



HAL
open science

The Unit Graphs Mathematical Framework

Maxime Lefrançois, Fabien Gandon

► **To cite this version:**

Maxime Lefrançois, Fabien Gandon. The Unit Graphs Mathematical Framework. [Research Report] RR-8212, 2013, pp.60. hal-00780805v2

HAL Id: hal-00780805

<https://inria.hal.science/hal-00780805v2>

Submitted on 27 Mar 2013 (v2), last revised 1 Jul 2013 (v3)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



The Unit Graphs Mathematical Framework

Maxime Lefrançois, Fabien Gandon

**RESEARCH
REPORT**

N° 8212

March 2013

Project-Team Wimmics



The Unit Graphs Mathematical Framework

Maxime Lefrançois, Fabien Gandon

Project-Team Wimmics

Research Report n° 8212 — March 2013 — 57 pages

Abstract: In this paper we are interested in the choice of a knowledge representation formalism that enables the representation, manipulation, query, and reasoning over linguistic knowledge of the Explanatory and Combinatorial Dictionary (ECD) of the Meaning-Text Theory. We show that neither the semantic web formalisms nor the Conceptual Graphs (CGs) Formalism suit our needs, and justify the introduction of a new formalism denoted Unit Graphs (UGs). We thus introduce the fundamental concepts of the UGs mathematical framework: the Unit Types hierarchy and the Unit Graphs. We then show how one may use them to represent specific aspects of the ECD.

Key-words: Linguistic Knowledge Representation, Meaning-Text Theory, Explanatory and Combinatorial Dictionary, Conceptual Graphs, Unit Graphs

**RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE**

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Le formalisme mathématique des graphes d'unités

Résumé : Dans cet article nous nous intéressons au choix d'un formalisme de représentation des connaissances qui nous permette de représenter, manipuler, interroger et raisonner sur des connaissances linguistiques du Dictionnaire Explicatif et Combinatoire (DEC) de la Théorie Sens-Texte. Nous montrons que ni les formalismes du web sémantique ni le formalisme des Graphes conceptuels (GC) n'est adapté pour cela, et justifions l'introduction d'un nouveau formalisme dit des Graphes d'Unités. Nous introduisons donc les concepts fondamentaux du formalisme des Graphes d'Unités: la hiérarchie des Types d'Unités et les Graphes d'Unités. Nous montrons ensuite comment on peut les utiliser pour représenter certains aspects du DEC.

Mots-clés : Représentation de Connaissances Linguistiques, Théorie Sens-Texte, Dictionnaire Explicatif et Combinatoire, Graphes Conceptuels, Graphes d'Unités

Contents

I	Introduction	6
1	Motivations to Introduce a New Knowledge Representation Formalism	8
1.1	Semantic Web Formalisms	8
1.2	The Conceptual Graphs (CGs) Formalism	9
1.3	The new Unit Graphs Formalism	10
II	Unit Types	11
2	Primitive Unit Types (PUTs)	11
2.1	Definition of PUTs	11
2.2	PUT Participant Slots (PSlots)	12
2.3	PUT Hidden Slots (HSlots)	15
2.4	PUT Actant Slots (ASlots)	17
2.5	PUT Signatures	18
3	Conjunctive Unit Types (CUTs)	20
3.1	Definition of CUTs	20
3.2	CUT Slots and Signatures	20
3.3	Pre-order over CUTs	22
3.4	CUTs Hierarchy	25
4	Characterizing CUTs	26
4.1	Necessary Conditions to Compare Two CUTs	26
4.2	Properties of CUT Slots and Signatures	27
4.3	CUT Equivalence Class Sets	31
4.4	Maximal CUTs	32
4.5	Concise CUTs	35
III	Unit Graphs	36
5	Unit Graphs (UGs)	36
5.1	Circumstantial Dependency Symbols Hierarchy	36
5.2	Definition of UGs	37
5.3	Graphical Representation	40
5.4	Explicit Support Compliance	41
5.5	Semantics of UGs: Reasoning using First-Order Implications	42
5.5.1	Saturation	42
5.5.2	Closure	44
6	Mappings of UGs	45
6.1	Weak Homomorphism	45
6.2	Weak Hom-Equivalence	46
6.3	Weak Isomorphism	47
6.4	Strong Homomorphism: Inclusion	47

6.5	Strong Isomorphism: Equality	49
7	Elementary Operations	50
7.1	Explicitation Operations	50
7.2	Specialization Operations	51
8	Rules	52
8.1	λ -UG	52
8.2	Definition and Logical Semantics of Rules	52
8.3	PUTs Prints and Definitions	52
8.4	UG Support with PUTs Definitions and GDefs	54
IV	Conclusion	55

Glossary

ASlot Actant Slot 8, 12–15, 23, 29, 31

CG Conceptual Graph 5, 6, 8

CUT Conjunctive Unit Type 7, 14–31

DepG Dependency Grammar 5

DSymbol Dependency Symbol 7, 31, 32

ECD Explanatory Combinatorial Dictionary 5, 32

FLN French Lexical Network 5

HSlot Hidden Slot 8, 11, 12, 14, 15, 23, 31

MTT Meaning-Text Theory 5, 6, 31, 32

NL-expressible expressible in a natural language utterance 10–12, 15, 31

PSlot Participant Slot 8–12, 14–17, 19, 21, 22, 24, 30, 31

PSymbol Participation Symbol 9–13, 31

PUT Primitive Unit Type 7–16, 19–21, 24–28, 31, 32

UG Unit Graph 5–9, 19, 29–32

Part I

Introduction

In this research report we are interested in the choice of a knowledge representation formalism that enables to represent, manipulate, query, and reason with linguistic knowledge of the the **Explanatory Combinatorial Dictionary (ECD)**, which is the lexicon at the core of the **Meaning-Text Theory (MTT)** (c.f. for instance [Mel'čuk and Arbatchewsky-Jumarie, 1999](#); [Mel'čuk, 2006](#)).

We envision two validation scenarios of such a formalization:

- In a **ECD** lexicographic edition oriented project, we could enable the semi-automation of some of the lexicographers tasks. For instance we could check that a set of constraints is satisfied, or we could suggest preliminary drafts of article (e.g., lexical function key-value pairs, lexicographic definition sketches, government pattern).
- We could propose a syntax based on knowledge engineering standards. Thus the linguistic knowledge written with that syntax could be published to the web of linked data¹, as is WordNet today. This would support their use as a highly structured lexical resource by consumers of the linked data cloud.

Most past or current projects that consisted in implementing the **ECD** did it in a lexicographic perspective. For instance projects NADIA-DEC [Sérasset \(1997\)](#), Dicoùebe [Polguère \(2000\)](#), DicoInfo and DicoEnviro [L'Homme \(2008\)](#), DiCE [Alonso Ramos \(2003\)](#) for Spanish.

The RELIEF project [Lux-Pogodalla and Polguère \(2011\)](#) aims at representing a lexical system graph named RLF [Polguère \(2009\)](#) where lexical units are interlinked by paradigmatic and syntagmatic links of lexical functions (e.g., [Mel'čuk, 1996](#)). In the RELIEF project, the description of Lexical Functions is based on a formalization proposed by [Kahane and Polguère \(2001\)](#). Moreover, lexicographic definitions start to be partially formalized using the markup type that has been developed in the Definiens project [Barque and Polguère \(2008\)](#); [Barque et al. \(2010\)](#) which aims at formalizing lexicographic definitions with genus and specific differences for the TLFi².

In addition to these formalization works, our objective here is to propose a formalization from a knowledge engineering perspective, and compatible with standard formalisms. The term *formalization* here means not only *make non-ambiguous*, but also *make operational*, i.e., *make adapted to logic and rational operations* (e.g., knowledge manipulation, query, reasoning). We thus adopt a knowledge engineering approach applied to the domain of the **MTT**, and our research question here is *What knowledge representation formalism would be adapted to represent knowledge of the **ECD**?*

We are interested in two existing knowledge representation formalisms:

- the semantic web formalisms (RDF/S, OWL, SPARQL), because the linked data is built on them;
- the **Conceptual Graphs (CGs)** formalism [Sowa \(1984\)](#); [Chein and Mugnier \(2008\)](#), as we will lead logic reasoning on graphs.

Our research question may thus be decomposed in two sub-questions that we adress in this research report:

¹The web of data is a W3C initiative, highly active today, <http://linkeddata.org>

²Trésor de la Langue Française informatisé, <http://atilf.atilf.fr>

- Is one of these knowledge representation formalism adapted to represent knowledge of the **ECD** ?
- If one need to, how shall we modify them so that they are adapted ?

The rest of this research report is organized as follows.

We will first see that neither the semantic web formalisms nor the **CGs** are adapted to represent the knowledge of the **ECD**, and will justify the following choice: *We modify the **CGs** formalism basis, and keep in mind the idea to use **RDF** as a syntax for sharing knowledge and publishing over the web of data.*

As we will represent linguistic units of different nature (e.g., sense units, lexical units, grammatical units, words), we choose to use term *unit* in a generic manner, and name the result of this adaptation the **Unit Graph (UG)** mathematical framework.

Figure 1 below illustrates a **UGs** that corresponds to the semantic representations of the utterances *Peter tries to push the cat.*

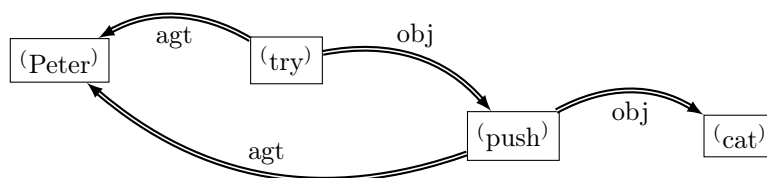


Figure 1: Illustration of the duality concept/relation of sense units in the **MTT**, semantic representation of utterance *Peter tries to push the cat.*

We will first introduce and study **Primitive Unit Types (PUTs)** (§2), then we will extend their definitions and results to **Conjunctive Unit Types (CUTs)**, and will introduce a hierarchy of **CUTs** (§3) and characterize it (§4). Section 5 will be devoted to the **UGs** *per se*.

1 Motivations to Introduce a New Knowledge Representation Formalism

Knowledge representation formalisms use the notion of types. Objects of the represented domain are named instances (or objects, or individuals), and are typed (or classified). They are interlinked through relations that are themselves typed. In this section we answer the following question: *What makes the semantic web formalisms and the CGs formalisms not directly adapted to the representation of knowledge of the ECD ?*

1.1 Semantic Web Formalisms

There is a world wide enthusiasm for the semantic web formalisms, and the RDF syntax is the standard for structured data exchange over the web of linked data. The expressivity of RDF would be sufficient to represent the knowledge of the ECD. Yet, the semantics of RDF, in the logic sense, is limited to that of oriented labelled graphs, and we wish also to enable the manipulation and reasoning with linguistic knowledge of the ECD. We thus need to introduce more semantics thanks to RDFS or OWL, while keeping the expressivity as low as possible to keep good complexity properties. OWL introduces semantics with axioms³ and classes and relation constructors⁴.

The ULiS project Lefrançois and Gandon (2011a) did envision an architecture for a multilingual knowledge base compatible with the MTT and based on OWL. In the ULiS project, axioms and class constructors are used in order to make each lexical unit support the projection of its lexicographic definition over itself. We identified three major problems with the usage of OWL for that.

- For each lexical unit definition, one need to introduce as much new semantic relations as there exists nodes in the definition graph of the lexical unit. This implies an overload of useless relations.
- These relations must be combined using the sub relation chains axioms `SubObjectPropertyOf(ObjectPropertyChain(OPE1 ... OPEn) OPE)`, in order to project little by little the lexical unit definition graph on the lexical unit itself. The OWL2 DL roughly corresponds to the *SR_QIQ* description logics fragment. In this fragment, the hierarchy of roles (= primary relations) must be *regular*⁵ (c.f., Rudolph, 2011, §2.1) in order to guarantee decidability of basic reasoning problems. We will show that this regularity is not ensured in the small example ontology given by Lefrançois and Gandon (2011a). We defined the lexical unit DIE^L , and wanted to define the lexical unit KILL^L roughly as follows: *Causing ones death*. We then wanted to model that the time of an event *kill* is the same as the time of the event *die* of the person killed. For that we defined a new role `hasKillTime`, and used two *relation* axioms: `SubObjectPropertyOf(ObjectPropertyChain(hasEvent hasTime) hasKillTime)`, and `SubObjectPropertyOf(hasKillTime hasTime)`. The first axiom implies that `hasKillTime` is a non-simple role, and this combined with the second axiom implies that `hasTime` is also a non-simple role. One needs to have a strict partial order $<$ over the set of non-simple roles. First axiom implies that `hasTime` $<$ `hasKillTime`, and second axiom implies that `hasKillTime` $<$ `hasTime`. Thus as $<$ is required to be strict, the role hierarchy is not regular and this example ontology slips into OWL Full and undecidability. The intuitive meaning of this is that `hasKillTime` is defined using `hasTime` in

³e.g., Sub-class `SubClassOf(CE1 CE2);` Functional Relation: `FunctionalObjectProperty(OPE)`

⁴e.g., Exact cardinality `ObjectExactCardinality(n OPE);` Inverse relation `ObjectInverseOf(OPE)`

⁵c.f. for instance, http://www.w3.org/TR/owl2-syntax#The_Restrictions_on_the_Axiom_Closure

the first axiom, and vice versa in the second axiom. This restriction is thus too important to represent definitions of the **ECD**.

- Finally, the semantics of the Sub Property Chains axiom makes that inference is anyways possible only in one direction (sub property and not equivalence). This means that when there is the definition of the lexical unit in the graph one may infer that there is the lexical unit, but not the other way around.

One alternative to represent lexicographic definitions of lexical units would be to use two reciprocal CONSTRUCT SPARQL rules. We then face the problem of rule languages and their compatibility with OWL (c.f., [Krisnadhi et al., 2011](#)), that led to no consensus nor standard today.

These different problems led us to consider another formalism to represent knowledge of the **ECD**. We nevertheless want to be able to export these knowledge in RDF to exchange them over the web of linked data.

1.2 The Conceptual Graphs (CGs) Formalism

The **CGs** formalism [Sowa \(1984\)](#); [Chein and Mugnier \(2008\)](#) has many similarities with the **MTT**. In their basic version, **CGs** represent typed instances interconnected by typed n -ary relations. Actually, the main goal of Sowa was natural language processing, and he originally inspired from the same works than **MTT** founders: [Tesnière \(1959\)](#). Two of the most important similarities are the following:

- [Sowa \(1989\)](#) early suggested to introduce type definition of concepts and relations that do look similar to lexical units definitions in the **ECD**. Later on [Leclère \(1998\)](#) also worked on the possibility to reason with type and concept definitions.
- The **MTT** massively uses rules, for instance to declare correspondences between utterances at different representation levels. Rules and their semantics, in the logical sense, have been thoroughly studied in the **CGs** literature.

One more asset of **CGs** is the fact that there exists transformations between **CGs** and RDF/S (c.f., [Corby et al., 2000](#); [Baget et al., 2010](#)). One could use these transformations to rewrite **CGs** in RDF for publication over the web of linked data. Moreover, one could adapt the architecture described in the ULiS ([Lefrançois and Gandon, 2011b](#)) project to **CGs**, that envisioned a **MTT**-compliant pivot-based multilingual knowledge base architecture, using the semantic web formalisms, hence:

Through the [Universal Linguistic System (ULiS)], a user could interact with an interlingual knowledge base (IKB) in controlled natural language. Linguistic resources themselves [would be] part of a specific IKB: The Universal Lexical Knowledge base (ULK), so that actors may enhance their controlled natural language, through requests in controlled natural language.

Yet it is also not natural to represent the knowledge of the **ECD** using the **CGs**. Here are two reasons for that:

- A sense-unit may be represented as a concept type as it is instantiated in utterance semantic representations. On the other hand, if the associated lexical unit is predicative and has **Semantic Actant Slots (SemASlots)**, then the semanteme may dually be represented as a

n -ary relation, so that its instances link other sense units. The CGs don't offer a natural representation of this duality. In fact, in CGs, one must alternate concepts and relations, and a semantic representation of an utterance such as the one on figure 1 can't be directly represented by a CG.

- **SemASlots** of a lexical unit may differ from those of the lexical unit from which its sense derives⁶ (c.f., Mel'čuk, 2004a,b). Yet in the CGs, the inheritance mechanism of relation types, that models the fact that *a relation type is more specific than another*, is constrained so that two relations with different arities must be incomparable. One thus cannot use this natural inheritance mechanism to model the sense units specialization.

Unfortunately, one cannot formalize the MTT with CGs in a natural manner, here are two reasons for that:

- A unit type in the MTT is formalizable *à priori* as a concept type as it is instantiated in linguistic situations. On the other hand, if this unit type is predicative (i.e., has actant slots) it may in a dual manner be formalized as a n -ary relation, such that one of its instances is linked to its actants. CGs thus don't allow to naturally represent this duality and hence the notion of linguistic predicate.
- In the theory of actants (c.f., Mel'čuk, 2004a,b) developed with the MTT, it is possible that the set of actant slots of a predicative unit type differ from that of another unit type from which it is derived.⁷ Yet, the specification of concept types and relation types in the CGs formalism (i.e., the *support*) imposes that two relations with different arity are incomparable. One thus cannot use the natural inheritance mechanism in the context of a predicative unit type.

1.3 The new Unit Graphs Formalism

To sum up, neither the semantic web formalisms nor the CGs formalism enable the natural representation of knowledge of the ECD. As the CG formalism is the closest from the MTT, we decide to use it as a starting point for designing a new formalism adapted to the representation of knowledge of the ECD.

As we are to represent linguistic units of different nature (e.g., sense units, lexical units, grammatical units, words), we choose to use the term *unit* in a generic manner and name the result of this adaptation *Unit Graphs (UGs) mathematical framework*.

We will also provide the UGs with a RDF syntax so as to be able to exchange linguistic knowledge and publish them over the web of linked data.

In the rest of this paper we will answer the following research question: *How shall we revisit the CGs formalism so as to make it adapted to represent knowledge of the ECD ?* This question may be decomposed in two sub-questions:

- What mathematical structure for a hierarchy of unit types that may have actant slots (part II)?
- What is a UG, and how to use them to represent advanced concepts of the TST (part III)?

⁶For instance, sense unit (rain) is more specific than (fall) but the meaning of *what falls* and *where it falls from* is fixed to (water drops) and (sky/cloud) Mel'čuk (2004a).

⁷For instance, the predicative meaning unit type (to rain) may be defined from (to fall) but its actant slots (what falls) and (from where it falls) are freezed to (water drops) and (sky/cloud).

Part II

Unit Types

In the **Unit Graphs (UGs)** mathematical formalism, the objects of the represented domain are named *units*, and are typed. We do establish a distinction between:

- Unit types (e.g., sense unit type, lexical unit type), described in the **ECD**;
- Units (e.g., sense unit, lexical unit), represented in the **UGs**.

This distinction corresponds to that of classes vs. instances in RDFS and OWL, and concept type vs. individual in **CGs**.

Unlike the **CGs** formalism, every type may be considered bot as a concept (i.e., it has instances), and as a n -ary relation (i.e., it links instances). As described in this research report, unit types enable units categorization, and also the specification of how units must be linked in a **UG**.

2 Primitive Unit Types (PUTs)

This section first introduces atomic types which are called *Primitive Unit Types (PUTs)*, and then introduces notions used to characterize gslplut as n -ary relation: **Participant Slots (PSlots)** (§2.2), **Hidden Slots (HSlots)** (§2.3), **Actant Slots (ASlots)** (§2.4), and signatures (§2.5).

2.1 Definition of PUTs

We introduce a finite set of *Primitive Unit Types (PUTs)* denoted \mathbf{T} and defined as follows.

Definition 2.1 (PUT Set). A *PUT Set* is a disjoint union denoted $\mathbf{T} \stackrel{\text{def}}{=} T_{\text{declared}} \cup \{\top\} \cup \{\perp\}$, where T_{declared} is a finite set of *declared PUTs*, \top is the *prime universal PUT*, and \perp is the *prime absurd PUT*.

We introduce an inheritance mechanism for **PUTs** which models a specialization relation. This takes the form of a pre-order \lesssim over the set \mathbf{T} , e.g., $(\text{to rain}) \lesssim (\text{to fall})$ means that (to rain) is more specific than (to fall) (more specific semantically here). The pre-order over the set of **PUTs** is induced by a set $\mathbf{C}_{\mathbf{T}}$ of comparisons of **PUTs**.

Definition 2.2 (Pre-order over \mathbf{T}). The **PUT** set is pre-ordered by a relation \lesssim , which is induced by a set $\mathbf{C}_{\mathbf{T}} \stackrel{\text{def}}{=} C_{\text{asserted}} \cup C_{\top} \cup C_{\perp} \subseteq \mathbf{T}^2$ where $C_{\text{asserted}} \subseteq \mathbf{T}^2$ is the set of *asserted comparisons*, $C_{\top} \stackrel{\text{def}}{=} \{(\top, t)\}_{t \in \mathbf{T}}$, and $C_{\perp} \stackrel{\text{def}}{=} \{(t, \perp)\}_{t \in \mathbf{T}}$. $(\mathbf{T}, \mathbf{C}_{\mathbf{T}})$ is a directed graph on \mathbf{T} . Let $\mathbf{C}_{\mathbf{T}}^*$ be the reflexo-transitive closure of $\mathbf{C}_{\mathbf{T}}$, i.e., $(t, t') \in \mathbf{C}_{\mathbf{T}}^*$ iff t' is a descendant of t in $\mathbf{C}_{\mathbf{T}}$. The pre-order relation \lesssim is equal to $\mathbf{C}_{\mathbf{T}}^*$, i.e., $\forall t, t' \in \mathbf{T}, t' \lesssim t$ iff $(t, t') \in \mathbf{C}_{\mathbf{T}}^*$.

By construction, the set of **PUTs** is bounded by \top , a maximal element of \mathbf{T} , and \perp , a minimal element of \mathbf{T} :

Proposition 2.1. \top and \perp are respectively a maximal and a minimal element of \mathbf{T} .

Proof. Let us prove that 1) \top is a maximal element of \mathbf{T} , and 2) \perp is a minimal element of \mathbf{T} .

1) Let $t \in \mathbf{T}$. From definition 2.2, $(\top, t) \in C_{\top}$.

$C_{\top} \subseteq \mathbf{C}_{\mathbf{T}} \subseteq \mathbf{C}_{\mathbf{T}}^*$, so $(\top, t) \in \mathbf{C}_{\mathbf{T}}^*$. So $t \lesssim \top$ (def. 2.2) and \top is a maximal element of \mathbf{T} .

2) Conversely, let $t \in \mathbf{T}$. From definition 2.2, $(t, \perp) \in C_{\perp}$.

$C_{\perp} \subseteq \mathbf{C}_{\mathbf{T}} \subseteq \mathbf{C}_{\mathbf{T}}^*$, so $(t, \perp) \in \mathbf{C}_{\mathbf{T}}^*$, so $\perp \lesssim t$ (def. 2.2) and \perp is a minimal element of \mathbf{T} . \square

Let \simeq be the natural *equivalence relation* over **PUTs** defined by $t \simeq t' \Leftrightarrow t \lesssim t'$ and $t' \lesssim t$. The set of equivalence classes defines a partition of \mathbf{T} . Let $t \in \mathbf{T}$, we denote $[t]$ the equivalence class to which t belongs, i.e., $[t] \stackrel{\text{def}}{=} \{t' \in \mathbf{T} \mid t' \simeq t\}$.

