Constituency and Dependency Relationship from a Tree Adjoining Grammar and Abstract Categorial Grammar Perspective

Aleksandre Maskharashvili, Sylvain Pogodalla

Constituency and Dependency Relationship from a Tree Adjoining Grammar and Abstract Categorial Grammar Perspective*

Aleksandre Maskharashvili and **Sylvain Pogodalla**
INRIA, 54600 Villers-lès-Nancy, France
LORIA, UMR 7503, 54506 Vandœuvre-lès-Nancy, France
Sylvain.Pogodalla@inria.fr
Aleksandre.Maskharashvili@inria.fr

Abstract

This paper gives an Abstract Categorial Grammar (ACG) account of (Kallmeyer and Kuhlmann, 2012)'s process of transformation of the derivation trees of Tree Adjoining Grammar (TAG) into dependency trees. We make explicit how the requirement of keeping a direct interpretation of dependency trees into strings results into lexical ambiguity. Since the ACG framework has already been used to provide a logical semantics from TAG derivation trees, we have a unified picture where derivation trees and dependency trees are related but independent equivalent ways to account for the same surface–meaning relation.

1 Introduction

Tree Adjoining Grammars (TAG) (Joshi et al., 1975; Joshi and Schabes, 1997) is a tree grammar formalism relying on two operations between trees: *substitution* and *adjunction*. In addition to the tree generated by a sequence of such operations, there is a *derivation tree* which records this sequence. Derivation trees soon appeared as good candidates to encode semantic-like relations between the elementary trees they glue together. However, some mismatch between these trees and the relative scoping of logical connectives and relational symbols, or between these trees and the dependency relations, have been observed. Solving these problems often leads to modifications of derivation tree structures (Schabes and Shieber, 1994; Kallmeyer, 2002; Joshi et al., 2003; Rambow et al., 2001; Chen-Main and Joshi, To appear).

While alternative proposals have succeeded in linking derivation trees to semantic representations using unification (Kallmeyer and Romero, 2004; Kallmeyer and Romero, 2007) or using an encoding (Pogodalla, 2004; Pogodalla, 2009) of TAG into the ACG framework (de Groote, 2001), only recently (Kallmeyer and Kuhlmann, 2012) has proposed a transformation from standard derivation trees to dependency trees.

This paper provides an ACG perspective on this transformation. The goal is twofold. First, it exhibits the underlying lexical blow up of the yield functions associated with the elementary trees in (Kallmeyer and Kuhlmann, 2012). Second, using the same framework as (Pogodalla, 2004; Pogodalla, 2009) allows us to have a shared perspective on a phrase-structure architecture and a dependency one and an equivalence on the surface-meaning relation they define.

2 Abstract Categorial Grammars

ACGs provide a framework in which several grammatical formalisms may be encoded (de Groote and Pogodalla, 2004). They generate languages of linear λ-terms, which generalize both string and tree languages. A key feature is to provide the user direct control over the parse structures of the grammar, the *abstract language*, which allows several grammatical formalisms to be defined in terms of ACG, in particular TAG (de Groote, 2002). We refer the reader to (de Groote, 2001; Pogodalla, 2009) for the details and introduce here only few relevant definitions and notations.

Definition. *A* higher-order linear signature *is defined to be a triple* $\Sigma = \langle A, C, \tau \rangle$*, where:*

- A *is a finite set of atomic types (also noted* A_Σ*),*
- C *is a finite set of constants (also noted* C_Σ*),*
- *and* τ *is a mapping from* C *to* $\mathscr{T}_A$ *the set of types built on* A*:* $\mathscr{T}_A ::= A | \mathscr{T}_A \multimap \mathscr{T}_A$ *(also noted* $\mathscr{T}_\Sigma$*).*

*Work supported by ANR (ANR-12-CORD-0004).

A higher-order linear signature will also be called a vocabulary. $\Lambda(\Sigma)$ *is the set of* λ*-terms built on* Σ*, and for* $t \in \Lambda(\Sigma)$ *and* $\alpha \in \mathscr{T}_\Sigma$ *such that* t *has type* α*, we note* $t :_\Sigma \alpha$ *(the* Σ *subscript is omitted when it is obvious from the context).*

Definition. *An* abstract categorial grammar *is a quadruple* $\mathscr{G} = \langle \Sigma, \Xi, \mathscr{L}, s \rangle$ *where:*

1. Σ *and* Ξ *are two higher-order linear signatures, which are called the* abstract vocabulary *and the* object vocabulary*, respectively;*
2. $\mathscr{L} : \Sigma \longrightarrow \Xi$ *is a lexicon from the abstract vocabulary to the object vocabulary. It is a homomorphism*[1] *that maps types and terms built on* Σ *to types and terms built on* Ξ*. We note* $t :=_\mathscr{G} u$ *if* $\mathscr{L}(t) = u$ *and omit the* $\mathscr{G}$ *subscript if obvious from the context.*
3. $s \in \mathscr{T}_\Sigma$ *is a type of the abstract vocabulary, which is called the* distinguished type *of the grammar.*

Definition. *The* abstract language *of an ACG* $\mathscr{G} = \langle \Sigma, \Xi, \mathscr{L}, s \rangle$ *is* $\mathcal{A}(\mathscr{G}) = \{t \in \Lambda(\Sigma) \mid t :_\Sigma s\}$

The object language *of the grammar* $\mathcal{O}(\mathscr{G}) = \{t \in \Lambda(\Xi) \mid \exists u \in \mathcal{A}(\mathscr{G}).\, t = \mathscr{L}_G(u)\}$

Since there is no structural difference between the abstract and the object vocabulary as they both are higher-order signatures, ACGs can be combined in different ways. Either by having a same abstract vocabulary shared by several ACGs in order to make two object terms (for instance a string and a logical formula) share the same underlying structure as $\mathscr{G}_{\textit{d-ed trees}}$ and $\mathscr{G}_{\textit{Log}}$ in Fig. 1. Or by making the abstract vocabulary of an ACG the object vocabulary of another ACG, allowing the latter to control the admissible structures of the former, as $\mathscr{G}_{\textit{yield}}$ and $\mathscr{G}_{\textit{d-ed trees}}$ in Fig. 1.

