) The D-STAG derivation tree



(b) The ACG encoding of the D-STAG derivation tree



(c) The derived syntactic tree

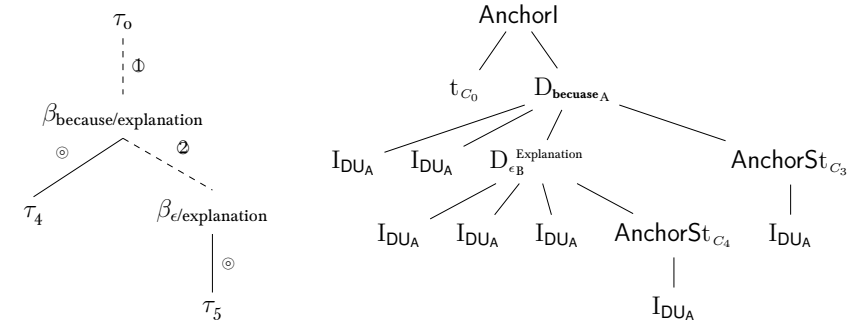
Figure 3.20: The D-STAG derivation tree, its ACG encoding, and the derived syntactic tree

- The semantic tree A is anchored with EXPLANATION. This tree is paired with the syntactic tree anchored with ***because***.
- The semantic tree B is anchored with EXPLANATION. This tree is paired with the syntactic tree anchored with the ***empty*** connective.

We encode the derivation tree of the discourse as the term t_2 defined in Equation (3.32), whose tree representation is provided in Figure 3.21(b).

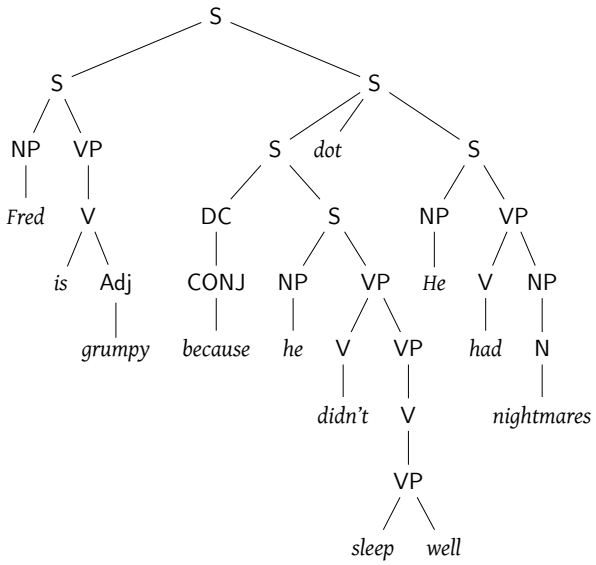
$$t_2 = \mathsf{AnchorI}\ t_{C_0}\ (D_{\textbf{becuase}_A}\ I_{DU_A}\ I_{DU_A}\ (D_{\epsilon_B}^{\text{Explanation}}\ I_{DU_A}\ I_{DU_A}\ I_{DU_A}\ (\mathsf{AnchorS}\ t_{C_4}\ I_{DU_A}))(\mathsf{AnchorS}\ t_{C_3}\ I_{DU_A})) : \mathsf{T} \tag{3.32}$$

In order to obtain the semantic interpretation of (29)(b), we translate the term t_2



(a) The D-STAG derivation tree

(b) The ACG encoding of the D-STAG derivation tree



(c) The derived syntactic tree

Figure 3.21: The D-STAG derivation tree, its ACG encoding, and the derived syntactic tree

by the lexicon $\mathscr{L}_{\text{DSTAG}}^{\text{SEM}}$. We obtain the following interpretation:

$$\begin{aligned}\mathscr{L}_{\text{DSTAG}}^{\text{SEM}}(\mathrm{t}_2) = \quad & (\textbf{EXPLANATION} \\ & \quad (\textbf{grumpy fred}) \\ & \quad (\neg(\textbf{sleep fred})) \\ &) \\ & \wedge \\ & (\textbf{EXPLANATION} \\ & \quad (\neg(\textbf{sleep fred})) \\ & \quad (\textbf{Plur}\ (\lambda x.\ \textbf{nightmare}\ x)\ (\lambda y.\textbf{have fred}\ y)) \\ &) \\ & : t\end{aligned} \tag{3.33}$$

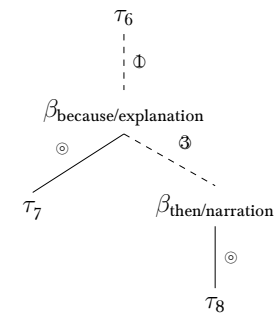
Example 3.8.

(29)(c) [Fred went to the supermarket]$_5$ because [his fridge was empty]$_6$. Then, [he went to the movies]$_7$.

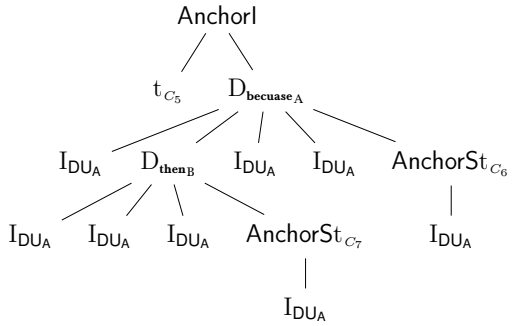
Interpretation: $(\textsc{Explanation}\ F_5\ F_6) \wedge (\textsc{Narration}\ F_5\ F_7)$

Figure 3.22(a) illustrates the derivation tree of the (29)(c) discourse. In D-STAG, in order to interpret it, D-STAG uses the semantic tree A and the semantic tree B.

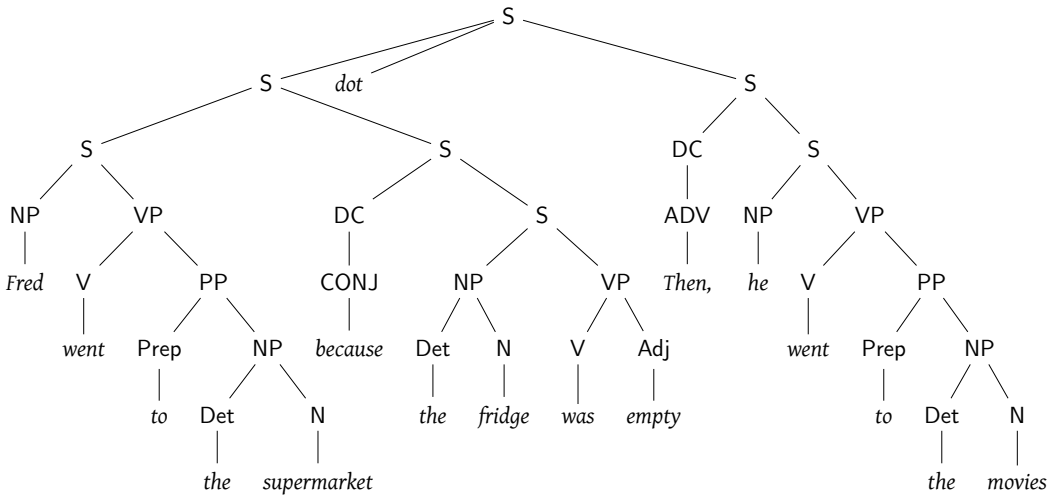
- The semantic tree A is anchored with Explanation, which is the semantic tree paired with the elementary tree anchored with *because*.
- The semantic tree B is anchored with Narration, which is the semantic tree paired with the elementary tree anchored with *then*.



(a) The D-STAG derivation tree



(b) The ACG encoding of the D-STAG derivation tree



(c) The derived syntactic tree

Figure 3.22: The D-STAG derivation tree, its ACG encoding, and the derived syntactic tree

We encode the derivation tree of the discourse with the term t_3 defined in Equation (3.34), whose tree representation is shown in Figure 3.22(b).

$$\begin{aligned} t_3 = {} & \mathsf{AnchorI}\ t_{C_5}\ (D_{\textbf{becuase}_A}\ I_{DU_A} \\ & (D_{\textbf{then}_B}\ I_{DU_A}\ I_{DU_A}\ I_{DU_A}\ (\mathsf{AnchorS}\ t_{C_7}\ I_{DU_A}))\ I_{DU_A}\ I_{DU_A}\ (\mathsf{AnchorS}\ t_{C_6}\ I_{DU_A}) \end{aligned} \tag{3.34}$$

In order to obtain the semantic interpretation of the discourse, we translate the term t_3 by the lexicon $\mathscr{L}_{\mathsf{DSTAG}}^{\mathsf{SEM}}$. We obtain the following semantic interpretation:

$$\begin{aligned}\mathscr{L}_{\mathsf{DSTAG}}^{\mathsf{SEM}}(\mathrm{t}_3) = \; & (\textbf{EXPLANATION} \\ & \quad (\exists!\, x.\ (\textbf{supermarket}\ x)\ \wedge\ (\textbf{go-to fred}\ x)) \\ & \quad (\exists!\, x.\ (\textbf{fridge}\ x)\ \wedge\ (\textbf{empty}\ x))) \\ & \wedge \\ & (\textbf{NARRATION} \\ & \quad (\exists!\, x.(\textbf{supermarket}\ x)\ \wedge\ (\textbf{go-to fred}\ x)) \\ & \quad (\exists!\, x.\ (\textbf{movies}\ x)\ \wedge\ (\textbf{go-to fred}\ x))) \\ & : t\end{aligned} \tag{3.35}$$

Example 3.9.

(29)(d) [Fred went to the supermarket]$_5$ because [his fridge was empty]$_6$. [he *then* went to the movies]$_{7^m}$.

Interpretation: (EXPLANATION $F_5\ F_6$) $\wedge$ (NARRATION $F_5\ F_7$)

In this case, we build the term $\mathrm{t}_3{}^{\mathrm{medial}}$, defined in Equation (3.36), whose tree representation is shown in Figure 3.23(a).

$$\begin{aligned}&\mathrm{t}_3{}^{\mathrm{medial}} = \mathsf{AnchorI}\ \mathrm{t}_{C_5}\ (\mathrm{D}_{\textbf{becuase}_\mathrm{A}}\ \mathrm{I}_{\mathsf{DU_A}} \\ &(\mathrm{D}_{\textbf{then}_\mathrm{B}}{}^{\mathrm{medial}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ (\mathsf{AnchorS}\ \mathrm{t}_{C_7}\ \mathrm{I}_{\mathsf{DU_A}}))\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ (\mathsf{AnchorS}\ \mathrm{t}_{C_6}\ \mathrm{I}_{\mathsf{DU_A}})\end{aligned} \tag{3.36}$$

In order to obtain the semantic interpretation of the (29)(d) discourse, we translate the term $\mathrm{t}_3{}^{\mathrm{medial}}$ by the lexicon $\mathscr{L}_{\mathsf{DSTAG}}^{\mathsf{SEM}}$. As a result, we obtain the interpretation (3.37), which (as it was expected) coincides with the interpretation of the discourse (29)(c).

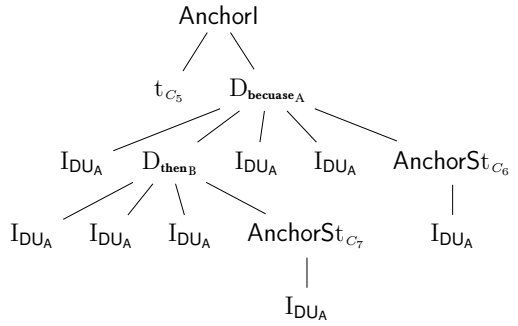
$$\begin{aligned}\mathscr{L}_{\mathsf{DSTAG}}^{\mathsf{SEM}}(\mathrm{t}_3{}^{\mathrm{medial}}) = \; & (\textbf{EXPLANATION} \\ & \quad (\exists!\, x.\ (\textbf{supermarket}\ x)\ \wedge\ (\textbf{go-to fred}\ x)) \\ & \quad (\exists!\, x.\ (\textbf{fridge}\ x)\ \wedge\ (\textbf{empty}\ x))) \\ & \wedge \\ & (\textbf{NARRATION} \\ & \quad (\exists!\, x.(\textbf{supermarket}\ x)\ \wedge\ (\textbf{go-to fred}\ x)) \\ & \quad (\exists!\, x.\ (\textbf{movies}\ x)\ \wedge\ (\textbf{go-to fred}\ x))) \\ & : t\end{aligned} \tag{3.37}$$

Example 3.10.

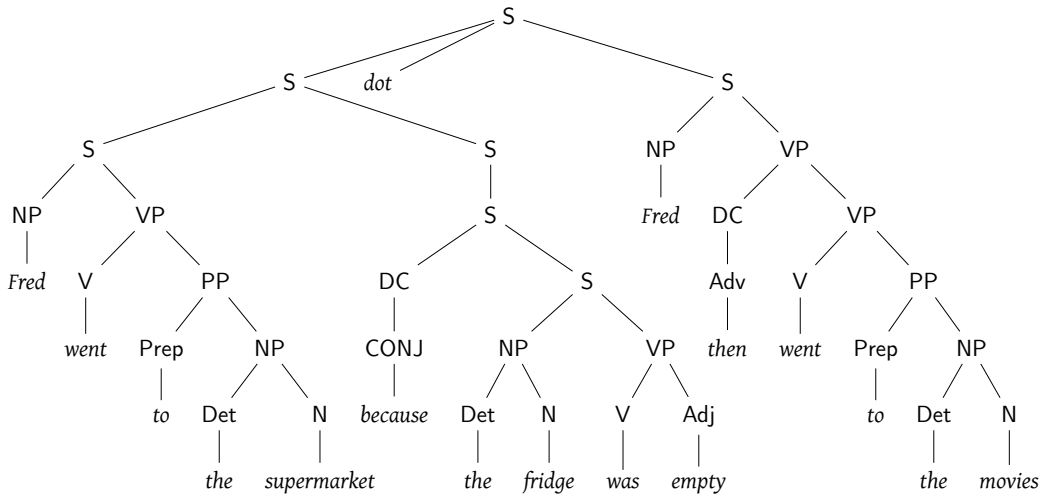
(29)(e) [Fred is grumpy]$_0$ because [his wife is away this week]$_8$. [This shows how much he loves her]$_9$.

Interpretation: COMMENTARY (EXPLANATION $F_0\ F_8$) F_9

In order to interpret the (29)(e) discourse, D-STAG uses two semantic trees A. One of them is anchored with COMMENTARY and the other one is anchored with



(a) The term encoding the derivation tree of a discourse with a clause-medial adverbial



(b) The derived syntactic tree

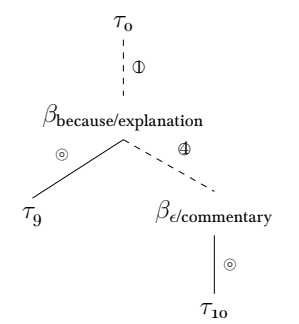
Figure 3.23: The abstract term encoding of a discourse with a clause-medial adverbial and the derived syntactic tree

EXPLANATION. Figure 3.24(b) illustrates the derivation tree of (29)(e). We encode this derivation tree as the term t_4 defined in Equation (3.38).

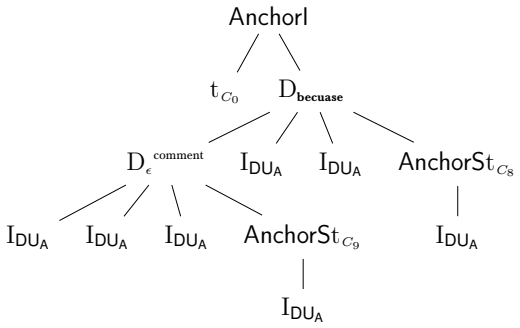
$$\begin{aligned} &t_4 = \mathsf{AnchorI}\ t_{C_0} \\ &(D_{\textbf{becuase}_A}\ (D_{\epsilon_A}^{\text{commentary}}\ I_{DU_A}\ I_{DU_A}\ I_{DU_A}\ (\mathsf{AnchorS}\ t_{C_9}\ I_{DU_A}))\ I_{DU_A}\ I_{DU_A}\ (\mathsf{AnchorS}\ t_{C_8}\ I_{DU_A}) \end{aligned} \tag{3.38}$$

In order to obtain the semantic interpretation of the discourse, we translate the term t_4 by the lexicon $\mathscr{L}_{\text{DSTAG}}^{\text{SEM}}$. We obtain the following interpretation of the discourse:

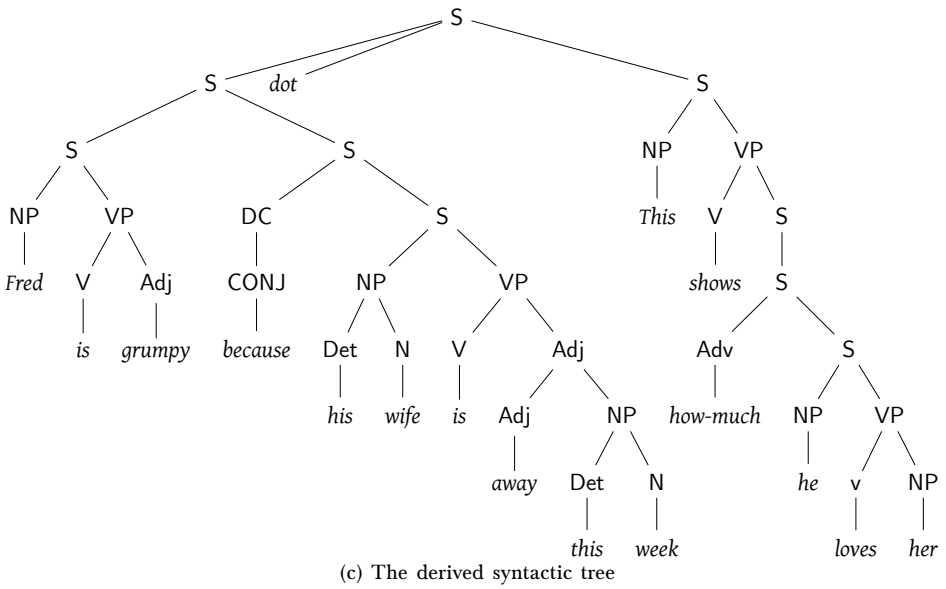
$$\begin{aligned} \mathscr{L}_{\text{DSTAG}}^{\text{SEM}}(t_4) = \ & \textbf{COMMENTARY} \\ & (\textbf{EXPLANATION} \\ & \quad (\textbf{grumpy fred}) \\ & \quad (\exists!\, x.\, (\textbf{wife}\ x\ \textbf{fred}) \wedge (\exists!\, y.\, (\textbf{week}\ y) \wedge (\textbf{away}\ x\ y))) \\ &) \\ & (\textbf{show-this}\ (\textbf{a-lot}\ (\exists!\, x.\, (\textbf{wife}\ x\ \textbf{fred}) \wedge (\textbf{love fred}\ x)))) \\ & : t \end{aligned} \tag{3.39}$$



(a) The D-STAG derivation tree



(b) The ACG encoding of the D-STAG derivation tree



(c) The derived syntactic tree

Figure 3.24: The D-STAG derivation tree, its ACG encoding, and the derived syntactic tree

3.6 Interpretation as Labeled Formulas

In D-STAG, the semantic interpretations follow the SDRT principles. As we saw in the Section 4.3, SDRT makes use of labels in order to encode the discourse interpretations. We develop an approach that enables us to obtain the interpretation of a discourse reminiscent of the SDRT interpretation of that discourse. Labeled formulas enable us to refer to sub-formulas of a formula. By labeling the expressions that are under the scope of the quantifiers, we can encode interaction between the scope of a discourse relation and the existential quantifier, which is not the case for the HOL interpretations that we defined in the previous section.

Thus, we encode the semantic interpretation of a discourse as a labeled formula, instead of a HOL one. The labeled interpretations specify the same DAGs as the ones specified by the (original) D-STAG interpretations. For the sake of illustratation, let us consider the discourse (29)(e), repeated as follows:

(29)(e), repeated
[Fred is grumpy]$_0$ because [his wife is away this week]$_8$. [This shows how much he loves her]$_9$.

Interpretation: COMMENTARY (EXPLANATION F_0 F_8) F_9

The unlabeled interpretation of (29)(e) is shown in Equation (3.39) on page 284. We define the following labeled interpretation of (29)(e):

$$
\begin{aligned}
u = \;& \exists_l\, l\; l_1\; l_2. \\
& \quad l : (\textbf{grumpy fred}) \wedge \\
& \quad (\exists_l\, l_3. \\
& \qquad (\exists!\, x.\; l_3 : (\textbf{wife}\; x\; \textbf{fred}) \wedge (\exists!\, x'.\; l_3 : (\textbf{week}\; x') \wedge l_3 : (\textbf{away}\; x\; x'))) \wedge \\
& \qquad (l_2 : (\textbf{EXPLANATION}\; l\; l_3) \wedge \\
& \qquad (\exists_l\, l_4.\; l_4 : (\textbf{show-this}(\exists_l\, l_5.\; \textbf{a-lot}\; (\exists! x.\; l_5 : (\textbf{wife}\; x\; \textbf{fred}) \wedge l_5 : (\textbf{love fred}\; x)))) \wedge \\
& \qquad (l_1 : (\textbf{COMMENTARY}\; l_2\; l_4) \wedge \top))) \\
& \quad) \\
& : t
\end{aligned}
\tag{3.40}
$$

The labeled formula u defined in Equation (3.40) contains the sub-formula $l_2 : (\textbf{EXPLANATION}\; l\; l_3)$, which makes explicit that the expressions labeled with l and l_3 are related with the **EXPLANATION** relation. Another sub-formula of u is $l_1 : (\textbf{COMMENTARY}\; l_2\; l_4)$, which encodes that l_2 and l_4 are in the **COMMENTARY** relation, where l_2 labels **EXPLANATION** $l\; l_3$. By contrast, in the unlabeled interpretations, the scope of a discourse predicate included all the material introduced in an interpretation of a clause. In other words, the approach with labels enables us to explicitly refer to the material that are related by the discourse relations.

With the help of labels we can capture the argument sharing between two discourse relations, for example as it is in the following discourse:

(29)(b), repeated [Fred is grumpy]$_0$ because [he didn't sleep well]$_3$. [He had nightmares]$_4$.
Interpretation: $(\textsc{Explanation}\ F_0\ F_3) \wedge (\textsc{Explanation}\ F_3\ F_4)$

As we already saw (see Equation 3.33), the unlabeled formula encoding the interpretation of (29)(b) is as follows:

$$
\begin{aligned}
\mathscr{L}^{\text{SEM}}_{\text{DSTAG}}(\mathrm{t}_2) = \ & (\textbf{EXPLANATION} \\
& \quad (\textbf{grumpy fred}) \\
& \quad (\neg(\textbf{sleep fred})) \\
&) \\
& \wedge \\
& (\textbf{EXPLANATION} \\
& \quad (\neg(\textbf{sleep fred})) \\
& \quad (\textbf{Plur}\ (\lambda x.\ \textbf{nightmare}\ x)\ (\lambda y.\textbf{have fred}\ y)) \\
&)\ : t
\end{aligned}
$$

$\mathscr{L}^{\text{SEM}}_{\text{DSTAG}}(\mathrm{t}_2)$ does not explicitly encode that $(\neg(\textbf{sleep fred}))$ is the interpretation of one and the same clause (*he didn't sleep well*). However, with the labeled formulas, we are able to do that. Namely, the labeled interpretation of (29)(b) is as follows:

$$
\begin{aligned}
& v = \\
& \exists_l\ l_0\ l_1. \\
& \quad (\textbf{grumpy fred}\ l)\ \& \\
& \quad (\exists_l\ l_2\ l_3. \\
& \qquad (l_3 :\ \neg(\textbf{sleep fred}\ ))\ \& \\
& \qquad (\exists_l\ l_4. \\
& \qquad\quad (l_4 :\ \textbf{PLUR}(\lambda\ x\ l_5.\ l_5 :\ (\textbf{nightmare}\ x))\ (\lambda\ y\ l_5.\ l_5 :\ (\textbf{have fred}\ y)))\ \& \\
& \qquad\quad ((l_1 : (\textbf{Explanation}\ l_0\ l_3)\ \&\ \top)\ \&\ l_2 :\ (\textbf{Explanation}\ l_3\ l_4)) \\
& \qquad) \\
& \quad) \\
& : t
\end{aligned}
$$

As v shows, the same label l_3, which labels the semantic interpretation of *he didn't sleep well*, is used twice (in (**Explanation** $l_0\ l_3$) and in (**Explanation** $l_3\ l_4$)).

3.6.1 A Signature $\Sigma^{\text{sem}}_{\text{LABEL}}$ For Encoding Labeled Semantic Representations

In order to build labeled formulas, we introduce the signature $\Sigma^{\text{sem}}_{\text{LABEL}}$. To encode labels, we introduce an atomic type ℓ in $\Sigma^{\text{sem}}_{\text{LABEL}}$. Besides ℓ, in $\Sigma^{\text{sem}}_{\text{LABEL}}$ we introduce the types e (for entities) and t (for truth values).

In D-STAG, where one uses unlabeled semantics to represent a meaning of a discourse, a clause is interpreted as a term of type t. The predicates in unlabeled semantics encoding discourse relations are of type $t \to t \to t$.

In the labeled language, instead of discourse units (expressions of type t), labels serve as arguments of predicates modeling discourse (rhetorical) relations (such as **EXPLANATION**). Predicates encoding discourse relations are of type $\ell \to \ell \to \ell \to t$. The type $\ell \to \ell \to \ell \to t$ encodes that one needs two discourse units (two arguments of type ℓ) in order to generate a new discourse unit (the third argument of type ℓ) by connecting them with a discourse relation. For example, the formula (**EXPLANATION** $l_1\ l_2\ l_3$) : t can be read as follows: The content labeled by l_1 and l_2 are connected by the relation EXPLANATION. The label of the new discourse unit built by connecting $l_1\ l_2$ by EXPLANATION is l_3. Thus, (**EXPLANATION** $l_1\ l_2\ l_3$) can be seen as a statement (proposition) and therefore it is of type t.

In the signature $\Sigma^{\mathsf{sem}}_{\mathsf{LABEL}}$, we introduce the constants of the signature $\Sigma^{\mathsf{sem}}_{\mathsf{DSTAG}}$ with the modified types so that we can represent labeled terms. Table 3.13 shows the constants in $\Sigma^{\mathsf{sem}}_{\mathsf{LABEL}}$. Together with the constants of type $(e \to t) \to t$ encoding quantifiers, we introduce the constant $\exists_l$ of type $(\ell \to t) \to t$. We use the constant $\exists_l$ in order to *introduce* labels denoting terms.

fred, **he** : e	**EXPLANATION** : $\ell \to \ell \to \ell \to t$
sleep, **bad-mood**, **exam** : $e \to \ell \to t$	**CONTINUATION** : $\ell \to \ell \to \ell \to t$
love, **miss**, **fail** : $e \to e \to \ell \to t$	**NARRATION** : $\ell \to \ell \to \ell \to t$
$\forall, \exists, \exists! : (e \to t) \to t$	$\exists_l : (\ell \to t) \to t$
...	...

Table 3.13: Constants in $\Sigma^{\mathsf{sem}}_{\mathsf{LABEL}}$

Convention: Instead of $P\,k_1 \ldots k_n\,l$ we write $l\ :\ P\,k_1 \ldots k_n$, where l is a label. For instance, we write l_3 : **NARRATION** $l_1\ l_2$ instead of **NARRATION** $l_1\ l_2\ l_3$.

Remark 3.6. *(Asher and Pogodalla, 2011) encodes the SDRT interpretations of discourses using a dynamic semantics approach from (de Groote, 2006). Their encoding of SDRT makes use of contexts. A context accumulates labels. One selects labels from a context in order to build a new discourse unit. In our case, the constant $\exists_l$ is the only 'tool' that we use for introducing labels in an interpretation of a discourse, because, in the present work, we do not encode a notion of a context.*

3.6.2 Interpretations as Types and Terms Built Upon $\Sigma^{\mathsf{sem}}_{\mathsf{LABEL}}$

We define the lexicon $\mathscr{L}^{\mathsf{SEM}}_{\mathsf{LABEL}}$ to interpret the abstract types and constants to the types and terms built over the signature $\Sigma^{\mathsf{sem}}_{\mathsf{LABEL}}$, respectively.

3.6.2.1 Interpretations of Types

In order to interpret the type DU into the labeled semantics, we slightly modify its standard, unlabeled interpretation. In particular, we interpret a term of type DU to a term $\lambda P.\, P\, F$ of type $(\ell \to t) \to t$, where F is of type ℓ and P is of type $\ell \to t$. Therefore, we translate DU to $(\ell \to t) \to t$, which we abbreviate as ℓtt.

$$(\ell \to t) \to t \triangleq \ell tt$$
$$(e \to \ell \to t) \to \ell \to t \triangleq qnpl$$

Table 3.14: Abbreviations of types

Remark 3.7. *The point that we translate a term of type* DU *to the term* $\lambda P.\, P\, F$ *in both, the unlabeled and labeled semantics, is due to the fact that* $\lambda P.\, P\, F$ *is a generic type-raising mechanism introduced by Montague (1973). In the case of D-STAG, one type-raises* $F : t$ *to the term* $\lambda P.\, P\, F$ *of type* $(t \to t) \to t$*, because an object (i.e.* F*) on which one operates is of type* t*. In the case of the labeled semantics, we predicate over the expressions of type* ℓ*, i.e., labels. Therefore, we use the term* $\lambda P.\, P\, F$ *of type* ℓtt *as a type-raised version of* F*.*

We interpret the type S as $\ell \to t$ (instead of t as it was in the unlabeled semantics). We will justify this choice in the next section. The interpretations both of the types S_A and V_A is $t \to t$. Table 3.15 shows the interpretations of the types from $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ to the types built over the set $\{e, t, \ell\}$.

$\mathsf{np} := qnpl$	$\mathsf{T} := t$
$\mathsf{n} := e \to \ell \to t$	$\mathsf{DU} := \ell tt$
$\mathsf{V}_\mathsf{A} := t \to t$	$\mathsf{S} := \ell \to t$
$\mathsf{np}_\mathsf{A} := qnpl \to qnpl$	$\mathsf{S}_\mathsf{A} := t \to t$
$\mathsf{n}_\mathsf{A} := (e \to \ell \to t) \to (e \to \ell \to t)$	$\mathsf{DU}_\mathsf{A} := \ell tt \to \ell tt$
$\mathsf{n}_d := (e \to \ell \to t) \to (e \to \ell \to t) \to \ell \to t$	

Table 3.15: Interpretations of the abstract types to the types over $\{e, t, \ell\}$ under the lexicon $\mathscr{L}^{\mathsf{SEM}}_{\mathsf{LABEL}}$

3.6.2.2 Interpretations of Constants

In SDRT, one starts to build a discourse structure by labeling the atomic discourse units, i.e., clauses in the discourse. A clause anchors an initial tree in D-STAG. We encode D-STAG initial trees anchored by clauses with the help of the constants AnchorS and AnchorI. We interpret the constants AnchorS and AnchorI so that whenever they are applied to a term encoding a derivation tree of a clause, they introduce a label for that clause. In other words, the clauses by default do not have labels. However, as clauses become introduced in a discourse (with the help of AnchorS and AnchorI), they are assigned labels. This explains why we interpret a type S as $\ell \to t$: A clause needs a label in order to become a part of a discourse structure.

The only device that we have for introducing labels is the constant $\exists_l : (\ell \to t) \to t$. We use it in the interpretations of AnchorS and AnchorI, which are listed in Table 3.16.

As Table 3.16 shows, the interpretation of AnchorS introduces a label l in order to label a clause s (i.e. to define $s\ l$). Since l labels s, one can further use l in a discourse, i.e., it can serve as an argument to a discourse relation $(Q\ l)$.

We use the constant AnchorI in order to encode the first clause of the discourse. The first clause in the discourse does not substitute in any other tree. The as it does

not have a continuation. With this in mind, we propose the interpretation of AnchorI in Table 3.16. Indeed, we use $\lambda\, l.\, \top$ in order to model ***the end*** (the stop sign) of the discourse. Similar to the case of AnchorS, the interpretation of AnchorI also introduces the label of a clause.

Constants in $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$	Their translations by $\mathscr{L}^{\mathsf{SEM}}_{\mathsf{LABEL}}$
AnchorS	$\lambda\ s\ mod.\ \lambda\ P.\exists\ l.\ mod\ (\lambda\ Q.\ (s\ l)\ \wedge\ (Q\ l))\ P$
AnchorI	$\lambda\ s\ mod.\ \exists\ l.\ mod\ (\lambda\ Q.\ (s\ l)\ \wedge\ (Q\ l))\ (\lambda\, l.\, \top)$

Table 3.16: Interpretations of the constants AnchorS and AnchorI by the lexicon $\mathscr{L}^{\mathsf{SEM}}_{\mathsf{LABEL}}$

3.6.2.2.1 Discourse Connectives

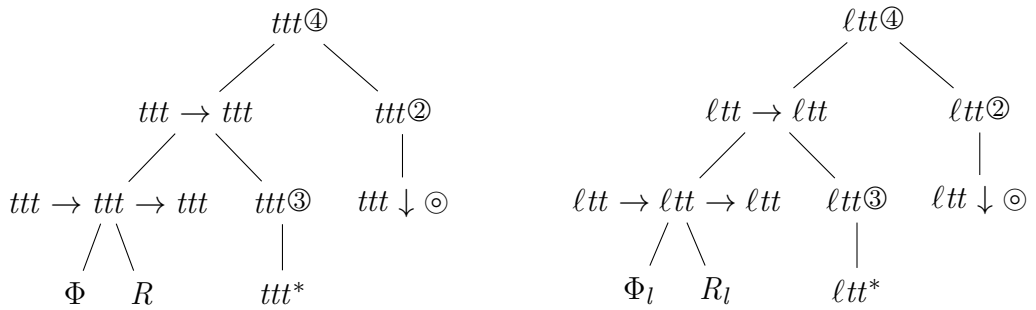
We saw that interpretations of terms encoding initial trees anchored by clauses introduce labels in a discourse. Clauses are atomic discourse units. To build larger discourse units, D-STAG makes use of auxiliary trees anchored by discourse connectives. The larger discourse units also should be labeled. To do that, we interpret auxiliary trees anchored by discourse connectives with the help of the constants $\exists_l : (\ell \to t) \to t$ so that they can introduce labels. In particular, instead of $\Phi^{'}_{\mathsf{new}}$ and $\Phi^{''}$ used in D-STAG semantic trees (see Section 3.4.1), we define $\Phi^{'}_{\mathsf{new}l}$ and $\Phi^{''}_{l}$, in Equation (3.41). The main difference between $\Phi^{'}_{\mathsf{new}}$ and $\Phi^{''}$ and their new versions, i.e., $\Phi^{'}_{\mathsf{new}l}$ and $\Phi^{''}_{l}$ is that the latter ones introduce labels. Now, instead of $R\,x\,y$ (where R is a rhetorical relation and x and y denote two pieces of discourse), we obtain $(R\,x\,y\,l)$, or written alternatively, $l : (R\,x\,y)$.

$$\begin{aligned}
\Phi^{''}_{l} \quad &= \lambda\ R\ X\ Y\ P.\ \exists_l\ l.\ X(\lambda\ x.\ Y(\lambda\ y.\ (P\ x)\ \wedge\ (R\ x\ y\ l))) : \\
&\qquad (\ell \to \ell \to \ell \to t) \to \ell tt \to \ell tt \to \ell tt \\
\Phi^{'}_{\mathsf{new}l} \quad &= \lambda\ R\ X\ Y\ P.\ \exists_l\ l.\ X(\lambda\ x.\ Y(\lambda\ y.\ (P\ (R\ x\ y\ l)))) : \\
&\qquad (\ell \to \ell \to \ell \to t) \to \ell tt \to \ell tt \to \ell tt
\end{aligned} \tag{3.41}$$

Figure 3.25 shows the unlabeled and labeled semantic trees of a discourse connective. These two trees have the same structure. The difference between them is that while the nodes in the original (unlabeled) semantic tree are ttt, in the labeled one, they are ℓtt (as in the case of the labeled semantics, we interpret the type DU as ℓtt).

