The Unit Graphs Mathematical Framework

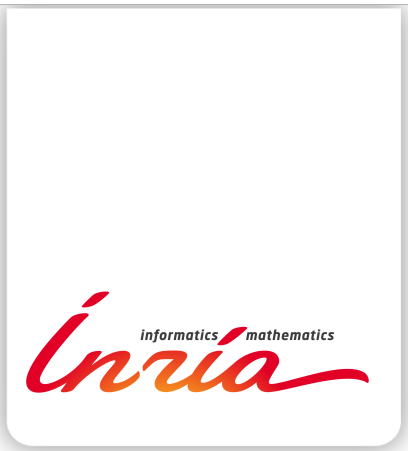
Maxime Lefrançois, Fabien Gandon

The Unit Graphs Mathematical Framework

Maxime Lefrançois, Fabien Gandon

RESEARCH REPORT
N° 8212
March 2013
Project-Team Wimmics

ISSN 0249-6399 ISRN INRIA/RR--8212--FR+ENG



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 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

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 data[1], 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), Dicouèbe 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 TLFi[2].

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:

[1] The web of data is a W3C initiative, highly active today, http://linkeddata.org
[2] Trésor 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.*

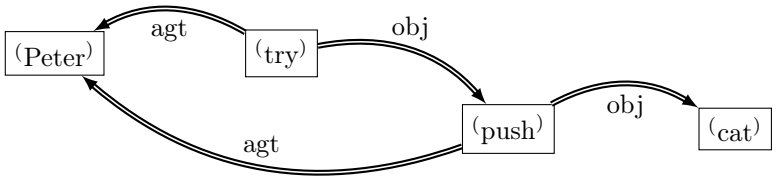


Figure 1: Illustration of the duality concept/relation of sense units in the MTT, semantic representation of utterance *Peter tries to push the cat.*

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 axioms[3] and classes and relation constructors[4].

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( OPE`$_1$ `...OPE`$_n$ `) 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 $\mathcal{SROIQ}$ 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 $\ulcorner$DIE$\urcorner^L$, and wanted to define the lexical unit $\ulcorner$KILL$\urcorner^L$ 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 $\prec$ over the set of non-simple roles. First axiom implies that `hasTime` $\prec$ `hasKillTime`, and second axiom implies that `hasKillTime` $\prec$ `hasTime`. Thus as $\prec$ 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

[3] e.g., Sub-class `SubClassOf( CE`$_1$ `CE`$_2$ `)`; Functional Relation: `FunctionalObjectProperty( OPE)`

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

[5] c.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 ther is the definition of the lexical unit in the graph one may infer that ther 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 derives[6] (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 instanciated 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) ?

[6] For 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).

[7] For 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 glsplput 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 $\boldsymbol{T}$ and defined as follows.

Definition 2.1 (PUT Set)**.** A *PUT Set* is a disjoint union denoted $\boldsymbol{T} \stackrel{\text{def}}{=} T_{declared} \cup \{\top\} \cup \{\bot\}$, where $T_{declared}$ is a finite set of *declared PUTs*, $\top$ is the *prime universal PUT*, and $\bot$ 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 $\boldsymbol{T}$, e.g., $^{(}$to rain$^{)} \lesssim {}^{(}$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 $\boldsymbol{C_T}$ of comparisons of PUTs.

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

By construction, the set of PUTs is bounded by $\top$, a maximal element of $\boldsymbol{T}$, and $\bot$, a minimal element of $\boldsymbol{T}$:

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

Proof. Let us prove that 1) $\top$ is a maximal element of $\boldsymbol{T}$, and 2) $\bot$ is a minimal element of $\boldsymbol{T}$.
1) Let $t \in \boldsymbol{T}$. From definition 2.2, $(\top, t) \in C_\top$.
$C_\top \subseteq \boldsymbol{C_T} \subseteq \boldsymbol{C_T^*}$, so $(\top, t) \in \boldsymbol{C_T^*}$. So $t \lesssim \top$ (def. 2.2) and $\top$ is a maximal element of $\boldsymbol{T}$.
2) Conversely, let $t \in \boldsymbol{T}$. From definition 2.2, $(t, \bot) \in C_\bot$.
$C_\bot \subseteq \boldsymbol{C_T} \subseteq \boldsymbol{C_T^*}$, so $(t, \bot) \in \boldsymbol{C_T^*}$, so $\bot \lesssim t$ (def. 2.2) and $\bot$ is a maximal element of $\boldsymbol{T}$. □