3 TAG as ACG

As Fig. 1 shows, the encoding of TAG into ACG uses two ACGs $\mathscr{G}_{\textit{d-ed trees}} = \langle \Sigma_{\textit{der}\theta}, \Sigma_{\textit{trees}}, \mathscr{L}_{\textit{d-ed trees}}, \mathsf{s} \rangle$ and $\mathscr{G}_{\textit{yield}} = \langle \Sigma_{\textit{trees}}, \Sigma_{\textit{string}}, \mathscr{L}_{\textit{yield}}, \tau \rangle$. We exemplify the encoding[2] of a TAG analyzing (1)[3]

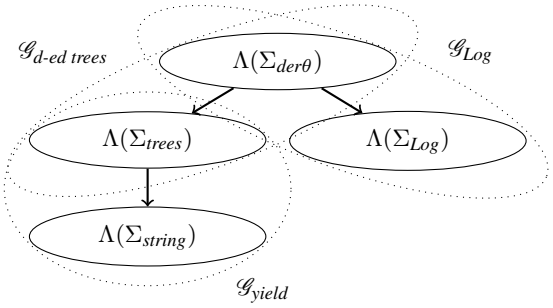


Figure 1: ACG architecture for TAG

(1) John Bill claims Mary seems to love

This sentence is usually analyzed in TAG with a derivation tree where the *to love* component scopes over all the other arguments, and where *claims* and *seems* are unrelated, as Fig. 2(a) shows.

The three higher-order signatures are:

$\Sigma_{\textit{der}\theta}$**:** Its atomic types include s, vp, np, s_A, $\mathsf{vp}_A \ldots$ where the X types stand for the categories X of the nodes where a substitution can occur while the X_A types stand for the categories X of the nodes where an adjunction can occur. For each elementary tree $\gamma_{\textit{lex. entry}}$ it contains a constant $C_{\textit{lex. entry}}$ whose type is based on the adjunction and substitution sites as Table 1 shows. It additionally contains constants $I_X : X_A$ that are meant to provide a fake auxiliary tree on adjunction sites where no adjunction actually takes place in a TAG derivation.

$\Sigma_{\textit{trees}}$**:** Its unique atomic type is τ the type of trees. Then, for any X of arity n belonging to the ranked alphabet describing the elementary trees of the TAG, we have a constant

$$X_n : \overbrace{\tau \multimap \cdots \multimap \tau}^{n \text{ times}} \multimap \tau$$

$\Sigma_{\textit{string}}$**:** Its unique atomic type is σ the type of strings. The constants are the terminal symbols of the TAG (with type σ), the concatenation $+ : \sigma \multimap \sigma \multimap \sigma$ and the empty string $\varepsilon : \sigma$.

Table 1 illustrates $\mathscr{L}_{\textit{d-ed trees}}$.[4] $\mathscr{L}_{\textit{yield}}$ is defined as follows:

- $\mathscr{L}_{\textit{yield}}(\tau) = \sigma$;
- for $n > 0$, $\mathscr{L}_{\textit{yield}}(X_n) = \lambda x_1 \cdots x_n . x_1 + \cdots + x_n$;

[1] In addition to defining $\mathscr{L}$ on the atomic types and on the constants of Σ, we have:
- If $\alpha \multimap \beta \in \mathscr{T}_\Sigma$ then $\mathscr{L}(\alpha \multimap \beta) = \mathscr{L}(\alpha) \multimap \mathscr{L}(\beta)$.
- If $x \in \Lambda(\Sigma)$ (resp. $\lambda x.t \in \Lambda(\Sigma)$ and $t\, u \in \Lambda(\Sigma)$) then $\mathscr{L}(x) = x$ (resp. $\mathscr{L}(\lambda x.t) = \lambda x.\mathscr{L}(t)$ and $\mathscr{L}(t\, u) = \mathscr{L}(t)\, \mathscr{L}(u)$)

with the proviso that for any constant $c :_\Sigma \alpha$ of Σ we have $\mathscr{L}(c) :_\Xi \mathscr{L}(\alpha)$.

[2] We refer the reader to (Pogodalla, 2009) for the details.

[3] The TAG literature typically uses this example, and (Kallmeyer and Kuhlmann, 2012) as well, to show the mismatch between the derivation trees and the expected semantics and the relative scopes of the predicates.

[4] With $\mathscr{L}_{\textit{d-ed trees}}(X_A) = \tau \multimap \tau$ and for any other type X, $\mathscr{L}_{\textit{d-ed trees}}(X_A) = \tau$.

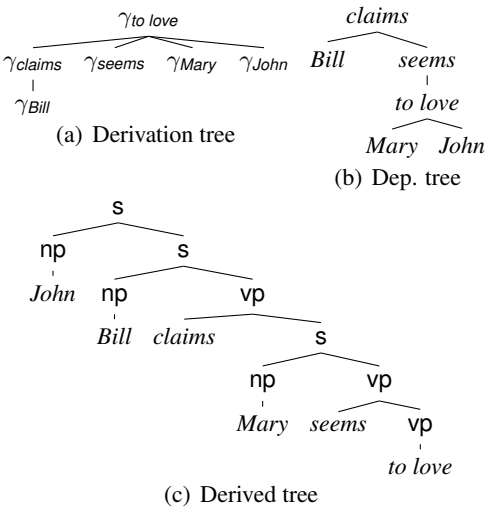
- for $n = 0$, $X_0 : \tau$ represents a terminal symbol and $\mathscr{L}_{yield}(X_0) = X$.