Consequently, we translate the constants $\mathrm{D}_{\mathbf{conn}_{\mathrm{A}}}$ and $\mathrm{D}_{\mathbf{conn}_{\mathrm{B}}}$ encoding the connective *conn* in $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ as it is shown in Table 3.17. In these interpretations, R_l is a predicate of type $\ell \to \ell \to \ell \to t$ modeling the discourse relation signaled by the connective *conn*. Since, there is no semantic difference between the constants $\mathrm{D}_{\mathbf{conn}_{\mathrm{A}}}$ (resp. $\mathrm{D}_{\mathbf{conn}_{\mathrm{B}}}$) and $\mathrm{D}_{\mathbf{conn}_{\mathrm{A}}}{}^{\mathrm{medial}}$ (resp. $\mathrm{D}_{\mathbf{conn}_{\mathrm{B}}}{}^{\mathrm{medial}}$), we translate $\mathrm{D}_{\mathbf{conn}_{\mathrm{A}}}{}^{\mathrm{medial}}$ (resp. $\mathrm{D}_{\mathbf{conn}_{\mathrm{B}}}{}^{\mathrm{medial}}$) to the same term to which we translate the constant $\mathrm{D}_{\mathbf{conn}_{\mathrm{A}}}$ (resp. $\mathrm{D}_{\mathbf{conn}_{\mathrm{B}}}$).

3.6.2.2.2 First Order Predicates



(a) unlabeled (b) labeled

Figure 3.25: The unlabeled and labeled semantic trees

Constants in $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$	Their translations by $\mathscr{L}^{\mathsf{SEM}}_{\mathsf{LABEL}}$
$\mathrm{D}_{\mathbf{conn_A}}$, $\mathrm{D}_{\mathbf{conn_A}}^{\text{medial}}$	$\lambda\, d_4\, d_3\, d_2.\, \lambda\, d_{subst}.\, \lambda\, d_{foot}.\, d_4\ ((\Phi'_{\mathsf{new}\, l}\ R_l)(d_3\ d_{foot})\ \ (d_2\ d_{subst}))$
$\mathrm{D}_{\mathbf{conn_B}}$, $\mathrm{D}_{\mathbf{conn_B}}^{\text{medial}}$	$\lambda\, d_4\, d_3\, d_2.\, \lambda\, d_{subst}.\, \lambda\, d_{foot}.\, d_4\ ((\Phi''_{l}\ R_l)\ \ (d_3\ d_{foot})\ \ (d_2\ d_{subst}))$

Table 3.17: Semantic interpretations of the constants in $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ encoding discourse connectives

Constants Encoding Nouns, Determiners, Proper Names

We encode nouns in $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ with constants of type $\mathsf{n_A} \multimap \mathsf{n}$ (the translation of the $\mathsf{n_A}$ and n types are given in Table 3.15). Therefore, the type of a term to which a constant encoding a noun translates is as follows: $((e \to \ell \to t) \to (e \to \ell \to t)) \to (e \to \ell \to t)$. In $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$, we encode determiners with the type $\mathsf{n} \multimap \mathsf{np}$. Therefore, the type of a term to which a constant encoding a determiner translates is of type $(e \to \ell \to t) \to \mathit{qnpl}$. We provide translations of the constants encoding nouns, determiners and proper names in Table 3.18.

Constants in $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$	Their interpretations under $\mathscr{L}^{\mathsf{SEM}}_{\mathsf{LABEL}}$
$\mathrm{D}_{\mathbf{noun}}$	$\lambda\, d\, a.\, d\, (a\, (\lambda\, x.\, \lambda l.\, \mathbf{noun}\, x\, l))$
$\mathrm{D}_{\mathbf{a}}$	$\lambda\, P\ Q\ l.\, \exists\, x.\, (P\ x\ l)\ \wedge\ (Q\ x\ l)$
$\mathrm{D}_{\mathbf{the}}$	$\lambda\, P\ Q\ l.\, \exists!\ x.\, (P\ x\ l)\ \wedge\ (Q\ x\ l)$
$\mathrm{D}_{\mathbf{fred}}$	$\lambda\, P\ l.\, P\ \mathbf{fred}$

Table 3.18: Semantic interpretations of the constants in $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ encoding nouns, determiners, and proper names

Constants Encoding Verbs

To interpret the constants modeling initial trees for verbs, one has to take into account that S translates to $\ell \to t$. For instance, in the case a D_{v} constant modeling a transitive verb, we propose the following translation:

$$\mathscr{L}^{\text{SEM}}_{\text{LABEL}}(\text{D}_{\text{v}}) =$$
$$\lambda\, sa\, va_1\, va_2\, sub\, obj.\, \lambda\, l.\, (sa\, (sub\, (\lambda\, x\, l_1.\, obj\, (\lambda\, y\, l_2.\, (va_2\, (va_1\, (\mathbf{Pv}\, x\, y\, l_2))))l_1)l)) :$$
$$(t \to t) \to (t \to t) \to qnpl \to qnpl \to \ell \to t \quad (3.42)$$

In Equation (3.42), $\mathbf{Pv} : e \to e \to \ell \to t$ is a labeled encoding of the predicate signaled by the transitive verb *v*.

In general, the difference with the unlabeled semantic translations and labeled ones is that we have one additional ***parameter*** for labels (i.e. variable of type ℓ), which corresponds to the fact that instead of the type t, we have $\ell \to t$.

3.7 Examples of Labeled Interpretations

We use the following examples of discourses:[87]

(43) a. [Fred is grumpy]$_0$ because [he lost his keys]$_1$. Moreover, [he failed his exam]$_2$.

b. [Fred is grumpy]$_0$ because [he didn't sleep well]$_3$. [He had nightmares]$_4$.

c. [Fred went to the supermarket]$_5$ because [his fridge was empty]$_6$. Then, [he went to the movies]$_7$.

d. [Fred went to the supermarket]$_5$ because [his fridge was empty]$_6$. [He *then* went to the movies]$_{7^m}$.

e. [Fred is grumpy]$_0$ because [his wife is away this week]$_8$. [This shows how much he loves her]$_9$.

In Section 3.7, we already defined terms t_1, t_2, t_3, $\text{t}_3{}^{\text{medial}}$, and t_4 that encode derivations trees of the discourses (43)(a), (43)(b), (43)(c), (43)(d), and (43)(e), respectively. Figure 3.26 illustrates tree representations of these terms.

The lexicon $\mathscr{L}^{\text{SEM}}_{\text{LABEL}}$ translates the terms t_1, t_2, t_3, $\text{t}_3{}^{\text{medial}}$, and t_4 to the terms shown in Equations (3.44), (3.45), (3.46), (3.47), and (3.48), respectively. As one can see, the terms defined in Equation (3.46) and (3.47) are the same as they represent

[87] In Appendix D.4.1, we provide the codes that can use in order to run these examples with the ACG toolkit.

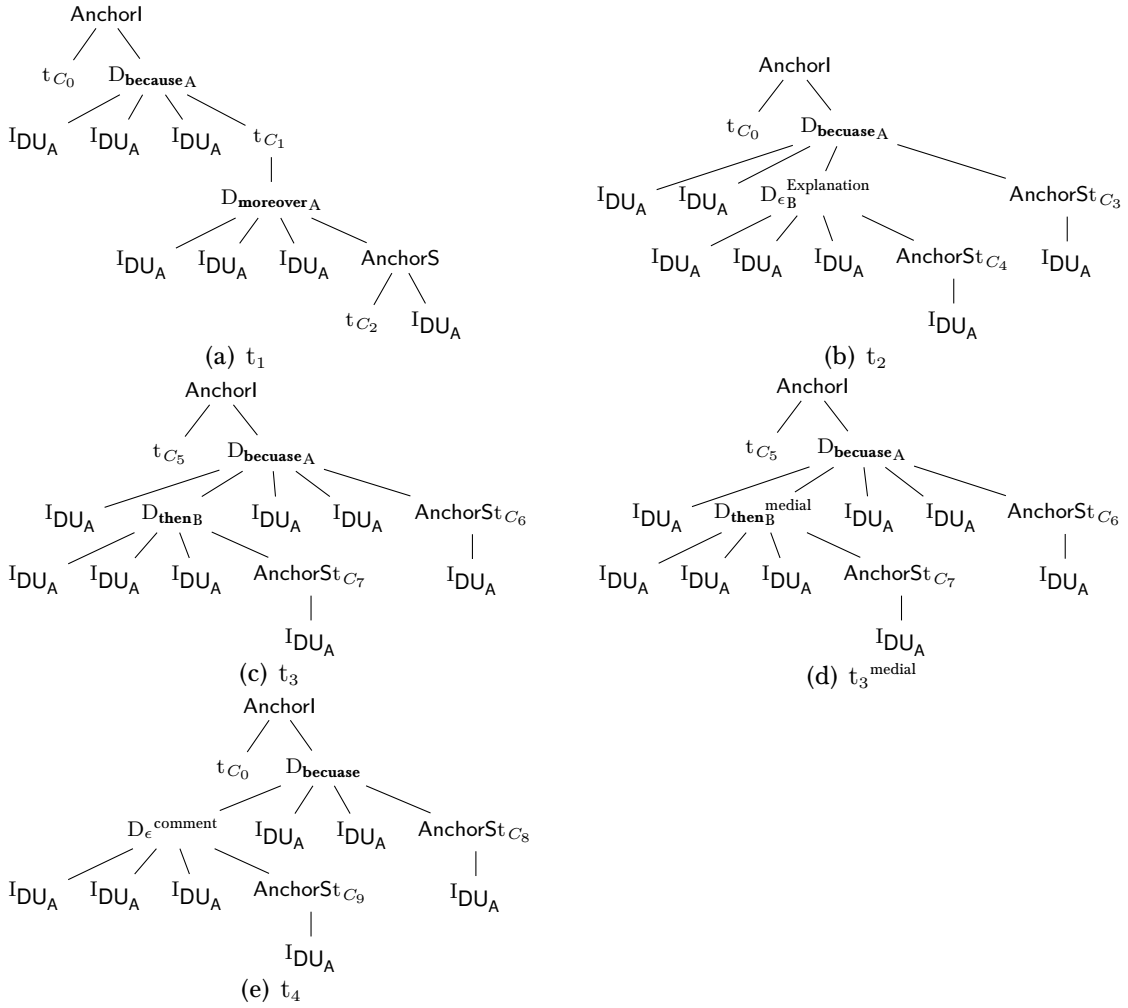


Figure 3.26: The ACG encodings of the D-STAG derivation trees of the examples

interpretations of the discourses (43)(c) and (43)(d), respectively.

$$
\begin{aligned}
&\mathscr{L}_{\text{LABEL}}^{\text{SEM}}(\text{t}_1) = \\
&\exists_l\ l_0\ l_1. \\
&\quad (l_0 :\ \textbf{grumpy fred})\ \& \\
&\qquad (\exists_l\ l_2\ l_3. \\
&\qquad\quad (\exists!\ x.\ l_2 : (\textbf{keys}\ x)\ \&\ l_2 : (\textbf{lose fred}\ x))\ \& \\
&\qquad\quad (\exists_l\ l_4.(\exists!\ x.\ l_4 :\ (\textbf{exam}\ x)\ \&\ l_4 :\ (\textbf{fail fred}\ x))\ \& \\
&\qquad\quad (l_3 :\ (\textbf{Continuation}\ l_2\ l_4\ )\ \&\ (l_1 : (\textbf{Explanation}\ l_0\ l_3)\ \&\ \top)) \\
&\qquad\quad) \\
&\qquad) : t
\end{aligned}
\tag{3.44}
$$

$$
\begin{array}{l}
\mathscr{L}^{\text{SEM}}_{\text{LABEL}}(\text{t}_2) = \\
\exists_l\ l_0\ l_1. \\
\quad (\textbf{grumpy fred}\ l)\ \& \\
\quad (\exists_l\ l_2\ l_3. \\
\qquad (l_3 :\ \neg(\textbf{sleep fred}\ ))\ \& \\
\qquad (\exists_l\ l_4. \\
\qquad\quad (l_4 :\ \textbf{PLUR}(\lambda\ x\ l_5.\ l_5 :\ (\textbf{nightmare}\ x))\ (\lambda\ y\ l_5.\ l_5 :\ (\textbf{have fred}\ y)))\ \& \\
\qquad\quad ((l_1 : (\textbf{Explanation}\ l_0\ l_3)\ \&\ \top)\ \&\ l_2 :\ (\textbf{Explanation}\ l_3\ l_4)) \\
\qquad) \\
\quad) \\
: t
\end{array}
\tag{3.45}
$$

$$
\begin{array}{l}
\mathscr{L}^{\text{SEM}}_{\text{LABEL}}(\text{t}_3) = \\
\exists_l\ l_0\ l_1\ l_2. \\
\qquad (\exists!\ x.\ l_0 :\ (\textbf{supermarket}\ x)\ \wedge\ l_0 :\ (\textbf{go\_to fred}\ x))\ \wedge \\
\qquad (\exists_l\ l_3. \\
\qquad\qquad (\exists!\ x\ .l_3 :\ (\textbf{movies}\ x)\ \wedge\ l_3 :\ (\textbf{go\_to fred}\ x))\ \wedge \\
\qquad\qquad ((\exists_l\ l_4.\ (\exists!x.\ l_4 :\ (\textbf{fridge}\ x)\ \wedge\ l_4 : (\textbf{empty}\ x))\ \wedge\ (l_1 : (\textbf{Explanation}\ l_0\ l_4)\top))\ \wedge \\
\qquad\qquad \wedge\ l_2 : (\textbf{Narration}\ l_0\ l_3)) \\
\qquad) : t
\end{array}
\tag{3.46}
$$

$$
\begin{array}{l}
\mathscr{L}^{\text{SEM}}_{\text{LABEL}}(\text{t}_3{}^{\text{medial}}) = \\
\exists_l\ l_0\ l_1\ l_2. \\
\qquad (\exists!\ x.\ l_0 :\ (\textbf{supermarket}\ x)\ \wedge\ l_0 :\ (\textbf{go\_to fred}\ x))\ \wedge \\
\qquad (\exists_l\ l_3. \\
\qquad\qquad (\exists!\ x\ .l_3 :\ (\textbf{movies}\ x)\ \wedge\ l_3 :\ (\textbf{go\_to fred}\ x))\ \wedge \\
\qquad\qquad ((\exists_l\ l_4.\ (\exists!x.\ l_4 :\ (\textbf{fridge}\ x)\ \wedge\ l_4 : (\textbf{empty}\ x))\ \wedge\ (l_1 : (\textbf{Explanation}\ l_0\ l_4)\top))\ \wedge \\
\qquad\qquad \wedge\ l_2 : (\textbf{Narration}\ l_0\ l_3)) \\
\qquad) : t
\end{array}
\tag{3.47}
$$

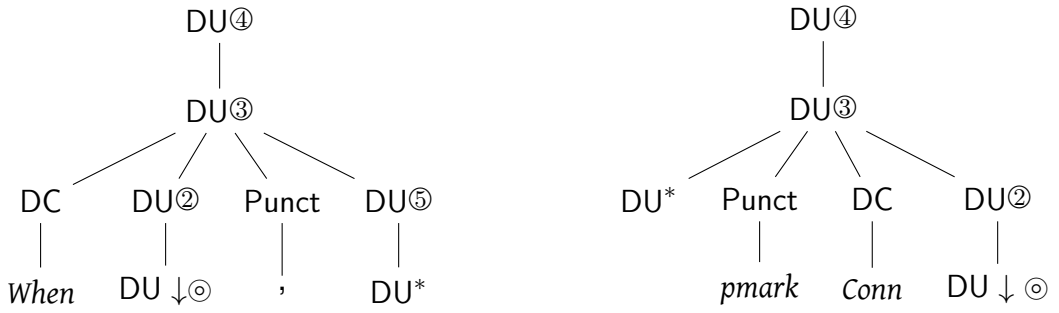
$$
\begin{array}{l}
\mathscr{L}^{\text{SEM}}_{\text{LABEL}}(\text{t}_4) = \\
\exists_l\ l\ l_1\ l_2. \\
\quad l :\ (\textbf{grumpy fred})\ \wedge \\
\quad (\exists_l\ l_3. \\
\qquad (\exists!\ x.\ l_3 :\ (\textbf{wife}\ x\ \textbf{fred})\ \wedge\ (\exists!\ x'.\ l_3 :\ (\textbf{week}\ x')\ \wedge\ l_3 :\ (\textbf{away}\ x\ x')))\ \wedge \\
\qquad (l_2 :\ (\textbf{EXPLANATION}\ l\ l_3)\ \wedge \\
\qquad (\exists_l\ l_4.\ l_4 :\ (\textbf{show-this}(\exists_l\ l_1.\ \textbf{a-lot}\ (\exists!x\ .l_1 : (\textbf{wife}\ x\ \textbf{fred})\ \wedge\ l_1 : (\textbf{love fred}\ x))))\ \wedge \\
\qquad (l_1 :\ (\textbf{COMMENTARY}\ l_2\ l_4)\ \wedge\ \top))) \\
\quad) \\
: t
\end{array}
\tag{3.48}
$$

3.8 Preposed Conjunctions

In D-STAG besides postposed conjunctions, one considers preposed conjunctions such as *when* in Example (49).

(49)
When [Fred was in Paris]$_0$, [he went to the Eiffel Tower]$_1$. Next, [he visited the Louvre]$_2$.

Interpretation: CIRCUMSTANCE (NARRATION F_3 F_2) F_1



(a) The tree anchored with a preposed conjunction

(b) The tree of a postposed connective/discourse adverbial

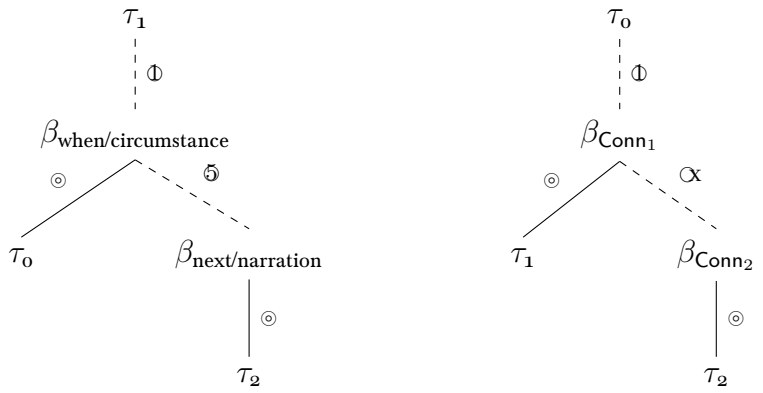
Figure 3.27: D-STAG syntactic elementary trees anchored by connectives

Figure 3.27(a) depicts an elementary tree anchored with a preposed discourse connective. It has four DU-adjunction sites DU②, DU③, DU④, and DU④, whereas elementary trees anchored by postposed conjunctions have three DU-adjunction sites (cf. Figure 3.27(b)). The DU⑤ node has a special usage in D-STAG. In particular, in a case where a preposed conjunction that plays the role of a *framing adverbial*, one uses adjunction on the DU⑤ node. For instance, in (49), *when* is a framing adverbial. Figure 3.28(a) on the following page illustrates the derivation tree of the discourse (49). The tree obtained by substituting τ_2 into $\beta_{\text{next/narration}}$ adjoins on the DU⑤ node of the elementary tree anchored with *when*. The DU⑤ is the mother node of the foot node (i.e. DU*). For instance, the interpretation of the discourse (49) is CIRCUMSTANCE (NARRATION F_2 F_1) F_0. In a case where a preposed conjunction is not a framing adverbial, the adjunction is not performed on the DU node with link ⑤, but on the other DU-adjunction sites (marked with ②, ③, and ④).

In order to encode a preposed conjunction, we introduce two constant $\mathrm{D}_{\textbf{conn-preposed}_\mathrm{A}}$ and $\mathrm{D}_{\textbf{conn-preposed}_\mathrm{B}}$ of type $\mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU} \multimap \mathsf{DU}_\mathsf{A}$ in the abstract vocabulary $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$.

3.8.1 Interpretation as TAG Derivation, and TAG Derived Trees

Figure 3.29 shows the way we analyze a discourse with a preposed connective. By encoding this analysis, we interpret the $\mathrm{D}_{\textbf{conn-preposed}}$ constant modeling the *conn-preposed* preposed connective to TAG derivation trees. Equation (3.50) shows this interpretation.



(a) DSTAG derivation: preposed case

(b) The DSTAG derivation tree in the postposed case ($X = 1, 2, 3, 4$)

Figure 3.28: D-STAG derivation trees of discourses with a preposed and a postposed conjunction

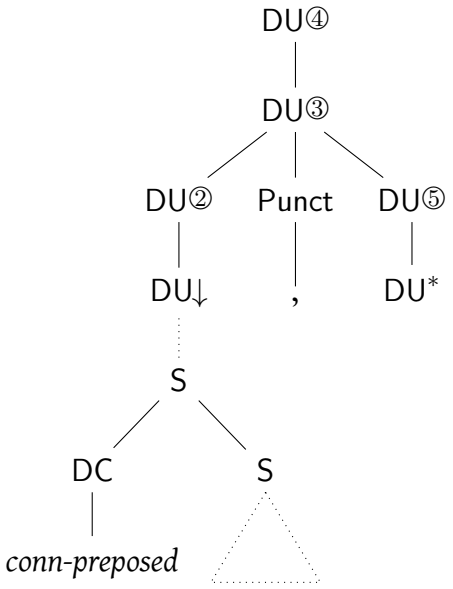


Figure 3.29: The interpretation of $\mathrm{D}_{\textbf{conn-preposed}}$ into TAG derivation trees

$$\mathscr{L}^{\mathsf{DSTAG}}_{\mathsf{TAG}}(\mathrm{D}_{\textbf{conn-preposed}}) = \lambda^{\mathrm{o}}\ d_5\ d_4\ d_3\ d_2\ d_{subst}\ .\ \mathrm{C}^{\text{preposed}}_{\textbf{concat}}\ d_5\ d_4\ d_3\ d_2\ (d_{subst}\ \mathrm{C}_{\textbf{conn-preposed}}{}^{\mathrm{S}}\ (\lambda^{\mathrm{o}} x.x))$$

$$\text{where } \mathrm{C}^{\text{preposed}}_{\textbf{concat}} \in \Sigma^{\mathsf{Der}}_{\mathsf{TAG}} \text{ is of type } \mathsf{S_A} \multimap \mathsf{S_A} \multimap \mathsf{S_A} \multimap \mathsf{S_A} \multimap \mathsf{S} \multimap \mathsf{S_A} \tag{3.50}$$

The constant $\mathrm{C}^{\text{preposed}}_{\textbf{concat}} \in \Sigma^{\mathsf{Der}}_{\mathsf{TAG}}$ the DU-rooted tree anchored with a *comma* shown in Figure 3.29. $\mathrm{C}_{\textbf{conn-preposed}}{}^{\mathrm{S}}$ is a constant of type $\mathsf{S_A}$ modeling the S-rooted auxiliary tree anchored with *conn-preposed*. By adjoining this auxiliary tree on the S-adjunction site of the host clause, one inserts *conn-preposed* into the clause-initial position of the clause.

$$\mathscr{L}^{\text{TAG}}_{\text{synt}}(\text{C}^{\text{preposed}}_{\textbf{concat}}) =$$
$$\lambda\ sa_5\ sa_4\ sa_3\ sa_2\ s_{subst}\ x.\ sa_4\ (sa_3\ (\mathsf{S}_3\ (sa_2\ s_{subst})\ (\mathsf{Punct}_1\ \textit{comma})\ (sa_5\ x))) \quad (3.51)$$

3.8.2 Interpretation as D-STAG Semantic Trees

In order to interpret the constant $\text{D}_{\textbf{conn-preposed}}$ encoding a preposed conjunction to a semantic term, we refer to the D-STAG semantic interpretation of a discourse with a preposed conjunction, discussed in Section 5.3.7 on page 189.

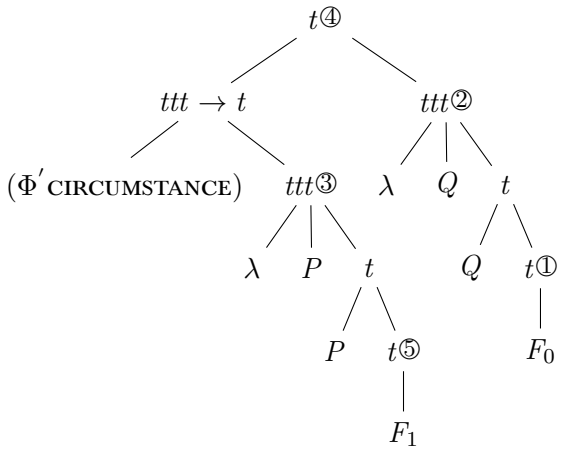


Figure 3.30: The D-STAG semantic tree for a preposed conjunction

For example, Figure 3.30 shows the semantic interpretation of a discourse with a preposed conjunction *when*. By encoding this tree, we obtain the following semantic interpretation of the constant $\text{D}_{\textbf{when}}$:

$$\mathscr{L}^{\text{SEM}}_{\text{DSTAG}}(\text{D}_{\textbf{when}_\text{A}}) =$$
$$\lambda\ d_5\ d_4\ d_3\ d_2\ .\lambda\ d_{subst}.\ \lambda\ d_{foot}.\ d_4\ \ ((\Phi'_{\mathsf{new}}\ \textsc{continuation})\ \ (d_3\ (d_5\ d_{foot}))\ \ (d_2\ d_{subst})) :$$
$$(ttt \rightarrow ttt) \rightarrow (ttt \rightarrow ttt) \rightarrow (ttt \rightarrow ttt) \rightarrow (ttt \rightarrow ttt) \rightarrow ttt \rightarrow (ttt \rightarrow ttt) \quad (3.52)$$

In general, we interpret the constants $\text{D}_{\textbf{conn-preposed}_\text{A}}$ and $\text{D}_{\textbf{conn-preposed}_\text{B}}$ as follows:

$$\mathscr{L}^{\text{SEM}}_{\text{DSTAG}}(\text{D}_{\textbf{conn-preposed}_\text{A}}) =$$
$$\lambda\ d_5\ d_4\ d_3\ d_2\ .\lambda\ d_{subst}.\ \lambda\ d_{foot}.\ d_4\ \ ((\Phi'_{\mathsf{new}}\ \textsc{Rconn})\ \ (d_3\ (d_5\ d_{foot}))\ \ (d_2\ d_{subst})) : \quad (3.53)$$
$$(ttt \rightarrow ttt) \rightarrow (ttt \rightarrow ttt) \rightarrow (ttt \rightarrow ttt) \rightarrow (ttt \rightarrow ttt) \rightarrow ttt \rightarrow (ttt \rightarrow ttt)$$

$$\mathscr{L}^{\text{SEM}}_{\text{DSTAG}}(\text{D}_{\textbf{conn-preposed}_\text{B}}) =$$

$$\lambda\ d_5\ d_4\ d_3\ d_2\ .\lambda\ d_{subst}.\ \lambda\ d_{foot}.\ d_4\ \ ((\Phi''\ \textsc{Rconn})\ \ (d_3\ (d_5\ d_{foot}))\ \ (d_2\ d_{subst})) : \quad (3.54)$$
$$(ttt \to ttt) \to (ttt \to ttt) \to (ttt \to ttt) \to (ttt \to ttt) \to ttt \to (ttt \to ttt)$$

where: RCONN is the relation signaled by the *conn-preposed* connective

Example 3.11.
Let us consider the following example[88] of a discourse with a preposed conjunction:

(49), repeated
When [Fred was in Paris]$_0$, [he went to the Eiffel Tower]$_1$. Next, [he visited the Louvre]$_2$.

Interpretation: CIRCUMSTANCE (NARRATION $F_2\ F_1$) F_0

As Figure 3.30 indicates, in order to derive the semantic interpretation of the discourse (49), D-STAG uses two semantic trees A, where one of them is anchored with CIRCUMSTANCE, and the other one is anchored with NARRATION. We encode the derivation tree of the discourse (49) and this choice of semantic trees by the term $\mathrm{t}_1{}^{\text{preposed}}$, defined in Equation (3.55).

$$\begin{aligned}&\mathrm{t}_1{}^{\text{preposed}} = \\ &\mathsf{AnchorI}\ \mathrm{t}_{C_1}\ (\mathrm{D}_{\textbf{when}_\mathrm{A}}\ (\mathrm{D}_{\textbf{next}_\mathrm{A}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ (\mathsf{AnchorS}\ \mathrm{t}_{C_2}\ \mathrm{I}_{\mathsf{DU_A}}))\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{t}_{C_0})\ : \mathsf{T}\end{aligned} \quad (3.55)$$

where:
$\mathrm{t}_{C_0} = \mathrm{D}_{\textbf{was-in}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{D}_{\textbf{fred}}\ \mathrm{D}_{\textbf{paris}}\ : \mathsf{S}$
$\mathrm{t}_{C_1} = \mathrm{D}_{\textbf{wento-to}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{D}_{\textbf{he}}\ \mathrm{D}_{\textbf{the-eiffel-tower}}\ : \mathsf{S}$
$\mathrm{t}_{C_2} = \mathrm{D}_{\textbf{visited}}\ \mathrm{I}_{\mathsf{S_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{I}_{\mathsf{V_A}}\ \mathrm{D}_{\textbf{fred}}\ \mathrm{D}_{\textbf{the-louvre}}\ : \mathsf{S}$

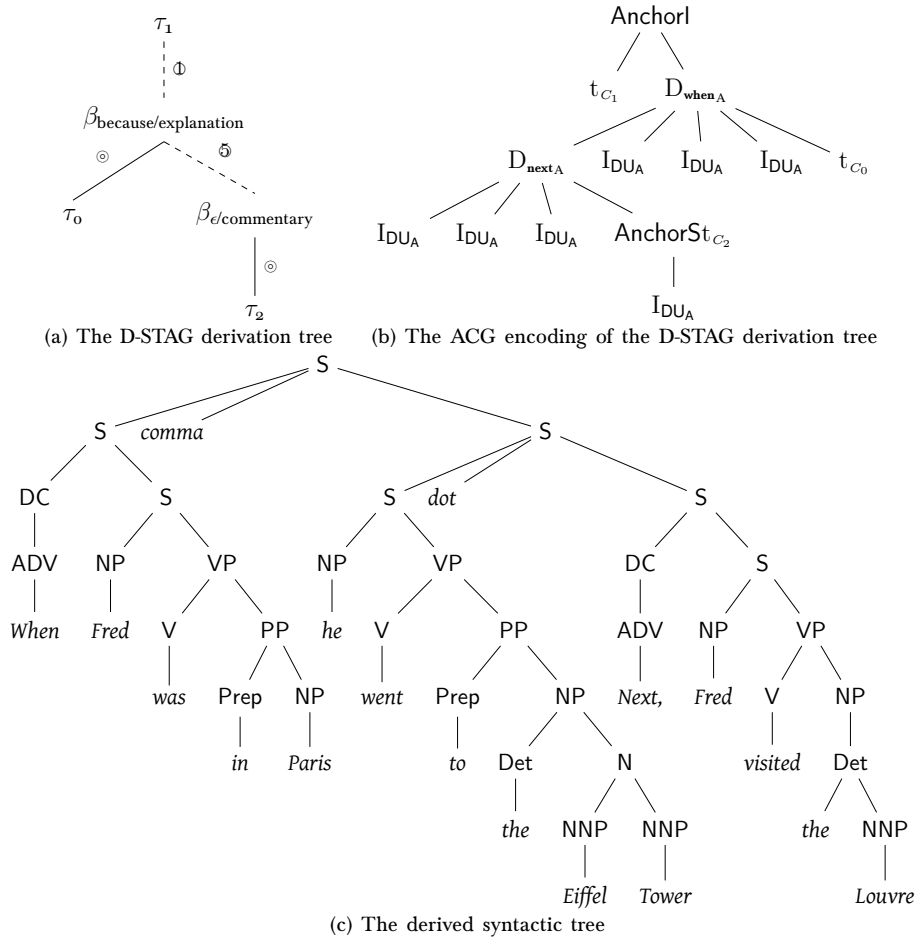
In the term $\mathrm{t}_1{}^{\text{preposed}}$, the terms t_{C_0}, t_{C_1}, and t_{C_2} encode the derivation trees of the clauses C_0, C_1 and C_2. Figure 3.31(b) illustrates the tree representation of the term $\mathrm{t}_1{}^{\text{preposed}}$.

In order to obtain the syntactic interpretation of the discourse (49), we translate the term $\mathrm{t}_1{}^{\text{preposed}}$ by the lexicon $\mathscr{L}^{\mathsf{DSTAG}}_{\mathsf{TAG}} \circ \mathscr{L}^{\mathsf{TAG}}_{\text{synt}}$. Figure 3.31(c) shows the produced derived (syntactic) tree.

By interpreting the term $\mathscr{L}^{\mathsf{DSTAG}}_{\mathsf{TAG}} \circ \mathscr{L}^{\mathsf{TAG}}_{\text{synt}}$ under the lexicon $\mathscr{L}^{\mathsf{SEM}}_{\mathsf{DSTAG}}$, we obtain the following semantic interpretation:

$$\begin{aligned}&\textbf{CIRCUMSTANCE} \\ &\quad (\textbf{NARRATION}\ (\textbf{go-to fred eiffel})\ (\textbf{go-to fred louvre})) \\ &\quad (\textbf{be-in fred paris}) \\ &: t\end{aligned} \quad (3.56)$$

[88]We list the ACG signatures, lexicons and commands in Appendix D.1 that we use in these examples.



(a) The D-STAG derivation tree (b) The ACG encoding of the D-STAG derivation tree

(c) The derived syntactic tree

Figure 3.31: The D-STAG derivation tree, its ACG encoding, and the derived syntactic tree

3.9 Modifiers of Discourse Connectives

In D-STAG, one analyzes a discourse where a connective is *modified*. For instance, in the discourse (57), *for example* modifies the subordinate conjunction *because*. Figure 3.32 shows the D-STAG analysis of the discourse (57). The modifier of the discourse connective *for example* anchors an auxiliary tree rooted in DC. This auxiliary tree adjoins into the auxiliary tree anchored by *because*. The DC-node in the tree anchored by *because* serves as an adjunction site. The link associated with the DC-adjunction site is ⑥.

(57) Fred is grumpy because, for example, he failed an exam.

To model a DC-adjunction site, we introduce a new type DC_A in the abstract vocabulary $\Sigma^{\text{Der}}_{\text{DSTAG}}$. We also introduce the constants of type DC_A in $\Sigma^{\text{Der}}_{\text{DSTAG}}$ to encode modifiers of discourse connectives. For instance, the constant $\mathrm{D}_{\textbf{for-example}} : \mathsf{DC}_\mathsf{A}$ encodes the discourse modifier *for-example*.

Thus, now an elementary tree anchored by a discourse connective has the DC-adjunction site. Therefore, we slightly change the types of the constants in $\Sigma^{\text{Der}}_{\text{DSTAG}}$ encoding trees anchored with discourse connectives. In particular, we add one more argument of type DC_A to a constant encoding a discourse connective. Table 3.19 shows the modified types of the constants $\mathrm{D}_{\textbf{conn}}$ and $\mathrm{D}_{\textbf{conn-preposed}}$ modeling a postposed and preposed discourse connective, respectively.

$$\begin{array}{ll} \mathrm{D}_{\textbf{conn}} & : \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}_\mathsf{A} \multimap \mathsf{DC}_\mathsf{A} \multimap \mathsf{DU} \multimap \mathsf{DU}_\mathsf{A} \\ \mathrm{D}_{\textbf{conn-preposed}} & : \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}_\mathsf{A} \multimap \mathsf{DU}_\mathsf{A} \multimap \mathsf{DC}_\mathsf{A} \multimap \mathsf{DU} \multimap \mathsf{DU}_\mathsf{A} \end{array}$$

Table 3.19: Constants in $\Sigma^{\text{Der}}_{\text{DSTAG}}$ encoding D-STAG trees anchored with discourse connectives

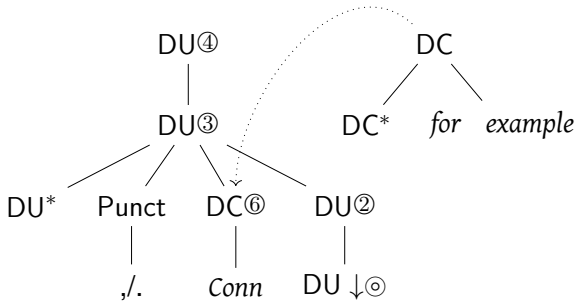


Figure 3.32: The tree anchored with a modifier adjoins on the DC node into the tree anchored with a discourse connective

In D-STAG, a modifier of a discourse connective appears at the clause-initial position, as it is in discourse (57). However, one can also consider a case where a discourse connective is at a clause-initial position, but its modifier appears at a clause-medial position, as it is in the following example:

(58) Fred is grumpy *because* he, *for example*, failed an exam.