Definition 2.3 (Equivalence **PUTs** class set). The *equivalence **PUTs** class set* \mathbf{T}^\sim is the quotient set of \mathbf{T} by \simeq , i.e., $\mathbf{T}^\sim \stackrel{\text{def}}{=} \mathbf{T}/\simeq = \{[t] \mid t \in \mathbf{T}\}$. We denote t^\sim a generic equivalence **PUTs** class set. We define a partial order \gtrsim over \mathbf{T}/\simeq with $t_1^\sim \gtrsim t_2^\sim$ if and only if $\exists t_1 \in t_1^\sim, t_2 \in t_2^\sim; t_1 \lesssim t_2$.

For instance, figures 2, 3, and 4 visualize equivalence classes using circles. In a same circle, all the **PUTs** are equivalent. In the illustrated examples, **PUTs** (drizzle) and (Scotch mist) are equivalent.

\perp is the prime absurd **PUT** and should by definition have no unit that has it as a type. Thus, any **PUT** that is more specific than \perp should also be absurd. As \perp is also a minimal element of \mathbf{T} , any **PUT** which is more specific than \perp is actually equivalent to \perp . The equivalence class to which \perp belongs is thus called the set of absurd **PUTs** and denoted \perp^\sim . Any $t \in \perp^\sim$ is said to be absurd and no unit should have this **PUT** as a type.

Definition 2.4 (Absurd **PUT** set). The set of absurd **PUTs** is denoted \perp^\sim and is the set: $\perp^\sim \stackrel{\text{def}}{=} [\perp]$.

PUTs may both be considered as a concept (i.e., they have instances), and as a n -ary relation (i.e., they link instances). Next sections are devoted to **PUTs** seen as relations.

2.2 PUT Participant Slots (PSlots)

Unlike the **CGs** formalism, every type may be considered both as a concept (i.e., it has instances), and as a n -ary relation (i.e., it links instances). We will now detail our approach to define unit types and represent how they may be interconnected.

Mel'čuk (2004a) distinguishes three kind of actant slots for a lexical unit. The **SemASlots**, the **Deep Syntactic Actant Slots (DSynASlots)**, and the **Surface Syntactic Actant Slots (SSynASlots)**. Yet the nodes of semantic representations are sense units, unlike the nodes of deep syntactic representations, surface syntactic representations, and deep morphologic representations, which are lexical units. As we adopt a knowledge engineering approach, whose formalisms offer primitives that enable to represent and reason over sense units as a whole, we propose to distinguish the lexical unit and its sense. We thus propose the following model:

- In a general manner, every unit type has **Actant Slots (ASlots)**.
- **ASlots** of a sense unit type corresponds to the **SemASlots** of its associated lexical unit type.
- **ASlots** of a lexical unit type corresponds to its **DSynASlots** and **SSynASlots**.

A problem we face with semantic representations is the following. Predicate-argument relations that link a sense-unit to another are numbered from 1 to n , and correspond to the **SemASlots** of its corresponding lexical unit (this numbering system is also used for the **CGs** n -ary relations). Yet, this numbering cannot be coherent in the hierarchy of sense unit types. To fully understand why, let us go back to the theory of semantic actants (c.f., Mel'čuk, 2004a). The **SemASlots** of a semantic unit type $\langle L \rangle$ correspond roughly to participants of the linguistic situation $\text{SIT}(L)$ denoted by $\langle L \rangle$ that may be expressed in the sentence in a favoured manner by the lexical unit L . Thus as one specialize L (i.e., the meaning of the new lexical unit is more specific than the meaning of L), the **SemASlots** may change. If the number 1 disappears, then

the new number 1 does not correspond to the old one. In a general manner, The **SemASlot** i of a lexical unit does not necessarily corresponds to the **SemASlot** i of other lexical units, and may correspond to the **SemASlot** j of another type of lexical unit.

We wish to enforce a strong coherence in the unit types hierarchy, and for that we use the following remark: unlike the **SemASlots**, all participants of the $SIT(L)$ are inherited as we specialize the meaning of lexical unit L . We thus introduce the notion of **PSlots**⁸ of a unit type. As we shall see, **PSlots** of a semantic unit type (L) is roughly : the union of (L) 's **ASlots** and of every semantic unit type more generic than (L) 's. Participant positions of (L) thus correspond to a subset of participants of $SIT(L)$ that is preserved by inheritance. We then abandon the numbering system of predicate-argument relation in favour of a naming system. We thus introduce a finite set of so-called **Participation Symbols (PSymbols)**, that are binary relation symbols of type predicate-argument. Every **PUT** t has thus zero or more so-called **PSlot**, which represents slots that instances of t have and which may be filled by other units. Each **PSlots** of a **PUT** has a symbol that is chosen among the set of **PSymbols**.

Definition 2.5 (set of **PSymbols**, **PSlots** of a **PUT**, and ω -valency of a **PUT**). A **Participation Symbol (PSymbol)** set is a finite set denoted $\mathcal{S}_{\mathcal{T}}$. Any **PUT** has a set of **PSlots** which is defined through a mapping ω from \mathbf{T} to $2^{\mathcal{S}_{\mathcal{T}}}$. For every $t \in \mathbf{T}$, $s \in \omega(t)$ means that t has a **PSlot** with **PSymbol** s . The number of **PSlots** of a **PUT** t is denoted the ω -valency of t , i.e., $valency_{\omega}(t) \stackrel{\text{def}}{=} |\omega(t)|$.

For instance, in our example illustrated on figure 2 it is assumed that there is a **PSymbol** $obj \in \mathcal{S}_{\mathcal{T}}$ to carry the participation meaning *what moves*. Moreover, it is assumed and illustrated that the **PUT** $(fall)$ has a **PSlot** having symbol obj , i.e., $obj \in \omega((fall))$.

Note. Let s be a **PSymbols**. As a shortcut, instead of "**PSlot** having symbol s ", we simply write: "**PSlot** s ".

We also introduce the notion the root of a **PSymbol**. Every **PSymbol** s has a so-called *root* in \mathbf{T} denoted $root(s)$ that introduces a **PSlot** with symbol s . We define the set of **PSlots** of a **PUT** $t \in \mathbf{T}$ as being the set of **PSlots** whose root is more general or equivalent to t :

Definition 2.6 (Root of a **PSlot**). The *root* of **PSlots** is defined through a mapping $root$ from $\mathcal{S}_{\mathcal{T}}$ to \mathbf{T} that links a **PSlot** s to the **PUT** that introduces s . The set of **PSlots** of a **PUT** $t \in \mathbf{T}$ is defined as the set of **PSlots** whose root is more general than t , i.e., $\forall t \in \mathbf{T}, \omega(t) \stackrel{\text{def}}{=} \{s \in \mathcal{S}_{\mathcal{T}} \mid t \lesssim root(s)\}$.

For instance, so that $(fall)$ has a **PSlot** obj , it must be more specific or equivalent than the root of obj . In our example we chose $root(obj) = (fact)$.

In the following, we use "more general" and "more specific" in their weak sense, i.e., respectively "more general or equivalent" and "more specific or equivalent", unless otherwise stated.

If we were to define a **PUT** more specific than $(fall)$ such as $(tumble)$, then it would inherit the **PSlot** obj . Actually, every **PUT** more specific than $root(obj)$, and only those **PUTs** more specific than $root(obj)$ will have a **PSlot** with symbol obj . The following property thus holds:

Proposition 2.2. Let $\downarrow root(s)$ be the smallest lower set of \mathbf{T} containing $root(s)$, i.e., $\downarrow root(s) \stackrel{\text{def}}{=} \{t \in \mathbf{T} \mid t \lesssim root(s)\}$. For any $s \in \mathcal{S}_{\mathcal{T}}$, $\{t \in \mathbf{T} \mid s \in \omega(t)\} = \downarrow root(s)$.

Proof. Let $s \in \mathcal{S}_{\mathcal{T}}$.

\subseteq : Let $t \in \{t \in \mathbf{T} \mid s \in \omega(t)\}$. $s \in \omega(t)$, by definition 2.6, $t \lesssim root(s)$, so $t \in \downarrow root(s)$.

\supseteq : Let $t \in \downarrow root(s)$. $t \lesssim root(s)$, and by definition 2.6, $s \in \omega(t)$, so $t \in \{t \in \mathbf{T} \mid s \in \omega(t)\}$. \square

⁸Despite the fact that the Moscow Semantic School likes to distinguish the notions of participants of a situation vs. semantic actants, we chose to use term *participant slot* instead of a neologism such that *actanciable slot* for instance.

In our example illustrated on figure 2, the root of **PSymbol** obj is illustrated by symbol \star , and the green zone represents the set of **PUTs** that have a **PSlot** obj , i.e., the principal lower set of \mathbf{T} generated by $root(obj)$.

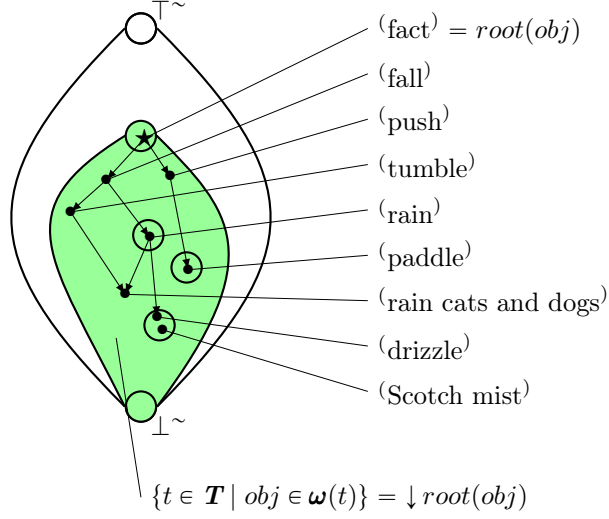


Figure 2: Illustration of the set of **PUTs** that have a **PSLOT** obj .

As a direct consequence, as units get more and more specific (i.e., as we go down the hierarchy of **PUTs**), the set of **PSlots** may only increase. We say that **PSlots** are inherited:

Proposition 2.3. *PSlots are inherited, i.e.,*

$$\forall t_x, t_y \in \mathbf{T} \text{ such that } t_x \lesssim t_y, \omega(t_y) \subseteq \omega(t_x) \quad (1)$$

Moreover, if $t_x \simeq t_y$, then $\omega(t_y) = \omega(t_x)$.

Proof. Let $t_x, t_y \in \mathbf{T}$ such that $t_x \lesssim t_y$, and $s \in \omega(t_y)$.

Definition 2.6 implies $t_y \lesssim root(s)$. So $t_x \lesssim t_y \lesssim root(s)$ and $s \in \omega(t_x)$.

The second result is obtained using twice this first result. \square

Moreover, as $root$ is a mapping, every **PSymbol** has a root. Any minimal element of \mathbf{T} will inherit all the **PSlots**:

Proposition 2.4. $\forall t \in \perp^\sim, \omega(t) = \mathcal{S}_{\mathcal{T}}$.

Proof. As \perp is a minimal element in \mathbf{T} , then $\forall s \in \mathcal{S}_{\mathcal{T}}, \perp \lesssim root(s)$. So $s \in \omega(\perp)$. Thus $\omega(\perp) = \mathcal{S}_{\mathcal{T}}$. Now, for all $t \in \perp^\sim$ we know that $t \lesssim \perp$, and using proposition 2.3 $\mathcal{S}_{\mathcal{T}} \subseteq \omega(t)$. So $\omega(t) = \mathcal{S}_{\mathcal{T}}$. \square

2.3 PUT Hidden Slots (HSlots)

Although the set of **PSlots** increases as we go down the hierarchy of units due to the introduction of the new **PSlots**, the set of **ASlots** of a unit type remains in the range of one to approximately six according to the **MTT Mel'čuk (2004a,b)**. Others are still part of the linguistic situation, but somehow hidden and no-longer expressible in a favoured manner in the natural language utterance representation. We thus say that those positions among the **PSlots** that are not actantial are *hidden*, and we hence introduce the notion of **Hidden Slots (HSlots)** of a **PUT**.

For instance when it drizzles, what moves, i.e., water, is not expressible in a favoured manner by the lexical unit of type (type) anymore. *obj* is thus a **HSlot** for (drizzle).

Like for the root, we consider that every **PSymbol** s has a set of so-called *hiders* in \mathbf{T} denoted $Hiders(s)$ that hide a **PSlot** s , and that the set of **HSlots** of a **PUT** $t \in \mathbf{T}$ is the set of **PSlots** that have a hider more general or equivalent to t :

Definition 2.7 (Hiders of a **PSymbol**, and **HSlots** of a **PUT**). The *hiders* of a **PSlot** is defined through a mapping $Hiders$ from $\mathbf{S}_{\mathcal{T}}$ to $2^{\downarrow root(s)} \setminus \emptyset$ that links a participation symbol s to the **PUTs** that have their **PSlot** s hidden. The set of **HSlots of PUTs** is defined through a mapping $\bar{\alpha}$ from \mathbf{T} to $2^{\mathbf{S}_{\mathcal{T}}}$ such that $\forall t \in \mathbf{T}, \bar{\alpha}(t) \stackrel{\text{def}}{=} \{s \in \mathbf{S}_{\mathcal{T}} \mid \exists t_h \in Hiders(s), t \lesssim t_h\}$

In our example, so that the **PSlot** *obj* cannot be expressed for (drizzle), this **PUTs** must be more specific than one of the hiders of *obj*. For instance, (rain) $\in Hiders(obj)$. Furthermore, if we were to define another **PUT** more specific than (rain) such as (rain cats and dogs), then in every utterance in which a linguistic unit of type (rain cats and dogs) would be involved, the **PSlot** *obj* is not actantial neither and is thus hidden. The hidden state of the **PSlot** *obj* will actually be inherited by every **PUT** more specific than any **PUT** in $Hiders(obj)$, and only those **PUTs** more specific than one **PUT** of $Hiders(obj)$ will have **PSlot** *obj* hidden⁹. The following property thus holds:

Proposition 2.5. Let $\downarrow Hiders(s)$ be the smallest lower set of \mathbf{T} that contains $Hiders(s)$, i.e., $\downarrow Hiders(s) \stackrel{\text{def}}{=} \{t \in \mathbf{T} \mid \exists t_h \in Hiders(s) : t \lesssim t_h\}$. For any $s \in \mathbf{S}_{\mathcal{T}}$, $\{t \in \mathbf{T} \mid s \in \bar{\alpha}(t)\} = \downarrow Hiders(s)$.

Proof. Let $s \in \mathbf{S}_{\mathcal{T}}$.

\subseteq : Let $t \in \{t \in \mathbf{T} \mid s \in \bar{\alpha}(t)\}$. $s \in \bar{\alpha}(t)$.

By definition 2.7, $\exists t_h \in Hiders(s), t \lesssim t_h$. Thus $t \in \downarrow Hiders(s)$.

\supseteq : Let $t \in \downarrow Hiders(s)$. $\exists t_h \in Hiders(s), t \lesssim t_h$

By definition 2.7, $s \in \bar{\alpha}(t)$. Thus $t \in \{t \in \mathbf{T} \mid s \in \bar{\alpha}(t)\}$. □

In our example illustrated on figure 3, the root of **PSymbol** *obj* is illustrated by symbol \star , circles represent equivalence classes, and the hiders of *obj* are illustrated by symbol \blacklozenge . The gray zone represents the set of **PUTs** that have a **HSlot** *obj*, i.e., the lower set of \mathbf{T} generated by $Hiders(obj)$.

As a direct consequence, as units get more and more specific (i.e., as we go down the hierarchy of **PUTs**), the set of **HSlots** may only increase. We say that **HSlots** are inherited:

Proposition 2.6. *Hidden slots are inherited, i.e.,*

$$\forall t_x, t_y \in \mathbf{T} \text{ such that } t_x \lesssim t_y, \bar{\alpha}(t_y) \subseteq \bar{\alpha}(t_x) \quad (2)$$

Moreover, if $t_x \simeq t_y$, then $\bar{\alpha}(t_y) = \bar{\alpha}(t_x)$.

⁹The good mathematical properties that this default behaviour brings do not limit the expressivity of our formalism. Indeed, we shall see later using unit type definitions that it is possible contravene this rule by inserting a new **PSlot** that corresponds to the **PSlot** hidden otherwise.

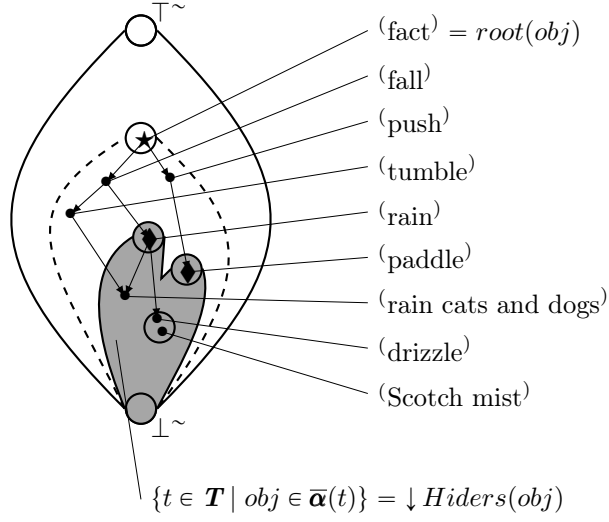


Figure 3: Illustration of the set of **PUTs** that have a **HSlot** obj .

Proof. Let $t_x, t_y \in \mathbf{T}$ such that $t_x \lesssim t_y$, and $s \in \bar{\alpha}(t_y)$.

Definition 2.7 implies $\exists t_h \in Hiders(t_x), t_y \lesssim t_h$. So $t_x \lesssim t_y \lesssim t_h$ and $s \in \bar{\alpha}(t_x)$.

The second result is obtained using twice this first result. \square

Second, as $Hiders$ is taken in the lower set generated by $root(s)$, i.e., $2^{\downarrow root(s)} \setminus \emptyset$, any $hiders$ is lower than $root(s)$. So the set of **HSlots** is always a subset of the set of **PSlots**:

Proposition 2.7. For any **PUT** $t \in \mathbf{T}$, $\bar{\alpha}(t)$ is a subset of $\omega(t)$.

Proof. Let $t \in \mathbf{T}$, and $s \in \bar{\alpha}(t)$.

From definition 2.7, $\exists t_h \in Hiders(s), t \lesssim t_h$.

Then as $Hiders(s) \in 2^{\downarrow root(s)} \setminus \emptyset$, we know that $t \lesssim t_h \lesssim root(s)$.

So by definition 2.6, $s \in \omega(t)$ and $\bar{\alpha}(t) \subseteq \omega(t)$. \square

Finally, as $Hiders$ is a mapping and cannot take value \emptyset , every **PSymbol** has at least one **hider**. Any minimal element of \mathbf{T} will hence inherit all of the **HSlots**:

Proposition 2.8. $\forall t \in \perp^\sim, \bar{\alpha}(t) = \mathbf{S}_{\mathcal{T}}$.

Proof. Let $s \in \mathbf{S}_{\mathcal{T}}$.

From definition 2.7 we know that $Hiders(s)$ is not empty and thus $\exists t_h \in Hiders(s)$.

As $\perp \lesssim t_h$, then $\perp \in \downarrow Hiders(s)$ and $s \in \bar{\alpha}(\perp)$.

Now, let $t \in \perp^\sim, t \lesssim \perp$ and using proposition 2.6, $\mathbf{S}_{\mathcal{T}} \subseteq \bar{\alpha}(t)$.

So for all $t \in \perp^\sim, \mathbf{S}_{\mathcal{T}} \subseteq \bar{\alpha}(t)$, and thus $\mathbf{S}_{\mathcal{T}} = \bar{\alpha}(t)$. \square

2.4 PUT Actant Slots (ASlots)

The set of **Actant Slots (ASlots)** of a **PUT** is defined as the set of its participant slots that are not hidden:

Definition 2.8 (**ASlots** and α -valency of a **PUT**). The set of **ASlots** of **PUTs** is defined through a mapping α from \mathbf{T} to $2^{\mathcal{S}^T}$ such that $\forall t \in \mathbf{T}, \alpha(t) \stackrel{\text{def}}{=} \omega(t) - \bar{\alpha}(t)$. The number of **ASlots** of a **PUT** t is denoted the α -valency of t , i.e., $\text{valency}_\alpha(t) \stackrel{\text{def}}{=} |\alpha(t)|$.

For instance, (paddle) inherits **PSlots** *who* and *obj* from (push), but as **PSlots** *obj* is hidden for (paddle), then it only has **ASlots** *who*.

The following proposition directly derives from definition 2.8:

Proposition 2.9. $\forall t \in \mathbf{T}, \omega(t) = \alpha(t) \cup \bar{\alpha}(t)$, i.e., $\omega(t)$ is the disjoint union of $\alpha(t)$ and $\bar{\alpha}(t)$.

Proof. From definition 2.8: $\forall t \in \mathbf{T}, \alpha(t) = \omega(t) - \bar{\alpha}(t) \Leftrightarrow \forall t \in \mathbf{T}, \omega(t) = \alpha(t) \cup \bar{\alpha}(t)$ \square

Furthermore, due to propositions 2.2 and 2.5, we know that any **PUT** with **ASlot** *obj* is in lower set $\downarrow \text{root}(obj)$ and is not in lower set $\downarrow \text{Hiders}(obj)$. So the following proposition holds:

Proposition 2.10. For any $s \in \mathcal{S}_{\mathcal{T}}$,

$$\{t \in \mathbf{T} \mid s \in \alpha(t)\} = \downarrow \text{root}(s) - \downarrow \text{Hiders}(s) = \{t \in \mathbf{T} \mid t \lesssim \text{root}(s) \text{ and } \forall t_h \in \text{Hiders}(s), t \not\lesssim t_h\} \quad (3)$$

Proof. Let $s \in \mathcal{S}_{\mathcal{T}}$.

\subseteq : Let $t \in \{t \in \mathbf{T} \mid s \in \alpha(t)\}$. $s \in \alpha(t)$.

By definition 2.8, $s \in \omega(t) - \bar{\alpha}(t)$.

By definition 2.6 and proposition 2.7, $s \lesssim \text{root}(s)$ and $\forall t_h \in \text{Hiders}(s), t \not\lesssim t_h$.

So $t \in \downarrow \text{root}(s) - \downarrow \text{Hiders}(s)$.

\supseteq : Let $t \in \downarrow \text{root}(s) - \downarrow \text{Hiders}(s)$.

$s \lesssim \text{root}(s)$ and $\forall t_h \in \text{Hiders}(s), t \not\lesssim t_h$.

By definition 2.6 and proposition 2.7, $s \in \omega(t)$ and $s \notin \bar{\alpha}(t)$.

So $s \in \alpha(t)$, and $t \in \{t \in \mathbf{T} \mid s \in \alpha(t)\}$. \square

In our example illustrated on figure 4, the root of **PSymbol** *obj* is illustrated by symbol \star , circles represent equivalence classes, and the hiders of *obj* are illustrated by symbol \blacklozenge . The yellow zone represents the set of **PUTs** that have a **ASlot** *obj*, i.e., the difference between the lower set of \mathbf{T} generated by $\text{root}(obj)$, and the lower set of \mathbf{T} generated by $\text{Hiders}(obj)$.