Then, the derivation tree, the derived tree, and the yield of Fig. 2 are represented by:

$$t_0 = C_{\textit{to love}}\ (C_{\textit{claims}}\ I_{\mathsf{s}}\ C_{\textit{Bill}})\ (C_{\textit{seems}}\ I_{\mathsf{vp}})\ C_{\textit{Mary}}\ C_{\textit{John}}$$

$$\mathscr{L}_{\textit{d-ed trees}}(t_0) = \mathsf{s}_2\ (\mathsf{np}_1\ \textit{John})\ (\mathsf{s}_2\ (\mathsf{np}_1\ \textit{Bill})\ (\mathsf{vp}_2\ \textit{claims}\ (\mathsf{s}_2\ (\mathsf{np}_1\ \textit{Mary})\ (\mathsf{vp}_2\ \textit{seems}\ (\mathsf{vp}_1\ \textit{to love}))))$$

$$\mathscr{L}_{yield}(\mathscr{L}_{\textit{d-ed trees}}(t_0)) = \textit{John} + \textit{Bill} + \textit{claims} + \textit{Mary} + \textit{seems} + \textit{to love}$$



(a) Derivation tree

(b) Dep. tree

(c) Derived tree

Figure 2: *John Bill claims Mary seems to love*

4 From Derivation Trees to Dependency Trees

(Kallmeyer and Kuhlmann, 2012)'s process to translate derivation trees into dependency trees is a two-step process. The first one does the actual transformation, using macro-tree transduction, while the second one modifies the way to get the yield from the dependency trees rather than from the derivation ones.

4.1 From Derivation To Dependency Trees

This transformation aims at modeling the differences in scope of the argument between the derivation tree for (1) shown in Fig. 2(a) and the corresponding dependency tree shown in Fig. 2(b). For instance, in the derivation trees, *claims* and *seems* are under the scope of *to love* while in the dependency tree this order is reversed. According to (Kallmeyer and Kuhlmann, 2012), such edge reversal is due to the fact that an edge between a *complement taking adjunction* (CTA) and an initial tree has to be reversed, while the other edges remain unchanged.

Moreover, in case an initial tree accepts several adjunction of CTAs, (Kallmeyer and Kuhlmann, 2012) hypothesizes that the farther from the head a CTA is, the higher it is in the dependency tree. In the case of *to love*, the s node is farther from the head than the vp node. Therefore any adjunction on the s node (e.g. *claims*) should be higher than the one on the vp node (e.g. *seems*) in the dependency tree. We represent the dependency tree for (1) as $t'_0 = d_{\textit{claims}}\ d_{\textit{Bill}}\ (d_{\textit{seems}}(d_{\textit{to love}}\ d_{\textit{John}}\ d_{\textit{Mary}}))$.

In order to do such reversing operations, (Kallmeyer and Kuhlmann, 2012) uses Macro Tree Transducers (MTTs) (Engelfriet and Vogler, 1985). Note that the MTTs they use are *linear*, i.e. non-copying. It means that any node of an input tree cannot be translated more than once. (Yoshinaka, 2006) has shown how to encode such MTTs as the composition $\mathscr{G}' \circ \mathscr{G}^{-1}$ of two ACGs, and we will use a very similar construct.

4.2 The Yield Functions

(Kallmeyer and Kuhlmann, 2012) adds to the transformation from derivation trees to dependency trees the additional constraint that the string associated with a dependency structure is computed directly from the latter, without any reference to the derivation tree. To achieve this, they use two distinct yield functions: $\textit{yield}_{\textit{TAG}}$ from derivation trees to strings, and $\textit{yield}_{\textit{dep}}$ from dependency trees to strings.

Let us imagine an initial tree γ_i and an auxiliary tree γ_a with no substitution nodes. The yield of the derived tree resulting from the operations of the derivation tree γ of Fig. 3 defined in (Kallmeyer and Kuhlmann, 2012) is such that

$$\begin{aligned}\textit{yield}_{\textit{TAG}}(\gamma) &= a_1 + w_1 + a_2 + w_2 + a_3\\ &= (\textit{yield}_{\textit{TAG}}(\gamma_i))(\textit{yield}_{\textit{TAG}}(\gamma_a))\\ &= (\lambda\langle x_1, x_2\rangle.a_1 + x_1 + a_2 + x_2 + a_3)\langle w_1, w_2\rangle\end{aligned}$$

where $\langle x, y\rangle$ denotes a tuple of strings.

Because of the adjunction, the corresponding dependency structure has a reverse order ($\gamma' = \gamma'_a(\gamma'_i)$), the requirement on $\textit{yield}_{\textit{dep}}$ imposes that

$$\begin{aligned}\textit{yield}_{\textit{dep}}(\gamma') &= a_1 + w_1 + a_2 + w_2 + a_3\\ &= (\textit{yield}_{\textit{dep}}(\gamma'_a))(\textit{yield}_{\textit{dep}}(\gamma'_i))\\ &= (\lambda\langle x_1, x_2, x_3\rangle.x_1 + w_1 + x_2 + w_2 + x_3)\langle a_1, a_2, a_3\rangle\end{aligned}$$

In the interpretation of derivation trees as strings, initial trees (with no substitution nodes)