In the discourse (58), the connective *because* occupies the clause-initial position, whereas *for example* occupies a clause-medial one.

Thus, we consider two different cases:

1. Both a discourse connective and its modifier appear at the clause-initial positions.
2. A discourse connective appears at the clause-initial position, but its modifier appears at the clause-medial position.

Figure 3.33 illustrates our analysis in the case where both a discourse connective and its modifier occupy the clause-initial positions. The DC-auxiliary tree adjoins into the S-auxiliary tree anchored by the connective. The resultant tree adjoins on the S-adjunction site (*clause-initial position*) into the derived tree of the clause.

Figure 3.34 depicts our analysis of a discourse where the discourse connective is at the clause-initial position but its modifier occupies the clause-medial position. The

S-auxiliary tree anchored by the connective adjoins on the S-adjunction site (*clause-initial position*) into the derived tree of the clause. The DC-auxiliary tree adjoins on the VP-adjunction site (*clause-medial position*) into the derived tree of the clause.

To model these cases, in addition to DC_A, we introduce a new type $\mathsf{DC}^\mathsf{V}_\mathsf{A}$ in the abstract vocabulary $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$. The difference between DC_A and $\mathsf{DC}^\mathsf{V}_\mathsf{A}$ is that we interpret them as a DC-adjunction and a VP-adjunction, respectively, into TAG derivation trees.

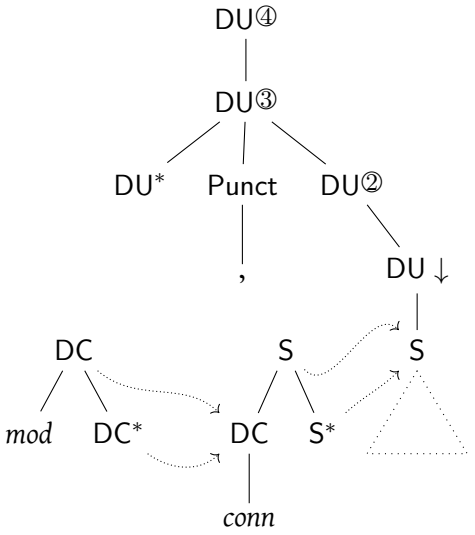


Figure 3.33: Both the discourse connective and its modifier at the clause-initial positions

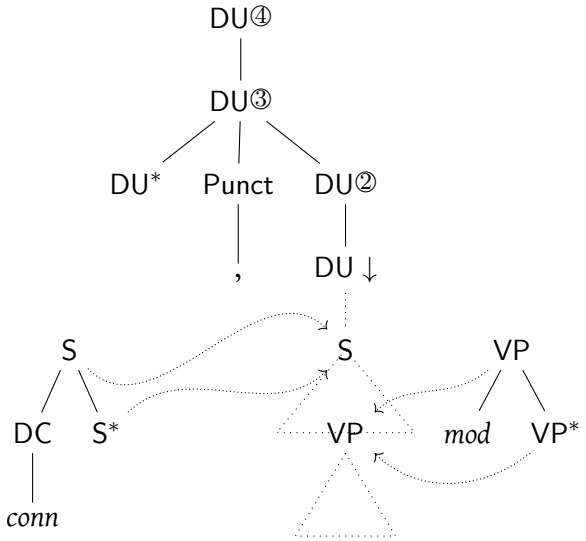


Figure 3.34: The discourse connective is at the clause-initial position, whereas its modifier is at the clause-medial position

Thus, to model the discourse modifier *mod*, we introduce together with the constant $\mathrm{D}_{\mathbf{mod}} : \mathsf{DC}_\mathsf{A}$, another constant $\mathrm{D}_{\mathbf{mod}^{\mathsf{medial}}_{\mathsf{initial}}} : \mathsf{DC}^\mathsf{V}_\mathsf{A}$. We use the constant $\mathrm{D}_{\mathbf{mod}^{\mathsf{medial}}_{\mathsf{initial}}}$ in a case

where an auxiliary tree anchored with *mod* adjoins on the VP node into the derived tree of a clause (see Figure 3.34).

Table 3.20 shows the constants that encode discourse connectives in $\Sigma^{\text{Der}}_{\text{DSTAG}}$. The constant $\mathrm{D}_{\mathbf{conn}}$ models the case where the discourse connective and its modifier, both appear at the clause-initial positions (see Figure 3.33). In the case where the discourse connective *conn* appears at the clause-initial position and its modifier occupies the clause-medial position (see Figure 3.34), we use the constant $\mathrm{D}_{\mathbf{conn}}{}^{\text{initial}}_{\text{medial}}$ of type $\mathsf{DU_A} \multimap \mathsf{DU_A} \multimap \mathsf{DU_A} \multimap \mathsf{DC^V_A} \multimap \mathsf{DU} \multimap \mathsf{DU_A}$.

Constants in $\Sigma^{\text{Der}}_{\text{DSTAG}}$	Their types
$\mathrm{D}_{\mathbf{conn}}$	$\mathsf{DU_A} \multimap \mathsf{DU_A} \multimap \mathsf{DU_A} \multimap \mathsf{DC_A} \multimap \mathsf{DU} \multimap \mathsf{DU_A}$
$\mathrm{D}_{\mathbf{conn}}{}^{\text{initial}}_{\text{medial}}$	$\mathsf{DU_A} \multimap \mathsf{DU_A} \multimap \mathsf{DU_A} \multimap \mathsf{DC^V_A} \multimap \mathsf{DU} \multimap \mathsf{DU_A}$

Table 3.20: Constants in $\Sigma^{\text{Der}}_{\text{DSTAG}}$ encoding discourse connectives

Remark 3.8. *One could also consider a case where both the discourse adverbial and its modifier occupy clause-medial positions, as it is in the following example:*

(59) Fred is grumpy. He failed an exam. He moreover, for example, *lost his keys.*

While we can encode such cases, we leave it for the future work to check the linguistic adequacy of a phenomenon of a modifier of a discourse adverbial.

3.9.1 Interpretations as TAG Derivation Trees

Thus, as Figure 3.33 indicates, we translate $\mathrm{D}_{\mathbf{mod}} : \mathsf{DC_A}$ into $\Lambda(\Sigma^{\text{Der}}_{\text{TAG}})$ as the DC-rooted auxiliary tree anchored with *mod*.

Since we use the $\mathrm{D}_{\mathbf{mod}}{}^{\text{medial}}_{\text{initial}} : \mathsf{DC^V_A}$ constant in order to encode the analysis shown in Figure 3.34, we translate $\mathrm{D}_{\mathbf{mod}}{}^{\text{medial}}_{\text{initial}}$ into $\Lambda(\Sigma^{\text{Der}}_{\text{TAG}})$ as the VP-rooted auxiliary tree anchored with *mod*. Table 3.21 shows the translations of the types and constants encoding discourse modifiers to $\Lambda(\Sigma^{\text{Der}}_{\text{TAG}})$.

Types and constants $\Sigma^{\text{Der}}_{\text{DSTAG}}$	Their translations by the $\mathscr{L}^{\text{DSTAG}}_{\text{TAG}}$ lexicon
$\mathsf{DC_A}$	$\mathsf{DC_A}$
$\mathsf{DC^V_A}$	$\mathsf{V_A} \multimap \mathsf{V_A}$
$\mathrm{D}_{\mathbf{mod}} : \mathsf{DC_A}$	$\mathrm{C}_{\mathbf{mod}} : \mathsf{DC_A}$
$\mathrm{D}_{\mathbf{mod}}{}^{\text{medial}}_{\text{initial}} : \mathsf{DC^V_A}$	$\mathrm{C}_{\mathbf{mod}}{}^{\text{VP}} : \mathsf{V_A} \multimap \mathsf{V_A}$

Table 3.21: Interpretations of the types and constants encoding modifiers of discourse connectives as the types and terms in TAG derivation trees

Figure 3.33 on the preceding page shows our analyses of the case where a discourse connective and its modifier occupy the clause-initial positions. We interpret the constant $\mathrm{D}_{\mathbf{conn}}$ modeling this case as it is shown in Table 3.22. We encode the adjunction of a modifier (dm) into the S-auxiliary tree anchored by the discourse connective ($\mathrm{C}_{\mathbf{conn}}{}^{\text{S}}$).

The resultant tree adjoins in a substituted tree (d_{subst}). Since we have no VP-adjunction, we use $\lambda^{o} x.x$.

Figure 3.34 on page 301 shows our analyses of the case where a discourse connective occupies the clause-initial position, whereas its modifier occupies a clause-medial one. We interpret the constant $\mathrm{D}_{\textbf{conn}_{\text{initial}}^{\text{medial}}}$ modeling this case as it is shown in Table 3.22. We encode the adjunction of a modifier (dm) into the S-auxiliary tree anchored by the discourse connective ($\mathrm{C}_{\textbf{conn}}{}^{\mathrm{S}}$). Since we have no adjunction in the S-auxiliary tree anchored by the discourse connective ($\mathrm{C}_{\textbf{conn}}{}^{\mathrm{S}}$), we use the constant modeling the empty DC_{A}-adjunction $\mathrm{I}_{\mathsf{DC}_{\mathsf{A}}}$.[89] In addition, we encode the adjunction of a modifier (dm) on the VP node into the substituted tree (d_{subst}).

Constants	Their translations by the $\mathscr{L}_{\text{TAG}}^{\text{DSTAG}}$ lexicon
$\mathrm{D}_{\textbf{conn}}$	$\lambda^{o}\ d_4\ d_3\ d_2\ dm\ d_{subst}.\ \mathrm{C}_{\textbf{concat}'}\ d_4\ d_3\ d_2\ (d_{subst}\ (\mathrm{C}_{\textbf{conn}}{}^{\mathrm{S}}\ dm)\ (\lambda^{o} x.x))$
$\mathrm{D}_{\textbf{conn}_{\text{initial}}^{\text{medial}}}$	$\lambda^{o}\ d_4\ d_3\ d_2\ dm\ d_{subst}.\ \mathrm{C}_{\textbf{concat}'}\ d_4\ d_3\ d_2\ (d_{subst}\ (\mathrm{C}_{\textbf{conn}}{}^{\mathrm{S}}\ \mathrm{I}_{\mathsf{DC}_{\mathsf{A}}})\ dm)$

Table 3.22: Interpretations of the abstract constants encoding discourse connectives as TAG derivation trees

3.9.2 Interpretation as D-STAG Semantic Trees

In D-STAG, the tree anchored by a modifier of a discourse connective adjoins on the tree anchored by a discourse relation. Since one encodes discourse relations as constants of type $t \to t \to t$, the semantic interpretation of a modifier of a discourse connective is of type $(t \to t \to t) \to (t \to t \to t)$. In order to interpret an abstract constant modeling a tree anchored by a modifier, we refer to the D-STAG semantic analysis of that modifier. For instance, in the case of the modifier *for example*, we obtain the following interpretation of $\mathrm{D}_{\textbf{for-example}}$:

$$\mathscr{L}_{\text{DSTAG}}^{\text{SEM}}(\mathrm{D}_{\textbf{for-example}}) = \mathscr{L}_{\text{DSTAG}}^{\text{SEM}}(\mathrm{D}_{\textbf{for-example}_{\text{initial}}^{\text{medial}}}) = \lambda\ R\ p\ q.\ \textbf{Exemplification}\ q\ (\lambda\ r.\ R\ r\ q) : \tag{3.60}$$

$$(t \to t \to t) \to t \to t \to t$$

where: **Exemplification** is of type $t \to (t \to t) \to t$

The constants $\mathrm{D}_{\textbf{conn}}$ and $\mathrm{D}_{\textbf{conn}_{\text{medial}}^{\text{initial}}}$ encode the elementary trees anchored by the discourse connective *conn*. Both of them are interpreted as the same semantic term since the difference between them is only *syntactic*. Table 3.23 shows the semantic interpretations of the constants representing discourse connectives, where mod encodes a modifier of the discourse relation RCONN signaled by the connective *conn*.

Example 3.12.

We consider the following examples:

[89] The interpretations in Table 3.22 use the constant $\mathrm{C}_{\textbf{concat}'}$, which was already defined in Section 3.3.6.1 on page 262.

Constants and Types in $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$	Their translations by the $\mathscr{L}^{\mathsf{SEM}}_{\mathsf{DSTAG}}$ lexicon
$\mathrm{D}_{\mathbf{conn}_{\mathrm{A}}}$, $\mathrm{D}_{\mathbf{conn}^{\text{initial}}_{\text{medial-A}}}$	$\lambda\ d_4\ d_3\ d_2.\lambda\ mod.\ \lambda\ d_{subst}.\ \lambda\ d_{foot}.$ $d_4\ (\Phi'_{\mathsf{new}}\ (mod\ \ \textsc{Rconn}))\ (d_3\ d_{foot})\ \ (d_2\ d_{subst}))$
$\mathrm{D}_{\mathbf{conn}_{\mathrm{B}}}$, $\mathrm{D}_{\mathbf{conn}^{\text{initial}}_{\text{medial-B}}}$	$\lambda\ d_4\ d_3\ d_2.\lambda\ mod.\ \lambda\ d_{subst}.\ \lambda\ d_{foot}.$ $d_4\ (\Phi''\ (mod\ \ \textsc{Rconn})\ \ (d_3\ d_{foot}))\ \ (d_2\ d_{subst}))$
DC_{A}, $\mathsf{DC}^{\mathsf{V}}_{\mathsf{A}}$	$(t \to t \to t) \to t \to t \to t$

Table 3.23: Interpretations of the constants and types from $\Sigma^{\mathsf{Der}}_{\mathsf{DSTAG}}$ to $\Lambda(\Sigma^{\mathsf{sem}}_{\mathsf{DSTAG}})$

(61) a. Fred is grumpy because, for example, he failed an exam.

b. Fred is grumpy because, he, for example, failed an exam.

In each of them, the discourse connective appears at the clause-initial position. In (61)(a), the modifier of the connective occupies the clause-initial position, whereas in (61)(b), it occupies the clause-medial position.

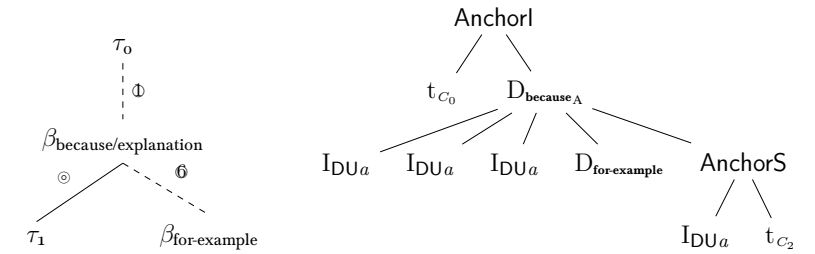
The D-STAG derivation tree in the case of the (61)(a) is shown in Figure 3.35(a) on the next page. We encode it with the term $\mathrm{t}_1{}^{\mathrm{mod}}$, defined in Equation (3.62). Figure 3.35(b) illustrates the tree representation of $\mathrm{t}_1{}^{\mathrm{mod}}$. By interpreting the term $\mathrm{t}_1{}^{\mathrm{mod}}$ by the lexicon $\mathscr{L}^{\mathsf{TAG}}_{\mathsf{synt}} \circ \mathscr{L}^{\mathsf{DSTAG}}_{\mathsf{TAG}}$ as a TAG derived tree, we obtain the derived tree depicted in Figure 3.35(c).

$$\mathrm{t}_1{}^{\mathrm{mod}} = \mathsf{AnchorI}\ \mathrm{t}_{C_0}\ (\mathrm{D}_{\mathbf{because}_{\mathrm{A}}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{D}_{\mathbf{for\text{-}example}}\ (\mathsf{AnchorS}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{t}_{C_2}))\ :\mathsf{T} \tag{3.62}$$

In the case of the discourse (61)(b), we build the term $\mathrm{t}_1{}^{\mathrm{mod}}_{\text{init-med}}$ given in Equation (3.63), whose tree representation is shown in Figure 3.36(a) on page 306. The lexicon $\mathscr{L}^{\mathsf{TAG}}_{\mathsf{synt}} \circ \mathscr{L}^{\mathsf{DSTAG}}_{\mathsf{TAG}}$ interprets $\mathrm{t}_1{}^{\mathrm{mod}}_{\text{init-med}}$ as the derived tree shown in Figure 3.36(b).

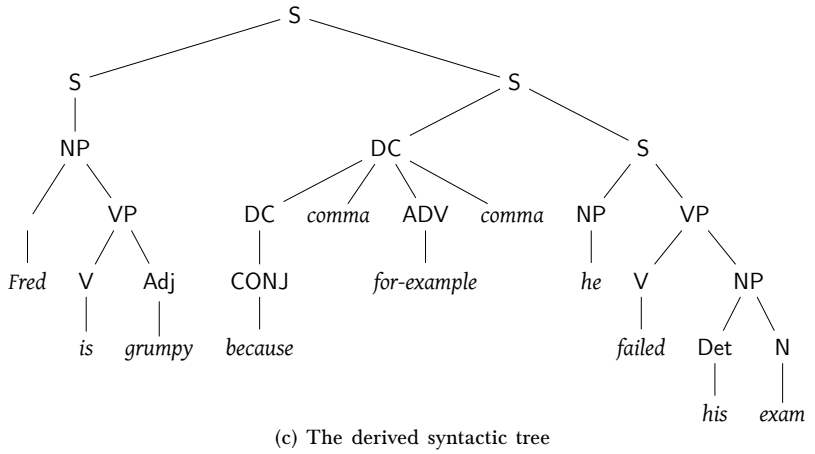
$$\mathrm{t}_1{}^{\mathrm{mod}}_{\text{init-med}} = \mathsf{AnchorI}\ \mathrm{t}_{C_0}\ (\mathrm{D}_{\mathbf{because}_{\mathrm{A}}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{D}^{\text{medial}}_{\mathbf{for\text{-}example}}\ (\mathsf{AnchorS}\ \mathrm{I}_{\mathsf{DU_A}}\ \mathrm{t}_{C_2}))\ :\mathsf{T} \tag{3.63}$$

In order to obtain the semantic representations of the discourses (61)(a) and (61)(b), we translate the terms $\mathrm{t}_1{}^{\mathrm{mod}}$ and $\mathrm{t}_1{}^{\mathrm{mod}}_{\text{init-med}}$ under the lexicon $\mathscr{L}^{\mathsf{SEM}}_{\mathsf{DSTAG}}$. In both of the cases,



(a) The D-STAG derivation tree

(b) The ACG encoding of the D-STAG derivation tree

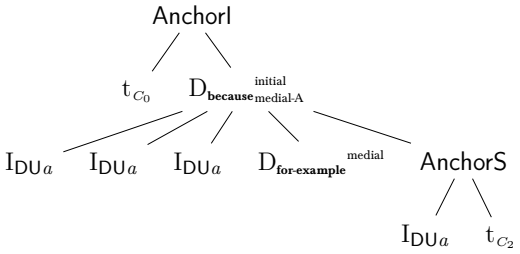


(c) The derived syntactic tree

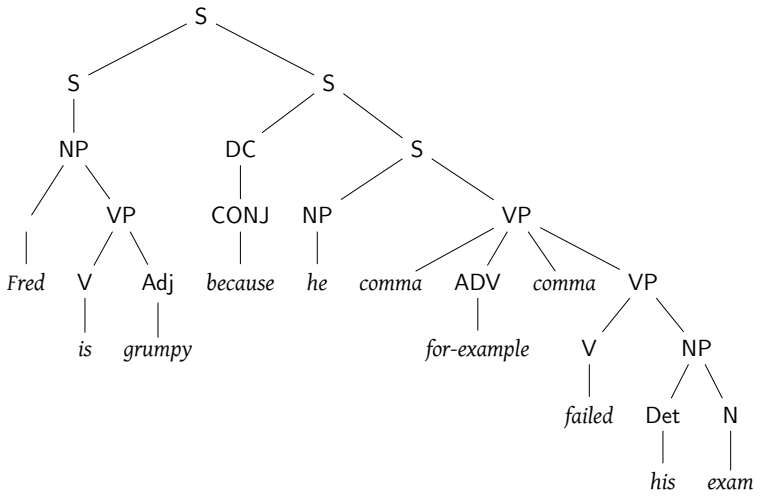
Figure 3.35: The D-STAG derivation tree, its ACG encoding and the derived syntactic tree

we obtain the following interpretation:

$$\begin{aligned}&\mathscr{L}^{\text{SEM}}_{\text{DSTAG}}(\text{t}_1{}^{\text{mod}}) = \mathscr{L}^{\text{SEM}}_{\text{DSTAG}}(\text{t}_1{}^{\text{mod}}_{\text{init-med}}) = \\ &\quad \textbf{Exemplification} \\ &\qquad (\exists!\, x.\, (\textbf{exam}\ x)\ \wedge\ (\textbf{fail fred}\ x)) \\ &\qquad (\lambda\, r.\, \textbf{EXPLANATION}\ (\textbf{bad-mood fred})\ r)\ :t\end{aligned} \tag{3.64}$$



(a) ACG encoding of the D-STAG derivation tree



(b) The derived syntactic tree

Figure 3.36: The D-STAG derivation tree, its ACG encoding, and the derived syntactic tree

Chapter 4

Related Work and Conclusive Remarks

Contents

In this chapter we discuss Discourse Combinatory Categorial Grammar (DCCG) as a related work to the one presented in this thesis. We will consider some questions regarding the discourse formalisms, G-TAG and D-STAG. The same questions one may pose to the ACG encodings of these formalisms, which we presented in this thesis. We provide our views on these problems by suggesting ways of solving them. Furthermore, we outline some directions for the future work.

4.1 Related Work

As for related work to the current one, we single out the work by Nakatsu and White (2010). Their approach follows the D-LTAG discourse structure principles (B. L. Webber, 2004; Bonnie Webber, Stone, Aravind Joshi, and Knott, 2003).[90] They develop

[90]See Section 5.1 on page 131.

a grammar for discourse based on Combinatory Categorial Grammar (CCG) (Steedman, 1987). CCG can be seen as an extension of Lambek Grammars (Lambek, 1958), where one has the type-raising and type-changing mechanisms together with the rules of composition and a functional application.

Nakatsu and White (2010) refer to their grammar for discourse as Discourse Combinatory Categorial Grammar (DCCG). With the help of DCCG, one can generate multi-sentential texts. The conceptual representations from which one generates texts are formulas of Hybrid Logic Dependency Semantics (HLDS) (Kruijff, 2001). HLDS is based on Hybrid Logics (Blackburn, 2000). One can see Hybrid Logics as an extension of Modal Logics such that Hybrid Logics enable one to explicitly name the states (worlds) within a formula. Formulas of HLDS may specify graphs that are not tree-shaped. DCCG follows the D-LTAG principles, DCCG allows for only those graphs that one obtains in D-LTAG. In D-LTAG interpretation of a discourse, anaphoric arguments of discourse adverbials are not specified. The same is true in the case of a DCCG interpretation of a discourse. However, in text generation, Nakatsu and White (2010) assume that anaphoric arguments are known in advance (that is, they are encoded in conceptual representations).

Following D-LTAG, DCCG classifies discourse connectives either as structural or anaphoric. Arguments of a structural connective are pieces of discourse (text segments). These text segments are adjacent to each other.[91] For example, the paired connectives *on the one hand, on the other hand* may have text segments consisting of several clauses (sentences) as their arguments, as it is in the discourse (65). The paired connectives relates the text segments (65)(a)-(65)(b) and (65)(c)-(65)(d).

(65) a. On the one hand, Bienvenue is a mediocre restaurant.

b. However, it has excellent service.

c. On the other hand, Sonia Rose is a good restaurant.

d. However, it has poor decor.

Thus, the paired connectives *on the one hand, on the other hand* relate text segments that are beyond the boundaries of the sentences they appear in. Indeed, the interpretations of the sentences (65)(b) and (65)(d) are parts of the text segments related by *on the one hand, on the other hand*, but the paired connectives *on the one hand, on the other hand* appear neither in (65)(b) nor in (65)(d).

In DCCG, some structural connectives may also appear at clause-medial positions. Namely, the adverbial *however* is considered as a structural connective in D-LTAG[92] and therefore DCCG also treats *however* as a structural one. The following examples illustrate the cases where *however* occupies the clause-initial and clause-medial positions.

[91]DCCG does not deal with attributed texts, i.e., texts with attitude verbs, which are in general problematic in terms of identifying arguments of discourse connectives, even for subordinate conjunctions (see Section 4.1).

[92]Based on the corpus study, (Forbes et al., 2003) claims that *however* exhibits behavior of a structural connective rather than of an anaphoric one.

(66) Mary smiled. *However*, John frowned.

(67) Mary smiled. John, *however*, frowned.

Like for paired connectives, at the discourse-level, the semantic scope of *however* should be extended so that its argument can be a sentence in which *however* does not appear. For instance, one has to encode that in the case of (66) and (67), one of the arguments of the structural connective *however* is *Mary smiled*, which is beyond the sentence boundaries where *however* occurs. In order to extend the scope of a structural connective, (Nakatsu and White, 2010) develops the *cue threading* technique within DCCG.

While one of two arguments of *however* is derived from the sentence that is adjacent to the one where *however* appears, the other argument (called the *host* argument) of *however* is obtained from the sentence where *however* appears. As we saw, in the case where a connective appears at the clause-initial position, it is rather straightforward to identify the host argument compared to the case where the connective appears at the clause-medial position.[93] With the cue threading technique, one overcomes the problem of identifying the host argument of a connective in the case it appears at the clause-medial position.

Moreover, since cue threading was introduced in order to extend the scope of a connective, it enables one to encode a clause-medial structural connective to have text segments as its arguments. For instance, in the case of (67), the clause-medial connective *however* can get the text segments as its arguments.

Remark 4.1. *In the cases of subordinate conjunctions, D-LTAG assumes that they obtain their arguments locally. In particular, both of the arguments of a subordinate conjunction are provided within the sentence where the subordinate conjunction appears (due to that (Nakatsu and White, 2010) refers to them as* intrasentential *conjunctions). Thus, to encode the arguments of a subordinate conjunction is not problematic. That is why to encode subordinate conjunctions, DCCG does not make use of the cue threading technique.*

Cues can be seen as features decorating categories of DCCG. These features indicate whether a discourse connective appears in a text segment or not.

> ... the cue feature is used to mark a clause as containing the structural connective in question. The cue feature is then threaded through the derivation until the point at which the semantic relation for the connective is introduced. Nakatsu and White (2010)

Thus, if a connective Conn appears in a text segment and the expression encoding that text segment has a feature $cue :=_{\mathsf{Conn}}$, it signals that one has to *discharge* the cue. To do that, one derives an expression encoding a text segment that can discharge $cue := \mathsf{Conn}$. Since cue threading is used only for structural connectives, the text segment discharging the cue introduced by a structural connective is adjacent to the text segment

[93] See Section 4.1.

where the structural connective appears. The semantic interpretations of these two text segments are declared as the arguments of the discourse relation signaled by the structural connective. In this way, DCCG extends the scope of a connective beyond the single sentence where it appears.

In the case of structural connectives like *however*, i.e., ones that can occupy clause-medial positions, the cue threading solves the problem of the syntax-semantics interface. Indeed, for a connective occupying either a clause-medial position or a clause-initial one within a clause, the cue denoting the connective becomes the cue value of the entire clause. Moreover, this cue value can be further thread (if the argument of the connective is a larger text segment than just the clause where it appears). In this way, the connective can get text segments as its arguments.

The cue threading is also useful to model the cases where two structural connectives appear in the same sentence, as it is in the following discourse:

(68) Elixir has no significant side effects. *But since* the medicine is for you, never give Elixir to other patients.

In (68), two structural connectives, *but* and *since*, appear in the same clause (*the medicine is for you*). According to DCCG, the cue value of a clause cannot be two connectives at the same time, but only one of them. That is, only one structural connective can be *active* during each step of a derivation. In (68), at first the connective *since* is activated, that is, DCCG assigns since as a value of the cue of the clause. This clause becomes one of the argument of the connective *since*. By finding the other argument of the connective *since*, one discharges the value of the cue of the clause. The other argument of the connective *since* is the clause *never give Elixir to other patients*. In this way, the cue value of the clause where structural connective *since* appears is discharged. Now, one can activate the connective *but*. The cue value of the clause where *but* appears becomes but. One threads this cue value so that the cue value of the entire sentence where *but* appears (*since the medicine is for you, never give Elixir to other patients*) becomes but. This sentence becomes an argument of *but*. DCCG finds that the other argument of *but* is the clause *Elixir has no significant side effects*. In this way, one obtains the interpretation of the discourse (68).

One can use cues for modeling paired connectives such as *on the one hand, on the other hand*. A linguistic assumption of DCCG is that in a discourse if one finds a text segment involving *on the one hand*, then one should be able find an adjacent text segment to that one involving *on the other hand*. DCCG models that by introducing two cues ot1h and otoh. To discharge ot1h, one needs to derive a text segment with the cue equal to otoh. DCCG encodes this fact by defining a *rule* involving these cues.

With the help of the cue threading, one can deal with a discourse such as (65), which gives rise to *nested contrast relations*. Namely, the interpretation of the discourse (65) is the following formula:

$$\textsc{Contrast}\,(\textsc{Contrast}(A, B),\ \textsc{Contrast}(C, D))$$

(65), repeated
a. On the one hand, [Bienvenue is a mediocre restaurant]$_A$.
b. However, [it has excellent service]$_B$.
c. On the other hand, [Sonia Rose is a good restaurant]$_C$.
d. However, [it has poor decor]$_D$.

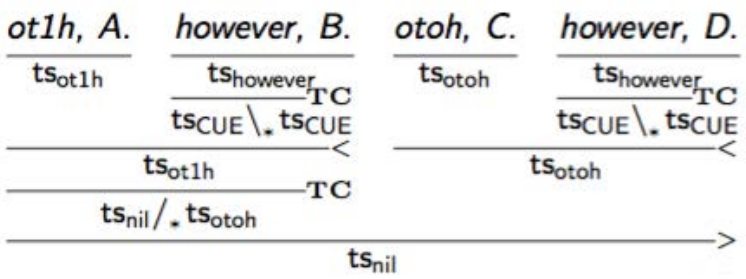


Figure 4.1: A DCCG derivation, Figure adapted from (Nakatsu and White, 2010)

To analyze the discourse (65), DCCG produces the derivation shown in Figure 4.1. Without getting into details of DCCG, we describe this derivation and the rules (denoted by TC) that DCCG defines in order to produce such a derivation. In Figure 4.1, *ot1h* and *otoh* stand for the shorthands for *on the one hand* and *on the other hand*, respectively. *ot1h* introduces the cue ot1h in the text segment *ot1h, A*. In the text segment *however, B*, the connective *however* introduces the cue however. One cannot *combine* these two text segments directly (none of them is a functor). That is why DCCG introduces a rule that enables one to convert the text segment containing *however* into a functor whose cue value is not specified. It takes a text segment as an argument and produces a new text segment that has the same cue value as the argument has. Thus, this functor applied to the text segment *ot1h, A* produces a new text segment with the cue value equal to ot1h. On the semantic side, by applying this functor (which is derived from the text segment *however, B*) to *ot1h, A*, the semantic interpretation of *however*, i.e., CONTRAST receives the interpretation of *A* as an argument. The other argument of CONTRAST is the interpretation of *B* (from *however, B*). The case with the text segment *otoh, C. however, D* is analogous. Thus, for *otoh, C. however, D*, one produces a text segment with the cue value otoh. On the semantic side, one obtains CONTRAST(C, D). Now, we have two expressions encoding the text segments with the cues ot1h and otoh. DCCG has a rule for combining such two expressions. Namely, the rule allows one to transform the text segment with the cue ot1h into a functor that takes a text segment with the cue otoh as an argument. The resultant text segment has a discharged cue ($cue :=$ nil). On the semantic side, the relation CONTRAST signaled by the paired connectives *ot1h, otoh* obtains its arguments: the interpretations of the text segments with the cues ot1h and otoh. The interpretations of these arguments are CONTRAST(A, B) (the interpretation of the text segment with the cue ot1h) and CONTRAST(C, D) (the interpretation of the text segment with the cue otoh). Thus, the interpretation of the discourse (65) is as follows:

$$\textsc{Contrast}\,(\textsc{Contrast}(A, B),\ \textsc{Contrast}(C, D))$$

Remark 4.2. *Since DCCG follows D-LTAG, an adverbial connective is considered to have an anaphoric argument. That is, a DCCG interpretation of an adverbial connective specifies only one of the arguments of the adverbial connective. The specified argument is the structural argument of the adverbial connective. The structural argument is the interpretation of the host clause (one where the connective appears). Thus, to find the structural argument, DCCG does not need to use cue threading.*

As Nakatsu and White (2010) note, one can also develop a purely lexicalized approach to discourse connectives using DCCG, without making use of cue threading. Cue treading is only used for structural connective, i.e., for the ones that obtain their arguments out of the adjacent text segments. Thus, encoding these arguments within the syntactic descriptions of connectives could be also possible. In that case,[94] for each connective, one has to have a number of entries for every syntactic position that connective may occupy (e.g. the clause-medial and clause-initial positions).

There are certain phenomena that DCCG does not give an account of. In particular, in the case where two structural connectives share an argument, the cue threading yields an incorrect semantic analysis. This is due to the fact that DCCG assumes that in each clause, only one structural connective is active and thereby a clause can be an argument of only one structural connective. Recall that neither D-LTAG nor G-TAG deal with this kind of phenomena. The only discourse grammar formalism discussed in the present work that gives a grammatical account of the phenomenon of the argument sharing between connectives is D-STAG. Therefore, the ACG encoding of G-TAG cannot deal with such structures, whereas the ACG encoding of D-STAG can.

On the other hand, for DCCG, it is not a problem to encode a discourse where a structural connective shares one argument with a discourse adverbial, whereas their other arguments are different. To illustrate that, let us consider the following discourse:

(69) [John ordered three cases of Barolo]$_0$. [But he had to cancel the order]$_1$ [*because then* he discovered he was broke]$_2$.

In (69), the structural connective *because* and the adverbial connective *then* share the argument *he discovered he was broke*. The other argument of *because* is *he had to cancel the order*. Since DCCG follows D-LTAG, it only finds one argument of the adverbial connective *then*, obtained from the clause where *then* appears (which is *he discovered he was broke*). The other argument is left underspecified.

The (fully specified) interpretation of (69) is $(\textsc{Contrast}\, F_0 F_1) \wedge (\textsc{Explanation}\, F_1 F_2) \wedge (\textsc{Narration}\, F_0 F_2)$. That is, while *because* and *then* share the argument *he discovered he was broke*, their other arguments are *he had to cancel the order* (the other argument of *because*) and *John ordered three cases of Barolo* (the other argument of *then*).

Contrary to D-LTAG and DCCG, D-STAG interpretation of a discourse specifies both arguments of a discourse connective. In D-STAG, to model the cases such as (69), one makes use of the DNF of a discourse. In particular for (69), one constructs a DNF C_0 but C_1 because $\overline{C_2}$ then C_2. That is, one assumes that *because* has a copy of C_2 as its host clause.

[94] We here only consider grammatical approaches to discourse, such as DCCG, D-LTAG, G-TAG, D-STAG etc.

4.2 Questions

We discuss some questions that one may pose towards formalisms G-TAG and D-STAG, and subsequently to their ACG encodings presented in this thesis.

4.2.1 Paired Connectives and Nested Relations

Nakatsu and White (2010) pose a question regarding G-TAG. Their question is whether G-TAG can analyze paired connectives whose arguments can span multiple sentences, such as the discourse (65) on page 308. One can ask the same question to the ACG encoding of G-TAG. Below, we provide an answer to this question.