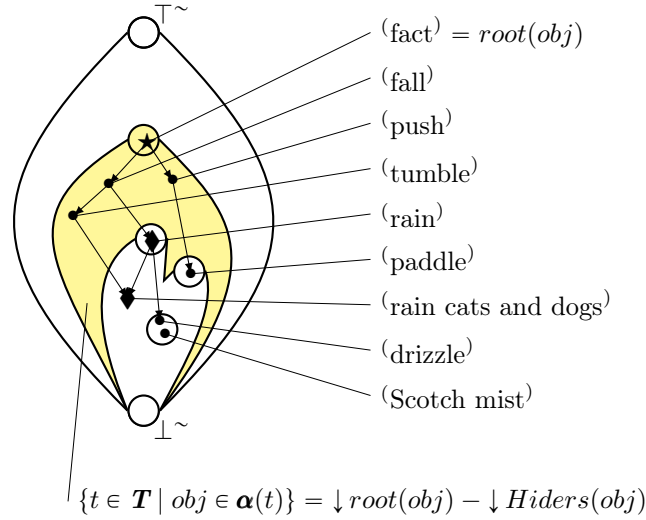


Figure 4: Illustration of the set of **PUT** that have **ASlots** *obj*.

Thus in accordance with the **MTT**, **ASlots** are not always inherited by **PUTs**: they may be introduced, be inherited for a while, and disappear. Now from propositions 2.3 and 2.6, one concludes that two equivalent **PUTs** share the same set of **ASlots**:

Proposition 2.11. $\forall t, t' \in \mathbf{T}$ such that $t \simeq t'$, $\alpha(t) = \alpha(t')$.

Proof. From definition 2.8, $\alpha(t) = \omega(t) - \bar{\alpha}(t)$.

Moreover, from propositions 2.3 and 2.6, $\omega(t') = \omega(t)$ and $\bar{\alpha}(t') = \bar{\alpha}(t)$.

So $\omega(t') = \omega(t)$. □

Finally, from propositions 2.4 and 2.8, one concludes that any absurd **PUT** has no **ASlot**:

Proposition 2.12. $\forall t \in \perp\sim, \alpha(t) = \emptyset$.

Proof. Let $t \in \perp\sim$. From definition 2.8 and propositions 2.4 and 2.8, $\alpha(t) = \mathcal{S}_{\mathcal{T}} - \mathcal{S}_{\mathcal{T}} = \emptyset$. □

2.5 PUT Signatures

For any **PUT**, not any unit may fill one of its specific **ASlot**. For instance, any unit that fills **ASlot** *obj* of a unit with type (move) should be of type (object), and absolutely not of type (idea) nor (eat). *Signatures* enable the assertion of what type fillers of **PSlots** are.

As **PUTs** get more and more specific, the signature of a given common **ASlot** may only become more and more specific. For instance, any unit that fills **ASlot** *obj* of a unit with type (move) should be of type (object), but only units with type (weighing object) may fill **ASlots** *obj* of a unit with type (fall).

Moreover, when a **PSlot** becomes hidden, its signature still is relevant and may be further restricted. For instance, although **PSlot** *obj* is hidden for (rain), any unit that fills that **HSlot** should be of type (water drops). And for (drizzle) it becomes (small water drops).

Definition 2.9 (Signatures of **PUTs**). The set of signatures of **PUTs** $\{\varsigma_t\}_{t \in \mathbf{T}}$ is a set of functions from $\mathcal{S}_{\mathcal{T}}$ to $2^{\mathcal{T}}$. For every **PUT** t , ς_t is a function with $domain(\varsigma_t) = \omega(t)$ that associates to each **PSlot** s of t a set of **PUT** $\varsigma_t(s)$ that characterize the type units that may fill this slot should be.

The set of signatures $\{\mathfrak{s}_t\}_{t \in \mathbf{T}}$ must be such that for all $t_1, t_2 \in \mathbf{T}$, and $s \in \mathbf{S}_{\mathcal{T}}$ such that $t_1 \lesssim t_2$ and $s \in \boldsymbol{\omega}(t_2)$, $\forall t'_2 \in \mathfrak{s}_{t_2}(s), \exists t'_1 \in \mathfrak{s}_{t_1}(s) : t'_1 \lesssim t'_2$.

The above definition is complex due to the fact that important notions will be introduced in next section. We will see that $2^{\mathbf{T}}$ is the set of so-called **CUTs**, and thus the signature \mathfrak{s}_t of t is a **CUT**. Moreover, when the pre-order relation $\hat{\lesssim}$ over the set of **CUTs** will be introduced, we will see that $\forall t'_2 \in \mathfrak{s}_{t_2}(s), \exists t'_1 \in \mathfrak{s}_{t_1}(s) : t'_1 \lesssim t'_2$ implies $\mathfrak{s}_{t_1}(s) \hat{\lesssim} \mathfrak{s}_{t_2}(s)$. Thus the set of signatures $\{\mathfrak{s}_t\}_{t \in \mathbf{T}}$ must be such that for all $t_1, t_2 \in \mathbf{T}$, and $s \in \mathbf{S}_{\mathcal{T}}$ such that $s \in \boldsymbol{\omega}(t_2)$,

$$t_1 \lesssim t_2 \Rightarrow \mathfrak{s}_{t_1}(s) \hat{\lesssim} \mathfrak{s}_{t_2}(s) \quad (4)$$

3 Conjunctive Unit Types (CUTs)

In the **Meaning-Text Theory (MTT)**, a unit may actually have multiple types. For instance, a unit may have a lexical unit type and multiple grammatical unit types for instance, like $\{\lceil \text{def}^{\text{L}G}, \lceil \text{USA}^{\text{L}L} \rceil\}$ for "the USA".

In section 3.1 we hence introduce the set of **Conjunctive Unit Type (CUT)** as the set of all subsets of \mathbf{T} , and will define the set of asserted absurd CUT set, which is a set of combinations of PUTs that may not have instances.

Any CUT may also be seen both as a concept and as a n -ary relation, inheriting as such its constituting PUTs Participant Slots (PSlots), and Hidden Slots (HSlots) (but not their Actant Slots (ASlots) !). Section 3.2 introduces slots and signatures for CUTs.

Then, section 3.3 introduces a specialization pre-order over CUTs, which is build from the pre-order over constituent PUTs, asserted absurd CUTs and absurd signatures. A natural definition of absurd CUTs is then given.

Finally, a first definition of the hierarchy of CUTs is introduced in 3.4.

3.1 Definition of CUTs

A unit has multiple types. For instance $\{\lceil \text{def}^{\text{L}G}, \lceil \text{USA}^{\text{L}L} \rceil\}$ for "the USA" is composed of a lexical unit type $\lceil \text{USA}^{\text{L}L} \rceil$ and a grammatical unit type $\lceil \text{def}^{\text{L}G} \rceil$. We hence introduce the set of possible **Conjunctive Unit Type (CUT)** which is the powerset of \mathbf{T} , i.e., the set of all subsets of \mathbf{T} :

Definition 3.1 (CUT set over \mathbf{T}). The **CUT set** over \mathbf{T} is the set of all subsets of \mathbf{T} , i.e., $\mathbf{T}^\cap \stackrel{\text{def}}{=} 2^{\mathbf{T}}$. Every element in \mathbf{T}^\cap is denoted both **CUT** and by the set itself. $\top^\cap = \{\top\}$ is denoted the *prime universal CUT*, and $\perp^\cap = \{\perp\}$ the *prime absurd CUT*. Any singleton CUT $\{t\}$ is said to be primitive, i.e., a primitive CUT.

Some CUTs such as $\{\lceil \text{def}^{\text{L}G}, \lceil \text{indef}^{\text{L}G} \rceil\}$ are said to be absurd. This means there is no unit that has both types $\lceil \text{def}^{\text{L}G} \rceil$ and $\lceil \text{indef}^{\text{L}G} \rceil$. Asserted absurd CUT set is defined as follows:

Definition 3.2 (Asserted absurd CUT set). The set of *asserted absurd CUTs* is a set of CUTs that is denoted $\perp_{\text{asserted}}^\cap$. And $\perp_{\text{asserted}}^\cap \subseteq \mathbf{T}^\cap$.

3.2 CUT Slots and Signatures

PSlots, HSlots, ASlots and signatures are naturally extended to CUTs.

For instance, consider a CUT $t^\cap = \{(\text{move}), (\text{quick})\}$. t^\cap contains the PUT (move) , and as $\text{obj} \in \omega((\text{move}))$, obj corresponds to a participant of SIT(move). As SIT(move,quick) is a specialization of SIT(move), we consider that obj is also a PSlot of t^\cap :

Definition 3.3 (Participant Slots of CUTs, and ω -valency of a CUT). The set of *PSlots of CUTs* is defined through a mapping ω^\cap from \mathbf{T}^\cap to $2^{\mathcal{S}^\tau}$ Such that $\forall t^\cap \in \mathbf{T}^\cap, \omega^\cap(t^\cap) \stackrel{\text{def}}{=} \bigcup_{t \in t^\cap} \omega(t)$. The number of PSlots of a CUT t^\cap is denoted the ω -valency of t^\cap , i.e., $\text{valency}_\omega(t^\cap) \stackrel{\text{def}}{=} |\omega^\cap(t^\cap)|$.

Now, consider CUT $t^\cap = \{(\text{move}), (\text{rain})\}$. PSlot obj is hidden for (rain) , but is not for (move) . PSlot obj is a PSlot for t^\cap and we consider that being hidden has priority over being an actant:

Definition 3.4 (Hidden Slots of a CUTs). The set of *HSlots of CUTs* is defined through a mapping $\bar{\alpha}^\cap$ from \mathbf{T}^\cap to $2^{\mathcal{S}^\tau}$ Such that $\forall t^\cap \in \mathbf{T}^\cap, \bar{\alpha}^\cap(t^\cap) \stackrel{\text{def}}{=} \bigcup_{t \in t^\cap} \bar{\alpha}(t)$.

Naturally, a **HSlot** is a hidden **PSlot**, so it is a **PSlot**. The following proposition underlines the fact that the previous definitions are coherent with this natural thought:

Proposition 3.1. For any **CUT** $t^\wedge \in \mathbf{T}^\wedge$, $\bar{\alpha}^\wedge(t^\wedge)$ is a subset of $\omega^\wedge(t^\wedge)$.

Proof. Let $t^\wedge \in \mathbf{T}^\wedge$ and $s \in \bar{\alpha}^\wedge(t^\wedge)$.

By definition 3.4, $\exists t \in t^\wedge, s \in \bar{\alpha}(t)$.

From proposition 2.7, $s \in \omega(t)$.

By definition 3.3, $s \in \omega^\wedge(t^\wedge)$. □

Finally, similarly as for **PUTs**, the **ASlots** of a **CUT** is defined as the difference between the set of **PSlots** and the set of **HSlots**:

Definition 3.5 (Actant Slots of a **CUTs**, and α -valency of a **CUT**). The set of **ASlots** of **CUTs** is defined through a mapping α^\wedge from \mathbf{T}^\wedge to $2^{\mathcal{S}_T}$ Such that $\forall t^\wedge \in \mathbf{T}^\wedge, \alpha^\wedge(t^\wedge) \stackrel{\text{def}}{=} \omega^\wedge(t^\wedge) - \bar{\alpha}^\wedge(t^\wedge)$. The number of **ASlots** of a **CUT** t^\wedge is denoted the α -valency of t^\wedge , i.e., $\text{valency}_\alpha(t^\wedge) \stackrel{\text{def}}{=} |\alpha^\wedge(t^\wedge)|$.

Just like for **PUTs**, any **PSlot** is either an actant, or hidden:

Proposition 3.2. $\forall t^\wedge \in \mathbf{T}^\wedge, \omega^\wedge(t^\wedge) = \alpha^\wedge(t^\wedge) \cup \bar{\alpha}^\wedge(t^\wedge)$, i.e., $\omega^\wedge(t^\wedge)$ is the disjoint union of $\alpha^\wedge(t^\wedge)$ and $\bar{\alpha}^\wedge(t^\wedge)$.

Proof. From proposition 4.6,

$\forall t^\wedge \in \mathbf{T}^\wedge, \alpha^\wedge(t^\wedge) = \omega^\wedge(t^\wedge) - \bar{\alpha}^\wedge(t^\wedge) \Leftrightarrow \forall t^\wedge \in \mathbf{T}^\wedge, \omega^\wedge(t^\wedge) = \alpha^\wedge(t^\wedge) \cup \bar{\alpha}^\wedge(t^\wedge)$. □

Signatures are also naturally extended to **CUTs**, but for the **PSlot** *obj* for instance, we must take the union of signatures of **PUTs** constituents over a more complex set: the set of **PUTs** that have the **PSlot** *obj*: $\{t \in t^\wedge \mid \text{obj} \in \omega(t)\}$:

Definition 3.6 (Signature of a **CUT**). The set of signatures of **CUTs** $\{\zeta_{t^\wedge}^\wedge\}_{t^\wedge \in \mathbf{T}^\wedge}$ is a set of functions from \mathcal{S}_T to \mathbf{T}^\wedge . For every **CUT** t^\wedge , $\zeta_{t^\wedge}^\wedge$ is a function with $\text{domain}(\zeta_{t^\wedge}^\wedge) = \omega^\wedge(t^\wedge)$ such that for all s in $\omega^\wedge(t^\wedge)$, $\zeta_{t^\wedge}^\wedge(s) \stackrel{\text{def}}{=} \bigcup_{t \in t^\wedge \mid s \in \omega(t)} \zeta_t(s)$.

Straightforwardly, the only **PSlots** of a singleton **CUT** are the **PSlots** of its single **PUT** element:

Proposition 3.3. For all $t \in \mathbf{T}$, all of the following is true:

- $\omega^\wedge(\{t\}) = \omega(t)$.
- $\bar{\alpha}^\wedge(\{t\}) = \bar{\alpha}(t)$.
- $\alpha^\wedge(\{t\}) = \alpha(t)$.
- for all $s \in \omega(t)$, $\zeta_{\{t\}}^\wedge(s) = \zeta_t(s)$

Proof. Let $t \in \mathbf{T}$.

1) $\omega^\wedge(\{t\}) = \bigcup_{t \in \{t\}} \omega(t) = \omega(t)$;

2) $\bar{\alpha}^\wedge(\{t\}) = \bigcup_{t \in \{t\}} \bar{\alpha}(t) = \bar{\alpha}(t)$;

3) $\alpha^\wedge(\{t\}) = \omega^\wedge(\{t\}) - \bar{\alpha}^\wedge(\{t\}) = \omega(t) - \bar{\alpha}(t) = \alpha(t)$.

4) let $s \in \omega(t)$. $t \in \{t \in t^\wedge \mid s \in \omega(t)\}$, so $\zeta_{\{t\}}^\wedge(s) = \bigcup_{t \in \{t\} \mid s \in \omega(t)} \zeta_t(s) = \zeta_t(s)$. □

3.3 Pre-order over CUTs

In this section we define a Pre-order over \mathbf{T}° which models a specialization relation, e.g., $\{(\text{happy}), (\text{cat})\} \lesssim \{(\text{hasMood}), (\text{animal})\}$ means that $\{(\text{happy}), (\text{cat})\}$ is more specific (semantically) than $\{(\text{hasMood}), (\text{animal})\}$.

Let us first introduce the definition of an iterative construction process of a comparisons set over \mathbf{T}° , and we will describe the intuitive meaning of every element of this definition just after:

Definition 3.7 (Comparisons set over \mathbf{T}°). Let $(\mathbf{C}_n^\circ)_{n \in \mathbb{N}}$ be a sequence of comparisons sets over \mathbf{T}° , i.e., for all $n \in \mathbb{N}$, $\mathbf{C}_n^\circ \subseteq \mathbf{T}^{\circ 2}$, defined by

- for $i = 0$, $\mathbf{C}_0^\circ \stackrel{\text{def}}{=} \mathbf{C}_\lesssim^\circ \cup \mathbf{C}_\top^\circ \cup \mathbf{C}_\perp^\circ$, where:

$$\mathbf{C}_\lesssim^\circ \stackrel{\text{def}}{=} \{(t_y^\circ, t_x^\circ) \in \mathbf{T}^{\circ 2} \mid \forall t_y \in t_y^\circ, \exists t_x \in t_x^\circ : t_x \lesssim t_y\} \quad (5)$$

$$\mathbf{C}_\top^\circ \stackrel{\text{def}}{=} \{(\top^\circ, \emptyset)\} \quad (6)$$

$$\mathbf{C}_\perp^\circ \stackrel{\text{def}}{=} \{(\perp^\circ, t^\circ) \in \mathbf{T}^{\circ 2} \mid t^\circ \in \perp_{\text{asserted}}^\circ\} \quad (7)$$

- for all $i > 0$, $\mathbf{C}_i^\circ \stackrel{\text{def}}{=} \mathbf{C}_{i-1}^\circ \cup \mathbf{C}_{\xi_i}^\circ \cup \mathbf{C}_{+i}^\circ$ where:

$$\mathbf{C}_{\xi_i}^\circ \stackrel{\text{def}}{=} \{(\perp^\circ, t^\circ) \in \mathbf{T}^{\circ 2} \mid \exists s \in \omega^\circ(t^\circ), (\perp^\circ, \xi_{i^\circ}(s)) \in \mathbf{C}_{i-1}^\circ\} \quad (8)$$

$$\mathbf{C}_{+i}^\circ \stackrel{\text{def}}{=} \{(t_z^\circ, t_x^\circ) \in \mathbf{T}^{\circ 2} \mid \exists t_y^\circ \in \mathbf{T}^\circ, (t_z^\circ, t_y^\circ) \in \mathbf{C}_{i-1}^\circ \text{ and } (t_y^\circ, t_x^\circ) \in \mathbf{C}_{i-1}^\circ\} \quad (9)$$

The sequence (\mathbf{C}_n°) is a bounded monotonic increasing sequence, i.e., for all $n \in \mathbb{N}$, $\mathbf{C}_n^\circ \subseteq \mathbf{C}_{n+1}^\circ \subseteq \mathbf{T}^{\circ 2}$, so it is convergent. The least upper bound of the sequence $(\mathbf{C}_n^\circ)_{n \in \mathbb{N}}$ is denoted \mathbf{C}° : the *comparisons set over \mathbf{T}°* .

The intuitive meaning of the iterative construction process is the following:

- $\mathbf{C}_\lesssim^\circ$ represents the natural extension of a pre-order over a set to a pre-order over its powerset;
- \mathbf{C}_\top° is introduced to flatten the top of the powerset so that \top° is the most generic **CUT**;
- \mathbf{C}_\perp° is introduced to flatten the bottom of the powerset so that any asserted absurd **CUT** is considered as a minimal **CUT** (a most specific **CUT**);
- $\mathbf{C}_{\xi_i}^\circ$ represents the fact that if a signature of a **CUT** for a given **PSlot** is absurd, then that **CUT** is absurd;
- \mathbf{C}_{+i}° transitively closes the set of comparisons \mathbf{C}° .

It is a well know result that $\mathbf{C}_\lesssim^\circ$ defines a pre-order over $2^{\mathbf{T}^\circ}$ (i.e., it is reflexive and transitive), let us recall how this result is obtained:

Proposition 3.4. $\mathbf{C}_\lesssim^\circ$ is reflexive and transitive.

Proof. Reflexivity: Let $t^\circ \in \mathbf{T}^\circ$.

$\forall t \in t^\circ, \exists t' (= t) \in t^\circ : t' \lesssim t$.

So $(t^\circ, t^\circ) \in \mathbf{C}_\lesssim^\circ$.

Transitivity: Let $t_x^\circ, t_y^\circ, t_z^\circ \in \mathbf{T}^\circ$ such that $(t_z^\circ, t_y^\circ) \in \mathbf{C}_\lesssim^\circ$, and $(t_y^\circ, t_x^\circ) \in \mathbf{C}_\lesssim^\circ$.

$\forall t_z \in t_z^\circ, \exists t_y \in t_y^\circ : t_y \lesssim t_z$, and $\forall t'_y \in t_y^\circ, \exists t_x \in t_x^\circ : t_x \lesssim t'_y$.

Let $t'_y = t_y$, then $\forall t_z \in t_z^\circ, \exists t_x \in t_x^\circ : t_x \lesssim t_z$,

and thus $(t_z^\circ, t_x^\circ) \in \mathbf{C}_\lesssim^\circ$. □

Now from that result and the construction process of \mathbf{C}^\cap , we may prove the reflexivity and the transitivity of \mathbf{C}^\cap :

Proposition 3.5. \mathbf{C}^\cap is reflexive and transitive.

Proof. As $\mathbf{C}_{\lesssim}^\cap \subseteq \mathbf{C}^\cap$ and $\mathbf{C}_{\lesssim}^\cap$ is reflexive, then \mathbf{C}^\cap is reflexive.

Transitivity is obtained from \mathbf{C}_+^\cap in the construction process of \mathbf{C}^\cap . \square

Now that we know \mathbf{C}^\cap is reflexive and transitive, it defines a pre-order relation over \mathbf{T}^\cap :

Definition 3.8 (Pre-order relation over \mathbf{T}^\cap). \mathbf{C}^\cap is a binary relation over \mathbf{T}^\cap which is transitive and reflexive. So it defines a pre-order relation $\hat{\lesssim}$ over \mathbf{T}^\cap , i.e., $t_x^\cap \hat{\lesssim} t_y^\cap$ if and only if $(t_y^\cap, t_x^\cap) \in \mathbf{C}^\cap$.

The following proposition underlines sufficient conditions so that a **CUT** is more specific than another **CUT**. Each of these conditions is the expression of one of the intuitive meaning of the elements in the construction process of the comparisons set over \mathbf{T}^\cap :

Proposition 3.6. The pre-order relation $\hat{\lesssim}$ is such that:

- **Extension of \lesssim** For all $t_x^\cap, t_y^\cap \in \mathbf{T}^\cap$, if $\forall t_y \in t_y^\cap, \exists t_x \in t_x^\cap : t_x \lesssim t_y$, then $t_x^\cap \hat{\lesssim} t_y^\cap$;
- **Top CUT** $\emptyset \hat{\lesssim} \top^\cap$;
- **Asserted absurd types** For all $t^\cap \in \perp_{\text{asserted}}^\cap$, $t^\cap \hat{\lesssim} \perp^\cap$;
- **Abstrud signatures** For all $t^\cap \in \mathbf{T}^\cap$, if there exists $s \in \omega^\cap(t^\cap)$ such that $\mathfrak{S}_{t^\cap}^\cap(s) \hat{\lesssim} \perp^\cap$, then $t^\cap \hat{\lesssim} \perp^\cap$;

Proof. **Extension of \lesssim :** For all $t_x^\cap, t_y^\cap \in \mathbf{T}^\cap$, if $\forall t_y \in t_y^\cap, \exists t_x \in t_x^\cap : t_x \lesssim t_y$, then $(t_y^\cap, t_x^\cap) \in \mathbf{C}_{\lesssim}^\cap \subseteq \mathbf{C}^\cap$, so $t_x^\cap \hat{\lesssim} t_y^\cap$.

Top CUT: We know that $(\top^\cap, \emptyset) \in \mathbf{C}^\cap$, so $\emptyset \hat{\lesssim} \top^\cap$.

Asserted absurd types: For all $t^\cap \in \perp_{\text{asserted}}^\cap$, $(\perp^\cap, t^\cap) \in \mathbf{C}_\perp^\cap \subseteq \mathbf{C}^\cap$, so $t^\cap \hat{\lesssim} \perp^\cap$.