Abstract constants of $\Sigma_{der\theta}$	Their images by $\Sigma_{der\theta}$	The corresponding TAG trees
$C_{John} : \mathsf{np}$	$c_{John} \begin{array}{l} : \tau \\ = \mathsf{np}_1\ John \end{array}$	$\gamma_{John} =$ np | *John*
$C_{seems} : \mathsf{vp}_A \multimap \mathsf{vp}_A$	$c_{seems} \begin{array}{l} : (\tau \multimap \tau) \multimap (\tau \multimap \tau) \\ = \lambda^{o} vx.v\ (\mathsf{vp}_2\ seems\ x) \end{array}$	$\gamma_{seems} =$ vp, *seems*, vp*
$C_{to\ love} : \begin{array}{l} \mathsf{s}_A \multimap \mathsf{vp}_A \multimap \mathsf{np} \\ \multimap \mathsf{np} \multimap \mathsf{s} \end{array}$	$c_{to\ love} \begin{array}{l} : (\tau \multimap \tau) \multimap (\tau \multimap \tau) \multimap \tau \multimap \tau \multimap \tau \\ = \lambda^{o} avso.\mathsf{s}_2\ o \\ \quad (a\ (\mathsf{s}_2\ s\ (v\ (\mathsf{vp}_1\ to\ love)))) \end{array}$	$\gamma_{to\ love} =$ s, np, s, np, vp, *to love*
$C_{claims} : \begin{array}{l} \mathsf{s}_A \multimap \mathsf{vp}_A \\ \multimap \mathsf{np} \multimap \mathsf{s}_A \end{array}$	$c_{claims} \begin{array}{l} : (\tau \multimap \tau) \multimap (\tau \multimap \tau) \multimap \tau \multimap \tau \multimap \tau \\ = \lambda^{o} avsc.a\ (\mathsf{s}_2\ s\ (a\ (\mathsf{vp}_2\ claims\ c))) \end{array}$	$\gamma_{claims} =$ s, np, vp, *claims*, s*
$I_X : X_A$	$\lambda x.x : \tau \multimap \tau$	

Table 1: TAG as ACG: the $\mathscr{L}_{d\text{-}ed\ trees}$ lexicon

are interpreted as functions from tuples of strings into strings, and auxiliary trees as tuples of strings. The interpretation of dependency trees as strings leads us to interpret initial trees as tuples of strings and auxiliary trees as function from tuples of strings to strings.

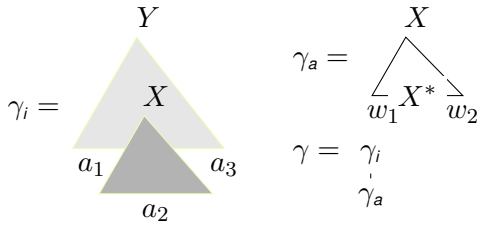


Figure 3: Yield from derivation trees

Indeed, an initial tree can have several adjunction sites. In this case, to be ready for another adjunction after a first one, the first result itself should be a tuple of strings. So an initial tree (with no substitution nodes) with n adjunction sites is interpreted as a $(2n+1)$-tuple of strings. Accordingly, depending on the location where it can adjoin, an auxiliary tree is interpreted as a function from $(2k+1)$-tuple of strings to $(2k-1)$-tuple of strings.

Taking into account that to model trees having the substitution nodes is then just a matter of adding k string parameters where k is the number of substitution nodes in a tree. Then using the interpretation:

$$yield_{dep}(d_{to\ love}) = \lambda x_{11}\ x_{21}.\langle x_{11}, x_{21}, to\ love, \varepsilon, \varepsilon\rangle$$
$$yield_{dep}(d_{seems}) = \lambda\langle x_{11}, x_{12}, x_{13}, x_{14}, x_{15}\rangle.$$
$$\langle x_{11}, x_{12} + seems + x_{13}x_{14}, x_{15}\rangle$$
$$yield_{dep}(d_{claims}) = \lambda x_{21}\ \langle x_{11}, x_{13}, x_{14}\rangle.$$
$$\langle x_{11} + x_{21} + claims + x_{14} + x_{13}\rangle$$

we can check that

$$yield_{dep}(d_{claims}\ d_{Bill}\ (d_{seems}(d_{to\ love}\ d_{Mary}\ d_{John}))) =$$
$$\langle John + Bill + claims + Mary + seems + to\ love\rangle$$

Remark. *The given interpretation of $d_{to\ love}$ is only valid for structures reflecting adjunctions both on the* s *node and on the* vp *node of $\gamma_{to\ love}$. So actually, an initial tree such as $\gamma_{to\ love}$ yields* four *interpretations: one with the two adjunctions (5-tuple), two with one adjunction either on the* vp *node or on the* s *node (3-tuple), and one with no adjunction (1-tuple). The two first cases correspond to the sentences (2a) and (2b).*[5] *Accordingly, we need multiple interpretations for the auxiliary trees, for instance for the two occurrences of seems in (3) where the yield of the last one $yield_{dep}(d_{seems})$ maps a 5-tuple to a 3-tuple, and the yield of the first one maps a 3-tuple to a 3-tuple. And $yield_{dep}(d_{claims})$ maps a 3-tuple to a 1-tuple of strings. We will mimic this behavior by introducing as many different non-terminal symbols for the dependency structures in our ACG setting.*

(2) a. John Bill claims Mary seems to love
b. John Mary seems to love

(3) John Bill seems to claim Mary seems to love

Remark. *Were we not interested in the yields but only in the dependency structures, we wouldn't have to manage this ambiguity. This is true both for (Kallmeyer and Kuhlmann, 2012)'s approach and ours. But as we have here a unified framework for the two-step process they propose, this lexical blow up will result in a multiplicity of types as Section 5 shows.*

[5] As the two other ones are not correct English sentences, we can rule them out. However, from a general perspective, we should take such cases into account.