4.2.2 Asymmetry of Clause-medial Connectives

Both arguments of a discourse connective can be text segments according to DCCG, which is also the case in G-TAG. Indeed, a constant modeling a tree anchored by a discourse connective is of type $\mathsf{T} \multimap \mathsf{T} \multimap \mathsf{T}$, where T stands for a text segment. However, notice that we encode clause-medial connectives by constants of type $\mathsf{T} \multimap \mathsf{S} \multimap \mathsf{T}$, where S stands for sentences. Thus, in this case, one of the arguments (the host segment) of the clause-medial connective can be only a sentence but not a text. This makes our encoding of clause-medial connectives asymmetric. Below, in Section 4.3.2, we provide a solution that enables a clause-medial connective to have both of the arguments texts.

4.2.3 Multiple Connectives within a Clause

In this thesis, we indeed develop a lexicalized approach, which does not make use of cue threading. However, we have not presented encoding of a case where an adverbial connective and a structural one appear in the same clause. In G-TAG this case is not studied. While D-STAG deals with such cases, it makes use of a preprocessing step in order to interpret discourses such as (69). Below, we will discuss how one can deal with this case without using a preprocessing step. We provide a possible solution for this problem by extending the ACG encoding of D-STAG.

(69), repeated

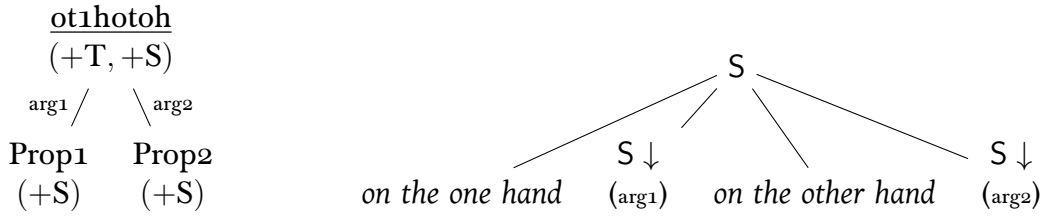
> [John ordered three cases of Barolo]$_0$. [But he had to cancel the order]$_1$ [*because then* he discovered he was broke]$_2$.
> Interpretation: $(\textsc{Contrast}\, F_0 F_1) \wedge (\textsc{Explanation}\, F_1 F_2) \wedge (\textsc{Narration}\, F_0 F_2)$

4.3 Answers

4.3.1 Paired Connectives and Nested Relations

As a formalism, G-TAG can encode the paired connectives in the same way as it is done by D-LTAG. Indeed, every connective in G-TAG is structural and this true for

the paired connectives as well. The underspecified g-derivation tree in Figure 4.2(a) illustrates the lexical entry for the paired connectives ***ot1h, otoh***, whereas Figure 4.2(b) depicts its corresponding initial tree anchored by the paired connectives ***ot1h, otoh***. The feature (+S) indicates that arguments of the lexical entry of the paired connectives can be either sentences or texts.



(a) The lexical entry ot1hotoh

(b) The initial tree for a paired connective

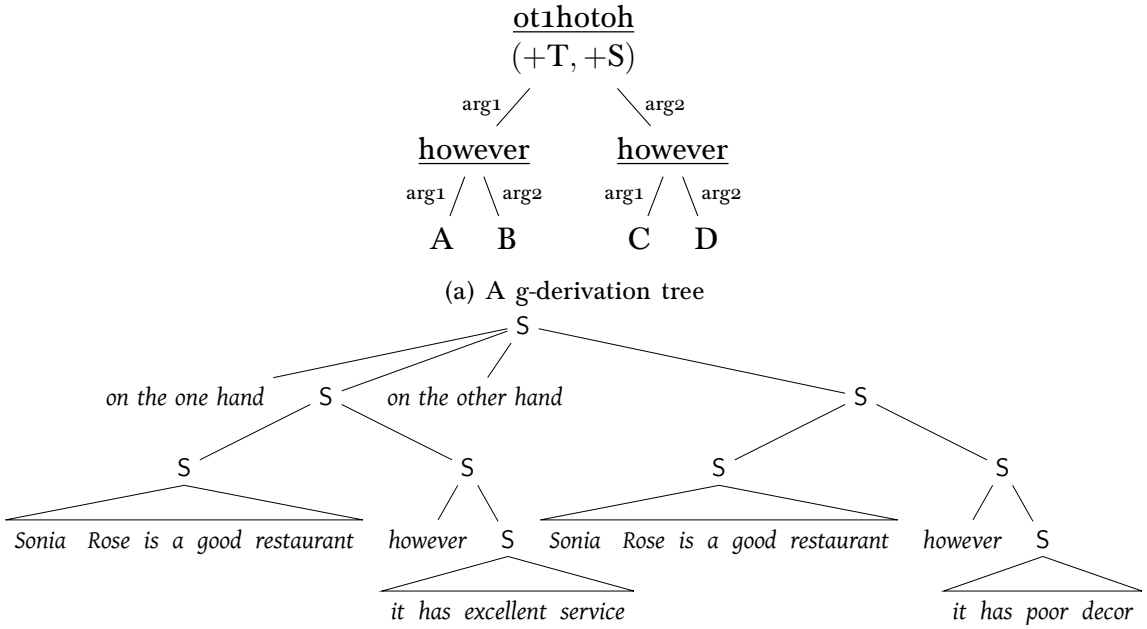
Figure 4.2: The G-TAG lexical entry and the corresponding elementary tree for the paired connectives ***ot1h, otoh***

The concept whose lexicalization is the lexical entry (represented as the tree) in in Figure 4.2(a) is CONTRAST(p_1, p_2), where p_1 and p_2 are the conceptual representations of Prop1 and Prop2, respectively.[95] For instance, one can encode the discourse (65) with the help of G-TAG by constructing the g-derivation tree shown in Figure 4.3(a). A, B, C, and D denote g-derivation trees of the clauses A, B, C, and D respectively. The conceptual representation corresponding to the g-derivation tree is the interpretation of the discourse (65).

(65), repeated
a. On the one hand, [Bienvenue is a mediocre restaurant]$_A$.
b. However, [it has excellent service]$_B$.
c. On the other hand, [Sonia Rose is a good restaurant]$_C$.
d. However, [it has poor decor]$_D$.

Thus, the answer to the question whether G-TAG can encode discourses such as (65) is positive. The same is true for the ACG encoding of G-TAG (see Chapter 1). Indeed, we encode the g-derivation tree and shown in Figure 4.3(b) on the facing page as a term over the abstract vocabulary Σ_{GTAG}. Then we can interpret the term into derived tree in order to obtain the parse (syntactic) tree of the discourse. By interpreting the term with the semantic lexicon, we obtains the semantic interpretation of the discourse. Moreover, the ACG encoding of G-TAG can deal with the paired connectives in the cases where they appear at clause-medial positions, which is also possible to do with the help of DCCG.

[95]Here, for the sake of simplicity of explanation, we do not use LOGIN but HOL for G-TAG conceptual representations.



(a) A g-derivation tree

(b) The derived tree with a paired connectives

Figure 4.3: The g-derivation tree and the syntactic tree it gives rise to

4.3.2 Asymmetry of Clause-medial Connectives

In the ACG encoding of G-TAG, a constant $\mathrm{G}_{\mathbf{adv}_\mathsf{T}}^{\mathsf{T}}$ is of type $\mathsf{T} \multimap \mathsf{T} \multimap \mathsf{T}$. Thus, a discourse connective *adv* has text segments as its arguments.

However, in the case of clause-medial adverbials, we type the constant $\mathrm{G}_{\mathbf{adv}}^{\text{medial}}$ with $\mathsf{T} \multimap \mathsf{S} \multimap \mathsf{T}$. In this case, the first argument is a text (of type T), whereas the other one is a sentence (of type S). Hence, in such cases, our analysis is not capable of relating two text segments, but rather a text segment and a sentence. We can extend our approach to include this case as well. Namely, we introduce a constant $\mathrm{G}_{\mathbf{adv}_2}^{\text{medial}} : \mathsf{T} \multimap \mathsf{T}' \multimap \mathsf{T}$, where T' stands for a text segment where *adv* appears at a clause-medial position. Thus, we should be able to express that in the text segment represented by a term of type T', the first clause can receive a VP adjunction that inserts a connective in a clause-medial position in that clause. In G-TAG, a text segment can only be obtained with the help of some discourse connective *conn* that relates two text segments. Let us assume that these text segments are a sentence (a term of type S) and a text (a term of type T). Figure 4.4 on the next page shows the analysis that we propose. The first piece is a text. The second one is also a text where we single out the first clause. The tree anchored by the connective adjoins on the VP adjunction site (clause-medial position) in the derived tree of this clause.

We encode the analysis illustrated in Figure 4.4 by introducing a constant $\mathrm{AcnhorT}'$ of type $\mathsf{S} \multimap \mathsf{DC} \multimap \mathsf{T} \multimap \mathsf{T}'$, where DC models the type of underspecified g-derivation trees of a discourse connective *conn*.[96]

[96]We could type $\mathrm{AcnhorT}'$ with $(\mathsf{S} \multimap \mathsf{T} \multimap \mathsf{T}) \multimap \mathsf{S} \multimap \mathsf{T} \multimap \mathsf{T}'$, but, if we did so, then the abstract vocabulary would become third-order.

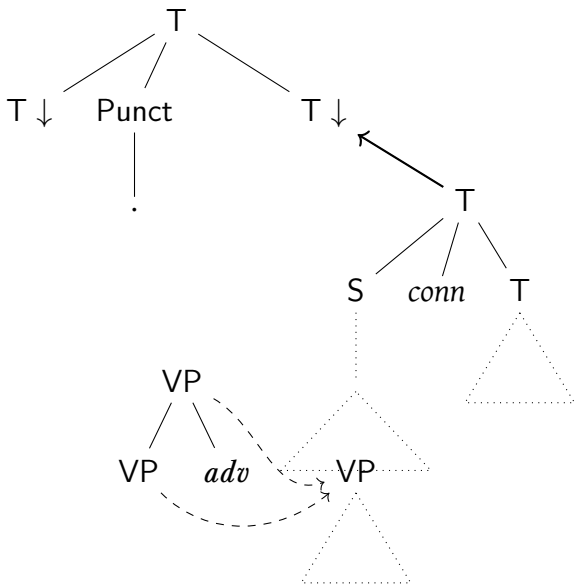


Figure 4.4: An analysis of a case with a connective at a clause-medial position

Now, we can define the semantic and syntactic interpretations of AcnhorT′. We interpret AcnhorT′ into semantics as the term $\lambda^{o}\, s_1\, R\, s_2.\ R\, s_1\, s_2$. Indeed, the arguments of AcnhorT′ are the clause (s_1), the discourse connective (R), and the text segment (s_2). The discourse connective relates these two discourse units ($R\, s_1\, s_2$).

To define a syntactic interpretation of the constant AcnhorT′, we interpret it into TAG derivation trees. Its interpretation gets three arguments, a clause, a connective, and a text and produces out of them their *concatenation*. Thus, we define the following interpretation of AcnhorT′:

$$\mathscr{L}_{\mathsf{GTAG-TAG}}(\mathsf{AcnhorT}') = \lambda^{o}\, s\ \ c\ \ t\ \ mod.\ \mathrm{C}_{\mathbf{Concat}}{}^{3}\,(s\, mod)\, c\, t \tag{4.70}$$

Where $\mathrm{C}_{\mathbf{Concat}}{}^{3}$ stands for the tree shown in Figure 4.5(a).

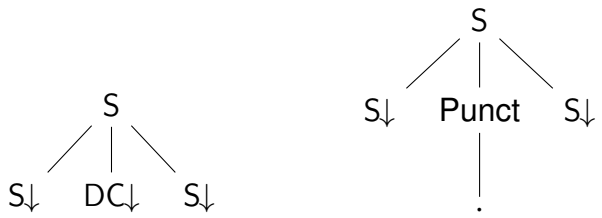
We interpret $\mathrm{G}^{\mathrm{medial}}_{\mathbf{adv}_2}$ as follows: It takes two arguments that stand for derived trees of texts such that the one of them can receive an adjunction. Thus, we propose the following interpretation of $\mathrm{G}^{\mathrm{medial}}_{\mathbf{adv}_2}$:

$$\mathscr{L}_{\mathsf{GTAG-TAG}}(\mathrm{G}^{\mathrm{medial}}_{\mathbf{adv}_2}) = \lambda^{o} t_1\, t_2.\, \mathrm{C}_{\mathbf{concat}}{}^{2}\, t_1\, (t_2\, \mathrm{C}^{\mathrm{VP}}_{\mathbf{adv}}) \tag{4.71}$$

Where $\mathrm{C}_{\mathbf{concat}}{}^{2}$ stands for a tree shown in Figure 4.5(b), and $\mathrm{C}^{\mathrm{VP}}_{\mathbf{adv}}$ denotes the VP-auxiliary tree anchored by *adv*.

Thus, one can model a derivation tree of a discourse where a clause-medial connective *adv* relates two text segments. Let this discourse consist of T_1 segment related to T'. The first clause in T' is s_1. Let some discourse connective *conn* relates the clause s_1 to the rest of the discourse in T', denoted by T_2. To model such a discourse, we define the following term over the abstract vocabulary modeling G-TAG derivation trees:

$$t_2^{\mathrm{m}} = \mathrm{G}^{\mathrm{medial}}_{\mathbf{adv}_2}\ \ t_{T_1}\ \ (\mathsf{AcnhorT}'\ \ t_{s_1}\ \ \mathrm{G}^{\mathrm{DC}}_{\mathbf{conn}}\ \ t_{T_2})\ : \mathsf{T}$$



(a) The tree corresponding to $\mathrm{C}_{\mathbf{Concat}}{}^{3}$

(b) The tree corresponding to $\mathrm{C}_{\mathbf{Concat}}{}^{2}$

Figure 4.5: Trees corresponding to constants $\mathrm{C}_{\mathbf{Concat}}{}^{3}$ and $\mathrm{C}_{\mathbf{Concat}}{}^{2}$

Where t_{T_1} and t_{T_2} encodes the derivation tree of the text segments T_1 and T_2. The term t_{s_1} encodes the derivation tree of the sentence s_1. The constant $\mathrm{G}_{\mathbf{conn}}^{\mathrm{DC}}$ encodes the connective *conn*. By interpreting the term $t_2{}^{\mathrm{m}}$ into TAG derivation trees, we obtain the following:

$$\mathscr{L}_{\mathsf{GTAG-TAG}}(t_2{}^{\mathrm{m}}) = \\ \mathrm{C}_{\mathbf{concat}}{}^{2}\ \mathscr{L}_{\mathsf{GTAG-TAG}}(t_{T_1})\ (\mathrm{C}_{\mathbf{concat}}{}^{3}\ (\mathscr{L}_{\mathsf{GTAG-TAG}}(t_{s_1})\,\mathrm{C}_{\mathbf{adv}}^{\mathrm{VP}})\ \mathrm{C}_{\mathbf{conn}}^{\mathrm{DC}}\ \mathscr{L}_{\mathsf{GTAG-TAG}}(t_{T_2})) \quad (4.72)$$

As Equation (4.72) shows, the tree anchored by the connective *adv* indeed adjoins on the derived tree of the clause in the second text segment ($\mathscr{L}_{\mathsf{GTAG-TAG}}(t_{s_1})\,\mathrm{C}_{\mathbf{adv}}^{\mathrm{VP}}$).

Our encoding of clause-medial connectives is still second-order. However, the quite different modeling of the clause-medial connectives from the clause-initial ones can be considered as a drawback of this encoding. Moreover, we have assumed that one can always split a text into a sentence and a text that are related by a discourse relation. This obviously is not the case in general. Thus, in order to give an account of a more generic case, one has to develop a different approach from the one presented here. At the same time, it would be also interesting to check the linguistic adequacy of the phenomenon of a clause-medial connective whose discursive scope goes beyond the clause where it appears.

4.3.3 Multiple Connectives within a Clause

D-STAG encodes discourses such as (69) with the help of an extra-grammatical processing, involving a duplication of a clause in the DNF of the discourse. To be able to encode the discourses such as (69) with a purely grammatical approach, i.e., without making use of DNF, one has to analyze the behavior of two connectives appearing in the same clause. The connectives *because* and *then*, each signals a discourse relation with two arguments. Thus, neither *because* nor *then* is a parasitic modifier of the other one, as it is in certain cases (see Section 5.3.8). Recall that D-STAG encodes modifiers of discourse connectives with the help of adjunction (since D-STAG is a TAG-based formalism).

Although in (69) *then* is a discourse connective, we offer an analysis according to which *then* syntactically behaves as a modifier of the discourse connective *because*. Thus,

we propose to model *then* by an auxiliary tree anchored with *then* that adjoins into tree with *because* (see Figure 4.6). We introduce a new adjunction site DR in the tree anchored by *because*. To encode this adjunction site, we introduce a new type DR_{A} in the abstract vocabulary of the ACG encoding of D-STAG, where we encode D-STAG derivation trees.

(69), repeated

[John ordered three cases of Barolo]$_0$. [But he had to cancel the order]$_1$ [*because then* he discovered he was broke]$_2$.
Interpretation: $(\textsc{Contrast}\, F_0 F_1) \wedge (\textsc{Explanation}\, F_1 F_2) \wedge (\textsc{Narration}\, F_0 F_2)$

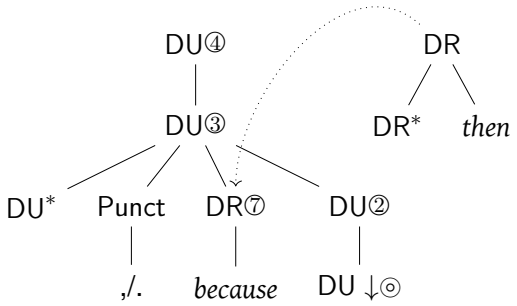


Figure 4.6: The tree anchored with a discourse connective adjoins on the DR node into the tree anchored with a discourse connective

The challenge is to interpret such an analysis into semantics. We have two trees anchored by discourse connectives, one of them is adjoined into the other. The resultant tree has one substitution site where the host clause of these connectives substitutes. In the semantic interpretation, the host clause is shared by these two connectives, but their other arguments are different from each other. Thus, one may define a composition of two discourse relations so that the result of the composition has three arguments: One of these arguments is shared by these relations, whereas the other two arguments, each serves as the other argument of each of these relations. We propose the semantic interpretation of the tree anchored by *then* modifying the tree anchored by *because* so that one obtains the following:

$$\lambda f_2 f_1 f_0.\, (\textsc{Explanation}\, f_1 f_2) \wedge (\textsc{Narration}\, f_0 f_2) : (t \to t \to t) \to t \to t \to t \to t$$

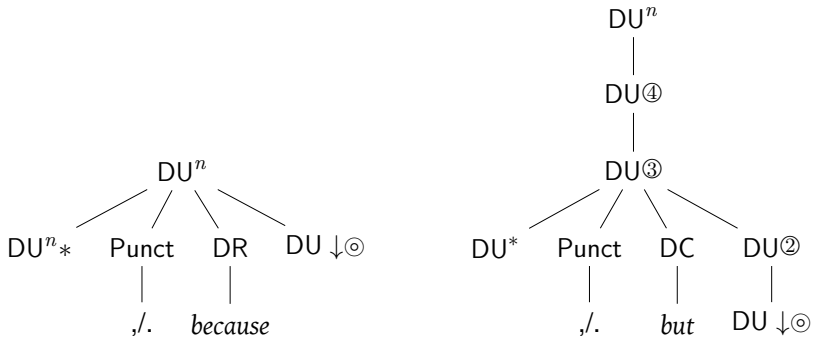
Indeed, as one can see Narration and Explanation share the argument f_2. The argument f_2 is the interpretation of the host clause for *because*. The interpretations of *because* and *then* also take the arguments that differ from each other, denoted by f_0 and f_1. These arguments (f_0 and f_1) come from the piece of discourse: ***John ordered three cases of Barolo. But he had to cancel the order.*** in (69), f_0 and f_1 are also related to each other through the discourse relation Contrast, signaled by the connective ***but***.

For now, let us forget about adjunction sites of a tree anchored by a discourse connective *because*. In the abstract vocabulary, where we encode D-STAG derivation

trees,[97] we introduce two constants $\mathrm{D}_{\textbf{because}}$ and $\mathrm{D}_{\textbf{then}_2}^{\ \mathrm{m}}$ to model the trees anchored by ***because*** and ***then***. We propose the following interpretations of $\mathrm{D}_{\textbf{because}}$ and $\mathrm{D}_{\textbf{then}_2}^{\ \mathrm{m}}$:

$$\begin{aligned}\mathrm{D}_{\textbf{because}} \;=\; &\lambda\, DRadj.\, \lambda\, F_3\; Q\; F_2\; F_1.\, \lambda\, D.\\ &D((Q\; F_2\; F_1\; (\lambda\, x.\, x)) \wedge (F_3\; (\lambda\, f_3.\, (F_2\; (\lambda\, f_2.\, (F_1\; (\lambda\, f_1.\, DRadj\; \textsc{Explanation}\; f_3\; f_2\; f_1)))))))) :\\ &(ttt \to ttt \to ttt) \to ttt \to ttt \to ttt \end{aligned} \tag{4.73}$$

$$\mathrm{D}_{\textbf{then}_2}^{\ \mathrm{m}} \;=\; \lambda\, R\, f_3\, f_1\, f_2.\, (R\, f_2\, f_3) \wedge (\textsc{Narration}\; f_1\; f_3) : (t \to t \to t) \to t \to t \to t \to t \tag{4.74}$$



(a) The tree anchored with ***because*** (b) The tree anchored with ***but***

Figure 4.7: Trees anchored by discourse connectives

As one can see the type of the semantic interpretation of $\mathrm{D}_{\textbf{because}}$ is not the same as the interpretation of a constant encoding a tree anchored by a connective, in the case of ACG encoding of D-STAG (cf. Chapter 3). Therefore, we need to introduce a new type $\mathsf{DU_A}^n$ to type the constant $\mathrm{D}_{\textbf{because}}$. Thus, the constant $\mathrm{D}_{\textbf{because}}$ is of type $\mathsf{DR_A} \multimap \mathsf{DU} \multimap \mathsf{DU_A}^n$. It corresponds to a tree shown in Figure 4.7(a). Thus, instead of DU-adjunction, we make use of DU^n adjunction. Therefore, to be able to adjoin the tree anchored by ***because*** into the tree anchored by ***but***, the latter should have a DU^n-adjunction site. Figure 4.7(b) depicts the tree anchored by ***but***. Thus, we encode the constant $\mathrm{D}_{\textbf{but}}$ with the type $\mathsf{DU_A}^n \multimap \mathsf{DU_A} \multimap \mathsf{DU_A} \multimap \mathsf{DU_A} \multimap \mathsf{DU} \multimap \mathsf{DU_A}$.

Now, to encode the derivation tree of discourse (69), we define the following term:[98]

$$\mathsf{Anchorl}\, C_0\; (\mathrm{D}_{\textbf{but}}\; (\mathrm{D}_{\textbf{because}}\; \mathrm{D}_{\textbf{then}_2}^{\ \mathrm{m}}\; C_2)\; \mathrm{I}_{\mathsf{DU_A}}\, \mathrm{I}_{\mathsf{DU_A}}\, \mathrm{I}_{\mathsf{DU_A}}\; C_1) \tag{4.75}$$

By interpreting this term with the help of the above defined interpretations, one indeed obtains $(\textsc{Contrast}\, F_0 F_1) \wedge (\textsc{Explanation}\, F_1 F_2) \wedge (\textsc{Narration}\, F_0 F_2)$.

However, as one may notice, the interpretation of type $\mathsf{DR_A}$ is $(t \to t \to t) \to t \to t \to t \to t$, which is not exactly the semantic interpretation of an adjunction (modifier),

[97] See Chapter 3.

[98] In Appendix E, we provide the ACG codes that one can run on the ACG toolkit for this example.

which would rather be $(t \to t \to t) \to (t \to t \to t)$. At the same time, one can argue that *then* is not a modifier of *because* either, and thereby, its interpretation is not an interpretation of a modifier. One can view the interpretation the type DR_A to indicate that: Instead of $(t \to t \to t) \to t \to t \to t$, the semantic interpretation of DR_A is $(t \to t \to t) \to (t \to t \to t) \to t$.

4.4 Anaphora Resolution and Referring Expression Generation

The tasks of anaphora resolution and referring expression generation can be considered beyond the problems of the syntax-semantics interface. One can argue that referring expression generation is even domain (genre) related. Nevertheless, there are certain tasks that one can try to solve within the syntax-semantic interface. For example, one may try to encode a set of possible antecedents to a given anaphoric pronoun. For that, one can make use of the approach developed in (de Groote, 2006). It presents a dynamic logic approach to discourse modeling in the spirit of Montague, called *Type Theoretic Dynamic Logic* (TTDL). According to the main principles of TTDL, the meaning, i.e., the interpretation of a sentence must be computed with respect to its *left* and *right* contexts: *A meaning of a sentence is a function of its left and right contexts.* In TTDL, the type of a sentence is $\gamma \to (\gamma \to o) \to o$, where γ is the type of a left context, $(\gamma \to o)$ is the type of a right context, and o is the type of propositions. Thus, the interpretation of the sentence is a *function* of its left and right contexts.

One of the main reasons why introducing dynamic frameworks becomes necessary is a problem of *anaphoric accessibility* that Montague Grammar faces. Therefore, it is interesting to see how TTDL deals with this problem. However, before doing so, let us provide the TTDL operation of discourse *updating*.

$$[\![\mathsf{D.S}]\!] \;=\; \lambda \mathrm{i}.\lambda \mathrm{r}.\, [\![\mathsf{D}]\!]\, \mathrm{i}\; (\lambda \mathrm{i}'.\, [\![\mathsf{S}]\!] \mathrm{i}'\, \mathrm{r}) \tag{4.76}$$

$$\overbrace{\qquad}^{\mathrm{i}:\gamma}\ \mathsf{D}.\,\mathsf{S}\ \overbrace{\qquad\qquad}^{\mathrm{r}:\gamma\to t} \qquad \underbrace{\qquad\qquad}_{\lambda \mathrm{i}'.[\![\mathsf{S}]\!]\,\mathrm{i}'\,\mathrm{r}\,:\gamma\to t} \tag{4.77}$$

The Equation (4.76) (pictorially illustrated in (4.77)) presents a formula for computing the result of *appending/updating* a new sentence S to a given piece of discourse D. If D is represented by a single sentence, then Equation (4.76) shows a formula for computing the *conjunction* of two sentences. Note that in order for the right-hand side of Equation (4.76) be well-typed, the type of $[\![\mathsf{D}]\!]$ has to be $\gamma \to (\gamma \to o) \to o$ that is the same as the type of $[\![\mathsf{S}]\!]$, which means that the *type of a piece of discourse is the same as the type of a sentence.* Indeed, the resulting piece of discourse $[\![\mathsf{D.S}]\!]$ has its left (denoted as i) and right (denoted as r) contexts. The left context of $[\![\mathsf{D.S}]\!]$ coincides with the left context of $[\![\mathsf{D}]\!]$; the right context of $[\![\mathsf{D.S}]\!]$ is the right context of $[\![\mathsf{S}]\!]$. That is why in Equation (4.76), in order to compute $[\![\mathsf{D.S}]\!]$, $[\![\mathsf{D}]\!]$ is applied to the left context

of $[\![\mathsf{D.S}]\!]$, and $[\![\mathsf{S}]\!]$ is applied to the right context of $[\![\mathsf{D.S}]\!]$. Such architecture of TTDL allows it to encode DRT (Kamp, 1988) in a straightforward way:[99]

$$\text{A DRS} \quad K = \{x_1, \ldots, x_n\}\{C_1, \ldots, C_m\} \quad \text{is translated as follows:}$$
$$\lambda \mathrm{i}.\lambda r.\, \exists x_1, \ldots, \exists x_n.\, C_1 \wedge \ldots \wedge\, C_m \wedge\, r(x_1 :: \ldots :: x_n :: \mathrm{i}) \quad (4.78)$$

Where $(x_1 :: \ldots :: x_n :: \mathrm{i})$ is the result of updating a (left) context i with the discourse referents $x_1, \ldots, x_n$ (:: is a list update operator).

For the sake of illustration, let us consider an example of a discourse (79). TTDL provides the compositional means for interpreting the discourse (79) as u defined in Equation (4.80). As u indicates, **snoring** does not apply to x (the discourse referent introduced by ***a man***) but $\mathsf{sel}(x :: \mathrm{i})$. The expression $\mathsf{sel}(x :: \mathrm{i})$ stands for an entity that can be selected in the context $(x :: \mathrm{i})$. The context $(x :: \mathrm{i})$ is a result of updating the left context of the discourse (79) with x, which comes from the first sentence in (79) and thus, it is part of the left context for ***he is snoring.*** The tasks of computing the value of $\mathsf{sel}(x :: \mathrm{i})$ is beyond the compositional approach to the problem of the syntax-semantics interface. Nevertheless, the value of $\mathsf{sel}(x :: \mathrm{i})$ cannot be an arbitrary entity but only from the context $(x :: \mathrm{i})$. In the case of (79), the value of $\mathsf{sel}(x :: \mathrm{i})$ should be x.

(79) A man is sleeping. He is snoring.

$$u = \lambda \mathrm{ir}.(\exists x.(\mathbf{man}\, x) \wedge (\mathbf{sleeping}\, x) \wedge (\mathbf{snoring}\, \mathsf{sel}(x :: \mathrm{i})) \wedge (\mathrm{r}(x :: \mathrm{i}))) \quad (4.80)$$

It would be useful to employ the TTDL notions for dealing with anaphora resolution in the discourse parsing task and also for the referring expression generation in the discourse generation task. Since one deems these tasks to be outside of the scope of a compositional approach, one can avoid dealing with anaphora resolution (resp. referring expression generation) but still parse or interpret a discourse (resp. generate a discourse). In the interpretation of a discourse, there will not be solved anaphoric links. In the case of generation, in the generated discourse, the reference expressions would not be generated. Nevertheless, in both of the cases, one would have a restriction on the context from where a discourse referent comes from. According to that, one would have to resolve an anaphoric link in the parsing task, or generate a reference expression in the generation task.

Asher and Pogodalla (2011) propose an encoding of SDRT in the style of TTDL. However, one needs to define a richer notion of a left context than just a list of entities. Indeed, since SDRSs incorporate labels of discourse units and also information about the attachment points, where new discourse units can be added, one should encode all these. To adjust the notion of a left context to SDRT, a context is interpreted as a record with the following fields:

- A field for labels;

[99]Besides DRT (Kamp, 1988), DPL (Groenendijk and Stokhof, 1991) is also encoded in TTDL, because the notions of left and right contexts defined in TTDL generalize the notions of states, assignments, etc.

- a field for accessible labels;
- a field for accessible discourse referents;
- a field for a proposition.

The richer left context enables one to select more than just an entity from the context. Also, one can update a discourse not just with an entity but with a new content coming from the new piece that attaches to the current discourse. To model that, Asher and Pogodalla (2011) define the following selector and update functions:

- $\mathsf{sel}_L : \gamma \to \ell$ selects an accessible label from the left context;
- $\mathsf{sel}_E : \gamma \to \iota$ selects an accessible discourse referent from the left context;
- $\mathsf{sel}_\rho : \gamma \to \ell \to \ell \to \ell \to t$ selects a discourse relation, i.e., a relation that holds between three labels, and in result produces a proposition;
- $v : \gamma \to \ell \to \gamma$ updates a context with the label coming from the new piece by establishing the links (via rhetorical relations) between the label of the new piece and the labels contained in the current context.

Developing a similar approach to the one of Asher and Pogodalla's (2011) may help to overcome the problems of anaphora resolution/referring expression generation. In particular, one can model the syntax-semantics interface without explicitly naming discourse referents but only the contexts from which they can be selected. Furthermore, by developing an approach following the TTDL style, one can also think of encoding D-LTAG where certain connectives have anaphoric arguments. Indeed, D-LTAG considers finding one of the arguments of an adverbial connective to be beyond a compositional account of the syntax-discourse interface. Therefore, to develop a TTDL style approach for encoding anaphoric arguments of adverbial connectives of D-LTAG would allow one to encode D-LTAG within a type-logical framework.

Chapter 5

Conclusion

In this thesis, we took the first steps in the ACG modeling of discourse by encoding the discourse grammar formalisms, G-TAG and D-STAG, as ACGs. This enabled us to develop an approach to the syntax-semantics interface for discourse with ACGs.

De Groote (2001) introduced ACGs to address the problems of the syntax-semantics interface so that both syntax and semantics are encoded in a uniform way. ACGs proved to be useful for encoding a number of formal grammars, including Tree-Adjoining Grammars (TAGs). In the ACG encoding of TAG, derivation trees of TAG are modeled as the pivots to TAG derived trees. In particular, TAG derivation trees were presented as abstract terms and derived trees as object ones. In addition, by computing out of TAG derivation trees the Montague style semantic interpretations, the ACG encoding of TAG proved to be useful for modeling the syntax-semantics interface for sentences (Pogodalla, 2004). The main objective of this thesis was to provide modeling of discourse-level phenomena with ACGs. In the present work, we have studied the formalisms that offer solutions to the problem of discourse modeling based on formal grammars. Since the ACG encoding of TAG with Montague semantics proved to be successful, the formalisms that offered discourse grammars based on the TAG principles were good candidates for being encoded as ACGs. We have selected among such formalisms G-TAG (Danlos, 1998) and D-STAG (Danlos, 2009).

G-TAG was introduced for text generation, with an aim to implement it in practical applications, whereas D-STAG was introduced for discourse parsing. Although both G-TAG and D-STAG are based on TAG, they were designed under different assumptions about the discourse structure. While in G-TAG discourse structures are trees, in D-STAG they can be non-tree shaped directed acyclic graphs (DAGs). That is why G-TAG and D-STAG grammars are also different from each other. Consequently, the ACGs that we have constructed to encode G-TAG and D-STAG also differ. Nevertheless, the same generic architecture serves to both of these ACG encodings. We have modeled derivation trees (either of G-TAG or of D-STAG) as abstract terms. In each of these ACG encodings, we have defined two lexicons. One lexicon interprets the abstract terms encoding derivation trees into derived trees. The other one interprets the same abstract terms as logical formulas. Within a logical formula, in addition to the interpretations of clauses in a discourse, one encodes the structure of the discourse.

The notable difference between G-TAG and D-STAG and their corresponding ACG encodings is that in order to deal with a certain kind of texts, both G-TAG and D-STAG make use of an extra-grammatical step, whereas their encodings do not. Namely, in order to model a text where a discourse connective occupies a clause-medial position, both G-TAG and D-STAG perform an additional, non-grammatical processing of a discourse. G-TAG does that after the grammatical processing step. The grammatical processing step gives rise to a discourse where a discourse connective can only occupy a clause-initial position. In order to generate a discourse where a connective appears at the clause-medial position, the G-TAG post processing module moves some connectives (if any) from clause-initial positions to clause-medial ones. In D-STAG, it is the preprocessing step that takes care of clause-medial connectives. During the preprocessing step, one normalizes clauses in a discourse containing connectives at clause-medial positions by moving those connective to clause-initial positions. Afterwards, one applies the D-STAG grammar in order to obtain the syntactic and semantic interpretations of the discourse. Thus, each of the G-TAG and D-STAG grammars encodes discourse connectives only at clause-initial positions.

In contrast to G-TAG and D-STAG, in their ACG encodings, we have modeled discourse connectives at clause-medial positions as grammar entries. Namely, in each of these ACG encodings, we have introduced a constant in the abstract vocabulary for modeling a connective at a clause-medial position. This constant is different from the one that represents the same connective at a clause-initial position. Having different constants for modeling a connective in the clause-initial and clause-medial positions enabled us to define different syntactic interpretations of these constants. At the same time, we have defined the same semantic interpretations of these constants. In this way, one obtains the correct semantic and syntactic interpretations of a discourse containing connectives at clause-medial positions. For each of the formalisms G-TAG or D-STAG, in its ACG encoding, we have encoded clause-medial connectives without deviating from the general principles of that formalism regarding discourse connectives and discourse structure. Although the grammar entries of G-TAG and D-STAG significantly differ from each other, in their ACG encodings, we propose modelings of clause-medial connectives based on the same principles. The method we have developed for encoding clause-medial connectives within the ACG encodings of G-TAG and D-STAG can be applied for TAG-based approaches in general. Indeed, both of the abstract vocabularies where we encode derivation trees of G-TAG and D-STAG are second-order, which is also the case for the ACG encoding of TAG. In other words, to encode clause-medial connectives, we do not make use of a more expressive abstract vocabulary than the one that one designs for encoding TAG derivation trees. While in the abstract vocabulary (either of the ACG encoding of G-TAG or of D-STAG), we do not encode the constraints needed for modeling clause-medial connectives, we define *rich* syntactic interpretations where we express those constraints. That is, in the syntactic interpretations we encode the different behaviors of clause-medial and clause-initial connectives. Hence, for TAG-based approaches, which one can encode with the help of second-order ACGs, one can make use of the method of encoding clause-medial connectives proposed within the present work.