Abstrud signatures: For all $t^\cap \in \mathbf{T}^\cap$, if there exists $s \in \omega^\cap(t^\cap)$ such that $\mathfrak{S}_{t^\cap}^\cap(s) \hat{\lesssim} \perp^\cap$, then $(\perp^\cap, \mathfrak{S}_{t^\cap}^\cap(s)) \in \mathbf{C}^\cap$. There exists $n \in \mathbb{N}$ such that $(\perp^\cap, \mathfrak{S}_{t^\cap}^\cap(s)) \in \mathbf{C}_n^\cap$, and thus for $n + 1$, $(\perp^\cap, t^\cap) \in \mathbf{C}_{n+1}^\cap \subseteq \mathbf{C}^\cap$. So $t^\cap \hat{\lesssim} \perp^\cap$. \square

In the mathematical theory of categories, pre-ordered sets has category denoted **Ord** with monotonic functions as homomorphisms. In our case here, we have two pre-ordered sets: the powerset of \mathbf{T}^\cap with the inclusion relation $(\mathbf{T}^\cap, \subseteq)$ and the powerset of \mathbf{T}^\cap with the specialization relation $(\mathbf{T}^\cap, \hat{\lesssim})$. The morphism f that maps one onto the other and such that $f(t^\cap) = t^\cap$ is anti-monotonic:

Proposition 3.7. The function $f : (\mathbf{T}^\cap, \subseteq) \rightarrow (\mathbf{T}^\cap, \hat{\lesssim})$ such that $f(t^\cap) = t^\cap$ is an anti-monotonic pre-order homomorphism, i.e., for all $t_x^\cap, t_y^\cap \in \mathbf{T}^\cap$, $t_x^\cap \subseteq t_y^\cap \Rightarrow t_y^\cap \hat{\lesssim} t_x^\cap$.

Proof. Let $t_x^\cap, t_y^\cap \in \mathbf{T}^\cap$ such that $t_x^\cap \subseteq t_y^\cap$. Then $\forall t_x \in t_x^\cap, t_x \in t_y^\cap$.

Thus $\forall t_x \in t_x^\cap, \exists t_y (= t_x) \in t_y^\cap : t_y \lesssim t_x$, so by proposition 3.6, $t_y^\cap \hat{\lesssim} t_x^\cap$. \square

The set of comparisons \mathbf{C}^\cap is constructed such that the top (the less specific **CUTs**) and the bottom (the most specific **CUTs**) of the pre-ordered set $(\mathbf{T}^\cap, \hat{\lesssim})$ is flattened. The following proposition underlines the important maximal and minimal elements of $(\mathbf{T}^\cap, \hat{\lesssim})$.

Proposition 3.8. \top^\cap and \emptyset are maximal elements in \mathbf{T}^\cap , and \mathbf{T} , \perp^\cap , and \perp^\sim are minimal elements in $(\mathbf{T}^\cap, \hat{\lesssim})$.

Proof. Let $t^\circ \in \mathbf{T}^\circ$.

\emptyset : $\emptyset \subseteq t^\circ$, so using proposition 3.7: $t^\circ \lesssim \emptyset$.

\top° : Let $t^\circ \neq \emptyset$. We know that for all $t \in \mathbf{T}$, $t \lesssim \top$, so for \top in \top° , $\exists t \in t^\circ : t \lesssim \top$.

So using proposition 3.6, $t^\circ \lesssim \top^\circ$.

As we moreover know that $\emptyset \lesssim \top^\circ$, then for all $t^\circ \in \mathbf{T}^\circ$, $t^\circ \lesssim \top^\circ$.

\mathbf{T} : $t^\circ \subseteq \mathbf{T}$, so using proposition 3.7: $\mathbf{T} \lesssim t^\circ$.

\perp° : Let $t^\circ \neq \emptyset$. We know that for all $t \in \mathbf{T}$, $\perp \lesssim t$, so for all $t \in t^\circ$, $\perp \lesssim t$.

So using proposition 3.6, $\perp^\circ \lesssim t^\circ$.

As we moreover know that \emptyset is a maximal element, $\perp^\circ \lesssim \emptyset$, then for all $t^\circ \in \mathbf{T}^\circ$, $\perp^\circ \lesssim t^\circ$.

\perp^\sim : We know that $\perp \in \perp^\sim$, so item above also applies for \perp^\sim , i.e.,

for $t^\circ \neq \emptyset$, $\forall t \in t^\circ, \exists t' (= \perp) \in \perp^\sim : t' \lesssim t$, and $\perp^\sim \lesssim \emptyset$

So for all $t^\circ \in \mathbf{T}^\circ$, $\perp^\sim \lesssim t^\circ$. □

We introduce the natural equivalence relation \simeq defined by $t_x^\circ \simeq t_y^\circ \Leftrightarrow t_x^\circ \lesssim t_y^\circ$ and $t_y^\circ \lesssim t_x^\circ$. Elements in the flattened bottom of the pre-ordered set $(\mathbf{T}^\circ, \lesssim)$ are not only the most specific **CUTs**, but they all are considered absurd, and may not have instances.

Definition 3.9 (Absurd **CUT** set). The set of absurd **CUTs** is denoted \perp^\square and is the set: $\perp^\square \stackrel{\text{def}}{=} \{t^\circ \in \mathbf{T}^\circ \mid t^\circ \simeq \perp^\circ\}$.

Naturally, the prime absurd **CUT**, \perp° , is absurd.

The property of being absurd is hereditary: if a **CUT** t° is absurd, all conjunctive types lesser than t° are also absurd. For instance if $\{\text{def}^G, \text{indef}^G\}$ is asserted to be absurd and $\text{USA}^L \lesssim \text{def}^G$, then $\{\text{USA}^L, \text{indef}^G\}$ is absurd.

What is not automatic for **PUT**, is the fact that if a **PUT** has an absurd signature, then it is absurd. Consider for instance that one wants to define $\langle \text{rain-hail} \rangle$ as being a specialization of both $\langle \text{rain} \rangle$ and $\langle \text{hail} \rangle$. From definition 2.9, $\langle \text{rain-hail} \rangle$ has a **PSlot obj**, whose signature is the union of those of $\langle \text{rain} \rangle$ and $\langle \text{hail} \rangle$:

$$\begin{aligned} \mathfrak{s}_{\langle \text{rain-hail} \rangle}(\text{obj}) &= \mathfrak{s}_{\langle \text{rain} \rangle}(\text{obj}) \cup \mathfrak{s}_{\langle \text{hail} \rangle}(\text{obj}) \\ &= \{\langle \text{water} \rangle, \langle \text{liquid} \rangle\} \cup \{\langle \text{water} \rangle, \langle \text{solid} \rangle\} \\ &= \{\langle \text{water} \rangle, \langle \text{liquid} \rangle, \langle \text{solid} \rangle\} \end{aligned}$$

If one asserted that $\{\langle \text{liquid} \rangle, \langle \text{solid} \rangle\}$ is absurd and as being absurd is hereditary, the signature of **PUT** $\langle \text{rain-hail} \rangle$ is absurd. Yet, **PUT** is not automatically absurd. Equation 8 in the definition of the pre-order \lesssim enables **PUT** $\{\langle \text{rain-hail} \rangle\}$ to be absurd as it should be.

Finally, if one asserts that $\top^\circ \in \perp_{\text{asserted}}^\square$, then the whole hierarchy collapses and $\perp^\square = \mathbf{T}^\circ$. Same goes if $t^\circ \in \mathbf{T}^\circ$ such that $\top^\circ \lesssim t^\circ$, and $s \in \omega^\circ(t^\circ)$ such that $\mathfrak{s}_{t^\circ}^\circ(s) \lesssim \perp^\circ$. Then from proposition 3.6, $t^\circ \lesssim \perp^\circ$, and by the pre-order transitivity, $\perp^\square = \mathbf{T}^\circ$. Such situations must absolutely be avoided.

3.4 CUTs Hierarchy

We are now ready to introduce the **CUT** hierarchy, core of the **UGs** mathematical framework.

Definition 3.10 (CUT hierarchy). A **CUT hierarchy**

$\mathcal{T}^\cap = (T_{declared}, C_{asserted}, \perp_{asserted}^\cap, \mathcal{S}_\mathcal{T}, root, Hiders, \{\mathfrak{s}_t\}_{t \in \mathcal{T}})$, is composed of:

- $T_{declared}$ a set of declared **PUTs**;
- $C_{asserted}$ a set of asserted comparisons;
- $\perp_{asserted}^\cap$ a set of asserted absurd **CUTs**;
- $\mathcal{S}_\mathcal{T}$ a set of participation symbols;
- $root$ a mapping that associates to each **PSlot** its root **PUT**;
- $Hiders$ a mapping that associates to each **PSlot** the set of its hider **PUTs**;
- $\{\mathfrak{s}_t\}_{t \in \mathcal{T}}$ the set of signatures of **PUTs**;

The **CUTs** hierarchy is the minimal set of mathematical objects that is necessary to form the consistent core of the **UG** formalism.

4 Characterizing CUTs

This section introduces remarkable aspects of the **CUTs**.

First, for two **CUTs** $t_x^\circ, t_y^\circ \in \mathbf{T}^\circ$ and according to the construction of the pre-order over **CUTs**, we introduce in §4.1 the necessary condition to have $t_x^\circ \lesssim t_y^\circ$.

Second, now that the pre-order \lesssim is introduced, we will see in §4.2 that most properties of **PUTs** slots and signatures are valid for **CUTs** slots and signatures, except for some deteriorated cases.

Next, §4.3 introduces a natural equivalence relation over the set of **CUTs**, and the partially ordered set of equivalent classes of **CUTs**.

Finally, §4.4 and 4.5 are devoted to the introduction of remarkable subsets of **CUTs** that will be important later on for the **UGs** mathematical framework.

4.1 Necessary Conditions to Compare Two CUTs

The pre-order \lesssim over **CUTs** is slightly more complex than the simple natural extension of a pre-order over a set to a pre-order over its powerset. The necessary condition to have two **CUTs** $t_x^\circ, t_y^\circ \in \mathbf{T}^\circ$ comparable may be stated as follows:

If $t_x^\circ \lesssim t_y^\circ$, then at least one of the following is true:

- t_x° is absurd;
- t_x° is the empty set (and thus t_y° is a maximal element of \mathbf{T}°);
- t_x° is naturally more specific than t_y° , i.e., according to the natural extension of a pre-order over a set to a pre-order over its powerset.

Or more formally:

Proposition 4.1. *Let $t_x^\circ, t_y^\circ \in \mathbf{T}^\circ$. If $t_x^\circ \lesssim t_y^\circ$, then at least one of the following is true:*

- $t_x^\circ \cong \perp^\circ$;
- $t_x^\circ = \emptyset$ (and thus $\top^\circ \cong t_y^\circ$);
- $\forall t_y \in t_y^\circ, \exists t_x \in t_x^\circ : t_x \lesssim t_y$ (i.e., $(t_y^\circ, t_x^\circ) \in \mathbf{C}_{\lesssim}^\circ$).

Proof. Let $t_x^\circ, t_y^\circ \in \mathbf{T}^\circ$.

As \top° (resp. \perp°) is a maximal (resp. minimal) element of \mathbf{T}° ,

$\top^\circ \lesssim t_y^\circ$ implies $\top^\circ \cong t_y^\circ$ (resp. $t_x^\circ \lesssim \perp^\circ$ implies $t_x^\circ \cong \perp^\circ$).

So it is sufficient to prove that one of the following is true:

i) $t_x^\circ \lesssim \perp^\circ$, ii) $t_x^\circ = \emptyset$ (and thus $\top^\circ \lesssim t_y^\circ$), or iii) $(t_y^\circ, t_x^\circ) \in \mathbf{C}_{\lesssim}^\circ$.

\mathbf{C}° is the limit the sequence (\mathbf{C}_n°) , so let $n \in \mathbb{N}$ be such that $(t_y^\circ, t_x^\circ) \in \mathbf{C}_n^\circ$ and $(t_y^\circ, t_x^\circ) \notin \mathbf{C}_{n-1}^\circ$.

We assume that $t_x^\circ \not\lesssim \perp^\circ$, $(t_y^\circ, t_x^\circ) \notin \mathbf{C}_{\lesssim}^\circ$, and that either $\top^\circ \not\lesssim t_y^\circ$ or $t_x^\circ \neq \emptyset$.

1) If $(t_y^\circ, t_x^\circ) \in \mathbf{C}_{\leq}^\circ$, then $t_x^\circ \lesssim \perp^\circ$.

This is impossible, so $(t_y^\circ, t_x^\circ) \in \mathbf{C}_{>}^\circ$ and there exists $t^\circ \in \mathbf{T}^\circ$ such that $(t_y^\circ, t^\circ) \in \mathbf{C}_{n-1}^\circ$ and $(t^\circ, t_x^\circ) \in \mathbf{C}_{n-1}^\circ$.

As $\mathbf{C}_{n-1}^\circ \subseteq \mathbf{C}^\circ$, $t_x^\circ \lesssim t^\circ$.

If $t^\circ \lesssim \perp^\circ$, then $t_x^\circ \lesssim \perp^\circ$ which is impossible.

So we know that $t^\circ \not\lesssim \perp^\circ$.

2) From proposition 3.4, we know that $\mathcal{C}_{\approx}^{\circ}$ is transitive. So either 2a) $(t^{\circ}, t_x^{\circ}) \notin \mathcal{C}_{\approx}^{\circ}$, or 2b) $(t_y^{\circ}, t^{\circ}) \notin \mathcal{C}_{\approx}^{\circ}$.

2a) In this case, consider $m \leq n - 1$ such that $(t^{\circ}, t_x^{\circ}) \in \mathcal{C}_m^{\circ}$ and $(t^{\circ}, t_x^{\circ}) \notin \mathcal{C}_{m-1}^{\circ}$.

We still have $t_x^{\circ} \not\lesssim^{\circ} \perp^{\circ}$ and we have now $(t^{\circ}, t_x^{\circ}) \notin \mathcal{C}_{\approx}^{\circ}$.

2b) In this case, consider $m \leq n - 1$ such that $(t_y^{\circ}, t^{\circ}) \in \mathcal{C}_m^{\circ}$ and $(t_y^{\circ}, t^{\circ}) \notin \mathcal{C}_{m-1}^{\circ}$.

We just showed in 1) that $t^{\circ} \not\lesssim^{\circ} \perp^{\circ}$ and we have now $(t_y^{\circ}, t^{\circ}) \notin \mathcal{C}_{\approx}^{\circ}$.

3) In both cases 2a) and 2b), the possibility of our hypothesis depend on the possibility of our hypothesis with different CUTs, and with a strictly lower natural number $m < n$.

Thus for each exploratory branch, at some point we will reach the case where (for instance for t_a° and t_b°): $(t_b^{\circ}, t_a^{\circ}) \in \mathcal{C}_0^{\circ}$, $t_a^{\circ} \not\lesssim^{\circ} \perp^{\circ}$ so $(\perp^{\circ}, t_a^{\circ}) \notin \mathcal{C}_{\perp}^{\circ}$, and $(t_b^{\circ}, t_a^{\circ}) \notin \mathcal{C}_{\approx}^{\circ}$. So $(t_b^{\circ}, t_a^{\circ}) \in \mathcal{C}_{\top}^{\circ}$.

Neither $\top^{\circ} \not\lesssim^{\circ} t_b^{\circ}$ nor $t_b^{\circ} \neq \emptyset$ is compatible with this last statement, so we highlighted a contradiction.

As a conclusion, if $t_x^{\circ} \approx^{\circ} t_y^{\circ}$, it is impossible that all of the following is false :

i) $t_x^{\circ} \lesssim^{\circ} \perp^{\circ}$, ii) $t_x^{\circ} = \emptyset$, iii) $(t_y^{\circ}, t_x^{\circ}) \in \mathcal{C}_{\approx}^{\circ}$. □

4.2 Properties of CUT Slots and Signatures

Now that the pre-order \approx° is introduced, we will see how the properties of PUTs slots and signatures are valid for CUTs slots and signatures except for specific deteriorated cases: the void CUT and absurd CUTs.

Participant Slots First, remember proposition 2.2 and the fact that the PSlot *obj* is inherited by every PUT more specific than *root(obj)*, and only those PUTs more specific than *root(obj)* have a PSlot with symbol *obj*. From the definition of PSlots of CUTs, this property is also valid for CUTs:

If $t_x^{\circ} = \{\text{fall}, \text{past}\}$ and $t_y^{\circ} = \{\text{move}\}$, then $t_x^{\circ} \approx^{\circ} t_y^{\circ}$, and in every linguistic situation in which a linguistic unit of type t_x° would be involved, there would still be something that moves. Apart for some deteriorated cases, the PSlot *obj* is inherited by every CUT more specific than the CUT $\{root(obj)\}$, and only those CUTs more specific than $\{root(obj)\}$ will have a PSlot with symbol *obj*. The following property thus holds:

Proposition 4.2. Let $\downarrow^{\circ} root(s)$ be the smallest lower set of \mathbf{T}° that contains $\{root(s)\}$:

$$\downarrow^{\circ} root(s) \stackrel{def}{=} \{t^{\circ} \in \mathbf{T}^{\circ} \mid t^{\circ} \approx^{\circ} \{root(s)\}\} \quad (10)$$

For any $s \in \mathbf{S}_{\mathcal{T}}$, the following holds:

$$\{t^{\circ} \in \mathbf{T}^{\circ} \mid s \in \omega^{\circ}(t^{\circ})\} \cup \perp^{\circ} \setminus \emptyset = \downarrow^{\circ} root(s) \setminus \emptyset \quad (11)$$

Proof. Let $s \in \mathbf{S}_{\mathcal{T}}$.

\subseteq_1 : Let $t^{\circ} \in \{t^{\circ} \in \mathbf{T}^{\circ} \mid s \in \omega^{\circ}(t^{\circ})\}$.

By definition 3.3, $\exists t \in t^{\circ}, s \in \omega(t)$. So $t^{\circ} \neq \emptyset$.

On the other hand, from definition 2.6, $t \lesssim root(s)$.

By proposition 3.6, $t^{\circ} \approx^{\circ} \{root(s)\}$ so $t^{\circ} \in \downarrow^{\circ} root(s)$.

So $t^{\circ} \in \downarrow^{\circ} root(s) \setminus \emptyset$, and $\{t^{\circ} \in \mathbf{T}^{\circ} \mid s \in \omega^{\circ}(t^{\circ})\} \subseteq \downarrow^{\circ} root(s) \setminus \emptyset$.

\subseteq_2 : Let $t^{\circ} \in \perp^{\circ} \setminus \emptyset$. $t^{\circ} \approx^{\circ} \perp^{\circ}$ which is a minimum element in \mathbf{T}° .

So $t^{\circ} \approx^{\circ} \{root(s)\}$ and $t^{\circ} \in \downarrow^{\circ} root(s)$ and $t^{\circ} \in \downarrow^{\circ} root(s) \setminus \emptyset$.

So $\perp^{\circ} \setminus \emptyset \subseteq \downarrow^{\circ} root(s) \setminus \emptyset$.

\supseteq : Let $t^\cap \in \downarrow^\cap \text{root}(s) \setminus \emptyset$. $t^\cap \lesssim \{\text{root}(s)\}$.

We use proposition 4.1, so at least one of the following is true:

- i) $t^\cap \lesssim \perp^\cap$: then $t^\cap \in \perp^\cap$. And as $t^\cap \neq \emptyset$, $t^\cap \in \perp^\cap \setminus \emptyset$.
- ii) $t^\cap = \emptyset$: impossible case.
- iii) $\exists t \in t^\cap : t \lesssim \text{root}(s)$: then using definition 2.6, $\exists t \in t^\cap : s \in \omega(t)$, and by definition 3.3, $s \in \omega^\cap(t^\cap)$. So $t^\cap \in \{t^\cap \in \mathbf{T}^\cap \mid s \in \omega^\cap(t^\cap)\}$. \square

One may have noticed that two kinds of **CUTs** are not considered here: the **CUT** \emptyset , and any absurd **CUT** $t^\cap \in \perp^\cap$:

- the **CUT** \emptyset : a problem would arise for instance with $s \in \mathbf{S}_\mathcal{T}$ such that $\text{root}(s) = \top$. By construction of the pre-order \lesssim , $\emptyset \lesssim \top$, yet \emptyset has no **PSlot** ($\omega^\cap(\emptyset) = \bigcup_{t \in \emptyset} \omega(t) = \emptyset$). So $s \notin \omega^\cap(\emptyset)$ and some **PSlots** may not be inherited from the **CUT** \emptyset ;
- any absurd **CUT** $t^\cap \in \perp^\cap$: problems arise for any $s \in \mathbf{S}_\mathcal{T}$ such that $s \notin \omega^\cap(t^\cap)$. By construction of the pre-order \lesssim , $t^\cap \lesssim \perp^\cap$, yet, from propositions 2.4 and 3.3, $\omega^\cap(\perp^\cap) = \mathbf{S}_\mathcal{T}$. So some **PSlots** are not inherited among absurd **CUTs**.

In fact, these two deteriorated cases are not considered in all of the following properties of this section.

As **CUTs** get more and more specific (i.e., as we go down the hierarchy of **CUTs**), the set of **PSlots** may only increase. **PSlots** are inherited also for **CUTs**, as long as they are not the empty conjunctive unit type, or absurd types.

Proposition 4.3. *PSlots are inherited as long as CUTs are not absurd and are not \emptyset , i.e., $\forall t_x^\cap \in \mathbf{T}^\cap \setminus \emptyset - \perp^\cap$, and $t_y^\cap \in \mathbf{T}^\cap$,*

$$t_x^\cap \lesssim t_y^\cap \Rightarrow \omega^\cap(t_y^\cap) \subseteq \omega^\cap(t_x^\cap) \quad (12)$$

Proof. Let $t_x^\cap \in \mathbf{T}^\cap \setminus \emptyset - \perp^\cap$ and $t_y^\cap \in \mathbf{T}^\cap$ such that $t_x^\cap \lesssim t_y^\cap$.

Let $s \in \omega^\cap(t_y^\cap)$. By definition 3.3, $\exists t \in t_y^\cap, s \in \omega(t)$.

We use proposition 4.1, but as $t_x^\cap \notin \perp^\cap$ and $t_x^\cap \neq \emptyset$, $t_x^\cap \lesssim t_y^\cap$ simply implies: $\forall t_y \in t_y^\cap, \exists t_x \in t_x^\cap : t_x \lesssim t$. So $\exists t_x \in t_x^\cap : t_x \lesssim t_y$.

Using proposition 2.3, $s \in \omega(t_x)$, and using definition 3.3, $s \in \omega^\cap(t_x^\cap)$. \square

Hidden Slots Next, consider the **CUT** $\{\text{rain}\}$. **PSlot** *obj* is hidden for $\{\text{rain}\}$, so is it for $\{\{\text{rain}\}\}$.