5 Disambiguated Derivation Trees

In order to encode the MTT acting on derivation trees, we introduce a new *abstract* vocabulary $\Sigma'_{der\theta}$ for *disambiguated derivation trees* as in (Yoshinaka, 2006). Instead of having only one constant for each initial tree as in $\Sigma_{der\theta}$, we have as many of them as adjunction combinations. For instance, $\gamma_{\textit{to love}}$ gives rise to the several constants in $\Sigma'_{der\theta}$:

$C^{11}_{\textit{to love}}$: $\mathsf{s}^{31}_A \multimap \mathsf{vp}^{53}_A \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{s}$
$C^{10}_{\textit{to love}}$: $\mathsf{s}^{31}_A \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{s}$
$C^{01}_{\textit{to love}}$: $\mathsf{vp}^{31}_A \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{s}$
$C^{00}_{\textit{to love}}$: $\mathsf{np} \multimap \mathsf{np} \multimap \mathsf{s}$

Here, $C^{11}_{\textit{to love}}$ is used to model sentences where both adjunctions are performed into $\gamma_{\textit{to love}}$. $C^{10}_{\textit{to love}}$ and $C^{01}_{\textit{to love}}$ are used for sentences where only one adjunction at the s or at the vp node occurs respectively. $C^{00}_{\textit{to love}} : \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{s}$ is used when no adjunction occurs.[6] This really mimics (Yoshinaka, 2006)'s encoding of (Kallmeyer and Kuhlmann, 2012) MTT rules:

$\langle q_0, C_{\textit{to love}}(x_1, x_2, x_3, x_4)\rangle \to$
$\quad \langle q_2, x_2\rangle(\langle q_4, x_4\rangle(d_{\textit{to love}}(\langle q_1, x_1\rangle, \langle q_3, x_3\rangle)))$
$\langle q_0, C_{\textit{to love}}(x_1, x_2, x_3)\rangle \to$
$\quad \langle q_2, x_2\rangle(d_{\textit{to love}}(\langle q_1, x_1\rangle, \langle q_3, x_3\rangle))$
$\langle q_0, C_{\textit{to love}}(x_1, x_2, x_3)\rangle \to$
$\quad \langle q_4, x_4\rangle(d_{\textit{to love}}(\langle q_1, x_1\rangle, \langle q_3, x_3\rangle))$
$\langle q_0, C_{\textit{to love}}(x_1, x_2)\rangle \to$
$\quad d_{\textit{to love}}(\langle q_1, x_1\rangle, \langle q_3, x_3\rangle)$

where the states q_0, q_1, q_2, q_3 and q_4 are given the names s, np, s^{31}_A, np, and vp^{53}_A resp.

Moreover, s^{31}_A, vp^{31}_A, $\ldots$, $\mathsf{vp}^{2(n+1)2(n-1)}_A \ldots$ are designed in order to indicate that a given adjunction has n adjunctions above it (i.e. which scope over it). The superscripts $(2(n+1))(2(n-1))$ express that an adjunction that has n adjunctions above it is translated as a function that takes a $2(n+1)$-tuple as argument and returns a $2(n-1)$-tuple.

To model auxiliary trees which are CTAs we need a different strategy. For each such adjunction tree T we have two sets in $\Sigma'_{der\theta}$: $\mathcal{S}^1_T$ the set of constants which can be adjoined into **initial** trees and $\mathcal{S}^2_T$ the set of constants which can be adjoined into **auxiliary** trees.

For instance, $\gamma_{\textit{seems}}$ would generate $\mathcal{S}^1_{\textit{seems}}$ that includes $C^{11}_{\textit{seems}31}$, $C^{10}_{\textit{seems}31}$, $C^{01}_{\textit{seems}31}$, $C^{00}_{\textit{seems}31}$, $C^{11}_{\textit{seems}53}$ etc. $C^{00}_{\textit{seems}31}$ is of type vp^{31}_A, which means that it can be adjoined into initial trees which contain vp^{31}_A as its argument type (e.g. $C^{01}_{\textit{to love}}$). $C^{11}_{\textit{seems}31}$ is of type $\mathsf{s}^{3-3}_A \multimap \mathsf{vp}^{3-3}_A \multimap \mathsf{vp}^{31}_A$. It means it expects two adjunctions at its s and vp nodes respectively and returns back a term of type vp^{31}_A (as in *John claims to appear to seem to love Mary*). Here, s^{3-3}_A and vp^{3-3}_A are types used for modeling *adjunction on adjunctions*.

When an auxiliary tree is adjoined into another auxiliary tree as in (3), we do not allow the former to modify the tupleness of the latter. For instance $\gamma_{\textit{seems}}$ would generate $\mathcal{S}^2_{\textit{seems}}$ that includes $C^{11}_{\textit{seems}3-3}$, $C^{10}_{\textit{seems}3-3}$, $C^{01}_{\textit{seems}3-3}$, $C^{00}_{\textit{seems}3-3}$, $C^{11}_{\textit{seems}5-5}$ etc. $C^{00}_{\textit{seems}3-3}$ has a subscript $(k-k)$ that correspond to adjunctions into adjunction trees. The type of $C^{00}_{\textit{seems}3-3}$ is vp^{3-3}_A, meaning that it can directly adjoin into auxiliary trees which have arguments of type vp^{3-3}_A. $C^{01}_{\textit{seems}3-3}$ is of type $\mathsf{vp}^{3-3}_A \multimap \mathsf{vp}^{3-3}_A$, which means that it itself expects an adjunction and the result can be adjoined into another adjunction tree.