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 $\boldsymbol{T}$. Let $t \in \boldsymbol{T}$, we denote $[t]$ the equivalence class to which t belongs, i.e., $[t] \stackrel{\text{def}}{=} \{t' \in \boldsymbol{T} \mid t' \simeq t\}$.

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

$\bot$ 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 $\bot$ should also be absurd. As $\bot$ is also a minimal element of $\boldsymbol{T}$, any PUT which is more specific than $\bot$ is actually equivalent to $\bot$. The equivalence class to which $\bot$ belongs is thus called the set of absurd PUTs and denoted $\bot^{\sim}$. Any $t \in \bot^{\sim}$ 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 $\bot^{\sim}$ and is the set: $\bot^{\sim} \stackrel{\text{def}}{=} [\bot]$.

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 bot 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 beck to the theory of semantic actants (c.f., Mel'čuk, 2004a). The SemASlots of a semantic unit type ⟨L⟩ correspond roughly to participants of the linguistic situation SIT(L) denoted by ⟨L⟩ 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 $\boldsymbol{S_\mathcal{T}}$. Any PUT has a set of *PSlots* which is defined through a mapping $\boldsymbol{\omega}$ from $\boldsymbol{T}$ to $2^{\boldsymbol{S_\mathcal{T}}}$. For every $t \in \boldsymbol{T}$, $s \in \boldsymbol{\omega}(t)$ means that t has a PSlot with PSymbol s. The number of PSlots of a PUT t is denoted the $\boldsymbol{\omega}$-*valency* of t, i.e., $valency_\omega(t) \stackrel{\text{def}}{=} |\boldsymbol{\omega}(t)|$.

For instance, in our example illustrated on figure 2 it is assumed that there is a PSymbol $obj \in \boldsymbol{S_\mathcal{T}}$ 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., $obj \in \boldsymbol{\omega}(^{(}\text{fall}^{)})$.

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 $\boldsymbol{T}$ denoted $root(s)$ that introduces a PSlot with symbol s. We define the set of PSlots of a PUT $t \in \boldsymbol{T}$ 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 $\boldsymbol{S_\mathcal{T}}$ to $\boldsymbol{T}$ that links a PSlot s to the PUT that introduces s. The set of PSlots of a PUT $t \in \boldsymbol{T}$ is defined as the set of PSlots whose root is more general than t, i.e., $\forall t \in \boldsymbol{T}, \boldsymbol{\omega}(t) \stackrel{\text{def}}{=} \{s \in \boldsymbol{S_\mathcal{T}} \mid t \lesssim root(s)\}$.

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 $root(obj) = {}^{(}\text{fact}^{)}$.

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.,* $\downarrow root(s) \stackrel{def}{=} \{t \in \mathbf{T} \mid t \lesssim root(s)\}$*. For any* $s \in \mathbf{S}_\mathcal{T}$*,* $\{t \in \mathbf{T} \mid s \in \boldsymbol{\omega}(t)\} = \downarrow root(s)$*.*

Proof. Let $s \in \boldsymbol{S_\mathcal{T}}$.
$\subseteq$: Let $t \in \{t \in \boldsymbol{T} \mid s \in \boldsymbol{\omega}(t)\}$. $s \in \boldsymbol{\omega}(t)$, 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, $s \in \boldsymbol{\omega}(t)$, so $t \in \{t \in \boldsymbol{T} \mid s \in \boldsymbol{\omega}(t)\}$. □

[8] Despite 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 ★, and the green zone represents the set of PUTs that have a PSlot *obj*, i.e., the principal lower set of $\boldsymbol{T}$ generated by $root(obj)$.