Consider any **CUT** more specific than $\{\{\text{rain}\}\}$ such as $\{\{\text{rain}\}, \{\text{past}\}\}$. $\{\{\text{rain}\}, \{\text{past}\}\}$ does not have *obj* as an actant slot. The hidden state of the **PSlot** *obj* will hence be inherited by every **CUT** more specific than any **CUT** in the set of singletons $\{\{Hiders(obj)\}\}$, and only those **CUTs** more specific than one **CUT** of $\{\{Hiders(obj)\}\}$ will have **PSlot** *obj* hidden. With the exception, of course, of the **CUT** \emptyset , and any absurd **CUT** $t^\cap \in \perp^\cap$. The following property thus holds:

Proposition 4.4. *Let $\downarrow^\cap Hiders(s)$ be the lower set of \mathbf{T}^\cap generated by $\{\{t_h\}\}_{t_h \in Hiders(s)}$, i.e.,*

$$\downarrow^\cap Hiders(s) \stackrel{\text{def}}{=} \{t^\cap \in \mathbf{T}^\cap \mid \exists t_h \in Hiders(s) : t^\cap \lesssim \{t_h\}\} \quad (13)$$

For any $s \in \mathbf{S}_\mathcal{T}$, the following holds:

$$\{t^\cap \in \mathbf{T}^\cap \mid s \in \bar{\alpha}^\cap(t^\cap)\} \cup \perp^\cap \setminus \emptyset = \downarrow^\cap Hiders(s) \setminus \emptyset \quad (14)$$

Proof. Let $s \in \mathbf{S}_{\mathcal{T}}$.

\subseteq_1 : Let $t^\cap \in \{t^\cap \in \mathbf{T}^\cap \mid s \in \bar{\alpha}^\cap(t^\cap)\}$.

a) By definition 3.4, $\exists t \in t^\cap, s \in \bar{\alpha}(t)$. So $t^\cap \neq \emptyset$.

b) On the other hand, from definition 2.7, $\exists t_h \in \text{Hiders}(s), t \lesssim t_h$. Thus using proposition 3.6 $t^\cap \hat{\lesssim} \{t_h\}$, and $t^\cap \in \downarrow^\cap \text{Hiders}(s)$.

So $t^\cap \in \downarrow^\cap \text{Hiders}(s) \setminus \emptyset$, and $\{t^\cap \in \mathbf{T}^\cap \mid s \in \bar{\alpha}^\cap(t^\cap)\} \subseteq \downarrow^\cap \text{Hiders}(s) \setminus \emptyset$.

\subseteq_2 : Let $t^\cap \in \perp^\cap \setminus \emptyset$.

$t^\cap \hat{\lesssim} \perp^\cap$ which is a minimum element in \mathbf{T}^\cap .

So $\forall t_h \in \text{Hiders}(s), t^\cap \hat{\lesssim} \{t_h\}$, i.e., $t^\cap \in \downarrow^\cap \text{Hiders}(s)$.

As we chose that t^\cap can't be \emptyset , then $t^\cap \in \downarrow^\cap \text{Hiders}(s) \setminus \emptyset$.

So $\perp^\cap \setminus \emptyset \subseteq \downarrow^\cap \text{Hiders}(s) \setminus \emptyset$.

\supseteq : Let $t^\cap \in \downarrow^\cap \text{Hiders}(s) \setminus \emptyset$.

$\exists t_h \in \text{Hiders}(s), t^\cap \hat{\lesssim} \{t_h\}$.

We use proposition 4.1, so at least one of the following is true:

i) $t^\cap \hat{\cong} \perp^\cap$: then $t^\cap \in \perp^\cap$ and $t^\cap \in \perp^\cap \setminus \emptyset$.

ii) $t^\cap = \emptyset$: impossible case.

iii) $\exists t \in t^\cap : t \lesssim t_h$: then by definition 2.7 $\exists t \in t^\cap : s \in \bar{\alpha}(t)$, and by definition 3.4, $s \in \bar{\alpha}^\cap(t^\cap)$.

So $t^\cap \in \{t^\cap \in \mathbf{T}^\cap \mid s \in \bar{\alpha}^\cap(t^\cap)\}$

So $t^\cap \in \{t^\cap \in \mathbf{T}^\cap \mid s \in \bar{\alpha}^\cap(t^\cap)\} \cup \perp^\cap \setminus \emptyset$. \square

As a direct consequence, as **CUTs** get more and more specific (i.e., as we go down the hierarchy of **CUTs**), the set of **HSlots** may only increase. **HSlots** are thus inherited also for **CUTs**, as long as they are not the empty conjunctive unit type, or absurd types:

Proposition 4.5. *Hidden slots are inherited as long as **CUTs** are not absurd and are not \emptyset , i.e., $\forall t_x^\cap \in \mathbf{T}^\cap \setminus \emptyset - \perp^\cap$ and $t_y^\cap \in \mathbf{T}^\cap$,*

$$t_x^\cap \hat{\lesssim} t_y^\cap \Rightarrow \bar{\alpha}^\cap(t_x^\cap) \subseteq \bar{\alpha}^\cap(t_y^\cap) \quad (15)$$

Proof. Let $t_x^\cap \in \mathbf{T}^\cap \setminus \emptyset - \perp^\cap$ and $t_y^\cap \in \mathbf{T}^\cap$ such that $t_x^\cap \hat{\lesssim} t_y^\cap$.

Let $s \in \bar{\alpha}^\cap(t_y^\cap)$. By definition 3.4, $\exists t \in t_y^\cap, s \in \bar{\alpha}(t)$.

From proposition 4.1 and the facts that $t_x^\cap \notin \perp^\cap$ and $t_x^\cap \neq \emptyset$, we know that $t_x^\cap \hat{\lesssim} t_y^\cap$ simply implies $\forall t_y \in t_y^\cap, \exists t_x \in t_x^\cap : t_x \lesssim t_y$. So $\exists t_x \in t_x^\cap : t_x \lesssim t$.

Using proposition 2.6, $s \in \bar{\alpha}(t_x)$, and using definition 3.4, $s \in \bar{\alpha}^\cap(t_x^\cap)$. \square

Actant Slots Furthermore, due to propositions 4.2 and 4.4, we know that apart from absurd **CUTs** and \emptyset , any **CUT** with **ASlot** obj is in lower set $\downarrow^\cap \text{root}(obj)$ and is not in lower set $\downarrow^\cap \text{Hiders}(obj)$. So the following proposition holds:

Proposition 4.6. *For any $s \in \mathbf{S}_{\mathcal{T}}$,*

$$\begin{aligned} & \{t^\cap \in \mathbf{T}^\cap \mid s \in \alpha^\cap(t^\cap)\} \cup \perp^\cap \setminus \emptyset \\ &= (\downarrow^\cap \text{root}(s) \setminus \emptyset - \downarrow^\cap \text{Hiders}(s) \setminus \emptyset) \cup \perp^\cap \setminus \emptyset \\ &= \left\{ t^\cap \in \mathbf{T}^\cap \mid t^\cap \hat{\lesssim} \{\text{root}(s)\} \text{ and } \forall t_h \in \text{Hiders}(s), t^\cap \not\hat{\lesssim} \{t_h\} \right\} \cup \perp^\cap \setminus \emptyset \end{aligned}$$

Proof. We use propositions 2.10, 4.2 and 4.4:

$$\begin{aligned}
& \{t^\circ \in \mathbf{T}^\circ \mid s \in \alpha^\circ(t^\circ)\} \cup \perp^\circ \setminus \emptyset \\
&= \{t^\circ \in \mathbf{T}^\circ \mid s \in \omega^\circ(t^\circ) - \bar{\alpha}^\circ(t^\circ)\} \cup \perp^\circ \setminus \emptyset \\
&= (\{t^\circ \in \mathbf{T}^\circ \mid s \in \omega^\circ(t^\circ)\} - \{t^\circ \in \mathbf{T}^\circ \mid s \in \bar{\alpha}^\circ(t^\circ)\}) \cup \perp^\circ \setminus \emptyset \\
&= ((\{t^\circ \in \mathbf{T}^\circ \mid s \in \omega^\circ(t^\circ)\} \cup \perp^\circ \setminus \emptyset) - (\{t^\circ \in \mathbf{T}^\circ \mid s \in \bar{\alpha}^\circ(t^\circ)\} \cup \perp^\circ \setminus \emptyset)) \cup \perp^\circ \setminus \emptyset \\
&= (\downarrow^\circ \text{root}(s) \setminus \emptyset - \downarrow^\circ \text{Hiders}(s) \setminus \emptyset) \cup \perp^\circ \setminus \emptyset
\end{aligned}$$

□

Signatures Just like for PUTs, as CUTs get more and more specific, the signature of a given common PSlot may only become more and more specific:

Proposition 4.7. For all $t_x^\circ \in \mathbf{T}^\circ \setminus \emptyset - \perp^\circ$, $t_y^\circ \in \mathbf{T}^\circ$ and $s \in \mathbf{S}_\mathcal{T}$ such that $s \in \omega^\circ(t_y^\circ)$,

$$t_x^\circ \overset{\circ}{\lesssim} t_y^\circ \Rightarrow \mathfrak{s}_{t_x^\circ}^\circ(s) \overset{\circ}{\lesssim} \mathfrak{s}_{t_y^\circ}^\circ(s) \quad (16)$$

Proof. Let $t_x^\circ \in \mathbf{T}^\circ \setminus \emptyset - \perp^\circ$ and $t_y^\circ \in \mathbf{T}^\circ$ and $s \in \mathbf{S}_\mathcal{T}$ such that $t_x^\circ \overset{\circ}{\lesssim} t_y^\circ$ and $s \in \omega^\circ(t_y^\circ)$.

1) Let $t_a \in \mathfrak{s}_{t_y^\circ}^\circ(s)$. Using definition 3.6, $\exists t_b \in t_y^\circ : s \in \omega(t_b)$ and $t_a \in \mathfrak{s}_{t_b}^\circ(s)$.

2) We use $t_x^\circ \overset{\circ}{\lesssim} t_y^\circ$ and proposition 4.1.

As $t_x^\circ \notin \perp^\circ$ and $t_x^\circ \neq \emptyset$, we know that $\forall t_y \in t_y^\circ, \exists t_x \in t_x^\circ : t_x \lesssim t_y$.

We restrict those t_y to be such that $s \in \omega(t_y)$.

We use definitions 2.9 and proposition 3.6: $\forall t_y \in t_y^\circ$ such that $s \in \omega(t_y)$, there exists $t_x \in t_x^\circ$ such that $\forall t' \in \mathfrak{s}_{t_y}^\circ(s), \exists t \in \mathfrak{s}_{t_x}^\circ(s) : t \lesssim t'$

3) This is rewritten as: $\forall t_y \in t_y^\circ$ such that $s \in \omega(t_y), \forall t' \in \mathfrak{s}_{t_y}^\circ(s), \exists t_x, t$ such that $t_x \in t_x^\circ, t \in \mathfrak{s}_{t_x}^\circ(s)$ and $t \lesssim t'$

4) combining 1) and 3) with $t_y = t_b$, and $t' = t_a$, one obtains:

$\exists t_b \in t_y^\circ$ such that $s \in \omega(t_b)$ and $t_a \in \mathfrak{s}_{t_b}^\circ(s), \exists t_x, t$ such that $t_x \in t_x^\circ, t \in \mathfrak{s}_{t_x}^\circ(s)$ and $t \lesssim t_a$

5) So there exists t and t_x such that $t_x \in t_x^\circ$ and $t \in \mathfrak{s}_{t_x}^\circ(s)$ and $t \lesssim t_a$,

6) using definition 3.6, $t \in \mathfrak{s}_{t_x^\circ}^\circ(s)$. So $\forall t_a \in \mathfrak{s}_{t_y^\circ}^\circ(s), \exists t \in \mathfrak{s}_{t_x^\circ}^\circ(s) : t \lesssim t_a$, and using proposition 3.6,

$$\mathfrak{s}_{t_x^\circ}^\circ(s) \overset{\circ}{\lesssim} \mathfrak{s}_{t_y^\circ}^\circ(s).$$

□

4.3 CUT Equivalence Class Sets

Let \cong be the natural *equivalence* relation defined by $t_x^\circ \cong t_y^\circ \Leftrightarrow (t_x^\circ \lesssim t_y^\circ \wedge t_y^\circ \lesssim t_x^\circ)$. The set of equivalence classes defines a partition of \mathbf{T}° . Let $t^\circ \in \mathbf{T}^\circ$, we denote $[t^\circ] \stackrel{\text{def}}{=} \{t_x^\circ \in \mathbf{T}^\circ \mid t_x^\circ \cong t^\circ\}$ the equivalence class to which t° belongs. We will usually use the notation t^\square for an unknown equivalence class.

Definition 4.1 (CUTs equivalence class set). The *CUTs equivalence class set* \mathbf{T}^\square is the quotient set of \mathbf{T}° by \cong , i.e., $\mathbf{T}^\square \stackrel{\text{def}}{=} \mathbf{T}^\circ / \cong = \{[t^\circ] \mid t^\circ \in \mathbf{T}^\circ\}$. We define a partial order \leq over \mathbf{T}^\square with $[t_x^\circ] \leq [t_y^\circ]$ if and only if $t_x^\circ \lesssim t_y^\circ$.

We define $\top^\square \stackrel{\text{def}}{=} [\top^\circ]$ the top (less specific) *CUTs* equivalence class set, and $\perp^\square \stackrel{\text{def}}{=} [\perp^\circ]$ the absurd (most specific) *CUTs* equivalence class set.

Proposition 4.8. \top^\square and \perp^\square are respectively the greatest element and the least element of \mathbf{T}^\square .

Proof. \top^\square : From proposition 3.8, \top° is a maximal element for \lesssim .

So $\forall t^\circ \in \mathbf{T}^\circ, t^\circ \lesssim \top^\circ$. So $\forall t^\square \in \mathbf{T}^\square, t^\square \leq \top^\square$, and \top^\square is the greatest element of \mathbf{T}^\square .

\perp^\square : From proposition 3.8, \perp° is a minimal element for \lesssim . So $\forall t^\circ \in \mathbf{T}^\circ, \perp^\circ \lesssim t^\circ$.

So $\forall t^\square \in \mathbf{T}^\square, \perp^\square \leq t^\square$, so \perp^\square is the least element of \mathbf{T}^\square . □

A set of equivalence class *CUTs* is closed under the union operation:

Proposition 4.9. Let $t^\square \in \mathbf{T}^\square$, and $t_x^\circ, t_y^\circ \in t^\square$. Then $t_x^\circ \cup t_y^\circ \in t^\square$.

Proof. Let $t^\square \in \mathbf{T}^\square$, and $t_x^\circ, t_y^\circ \in t^\square$. We will prove that $t_x^\circ \cup t_y^\circ \cong t_x^\circ$.

\lesssim : We know from proposition 3.7 that $t_x^\circ \subseteq t_x^\circ \cup t_y^\circ$ implies $t_x^\circ \cup t_y^\circ \lesssim t_x^\circ$.

\gtrsim : We know that $t_x^\circ \gtrsim t_y^\circ$. From proposition 3.6, at least one of the followings holds:

i) $t_x^\circ \lesssim \perp^\circ$: then $t_x^\circ \cup t_y^\circ \cong t_x^\circ \cong \perp^\circ$;

ii) $t_x^\circ = \emptyset$: then $t_x^\circ \cup t_y^\circ = t_y^\circ \cong t_x^\circ$;

iii) $\forall t_y \in t_y^\circ, \exists t_x \in t_x^\circ : t_x \lesssim t_y$: So $\forall t \in t_x^\circ \cup t_y^\circ, \exists t_x \in t_x^\circ : t_x \lesssim t$, and $t_x^\circ \gtrsim t_x^\circ \cup t_y^\circ$. □

The following lemma is useful to prove upcoming propositions. It states that adding a *PUT* t to a *CUT* t° does not make it change its equivalence class, provided that t is greater than at least one *PUT* in t° :

Lemma 4.10. Let $t^\circ \in \mathbf{T}^\circ$ and $t \in \mathbf{T}$ such that $\exists t' \in t^\circ, t' \lesssim t$. Then $t^\circ \cup \{t\} \cong t^\circ$.

Proof. Let $t^\circ \in \mathbf{T}^\circ$ and $t \in \mathbf{T}$ such that $\forall t' \in t^\circ, t' \lesssim t$.

\lesssim : $t_y^\circ \subseteq t_y^\circ \cup \{t\}$ so $t^\circ \cup \{t\} \lesssim t^\circ$;

\gtrsim : $t^\circ \neq \emptyset$. Let $t' \in t^\circ \cup \{t\}$.

If $t' \in t^\circ$, then there exists $t'' (= t') \in t^\circ : t'' \lesssim t'$.

If $t' = t$, then there exists $t'' \in t^\circ : t'' \lesssim t'$.

So $\forall t' \in t^\circ \cup \{t\}, \exists t'' \in t^\circ : t'' \lesssim t'$, and $t^\circ \cup \{t\} \gtrsim t^\circ$. □

4.4 Maximal CUTs

CUTs enable a first inference mechanism, which is type inference. If a **CUT** t_x^\square is lower than t_y^\square , then any instance of t_x^\square is also an instance of every **PUT** in t_y^\square . In this section we consider a remarkable subset of **CUTs** that we call the set of maximal **CUTs**.

Let us first introduce a maximization operator \uparrow that associates to any **CUT** t^\square the union of all **CUTs** in $[t^\square]$:

Definition 4.2 (Maximization operator). The maximization operator \uparrow defined on \mathbf{T}^\square associates to every **CUT** t^\square the **CUT** $\uparrow t^\square \stackrel{\text{def}}{=} \bigcup_{t^\square \in [t^\square]} t^\square$.

The set of maximal **CUTs** \mathbf{T}_{max}^\square is then denoted \mathbf{T}_{max}^\square , and defined as follows:

$$\mathbf{T}_{max}^\square = \{\uparrow t^\square \mid t^\square \in \mathbf{T}^\square\} \quad (17)$$

In the rest of this section we will list useful facts about the maximization operator \uparrow and the set of maximal **CUTs** \mathbf{T}_{max}^\square .

First, the maximization operator preserves the equivalence class. This means that a **CUT** t^\square and its maximized **CUT** $\uparrow t^\square$ are both in the same equivalence class set.

Proposition 4.11. *The restriction of the closure operator \uparrow to any **CUTs** equivalence class set is an endomorphism of this class, i.e., for all $t^\square \in \mathbf{T}^\square$, if $t^\square \in t^\square$, then $\uparrow t^\square \in t^\square$. Or equivalently, $\forall t^\square \in \mathbf{T}^\square, \uparrow t^\square \simeq t^\square$.*

Proof. Let $t^\square \in \mathbf{T}^\square$, and $t^\square \in t^\square$. By definition, $\uparrow t^\square$ is the union of all **CUT** in t^\square . We know from proposition 4.9 that t^\square is closed under the union operation, so $\uparrow t^\square \in t^\square$. \square

Next, all **CUTs** of a given equivalence class set have the same maximized **CUTs**.

Proposition 4.12. *The kernel of the closure operator \uparrow on \mathbf{T} is the equivalence relation \simeq , i.e., if $t_x^\square \simeq t_y^\square$, then $\uparrow t_x^\square = \uparrow t_y^\square$*

Proof. By definition, if $t_x^\square \simeq t_y^\square$, then $[t_x^\square] = [t_y^\square]$, and $\uparrow t_x^\square = \uparrow t_y^\square$ \square

Thus, there is a one-to-one correspondence between the set of a set of equivalence **CUTs** class sets \mathbf{T}^\square and the set of maximal **CUTs** \mathbf{T}_{max}^\square . Moreover, **PUTs** is a maximal **CUT** in the sense that among its equivalence class set, it is the unique **CUT** that has the maximal cardinality.

Proposition 4.13. *The closure operator \uparrow is order-embedding, i.e.,*

$$t_x^\square \lesssim t_y^\square \Leftrightarrow \uparrow t_x^\square \lesssim \uparrow t_y^\square \quad (18)$$

Proof. Let $t_x^\square, t_y^\square \in \mathbf{T}^\square$. From proposition 4.11 we know that $t_x^\square \simeq \uparrow t_x^\square$ and $t_y^\square \simeq \uparrow t_y^\square$.

\Rightarrow : If $t_x^\square \lesssim t_y^\square$, then $t_x^\square \simeq \uparrow t_x^\square \lesssim \uparrow t_y^\square \simeq t_y^\square$. So $\uparrow t_x^\square \lesssim \uparrow t_y^\square$.

\Leftarrow : If $\uparrow t_x^\square \lesssim \uparrow t_y^\square$, then $\uparrow t_x^\square \simeq t_x^\square \lesssim t_y^\square \simeq \uparrow t_y^\square$. So $t_x^\square \lesssim t_y^\square$. \square

Let $t^\square \in \mathbf{T}^\square$, we denote $\uparrow t^\square$ the upper-set of \mathbf{T} generated by **PUTs** in t^\square , i.e.,

$$\uparrow t^\square \stackrel{\text{def}}{=} \{t \in \mathbf{T} \mid \exists t' \in t^\square, t' \lesssim t\} \quad (19)$$

Proposition 4.14. *The maximization operator \uparrow is such that all of the following is true:*

- if $t^\square \in \mathbf{T}^\square \setminus \emptyset - \perp^\square$, then $\uparrow t^\square = \uparrow t^\square$;
- if $t^\square \in \perp^\square$, then $\uparrow t^\square = \mathbf{T}$;

- $\uparrow \emptyset = \uparrow \top^\sim$;

Proof. Let $t^\circ \in \mathbf{T}^\circ$:

1) if $t^\circ \in \mathbf{T}^\circ \setminus \emptyset - \perp^\circ$:

\subseteq : Let $t \in \uparrow t^\circ$. By definition, $\exists t_x^\circ \in [t^\circ], t \in t_x^\circ$. So $t^\circ \cong t_x^\circ$ and $t^\circ \lesssim t_x^\circ$.

From proposition 3.6, at least one of the followings holds:

- i) $t^\circ \lesssim \perp^\circ$ (impossible from our hypothesis);
- ii) $t^\circ = \emptyset$ (impossible from our hypothesis);
- iii) $\forall t_x \in t_x^\circ, \exists t' \in t^\circ : t' \lesssim t_x$. So for $t \in t_x^\circ, \exists t' \in t^\circ : t' \lesssim t$. And thus $t \in \uparrow t^\circ$.

\supseteq : Let $t \in \uparrow t^\circ$. Then $\exists t' \in t^\circ, t' \lesssim t$. From lemma 4.10, $t^\circ \cup \{t\} \cong t^\circ$, and thus $t^\circ \cup \{t\} \in [t^\circ]$, and thus $t \in \uparrow t^\circ$.

2) if $t^\circ \in \perp^\circ$, then $\mathbf{T} \in [t^\circ]$ and $\uparrow t^\circ = \mathbf{T}$;

3) if $\emptyset \cong \top^\circ$, then from proposition 4.12, $\uparrow \emptyset = \uparrow \top^\circ$. □

Figure 5 illustrates the most interesting case in proposition 4.14 which is the case where $t^\circ \notin \perp^\circ$ and $t^\circ \neq \emptyset$: black dots represent PUTs of $t^\circ \in \mathbf{T}^\circ$, and the fact that a dot is lower than another dot roughly means that the former is more specific than the latter. Circles represent equivalence classes. If t° is a CUT consisting of all the black dots, then the gray zone represents the set of PUTs that belong to $\uparrow t^\circ$.

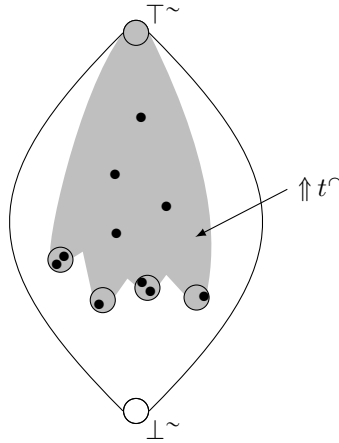


Figure 5: Illustration of a maximal CUT $\uparrow t^\circ$ with $t^\circ \in \mathbf{T}^\circ \setminus \emptyset - \perp^\circ$.