The encodings of clause-medial and clause-initial connectives in the abstract vo-

cabulary are uniform. For pinpointing *where* the difference between clause-medial and clause-initial connectives should be encoded, one may search for an answer within the cognitive aspects of linguistics. We encoded the difference between them in their syntactic interpretations. One may see it as a technical point of our encodings. Nevertheless, one may consider it as a suggestion that in the derivation-level (in the abstract vocabulary), the clause-medial and clause-initial connectives have the same properties. On the other hand, it has been argued that clause-medial connectives have different pragmatic effects from clause-initial ones (Forbes et al., 2003). However, the grammars we encoded as ACGs, (a) they do not provide encodings of clause-medial connectives; (b) they do not consider other aspects of meaning but only discourse semantics. Due to that neither our encodings of these formalisms takes into account pragmatic phenomena.

The class of second-order ACGs consists of intrinsically reversible grammars. For grammars of this class, one uses the same polynomial algorithm to build parse structures both for strings and logical formulas (Kanazawa, 2007; Salvati, 2005). Since the ACG encodings we have proposed are second-order, the problems of discourse parsing and generation with the ACG encodings of G-TAG and D-STAG are of polynomial complexity.

As for future work, we have identified several problems. For some of them, we have suggested solutions that could be further refined. Among them is a problem of an interaction of two connectives appearing in the same clause. While that clause serves as an argument to both of the connectives, the other arguments of the two connectives differ. Although such cases are studied in D-STAG, to encode them, D-STAG makes use of an extra-grammatical processing. By contrast, we have provided a purely grammatical modeling of the phenomenon of two connectives appearing in the same clause. Our encoding of the complex interaction of two discourse connectives is in the boundaries of second-order ACGs. However, it would be interesting to examine whether our encoding of this phenomenon is linguistically sound.

In addition, we have also pointed out that to deal with the referring expression generation and anaphora resolution tasks within a compositional framework, one may develop an approach based on the TTDL principles (de Groote, 2006). TTDL does not provide a tool for generating referring expressions or resolving anaphora; it rather allows one to develop a compositional approach to modeling of anaphoric expressions without specifying their antecedents but the contexts from where they can be selected. Such an approach could be useful for modeling discourse formalisms such as D-LTAG, where certain connectives have anaphoric arguments and thereby their values are not specified in a D-LTAG interpretation of a discourse.

To sum up, our work makes explicit the ACGs that one can use for discourse modeling. In particular, we have provided the ACG encodings of two discourse grammar formalisms, G-TAG and D-STAG. In this way, we have given an answer to the main inquiry of this thesis. Since the ACG encodings of both G-TAG and D-STAG are second-order, our results suggest that the second-order ACGs are suitable for expressing constraints that one needs in the case of modeling the discourse-level phenomena. Therefore, the results of the current thesis motivate to further investigate the problems related to discourse with second-order ACGs and/or identify the limits of second-order ACGs in terms of discourse modeling.

Appendix A

TAG as ACG codes

ACG signatures and lexicons that we use in the examples are listed in Section A.1. It has In order to obtain syntactic and semantic analyses for TAG with Montague semantics, we use lexicon `tag_syntax` and `tag_semantics` , as the following commands show:

```
tag_syntax tag_semantics realize
C_grumpy (C_because I_s (C_failed I_s I_vp C_john (C_exam C_an I_n))) (C_is I_vp) C_john:S;

tag_syntax tag_semantics realize
C_grumpy (C_because I_s (C_failed I_s I_vp C_john (C_exam C_an I_n))) C_is C_john:S;
```

A.1 TAG as ACGs: Signatures and Lexicons

```
signature derivation_trees =

(* types with as Xa with an "a" index indicate they are meant
for adjunction. See https://hal.inria.fr/inria-00141913 for further
explanation of the TAG into ACG encoding *)

Sa,Na, Na_d, N, VPa, S,WH : type;

(* Declatiation of abstract constants together with their
types. -> stands for the linear implication and => (not used in this
signature) stands for the intuituinnistic implication *)

C_dog,C_cat,C_exam:Na_d -> Na-> N;
C_sleeps:Sa -> VPa -> N -> S;
C_chases, C_loves, C_to_love, C_failed:Sa -> VPa -> N -> N -> S;
C_every,C_a,C_an:Na_d;
C_slowly,C_seems : VPa -> VPa ;
C_new,C_big,C_black : Na -> Na;
C_claims,C_said : Sa -> VPa -> N -> Sa ;
C_john,C_paul,C_mary,C_bill : N ;
C_who : WH;
C_liked : Sa -> VPa -> WH -> N -> S ;
C_does_think : Sa -> VPa -> N -> Sa ;
C_grumpy : Sa -> VPa -> N -> S;
C_is : VPa;

     C_because : Sa -> S ->Sa;


(* Dummy element to specify the end of adjunctions *)
I_vp : VPa;
I_n : Na;
I_s : Sa;
end

(* Now we specify the signature for derived trees *)

signature derived_trees =

(* It uses ony one type : tye type of tree *)

tree:type;
```

```
(* Here are the non terminal symbol we find in the trees, with
an index indicating their arity *)
WH1,N1,VP1 : tree -> tree;
N2,S2,VP2:tree -> tree -> tree;

(* Here are the terminal symbols *)

every,dog,chases,a,cat,sleeps,slowly,new,big,black,seems,john,mary,bill,paul,
claims,loves,to_love,who,said,liked,does,think,grumpy, is, epsilon, failed, exam, an, because:tree;

(* We define feww constants that will make the lexicon definitions easier. *)

n = lambda n . lambda d a.d (a(N1 n)) : tree -> (tree -> tree) -> (tree -> tree) -> tree;
iv = lambda v . lambda s a np0 .s (S2 np0 (a (VP1 v))) : tree -> (tree -> tree)  -> (tree -> tree) -> tree -> tree ;
tv = lambda v . lambda s a np0 np1 .s (S2 np0 (a (VP2 v np1))) : tree -> (tree -> tree) -> (tree -> tree) -> tree -> tree -> tree ;
ph_arg_v = lambda v . lambda s_root a np0 s_foot .s_root (S2 np0 (a (VP2 v s_foot))) : tree -> (tree -> tree) -> (tree -> tree) -> tree -> tree -> tree ;
det = lambda d . lambda n . N2 d n : tree -> (tree -> tree) ;
adv = lambda adv . lambda a v . a (VP2 v adv) : tree -> (tree -> tree) -> (tree -> tree) ;
l_adj = lambda adj. lambda a n . a (N2 adj n) : tree -> (tree -> tree) -> (tree -> tree) ;
r_adj = lambda adj. lambda a n . a (N2 n adj) : tree -> (tree -> tree) -> (tree -> tree) ;
ctrl_v = lambda v. lambda v_root v_foot .v_root (VP2 v v_foot) : tree -> (tree -> tree) -> (tree -> tree) ;
np = lambda proper_name . N1 proper_name : tree -> tree;
inf_tv = lambda v . lambda s a np0 np1 .S2 np1 (s (S2 np0 (a (VP1 v)))) : tree -> (tree -> tree) -> (tree -> tree) -> tree -> tree -> tree ;
wh_extract_tv = lambda v . lambda s adv wh subj . S2 wh (s (S2 subj (adv (VP1 v)))) : tree -> (tree -> tree) -> (tree -> tree) -> tree -> tree -> tree;

padj = lambda adj. lambda s a np0. s (S2 np0 (a (VP2 (VP1 epsilon) adj)))  : tree -> (tree -> tree) -> (tree -> tree) -> tree -> tree ;
end

(* Then a signature for the strings *)
signature strings =

string: type;

(* we can define infix and prefix symbols. Note that as for now, the length of symbols can only be 1 *)

infix + : string -> string -> string;

every,dog,chases,a,cat,sleeps,slowly,new,big,black,seems,john,mary,bill,paul,
claims,loves,to,love,who,said,liked,does,think,grumpy, is, epsilon, failed, exam, an, because:string;
end

(* Ok. Now is our first lexicon. It translates derived trees into strings *)

lexicon tag_strings(derived_trees) : strings =

(* So every tree result in a string *)
tree := string;

every := every;
dog := dog;
chases := chases;
      exam := exam;
      failed := failed;
a := a;
      an := an;
cat := cat;
sleeps := sleeps;
slowly := slowly;
new := new;
big := big;
black := black;
seems := seems;
john := john;
mary := mary;
bill := bill;
paul := paul;

claims := claims;
loves := loves;
to_love := to + love;
who := who;
said := said;
liked := liked;
does := does;
think:=think;
grumpy:=grumpy;
epsilon:=epsilon;
      is := is;

      because:=because;

WH1,N1,VP1 := lambda f.f;
N2,S2,VP2:=lambda f g . f + g;

end

(* We also provide a signature for the semantics *)

signature semantics =

(* We define the usual types *)
```

```
e,t:type;

(* Then few non logical-constants *)
dog,cat,sleep,grumpy,exam : e->t;
love,chase,like, fail:e -> e -> t;
j,m,b,p:e;
slowly : t -> t;
seem : (e -> t) -> e -> t;
new,big,black:e ->t;
claim,say,think : e -> t -> t;
      PresTense:  (e -> t) -> e -> t;

      Because: t->t->t;

WHO : (e -> t) -> t;

(* And finally, here are the logical constants *)

infix & : t -> t -> t;
infix > : t -> t -> t;
binder All : (e=>t) -> t;
binder Ex : (e=>t) -> t;

end


(* We now define the semantics associated to each derivation tree *)

lexicon tag_semantics(derivation_trees) : semantics =
S := t;
N := (e -> t) -> t;
Sa := t -> t;
Na := (e =>t) -> (e =>t);
VPa := (e -> t) -> (e -> t);
Na_d := (e => t) -> (e -> t) -> t;
WH := (e ->t) -> t;

      C_because := lambda a. lambda s x. a (Because s x);

C_every := lambda n.lambda P.All x. (n x) > (P x) ;
C_a, C_an := lambda n.lambda P.Ex x. (n x) & (P x);

C_dog := lambda d a . d (a (Lambda x.dog x)) ;
C_cat := lambda d a . d (a (Lambda x.cat x)) ;
C_exam := lambda d a . d (a (Lambda x.exam x)) ;

C_sleeps := lambda s a S.s(S(a(lambda x.(sleep x))));

C_grumpy := lambda s a S.s(S(a(lambda x.(grumpy x))));
      C_is := lambda x.  PresTense x;

C_chases := lambda s a S O.s(S(a(lambda x.O(lambda y.(chase x y)))));
C_loves := lambda s a S O.s(S(a(lambda x.O(lambda y.(love x y)))));
C_failed := lambda s a S O.s(S(a(lambda x.O(lambda y.(fail x y)))));

C_to_love := lambda s a O S.s(S(a(lambda x.O(lambda y.(love x y)))));
C_slowly := lambda vp r. vp (lambda x. slowly (r x));
C_seems := lambda vp r.  vp (lambda x. seem r x);
C_new := lambda a n . a (Lambda x.(new x)&(n x));
C_big := lambda a n . a (Lambda x.(big x)&(n x));
C_black := lambda a n . a (Lambda x.(black x)&(n x));
C_claims := lambda sa a S comp. sa (S(a(lambda x.claim x comp)));
C_said := lambda sa a S comp. sa (S(a(lambda x.say x comp)));
C_john := lambda P.P j;
C_mary := lambda P.P m;
C_paul := lambda P.P p;
C_bill := lambda P.P b;
C_who := lambda P.WHO P;
C_liked := lambda sa a w S.w(lambda y.sa(S(a(lambda x.(like x y)))));
C_does_think := lambda sa a S comp. sa(S(a(lambda x.(think x comp))));
I_vp := lambda x.x;
I_n := lambda x.x;
I_s := lambda x.x;
end

(* And a lexicon from derivation trees to derived trees *)

lexicon tag_syntax (derivation_trees) : derived_trees =
N, S, WH := tree;
Sa,Na,VPa,Na_d := tree -> tree ;
C_john := np john;
C_mary := np mary;
C_bill := np bill;
C_paul := np paul;
C_dog := n dog;
C_cat := n cat;
C_exam := n exam;
C_chases :=  tv chases;
C_loves := tv loves ;
      C_failed := tv failed ;

C_to_love := inf_tv to_love ;
```

```
C_sleeps := iv sleeps;
C_seems := ctrl_v seems;
C_claims := ph_arg_v claims;
C_every := det every;
C_a := det a;
      C_an := det an;
C_slowly := adv slowly;
C_new := l_adj new;
C_big := l_adj big;
C_black := l_adj black;

C_who := WH1 who ;
C_liked := wh_extract_tv liked;
C_said := ph_arg_v said;
C_does_think := lambda s_root a subj s_foot . s_root (S2 does (S2 subj (a (VP2 think s_foot))));

C_grumpy := padj grumpy;
      C_is := lambda x. VP2 is x;

      C_because := lambda a s. lambda x. a (S2 x (S2 because s));

I_n,I_vp,I_s := lambda x.x;
end
```

Appendix B

G-TAG as ACG codes

The ACG signature and lexicons that we use in the examples are listed in Section B.3 on the next page.

B.1 Examples

In order to obtain an interpretation of g-derivation trees into:

- TAG derivation trees, we use the *GTAGtoTAG* lexicon;
- syntactic trees, we use the *lexsyntax* lexicon;
- logical semantics trees, we use the *lexsemantics* lexicon.

For example, the ACG code of the term $t_{2_{\text{ex}}}^{\text{GTAG}}$, which encodes the g-derivation tree in Example 1.2 on page 214, is the following:

```
G_ensuiteSS
     (G_pour_r G_jean (G_passe_laspirateur_sws I_s G_a) (G_etre_recomepnse_par I_s I_vp G_marie))
     (G_fait_une_sieste I_s G_a G_jean)
:T
```

To interpret the term $t_{2_{\text{ex}}}^{\text{GTAG}}$ under the lexicon *lexsyntax*, we use the following code:

```
lexsyntax analyse
G_ensuiteSS
     (G_pour_r G_jean (G_passe_laspirateur_sws I_s I_vp) (G_etre_recomepnse_par I_s I_vp G_marie))
     (G_fait_sieste I_s I_vp G_jean)
:T;
```

In order to obtain the semantic interpretation provided in Example 1.3 on page 223, we interpret the same term with the help of the lexicon *lexsemantics*. We obtain the following semantic representation

```
SUCC (GOAL (VACUUM j) (REWARD m j)) (NAP j) : t
```

B.2 An Example of Generation

We parse a term as follows:

```
lexsemantics parse  SUCC (VACUUM j) (NAP j) : T;
```

We generate the set of terms over signature Gderivation (see below):

```
G_ensuiteSS (G_passe_laspirateur I_s G_a G_jean) (G_fait_une_sieste I_s G_a G_jean):T;
G_ensuiteST (G_passe_laspirateur I_s G_a G_jean) (AnchorT (G_fait_une_sieste I_s G_a G_jean)) :T;
G_ensuiteTS (AnchorT (G_passe_laspirateur I_s G_a G_jean)) (G_fait_une_sieste I_s G_a G_jean):T;
G_ensuiteTT (AnchorT (G_passe_laspirateur I_s G_a G_jean)) (AnchorT (G_fait_une_sieste I_s G_a G_jean)):T;

G_auparavantSS (G_fait_une_sieste I_s G_a G_jean) (G_passe_laspirateur I_s G_a G_jean):T;
G_auparavantST (G_fait_une_sieste I_s G_a G_jean) (AnchorT (G_passe_laspirateur I_s G_a G_jean)):T;
G_auparavantTS (AnchorT (G_fait_une_sieste I_s G_a G_jean)) (G_passe_laspirateur I_s G_a G_jean):T;
G_auparavantTT (AnchorT (G_fait_une_sieste I_s G_a G_jean)) (AnchorT (G_passe_laspirateur I_s G_a G_jean)):T;

AnchorT (G_apres_c  (G_fait_une_sieste I_s G_a G_jean) (G_passe_laspirateur I_s G_a G_jean)) :T;
AnchorT (G_apres_r G_jean (G_fait_une_sieste_sws  I_s G_a) (G_avoir_passe_laspirateur I_s I_vp)) :T;

AnchorT (G_avant_c  (G_passe_laspirateur I_s G_a G_jean) (G_fasse_une_sieste_subjunctive I_s I_vp G_jean)) :T;
AnchorT (G_avant_r G_jean (G_passe_laspirateur_sws  I_s G_a) (G_faire_une_sieste I_s I_vp)) :T;
```

We translate them using the lexicon *lexyield* to strings.
For instance, we translate one of the obtained term as follows:

```
lexyield analyse
G_ensuiteTS (AnchorT (G_passe_laspirateur I_s G_a G_jean)) (G_fait_une_sieste I_s G_a G_jean):T;
```

B.3 GTAG as ACG: Signatures and Lexicons

```
signature Gderivation =  (* The Abstract Vocabulary for Encoding G-derivation Trees *)
Sa, N, Na_d, Na, VPa,  S, T, Sws, Sinf: type;
I_vp : VPa;
I_s : Sa;
   I_n : Na;


   G_jean, G_marie :N;
   G_petit_dejeuner:  Na_d -> Na -> N;
   G_le:Na_d;
   G_delicieux : Na -> Na;


      G_ensuiteST : S -> T -> T;     (* constant encoding adverbial :ensuite *)
      G_ensuiteTS : T -> S -> T;     (* constant encoding adverbial :ensuite *)
      G_ensuiteTT : T -> T -> T;     (* constant encoding adverbial :ensuite *)
      G_ensuiteSS : S -> S -> T;     (* constant encoding adverbial :ensuite *)

      G_auparavantST : S -> T  -> T;  (* constant encoding adverbial :auparavant *)
      G_auparavantTS : T -> S -> T;   (* constant encoding adverbial :auparavant *)
      G_auparavantTT : T -> T -> T;   (* constant encoding adverbial :auparavant *)
      G_auparavantSS : S -> S -> T;   (* constant encoding adverbial :auparavant *)

G_avant_c: S -> S ->S;           (* canonical g-derivation tree avant *)
G_avant_r: N -> Sws -> Sinf ->S;  (* reduced conjunction g-derivation tree avant *)

G_pour_c: S -> S ->S;          (* canonical g-derivation tree pour *)
G_pour_r: N -> Sws -> Sinf ->S;   (* reduced conjunction g-derivation tree pour *)

G_apres_c: S -> S ->S;     (* canonical g-derivation tree apres *)
G_apres_r: N -> Sws -> Sinf ->S;  (* reduced conjunction g-derivation tree apres *)

G_recomepnse: Sa -> VPa -> N -> N -> S;
G_recomepnse_subjunctive: Sa -> VPa -> N -> N -> S;

G_etre_recomepnse_par: Sa -> VPa -> N -> Sinf;  (* reduced clause - infinitive clause *)

G_passe_laspirateur: Sa -> VPa -> N->S;
G_passe_laspirateur_subjunctive: Sa -> VPa -> N->S;
G_passe_laspirateur_sws: Sa -> VPa -> Sws; (* clause lacking a subject*)
G_passer_laspirateur_inf: Sa -> VPa -> Sinf;    (* reduced clause - infinitive clause *)
G_avoir_passe_laspirateur: Sa -> VPa -> Sinf;   (* reduced clause - infinitive clause *)


G_fait_une_sieste: Sa -> VPa -> N->S;
G_fasse_une_sieste_subjunctive: Sa -> VPa -> N->S;
G_fait_une_sieste_sws: Sa -> VPa ->Sws; (* clause lacking a subject*)
G_faire_une_sieste: Sa -> VPa -> Sinf; (* reduced clause - infinitive clause *)
G_avoir_fait_une_sieste: Sa -> VPa -> Sinf; (* reduced clause - infinitive clause *)



G_fait: Sa -> VPa -> N->N->S;
G_fait_sws: Sa -> VPa -> N ->Sws;    (* clause lacking a subject*)
G_faire: Sa -> VPa -> N ->Sinf; (* reduced clause - infinitive clause *)
G_fasse_subjunctive: Sa -> VPa -> N -> N->S;
G_avoir_fait: Sa -> VPa -> N->Sinf; (* reduced clause - infinitive clause *)

G_vraiment : VPa -> VPa;
```

```
G_a : VPa;

AnchorT: S->T;

end



signature  TAGDER =        (* The Object Vocabulary for TAG derivation Trees *)

Sa, N, Na_d, Na, VPa, S: type;

c_jean, c_marie : N;
      c_petit_dejeuner : Na_d->Na->N;
      c_le:Na_d;
      c_delicieux:Na->Na;


   c_fait_une_sieste, c_fasse_une_sieste, c_etre_recomepnse_par, c_passe_laspirateur  :Sa -> VPa -> N -> S;
c_recomepnse :Sa -> VPa -> N -> N -> S;



Concat : S -> S -> S;
c_avantque : S -> S -> S;
c_avantde :  S -> S -> S;

c_pourque : S -> S ->  S;
c_pour :  S -> S -> S;

c_apres, c_apresque: S -> S -> S;


c_disc_ensuite : S -> S  -> S;
c_disc_auparavant : S -> S  -> S;



c_ensuite_s: Sa;

c_ensuite_v: VPa;


(* Dummy element to specify the end of adjunctions *)
I_vp : VPa;
I_s : Sa;
   I_n : Na;



c_fait: Sa -> VPa -> N->N->S;
c_faire: Sa -> VPa -> N->S;
c_fasse_subjunctive: Sa -> VPa -> N -> N->S;

c_faire_une_sieste: Sa -> VPa ->S;


c_fasse_une_sieste_subjunctive: Sa -> VPa -> N->S;

c_recomepnse_subjunctive: Sa -> VPa -> N -> N -> S;


c_avoir_passe_laspirateur: Sa -> VPa -> S;

c_avoir_fait_une_sieste: Sa -> VPa -> S;

c_avoir_fait: Sa -> VPa -> N->S;


c_passe_laspirateur_subjunctive : Sa -> VPa -> N->S;
c_passer_laspirateur_inf : Sa -> VPa ->S;

c_vraiment : VPa -> VPa;

c_a : VPa;


end



lexicon GTAGtoTAG (Gderivation) : TAGDER =     (* Interpreting g-derivation trees to TAG derivation Trees *)


T :=S;

S, Sinf, Sinf, S :=S;
```

```
N:=N;
Na_d:=Na_d;
Na:=Na;

Sws:= N->S;

Sa:=Sa;  VPa := VPa;

I_vp:= I_vp;
I_s := I_s;

I_n := I_n;

G_marie:=c_marie;
G_jean:=c_jean;
G_petit_dejeuner:= c_petit_dejeuner;
G_le:=c_le;
G_delicieux :=  c_delicieux;

G_ensuiteSS := lambda s1 s2. c_disc_ensuite s1 s2;
G_ensuiteST := lambda s1 s2. c_disc_ensuite s1 s2;
G_ensuiteTS := lambda s1 s2. c_disc_ensuite s1 s2;
G_ensuiteTT := lambda s1 s2. c_disc_ensuite s1 s2 ;

G_auparavantSS, G_auparavantTT, G_auparavantTS, G_auparavantST :=lambda s1 s2. c_disc_auparavant s1 s2 ;



G_avant_c := lambda s1 s2. c_avantque s1 s2  ;

G_avant_r := lambda np s1 s2. c_avantde (s1 np) s2;

G_pour_c := lambda s1 s2. c_pourque s1 s2 ;

G_pour_r := lambda np s1 s2. c_pour (s1 np) s2;

G_apres_r := lambda np s1 s2. c_apres (s1 np) s2;

G_apres_c := lambda s1 s2. c_apresque s1 s2  ;

G_recomepnse :=  c_recomepnse ;
G_recomepnse_subjunctive := c_recomepnse_subjunctive;


G_etre_recomepnse_par :=  c_etre_recomepnse_par ;

G_passe_laspirateur_sws :=   c_passe_laspirateur ;


G_fasse_une_sieste_subjunctive:= c_fasse_une_sieste;

G_fait_une_sieste := c_fait_une_sieste;



AnchorT := lambda  S. S ;

G_avoir_fait_une_sieste := c_avoir_fait_une_sieste;
G_avoir_passe_laspirateur := c_avoir_passe_laspirateur;
G_fasse_subjunctive := c_fasse_subjunctive;
G_faire := c_faire;
G_fait_sws := c_fait;
G_faire_une_sieste := c_faire_une_sieste;
G_fait := c_fait;
G_fait_une_sieste_sws := c_fait_une_sieste;
G_avoir_fait := c_avoir_fait;

G_passe_laspirateur := c_passe_laspirateur;
G_passer_laspirateur_inf := c_passer_laspirateur_inf;
G_passe_laspirateur_subjunctive := c_passe_laspirateur_subjunctive;

G_vraiment :=  c_vraiment;

G_a :=  c_a;


end



signature DerivedTrees =       (* The Object Vocabulary for Derived Trees *)
tree :type;

NP1, N1, VP1, V1, Adv1, P1, C, Det1, Punct1 : tree -> tree;
NP2, N2, S2, VP2, V2, PP2, Adv2, D2 :tree -> tree -> tree;
S3: tree -> tree ->tree -> tree;

laspirateur, l, le, aspirateur, passe, etre, recomepnse, avant, de, que, ensuite, auparavant,
      Ensuite, Auparavant, jean, marie, fred, paul, par, une, sieste, fait, faire, fasse, avoir,
      apres, pour, petit_dejeuner, delicieux, COMMA, DOT, PRO, epsilon, a, vraiment : tree;
```

```
end

lexicon Syntax (TAGDER) : DerivedTrees =     (* Translating TAG derivation trees to derived trees *)

S, N := tree;

Sa, VPa, Na_d, Na := tree -> tree;

I_vp,I_s, I_n := lambda x.x;

c_avantque := lambda s1. lambda s2.  S2 s1 (S2 (PP2 (Adv1 avant) (C que)) s2);

c_avantde := lambda s1. lambda s2.  S2 s1 (S2 (PP2 (Adv1 avant) (C de)) s2);

c_pourque := lambda s1. lambda s2.  S2 s1 (S2 (PP2 (Adv1 pour) (C que)) s2);

c_pour := lambda s1. lambda s2.  S2 s1 (S2 (PP2 (Adv1 pour) (C epsilon)) s2);


c_apresque:=  lambda s1. lambda s2.  S2 s1 (S2 (PP2 (Adv1 apres) (C que)) s2);
c_apres := lambda s1. lambda s2.  S2 s1 (S2 (PP2 (Adv1 apres) (C epsilon)) s2);

c_jean := NP1 jean;

c_marie := NP1 marie;

c_le := lambda n . N2 le n ;

c_petit_dejeuner := lambda d a.d (a(N1 petit_dejeuner));

c_delicieux := lambda a n . a (N2 delicieux n);

Concat :=  lambda s1. lambda s2. S2 s1 (S2 (Punct1 DOT) s2);



c_disc_ensuite := lambda s1. lambda s2. S3 s1 (Punct1 DOT) (S2 (Adv2 Ensuite (Punct1 COMMA) ) s2);
c_disc_auparavant :=  lambda s1. lambda s2. S3 s1 (Punct1 DOT) (S2  (Adv2  Auparavant (Punct1 COMMA) ) s2);




c_ensuite_s:= lambda x. S2 (Adv2 Ensuite COMMA) x;

c_ensuite_v:= lambda x. S2 x (Adv2 Ensuite COMMA);


(* Dummy element to specify the end of adjunctions *)
I_s, I_vp :=lambda x.x;



c_fait:= lambda sa va np0 np1 .sa (S2 np0 (va (VP2 fait np1)));
c_faire:=  lambda sa va np1 .sa (S2 (NP1 PRO) (va (VP2 faire np1)));
c_fasse_subjunctive:=lambda sa va np0 np1 .sa (S2 np0 (va (VP2 fasse np1)));



c_passe_laspirateur := lambda s a np0. s (S2 np0 (a (VP2 (V1 passe) (NP2 (Det1 l) (N1 aspirateur)) )));

c_passe_laspirateur_subjunctive := lambda s a np0. s (S2 np0 (a (VP2 (V1 passe) (NP2 (Det1 l) (N1 aspirateur)) )));

c_passer_laspirateur_inf :=lambda s a. s (S2 (NP1 PRO) (a (VP2 (V1 faire) (NP2 (Det1 l) (N1 aspirateur) )  )));


c_recomepnse := lambda s a np1 np0 .s (S2 np0 (a (VP2 recomepnse np1)));
c_recomepnse_subjunctive:=  lambda s a np1 np0 .s (S2 np0 (a (VP2 recomepnse np1)));

c_etre_recomepnse_par := lambda s a np1 .s (S2 (NP1 PRO) (a (VP2 (V2 etre (V1 recomepnse)) (PP2 (P1 par) np1))));



c_fasse_une_sieste := lambda s a np0. s (S2 np0 (a (VP2 (V1 fasse) (NP2 (Det1 une) (N1 sieste) )  )));

c_fait_une_sieste :=  lambda s a np0. s (S2 np0 (a (VP2 (V1 fait) (NP2 (Det1 une) (N1 sieste) )  )));

c_faire_une_sieste:= lambda s a. s (S2 (NP1 PRO) (a (VP2 (V1 faire) (NP2 (Det1 une) (N1 sieste) )  )));


c_fasse_une_sieste_subjunctive:= lambda s a np0. s (S2 np0 (a (VP2 (V1 fasse) (NP2 (Det1 une) (N1 sieste) )  )));
```

```
c_avoir_passe_laspirateur:=lambda s a. s (S2 (NP1 PRO) (a (VP2 (VP2 (V1 avoir) (V1 passe)) (NP2 (Det1 l) (N1 aspirateur) ) )));

c_avoir_fait_une_sieste:=lambda s a. s (S2 (NP1 PRO) (a (VP2 (VP2 (V1 avoir) (V1 fait)) (NP2 (Det1 une) (N1 sieste) ) )));

c_avoir_fait:=lambda s a np1. s (S2 (NP1 PRO) (a (VP2 (VP2 (V1 avoir) (V1 fait)) np1 )));



c_vraiment := lambda va x. va (VP2 vraiment x);
c_a :=  lambda  x. VP2 a x;


end


signature strings =   (* The Object Vocabulary for Encoding Surface Realizations *)

string: type;

(* we can define infix and prefix symbols. Note that as for now, the length of symbols can only be 1 *)

infix + : string -> string -> string;

laspirateur, l, le, aspirateur, passe, etre, recomepnse, avant, de, que, ensuite,
      auparavant, Ensuite, Auparavant, Jean, Marie, Fred, Paul, par, une, sieste, fait,
      faire, fasse, avoir, apres, pour, petit, dejeuner, delicieux, COMMA, DOT, PRO, epsilon, a, vraiment : string;

end



lexicon Yield(DerivedTrees) : strings =      (* Interpreting derived trees as surface strings *)

(* So every tree result in a string *)
tree := string;

NP1, N1, VP1, V1, Adv1, P1, C, Det1, Punct1 := lambda f.f;
NP2, N2, S2, VP2, V2, PP2, Adv2, D2 :=lambda f g . f + g;
S3 := lambda f g h. f + g +h;
faire := faire;
Ensuite :=  Ensuite ;
l := l ;
   le := le;
epsilon :=  epsilon ;
fait :=  fait ;
auparavant := auparavant  ;
PRO := epsilon  ;
sieste := sieste  ;
ensuite :=  ensuite ;
DOT := DOT ;
une :=  une ;
que :=  que ;
COMMA :=  COMMA ;
par := par  ;
de :=  de ;
petit_dejeuner := petit+dejeuner  ;
paul := Paul;
avant :=  avant ;
pour := pour;
fred := Fred ;
recomepnse := recomepnse  ;
apres := apres  ;
marie := Marie  ;
etre :=  etre ;
avoir := avoir  ;
jean := Jean  ;
passe := passe  ;
fasse := fasse  ;
Auparavant :=  Auparavant ;
aspirateur := aspirateur  ;
laspirateur :=  l+aspirateur ;
   delicieux := delicieux;
   a := a;
   vraiment:=vraiment;
end



signature semantics =

(* We define the usual types *)

e,t:type;
qnp = (e => t) -> t : type;


(* Then few non logical-constants *)

SUCC, GOAL, CAUSE : t->t->t;
sleep, breakfast : e->t;
```

```
REWARD, LOVE, MAKE : e -> e -> t;
j, m :e;
KINDLY, really: t -> t;
seem : (e -> t) -> e -> t;
new,big,black, delicious: e ->t;
claim,say,think : e -> t -> t;
      NAP, VACUUM : e->t;


(* And finally, here are the logical constants *)

infix & : t -> t -> t;
infix > : t -> t -> t;
binder All : (e=>t) -> t;
binder Ex : (e=>t) -> t;

end


lexicon lexsemantics (Gderivation) : semantics =    (* The semantic translations of the abstract constants *)

S, T := t;
Sinf, Sws := qnp -> t;
   N  := qnp;
   N  := (e => t) -> t;
   Na_d:= (e => t) -> (e => t) -> t;
   Na :=(e =>t) -> (e =>t);
VPa := (e => t) -> (e => t);
   Sa := t -> t;
   I_vp,I_s,I_n := lambda x.x;

      G_jean := lambda P.P j;
      G_marie := lambda P.P m;
      G_petit_dejeuner := lambda d a . d (a (Lambda x. breakfast x));
      G_le:=lambda n.lambda P.Ex x. (n x) & (P x);
      G_delicieux := lambda a n . a (Lambda x.(delicious x)&(n x));


G_ensuiteSS, G_ensuiteTT, G_ensuiteTS, G_ensuiteST := lambda s1 s2. SUCC s1 s2;

G_auparavantSS, G_auparavantTT, G_auparavantTS, G_auparavantST := lambda s1 s2. SUCC s2 s1;



G_avant_c :=  lambda s1. lambda s2. SUCC s1 s2;

G_avant_r := lambda S. lambda s1. lambda s2. S(Lambda x. (SUCC (s1 (lambda P. P(x)))) (s2 (lambda P. P(x))));


G_apres_c :=  lambda s1. lambda s2. SUCC s2 s1;

G_apres_r := lambda S. lambda s1. lambda s2. S(Lambda x. (SUCC (s2 (lambda P. P(x)))) (s1 (lambda P. P(x))));


G_pour_c :=  lambda s1. lambda s2. GOAL s1 s2;

G_pour_r := lambda S. lambda s1. lambda s2. S(Lambda x. (GOAL (s1 (lambda P. P(x)))) (s2 (lambda P. P(x))));

G_recomepnse, G_recomepnse_subjunctive := lambda s a  S O . s(S(a(Lambda x.O(Lambda y.(REWARD x y)))));


G_etre_recomepnse_par := lambda s a S O. s(S(a(Lambda x.O(Lambda y.(REWARD x y)))));


G_passe_laspirateur_sws, G_passer_laspirateur_inf, G_avoir_passe_laspirateur := lambda s a S.s(S(a(Lambda x.(VACUUM x))));

G_passe_laspirateur := lambda s a S.s(S(a(Lambda x.(VACUUM x))));
G_passe_laspirateur_subjunctive:= lambda s a S.s(S(a(Lambda x.(VACUUM x))));


G_fait_une_sieste_sws, G_avoir_fait_une_sieste, G_faire_une_sieste := lambda s a S.s(S(a(Lambda x.(NAP x))));

G_fait_une_sieste,  G_fasse_une_sieste_subjunctive := lambda s a S.s(S(a(Lambda x.(NAP x))));


G_fasse_subjunctive :=  lambda s a  S O. s(S(a(Lambda x.O(Lambda y.(MAKE x y)))));


G_fait_sws, G_faire, G_avoir_fait:= lambda s a  O S. s(S(a(Lambda x.O(Lambda y.(MAKE x y)))));
G_fait:= lambda s a S O . s(S(a(Lambda x.O(Lambda y.(MAKE x y)))));

G_vraiment := lambda vp r. vp (Lambda x. really (r x));
G_a := lambda x.x;


AnchorT := lambda x.x;


end
```

```
lexicon lexsyntax  = Syntax << GTAGtoTAG

lexicon lexyield =  Yield << lexsyntax
```