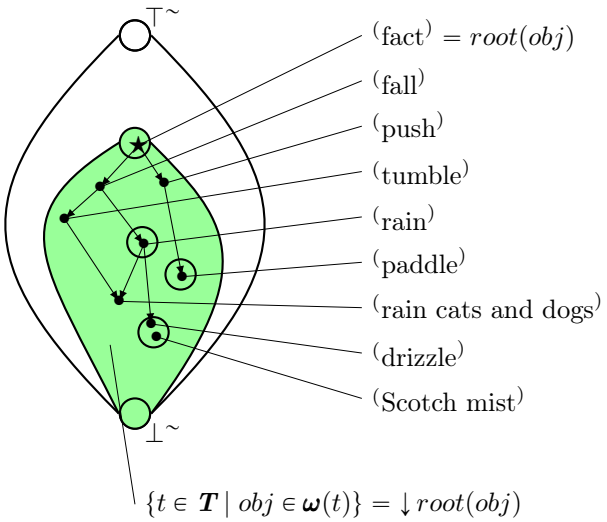


Figure 2: Illustration of the set of PUTs that have a PSlot *obj*.

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.,*

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

Moreover, if $t_x \simeq t_y$, *then* $\boldsymbol{\omega}(t_y) = \boldsymbol{\omega}(t_x)$.

Proof. Let $t_x, t_y \in \boldsymbol{T}$ such that $t_x \lesssim t_y$, and $s \in \boldsymbol{\omega}(t_y)$.
Definition 2.6 implies $t_y \lesssim root(s)$. So $t_x \lesssim t_y \lesssim root(s)$ and $s \in \boldsymbol{\omega}(t_x)$.
The second result is obtained using twice this first result. □

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

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

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

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 $\boldsymbol{T}$ denoted $Hiders(s)$ that hide a PSlot s, and that the set of HSlots of a PUT $t \in \boldsymbol{T}$ 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 $\boldsymbol{S_\mathcal{T}}$ to $2^{\downarrow root(s)} \backslash \varnothing$ 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 $\overline{\boldsymbol{\alpha}}$ from $\boldsymbol{T}$ to $2^{\boldsymbol{S_\mathcal{T}}}$ such that $\forall t \in \boldsymbol{T}, \overline{\boldsymbol{\alpha}}(t) \stackrel{\text{def}}{=} \{s \in \boldsymbol{S_\mathcal{T}} \mid \exists t_h \in Hiders(s), t \lesssim t_h\}$

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* hidden [9]. 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.,* $\downarrow Hiders(s) \stackrel{def}{=} \{t \in \mathbf{T} \mid \exists t_h \in Hiders(s) : t \lesssim t_h\}$*. For any* $s \in \mathbf{S}_\mathcal{T}$*,* $\{t \in \mathbf{T} \mid s \in \overline{\boldsymbol{\alpha}}(t)\} = \downarrow Hiders(s)$*.*

Proof. Let $s \in \boldsymbol{S_\mathcal{T}}$.
$\subseteq$: Let $t \in \{t \in \boldsymbol{T} \mid s \in \overline{\boldsymbol{\alpha}}(t)\}$. $s \in \overline{\boldsymbol{\alpha}}(t)$.
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, $s \in \overline{\boldsymbol{\alpha}}(t)$. Thus $t \in \{t \in \boldsymbol{T} \mid s \in \overline{\boldsymbol{\alpha}}(t)\}$. □

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

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

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

[9] The 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.

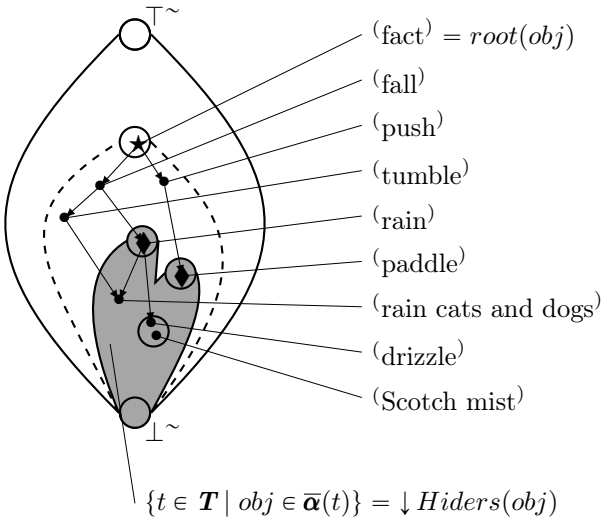


Figure 3: Illustration of the set of PUTs that have a HSlot *obj*.