Now it is easy to define $\mathscr{L}_{der}$ from $\Sigma'_{der\theta}$ to $\Sigma_{der\theta}$. It maps every type $X \in \Sigma'_{der\theta}$ to $X \in \Sigma_{der\theta}$ and every X^N_A to X_A; types without numbers are mapped to themselves, i.e. s to s, np to np, etc. Moreover, the different *versions* of some constant, that were introduced in order to extract the yield, are translated using only one constant and *fake* adjunctions. For instance:

$$\mathscr{L}_{der}(C^{11}_{\textit{to love}}) = C_{\textit{to love}}$$
$$\mathscr{L}_{der}(C^{10}_{\textit{to love}}) = \lambda xso.C_{\textit{to love}}\ x\ I_{\mathsf{vp}}\ s\ o$$
$$\mathscr{L}_{der}(C^{00}_{\textit{to love}}) = C_{\textit{to love}}\ I_{\mathsf{s}}\ I_{\mathsf{vp}}$$

6 Encoding a Dependency Grammar

The ACG of (Pogodalla, 2009) mapping TAG derivation trees to logical formulas already encoded some reversal of the predicate-argument structure. Here we map the disambiguated derivation trees to dependency structures. The vocabulary that define these *dependency trees* is Σ_{dep}. It is also designed to allow us to build two lexicons from it to Σ_{string} (to provide a direct yield function) and to Σ_{Log} (to provide a logical semantic representation).

In Σ_{dep} constants are typed as follows: $d^5_{\textit{to love}} : \tau^1_{\mathsf{np}} \multimap \tau^1_{\mathsf{np}} \multimap \tau^5$. Here, τ^1_{np} is the type into which the np type is translated from *disambiguated derivation tree*. The superscript 1 indicates that τ^1_{np} will be translated into 1-tuple into Σ_{string}. Now, it is easy to see that in order to translate $C^{10}_{\textit{to love}} : \mathsf{s}^{31}_a \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{s}$ and $C^{01}_{\textit{to love}} : \mathsf{vp}^{31}_a \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{s}$, we need to

[6] See note 5.

have constants like: $d^{3\mathsf{s}}_{\textit{to love}} : \tau^1_{\mathsf{np}} \multimap \tau^1_{\mathsf{np}} \multimap \tau^3$ and $d^{3\mathsf{v}}_{\textit{to love}} : \tau^1_{\mathsf{np}} \multimap \tau^1_{\mathsf{np}} \multimap \tau^3$.

Moreover, we have constants for adjunction trees, like $d^5_{seems} : \tau^5 \multimap \tau^3$ that will be used in the translation of $C^{01}_{\textit{seems}53}$, and $d^{5-5}_{seems} : \tau^5 \multimap \tau^5$ for $C^{00}_{\textit{seems}5-5}$. Furthermore, additional constants are needed to have things *correctly* typed. For this reason, the constants d^1_3, d^3_5 etc. are introduced. Each d^{2n-1}_{2n+1} has type $\tau^{2n+1} \multimap \tau^{2n-1}$.

Finally, non-CTAs like n_A, n^d_A, vp_A and s_A are translated as $\tau^2_{\mathsf{n}_A}$, $\tau^2_{\mathsf{n}^d_A}$, τ^2_{vp}, and τ^2_{s} respectively. A superscript 2 indicates that they are modeled as 2-tuples in $\Sigma_{\textit{string}}$.

Now we can define $\mathscr{L}_{dep}$, the lexicon from $\Sigma'_{der\theta}$ to Σ_{dep} translating disambiguated derivation trees into dependency trees:

$$
\begin{array}{ll}
\mathscr{L}_{dep}(\mathsf{s}) & = \tau^1 \\
\mathscr{L}_{dep}(\mathsf{np}) & = \tau^1_{\mathsf{np}} \\
\mathscr{L}_{dep}(X^{(2n+1)(2n-1)}_A) & = \tau^{2n+1} \multimap \tau^{2n-1} \\
\mathscr{L}_{dep}(X^{(2n+1)(2n+1)}_A) & = \tau^{2n+1} \multimap \tau^{2n+1} \\
 & \text{for } X \in \mathsf{s}^{31}_A, \mathsf{vp}^{53}_A \ldots
\end{array}
$$

$$
\begin{array}{ll}
\mathscr{L}_{dep}(C^{11}_{\textit{to love}}) & = \lambda S\ V\ s\ o.S(V(d^5_{\textit{to love}}\ s\ o)) \\
\mathscr{L}_{dep}(C^{10}_{\textit{to love}}) & = \lambda S\ s\ o.S(d^3_{\textit{to love}}\ s\ o) \\
\mathscr{L}_{dep}(C^{01}_{\textit{to love}}) & = \lambda S\ V\ s\ o.V(d^3_{\textit{to love}}\ s\ o) \\
\mathscr{L}_{dep}(C^{00}_{\textit{seems}53}) & = \lambda x.d^5_{\textit{seems}}x \\
\mathscr{L}_{dep}(C^{01}_{\textit{seems}53}) & = \lambda V\ x.d_{53}(V(d^5_{\textit{seems}}\ x)) \\
\mathscr{L}_{dep}(C^{11}_{\textit{seems}53}) & = \lambda S\ V\ x.d_{53}(S(V(d^5_{\textit{seems}}\ x))) \\
\mathscr{L}_{dep}(C^{01}_{\textit{seems}5-5}) & = \lambda V\ x.V(d^{5-5}_{\textit{seems}}\ x) \\
\mathscr{L}_{dep}(C^{00}_{\textit{seems}5-5}) & = \lambda x.d^{5-5}_{\textit{seems}}\ x
\end{array}
$$

Furthermore, we describe Σ_{Log} [7] and define two lexicons: $\mathscr{L}_{\textit{dep. yield}} : \Sigma_{dep} \longrightarrow \Sigma_{\textit{string}}$ and $\mathscr{L}_{\textit{dep. log}} : \Sigma_{dep} \longrightarrow \Sigma_{Log}$. Table 2 provides examples of these two translations.