Appendix C

Encoding Clause-Medial Connectives

C.1 Examples

In order to generate the text of Example 2.1 on page 246, we translate the term encoding its g-derivation tree with the lexyield lexicon.

```
lexyield analyse
G_ensuite_m
 (AnchorT
  (G_pour_r G_jean (G_passe_laspirateur_sws I_s G_a)
             (G_etre_recomepnse_par I_s  I_vp G_marie))
 )
 (G_fait_une_sieste I_s G_a G_jean)
:T;
```

By interpreting the same term with the *lexsemantics* lexicon, one obtain the following semantic representation:

```
SUCC
    (GOAL
       (VACUUM j) (REWARD m j)
    )
    (SUCC
       (Ex x. ((delicious x) & (breakfast x)) & (MAKE x j))
       (NAP j)
    )
: t
```

C.2 ACG Signatures and Lexicons: Clause-Medial Connectives

```
signature Gderivation =            (* The Abstract Vocabulary for Encoding G-derivation Trees *)
Sa, N, Na_d, Na, VPa,  S, T, Sws, Sinf: type;
   I_vp : VPa;
   I_s : Sa;
   I_n : Na;


   G_jean, G_marie :N;
   G_petit_dejeuner:  Na_d -> Na -> N;
   G_le:Na_d;
```

```
   G_delicieux : Na -> Na;

      G_ensuiteTT : T -> T -> T;     (* constant encoding adverbial :ensuite *)
      G_ensuite_m : T -> S -> T;     (* constant encoding clause-medial adverbial :ensuite *)

      G_auparavantTT : T -> T -> T;  (* constant encoding adverbial :auparavant *)
      G_auparavant_m : T -> S -> T;     (* constant encoding clause-medial adverbial :ensuite *)


G_avant_c: S -> S ->S;           (* canonical g-derivation tree avant *)
G_avant_r: N -> Sws -> Sinf ->S;    (* reduced conjunction g-derivation tree avant *)

G_pour_c: S -> S ->S;            (* canonical g-derivation tree pour *)
G_pour_r: N -> Sws -> Sinf ->S;     (* reduced conjunction g-derivation tree pour *)

G_apres_c: S -> S ->S;           (* canonical g-derivation tree apres *)
G_apres_r: N -> Sws -> Sinf ->S;    (* reduced conjunction g-derivation tree apres *)

G_recomepnse: Sa -> VPa -> N -> N -> S;
G_recomepnse_subjunctive: Sa -> VPa -> N -> N -> S;

G_etre_recomepnse_par: Sa -> VPa -> N -> Sinf;  (* reduced clause - infinitive clause *)

G_passe_laspirateur: Sa -> VPa -> N->S;
G_passe_laspirateur_subjunctive: Sa -> VPa -> N->S;
G_passe_laspirateur_sws: Sa -> VPa -> Sws;     (* clause lacking a subject*)
G_passer_laspirateur_inf: Sa -> VPa -> Sinf;   (* reduced clause - infinitive clause *)
G_avoir_passe_laspirateur: Sa -> VPa -> Sinf;  (* reduced clause - infinitive clause *)


G_fait_une_sieste: Sa -> VPa -> N->S;
G_fasse_une_sieste_subjunctive: Sa -> VPa -> N->S;
G_fait_une_sieste_sws: Sa -> VPa ->Sws;         (* clause lacking a subject*)
G_faire_une_sieste: Sa -> VPa -> Sinf;       (* reduced clause - infinitive clause *)
G_avoir_fait_une_sieste: Sa -> VPa -> Sinf;       (* reduced clause - infinitive clause *)



G_fait: Sa -> VPa -> N->N->S;
G_fait_sws: Sa -> VPa -> N ->Sws;           (* clause lacking a subject*)
G_faire: Sa -> VPa -> N ->Sinf;         (* reduced clause - infinitive clause *)
G_fasse_subjunctive: Sa -> VPa -> N -> N->S;
G_avoir_fait: Sa -> VPa -> N->Sinf;          (* reduced clause - infinitive clause *)

G_vraiment : VPa -> VPa;
G_a : VPa;

AnchorT: S->T;

end


signature gderivationtrees =  (* An object Vocabulary for Encoding G-derivation Trees *)
Sa, N, Na_d, Na, VPa,  S, T : type;
I_vp : VPa;
I_s : Sa;
   I_n : Na;


   g_jean, g_marie :N;
   g_petit_dejeuner:  Na_d -> Na -> N;
   g_le:Na_d;
   g_delicieux : Na -> Na;

      g_ensuiteTT : T -> T -> T;     (* constant encoding adverbial :ensuite *)
      g_ensuite_m : T -> S -> T;     (* constant encoding clause-medial adverbial :ensuite *)


      g_auparavantTT : T -> T -> T;  (* constant encoding adverbial :auparavant *)
      g_auparavant_m : T -> S -> T;     (* constant encoding clause-medial adverbial :ensuite *)


g_avant_c: S -> S ->S;           (* canonical g-derivation tree avant *)
g_avant_r: S -> S ->S;  (* reduced conjunction g-derivation tree avant *)

g_pour_c: S -> S ->S;          (* canonical g-derivation tree pour *)
g_pour_r: S -> S ->S;   (* reduced conjunction g-derivation tree pour *)

g_apres_c: S -> S ->S;     (* canonical g-derivation tree apres *)
g_apres_r: S -> S ->S;  (* reduced conjunction g-derivation tree apres *)

g_recomepnse: Sa -> VPa -> N -> N -> S;
g_recomepnse_subjunctive: Sa -> VPa -> N -> N -> S;

g_etre_recomepnse_par: Sa -> VPa -> N -> S;  (* reduced clause - infinitive clause *)

g_passe_laspirateur: Sa -> VPa -> N->S;
g_passe_laspirateur_subjunctive: Sa -> VPa -> N->S;
g_passer_laspirateur_inf: Sa -> VPa -> S;    (* reduced clause - infinitive clause *)
g_avoir_passe_laspirateur: Sa -> VPa -> S;   (* reduced clause - infinitive clause *)
```

```
g_fait_une_sieste: Sa -> VPa -> N->S;
g_fasse_une_sieste_subjunctive: Sa -> VPa -> N->S;
g_faire_une_sieste: Sa -> VPa -> S; (* reduced clause - infinitive clause *)
g_avoir_fait_une_sieste: Sa -> VPa -> S; (* reduced clause - infinitive clause *)



g_fait: Sa -> VPa -> N->N->S;
g_faire: Sa -> VPa -> N ->S; (* reduced clause - infinitive clause *)
g_fasse_subjunctive: Sa -> VPa -> N -> N->S;
g_avoir_fait: Sa -> VPa -> N->S; (* reduced clause - infinitive clause *)

g_vraiment : VPa -> VPa;
g_a : VPa;

AnchorT: S->T;

end

lexicon LGg (Gderivation) : gderivationtrees =     (* Interpreting G-derivations to g-derivation trees *)


T :=T;


S, Sinf :=S;

N:=N;
Na_d:=Na_d;
Na:=Na;

Sws:= N->S;

Sa:=Sa;  VPa := VPa;

I_vp:= I_vp;
I_s := I_s;

I_n := I_n;

G_marie:=g_marie;
G_jean:=g_jean;
G_petit_dejeuner:= g_petit_dejeuner;
G_le:=g_le;
G_delicieux :=  g_delicieux;




G_avant_c := lambda s1 s2. g_avant_c s1 s2  ;

G_avant_r := lambda np s1 s2. g_avant_r (s1 np) s2;

G_pour_c := lambda s1 s2. g_pour_c s1 s2 ;

G_pour_r := lambda np s1 s2. g_pour_r (s1 np) s2;

G_apres_r := lambda np s1 s2. g_apres_r (s1 np) s2;

G_apres_c := lambda s1 s2. g_apres_c s1 s2  ;

G_recomepnse :=  g_recomepnse ;
G_recomepnse_subjunctive := g_recomepnse_subjunctive;


G_etre_recomepnse_par :=  g_etre_recomepnse_par ;

G_passe_laspirateur_sws := lambda s a. lambda  subj.  g_passe_laspirateur  s a  subj;
(* the argument modeling the  subject of 'the clause missing the subject'  is the last abstraction (lambda  subj)*)


G_fasse_une_sieste_subjunctive:= g_fasse_une_sieste_subjunctive;

G_fait_une_sieste := g_fait_une_sieste;



AnchorT := AnchorT ;

   G_avoir_fait_une_sieste := g_avoir_fait_une_sieste;
   G_avoir_passe_laspirateur := g_avoir_passe_laspirateur;
   G_fasse_subjunctive := g_fasse_subjunctive;
   G_faire := g_faire;

   G_fait_sws := lambda s a obj. lambda subj. g_fait  s a  subj obj;
   (* the argument modeling the  subject of 'the clause missing the subject'  is the last abstraction (lambda  subj)*)

   G_faire_une_sieste := g_faire_une_sieste;
   G_fait := g_fait;
```

```
   G_fait_une_sieste_sws := lambda s a  subj.  g_fait_une_sieste  s a  subj;
   (* the argument modeling the  subject of 'the clause missing the subject'  is the last abstraction (lambda  subj)*)

   G_avoir_fait := g_avoir_fait;

G_passe_laspirateur := g_passe_laspirateur;
G_passer_laspirateur_inf := g_passer_laspirateur_inf;
G_passe_laspirateur_subjunctive := g_passe_laspirateur_subjunctive;

G_vraiment :=  g_vraiment;

G_a :=  g_a;

 G_ensuiteTT :=  g_ensuiteTT;
G_auparavantTT :=  g_auparavantTT;

 G_ensuite_m := g_ensuite_m;
 G_auparavant_m := g_auparavant_m;

end

signature  TAGDER =        (* The Object Vocabulary for TAG derivation Trees *)

   Sa, N, Na_d, Na, VPa, S: type;

      c_jean, c_marie : N;
      c_petit_dejeuner : Na_d->Na->N;
      c_le:Na_d;
      c_delicieux:Na->Na;


   c_fait_une_sieste, c_fasse_une_sieste, c_etre_recomepnse_par, c_passe_laspirateur  :Sa -> VPa -> N -> S;
   c_recomepnse :Sa -> VPa -> N -> N -> S;



Concat : S -> S -> S;
c_avantque : S -> S -> S;
c_avantde :  S -> S -> S;

c_pourque : S -> S ->  S;
c_pour :  S -> S -> S;

c_apres, c_apresque: S -> S -> S;


c_disc_ensuite : S -> S  -> S;
c_disc_auparavant : S -> S  -> S;



c_ensuite_s: Sa;

c_ensuite_v: VPa;
c_auparavant_v: VPa;



(* Dummy element to specify the end of adjunctions *)
   I_vp : VPa;
   I_s : Sa;
   I_n : Na;



c_fait: Sa -> VPa -> N->N->S;
c_faire: Sa -> VPa -> N->S;
c_fasse_subjunctive: Sa -> VPa -> N -> N->S;

c_faire_une_sieste: Sa -> VPa ->S;


c_fasse_une_sieste_subjunctive: Sa -> VPa -> N->S;

c_recomepnse_subjunctive: Sa -> VPa -> N -> N -> S;


c_avoir_passe_laspirateur: Sa -> VPa -> S;

c_avoir_fait_une_sieste: Sa -> VPa -> S;

c_avoir_fait: Sa -> VPa -> N->S;


c_passe_laspirateur_subjunctive : Sa -> VPa -> N->S;
c_passer_laspirateur_inf : Sa -> VPa ->S;

c_vraiment : VPa -> VPa;
```

```
c_a : VPa;

APP: VPa -> VPa -> VPa;

end
lexicon gdertoTAGder (gderivationtrees) : TAGDER =     (* Interpreting g-derivation trees as TAG derivation Trees *)

T :=S;



N:=N;
Na_d:=Na_d;
Na:=Na;


Sa:=Sa;

S:= VPa -> S;  VPa := VPa -> VPa;

I_vp:= lambda x.x;
I_s := I_s;

I_n := I_n;

g_marie:=c_marie;
g_jean:=c_jean;
g_petit_dejeuner:= c_petit_dejeuner;
g_le:=c_le;
g_delicieux :=  c_delicieux;

g_ensuiteTT := lambda s1 s2. c_disc_ensuite s1 s2 ;

g_auparavantTT :=lambda s1 s2. c_disc_auparavant s1 s2 ;

 g_ensuite_m := lambda s1 s2. Concat s1 (s2 c_ensuite_v);
 g_auparavant_m := lambda s1 s2. Concat s1 (s2 c_auparavant_v);

g_avant_c := lambda s1 s2 mod. c_avantque (s1 mod) (s2 I_vp)  ;

g_avant_r := lambda s1 s2. lambda mod. c_avantde (s1 mod) (s2 I_vp);

g_pour_c := lambda s1 s2 mod. c_pourque (s1 mod) (s2 I_vp);

g_pour_r := lambda  s1 s2 mod. c_pour (s1 mod) (s2 I_vp);

g_apres_r := lambda  s1 s2 mod. c_apres (s1 mod) (s2 I_vp);

g_apres_c := lambda s1 s2 mod. c_apresque (s1 mod) (s2 I_vp);

g_recomepnse :=  lambda s a subj obj. lambda mod. c_recomepnse s (a mod) subj obj;
g_recomepnse_subjunctive := lambda s a subj obj. lambda mod. c_recomepnse_subjunctive s (a mod) subj obj;


g_etre_recomepnse_par :=  lambda s a subj. lambda mod. c_etre_recomepnse_par s (a mod)  subj;

g_fasse_une_sieste_subjunctive:= lambda s a subj.  lambda mod. c_fasse_une_sieste s (a mod) subj;


g_fait_une_sieste := lambda s a subj. lambda mod. c_fait_une_sieste s (a mod) subj;




AnchorT := lambda  S. S I_vp ;

   g_avoir_fait_une_sieste := lambda s a. lambda mod. c_avoir_fait_une_sieste s (a mod) ;
   g_avoir_passe_laspirateur := lambda s a. lambda mod. c_avoir_passe_laspirateur s (a mod) ;
   g_fasse_subjunctive :=  lambda s a subj obj. lambda mod. c_fasse_subjunctive s (a mod) subj obj;
   g_faire := lambda s a obj. lambda mod.  c_faire s (a mod) obj ;
   g_faire_une_sieste := lambda s a. lambda mod.  c_faire_une_sieste s (a mod) ;
   g_fait := lambda s a subj obj. lambda mod. c_fait s (a mod) subj obj;
   g_avoir_fait := lambda s a obj. lambda mod.  c_avoir_fait s (a mod) obj ;

g_passe_laspirateur := lambda s a subj. lambda mod. c_passe_laspirateur s (a mod) subj ;
g_passer_laspirateur_inf := lambda s a. lambda mod. c_passer_laspirateur_inf  s (a mod) ;
g_passe_laspirateur_subjunctive :=  lambda s a subj. lambda mod. c_passe_laspirateur_subjunctive s (a mod) subj ;


g_a := lambda x. APP c_a x;



g_vraiment := lambda A. lambda x. A (c_vraiment x) ;


end
```

```
signature DerivedTrees =              (* The Object Vocabulary for Derived Trees *)
   tree :type;

   NP1, N1, VP1, V1, Adv1, P1, C, Det1, Punct1 : tree -> tree;
   NP2, N2, S2, VP2, V2, PP2, Adv2, D2 :tree -> tree -> tree;
   S3: tree -> tree ->tree -> tree;

   laspirateur, l, le, aspirateur, passe, etre, recomepnse, avant, de, que, ensuite, auparavant,
      Ensuite, Auparavant, jean, marie, fred, paul, par, une, sieste, fait, faire, fasse, avoir,
      apres, pour, petit_dejeuner, delicieux, COMMA, DOT, PRO, epsilon, a, vraiment : tree;


end

lexicon Syntax (TAGDER) : DerivedTrees =     (* Translating TAG derivation trees to derived trees *)

S, N := tree;

Sa, VPa, Na_d, Na := tree -> tree;

I_vp,I_s, I_n := lambda x.x;

c_avantque := lambda s1. lambda s2.  S2 s1 (S2 (PP2 (Adv1 avant) (C que)) s2);

c_avantde := lambda s1. lambda s2.  S2 s1 (S2 (PP2 (Adv1 avant) (C de)) s2);

c_pourque := lambda s1. lambda s2.  S2 s1 (S2 (PP2 (Adv1 pour) (C que)) s2);

c_pour := lambda s1. lambda s2.  S2 s1 (S2 (PP2 (Adv1 pour) (C epsilon)) s2);


c_apresque:=  lambda s1. lambda s2.  S2 s1 (S2 (PP2 (Adv1 apres) (C que)) s2);
c_apres := lambda s1. lambda s2.  S2 s1 (S2 (PP2 (Adv1 apres) (C epsilon)) s2);

c_jean := NP1 jean;

c_marie := NP1 marie;

c_le := lambda n . N2 le n ;

c_petit_dejeuner := lambda d a.d (a(N1 petit_dejeuner));

c_delicieux := lambda a n . a (N2 delicieux n);

Concat :=  lambda s1. lambda s2. S2 s1 (S2 (Punct1 DOT) s2);



c_disc_ensuite := lambda s1. lambda s2. S3 s1 (Punct1 DOT) (S2 (Adv2 Ensuite (Punct1 COMMA) ) s2);
c_disc_auparavant :=  lambda s1. lambda s2. S3 s1 (Punct1 DOT) (S2  (Adv2  Auparavant (Punct1 COMMA) ) s2);






c_ensuite_s:= lambda x. S2 (Adv2 Ensuite COMMA) x;

c_ensuite_v:= lambda x. VP2 (Adv1 ensuite) x;

c_auparavant_v := lambda x. VP2 (Adv1  auparavant) x;


(* Dummy element to specify the end of adjunctions *)
I_s, I_vp :=lambda x.x;




c_fait:= lambda sa va np0 np1 .sa (S2 np0 (va (VP2 fait np1)));
c_faire:=  lambda sa va np1 .sa (S2 (NP1 PRO) (va (VP2 faire np1)));
c_fasse_subjunctive:=lambda sa va np0 np1 .sa (S2 np0 (va (VP2 fasse np1)));




c_passe_laspirateur := lambda s a np0. s (S2 np0 (a (VP2 (V1 passe) (NP2 (Det1 l) (N1 aspirateur)) )));

c_passe_laspirateur_subjunctive := lambda s a np0. s (S2 np0 (a (VP2 (V1 passe) (NP2 (Det1 l) (N1 aspirateur)) )));

c_passer_laspirateur_inf :=lambda s a. s (S2 (NP1 PRO) (a (VP2 (V1 faire) (NP2 (Det1 l) (N1 aspirateur) )  )));


c_recomepnse := lambda s a np1 np0 .s (S2 np0 (a (VP2 recomepnse np1)));
c_recomepnse_subjunctive:=  lambda s a np1 np0 .s (S2 np0 (a (VP2 recomepnse np1)));

c_etre_recomepnse_par := lambda s a np1 .s (S2 (NP1 PRO) (a (VP2 (V2 etre (V1 recomepnse)) (PP2 (P1 par) np1))));
```

```
c_fasse_une_sieste := lambda s a np0. s (S2 np0 (a (VP2 (V1 fasse) (NP2 (Det1 une) (N1 sieste) ) )));

c_fait_une_sieste :=  lambda s a np0. s (S2 np0 (a (VP2 (V1 fait) (NP2 (Det1 une) (N1 sieste) ) )));

c_faire_une_sieste:= lambda s a. s (S2 (NP1 PRO) (a (VP2 (V1 faire) (NP2 (Det1 une) (N1 sieste) ) )));


c_fasse_une_sieste_subjunctive:= lambda s a np0. s (S2 np0 (a (VP2 (V1 fasse) (NP2 (Det1 une) (N1 sieste) ) )));


c_avoir_passe_laspirateur:=lambda s a. s (S2 (NP1 PRO) (a (VP2 (VP2 (V1 avoir) (V1 passe)) (NP2 (Det1 l) (N1 aspirateur) ) )));

c_avoir_fait_une_sieste:=lambda s a. s (S2 (NP1 PRO) (a (VP2 (VP2 (V1 avoir) (V1 fait)) (NP2 (Det1 une) (N1 sieste) ) )));

c_avoir_fait:=lambda s a np1. s (S2 (NP1 PRO) (a (VP2 (VP2 (V1 avoir) (V1 fait)) np1 )));



c_vraiment := lambda va x. va (VP2 vraiment x);
c_a :=  lambda  x. VP2 a x;

APP := lambda a1 a2 x. a1 (a2 x);
end


signature strings =   (* The Object Vocabulary for Encoding Surface Realizations *)

   string: type;

   (* we can define infix and prefix symbols. Note that as for now, the length of symbols can only be 1 *)

   infix + : string -> string -> string;

   laspirateur, l, le, aspirateur, passe, etre, recomepnse, avant, de, que, ensuite,
      auparavant, Ensuite, Auparavant, Jean, Marie, Fred, Paul, par, une, sieste, fait,
      faire, fasse, avoir, apres, pour, petit, dejeuner, delicieux, COMMA, DOT, PRO, epsilon, a, vraiment : string;

end


lexicon Yield(DerivedTrees) : strings =      (* Interpreting derived trees as surface strings *)

   (* So every tree result in a string *)
   tree := string;

   NP1, N1, VP1, V1, Adv1, P1, C, Det1, Punct1 := lambda f.f;
   NP2, N2, S2, VP2, V2, PP2, Adv2, D2 :=lambda f g . f + g;
   S3 := lambda f g h. f + g +h;
   faire := faire;
   Ensuite :=  Ensuite ;
   l := l ;
   le := le;
   epsilon :=  epsilon ;
   fait :=  fait ;
   auparavant := auparavant  ;
   PRO := epsilon  ;
   sieste := sieste  ;
   ensuite :=  ensuite ;
   DOT := DOT ;
   une :=  une ;
   que :=  que ;
   COMMA :=  COMMA ;
   par := par  ;
   de :=  de ;
   petit_dejeuner := petit+dejeuner  ;
   paul := Paul;
   avant :=  avant ;
   pour := pour;
   fred := Fred ;
   recomepnse := recomepnse  ;
   apres := apres  ;
   marie := Marie  ;
   etre :=  etre ;
   avoir := avoir  ;
   jean := Jean  ;
   passe := passe  ;
   fasse := fasse  ;
   Auparavant :=  Auparavant ;
   aspirateur := aspirateur  ;
   laspirateur :=  l+aspirateur ;
   delicieux := delicieux;
   a := a;
   vraiment:=vraiment;
end
```

```
signature semantics =

   (* We define the usual types *)

   e,t:type;
   qnp = (e => t) -> t : type;


   (* Then few non logical-constants *)

   SUCC, GOAL, CAUSE : t->t->t;
   sleep, breakfast : e->t;
   REWARD, LOVE, MAKE : e -> e -> t;
   j, m :e;
   KINDLY, really: t -> t;
   seem : (e -> t) -> e -> t;
   new,big,black, delicious: e ->t;
   claim,say,think : e -> t -> t;
      NAP, VACUUM : e->t;


   (* And finally, here are the logical constants *)

   infix & : t -> t -> t;
   infix > : t -> t -> t;
   binder All : (e=>t) -> t;
   binder Ex : (e=>t) -> t;

end


lexicon lexsemantics (Gderivation) : semantics =    (* The semantic translations of the abstract constants *)

   S, T := t;
   Sinf, Sws := qnp -> t;
   N  := qnp;
   N  := (e => t) -> t;
   Na_d:= (e => t) -> (e => t) -> t;
   Na :=(e =>t) -> (e =>t);
   VPa := (e => t) -> (e => t);
   Sa := t -> t;
   I_vp,I_s,I_n := lambda x.x;

      G_jean := lambda P.P j;
      G_marie := lambda P.P m;
      G_petit_dejeuner := lambda d a . d (a (Lambda x. breakfast x));
      G_le:=lambda n.lambda P.Ex x. (n x) & (P x);
      G_delicieux := lambda a n . a (Lambda x.(delicious x)&(n x));


   G_ensuiteTT, G_ensuite_m := lambda s1 s2. SUCC s1 s2;
  G_auparavantTT, G_auparavant_m := lambda s1 s2. SUCC s2 s1;
(* the semantic interpretations of clause-medial and clause-initial connectives are the same - they differ only by their syntactic properties *)



G_avant_c :=  lambda s1. lambda s2. SUCC s1 s2;

G_avant_r := lambda S. lambda s1. lambda s2. S(Lambda x. (SUCC (s1 (lambda P. P(x)))) (s2 (lambda P. P(x))));


G_apres_c :=  lambda s1. lambda s2. SUCC s2 s1;

G_apres_r := lambda S. lambda s1. lambda s2. S(Lambda x. (SUCC (s2 (lambda P. P(x)))) (s1 (lambda P. P(x))));


G_pour_c :=  lambda s1. lambda s2. GOAL s1 s2;

G_pour_r := lambda S. lambda s1. lambda s2. S(Lambda x. (GOAL (s1 (lambda P. P(x)))) (s2 (lambda P. P(x))));

G_recomepnse, G_recomepnse_subjunctive := lambda s a  S O . s(S(a(Lambda x.O(Lambda y.(REWARD x y)))));


G_etre_recomepnse_par := lambda s a S O. s(S(a(Lambda x.O(Lambda y.(REWARD x y)))));


G_passe_laspirateur_sws, G_passer_laspirateur_inf, G_avoir_passe_laspirateur := lambda s a S.s(S(a(Lambda x.(VACUUM x))));
(* the argument modeling the  subject of 'the clause missing the subject' and of 'the infinitive clause' is the last abstraction *)


G_passe_laspirateur := lambda s a S.s(S(a(Lambda x.(VACUUM x))));
G_passe_laspirateur_subjunctive:= lambda s a S.s(S(a(Lambda x.(VACUUM x))));


G_fait_une_sieste_sws, G_avoir_fait_une_sieste, G_faire_une_sieste := lambda s a S.s(S(a(Lambda x.(NAP x))));

G_fait_une_sieste,  G_fasse_une_sieste_subjunctive := lambda s a S.s(S(a(Lambda x.(NAP x))));


G_fasse_subjunctive :=  lambda s a  S O. s(S(a(Lambda x.O(Lambda y.(MAKE x y)))));
```

```
G_fait_sws, G_faire, G_avoir_fait:= lambda s a  O. lambda S. s(S(a(Lambda x.O(Lambda y.(MAKE x y)))));
(* the argument modeling the  subject of 'the clause missing the subject' and of 'the infinitive clause' is the last abstraction *)

G_fait:= lambda s a S O . s(S(a(Lambda x.O(Lambda y.(MAKE x y)))));

G_vraiment := lambda vp r. vp (Lambda x. really (r x));
G_a := lambda x.x;


AnchorT := lambda x.x;


end


lexicon GTAGtoTAG =  gdertoTAGder << LGg

lexicon lexsyntax  = Syntax << GTAGtoTAG

lexicon lexyield =  Yield << lexsyntax
```

Appendix D

D-STAG as ACG codes

D.1 D-STAG Syntax, Semantics, Postposed & Preposed Connectives, Modifiers of Discourse

In Section D.3, we provide the codes for the ACG toolkit consisting of the ACG signatures and lexicons for interpreting D-STAG derivation trees into syntactic derived trees and (unlabeled) semantics are provided. In Section D.4.1, we list the commands that one can use in order to check the examples used in the ACG encoding of D-STAG after running the ACG codes given in Section D.3.

D.2 Examples

We use the following piece of code and commands in order to encode the derivation tree, and interpret the yield, syntactic, and semantic interpretations of *Fred is grumpy because he lost his keys.*

```
d_yield d_syntax discourse_semantics analyse
   d_initial_anchor_s C0
      (d_because I_DU I_DU I_DU (d_anchor_s C1 I_DU)): T;
```

We use the following piece of code and commands in order to encode the derivation tree, and interpret the yield, syntactic, and semantic interpretations of *Fred is grumpy because he lost his keys. Moreover, he failed his exam.*

```
d_yield d_syntax discourse_semantics analyse
   d_initial_anchor_s C0
      (d_because I_DU I_DU I_DU (d_anchor_s C1
         (d_punct_moreover I_DU I_DU I_DU (d_anchor_s C2 I_DU)))): T;
```

We use the following piece of code and commands in order to encode the derivation tree, and interpret the yield, syntactic, and semantic interpretations of *Fred is grumpy because he did not sleep well. He had nightmares.*

```
d_yield d_syntax discourse_semantics analyse
   d_initial_anchor_s C0
      (d_because I_DU I_DU
         (d_discourse_empty I_DU I_DU I_DU (d_anchor_s C4 I_DU))
            (d_anchor_s C3 I_DU)): T;
```

We use the following piece of code and commands in order to encode the derivation tree, and interpret the yield, syntactic, and semantic interpretations of *Fred went to the supermarket because the fridge was empty. Then, Fred went to the movies.*

```
d_yield d_syntax discourse_semantics analyse
   d_initial_anchor_s C5
      (d_because I_DU
         (d_punct_then_s I_DU I_DU I_DU
            (d_anchor_s C7 I_DU)) I_DU
               (d_anchor_s C6 I_DU)) : T;
```

We use the following piece of code and commands in order to encode the derivation tree, and interpret the yield, syntactic, and semantic interpretations of *Fred went to the supermarket because the fridge was empty. Fred* then *went to the movies.*

```
d_yield d_syntax discourse_semantics analyse
   d_initial_anchor_s C5
      (d_because I_DU
         (d_then_v I_DU I_DU I_DU
            (d_anchor_s C7 I_DU)) I_DU
               (d_anchor_s C6 I_DU)) : T;
```

We use the following piece of code and commands in order to encode the derivation tree, and interpret the yield, syntactic, and semantic interpretations of *Fred is grumpy because his wife is away this week. This shows how much he loves her.*

```
d_yield d_syntax discourse_semantics analyse
    d_initial_anchor_s C0
        (d_because
           (d_discourse_empty_comment I_DU I_DU I_DU
              (d_anchor_s C9 I_DU)) I_DU  I_DU
                 (d_anchor_s C8 I_DU)) : T;
```

We use the following piece of code and commands in order to encode the derivation tree, and interpret the yield, syntactic, and semantic interpretations of *When Fred was is Paris, he went to the Eiffel Tower. Next, he visited the Louvre.*

```
d_yield d_syntax discourse_semantics analyse
    d_initial_anchor_s C11
        (d_when
           (d_punct_next_s I_DU I_DU I_DU
              (d_anchor_s C12 I_DU)) I_DU I_DU I_DU
                 (d_anchor_s C10 I_DU)) : T;
```

We use the following piece of code and commands in order to encode the derivation tree, and interpret the yield, syntactic, and semantic interpretations of *Fred is grumpy because, for example, he failed his exam.*

```
d_yield d_syntax discourse_semantics analyse
    d_initial_anchor_s C0
        (d_because_mod I_DU I_DU I_DU d_forexample
           (d_anchor_s C2 I_DU) ) : T;
```

We use the following piece of code and commands in order to encode the derivation tree, and interpret the yield, syntactic, and semantic interpretations of *Fred is grumpy, because, he, for example, failed his exam.*

```
d_yield d_syntax discourse_semantics analyse
    d_initial_anchor_s C0
        (d_because_initial_medial  I_DU I_DU I_DU d_forexample_medial
           (d_anchor_s C2 I_DU)):T;
```