Proof. Let $t_x, t_y \in \boldsymbol{T}$ such that $t_x \lesssim t_y$, and $s \in \overline{\boldsymbol{\alpha}}(t_y)$.
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 $s \in \overline{\boldsymbol{\alpha}}(t_x)$.
The second result is obtained using twice this first result. □

Second, as $Hiders$ is taken in the lower set generated by $root(s)$, i.e., $2^{\downarrow root(s)} \backslash \varnothing$, 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 $t \in \mathbf{T}$, $\overline{\boldsymbol{\alpha}}(t)$ is a subset of $\boldsymbol{\omega}(t)$.*

Proof. Let $t \in \boldsymbol{T}$, and $s \in \overline{\boldsymbol{\alpha}}(t)$.
From definition 2.7, $\exists t_h \in Hiders(s), t \lesssim t_h$.
Then as $Hiders(s) \in 2^{\downarrow root(s)} \backslash \varnothing$, we know that $t \lesssim t_h \lesssim root(s)$.
So by definition 2.6, $s \in \boldsymbol{\omega}(t)$ and $\overline{\boldsymbol{\alpha}}(t) \subseteq \boldsymbol{\omega}(t)$. □

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

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

Proof. Let $s \in \boldsymbol{S}_{\mathcal{T}}$.
From definition 2.7 we know that $Hiders(s)$ is not empty and thus $\exists t_h \in Hiders(s)$.
As $\bot \lesssim t_h$, then $\bot \in \downarrow Hiders(s)$ and $s \in \overline{\boldsymbol{\alpha}}(\bot)$.
Now, let $t \in \bot^{\sim}$, $t \lesssim \bot$ and using proposition 2.6, $\boldsymbol{S}_{\mathcal{T}} \subseteq \overline{\boldsymbol{\alpha}}(t)$.
So for all $t \in \bot^{\sim}$, $\boldsymbol{S}_{\mathcal{T}} \subseteq \overline{\boldsymbol{\alpha}}(t)$, and thus $\boldsymbol{S}_{\mathcal{T}} = \overline{\boldsymbol{\alpha}}(t)$. □

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

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

Proof. From definition 2.8: $\forall t \in \boldsymbol{T}, \boldsymbol{\alpha}(t) = \boldsymbol{\omega}(t) - \overline{\boldsymbol{\alpha}}(t) \Leftrightarrow \forall t \in \boldsymbol{T}, \boldsymbol{\omega}(t) = \boldsymbol{\alpha}(t) \cup \overline{\boldsymbol{\alpha}}(t)$ □

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

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

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

Proof. Let $s \in \boldsymbol{S}_\mathcal{T}$.
$\subseteq$: Let $t \in \{t \in \boldsymbol{T} \mid s \in \boldsymbol{\alpha}(t)\}$. $s \in \boldsymbol{\alpha}(t)$.
By definition 2.8, $s \in \boldsymbol{\omega}(t) - \overline{\boldsymbol{\alpha}}(t)$.
By definition 2.6 and proposition 2.7, $s \lesssim root(s)$ and $\forall t_h \in Hiders(s), t \not\lesssim t_h$.
So $t \in \downarrow root(s) - \downarrow Hiders(s)$.
$\supseteq$: Let $t \in \downarrow root(s) - \downarrow Hiders(s)$.
$s \lesssim root(s)$ and $\forall t_h \in Hiders(s), t \not\lesssim t_h$.
By definition 2.6 and proposition 2.7, $s \in \boldsymbol{\omega}(t)$ and $s \notin \overline{\boldsymbol{\alpha}}(t)$.
So $s \in \boldsymbol{\alpha}(t)$, and $t \in \{t \in \boldsymbol{T} \mid s \in \boldsymbol{\alpha}(t)\}$. □