$\mathscr{L}_{\textit{dep. yield}}$: It translates any atomic type τ^n or τ^n_X with $X \in \{\mathsf{n}_A, \mathsf{n}^d_A \ldots\}$ as a n-tuple of string $\underbrace{(\sigma \multimap \sigma \cdots \multimap \sigma)}_{n+1\text{-times}} \multimap \sigma$.[8]

Σ_{Log}: Its atomic types are e and t and we have the constants: **john**, **mary**, **bill** of type e, the constant **love** of type $e \multimap e \multimap t$, the constant **claim** of type $e \multimap t \multimap t$ and the constant **seem** of type $t \multimap t$.

$\mathscr{L}_{\textit{dep. log}}$: Each $\tau^{2(n+1)}$ is mapped to t, τ^1_{np} is mapped to $(e \multimap t) \multimap t$, $\tau^2_{\mathsf{n}_A{}^d}$ is mapped to $(e \to t) \multimap (e \multimap t) \multimap t$. The types of non-complement-taking verbal or sentential adjunctions τ^2_{vp} and τ^2_{s} are translated as $t \multimap t$.

Let us show for the sentence (1) how the ACGs defined above work with the data provided in Table 2. Its representation in $\Sigma'_{der\theta}$ is: $T_0 = C^{11}_{\textit{to love}}\ (C_{\textit{claims}31}\ C_{\textit{Bill}})\ C_{\textit{seems}53}\ C_{\textit{Mary}}\ C_{\textit{John}}$. Then

$\mathscr{L}_{der}(T_0) = t_0$

and

$\mathscr{L}_{dep}(T_0) = d^1_{\textit{claims}} d_{\textit{Bill}} (d^{53}_{\textit{seems}} (d^5_{\textit{to love}} d_{\textit{Mary}} d_{\textit{John}})) = t'_0$

and finally

$$
\begin{aligned}
&\mathscr{L}_{\textit{dep. yield}}(t'_0) \\
&\quad = \mathscr{L}_{\textit{yield}}(\mathscr{L}_{\textit{d-ed trees}}(t_0)) \\
&\quad = \lambda f.f(\textit{John} + (\textit{Bill} + (\textit{claims} \\
&\quad + ((\textit{Mary} + ((\textit{seems} + \textit{to love}) + \epsilon)) + \epsilon))))
\end{aligned}
$$

and

$\mathscr{L}_{\textit{dep. log}}(t'_0) = \textbf{claim bill}\ (\textbf{seem}\ (\textbf{love john mary})$

7 Conclusion

In this paper, we have given an ACG perspective on the transformation of the derivation trees of TAG to the dependency trees proposed in (Kallmeyer and Kuhlmann, 2012). Figure 4 illustrates the architecture we propose. This transformation is a two-step process using first a macro-tree transduction then an interpretation of dependency trees as (tuples of) strings. It was known from (Yoshinaka, 2006) how to encode a macro-tree transducer into a $\mathscr{G}_{\text{dep}} \circ \mathscr{G}^{-1}_{\text{der}}$ ACG composition. Dealing with typed trees to represent derivation trees allows us to provide a meaningful (wrt. the TAG formalism) abstract vocabulary $\Sigma'_{der\theta}$ encoding this macro-tree transducer. The encoding of the second step then made explicit the lexical blow up for the interpretation of the functional symbols of the dependency trees in (Kallmeyer and Kuhlmann, 2012)'s construct. It also provides a push out (in the categorical sense) of the two morphisms from the disambiguated derivation trees to the derived trees and to the dependency trees. The diagram is completed with the yield function from the derived trees and from the dependency trees to the string vocabulary.

Finally, under the assumption of (Kallmeyer and Kuhlmann, 2012) of plausible dependency structures, we get two possible grammatical approaches to the surface-semantics relation that are related but independent: it can be equivalently modeled using either a phrase structure or a dependency model.

[7] We refer the reader to (Pogodalla, 2009) for the details.

[8] We encode a n-tuple $\langle M_1, \ldots, M_n \rangle$ as $\lambda f.f\ M_1\ M_2\ \ldots\ M_n$ where each M_i has type σ.