A CUT is thus maximal if and only if: either it is the whole set of PUTs, or it is a non-absurd non-empty upper set of PUTs:

Proposition 4.15. A CUT t° is maximal if and only if one of the following is true:

- $t^\circ = \mathbf{T}$,
- $t^\circ \in \mathbf{T}^\circ \setminus \emptyset - \perp^\circ$, and t° is a upper set of \mathbf{T} .

Proof. Let us prove both implications:

\Rightarrow : If t° is maximal, then there exists $t^{\circ'} \in \mathbf{T}^\circ, t^\circ = \uparrow t^{\circ'}$.

- i) if $t^{\circ'} \in \perp^\circ$, then $t^\circ = \mathbf{T}$;
- ii) if $t^{\circ'} = \emptyset$, then $t^\circ = \uparrow \perp^\circ$, and $\uparrow \perp^\circ$ is either \mathbf{T} or a non-empty upper set of \mathbf{T} ;
- iii) if $t^{\circ'} \in \mathbf{T}^\circ \setminus \emptyset - \perp^\circ$, then t° is an non-empty upper set of \mathbf{T} .

\Leftarrow :

- i) if $t^\circ = \mathbf{T}$, then for all $t^{\circ'} \in \perp^\circ$, $t^\circ = \uparrow t^{\circ'}$ so t° is maximal;
- ii) if $t^\circ \notin \perp^\circ$, and t° is a non-empty upper set of \mathbf{T} , then using propositions 4.12 and 4.11, $\uparrow t^\circ = t^\circ$, so t° is maximal. \square

Remark. Anyways, as \mathbf{T} is an upper set of \mathbf{T} , every maximal CUT is an upper set of \mathbf{T} .

The last result of this section is the fact the operator \uparrow defines a closure operator on \mathbf{T} .

Proposition 4.16. \uparrow is a closure operator on \mathbf{T} , i.e., it satisfies the following conditions for all CUTs $t_x^\circ, t_y^\circ \in \mathbf{T}^\circ$:

- $t_x^\circ \subseteq \uparrow t_x^\circ$, (\uparrow is extensive);
- $t_x^\circ \subseteq t_y^\circ \implies \uparrow t_x^\circ \subseteq \uparrow t_y^\circ$, (\uparrow is increasing with respect to \subseteq);
- $\uparrow \uparrow t_x^\circ = \uparrow t_x^\circ$, (\uparrow is idempotent);

Proof. 1) *Extensive:* Let $t^\circ \in \mathbf{T}^\circ$. $t^\circ \in [t^\circ]$ so from definition 4.2, $t^\circ \subseteq \uparrow t^\circ$, and \uparrow is extensive.

2) *Increasing w.r.t. \subseteq :* Let $t_x^\circ, t_y^\circ \in \mathbf{T}^\circ$ such that $t_x^\circ \subseteq t_y^\circ$.

First, we know from proposition 3.7 that $t_y^\circ \lesssim t_x^\circ$.

i) If $t_y^\circ \in \perp^\circ$, then $\uparrow t_y^\circ = \mathbf{T}$ (prop. 4.14) and $\uparrow t_x^\circ \subseteq \uparrow t_y^\circ$.

ii) If $t_x^\circ \in \perp^\circ$, then so is t_y° , and $\uparrow t_x^\circ = \uparrow t_y^\circ = \mathbf{T}$ (prop. 4.14).

iii) If $t_y^\circ = \emptyset$, then $t_x^\circ = t_y^\circ = \emptyset$, and $\uparrow t_x^\circ = \uparrow t_y^\circ$.

iv) If $t_y^\circ \notin \perp^\circ$, $t_y^\circ \neq \emptyset$, $t_x^\circ \notin \perp^\circ$, and $t_x^\circ \neq \emptyset$: Let $t \in \uparrow t_x^\circ = \uparrow t_y^\circ$. We know that $\exists t' \in t_y^\circ, t' \lesssim t$.

As $t_x^\circ \subseteq t_y^\circ$, then $\exists t' \in t_y^\circ, t' \lesssim t$, and $t' \in \uparrow t_y^\circ = \uparrow t_x^\circ$.

v) If $t_x^\circ = \emptyset$, then $\uparrow t_x^\circ = \uparrow \top^\circ$. \top is a maximal element of \mathbf{T} , so using lemma 4.10, $t_y^\circ \simeq t_y^\circ \cup \top^\circ$, and from definition 4.2, $\uparrow t_y^\circ = \uparrow t_y^\circ \cup \top^\circ$. Using items i) to iv), we know that $\uparrow t_x^\circ = \uparrow \top^\circ \subseteq \uparrow t_y^\circ \cup \top^\circ = \uparrow t_y^\circ$.

So $\uparrow t_x^\circ \subseteq \uparrow t_y^\circ$, and \uparrow is increasing.

3) *Idempotent:* Let $t^\circ \in \mathbf{T}^\circ$.

i) if $t^\circ \in \perp^\circ$, then $\uparrow t^\circ = \mathbf{T}$, and as we know that $\mathbf{T} \in \perp^\circ$, $\uparrow \uparrow t^\circ = \uparrow t^\circ = \mathbf{T}$.

ii) if $t^\circ \notin \perp^\circ$ and $t^\circ \neq \emptyset$, then:

\supseteq : We know from extensivity of \uparrow that $\uparrow t^\circ \subseteq \uparrow \uparrow t^\circ$.

\subseteq : Let $t \in \uparrow \uparrow t^\circ$. $\uparrow t^\circ \simeq t^\circ$ so (prop: 4.9 $\uparrow t^\circ \notin \perp^\circ$, and we also know that $t^\circ \subseteq \uparrow t^\circ$ so $\uparrow t^\circ \neq \emptyset$). Using upper-set properties: $\uparrow \uparrow t^\circ = \uparrow \uparrow t^\circ = \uparrow \uparrow t^\circ = \uparrow t^\circ = \uparrow t^\circ$.

iii) if $t^\circ = \emptyset$, then $\uparrow t^\circ = \uparrow \perp^\circ$, and using ii), $\uparrow t^\circ = \uparrow \perp^\circ = \uparrow \uparrow \perp^\circ = \uparrow \uparrow t^\circ$. \square

Thus a maximal CUTs t_\uparrow° is also maximal in the sense that any CUT of its equivalence class is contained in t_\uparrow° .

4.5 Concise CUTs

Concise **CUTs** will be of special interest in the following of this work. They are those (non-unique) **CUTs** that have a minimal cardinality in a given equivalence class:

Definition 4.3 (Concise **CUT**). Let t^\cap be a **CUT**. t^\cap is said to be *concise* if and only if:

- if t^\cap is absurd, then $t^\cap = \perp^\cap$.
- else, $\forall t \in t^\cap, \nexists t' \in t^\cap$ such that $t' < t$ and $\forall t, t' \in t^\cap, t \neq t'$.

Figure 6 below illustrates definition 4.3: black and white dots represent **PUTs**, and the fact that a dot is lower than another dot roughly means that the former is more specific than the latter. Circles represent equivalence classes. If t^\cap is a **CUT** consisting of all the black and white dots, then the set of black dots represents a concise **CUT** that is a subset of t^\cap . In the illustrated case, there are $2 \times 1 \times 2 \times 1$ such concise **CUT** that are subsets of t^\cap .

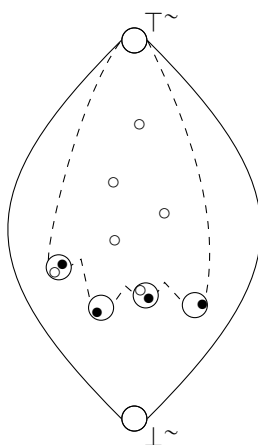


Figure 6: Let the union of black and white dots represent a **CUT** t^\cap . The set of black dots represents a concise **CUT** that is a subset of t^\cap .

Part III

Unit Graphs

In the previous part we introduced the types that nodes of any **Unit Graph (UG)** have, and that specify through slots and signatures how these nodes must be linked to other nodes in the graph. Now is time to introduce the main objects of the Unit Graphs mathematical framework: the Unit Graphs, such as illustrated in figures 1 and 7.

5 Unit Graphs (UGs)

Unit types specify how units *must* be linked to other nodes in the **UGs**, but units *may* also be linked to other units through circumstantial dependencies. Circumstantial dependency symbols are first introduced in section 5.1.

Then section 8.3 introduce the **UG** support and the **UGs** themselves in section 8.3.

5.1 Circumstantial Dependency Symbols Hierarchy

Unit types specify how units are linked to other nodes in the **Unit Graphs (UGs)**. As for any slot in a predicate, one **ASlot** of a unit may be filled by only one unit at a time. Now, one may also encounter dependencies of another type in some utterance representations: circumstantial dependencies (Mel'čuk, 2004a). Circumstantial dependency relations are instance-instance relation contrary to participation relations that are predicate-argument relations. Such dependencies may be optional or multiple (i.e., a unit may govern zero or more other units through relations of a same type). Example of such relations are the deep syntactic representation relations **ATTR**, **COORD**, **APPEND** such as in figure 7. But we may also use such relations to represent the link between a lexical unit and its sense for instance.

We use symbols to distinguish the different kinds of circumstantial dependencies, and thus introduce a finite set of **Circumstantial Dependency Symbols (CSymbols)**.

Definition 5.1 (Circumstantial Dependency Symbols Set). A **CSymbols** set is a set of binary relation symbols denoted \mathcal{S}_C .

CSymbols are often classified in sets and subsets, we thus take into account this need classification and hierarchy, and introduce a partial order over the set of **CSymbols**.

Definition 5.2 (Pre-order over \mathcal{S}_C). The **CSymbols** set is pre-ordered by a relation \lesssim , which is induced by a set of *asserted comparisons* $\mathcal{C}_{\mathcal{S}_C} \subseteq \mathcal{T}^\cap$. $(\mathcal{S}_C, \mathcal{C}_{\mathcal{S}_C})$ is a directed graph on \mathcal{S}_C . The pre-order \lesssim is equal to $\mathcal{C}_{\mathcal{S}_C}^*$ the reflexo-transitive closure of $\mathcal{C}_{\mathcal{S}_C}$, i.e., $\forall s, s' \in \mathcal{S}_C, s' \lesssim s$ iff $(s, s') \in \mathcal{C}_{\mathcal{S}_C}^*$, i.e., s' is a descendant of s , and s is an ascendant of s' .

Every **CSymbol** is assigned a signature that specifies what kind of units may be linked through a relation having this symbol.

Definition 5.3 (Signature of **CSymbols**). The set of signatures of **CSymbols** $\{\sigma_s\}_{s \in \mathcal{S}_C}$ is a set of couples in \mathcal{T}^\cap . For every dependency relation symbol s , $\sigma_s \stackrel{\text{def}}{=} (\text{domain}(s), \text{range}(s))$. The set of signatures $\{\sigma_s\}_{s \in \mathcal{S}_C}$ must be such that for all $s_1, s_2 \in \mathcal{S}_C$ such that $s_1 \lesssim s_2$, $\text{domain}(s_1) \lesssim \text{domain}(s_2)$ and $\text{range}(s_1) \lesssim \text{range}(s_2)$.

We finally introduce the full hierarchy of **CSymbols**.

Definition 5.4 (CSymbols hierarchy). A *Circumstantial Dependency Symbols (CSymbols) hierarchy* $\mathcal{C} \stackrel{\text{def}}{=} (\mathcal{S}_{\mathcal{C}}, \mathcal{C}_{\mathcal{S}_{\mathcal{C}}}, \mathcal{T}^{\cap}, \{\sigma_s\}_{s \in \mathcal{S}_{\mathcal{C}}})$, is composed of:

- $\mathcal{S}_{\mathcal{C}}$ a set of **Circumstantial Dependency Symbols (CSymbols)**;
- $\mathcal{C}_{\mathcal{S}_{\mathcal{C}}}$ a set of asserted comparisons;
- $\mathcal{T}^{\cap} = (T_{\text{declared}}, C_{\text{asserted}}, \perp_{\text{asserted}}, \mathcal{S}_{\mathcal{T}}, \text{root}, \text{Hiders}, \{\mathfrak{S}_t\}_{t \in \mathcal{T}})$ a **Conjunctive Unit Types (CUTs)** hierarchy;
- $\{\sigma_s\}_{s \in \mathcal{S}_{\mathcal{C}}}$ a set of signatures of the **CSymbols**;

5.2 Definition of UGs

The **Unit Graphs (UGs)** will enable the description of utterance representation at different representation levels. Parallel with the **Conceptual Graphs (CGs)**, **UGs** are defined over a so-called support, which is composed of a **CUTs** hierarchy, a **CSymbols** hierarchy, and a set of unit markers.

Definition 5.5 (Support). A *Support* is a tuple $\mathcal{S} \stackrel{\text{def}}{=} (\mathcal{T}^{\cap}, \mathcal{C}, \mathcal{M})$ where:

- \mathcal{T}^{\cap} is a **Conjunctive Unit Types (CUTs) hierarchy**;
- \mathcal{C} is a **Circumstantial Dependency Symbols (CSymbols) hierarchy**;
- \mathcal{M} is the set of so-called *unit markers*. $\mathcal{M}^{\cap} = 2^{\mathcal{M}}$ is the set of unit node markers, with \emptyset the *generic unit node marker*.

Let us precise what we mean by unit markers. We establish a distinction between:

- units, that are the objects of the represented domain;
- unit markers, that are chosen in the set \mathcal{M} , and that each identify a specific unit;
- unit nodes, that are interconnected in **UGs** and that each represent a unit;
- unit nodes markers, that are chosen in the powerset of \mathcal{M} which is denoted \mathcal{M}^{\cap} : $\mathcal{M}^{\cap} \stackrel{\text{def}}{=} 2^{\mathcal{M}}$, and that label unit nodes so as to specify what unit each unit node represents.

This may seem a little bit complex on first sight, but it is actually an extension for the **MTT** that enable to be close to the **CGs**, and a simple articulation with the semantic web formalisms. In fact, to every marker will correspond a URI. If a unit node is labeled by the generic unit node marker \emptyset (we say it is a generic unit node), then the represented unit is unknown, it will be written as a blank node in RDF. In the literature of **MTT**, we consider that every unit nodes of utterance representations is generic. On the other hand, if a unit node is marked $\{m_1, m_2\}$, then the unit markers m_1 and m_2 actually identify the same unit, their corresponding RDF resource will be linked by a relation `owl:sameas`. This slightly differs from the **CGs** because the set of unit node markers is the powerset of unit markers, and we do not introduce the classic generic marker `*` from the **CGs**. We chose to do so because we want to define semantic preserving operations over the set of **UGs**, hence when we will merge two asserted equivalent individual unit nodes, we need to keep record that their unit markers are all equivalent in the **UG**. This is only possible if a unit node may have multiple markers.

In their simple version, the **CGs** have an equivalence relation over concept nodes named *coreference*. As this relation does not correspond to the linguistic term and that we will represent linguistic coreference differently, we deactivate ambiguity and use term *declare unit node*

equivalence relation, denoted Eq . Two unit nodes declared equivalent represent the same unit. Moreover, unlike the **CGs** *coref* relation, the Eq relation is not an equivalence relation over unit nodes, i.e., it is not a reflexive, symmetric, and transitive relation. This enables to distinguish between explicit and implicit knowledge, thus easing the articulation with semantic web formalisms.

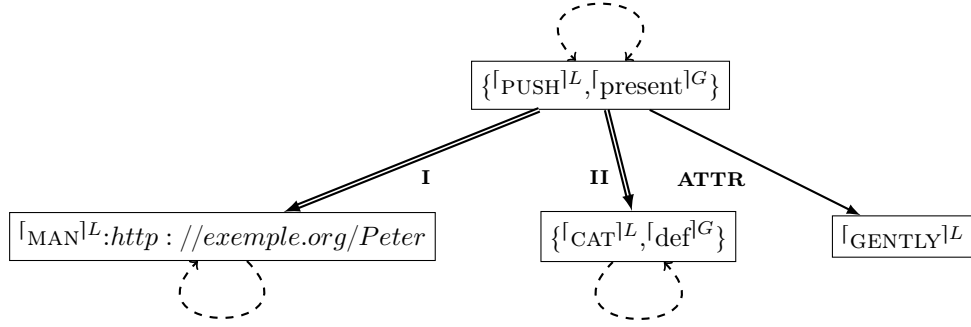


Figure 7: Example of a **UG**: Deep syntactic representation of sentence *Peter gently pushes the cat*.

The **UGs** enable to represent utterances at different representation levels. We already met a semantic representation on figure 1, and figure 7 illustrates a deep syntactic representation. The natural graph representation we use is a finite oriented labelled multigraph, composed of unit nodes, participations, and circumstantial dependencies, plus a set of asserted unit node equivalences. A unit node is labelled by a couple composed of a **CUT** that specifies the nature of the unit that is represented, and a unit node marker that enables to identify the represented unit. If the represented unit is unknown, then the unit node may have the generic unit node marker \emptyset . Every dependency arc is labelled by a symbol in $\mathcal{S}_{\mathcal{T}} \cup \mathcal{S}_{\mathcal{C}}$ that specifies the nature of the link that exists between the unit nodes that this arc connects. Dependency arcs and their symbols are defined in terms of so-called triples. A **UG** is thus a combination of interconnected unit nodes defined over a given support.

Definition 5.6 (UG). The set of **UGs** defined over a support \mathcal{S} is denoted $\mathcal{G}(\mathcal{S})$, and each **UG** $G \in \mathcal{G}(\mathcal{S})$ is a tuple $G \stackrel{\text{def}}{=} (U, \mathbf{l}, P, C, Eq)$ where:

- U is the set of *unit nodes*. They are illustrated by rectangles as on figure 7.
- \mathbf{l} is a labelling mapping over U that assigns to each unit node $u \in U$ a couple $\mathbf{l}(u) = (t^\wedge, m^\wedge) \in \mathbf{T}^\wedge \times \mathbf{M}^\wedge$. u is thus a unit node that has type t^\wedge and marker m^\wedge . We denote $t^\wedge = \text{type}(u)$ and $m^\wedge = \text{marker}(u)$. On figure 7, unit nodes are all typed by singletons, except on that is typed $\{ [\text{PUSH}]^L, [\text{present}]^G \}$. Moreover, unit nodes are all generic, except one marked $\{ \text{http} : // \text{exemple.org} / \text{Peter} \}$.
- P is the set of *participation triples* $(u, r, v) \in U \times \mathcal{S}_{\mathcal{T}} \times U$. For all $p = (u, r, v) \in P$, the unit represented by v fills the **PSlot** r of the unit represented by u . We denote $u = \text{governor}(p)$, $r = \text{symbol}(p)$ and $v = \text{participant}(p)$. We also denote $\text{arc}(p) = (u, v)$. They are illustrated by double arrows on figure 7.
- C is the set of *circumstantial dependency triples* $(u, r, v) \in U \times \mathcal{S}_{\mathcal{C}} \times U$. For all $c = (u, r, v) \in C$, the unit represented by u governs the unit represented by v with respect to r .

Conversely, the unit represented by v depends on the unit represented by u with respect to r . We denote $u = \text{governor}(c)$, $r = \text{symbol}(c)$ and $v = \text{circumstant}(c)$. We also denote $\text{arc}(c) = (u, v)$. They are illustrated by simple arrows on figure 7.

- $Eq \subseteq U^2$ is the set of so-called *asserted unit node equivalences*. For all $(u_1, u_2) \in U^2$, $(u_1, u_2) \in Eq$ means that u_1 and u_2 represent the same unit. They are illustrated by dashed arrows on figure 7.
- The *underlying graph* of G denoted $\text{graph}(G)$ is a finite, oriented, labelled multigraph where:
 - for all $u \in U$, u is a node of $\text{graph}(G)$ labelled by $l(u)$;
 - for all $p \in P$, $\text{arc}(p)$ is an arc of $\text{graph}(G)$ labelled by $\text{symbol}(p)$;
 - for all $c \in C$, $\text{arc}(c)$ is an arc of $\text{graph}(G)$ labelled by $\text{symbol}(c)$;
 - for all $(u_1, u_2) \in Eq$, (u_1, u_2) is an arc of $\text{graph}(G)$ labelled $=$;

Again, we draw the attention of the reader on the fact that we distinguish units, unit markers, and unit nodes. Several mechanisms may imply that two unit nodes u_1, u_2 represent the same unit, or that two unit markers m_1, m_2 identify the same unit. For **UGs** as defined above, we identify the following mechanisms:

- u_1, u_2 are asserted to be equivalent, i.e., $(u_1, u_2) \in Eq$. Thus all of the markers in $\text{marker}(u_1) \cup \text{marker}(u_2)$ identify the same unit.
- as a unit represents a predicate, only one unit may fill each of its **PSlot**. Hence if (u, r, v_1) and (u, r, v_2) both are in P , then v_1 and v_2 represent the same unit.
- after merging unit nodes that represent the same unit, new couple of unit nodes may appear to represent the same unit according to previous item.

UGs so defined are base objects of the **UGs** mathematical framework, with which one may formalize among others:

- utterance representations at different representation levels;
- semantic decompositions of lexical units (for lexicographic definitions);
- premises and conclusions of linguistic and grammatical rules;

5.3 Graphical Representation

The graphical representation of a **UG** is a drawing of the underlying finite, oriented, labelled multigraph. A unit node is represented by a rectangle with its label written inside in the form: *type*:"*marker*", with the following supplementary rules chosen to ease the reading:

- if the type or the maker is a singleton, brackets may be avoided;
- if the marker is the generic marker (the empty set), it may be avoided along with its semi-column;
- if the type is the empty set, it may be avoided, but the semi-column must be written;
- if the type and the marker are the empty sets, there may be nothing written in the nodes.

Figure 8 below illustrates these rules. All the unit nodes of each line share the same label.

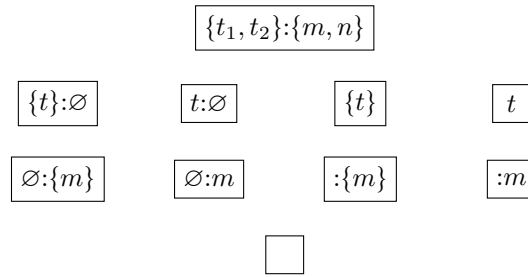


Figure 8: Graphical Representation of unit nodes and their labels: All the unit nodes of each line share the same label.

We also use the following conventions to distinguish the different relations:

- participation dependencies are drawn with a double arrow. If a participation dependency is known to be hidden, then it may be drawn with a double dashed arrow.
- circumstantial dependencies are drawn with a single arrow.
- *Eq* relations are drawn with a dashed line. The label may not be written. If the relation is symmetric, then the arc is non-oriented.

Figure 9 below illustrates these conventions.

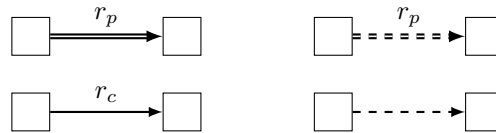


Figure 9: Graphical Representation of arcs and their labels. Top left: a participation; right: a hidden participation. Bottom left: a circumstantial dependency; right: an asserted unit nodes equivalence.