D.3 ACG Signatures and Lexicons: Syntax and Unlabeled Semantics

```
signature DSTAG =
   DU,DUa,T:type;
   NP,N,Adj,Det,S:type;
   Na, Na_d,VPa,Adja,Sa, DCa, Dcav:type;

   DC = DUa-> DUa-> DUa-> DU-> DUa : type;
   DC_preposed = DUa-> DUa-> DUa-> DUa-> DU-> DUa : type;
   DC_mod = DUa-> DUa-> DUa-> DCa -> DU-> DUa : type;
   DC_modv = DUa-> DUa-> DUa-> Dcav ->DU-> DUa : type;


   In:Na;
   Ivp:VPa;
   Iadj:Adja;
   Is : Sa;

   d_forexample:DCa;
   d_forexample_medial:Dcav;

   d_fred,d_this,d_he,d_her, d_Eiffel, d_Louvre, d_Paris : NP;
   d_is: Sa -> VPa -> VPa -> NP -> Adj -> S;
   d_was: Sa -> VPa -> VPa -> NP -> Adj -> S;
   d_was_in: Sa -> VPa -> VPa -> NP -> NP -> S;
   d_grumpy,d_empty: Adja -> Adj;
   d_away: Adja -> NP -> Adj;


   d_the,d_his,d_a,d_plur : Na_d;
   d_keys,d_nightmare,d_exam,
   d_supermarket,d_fridge,d_movies,d_wife,d_week : Na_d -> Na-> NP;

   d_shows: Sa -> VPa -> VPa -> NP -> S -> S;

   d_lost,d_had,d_failed,d_loves: Sa -> VPa -> VPa -> NP -> NP -> S;
   d_went_to, d_visited : Sa -> VPa -> VPa -> NP -> NP -> S;


   d_didnt:VPa -> VPa;
   d_sleep: Sa -> VPa -> VPa -> NP -> S;
   d_how_much:Sa;

   d_when:DC_preposed;
   d_because:DC;
   d_moreover,d_punct_moreover: DC ;
   d_then_s,d_punct_then_s:DC;
   d_then_v:DC;
   d_punct_next_s:DC;

   d_discourse_empty:DC;
   d_discourse_empty_comment:DC;

   d_initial_anchor_s:S -> DUa -> T;
   d_anchor_s:S -> DUa -> DU;
```

```
   I_DU : DUa;


   d_because_mod : DC_mod;
   d_moreover_modv: DC_modv;
   d_because_modv: DC_modv;

   d_because_initial_medial: DC_modv;



   C0=d_is Is Ivp Ivp d_fred (d_grumpy Iadj) : S;
   C1=d_lost Is Ivp Ivp d_he ( d_keys d_his In) :S;
   C2=d_failed Is Ivp Ivp d_he (d_exam  d_his In) :S;
   C3=d_sleep Is Ivp (d_didnt Ivp) d_he :S;
   C4= d_had Is Ivp Ivp d_he ( d_nightmare d_plur In):S;

   C5= d_went_to Is Ivp Ivp d_fred (d_supermarket d_the In) :S;
   C6= d_was Is Ivp Ivp (d_fridge d_the In) (d_empty Iadj):S;
   C7= d_went_to Is Ivp Ivp d_fred (d_movies d_the In) :S;
   C8= d_is Is Ivp Ivp (d_wife d_his  In) (d_away Iadj (d_week d_the In)):S;
   C9= d_shows Is Ivp Ivp d_this (d_loves d_how_much Ivp Ivp d_he d_her) :S;

   C11=d_was_in Is Ivp Ivp d_fred d_Paris :S;
   C12=d_went_to Is Ivp Ivp d_he d_Eiffel :S;
   C13=d_visited Is Ivp Ivp d_fred d_Louvre :S;

   C20=d_loves Is Ivp Ivp d_fred d_Paris :S;
   C21=d_loves Is Ivp Ivp d_fred d_Eiffel :S;
   C22=d_loves Is Ivp Ivp d_fred d_Louvre :S;


end

signature logic =
   e,t,l:type;

   ttt=(t => t) => t:type;

   qnp=(e=>t)=>t:type;
   infix & : t => t => t;
   prefix - : t => t;

   (* Implications*)
   infix > : t => t => t;

   (* Quantifiers *)
   binder All : (e=>t) => t;
   binder Ex : (e=>t) => t;
   binder ExUni : (e=>t) => t;


   fred,this, eiffel, louvre, paris :e;
   bad_mood:e=>t;
   away:e=> e => t;
   empty:e=>t;
   key,nightmare,license,supermarket,fridge,movies,week:e=>t;
   wife:e=>e=>t;

   lose,have,miss,love,go_to,fail,be_in:e=>e=>t;

   show:e=>t=>t;

   PAST:t=>t;
    PLUR: (e =>t) => qnp;

   sleep:e=>t;
   badly,a_lot:t=>t;

   Explication : t=> t=> t ;
   Continuation : t=> t=> t ;
   Narration : t=> t=> t ;
   Commentary : t => t => t;
   Circumstance: t => t => t;

   Exemplification : t => (t => t)=>t;


   missing_arg : (t=>t=>t) => t=>t;

   transitive_verb =
Lambda v. Lambda S A1 A2 s o mod.S(s(Lambda x.o(Lambda y.A2 (mod (A1(v x y)))))):
   (e=>e=>t) => (t=>t) => (t=>t) => (t=>t) => qnp => qnp => (t=>t) => t;

   intransitive_verb =
Lambda v. Lambda S A1 A2 s mod.S(s(Lambda x.A2 (mod (A1 (v x))))):
   (e=>t) => (t=>t) => (t=>t) => (t=>t) => qnp => (t=>t) => t;

   noun = Lambda n. Lambda d a. d (a (Lambda x. n x)):
(e=>t) => ((e => t) => (e => t) => t) => ((e => t) => (e => t)) => (e => t) => t;
```

```
   phi'' = Lambda R X Y P.X(Lambda x.Y(Lambda y.(P x) & (R x y))):
(t => t=> t)=> ((t => t)=> t)=> ((t => t)=> t)=> (t=> t)=> t;

   my_phi' = Lambda R. Lambda X Y P.X(Lambda x.Y(Lambda y. P (R x y))):
(t => t=> t)=> ((t => t)=> t)=> ((t => t)=> t)=> (t=> t)=> t;

   B = Lambda R.
Lambda d4 d3 d2.
        Lambda d_subst.
Lambda d_foot.
   d4 (
(phi'' R)
   (d3 d_foot)
(d2 d_subst)
   ) :
(t => t => t) =>
   (ttt => ttt) =>
(ttt => ttt) =>
   (ttt => ttt) =>
ttt =>
   (ttt => ttt);

   A' = Lambda R.
Lambda d4 d3 d2.
        Lambda d_subst.
Lambda d_foot.
   d4 (
(my_phi' R)
   (d3 d_foot)
(d2 d_subst)
   ) :
(t => t => t)  =>
   (ttt => ttt) =>
(ttt => ttt) =>
   (ttt => ttt) =>
ttt =>
   (ttt => ttt);
S' = Lambda R.
Lambda  d4 d3 d2 mod.
        Lambda d_subst.
Lambda d_foot.
   d4 (
(my_phi' (mod R))
   (d3 d_foot)
(d2 d_subst)
   ) :
(t => t => t)  =>
   (ttt => ttt) =>
(ttt => ttt) =>
   (ttt => ttt) => ((t => t => t)=>(t => t => t)) =>
ttt =>
   (ttt => ttt);


 S''=  Lambda R.
   Lambda  d4 d3 d2 mod.
         Lambda d_subst.
      Lambda d_foot.
         d4 (
         (phi'' (mod R))
            (d3 d_foot)
            (d2 d_subst)
               ) :
               (t => t => t)  =>
                  (ttt => ttt) =>
                  (ttt => ttt) =>
                     (ttt => ttt) => ((t => t => t)=>(t => t => t)) =>
                     ttt =>
                        (ttt => ttt);

   A_preposed = Lambda R.
   Lambda d5 d4 d3 d2.
         Lambda d_subst.
      Lambda d_foot.
         d4 (
         (my_phi' R)
            (d3 (d5 d_foot))
            (d2 d_subst)
               ) :
               (t => t => t)  =>
                (ttt => ttt) =>
                  (ttt => ttt) =>
                  (ttt => ttt) =>
                     (ttt => ttt) =>
                     ttt =>
                        (ttt => ttt);

   cont = Lambda t.Lambda P.P t:t => (t=>t) => t;
```

```
   end

nl_lexicon discourse_semantics(DSTAG):logic =
   NP := qnp;
   N := e=>t;
   Adj := e=>t;
   S := (t=>t) => t;
   Det := (e=>t) => qnp;

   Na_d := (e => t) => (e => t) => t;

   Na := (e=>t) => (e=>t);
   VPa := t => t;
   Adja:= (e=>t) => (e=>t);
   Sa:= t => t;
   DCa, Dcav := (t => t => t)=>(t => t => t);

   d_plur:= Lambda P Q.(PLUR P) Q;

   Ivp,Iadj,Is,In:=Lambda x.x;

   (* To be changed for pronouns *)
   d_fred,d_he := Lambda P.P fred;
   d_Eiffel := Lambda P.P eiffel;
   d_Louvre := Lambda P.P louvre;
   d_Paris := Lambda P.P paris;

   d_her := Lambda P.ExUni x. (wife x fred) & (P x);
   d_this:= Lambda P.P  this;

   d_is,d_was := Lambda S A1 A2 s adj mod. (S(s (Lambda x.A2 (mod (A1 (adj x))))));
   d_grumpy := Lambda m.m bad_mood;
   d_away := Lambda m P z .P(Lambda x. m (Lambda y. away y x) z)  ;
   d_empty := Lambda m.m empty;

   d_the,d_his := Lambda P Q.ExUni x.(P x) & (Q x);
   d_a := Lambda P Q.Ex x.(P x) & (Q x);

   d_keys:= noun key;
   d_nightmare := noun nightmare;
   d_exam  := noun license;
   d_supermarket := noun supermarket;
   d_fridge := noun fridge;
   d_movies := noun movies ;
   d_wife := noun (Lambda x. wife x fred);
   d_week := noun week;

   d_shows := Lambda S A1 A2 s c mod. mod (S ( s ( Lambda x . A2 (A1 (show x  (c (Lambda y.y) ) ))));

   d_lost := transitive_verb lose;
   d_had := transitive_verb have;
   d_failed := transitive_verb fail;
   d_loves := transitive_verb love;
   d_went_to, d_visited := transitive_verb go_to;
   d_was_in := transitive_verb be_in;


   d_sleep := intransitive_verb sleep;

   d_didnt := Lambda m P.m (- P);
   d_how_much := Lambda s.a_lot s;

   d_forexample_medial, d_forexample := lambda R p q. Exemplification q (lambda r. R p r);



   DU := ttt;
   DUa := ttt => ttt;
   T := t;
   I_DU := Lambda x.x;

   d_when := A_preposed Circumstance;
   d_because := A' Explication;
   d_moreover, d_punct_moreover := A' Continuation;
   d_then_s, d_then_v, d_punct_then_s, d_then_v  := B Narration;
   d_punct_next_s:= A' Narration;

   d_discourse_empty_comment := A' Commentary;
   d_discourse_empty := B Explication;

   d_initial_anchor_s := Lambda s mod. mod (Lambda Q.Q (s (Lambda x.x))) (Lambda x.x);

   d_anchor_s := Lambda s mod.Lambda P. mod (Lambda Q.Q (s (Lambda x.x))) P;

   d_because_mod, d_because_initial_medial, d_because_modv := S' Explication;
   d_moreover_modv:= S'' Continuation;

end
```

```
signature TAG =
   NP,N,Adj,S,Det:type;
   Na,Na_d,VPa,Adja,Sa:type;
   In:Na;
   Ivp:VPa;
   Iadj:Adja;
   Is : Sa;

   c_fred,c_this,c_he,c_her : NP;
   c_Paris  : NP;
   c_Eiffel : NP;
   c_Louvre : NP;
   c_is,c_was: Sa -> VPa -> NP -> Adj -> S;
   c_grumpy,c_empty: Adja -> Adj;
   c_away: Adja -> NP -> Adj;


   c_the,c_his,c_a,c_plur :Na_d;
   c_keys,c_nightmare,c_exam,
   c_supermarket,c_fridge,c_movies,c_wife,c_week : Na_d -> Na-> NP;

   c_lost,c_had,c_failed,c_loves: Sa -> VPa -> NP -> NP -> S;
   c_went_to, c_visited : Sa -> VPa -> NP -> NP -> S;
   c_was_in : Sa -> VPa -> NP -> NP -> S;


   c_shows: Sa -> VPa -> NP -> S -> S;

   c_didnt:VPa -> VPa;
   c_sleep: Sa -> VPa -> NP -> S;
   c_how_much:Sa -> Sa;

   c_because:Sa;
   c_when:Sa;
   c_moreover_comma: Sa ;
   c_moreover_v:VPa -> VPa;
   c_because_v:VPa -> VPa;

   c_then_comma_s:Sa;
   c_then_v:VPa -> VPa;
   c_next_comma_s:Sa;

   c_forexample:Sa->Sa;
   c_forexample_medial:(VPa->VPa)->(VPa->VPa);

   transitive_verb=
   lambda v.
      lambda sa va1 va2 subj obj.
      lambda dc mod.
         v (sa dc) (va2 (mod (va1 Ivp))) subj obj:
         (Sa -> VPa -> NP -> NP -> S) ->
            (Sa -> Sa) -> (VPa -> VPa) -> (VPa -> VPa) -> NP -> NP ->
            Sa -> (VPa -> VPa) -> S;

   intransitive_verb=
   lambda v.
      lambda sa va1 va2 subj.
      lambda dc mod.
         v (sa dc) (va2 (mod (va1 Ivp))) subj:
         (Sa -> VPa -> NP -> S) ->
            (Sa -> Sa) -> (VPa -> VPa) -> (VPa -> VPa) -> NP ->
            Sa -> (VPa -> VPa) -> S;

   aux = lambda aux.
   lambda va v .aux (va v) :
      (VPa -> VPa) ->
      (VPa -> VPa) -> VPa -> VPa;

   v_adv = lambda adv.
   lambda va v. adv (va v) :
      (VPa -> VPa) ->
      (VPa -> VPa) -> VPa -> VPa;

   pont = lambda v.
   lambda sa va1 va2 subj scomp.lambda dc mod.v (sa dc) (va2 (mod (va1 Ivp))) subj (scomp Is (lambda x.x)):
   (Sa -> VPa -> NP -> S -> S) ->
   (Sa -> Sa) -> (VPa -> VPa) -> (VPa -> VPa) -> NP -> (Sa -> (VPa -> VPa) -> S) -> Sa -> (VPa -> VPa) -> S;


   I_va = lambda x.x : VPa -> VPa;
   I_va_va = lambda x.x : (VPa -> VPa) -> VPa -> VPa;

   concat_wo_punct,concat_w_punct : Sa -> Sa -> Sa -> S -> Sa;
   concat_wo_punct_preposed : Sa -> Sa -> Sa -> Sa -> S -> Sa;

   c_discourse_wo_punct_s = lambda dm_s dm_v. lambda dua1 dua2 dua3 s.
   concat_wo_punct dua1 dua2 dua3 (s dm_s dm_v):
   Sa -> (VPa -> VPa) -> Sa -> Sa -> Sa -> (Sa -> (VPa -> VPa) -> S) -> Sa;
   c_discourse_w_punct_s = lambda dm_s dm_v. lambda dua1 dua2 dua3 s.
   concat_w_punct dua1 dua2 dua3 (s dm_s dm_v):
   Sa -> (VPa -> VPa) -> Sa -> Sa -> Sa -> (Sa -> (VPa -> VPa) ->S) -> Sa;
```

```
   c_discourse_preposed = lambda dm_s dm_v. lambda dua5 dua1 dua2 dua3 s.
   concat_wo_punct_preposed dua5 dua1 dua2 dua3 (s dm_s dm_v):
      Sa -> (VPa -> VPa) ->  Sa -> Sa -> Sa -> Sa -> (Sa -> (VPa -> VPa) ->S) -> Sa;


   clause_modification: S -> Sa -> S;



end

signature trees =
   tree:type;
   N1,NP1,V1,Adv1,VP1,Adj1,S1,Det1,Prep1: tree -> tree;
   N2,NP2,PP2,VP2,Adj2,S2:tree -> tree -> tree;
   S3: tree -> tree -> tree -> tree;
   S4: tree -> tree -> tree -> tree -> tree;
   S5: tree -> tree -> tree -> tree -> tree -> tree;

   Id=lambda x.x:tree -> tree;

   dot,comma:tree;

   Fred,is,was,didnt,grumpy,lost,the,keys,failed,exam,
   sleep,well,had,nightmare,s,went,to, in, supermarket,fridge,empty,movies,
   his,wife,away,a,week,this,shows,he,her,loves, visited, forexample,
   moreover,moreover_comma,then,then_comma,because,how_much, when, next_comma, Paris, Louvre, Eiffel, tower:tree;

   epsilon:tree;


   transitive_verb=
   lambda v. lambda S A s o.S(S2 s (A (VP2 (V1 v) o))):
   tree ->
   (tree -> tree) ->
   (tree -> tree) ->
   tree ->
   tree ->
   tree ;


   intransitive_verb=
   lambda v. lambda S A s.S(S2 s (A (VP1 (V1 v)))):
   tree ->
   (tree -> tree) ->
   (tree -> tree) ->
   tree ->
   tree ;

   aux =
   lambda a. lambda A x. VP2 (A (VP1 a)) x :
   tree ->
   (tree -> tree) ->
   tree ->
   tree ;

   v_adv = lambda a. lambda A x. A(VP2 (Adv1 a) x) :
   tree ->
   (tree -> tree) ->
   tree ->
   tree ;

   s_adv = lambda a. lambda S x. S(S2 (Adv1 a) x) :
   tree ->
   (tree -> tree) ->
   tree ->
   tree ;


   pont=
   lambda v. lambda S A s comp.S(S2 s (A (VP2 (V1 v) (S1 comp)))):
   tree ->
   (tree -> tree) ->
   (tree -> tree) ->
   tree ->
   tree ->
   tree ;

   close = lambda f.f epsilon:(tree -> tree) -> tree;

   c_discourse_wo_punct = lambda s1 s2 s3 s x.
       s1 (s2 (S2 x (s3 s))) :
      (tree -> tree) ->
         (tree -> tree) ->
         (tree -> tree) ->
            tree ->
            tree -> tree;

   c_discourse_w_punct = lambda p. lambda s1 s2 s3 s x.
       s1 (s2 (S3 x p (s3 s))) :
      tree ->
```

```
      (tree -> tree) ->
         (tree -> tree) ->
         (tree -> tree) ->
            tree ->
            tree -> tree;

   concat_discourse_punct_preposed = lambda s5 s4 s3 s2 s x.
         s4 (s3 (S3 (s2 s) comma (s5 x))) :
      (tree -> tree) ->
      (tree -> tree) ->
         (tree -> tree) ->
         (tree -> tree) ->
            tree ->
            tree -> tree;



   noun = lambda n. lambda d a. d (a (N1 n)): tree -> (tree -> tree) -> (tree -> tree) -> tree;
   det =  lambda d. lambda n. N2 d n : tree -> (tree -> tree);
   plur = lambda n. NP2 n s : tree -> tree ;
   l_adj = lambda adj. lambda a n. a (N2 adj n): tree -> (tree -> tree) -> (tree -> tree);



end

lexicon derived_trees(TAG):trees =
   NP,N,Adj,S,Det:=tree;

   Na_d,Na,VPa,Adja,Sa:=tree -> tree;

   Iadj,Ivp,In,Is := lambda x.x;

   c_fred:= NP1 Fred;
   c_this := NP1 this;
   c_he := NP1 he;
   c_her := NP1 her;
   c_is:= lambda S A s adj. S(S2 s (A (VP2 (V1 is) adj)));
   c_was:= lambda S A s adj. S(S2 s (A (VP2 (V1 was) adj)));
   c_grumpy:= lambda m. m(Adj1 grumpy);
   c_empty := lambda m.m(Adj1 empty);

   c_Eiffel:= NP2 (Det1 the) (N2 Eiffel tower);
   c_Louvre :=  NP2 (Det1 the) (N1 Louvre);
   c_Paris :=  NP1 Paris;


   c_away := lambda m n .m (Adj2 (Adj1 away) n) ;

   c_the := det the;
   c_his := det his;
   c_a := det a;
   c_plur := plur;
   c_keys:= noun keys;
   c_fridge:= noun fridge;
   c_movies:= noun movies;
   c_exam:= noun exam;
   c_supermarket:= noun supermarket;
   c_nightmare:= noun nightmare;
   c_wife:= noun wife;
   c_week:= noun week;

   c_lost:=transitive_verb lost;
   c_had:=transitive_verb had;
   c_failed:=transitive_verb failed;
   c_loves:=transitive_verb loves;
   c_visited:=transitive_verb visited;


   c_went_to:=  lambda S A s c.S(S2 s (A (VP2 (V1 went) (PP2 (Prep1 to) c))));
   c_was_in:=  lambda S A s c.S(S2 s (A (VP2 (V1 was) (PP2 (Prep1 in) c))));

   c_forexample := lambda S x. S(S2 (Adv1 forexample) x);

   c_forexample_medial:= lambda B. lambda Z. lambda u. Z (B (lambda y. VP2 y u) (Adv1 forexample));

   c_didnt:= aux didnt;
   c_sleep:= intransitive_verb (VP2 sleep well);

   c_shows:= pont shows;
   c_how_much := s_adv how_much;

   c_then_comma_s := s_adv then_comma (lambda x.x);
   c_then_v := v_adv then;
   c_moreover_v :=  v_adv moreover;
   c_because_v :=  v_adv because;

   c_moreover_comma := s_adv moreover_comma (lambda x.x);
   c_because := s_adv because (lambda x.x);
   c_when := s_adv when (lambda x.x);
   c_next_comma_s := s_adv next_comma (lambda x.x);
```

```
   concat_wo_punct_preposed:=concat_discourse_punct_preposed;
   concat_w_punct:= c_discourse_w_punct dot;
   concat_wo_punct:= c_discourse_wo_punct;

   clause_modification := lambda s m.m s;

end


signature strings =
   o:type;
   string = o->o:type;
   infix + = lambda a b z.a (b z):string -> string -> string;
   binary = lambda x y. x + y: string -> string -> string ;
   ternary = lambda x y z. x + y + z:  string -> string -> string -> string;
   E=lambda x.x:string;

   Fred,is,was,did,not,grumpy,lost,the,keys,failed,exam,
   well,sleep,had,nightmare,s,went,to, in, supermarket,fridge,empty,movies,
   his,wife,away,a,week,this,shows,he,her,loves, visited, for, example,
   moreover,then,because,how,much, when, next, Paris, Louvre, Eiffel, Tower:string;

   dot,comma,epsilon:string;
end


lexicon disc_clause_interface(DSTAG):TAG =


   T := S;

   NP:=NP;
   N:=N;
   Adj:=Adj;
   Det:=Det;
   Na:=Na;
   Adja:=Adja;
   Na_d:=Na_d;


   VPa:=VPa -> VPa;
   Sa:=Sa->Sa;
   S:= Sa -> (VPa -> VPa) -> S;
   DU:= Sa -> (VPa -> VPa)-> S;
   DUa:= Sa;

   Dcav:=(VPa -> VPa)-> VPa -> VPa;
   DCa := Sa->Sa;


   In:=In;
   Ivp:=lambda P.P;
   Iadj:=Iadj;
   Is := lambda P.P;

   d_fred := c_fred;
   d_this := c_this;
   d_Louvre := c_Louvre;
   d_Paris := c_Paris;
   d_Eiffel := c_Eiffel;
   d_he :=c_he;
   d_her  := c_her ;
   d_is := lambda sa va1 va2 np adj.lambda dc mod. c_is (sa dc) (va2 (mod (va1 Ivp))) np adj ;
   d_was := lambda sa va1 va2 np adj.lambda dc mod. c_was (sa dc) (va2 (mod (va1 Ivp))) np adj ;
   d_grumpy:=c_grumpy;
   d_empty:=c_empty;
   d_away := c_away;


   d_the := c_the ;
   d_his := c_his;
   d_a:=c_a;
   d_plur := c_plur;
   d_keys:=c_keys;
   d_nightmare := c_nightmare;
   d_exam := c_exam;
   d_supermarket := c_supermarket;
   d_fridge := c_fridge;
   d_movies := c_movies;
   d_wife := c_wife;
   d_week:=c_week;

   d_lost:=transitive_verb c_lost;
   d_had := transitive_verb c_had;
   d_failed := transitive_verb c_failed;
   d_loves:=transitive_verb c_loves;
   d_visited:=transitive_verb c_visited;

   d_went_to  := transitive_verb c_went_to ;
   d_was_in  := transitive_verb c_was_in ;
```

```
   d_shows := pont c_shows;

   d_didnt:=aux c_didnt;
   d_sleep := intransitive_verb c_sleep;
   d_how_much := c_how_much;

   I_DU := Is;
   d_then_v :=  c_discourse_w_punct_s Is c_then_v;
   d_then_s := c_discourse_wo_punct_s c_then_comma_s (lambda x.x);
   d_moreover :=  c_discourse_wo_punct_s c_moreover_comma (lambda x.x);
   d_punct_then_s := c_discourse_w_punct_s c_then_comma_s (lambda x.x);
   d_punct_moreover :=  c_discourse_w_punct_s c_moreover_comma (lambda x.x);
   d_because := c_discourse_wo_punct_s c_because (lambda x.x);
   d_when := c_discourse_preposed  c_when (lambda x.x);
   d_punct_next_s := c_discourse_wo_punct_s c_next_comma_s (lambda x.x);


   d_because_mod := lambda dua1 dua2 dua3 mod s.  concat_wo_punct dua1 dua2 dua3 (s (mod c_because) (lambda x.x) );
   d_moreover_modv  :=  lambda dua1 dua2 dua3 mod s. concat_wo_punct dua1 dua2 dua3 (s  Is (mod c_moreover_v));
   d_because_modv := lambda dua1 dua2 dua3 mod s. concat_wo_punct dua1 dua2 dua3 (s Is (mod c_because_v));

   d_because_initial_medial :=  lambda dua1 dua2 dua3 mod s. concat_wo_punct dua1 dua2 dua3 (s  c_because (mod  (lambda x.x)));


   d_forexample:=c_forexample;
   d_forexample_medial:=c_forexample_medial;

   d_discourse_empty := c_discourse_w_punct_s Is (lambda x.x);
   d_discourse_empty_comment := c_discourse_w_punct_s Is (lambda x.x);

   d_initial_anchor_s := lambda s mod.clause_modification (s Is (lambda x.x)) mod;
   d_anchor_s := lambda s mod dc_s dc_v. clause_modification (s dc_s dc_v) mod;
end

lexicon yields(trees):strings =
   tree:=string;


   N1,NP1,V1,Adv1,VP1,Adj1,S1,Det1,Prep1 := lambda x.x;
   N2,NP2,PP2,VP2,Adj2,S2 := lambda x y.x+y;
   S3  := lambda x y z. x+y+z;
   S4 := lambda x y z u. x+y+z+u;
   S5 := lambda x y z u t. x+y+z+u+t;

   dot :=  dot;
   comma := comma;
   epsilon:=epsilon;

   Fred:=Fred;
   is:=is;
   was:=was;
   didnt:=did+not;
   grumpy:=grumpy;
   lost:=lost;
   the:=the;
   keys:=keys;
   failed:=failed;
   exam:=exam;
   sleep:=sleep;
   well:=well;
   had:=had;
   nightmare:=nightmare;
   s:=s;
   went:=went;
   to:=to;
   in:=in;
   supermarket:=supermarket;
   fridge:=fridge;
   empty:=empty;
   movies:=movies;
   his:=his;
   wife:=wife;
   away:=away;
   a:=a;
   week:=week;
   this:=this;
   shows:=shows;
   he:=he;
   her:=her;
   loves:=loves;
   visited:=visited;
   forexample:=for+example;
   moreover:=moreover;
   moreover_comma:=moreover+comma;
   then:=then;
   then_comma:=then+comma;
   because:=because;
   how_much:= how+much;
   when:=when;
   next_comma:=next+comma;
```

```
   Paris:=Paris;
   Louvre:=Louvre;
   Eiffel:=Eiffel;
   tower:=Tower;

end

lexicon d_syntax = derived_trees << disc_clause_interface
lexicon d_yield = yields << d_syntax
```

D.4 The ACG Signatures and Lexicons for D-STAG as ACG - Labeled Semantics

D.4.1 Examples

We use the following code in order to obtain the derivation tree and the labeled semantic interpretation of *Fred is grumpy because he lost his keys.*

```
labeled_semantics analyse
   d_initial_anchor_s C0
      (d_because I_DU I_DU I_DU (d_anchor_s C1 I_DU)): T;
```

We use the following code in order to obtain the derivation tree and the labeled semantic interpretation of *Fred is grumpy because he lost his keys. Moreover, he failed his exam.*

```
labeled_semantics analyse
   d_initial_anchor_s C0
      (d_because I_DU I_DU I_DU (d_anchor_s C1
         (d_punct_moreover I_DU I_DU I_DU (d_anchor_s C2 I_DU)))): T;
```

We use the following command in order to obtain the derivation tree and the labeled semantic interpretation of *Fred is grumpy because he did not sleep well. He had nightmares.*

```
labeled_semantics analyse
   d_initial_anchor_s C0
      (d_because I_DU I_DU
         (d_discourse_empty I_DU I_DU I_DU (d_anchor_s C4 I_DU))
            (d_anchor_s C3 I_DU)): T;
```

We use the following code in order to obtain the derivation tree and the labeled semantic interpretation of *Fred went to the supermarket because the fridge was empty. Then, Fred went to the movies.*

```
labeled_semantics analyse
   d_initial_anchor_s C5
      (d_because I_DU
         (d_punct_then_s I_DU I_DU I_DU
            (d_anchor_s C7 I_DU)) I_DU
               (d_anchor_s C6 I_DU)) : T;
```

We use the following code in order to obtain the derivation tree and the labeled semantic interpretation of *Fred went to the supermarket because the fridge was empty. Fred* then *went to the movies.*

```
labeled_semantics analyse
   d_initial_anchor_s C5
      (d_because I_DU
         (d_then_v I_DU I_DU I_DU
            (d_anchor_s C7 I_DU)) I_DU
               (d_anchor_s C6 I_DU)) : T;
```

We use the following code in order to obtain the derivation tree and the labeled semantic interpretation of *Fred is grumpy because his wife is away this week. This shows how much he loves her.*

```
labeled_semantics analyse
    d_initial_anchor_s C0
        (d_because
           (d_discourse_empty_comment I_DU I_DU I_DU
              (d_anchor_s C9 I_DU)) I_DU  I_DU
                 (d_anchor_s C8 I_DU)) : T;
```

D.4.2 ACG Signatures and Lexicons for Interpreting D-STAG derivation trees into Labeled Semantics

```
signature DSTAG =
   DU,DUa,T:type;
   NP,N,Adj,Det,S:type;
   NPa,Na,VPa,Adja,Sa, Na_d:type;

   DC = DUa-> DUa-> DUa-> DU-> DUa : type;

   In:Na;
   Ivp:VPa;
   Iadj:Adja;
   Is : Sa;

   d_fred,d_this,d_he,d_her : NP;
   d_is: Sa -> VPa -> VPa -> NP -> Adj -> S;
   d_was: Sa -> VPa -> VPa -> NP -> Adj -> S;
   d_grumpy,  d_empty: Adja -> Adj;
   d_big, d_important : Adja -> Adja;
   d_away: Adja -> NP -> Adj;



   d_the, d_his,d_a, d_plur : Na_d;
   d_keys,d_nightmare,d_exam,
   d_supermarket,d_fridge,d_movies,d_wife,d_week : Na_d -> Na-> NP;



   d_lost,d_had,d_failed,d_loves: Sa -> VPa -> VPa -> NP -> NP -> S;
   d_went_to : Sa -> VPa -> VPa -> NP -> NP -> S;

   d_shows: Sa -> VPa -> VPa -> NP -> S -> S;

   d_didnt:VPa -> VPa;
   d_sleep: Sa -> VPa -> VPa -> NP -> S;
   d_how_much:Sa;

   d_because:DC;
   d_moreover,d_punct_moreover: DC ;
   d_then_s,d_punct_then_s:DC;
   d_then_v:DC;

   d_discourse_empty:DC;
   d_discourse_empty_comment:DC;

   d_initial_anchor_s:S -> DUa -> T;
```

```
   d_anchor_s:S -> DUa -> DU;

   I_DU : DUa;


   C0=d_is Is Ivp Ivp d_fred (d_grumpy Iadj) : S;
   C1=d_lost Is Ivp Ivp d_he ( d_keys d_his In) :S;
   C2=d_failed Is Ivp Ivp d_he (d_exam  d_his In) :S;
   C3=d_sleep Is Ivp (d_didnt Ivp) d_he :S;
   C4= d_had Is Ivp Ivp d_he ( d_nightmare d_plur In):S;

   C5= d_went_to Is Ivp Ivp d_fred (d_supermarket d_the In) :S;
   C6= d_was Is Ivp Ivp (d_fridge d_the In) (d_empty Iadj):S;
   C7= d_went_to Is Ivp Ivp d_fred (d_movies d_the In) :S;
   C8= d_is Is Ivp Ivp (d_wife d_his  In) (d_away Iadj (d_week d_the In)):S;
   C9= d_shows Is Ivp Ivp d_this (d_loves d_how_much Ivp Ivp d_he d_her) :S;

end




signature logic =
   e,t,l:type;

   ttt=(l => t) => t:type;

   qnp=(e=>l=>t)=>l=>t:type;
   infix & : t => t => t;

   (* Implications*)
   infix > : t => t => t;

   prefix - : t => t;
   (* Quantifiers *)
   binder All : (e=>t) => t;
   binder Ex : (e=>t) => t;
   binder ExUni : (e=>t) => t;
   binder Ex_l : (l=>t) => t;

   TOP:t;
   PLUR: (e =>l =>t) => qnp;

   fred,this,he,elle:e;
   grumpy:e=>l=>t;
   away:e=> e => l => t;
   empty, big, important:e=>l=>t;
   keys,nightmare,exam,supermarket,fridge,movies,week:e=>l=>t;
   wife:e=>e=>l=>t;

   lose,have,fail,love,go_to:e=>e=>l=>t;

   show:e=>t=>l=>t;

   PAST:t=>t;

   sleep:e=>l=>t;
   badly,a_lot:t=>t;

   Explanation : l=> l=> l => t ;
   Continuation : l=> l=> l =>t ;
   Narration : l=> l=> l=>t ;
   Comment : l => l => l=>t;


   missing_arg : (t=>t=>t) => t=>t;

   transitive_verb =
Lambda v. Lambda S A1 A2 s o mod.Lambda l.(S(s(Lambda x l'.o(Lambda y l''.(A2 (mod (A1(v x y l''))))) l' ) l )):
   (e=>e=>l=>t) => (t=>t) => (t=>t) => (t=>t) => qnp => qnp => (t=>t) => l => t;

   intransitive_verb =
Lambda v. Lambda S A1 A2 s mod.Lambda l.(S(s(Lambda x l'.(A2 (mod (A1 (v x l'))))) ) l)):
   (e=>l=>t) => (t=>t) => (t=>t) => (t=>t) => qnp => (t=>t) => l => t;

   noun = Lambda n. Lambda d a. d (a (Lambda x l. n x l)):
  (e => (l => t)) => ((e => (l => t)) => ((e => (l => t)) => (l => t))) => (((e => (l => t)) => (e => (l => t))) => ((e => (l => t)) => (l => t)));


   phi'' = Lambda R X Y P.Ex_l l.X(Lambda x.Y(Lambda y.(P x) & (R x y l))):
(l => l=> l =>t)=> ((l => t)=> t)=> ((l => t)=> t)=> (l=> t)=> t;

  my_phi' = Lambda R  X Y P.Ex_l l.X(Lambda x.Y(Lambda y. (R x y l) & (P l))): (l => l=> l=> t)=> ((l => t)=> t)=> ((l => t)=> t)=> (l=> t)=> t;
   B = Lambda R.

   Lambda d4 d3 d2.
        Lambda d_subst.
Lambda d_foot.
   d4 (
(phi'' R)
```

```
   (d3 d_foot)
(d2 d_subst)
   ) :
(l => l => l => t) =>
   (ttt => ttt) =>
(ttt => ttt) =>
   (ttt => ttt) =>
ttt =>
   (ttt => ttt);

   A' = Lambda R.
Lambda d4 d3 d2.
        Lambda d_subst.
Lambda d_foot.
   d4 (
(my_phi' R)
   (d3 d_foot)
(d2 d_subst)
   ) :
(l => l => l=>t)  =>
   (ttt => ttt) =>
(ttt => ttt) =>
   (ttt => ttt) =>
ttt =>
   (ttt => ttt);


   cont = Lambda t.Lambda P.P t:t => (t=>t) => t;


end

nl_lexicon labeled_semantics(DSTAG):logic =
   NP := qnp;
   N := e=> l => t;
   Adj := e=>l => t;
   S := (t=>t) => l => t;
   Det := (e=>l=>t) => qnp;
   Na_d := (e=>l=>t) => (e=>l=>t)=> l => t;


   d_plur:= Lambda P Q.(PLUR P) Q;

   NPa:= qnp => qnp;
   VPa := t => t;
   Na, Adja:= (e=>l=>t) => (e=>l=>t);
   Sa:= t => t;



   Ivp,Iadj,Is,In:=Lambda x.x;

   d_fred,d_he := Lambda P l .P fred l;
   d_her := Lambda P l .ExUni x. (wife x fred l) & (P x l);
   d_this:= Lambda P l .P  this l;

   d_is,d_was := Lambda S A1 A2 s adj mod.Lambda l. (S(s (Lambda x l.A2 (mod (A1 (adj x l)))) l ));
   d_grumpy := Lambda m.m grumpy;
   d_away := Lambda m P z l .P(Lambda x l'. m (Lambda y l''. away y x l'') z l') l  ;
   d_empty := Lambda m.m empty;

   d_the,d_his := Lambda P Q l.ExUni x.(P x l) & (Q x l);
   d_a := Lambda P Q l.Ex x.(P x l) & (Q x l);

   d_keys:= noun keys;
   d_nightmare := noun nightmare;
   d_exam := noun exam;
   d_supermarket := noun supermarket;
   d_fridge := noun fridge;
   d_movies := noun movies ;
   d_wife := noun (Lambda x l. wife x fred l);
   d_week := noun week;

   d_lost := transitive_verb lose;
   d_had := transitive_verb have;
   d_failed := transitive_verb fail;
   d_loves := transitive_verb love;
   d_went_to := transitive_verb go_to;

   d_shows := Lambda S A1 A2 s c mod l.S(s(Lambda x l'.A2 (mod(A1 (show x (Ex_l l_1.(c (lambda x.x) l_1)) l')))) l );

   d_sleep := intransitive_verb sleep;

   d_didnt := Lambda m P.m (- P);
   d_how_much := Lambda s.a_lot s;

   DU := ttt;
   DUa := ttt => ttt;
   T := t;
   I_DU := Lambda x.x;

   d_because := A' Explanation;
```

```
   d_moreover,d_punct_moreover := A' Continuation;
   d_then_s,d_then_v,d_punct_then_s := B Narration;
   d_then_v := B Narration;
   d_discourse_empty_comment := A' Comment;
   d_discourse_empty := B Explanation;

   d_initial_anchor_s := Lambda s mod.Ex_l l. mod (Lambda Q.(s (Lambda x.x) l) & Q l) (Lambda l.TOP);

   d_anchor_s := Lambda s mod.Lambda P.Ex_l l. mod (Lambda Q.(s (Lambda x.x) l) & (Q l)) P;

   d_important:=Lambda a n. a (Lambda x l. (important x l) & (n x l));
   d_big:= Lambda a n. a (Lambda x l. (big x l) & (n x l));
end
```

Appendix E

Related Work and Conclusive Remarks

Below, we provide the ACG code for modeling a discourse where two connectives appear in a clause, for instance as it is in the following discourse:

(69), repeated

[John ordered three cases of Barolo]$_0$. [But he had to cancel the order]$_1$ [*because then* he discovered he was broke]$_2$.
Interpretation: $(\textsc{Contrast}F_0F_1)\wedge(\textsc{Explanation}F_1F_2)\wedge(\textsc{Narration}F_0F_2)$

By running the following command, one obtains the semantic interpretation of the discourse (69).

```
discourse_semantics analyse d_initial_anchor_s C0
                              (d_but (d_because d_then
                                (d_anchor_s C2 I_DU))
                                   I_DU I_DU I_DU
                                     (d_anchor_s C1 I_DU)):T;
```

The ACG Code for Multiply Connectives in a Clause

```
signature discourse_grammar =
   DU,DUa ,T,  Ra, DUn, S :type;

   d_because : Ra->DU->DUn;
   d_then: Ra;
   d_but : DUn-> DUa-> DUa-> DUa-> DU-> DUa;

   I_DU : DUa;
   I_DUn: DUn;

   d_initial_anchor_s: S -> DUa -> T;
   d_anchor_s:S -> DUa -> DU;

   C0, C1, C2:S;

end
```

```
signature logic =
   e,t,l:type;

   ttt=(t => t) => t:type;

   infix & : t => t => t;


   EXPLANATION, NARRATION, CONTRAST : t => t => t;

   my_phi' = Lambda R. Lambda X Y P.X(Lambda x.Y(Lambda y. P (R x y))):
   (t => t=> t)=> ((t => t)=> t)=> ((t => t)=> t)=> (t=> t)=> t;

   F0, F1, F2: t;
end


nl_lexicon discourse_semantics(discourse_grammar):logic =


   DU,S := ttt;
   DUa := ttt => ttt;
   T := t;

   Ra:= (t=>t=>t)=>t=>t=>t=>t;
   DUn := (ttt=>ttt=>ttt)=>ttt=>ttt=>ttt;


   I_DU := Lambda x.x;
   I_DUn:=Lambda x.x;

   d_then := Lambda R f2 f0 f1. (R f1 f2) & (NARRATION f0 f2);

    d_because:=  Lambda DRa. Lambda F3 Q F2 F1. lambda D.
  D( (Q F2 F1 (lambda x. x)) & (F3 (Lambda f3. (F2 (Lambda f2. (F1 (Lambda f1. DRa EXPLANATION f3 f2 f1)))))) );

   d_but := Lambda P d4 d3 d2.
          Lambda d_subst.
       Lambda d_foot.
          d4(
         P(my_phi' CONTRAST)
            (d3 d_foot)
            (d2 d_subst)
               );

   d_anchor_s := Lambda s mod.Lambda P. mod (Lambda Q.Q (s (Lambda x.x))) P;

  d_initial_anchor_s := Lambda s mod. mod (Lambda Q.Q (s (Lambda x.x))) (Lambda x.x);

  C0:= Lambda P. P F0;
  C1:= Lambda P. P F1;
  C2:= Lambda P. P F2;


end
```

Bibliography

Abeillé, Anne (1988). "Parsing French with Tree Adjoining Grammar: Some Linguistic Accounts". In: *Proc. of the 12th COLING*. Budapest, Hungary, pp. 7–12. URL: http://aclweb.org/anthology/C/C88/C88-1002.pdf (cit. on pp. 203, 234).