Abstract constants of Σ_{dep}	Their images by $\mathscr{L}_{dep.\ yield}$	Their images by $\mathscr{L}_{dep.\ log}$
$d_{\textit{John}} : \tau^1_{\mathsf{np}}$	*John*	$\lambda P.P$ **john**
$d^5_{\textit{to love}} : \tau^1_{\mathsf{np}} \multimap \tau^1_{\mathsf{np}} \multimap \tau^5$	$\lambda S\ O.\lambda f.f\ O\ S$ *to love* $\epsilon\ \epsilon$	$\lambda O\ S.S(\lambda x.O(\lambda y.(\textbf{love}\ x\ y)))$
$d^{3s}_{\textit{to love}} : \tau^1_{\mathsf{np}} \multimap \tau^1_{\mathsf{np}} \multimap \tau^3$	$\lambda S\ O.\lambda f.f\ O\ (S + \textit{to love})\ \epsilon$	$\lambda O\ S.S(\lambda x.O(\lambda y.(\textbf{love}\ x\ y)))$
$d^{3v}_{\textit{to love}} : \tau^1_{\mathsf{np}} \multimap \tau^1_{\mathsf{np}} \multimap \tau^3$	$\lambda S\ O.\lambda f.f\ (O + S)$ *to love* ϵ	$\lambda O\ S.S(\lambda x.O(\lambda y.(\textbf{love}\ x\ y)))$
$d^1_{\textit{claims}} : \tau^1_{\mathsf{np}} \multimap \tau^1_{\mathsf{np}} \multimap \tau^1$	$\lambda S\ c.\lambda f.c(\lambda x_1\ x_2\ x_3.f(x_1 + S + \textit{claims} + x_2 + x_3))$	$\lambda S\ c.S(\lambda x.\textbf{claim}\ x\ c)$
$d^3_{\textit{claims}} : \tau^1_{\mathsf{np}} \multimap \tau^1_{\mathsf{np}} \multimap \tau^3$	$\lambda S\ c.\lambda f.c(\lambda x_1\ x_2\ x_3.f(x_1 + S)\ (\textit{claims} + x_2 + x_3))$	$\lambda S\ c.S(\lambda x.\textbf{claim}\ x\ c)$
$d^{5-5}_{\textit{seems}} : \tau^5 \multimap \tau^5$	$\lambda c.\lambda f.c(\lambda x_1 \ldots x_5.f\ x_1\ (x_2 + \textit{seems} + x_3 + x_4)x_5)$	$\lambda c.\textbf{seem}\ c$
$d^5_{\textit{seems}} : \tau^5 \multimap \tau^3$	$\lambda c.\lambda f.c(\lambda x_1 \ldots x_5.f\ x_1\ x_2\ (\textit{seems} + x_3)\ x_4\ x_5)$	$\lambda c.\textbf{seem}\ c$
$d^{2n-1}_{2n+1} : \tau^{2n+1} \multimap \tau^{2n-1}$	$\lambda g f.g(\lambda x_1 \ldots x_n.f\ x_1\ \ldots (x_n + x_{n+1} + x_{n+2}) \ldots x_{2n+1})$	$\lambda x.x : t \multimap t$

Table 2: Lexicons for yield and semantics from the dependency vocabulary

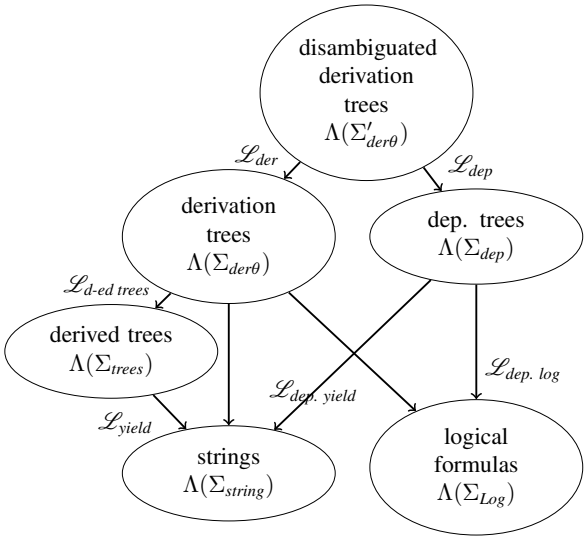


Figure 4: General architecture

References

Joan Chen-Main and Aravind K. Joshi. To appear. A dependency perspective on the adequacy of tree local multi-component tree adjoining grammar. *Journal of Logic and Computation*.

Philippe de Groote and Sylvain Pogodalla. 2004. On the expressive power of Abstract Categorial Grammars: Representing context-free formalisms. *Journal of Logic, Language and Information*, 13(4):421–438.

Philippe de Groote. 2001. Towards abstract categorial grammars. In *Proceedints of ACL*, pages 148–155.

Philippe de Groote. 2002. Tree-adjoining grammars as abstract categorial grammars. In *Proceedings of TAG+6*, pages 145–150. Università di Venezia.

Joost Engelfriet and Heiko Vogler. 1985. Macro tree transducers. *J. Comput. Syst. Sci.*, 31(1):71–146.

Aravin K. Joshi and Yves Schabes. 1997. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of formal languages*, volume 3, chapter 2. Springer.

Aravind K. Joshi, Leon S. Levy, and Masako Takahashi. 1975. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1):136–163.

Aravind K. Joshi, Laura Kallmeyer, and Maribel Romero. 2003. Flexible composition in ltag: Quantifier scope and inverse linking. In Harry Bunt, Ielka van der Sluis, and Roser Morante, editors, *Proceedings of IWCS-5*.

Laura Kallmeyer and Marco Kuhlmann. 2012. A formal model for plausible dependencies in lexicalized tree adjoining grammar. In *Proceedings of TAG+11*, pages 108–116.

Laura Kallmeyer and Maribel Romero. 2004. Ltag semantics with semantic unification. In *Proceedings of TAG+7*, pages 155–162.

Laura Kallmeyer and Maribel Romero. 2007. Scope and situation binding for ltag. *Research on Language and Computation*, 6(1):3–52.

Laura Kallmeyer. 2002. Using an enriched tag derivation structure as basis for semantics. In *Proceedings of TAG+6*.

Sylvain Pogodalla. 2004. Computing Semantic Representation: Towards ACG Abstract Terms as Derivation Trees. In *Proceedints of TAG+7*, pages 64–71, Vancouver, BC, Canada. `http://hal.inria.fr/inria-00107768`.

Sylvain Pogodalla. 2009. Advances in Abstract Categorial Grammars: Language Theory and Linguistic Modeling. ESSLLI 2009 Lecture Notes, Part II. `http://hal.inria.fr/hal-00749297`.

Owen Rambow, K. Vijay-Shanker, and David Weir. 2001. D-Substitution Grammars. *Computational Linguistics*.

Yves Schabes and Stuart M. Shieber. 1994. An alternative conception of tree-adjoining derivation. *Computational Linguistics*, 20(1):91–124.

Ryo Yoshinaka. 2006. *Extensions and Restrictions of Abstract Categorial Grammars*. Phd, University of Tokyo.