5.4 Explicit Support Compliance

In a **UG** G , dependency relations and unit nodes may be explicitly compliant with the support over which G is defined, by respecting **PSlots** or signatures for instance.

Let us underline that if a unit node or a dependency relation is not explicitly compliant with the support, this is not necessarily wrong: it only means that some knowledge is not explicit. We thus make the open-world assumption, which means that a **UG** along with the support on which it is defined represents explicit knowledge, and that additional knowledge may be inferred. As we shall see, this assumption eases the compatibility between the **UGs** mathematical framework and the **Semantic Web Formalisms (SWFs)**.

First, any participation relation incident from a unit node should represent a **PSlot** of the type of this unit node.

Definition 5.7 (Explicit support compliance of a unit node). In a **UG** $(U, \mathbf{l}, P, C, Eq) \in \mathcal{G}(\mathcal{S})$, a unit node u is said to *explicitly comply with the support* \mathcal{S} if and only if: for all $p \in P$ such that $\text{governor}(p) = u$, $\text{symbol}(p) \in \omega^\wedge(\text{type}(u))$.

Then, for a participation relation triple, the participation symbol must be a **PSlot** of the governor's type, and the participant's type should explicitly comply with the signature.

Definition 5.8 (Explicit support compliance of a participation triple). In a **UG** $(U, \mathbf{l}, P, C, Eq) \in \mathcal{G}(\mathcal{S})$, a participation triple $(u, r, v) \in P$ is said to *explicitly comply with the support* \mathcal{S} if and only if: $r \in \omega^\wedge(\text{type}(u))$ and $\zeta_{\text{type}(u)}^\wedge(r) \subseteq \text{type}(v)$.

Finally for a circumstantial dependency relation, the type of the governor and the dependent should explicitly comply with the signature of the symbol.

Definition 5.9 (Explicit support compliance of a circumstantial dependency triple). In a **UG** $(U, \mathbf{l}, P, C, Eq) \in \mathcal{G}(\mathcal{S})$, a circumstantial dependency triple $(u, r, v) \in P$ is said to *explicitly comply with the support* \mathcal{S} if and only if: $\text{domain}(r) \subseteq \text{type}(u)$ and $\text{range}(r) \subseteq \text{type}(v)$.

A **UG** is thus explicitly compliant with the support on which it is defined if and only if all its dependency arcs are explicitly compliant on the support.

Definition 5.10 (Explicit support compliance of a **UG**). A **UG** $G \in \mathcal{G}(\mathcal{S})$ is said to *explicitly comply with the support* \mathcal{S} if and only if all of its dependency arcs $d \in P \cup C$ is explicitly compliant with the \mathcal{S} .

5.5 Semantics of UGs: Reasoning using First-Order Implications

More generally, a **UG** G defined on a support \mathcal{S} represents explicit knowledge, and other knowledge is contained in \mathcal{S} . In this section, we will define how one can explicit in G all the knowledge that may be inferred from G and \mathcal{S} , thus defining the semantics of **UGs**.

5.5.1 Saturation

We first present an axiomatization of the **UGs** Semantics in the form of first-order implications. These rules are to be used to add inferable knowledge to a **UG** in the following way: apply a rule each time you can until you can't apply any more.

	For all	Ensure that
u-typ	$u \in U$	$\uparrow \text{type}(u) \subseteq \text{type}(u)$
eq-ref	$u \in U$	$(u, u) \in Eq$
eq-sym	$(u_1, u_2) \in Eq$	$(u_2, u_1) \in Eq$
eq-trans	(u_1, u_2) and $(u_2, u_3) \in Eq$	$(u_1, u_3) \in Eq$
eq-typ	$(u_1, u_2) \in Eq$	$\text{type}(u_1) \subseteq \text{type}(u_2)$
eq-mrk-1	$(u_1, u_2) \in Eq$	$\text{marker}(u_1) \subseteq \text{marker}(u_2)$
eq-mrk-2	$u_1, u_2 \in U$ such that $\text{marker}(u_1) \cap \text{marker}(u_2) \neq \emptyset$	$(u_1, u_2) \in Eq$
p-eq-s	$(u_1, r, v) \in P$ and $(u_1, u_2) \in Eq$	$(u_2, r, v) \in P$
p-eq-o	$(u, r, v_1) \in P$ and $(v_1, v_2) \in Eq$	$(u, r, v_2) \in P$
p-root	$(u, r, v) \in P$	$\text{root}(u) \in \text{type}(u)$
p-sig	$(u, r, v) \in P$	$\mathfrak{S}_{\text{type}(u)}(r) \subseteq \text{type}(v)$
p-fp	(u, r, v_1) and $(u, r, v_2) \in P$	$(v_1, v_2) \in Eq$
c-eq-s	$(u_1, r, v) \in C$ and $(u_1, u_2) \in Eq$	$(u_2, r, v) \in C$
c-eq-o	$(u, r, v_1) \in C$ and $(v_1, v_2) \in Eq$	$(u, r, v_2) \in C$
c-dom	$(u, r, v) \in C$	$\text{domain}(r) \subseteq \text{type}(u)$
c-rng	$(u, r, v) \in C$	$\text{range}(r) \subseteq \text{type}(v)$
c-sop	$(u, r_1, v) \in C$ and $r_1 \stackrel{\mathcal{C}}{\lesssim} r_2$	$(u, r_2, v) \in C$

Table 1: Semantics of the **UGs**.

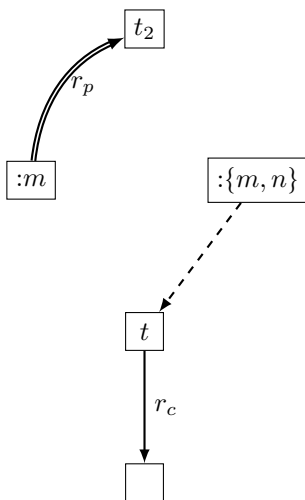
The fact of applying all of the entailment rules on a **UG** G until none of them has any effect on G is called saturating G , which results in G being saturated.

Definition 5.11 (Saturating a **UG**, Saturated **UG**). Let $G \in \mathcal{G}(\mathcal{S})$. The operation of *saturating* G consists in applying sequentially all of the entailment rules of table 1 over and over, until none of them has any effect on G .

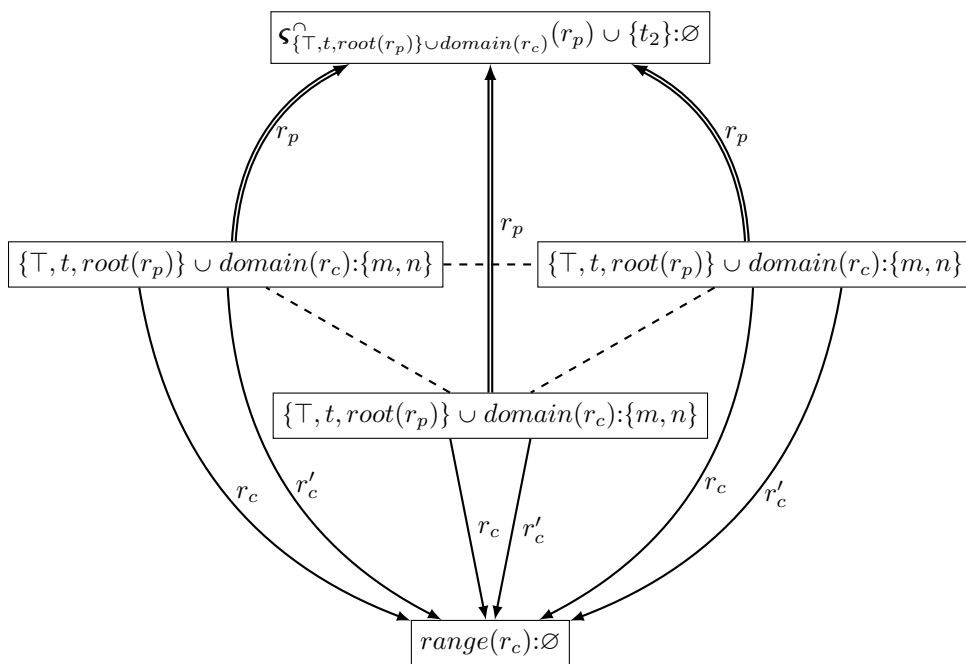
A **UG** for which none of the entailment rules of table 1 have effect is said to be *saturated*.

Consider a simple support \mathcal{S} made of a **CUTs** hierarchy \mathcal{T}^\cap with $t, t_2 \in T_{\text{declared}}$, $r_p \in \mathcal{S}_{\mathcal{T}}$, a **CSymbols** hierarchy with $r_c, r'_c \in \mathcal{S}_{\mathcal{C}}$ and $(r'_c, r_c) \in \mathcal{C}_{\mathcal{S}_{\mathcal{C}}}$, and $m, n \in \mathcal{M}$. Consider the **UG** $G \in \mathcal{G}(\mathcal{S})$ illustrated on figure 10a. The saturation of G is illustrated on figure 10b.

When G is saturated, it is clear that some informations are redundant, for instance, the three central nodes in figure 10b hold the same information about the entity they commonly represent.



(a) G



(b) $sat(G)$

Figure 10: Saturation of a simple **UG**.

5.5.2 Closure

The set of first-order implications of table 1 are non-expansive, i.e., their application adds no unit nodes. Let us precise that this set of entailment rules does not imply decidability of entailment. Indeed, as a unit *has* participants, which themselves *have* participants, the closure **UGs** may be infinite for finite **UGs**. In the knowledge representation terminology, the **PSlots** have a minimal cardinality of 1. This defines the difference between saturation (i.e., add all inferable knowledge without adding unit nodes), and closure (i.e., add all inferable knowledge).

To properly close a **UG**, one needs also to take into account the fact that every unit node represents a unit, and that a unit is potentially a predicate that do have other units that fill each of its **PSlots**. So for every unit node u , and for **PSlot** s of $\text{type}(u)$, if there is no unit node v such that $(u, s, v) \in P$, then one could add v and (u, s, v) in G . This is problematic even for the following basic example:

Consider a simple support \mathcal{S} made of a simple **CUTs** hierarchy \mathcal{T}^\cap such that $t \in T_{\text{declared}}$ and $s \in \mathcal{S}_{\mathcal{T}}$ such that $\text{root}(s) = t$ and $\zeta_t(s) = t$, a void **CSymbols** hierarchy, and a void set of unit markers. Consider the **UG** $G = (\{u\}, \mathbf{l}, \emptyset, \emptyset, \{(u, u)\}) \in \mathcal{G}(\mathcal{S})$, such that $\text{marker}(u) = \emptyset$ and $\text{type}(u) = \{\top, t\}$. G is already saturated, yet one knows that there is a unit that fills **PSlot** s of the unit represented by u . So one could add a unit node v to represent that argument, with $(u, s, v) \in P$. The saturation of that new graph will make v be of type $\{\top, t\}$, and thus the closure of G is an infinite chain of unit nodes having type $\{\top, t\}$ and that fill the s **PSlot** of one another. This is illustrated on figure 11.

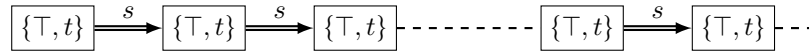


Figure 11: Illustration of an infinite closure of a simple **UG**

6 Mappings of UGs

A **UGs** mapping corresponds to a mapping of their underlying graphs.

Definition 6.1 (UG Mapping). Let $G = (U^g, \ell^g, P^g, C^g, Eq^g) \in \mathcal{G}(\mathcal{S}_G)$ and $H = (U^h, \ell^h, P^h, C^h, Eq^h) \in \mathcal{G}(\mathcal{S}_H)$ be two **UGs**. A *mapping* f from H to G , written $f : H \rightarrow G$, is a mapping $f : U^h \rightarrow U^g$ from the unit nodes of H to the unit nodes of G .

In the rest of this section will define useful different mappings that preserve some informations, such as arcs and labels.

6.1 Weak Homomorphism

There is a homomorphism of **UGs** if there is a homomorphism on their underlying oriented labelled multigraph. To define such a homomorphism, one needs to choose pre-orders over labels for unit nodes and arcs. We define the weak homomorphism using the following set of pre-orders:

- inclusion for markers;
- pre-order $\overset{\sim}{\lesssim}$ for types;
- equality for participation triples;
- pre-order $\overset{\sim}{\lesssim}$ for circumstantial dependency relations;
- equality for asserted equivalence relations.

Then the definition of a weak homomorphism is straightforward.

Definition 6.2 (UG Homomorphism). Let $G = (U^g, \ell^g, P^g, C^g, Eq^g) \in \mathcal{G}(\mathcal{S})$ and $H = (U^h, \ell^h, P^h, C^h, Eq^h) \in \mathcal{G}(\mathcal{S})$ be two **UGs** defined on the same support \mathcal{S} . A mapping $\pi : H \rightarrow G$ is a *homomorphism* if and only if all of the following is true:

- $\forall u \in U^h, \text{marker}(u) \subseteq \text{marker}(\pi(u))$;
- $\forall u \in U^h, \text{type}(\pi(u)) \overset{\sim}{\lesssim} \text{type}(u)$;
- $(u, r, v) \in P^h \Rightarrow (\pi(u), r, \pi(v)) \in P^g$
- $(u, r, v) \in C^h \Rightarrow \exists c \in C^g, \text{arc}(c) = (\pi(u), \pi(v))$ and $\text{symbol}(c) \overset{\sim}{\lesssim} r$
- $(u, v) \in Eq^h \Rightarrow (\pi(u), \pi(v)) \in Eq^g$;

For example, figure 12 illustrates a weak homomorphism from H to G .

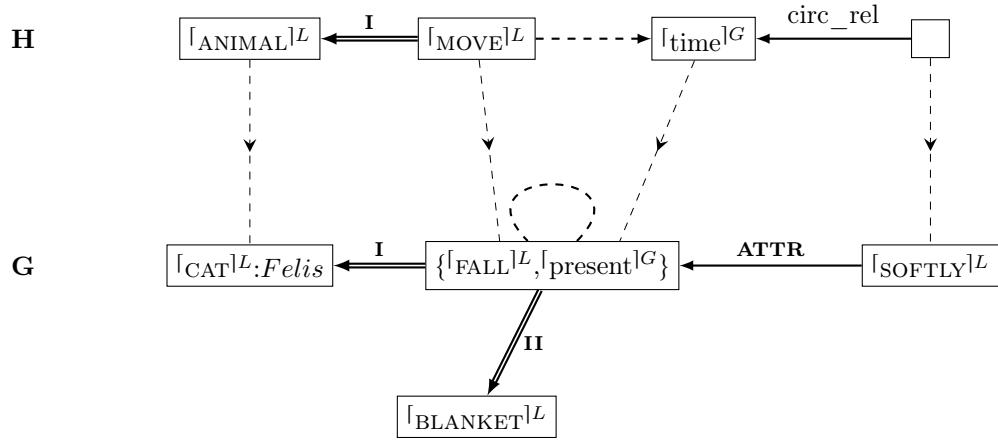


Figure 12: Illustration of a weak homomorphism from H to G .

6.2 Weak Hom-Equivalence

Two **UGs** are weakly hom-equivalent if there exists a homomorphism from one to the other and vice versa.

Definition 6.3 (Weak Hom-Equivalence). Let $G = (U^g, \mathcal{I}^g, P^g, C^g, Eq^g) \in \mathcal{G}(S)$ and $H = (U^h, \mathcal{I}^h, P^h, C^h, Eq^h) \in \mathcal{G}(S)$ be two **UGs** defined on the same support. G and H are Weakly Hom-Equivalent if there exists a weak homomorphism from G to H and a weak homomorphism from H to G .

Figure 13 below illustrates two weakly hom-equivalent **UGs** that are different.

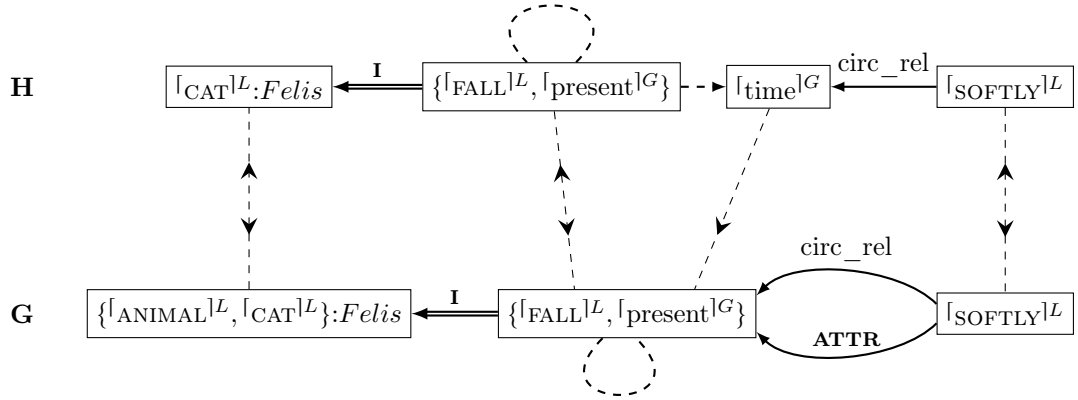


Figure 13: Illustration of a weak hom-equivalence between H and G .

6.3 Weak Isomorphism

A weak hom-equivalence is an isomorphism if the two mappings are the inverse one of another.

Definition 6.4 (Weak Isomorphism). Let $G = (U^g, \mathcal{I}^g, P^g, C^g, Eq^g) \in \mathcal{G}(\mathcal{S})$ and $H = (U^h, \mathcal{I}^h, P^h, C^h, Eq^h) \in \mathcal{G}(\mathcal{S})$ be two **UGs** defined on the same support. A mapping $\pi : G \rightarrow H$ is a *isomorphism* if and only if it is a bijective weak homomorphism.

Figure 14 below illustrates two weakly isomorphic **UGs** that are different.

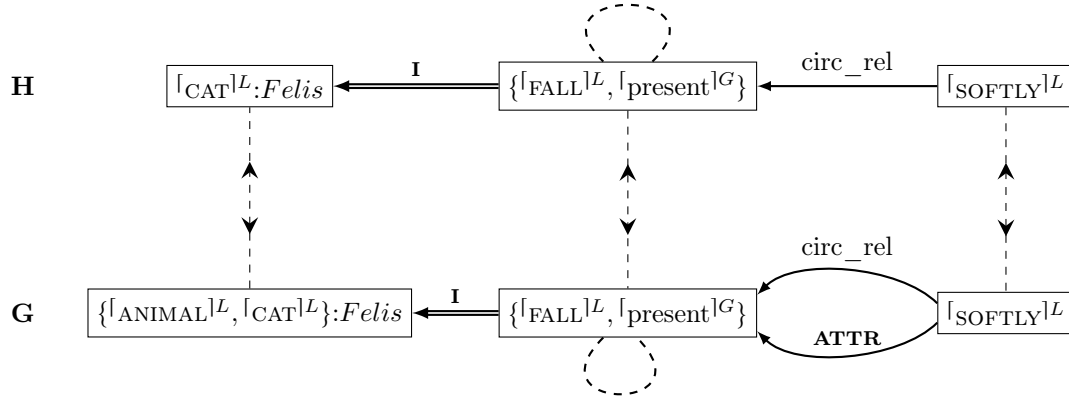


Figure 14: Illustration of a weak isomorphism between H and G .

6.4 Strong Homomorphism: Inclusion

We also define another type of homomorphism of **UGs** using stronger selection of pre-orders over labels for unit nodes and arcs: only inclusion or equality.

- inclusion for markers;
- inclusion for types;
- equality for participation triples;
- equality for circumstantial dependency relations;
- equality for asserted equivalence relations.

We say that there is a strong homomorphism from two **UG** H and G , or equivalently that H is included in a **UG** G , or that H is a sub-**UG** of G .

Definition 6.5 (**UG** Inclusion). Let $G = (U^g, \mathcal{I}^g, P^g, C^g, Eq^g) \in \mathcal{G}(\mathcal{S})$ and $H = (U^h, \mathcal{I}^h, P^h, C^h, Eq^h) \in \mathcal{G}(\mathcal{S})$ be two **UGs** defined on the same support \mathcal{S} . There is a *strong homomorphism* from H to G (H is included in G , H is a sub-**UG** of G) if and only if there exists a mapping $\pi : H \rightarrow G$ such that all of the following is true:

- $\forall u \in U^h, \text{marker}(u) \subseteq \text{marker}(\pi(u))$;
- $\forall u \in U^h, \text{type}(u) \subseteq \text{type}(\pi(u))$;
- $(u, r, v) \in P^h \Rightarrow (\pi(u), r, \pi(v)) \in P^g$

- $(u, r, v) \in C^h \Rightarrow (\pi(u), r, \pi(v)) \in C^h$
- $(u, v) \in Eq^h \Rightarrow (\pi(u), \pi(v)) \in Eq^g$;

For example, figure 15 illustrates a strong homomorphism from H to G .

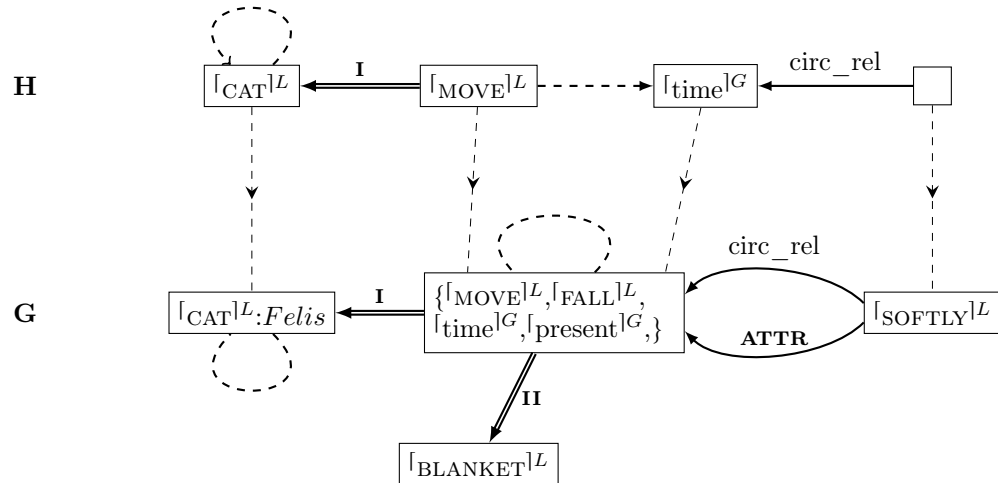


Figure 15: Illustration of a strong homomorphism from H to G .

6.5 Strong Isomorphism: Equality

Then two **UG** are equal if there is a strong isomorphism between them.

Definition 6.6 (UG Equality). Let $G = (U^g, \mathfrak{l}^g, P^g, C^g, Eq^g) \in \mathcal{G}(\mathcal{S})$ and $H = (U^h, \mathfrak{l}^h, P^h, C^h, Eq^h) \in \mathcal{G}(\mathcal{S})$ be two **UGs** defined on the same support. H is equal to G if and only if there exists a bijective inclusion mapping $\pi : G \rightarrow H$.

Figure 16 summarizes the different mappings we introduced in this section.