Aït-Kaci, Hassan and Roger Nasr (1986). "LOGIN: A logic programming language with built-in inheritance". In: *The Journal of logic programming* 3.3, pp. 185–215. DOI: 10.1016/0743-1066(86)90013-0. URL: http://www.hassan-ait-kaci.net/pdf/login-jlp-86.pdf (cit. on pp. 155, 216).

Ajdukiewicz, Kazimierz (1935). "Die syntaktische Konnexität". In: *Stud. Philos.* 1, pp. 1–27. URL: http://www.ifispan.waw.pl/studialogica/s-p-f/volumina_i-iv/I-03-Ajdukiewicz-small.pdf (cit. on p. 26).

Asher, Nicholas and Alex Lascarides (2003). *Logics of Conversation*. Cambridge University Press (cit. on pp. 16, 17, 20, 107, 108, 112, 119, 120, 123, 124, 125, 174, 251).

Asher, Nicholas and Sylvain Pogodalla (2011). "SDRT and Continuation Semantics". In: *New Frontiers in Artificial Intelligence JSAI-isAI 2010 Workshops, LENLS, JURISIN, AMBN, ISS, Tokyo, Japan, November 18-19, 2010, Revised Selected Papers*. Ed. by Takashi Onada, Daisuke Bekki, and Eric McCready. Vol. 6797. Lecture Notes in Computer Science. Springer, pp. 3–15. DOI: 10.1007/978-3-642-25655-4_2. URL: https://hal.inria.fr/inria-00565744 (cit. on pp. 288, 321, 322).

Asher, Nicholas and Laure Vieu (2005). "Subordinating and Coordinating Discourse Relations". In: *Lingua* 115.4, pp. 591–610. URL: https://www.irit.fr/publis/LILAC/AV-Lingua05.pdf (cit. on p. 123).

Barendregt, H. P. (1992). "Handbook of Logic in Computer Science (Vol. 2)". In: ed. by S. Abramsky, Dov M. Gabbay, and S. E. Maibaum. New York, NY, USA: Oxford University Press, Inc. Chap. Lambda Calculi with Types, pp. 117–309 (cit. on p. 54).

Bar-Hillel, Yehoshua (1953). "A Quasi-Arithmetical Notation for Syntactic Description". In: *Language* 29.1, pp. 47–58. DOI: 10.2307/410452 (cit. on p. 26).

Bernard, Timothée (2015). "Verbes d'attitude propositionnelle et analyse discursive". MA thesis. Université Paris Diderot - Paris 7. URL: https://hal.inria.fr/hal-01256344 (cit. on p. 92).

Bimbó, Katalin (2015). "The decidability of the intensional fragment of classical linear logic". In: *Theoretical Computer Science* 597, pp. 1–17 (cit. on p. 79).

Blackburn, Patrick (2000). "Representation, Reasoning, and Relational Structures: a Hybrid Logic Manifesto". In: *Logic Journal of IGPL* 8.3, pp. 339–365. DOI: 10.1093/jigpal/8.3.339 (cit. on p. 308).

Bloomfield, Leonard (1933). *Language*. Chicago: University of Chicago Press (cit. on p. 31).

Bos, Johan (1995). "Predicate Logic Unplugged". In: *Proceedings of the 10th Amsterdam Colloquium*. URL: http://www.let.rug.nl/bos/pubs/Bos1996AmCo.pdf (cit. on pp. 114, 148).

Brainerd, Walter S. (1969). "Tree generating regular systems". In: *Information and Control* 14.2, pp. 217–231. URL: http://calhoun.nps.edu/bitstream/handle/10945/40135/Brainerd_Tree_Generating.pdf?sequence=1 (cit. on pp. 35, 36).

Carlson, Lynn and Daniel Marcu (2001). *Discourse Tagging Reference Manual*. Tech. rep. ISI-TR-545. University of Southern California Information Sciences Institute. URL: http://www.isi.edu/~marcu/discourse/tagging-ref-manual.pdf (cit. on p. 98).

Charolles, Michel (2005). "Framing adverbials and their role in discourse cohesion, from connection to forward labelling". In: *Proceedings of Symposium on the Exploration and Modelling of Meaning*. Biarritz, France. URL: http://w3.erss.univ-tlse2.fr:8080/index.jsp?perso=bras&subURL=sem05/proceedings-final/02-Charolles.pdf (cit. on pp. 7, 95, 176, 189).

Chomsky, Noam (1956). "Three models for the description of language". In: *IRE Transactions on Information Theory* 2. http://www.chomsky.info/articles/195609--.pdf – last visited 14th January 2009, pp. 113–124 (cit. on pp. 26, 31, 32).

Chomsky, Noam (1959). "On certain formal properties of grammars". In: *Information and Control* 2.2, pp. 137–167. DOI: dx.doi.org/10.1016/S0019-9958(59)90362-6 (cit. on p. 35).

Curry, Haskell B. (1960). "Some Logical Aspects of Grammatical Structure". In: *Journal of Symbolic Logic* 25.4, pp. 341–341 (cit. on pp. 3, 52).

Dale, Robert (1995). "Introduction to Natural Language Generation. Barcelona, ESSLLI 1995". URL: http://comp.mq.edu.au/~rdale/teaching/esslli/ (cit. on p. 18).

Danlos, Laurence (1998). "G-TAG : un formalisme lexicalisé pour la génération de textes inspiré de TAG". In: *Traitement Automatique des Langues* 39.2. Article dans revue scientifique avec comité de lecture., 28 p. URL: https://hal.inria.fr/inria-00098489 (cit. on pp. 5, 20, 153, 163, 196, 323).

Danlos, Laurence (2000). "G-TAG: A lexicalized formalism for text generation inspired by Tree Adjoining Grammar". In: *Tree Adjoining Grammars: Formalisms, Linguistic Analysis, and Processing*. Ed. by Anne Abeillé and Owen Rambow. Vol. 107. CSLI Lecture Notes. CSLI Publications, pp. 343–370. URL: http://www.linguist.jussieu.fr/~danlos/Dossier%20publis/G-TAG-eng'01.pdf (cit. on pp. 20, 153, 169, 203).

Danlos, Laurence (2009). "D-STAG : un formalisme d'analyse automatique de discours basé sur les TAG synchrones". In: *Revue TAL* 50.1, pp. 111–143. URL: https://hal.inria.fr/inria-00524743 (cit. on pp. 90, 94, 174, 323).

Danlos, Laurence (2011). "D-STAG: a Formalism for Discourse Analysis based on SDRT and using Synchronous TAG". In: *14th conference on Formal Grammar - FG 2009*. Ed. by Philippe de Groote, Markus Egg, and Laura Kallmeyer. Vol. 5591. LNCS/LNAI. Springer, pp. 64–84. DOI: 10.1007/978-3-642-20169-1_5. URL: http:

//webloria.loria.fr/~degroote/FG09/Danlos.pdf (cit. on pp. 5, 17, 94, 95, 174, 175, 180, 181, 250).

Danlos, Laurence (2012). *Annotation manuelle ou semi-automatique du FDTB: Problèmes à l'interface syntaxe-sémantique pour les connecteurs de discours*. Université Paris Diderot - Paris 7. URL: http://mathilde.dargnat.free.fr/index_fichiers/Danlos-ppt.pdf (cit. on p. 93).

Danlos, Laurence (2013). "Connecteurs de discours adverbiaux: Problèmes à l'interface syntaxe-sémantique". In: *Linguisticae Investigationes*. Adverbes et compléments adverbiaux 36.2, pp. 261–275. URL: https://hal.inria.fr/hal-00932184 (cit. on pp. 91, 92, 93).

Danlos, Laurence, Frédéric Meunier, and Vanessa Combet (2011). "EasyText: an Operational NLG System". In: *ENLG 2011 - 13th European Workshop on Natural Language Generation*. Nancy, France. URL: https://hal.inria.fr/inria-00614760 (cit. on pp. 19, 153).

de Groote, Philippe (2001). "Towards abstract categorial grammars". In: *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter*. Colloque avec actes et comité de lecture. internationale. Toulouse, France, pp. 148–155. URL: https://hal.inria.fr/inria-00100529 (cit. on pp. 2, 3, 52, 53, 60, 61, 64, 69, 77, 78, 80, 85, 323).

de Groote, Philippe (2002). "Tree-Adjoining Grammars as Abstract Categorial Grammars". In: *Proceedings of the Sixth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+6)*. Università di Venezia, pp. 145–150. URL: http://www.loria.fr/equipes/calligramme/acg/publications/2002-tag+6.pdf (cit. on pp. 4, 53, 57, 69, 70, 77, 78).

de Groote, Philippe (2006). "Towards a Montagovian Account of Dynamics". In: *Proceedings of Semantics and Linguistic Theory XVI*. Ed. by Masayuki Gibson and Jonathan Howell (cit. on pp. 288, 320, 325).

de Groote, Philippe and Sarah Maarek (2007). "Type-theoretic extensions of Abstract Categorial Grammars". URL: https://hal.inria.fr/inria-00187759 (cit. on p. 201).

de Groote, Philippe, Sarah Maarek, and Ryo Yoshinaka (2007). "On two Extensions of Abstract Categorial Grammars". In: *14th International Conference on Logic for Programming, Artificial Intel ligence and Reasoning - LPAR 2007*. Ed. by Nachum Dershowitz and Andrei Voronkov. Vol. 4790. Yerevan, Armenia: Springer, pp. 273–287. DOI: 10.1007/978-3-540-75560-9_21. URL: https://hal.inria.fr/inria-00609120 (cit. on p. 201).

de Groote, Philippe and Sylvain Pogodalla (2003). "m-Linear Context-Free Rewriting Systems as Abstract Categorial Grammars". In: *Proceedings of Mathematics of Language - MOL-8*. Ed. by Richard T. Oehrle and James Rogers. Colloque avec actes et comité de lecture. internationale. Bloomington, Indiana, United States, pp. 71–80. URL: https://hal.inria.fr/inria-00107690 (cit. on p. 77).

de Groote, Philippe and Sylvain Pogodalla (2004). "On the expressive power of Abstract Categorial Grammars: Representing context-free formalisms". In: *Journal of Logic, Language and Information* 13.4. http://www.springerlink.com/content/1572-9583/, pp. 421–438. DOI: 10.1007/s10849-004-2114-x. URL: https://hal.inria.fr/inria-00112956 (cit. on pp. 4, 53, 77, 78).

Dines, Nikhil et al. (2005). "Attribution and the (Non-)Alignment of Syntactic and Discourse Arguments of Connectives". In: *Proceedings of the Workshop on Frontiers in Corpus Annotations II: Pie in the Sky*. CorpusAnno '05. Ann Arbor, Michigan: Association for Computational Linguistics, pp. 29–36 (cit. on p. 92).

Forbes, Katherine et al. (2003). "D-LTAG System: Discourse Parsing with a Lexicalized Tree-Adjoining Grammar". In: *Journal of Logic, Language and Information* 12.3. Special Issue: Discourse and Information Structure, pp. 261–279. DOI: 10.1023/A:1024137719751. URL: http://www.coli.uni-saarland.de/~korbay/esslli01-wsh/Jolli/Final/forbes-etal.pdf (cit. on pp. 131, 135, 308, 325).

Forbes-Riley, Katherine, Bonnie Webber, and Aravind Joshi (2006). "Computing Discourse Semantics: The Predicate-Argument Semantics of Discourse Connectives in D-LTAG". In: *Journal of Semantics* 23.1, pp. 55–106. DOI: 10.1093/jos/ffh032. eprint: http://jos.oxfordjournals.org/content/23/1/55.full.pdf+html (cit. on pp. 131, 144).

Girard, Jean-Yves (1987a). "Linear Logic". In: *Theoretical Computer Science* 50, pp. 1–102 (cit. on p. 79).

Girard, Jean-Yves (1987b). "Multiplicatives". In: *Logic and Computer Science: New Trends and Applications*. Ed. by G. Lolli. Rendiconti del Seminario Matematico dell'Università e Politecnico di Torino, pp. 11–34 (cit. on p. 79).

Goldreich, Oded (2008). *Computational Complexity: A Conceptual Perspective*. 1st ed. New York, NY, USA: Cambridge University Press (cit. on pp. 26, 33).

Greibach, Sheila A. (1965). "A New Normal-Form Theorem for Context-Free Phrase Structure Grammars". In: *J. ACM* 12.1, pp. 42–52. DOI: 10.1145/321250.321254 (cit. on p. 35).

Groenendijk, Jeroen and Martin Stokhof (1991). "Dynamic predicate logic". English. In: *Linguistics and Philosophy* 14.1, pp. 39–100. DOI: 10.1007/BF00628304 (cit. on pp. 17, 107, 321).

Hobbs, Jerry R. (1985). *On the Coherence and Structure of Discourse*. Tech. rep. CSLI-85-37. Center for the Study of Language and Information, Stanford, CA. URL: http://www.isi.edu/~hobbs/ocsd.pdf (cit. on pp. 15, 16).

Hopcroft, John E., Rajeev Motwani, and Jeffrey D. Ullman (2006). *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. (cit. on p. 26).

Joshi, Aravind K. (1985). "Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions?" In: *Natural language parsing*. Ed. by David R. Dowty, Lauri Karttunen, and Arnold M. Zwicky. Cambridge Books Online. Cambridge University Press, pp. 206–250 (cit. on pp. 26, 36).

Joshi, Aravind K. (1994). "Current Issues in Computational Linguistics: In Honour of Don Walker". In: ed. by Antonio Zampolli, Nicoletta Calzolari, and Martha Palmer. Dordrecht: Springer Netherlands. Chap. Some Recent Trends In Natural Language Processing, pp. 491–501. DOI: 10.1007/978-0-585-35958-8_26 (cit. on p. 37).

Joshi, Aravind K., Laura Kallmeyer, and Maribel Romero (2003). "Flexible Composition in LTAG: Quantifier Scope and Inverse Linking". In: *Proceedings of the Fifth International Workshop on Computational Semantics IWCS-5*. Ed. by Harry Bunt, Ielka van der Sluis, and Roser Morante (cit. on p. 149).

Joshi, Aravind K., Leon S. Levy, and Masako Takahashi (1975). "Tree Adjunct Grammars". In: *Journal of Computer and System Sciences* 10.1, pp. 136–163. DOI: 10.1016/S0022-0000(75)80019-5 (cit. on pp. 2, 26, 29, 37).

Joshi, Aravind K. and Yves Schabes (1997). "Tree-Adjoining Grammars". English. In: *Handbook of Formal Languages*. Ed. by Grzegorz Rozenberg and Arto Salomaa. Springer Berlin Heidelberg, pp. 69–123. DOI: 10.1007/978-3-642-59126-6_2. URL: http://www.cis.upenn.edu/~joshi/joshi-schabes-tag-97.pdf (cit. on pp. 37, 38, 39, 43).

Joshi, Aravind, Laura Kallmeyer, and Maribel Romero (2007). "Flexible Composition In Ltag: Quantifier Scope and Inverse Linking". English. In: *Computing Meaning*. Ed. by Harry Bunt and Reinhard Muskens. Vol. 83. Studies in Linguistics and Philosophy. Springer Netherlands, pp. 233–256. DOI: 10.1007/978-1-4020-5958-2_11 (cit. on p. 143).

Jurafsky, Daniel and James H. Martin (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 1st. Upper Saddle River, NJ, USA: Prentice Hall PTR (cit. on p. 35).

Kallmeyer, Laura (2010). *Parsing Beyond Context-Free Grammars*. 1st. Springer Publishing Company, Incorporated. DOI: 10.1007/978-3-642-14846-0 (cit. on p. 46).

Kamp, Hans (1981). "Référence Temporelle et Représentation Du Discours". In: *Language* 64, pp. 39–64 (cit. on p. 116).

Kamp, Hans (1988). "Discourse Representation Theory: What it is and where it ought to go". In: *Natural Language and the Computer*. Ed. by A. Bläser. Berlin: Springer, pp. 84–111 (cit. on pp. 17, 116, 321).

Kamp, Hans and Uwe Reyle (1993). *From Discourse to Logic: Introduction to Model-theoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Vol. 42. Studies in Linguistics and Philosophy. Dordrecht: Springer Netherlands. DOI: 10.1007/978-94-017-1616-1 (cit. on pp. 107, 116).

Kamp, Hans, Josef van Genabith, and Uwe Reyle (2011). *Discourse Representation Theory*. DOI: 10.1007/978-94-007-0485-5_3 (cit. on pp. 17, 116).

Kanazawa, Makoto (2006). "Abstract Families of Abstract Categorial Languages". In: vol. 165. Proceedings of the 13th Workshop on Logic, Language, Information and Computation (WoLLIC 2006) Logic, Language, Information and Computation 2006, pp. 65–80. URL: http://research.nii.ac.jp/~kanazawa/publications/afacl.pdf (cit. on p. 77).

Kanazawa, Makoto (2007). "Parsing and Generation as Datalog Queries". In: *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL)*. Prague, Czech Republic: Association for Computational Linguistics, pp. 176–183. acl: P07-1023. URL: http://www.aclweb.org/anthology/P07-1023 (cit. on pp. 5, 11, 53, 77, 79, 80, 82, 85, 197, 209, 219, 225, 236, 325).

Kanazawa, Makoto (2015). "Syntactic Features for Regular Constraints and an Approximation of Directional Slashes in Abstract Categorial Grammars". In: *Proceedings for ESSLLI 2015 Workshop 'Empirical Advances in Categorial Grammars'*. Ed. by Yusuke Kubota and Robert Levine, pp. 34–70. URL: http://research.nii.ac.jp/~kanazawa/publications/approx_proc.pdf (cit. on p. 198).

Kanazawa, Makoto and Ryo Yoshinaka (2005). *Lexicalization of second-order ACGs.* Technical Report NII-2005-012E. Tokyo, Japan: National Institute of Informatics, pp. 1–18. URL: http://www.nii.ac.jp/TechReports/public_html/05-012E.pdf (cit. on pp. 77, 78).

Ker, Andrew D. (2009). "Lambda Calculus, University of Oxford". URL: http://www.cs.ox.ac.uk/andrew.ker/docs/lambdacalculus-lecture-notes-ht2009.pdf (cit. on p. 182).

Kobele, Gregory M. (2012). "Idioms and extended transducers". In: *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+11).* Paris, France, pp. 153–161 (cit. on pp. 207, 259).

Kreinovich, Vladik et al. (1998). *Computational complexity and feasibility of data processing and interval computations.* Vol. 10. Applied optimization. Dordrecht, Boston: Kluwer Academic Publishers (cit. on p. 26).

Kruijff, Geert-jan M. (2001). *A Categorial-Modal Architecture of Informativity.* PhD thesis, Charles University, Prague (cit. on p. 308).

Lambek, Joachim (1958). "The Mathematics of Sentence Structure". In: *Americal Mathematical Monthly* 65, pp. 154–170. DOI: 10.2307/2271418 (cit. on pp. 26, 52, 308).

Lascarides, Alex and Nicholas Asher (2007). "Segmented Discourse Representation Theory: Dynamic Semantics with Discourse Structure". In: *Computing Meaning: Volume 3.* Ed. by Harry Bunt and Reinhard Muskens. Kluwer Academic Publishers, pp. 87–124. URL: http://homepages.inf.ed.ac.uk/alex/papers/iwcs4.pdf (cit. on pp. 109, 116).

Maletti, Andreas and Joost Engelfriet (2012). "Strong Lexicalization of Tree Adjoining Grammars". In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers).* Jeju Island, Korea: Association for Computational Linguistics, pp. 506–515. URL: http://www.aclweb.org/anthology/P12-1053 (cit. on p. 78).

Mann, William C. and Sandra A. Thompson (1986). "Assertions from Discourse Structure". In: *Strategic Computing - Natural Language Workshop: Proceedings of a Workshop Held at Marina del Rey, California, May 1-2, 1986*, pp. 257–270. URL: http://www.aclweb.org/anthology/H86-1024 (cit. on pp. 16, 98).

Mann, William C. and Sandra A. Thompson (1987). *Rhetorical Structure Theory: A Theory of Text Organization.* Tech. rep. ISI/RS-87-190. Information Sciences Institute (cit. on pp. 16, 17, 97).

Mann, William C. and Sandra A. Thompson (1988). "Rhetorical Structure Theory: Toward a functional theory of text organization". In: *Text* 8.3, pp. 243–281. URL: http://semanticsarchive.net/Archive/GMyNDBjO/RST%20towards%20a%20functional%20theory%20of%20text%20organization.pdf (cit. on pp. 97, 100, 101).

Marcu, Daniel (1997). "The Rhetorical Parsing, Summarization, and Generation of Natural Language Texts". Ph.D. dissertation. Department of Computer Science. URL: https://tspace.library.utoronto.ca/bitstream/1807/12342/1/NQ35238.pdf (cit. on pp. 16, 97, 102, 103, 104, 105, 106).

Marcu, Daniel (2000). *The Theory and Practice of Discourse Parsing and Summarization.* Cambridge, MA, USA: MIT Press (cit. on pp. 102, 104, 106).

McKeown, Kathleen R. (1992). *Text generation - using discourse strategies and focus constraints to generate natural language text.* Studies in natural language processing. Cambridge University Press, pp. I–X, 1–246 (cit. on p. 18).

Meunier, Frédéric (1997). "Implantation du formalisme de génération G-TAG". PhD thesis. Université Paris 7 — Denis Diderot (cit. on p. 153).

Michaelis, Jens (2001). "Transforming Linear Context-Free Rewriting Systems into Minimalist Grammars". English. In: *Logical Aspects of Computational Linguistics.* Ed. by Philippe de Groote, Glyn Morrill, and Christian Retoré. Vol. 2099. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 228–244. DOI: 10.1007/3-540-48199-0_14 (cit. on p. 53).

Miltsakaki, E. et al. (2004). "Annotating Discourse Connectives and Their Arguments". In: *HLT-NAACL 2004 Workshop: Frontiers in Corpus Annotation.* Ed. by A. Meyers. Boston, Massachusetts, USA: Association for Computational Linguistics, pp. 9–16. URL: http://www.aclweb.org/anthology/W04-2703 (cit. on p. 89).

Montague, Richard (1970a). "English as a Formal Language". In: *Linguaggi Nella Società e Nella Tecnica.* Ed. by B. Visentini. Edizioni di Communita, pp. 188–221 (cit. on p. 52).

Montague, Richard (1970b). "Universal Grammar". In: *Theoria* 36.3, pp. 373–398 (cit. on p. 52).

Montague, Richard (1973). "The Proper Treatment of Quantification in Ordinary English". In: *Formal Philosophy: Selected Papers of Richard Montague.* Ed. by Richmond Thomason. New Haven, CT: Yale University Press, pp. 247–270 (cit. on pp. 3, 17, 52, 53, 80, 81, 82, 219, 289).

Moortgat, Michael (1997). "Categorial Type Logics". In: *Handbook of Logic and Language.* Elsevier, pp. 93–177 (cit. on p. 52).

Morrill, G.V. (1994). *Type Logical Grammar.* 3Island Press (cit. on p. 52).

Muskens, Reinhard (2001). "Lambda-Grammars and the Syntax-Semantics Interface". English. In: *Proceedings of the Thirteenth Amsterdam Colloquium.* Ed. by R. van Rooy and M. Stokhof. Amsterdam, pp. 150–155 (cit. on p. 52).

Nakatsu, Crytal and Michael White (2010). "Generating with Discourse Combinatory Categorial Grammar". In: *Linguistic Issues in Language Technology* 4.1, pp. 1–64. URL: http://journals.linguisticsociety.org/elanguage/lilt/article/view/1277.html (cit. on pp. 307, 308, 309, 311, 312, 313).

Nesson, Rebecca, Giorgio Satta, and Stuart M. Shieber (2010). "Complexity, Parsing, and Factorization of Tree-local Multi-component Tree-adjoining Grammar". In: *Comput. Linguist.* 36.3, pp. 443–480. DOI: 10.1162/coli_a_00005 (cit. on p. 47).

Nicholas, Nick (1994). "Problems in the application of Rhetorical Structure Theory to text generation". MA thesis. Australia: University of Melbourne (cit. on p. 97).

Nijholt, Anton (1980). *Context-Free Grammars: Covers, Normal Forms, and Parsing.* Vol. 93. Lecture Notes in Computer Science. Berlin, Germany: Springer Verlag. URL: http://doc.utwente.nl/66928/ (cit. on p. 33).

Oehrle, Dick (1995). "Some 3-Dimensional Systems of Labelled Deduction". In: *Logic Journal of IGPL* 3.2-3, pp. 429–448. DOI: 10.1093/jigpal/3.2-3.429 (cit. on p. 52).

Oehrle, Richard T. (1988). "Multi-Dimensional Compositional Functions as a Basis for Grammatical Analysis". English. In: *Categorial Grammars and Natural Language*

Structures. Ed. by Richard T. Oehrle, Emmon Bach, and Deirdre Wheeler. Vol. 32. Studies in Linguistics and Philosophy. Springer Netherlands, pp. 349–389. DOI: 10.1007/978-94-015-6878-4_13 (cit. on p. 52).

Oehrle, Richard T. (1994). "Term-labeled categorial type systems". English. In: *Linguistics and Philosophy* 17.6, pp. 633–678. DOI: 10.1007/BF00985321 (cit. on p. 52).

Pogodalla, Sylvain (2004). "Computing Semantic Representation: Towards ACG Abstract Terms as Derivation Trees". In: *Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms - TAG+7*, pp. 64–71. URL: http://hal.inria.fr/inria-00107768 (cit. on pp. 5, 53, 80, 323).

Pogodalla, Sylvain (2009). "Advances in Abstract Categorial Grammars: Language Theory and Linguistic Modeling. ESSLLI 2009 Lecture Notes, Part II". URL: http://hal.inria.fr/hal-00749297 (cit. on pp. 5, 53, 80, 85, 196, 216, 219, 251).

Polanyi, Livia (1985). "A theory of discourse structure and discourse coherence". In: *Papers from the General Session at the 21st Regional Meeting of the Chicago Linguistics Society*. Ed. by P. D. Kroeber, W. H. Eilfort, and K. L. Peterson (cit. on p. 112).

Pompigne, Florent (2013). "Logical modelization of language and Abstract Categorial Grammars". Theses. Université de Lorraine. URL: https://tel.archives-ouvertes.fr/tel-00921040 (cit. on pp. 198, 201).

Prasad, Rashmi et al. (2008). "The Penn Discourse TreeBank 2.0." In: *LREC*. European Language Resources Association (cit. on pp. 88, 89).

Prévot, Laurent and L. Vieu (2008). "The moving right frontier". In: *PRAGMATICS AND BEYOND NEW SERIES*. Vol. 172. Amsterdam, p. 53. URL: https://hal.archives-ouvertes.fr/hal-01231937 (cit. on p. 109).

Ranta, Aarne (1994). *Type-theoretical Grammar*. Indices (Claredon). Clarendon Press (cit. on p. 52).

Salvati, Sylvain (2005). "Problèmes de filtrage et problèmes d'analyse pour les grammaires catógorielles abstraites". PhD thesis. Institut National Polytechnique de Lorraine. URL: http://www.labri.fr/perso/salvati/downloads/articles/these.pdf (cit. on pp. 5, 53, 77, 78, 79, 325).

Salvati, Sylvain (2010). "The Mathematics of Language: 10th and 11th Biennial Conference, MOL 10, Los Angeles, CA, USA, July 28-30, 2007, and MOL 11, Bielefeld, Germany, August 20-21, 2009, Revised Selected Papers". In: ed. by Christian Ebert, Gerhard Jäger, and Jens Michaelis. Berlin, Heidelberg: Springer Berlin Heidelberg. Chap. A Note on the Complexity of Abstract Categorial Grammars, pp. 266–271. DOI: 10.1007/978-3-642-14322-9_20 (cit. on p. 79).

Schabes, Yves, Anne Abeillé, and Aravind K. Joshi (1988). "Parsing Strategies with 'Lexicalized' Grammars: Application to Tree Adjoining Grammars". In: *Coling Budapest 1988 Volume 2: International Conference on Computational Linguistics*, pp. 578–583. URL: http://www.aclweb.org/anthology/C88-2121 (cit. on pp. 43, 44).

Schabes, Yves and Aravind K. Joshi (1988). "An Earley-type Parsing Algorithm for Tree Adjoining Grammars". In: *Proceedings of the 26th Annual Meeting on Association for Computational Linguistics*. ACL '88. Buffalo, New York: Association for Computational Linguistics, pp. 258–269. DOI: 10.3115/982023.982055 (cit. on pp. 37, 43).

Schabes, Yves and Aravind K. Joshi (1991). "Current Issues in Parsing Technology". In: ed. by Masaru Tomita. Boston, MA: Springer US. Chap. Parsing with Lexicalized

Tree Adjoining Grammar, pp. 25–47. DOI: 10.1007/978-1-4615-3986-5_3 (cit. on p. 43).

Schabes, Yves and Stuart M. Shieber (1994). "An Alternative Conception of Tree-Adjoining Derivation". In: *Computational Linguistics* 20.1, pp. 91–124. acl: J94-1004. URL: http://www.aclweb.org/anthology/J94-1004 (cit. on p. 60).

Schlenker, Philippe (2011). "DRT with local contexts". English. In: *Natural Language Semantics* 19.4, pp. 373–392. DOI: 10.1007/s11050-011-9069-7 (cit. on p. 21).

Shieber, Stuart M. (1985). "Evidence against the context-freeness of natural language". English. In: *Linguistics and Philosophy* 8.3, pp. 333–343. DOI: 10.1007/BF00630917 (cit. on pp. 26, 37).

Shieber, Stuart M. (1994). "Restricting the Weak-Generative Capacity of Synchronous Tree-Adjoining Grammars". In: *Computational Intelligence* 10.4, pp. 371–385. DOI: 10.1111/j.1467-8640.1994.tb00003.x (cit. on p. 49).

Shieber, Stuart M. and Yves Schabes (1990). "Synchronous Tree-Adjoining Grammars". In: *COLNG 1990 Volume 3: Papers presented to the 13th International Conference on Computational Linguistics*, pp. 253–258. URL: http://www.aclweb.org/anthology/C90-3045 (cit. on pp. 2, 47, 174, 176, 250).

Soricut, Radu and Daniel Marcu (2003). "Sentence Level Discourse Parsing using Syntactic and Lexical Information". In: *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*. URL: http://www.aclweb.org/anthology/N03-1030 (cit. on p. 95).

Steedman, Mark (1987). "Combinatory grammars and parasitic gaps". English. In: *Natural Language & Linguistic Theory* 5.3, pp. 403–439. DOI: 10.1007/BF00134555 (cit. on p. 308).

Steedman, Mark (2000). "Information Structure and the Syntax-Phonology Interface". In: *Linguistic Inquiry* 31.4, pp. 649–689 (cit. on p. 137).

Vijay-Shanker, K., David J. Weir, and Aravind K. Joshi (1987). "Characterizing Structural Descriptions Produced by Various Grammatical Formalisms". In: *Proceedings of the 25th Annual Meeting on Association for Computational Linguistics*. ACL '87. Stanford, California: Association for Computational Linguistics, pp. 104–111. DOI: 10.3115/981175.981190 (cit. on pp. 46, 235).

Webber, Bonnie Lynn (2004). "D-LTAG: extending lexicalized TAG to discourse". In: *Cognitive Science* 28.5. 2003 Rumelhart Prize Special Issue Honoring Aravind K. Joshi, pp. 751–779. DOI: 10.1207/s15516709cog2805_6. URL: http://homepages.inf.ed.ac.uk/bonnie/cogsci28bw.pdf (cit. on pp. 44, 89, 90, 131, 134, 135, 307).

Webber, Bonnie Lynn and Aravind K. Joshi (1998). "Anchoring a Lexicalized Tree-Adjoining Grammar for Discourse". In: *Discourse Relations and Discourse Markers*, pp. 86–92. URL: http://www.aclweb.org/anthology/W98-0315 (cit. on pp. 89, 90, 131).

Webber, Bonnie, Marcus Egg, and Evangelia Kordoni (2012). "Discourse structure and language technology". In: *Natural Language Engineering* 18 (04), pp. 437–490. DOI: 10.1017/S1351324911000337 (cit. on p. 92).

Webber, Bonnie and Rashmi Prasad (2009). "Discourse Structure: Swings and Roundabouts". In: *Linguistics, Special issue on Structuring Information in Discourse: the Ex-*

plicit/Implicit Dimension. Ed. by Cathrine Fabricius-Hansen and Bergljot Behrens. URL: http://folk.uio.no/larsbun/osla-1-1.pdf (cit. on pp. 88, 91).

Webber, Bonnie et al. (1999). "Discourse Relations: A Structural and Presuppositional Account Using Lexicalised TAG". In: *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*. College Park, Maryland, USA: Association for Computational Linguistics, pp. 41–48. URL: http://www.aclweb.org/anthology/P99-1006 (cit. on pp. 89, 131).

Webber, Bonnie et al. (2003). "Anaphora and Discourse Structure". In: *Computational Linguistics* 29.4, pp. 545–587. DOI: 10.1162/089120103322753347 (cit. on pp. 16, 89, 90, 91, 131, 146, 148, 151, 307).

Weir, David Jeremy (1988). "Characterizing Mildly Context-Sensitive Grammar Formalisms". Supervisor: Aravind K. Joshi. PhD thesis. Philadelphia, PA, USA (cit. on pp. 46, 53).

Wellner, Ben and James Pustejovsky (2007). "Automatically Identifying the Arguments of Discourse Connectives". In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. Prague, Czech Republic: Association for Computational Linguistics, pp. 92–101. URL: http://www.aclweb.org/anthology/D07-1010 (cit. on p. 95).

Wolf, Florian and Edward Gibson (2004). "Representing discourse coherence: A corpus-based analysis". In: *Proceedings of Coling 2004*. Geneva, Switzerland: COLING, pp. 134–140. URL: http://www.aclweb.org/anthology/C04-1020 (cit. on p. 17).

XTAG-Group (1998). *A Lexicalized Tree Adjoining Grammar for English*. Tech. rep. University of Pennsylvania. URL: http://arxiv.org/abs/cs.CL/9809024 (cit. on pp. 39, 73, 199, 234).

Yoshinaka, Ryo (2006). "Extensions and Restrictions of Abstract Categorial Grammars". PhD thesis. Tokyo, Japan: University of Tokyo. URL: http://www.iip.ist.i.kyoto-u.ac.jp/member/ry/PhD_Yoshinaka.pdf (cit. on pp. 77, 78, 79).