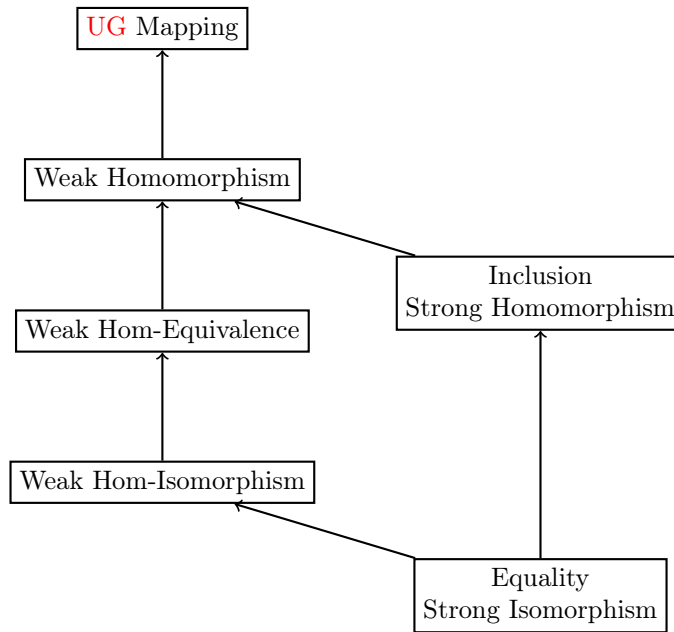


Figure 16: Illustration of the links between the different **UGs** mappings.

7 Elementary Operations

Now that we defined **UGs**, we are interested in this section in elementary operations that may be applied over **UGs**.

7.1 Explicitation Operations

A **UG** G defined on a support \mathcal{S} represents explicit knowledge, and other knowledge is contained in \mathcal{S} .

Let $G = (U, l, P, C, Eq) \in \mathcal{G}(\mathcal{S})$. The following operations explicit knowledge of \mathcal{S} in G . They are either partial saturation operations, or partial closure operations from section 5.5.

- **Copy** create a disjoint copy of G . $copy(G)$ is disjoint from G and equal to G .
- **u-tyt-top** Let $u \in U$. Add \top to $type(u)$;
- **u-tyt** Let $u \in U$, and $t, t' \in \mathbf{T}$ such that $t \in type(u)$ and $t \lesssim t'$. Add t' to $type(u)$;
- **eq-ref** Let $u \in U$. Add (u, u) in Eq ;
- **eq-sym** Let $(u_1, u_2) \in Eq$. Add (u_2, u_1) in Eq ;
- **eq-trans** Let $(u_1, u_2), (u_2, u_3) \in Eq$. Add (u_1, u_3) in Eq ;
- **eq-tyt** Let $(u_1, u_2) \in Eq$, and $t \in type(u_1)$. Add t in $type(u_2)$;
- **eq-mrk-1** Let $(u_1, u_2) \in Eq$, and $m \in type(u_1)$. Add m in $marker(u_2)$;
- **eq-mrk-2** Let $u_1, u_2 \in U$ such that $marker(u_1) \cap marker(u_2) \neq \emptyset$. Add (u_1, u_2) in Eq ;
- **p-eq-s** Let $(u_1, r, v) \in P$ and $(u_1, u_2) \in Eq$. Add (u_2, r, v) in P ;
- **p-eq-o** Let $(u, r, v_1) \in P$ and $(v_1, v_2) \in Eq$. Add (u, r, v_2) in P ;
- **p-root** Let $(u, r, v) \in P$. Add $root(u)$ in $type(u)$;
- **p-sig** Let $(u, r, v) \in P$, and $t \in \mathcal{S}_{type(u)}^{\cap}(r)$. Add t in $type(v)$;
- **p-fp** Let $(u, r, v_1), (u, r, v_2) \in P$. Add (v_1, v_2) in Eq ;
- **c-eq-s** Let $(u_1, r, v) \in C$ and $(u_1, u_2) \in Eq$. Add (u_2, r, v) in C ;
- **c-eq-o** Let $(u, r, v_1) \in C$ and $(v_1, v_2) \in Eq$. Add (u, r, v_2) in C ;
- **c-dom** Let $(u, r, v) \in C$, and $t \in domain(r)$. Add t in $type(u)$;
- **c-rng** Let $(u, r, v) \in C$, and $t \in range(r)$. Add t in $type(v)$;
- **c-sop** Let $(u, r_1, v) \in C$ and $r_1 \stackrel{c}{\lesssim} r_2$. Add (u, r_2, v) in C .
- **p-expl** Let $u \in U$ and $s \in \omega(u)$. Add v in U with $type(v) = \emptyset$ and $marker(v) = \emptyset$, and add (u, s, v) in P .
- **u-mrg** Let $(u, v) \in Eq$. Merge u and v as follows:
 - add a new node u , with $type(u) = type(u_1) \cup type(u_2)$ and $marker(u) = marker(u_1) \cup marker(u_2)$;
 - replace u_1 and u_2 by u in any dependency arc in $P \cup C$;
 - replace u_1 and u_2 by u in any element of Eq .

7.2 Specialization Operations

In this section we will only list two specialization operations that will be used later on in this report: Addition of two **UGs**, and merge of two unit nodes in a **UG**.

Addition of two UGs The disjoint addition of two **UGs** is a specialization operation defined by:

Definition 7.1 (Addition of two **UGs**). Let two **UGs** $G = (U^g, \mathbf{l}^g, P^g, C^g, Eq^g)$ and $C = (U^c, \mathbf{l}^c, P^c, C^c, Eq^c)$ be defined over a support \mathcal{S} , the addition of G and C is a **UG** $G + C$ such that:

$$G + C = (U^g \cup U^c, \mathbf{l}^g \cup \mathbf{l}^c, P^g \cup P^c, C^g \cup C^c, Eq^g \cup Eq^c)$$

Merge of unit nodes One simple specialization operation is to merge pairs of unit nodes u_1, u_2 of G , as defined by:

Definition 7.2 (Merge two unit nodes in a **UG**). Let a **UG** $G = (U, \mathbf{l}, P, C, Eq)$ be defined over a support \mathcal{S} , and $u_1, u_2 \in U$. The merge of u_1 and u_2 in G is denoted $merge(G, u_1, u_2) \in \mathcal{G}(\mathcal{S})$ and is defined by:

1. add a new node u , with $type(u) = type(u_1) \cup type(u_2)$ and $marker(u) = marker(u_1) \cup marker(u_2)$;
2. replace u_1 and u_2 by u in any dependency arc in $P \cup C$;
3. replace u_1 and u_2 by u in any element of Eq .

Associated with this operation we also define a mapping ρ from U to unit nodes of $merge(G, u_1, u_2)$ such that $\rho(v) = u$ if $v = u_1$ or $v = u_2$, and $\rho(v) = v$ otherwise.

8 Rules

8.1 λ -UG

It is useful to be able to distinguish some unit nodes of a **UG**, in particular for definitions of **PUTs** and rules. For this purpose we define generic unit nodes and λ -UGs.

Definition 8.1 (Genericity). In a **UG** $G \in \mathcal{G}(\mathcal{S})$, a unit node u is said to be *generic* if and only if $\text{marker}(u) = \emptyset$. G is said to be *generic* if and only if all of its unit nodes are generic.

Definition 8.2 (λ -UG and Freedom). A λ -UG $L = \{u_1, \dots, u_n\}G$ defined over a support \mathcal{S} , is composed of a **UG** $G = (U, \mathbf{l}, P, C, Eq) \in \mathcal{G}(\mathcal{S})$, and a set of generic unit nodes of G , $\{u_1, \dots, u_n\}$, denoted the free nodes of L . A unit node $u \in U$ is said to be *free* if and only if $u \in \{u_1, \dots, u_n\}$. G is said to be *free* if and only if all of its unit nodes are free, i.e., if $\{u_1, \dots, u_n\} = U$.

A **UG** may be considered as a λ -UG with $\{u_1, \dots, u_n\} = \emptyset$.

Definition 8.3 (Merge a λ -UG in a **UG** with respect to a mapping). Let a λ -UG $C = \{u_1^c, \dots, u_n^c\}C'$ and a **UG** $G = (U, \mathbf{l}, P, C, Eq)$ be defined over a support \mathcal{S} . Let η be a mapping from $\{u_1^c, \dots, u_n^c\}$ to U . The merge of C in G with respect to η is denoted $\text{merge}(G, C, \eta)$ and is obtained as follows:

1. add C' to G ;
2. for all $u^c \in \{u_1^c, \dots, u_n^c\}$, merge u^c and $\eta(u^c)$ in G .

8.2 Definition and Logical Semantics of Rules

Definition 8.4 (Rule). A *Rule* defined over a support \mathcal{S} is a triple $R \stackrel{\text{def}}{=} (H, C, \kappa)$ where:

- $H = \{u_1^h, \dots, u_n^h\}H'$ is a λ -UG defined over \mathcal{S} called the hypothesis;
- $C = \{u_1^c, \dots, u_n^c\}C'$ is a λ -UG defined over \mathcal{S} called the conclusion;
- κ is a bijection from $\{u_1^h, \dots, u_n^h\}$ to $\{u_1^c, \dots, u_n^c\}$.

Definition 8.5 (Applicable Rule). Let a rule $R = (H, C, \kappa)$ and a **UG** G be defined over a support \mathcal{S} . R is *applicable* to G if and only if there exists a homomorphism from H to G .

Definition 8.6 (Applying a Rule). Let $R = (H, C, \kappa)$ be a rule applicable to a **UG** G , and let π be a homomorphism from H to G . The application of R on G with respect to π is the graph obtained by merging C in G with respect to $\pi \circ \kappa^{-1}$.

8.3 PUTs Prints and Definitions

We will now formalize the notion of *definition* of a **PUT** to include a set of **PUT** definitions in the definition of the support. Intuitively, a definition corresponds to two reciprocal rules. First, let us introduce a kind of small **UG** that conveniently represents some informations about a **PUT** in \mathbf{T} : the print of a **PUT**.

Definition 8.7 (**PUT** print). Let \mathcal{S} be a support. The *print* of a **PUT** t is a λ -UG $D_t^- = \{u_t, v_1, \dots, v_{\text{valency}_\alpha(t)}\}(U^-, \mathbf{l}^-, P^-, \emptyset, \emptyset)$ such that:

- $U^- = \{u_t, v_1, \dots, v_{\text{valency}_\alpha(t)}\}$ (i.e., D_t^- is free);
- $\text{type}(u_t) = \{t\}$;

- there is a bijection \mathbf{v} from $\alpha(t)$ to $\{v_1, \dots, v_{valency_\alpha(t)}\}$;
- $P^- = \{(u_t, s, \mathbf{v}(s))\}_{s \in \alpha(t)}$ (i.e., u_t is explicit and any other unit node is implicit);
- for all $s \in \alpha(t)$, $type(\mathbf{v}(s)) = \zeta_t(s)$;
- u_t is called the *central unit node* of D_t^- .

For instance, figures 18 on the left and 17 represent prints of (push) and \perp respectively.

$$\boxed{\{\perp\}}$$

Figure 17: print of the PUT \perp . We know from proposition 2.12 that in \mathcal{S} , $\alpha(\perp) = \emptyset$.

The print of a PUT t conveniently represents its ASlots and associated signatures, but it cannot represent the signatures of its HSlots, nor comparisons with other PUTs. Finally, we want to be able to specify how exactly the PUT t is different from more generic PUTs.

We will hence introduce the notion of *definition* of a PUT t . Informally, a definition defines an equivalence between two λ -UG defined over the same support. One of the λ -UG is the print of t , while the other λ -UG is called the *expansion* of t (or the expansion of the print).

Definition 8.8 (Primitive Unit Type Definition). A definition D_t of a PUT t is a triple $D_t \stackrel{\text{def}}{=} (D_t^-, D_t^+, \kappa)$ where:

- $D_t^- = \{u_t^-, v_1^-, \dots, v_{valency_\alpha(t)}^-\}(U^-, \mathbf{I}^-, P^-, \emptyset, \emptyset)$ is the print of t with central node u_t , and bijection \mathbf{v} from $\alpha(t)$ to $\{v_1, \dots, v_{valency_\alpha(t)}\}$;
- $D_t^+ = \{u_t^+, v_1^+, \dots, v_{valency_\alpha(t)}^+\}(U^+, \mathbf{I}^+, P^+, C^+, Eq^+)$ is a λ -UG defined over \mathcal{S} , called the *expansion* of t ;
- κ is a bijection from $\{u_t^-, v_1^-, \dots, v_{valency_\alpha(t)}^-\}$ to $\{u_t^+, v_1^+, \dots, v_{valency_\alpha(t)}^+\}$, such that $\kappa(u_t^-) = u_t^+$, and for all i , $\kappa(v_i^-) = v_i^+$;
- $type(u_t^+) = \{t' \in \mathbf{T} \mid (t', t) \in C_{asserted}\}$ is called the *genus* of t and is denoted $genus(t)$;
- for all $s \in \alpha(t) \cap \alpha^\wedge(genus(t))$, $(u_t^+, s, \kappa \circ \mathbf{v}(s)) \in P^+$;
- for all $s \in \omega^\wedge(genus(t))$, $\forall v \in U^+$, $(u_t^+, s, v) \in P^+ \Rightarrow type(v) = \zeta_t(s)$;
- $\alpha(t) - \alpha^\wedge(genus(t)) = \{s \in \mathcal{S}_T \mid root(s) = t\}$;
- $\alpha^\wedge(genus(t)) - \alpha(t) = \{s \in \mathcal{S}_T \mid t \in Hiders(s)\}$;

(D_t^-, D_t^+, κ) and $(D_t^+, D_t^-, \kappa^{-1})$ are respectively called the $E \rightarrow P$ (expansion \rightarrow print) and the $P \rightarrow E$ (print \rightarrow expansion) rules of t .

Consider a set of definitions \mathcal{D} . If a definition explicitly or implicitly refers to the type of unit it defines, then there is a circularity in \mathcal{D} .

Definition 8.9 (Circularity). A set of definitions \mathcal{D} has a circularity if and only if there exists a series t_1, \dots, t_n of PUTs such that $t_1 = t_n$ and for each PUT t_i , $i \in [1..n-1]$, there exists a definition of t_i such that the expansion of t contains a unit node u with $t_{i+1} \in type(u)$.

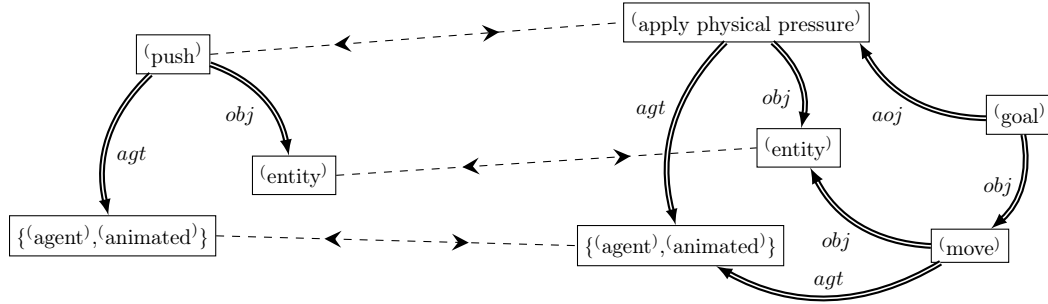


Figure 18: Definition of $[PUSH]^L$. On the left, the print of $(push)$.

8.4 UG Support with PUTs Definitions and GDefs

With this definition tool, we may now update the definition of support to take into account a set of definitions. support is now composed of a **CUTs** hierarchy, a **CSymbols** hierarchy, a set of unit markers, and a non-circular set of definitions.

Definition 8.10 (support). A *support* is a tuple $\mathcal{S} \stackrel{\text{def}}{=} (\mathcal{T}^\wedge, \mathcal{C}, \mathbf{M}, \mathcal{D})$ where:

- \mathcal{T}^\wedge is a *Conjunctive Unit Types (CUTs) hierarchy*;
- \mathcal{C} is a *Circumstantial Dependency Symbols (CSymbols) hierarchy*;
- \mathbf{M} is the set of so-called *unit markers*. $\mathbf{M}^\wedge = 2^{\mathbf{M}}$ is the set of unit node markers, with \emptyset the *generic unit node marker*.
- \mathcal{D} a non-circular set of **PUT** definitions.

This new definition of a support does not invalidate the previously introduced notions. From now on and otherwise specified, any use of the term support will refer to this definition with a non-circular set of definitions.

Part IV

Conclusion

We thus studied how to formalize, in the sense of knowledge engineering, the **ECD**, in order to represent, manipulate, query, and reason linguistic knowledge. We may now answer the different questions asked through this report.

What makes the semantic web formalisms and the **CGs** formalisms not directly adapted to the representation of knowledge of the **ECD** ? The semantics, in the logical sense, of RDF is insufficient to represent knowledge of the **ECD**, and we showed that there are serious problems in using OWL. The **CGs** formalism has strong similarities with the **MTT**, but does not enable to represent the concept/relation duality of a sense unit. As the **CG** formalism is the closest from the **MTT**, we proposed to use it as a starting point for designing the new **Unit Graphs (UGs)** formalism adapted to the representation of knowledge of the **ECD**.

What mathematical structure for a hierarchy of unit types that may have actant slots ? To take into account the concept/relation duality of sense units, the predicate-argument relations are symbolized by **Participation Symbols (PSymbols)**, and we associate to each **PSymbols** s a root **Primitive Unit Type (PUT)** $root(s)$ and a set of hiders $Hiders(s)$. Thus, in the pre-ordered set of **PUTs**, a **PSlot** having s as **PSymbol** is introduced by $root(s)$, and first defines a **ASlot** for any **PUT** more specific than $root(s)$ as long as that **PUT** is not more specific than one of the elements in $Hiders(s)$. If that happens, then the **PSlot** becomes non-actancial and defines a **HSlot**. Any **PUT** that possesses **ASlots** thus also models a relation type, that may link every instance of that type to the set of its actants in the utterance representation. Finally, to every **PUT** is assigned a signature that specifies the type of participants of its instances. We extended the hierarchy of unit types to their conjunctive version and introduced the **Conjunctive Unit Types (CUTs)** hierarchy.

What is a **UG**, and how to use them to represent advanced concepts of the **TST** ? We first introduced a hierarchy of **Circumstantial Dependency Symbols (CSymbols)** We then illustrated the definition of **UGs**, that represent unit nodes interconnected by dependency relations and declared equivalence relations. We then introduced advanced concepts in the **UGs** framework that we are currently working on:

- We defined the semantics of **UGs**, that enable reasoning with utterance representations.
- We defined rules that will enable us to represent sense-lexical unit associations, and correspondences between different representation levels (government patterns)
- We showed how to represent lexicographic definitions of the **ECD** using two reciprocal rules.

We are also currently working on a new version of this research report in which we introduce factorization of rules that will enable us to represent lexical functions links, and on a syntax based on semantic web formalisms standards to enable the standardized exchange of knowledge of the **ECD**.

Thanks We would like to warmly thank S. Kahane for his comments that led to radical changes in the presentation of this work.

References

- Alonso Ramos, M. (2003). Hacia un diccionario de colocaciones del español y su codificación. *Lexicografía computacional y semántica*, page 11–34.
- Baget, J. F., Croitoru, M., Gutierrez, A., Leclere, M., and Mugnier, M. L. (2010). Translations between RDF (S) and conceptual graphs. *Conceptual Structures: From Information to Intelligence*, page 28–41.
- Barque, L., Nasr, A., and Polguère, A. (2010). From the Definitions of the 'Trésor de la Langue Française' To a Semantic Database of the French Language. In *Proceedings of the XIV Euroalex International Congress*, Fryske Akademy, page 245–252, Leeuwarden (Pays-Bas). Anne Dykstra et Tanneke Schoonheim, dir.
- Barque, L. and Polguère, A. (2008). Enrichissement formel des définitions du Trésor de la Langue Française informatisé (TLFi) dans une perspective lexicographique. 22.
- Chein, M. and Mugnier, M. L. (2008). *Graph-based Knowledge Representation: Computational Foundations of Conceptual Graphs*. Springer.
- Corby, O., Dieng, R., and Hébert, C. (2000). A conceptual graph model for W3C resource description framework. In Ganter, B. and Mineau, G. W., editors, *Conceptual Structures: Logical, Linguistic, and Computational Issues*, number 1867 in Lecture Notes in Computer Science, pages 468–482. Springer Berlin Heidelberg.
- Kahane, S. and Polguère, A. (2001). Formal foundation of lexical functions. In *Proceedings of ACL/EACL 2001 Workshop on Collocation*, page 8–15.
- Krisnadhi, A., Maier, F., and Hitzler, P. (2011). OWL and Rules. *Reasoning Web. Semantic Technologies for the Web of Data*, page 382–415.
- Leclère, M. (1998). Raisonner avec des définitions de types dans le modèle des graphes conceptuels. (12):243–278.
- Lefrançois, M. and Gandon, F. (2011a). ILexicOn: Toward an ECD-Compliant Interlingual Lexical Ontology Described with Semantic Web Formalisms. In *Proc. of the 5th International Conference on Meaning-Text Theory (MTT 2011)*, page 155–164, Barcelona, Spain. INALCO.
- Lefrançois, M. and Gandon, F. (2011b). ULiS: An Expert System on Linguistics to Support Multilingual Management of Interlingual Knowledge bases. In Kageura, K. and Zweigenbaum, P., editors, *Proc. of the 9th International Conference on Terminology and Artificial Intelligence (TIA 2011)*, page 108–114, Paris, France. INALCO.
- L'Homme, M. C. (2008). Le DiCoInfo: méthodologie pour une nouvelle génération de dictionnaires spécialisés. *Traduire*, (217):78–103.
- Lux-Pogodalla, V. and Polguère, A. (2011). Construction of a French Lexical Network: Methodological Issues. In *Proceedings of the International Workshop on Lexical Resources*, Ljubljana.
- Mel'čuk, I. A. (1996). Lexical functions: a tool for the description of lexical relations in a lexicon. *Lexical functions in lexicography and natural language processing*, 31:37–102.
- Mel'čuk, I. A. (2004a). Actants in Semantics and Syntax I: Actants in Semantics. *Linguistics*, 42(1):1–66.

- Mel'čuk, I. A. (2004b). Actants in Semantics and Syntax II: Actants in Syntax. *Linguistics*, 42(2):247–291.
- Mel'čuk, I. A. and Arbatchewsky-Jumarie, N. (1999). *Dictionnaire explicatif et combinatoire du français contemporain: recherches lexico-sémantiques*, volume 4. PU Montréal.
- Mel'čuk, I. A. (2006). Explanatory Combinatorial Dictionary. *Open problems in linguistics and lexicography*, page 225.
- Polguère, A. (2000). Une base de données lexicales du français et ses applications possibles en didactique. *Revue de Linguistique et de Didactique des Langues (LIDIL)*, 21:75–97.
- Polguère, A. (2009). Lexical systems: graph models of natural language lexicons. *Language resources and evaluation*, 43(1):41–55.
- Rudolph, S. (2011). Foundations of description logics. *Reasoning Web. Semantic Technologies for the Web of Data*, pages 76–136.
- Sowa, J. F. (1984). *Conceptual structures: information processing in mind and machine*. System programming series. Addison-Wesley.
- Sowa, J. F. (1989). Using a lexicon of canonical graphs in a semantic interpreter. In *Relational models of the lexicon*, page 113–137.
- Sérasset, G. (1997). Le projet NADIA-DEC: vers un dictionnaire explicatif et combinatoire informatisé. *LTT*, 97:149–160.
- Tesnière, L. (1959). *Éléments de syntaxe structurale*. C. Klincksieck (Colombes, Impr. ITE).



**RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE**

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399