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

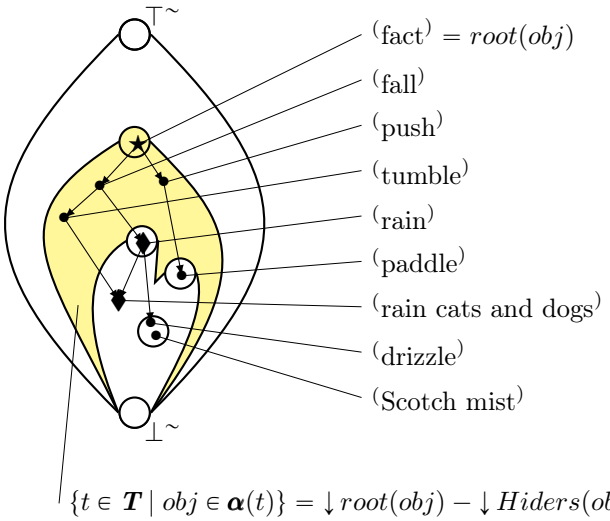


Figure 4: Illustration of the set of PUT that have ASlots *obj*.

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* $t \simeq t'$, $\boldsymbol{\alpha}(t) = \boldsymbol{\alpha}(t')$.

Proof. From definition 2.8, $\boldsymbol{\alpha}(t) = \boldsymbol{\omega}(t) - \overline{\boldsymbol{\alpha}}(t)$.
Moreover, from propositions 2.3 and 2.6, $\boldsymbol{\omega}(t') = \boldsymbol{\omega}(t)$ and $\overline{\boldsymbol{\alpha}}(t') = \overline{\boldsymbol{\alpha}}(t)$.
So $\boldsymbol{\omega}(t') = \boldsymbol{\omega}(t)$. □

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

Proposition 2.12. $\forall t \in \perp^{\sim}, \boldsymbol{\alpha}(t) = \varnothing$.

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

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 $\{\boldsymbol{\varsigma}_t\}_{t \in \boldsymbol{T}}$ is a set of functions from $\boldsymbol{S}_{\mathcal{T}}$ to $2^{\boldsymbol{T}}$. For every PUT t, $\boldsymbol{\varsigma}_t$ is a function with $domain(\boldsymbol{\varsigma}_t) = \boldsymbol{\omega}(t)$ that associates to each PSlot s of t a set of PUT $\boldsymbol{\varsigma}_t(s)$ that characterize the type units that may fill this slot should be.

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

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

$$t_1 \lesssim t_2 \Rightarrow \boldsymbol{\varsigma}_{t_1}(s) \overset{\frown}{\lesssim} \boldsymbol{\varsigma}_{t_2}(s) \tag{4}$$

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 have a lexical unit type and multiple grammatical unit types for instance, like $\{\ulcorner\text{def}\urcorner^G, \ulcorner\text{USA}\urcorner^L\}$ for *"the USA"*.

In section 3.1 we hence introduce the set of Conjunctive Unit Type (CUT) as the set of all subsets of $\boldsymbol{T}$, 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 $\{\ulcorner\text{def}\urcorner^G, \ulcorner\text{USA}\urcorner^L\}$ for *"the USA"* is composed of a lexical unit type $\ulcorner\text{USA}\urcorner^L$ and a grammatical unit type $\ulcorner\text{def}\urcorner^G$. We hence introduce the set of possible Conjunctive Unit Type (CUT) which is the powerset of $\boldsymbol{T}$, i.e., the set of all subsets of $\boldsymbol{T}$:

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

Some CUTs such as $\{\ulcorner\text{def}\urcorner^G, \ulcorner\text{indef}\urcorner^G\}$ are said to be absurd. This means there is no unit that has both types $\ulcorner\text{def}\urcorner^G$ and $\ulcorner\text{indef}\urcorner^G$. 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 $\bot^\sqcap_{asserted}$. And $\bot^\sqcap_{asserted} \subseteq \boldsymbol{T}^\cap$.

3.2 CUT Slots and Signatures

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

For instance, consider a CUT $t^\cap = \{\langle\text{move}\rangle, \langle\text{quick}\rangle\}$. $t^\cap$ contains the PUT $\langle\text{move}\rangle$, and as $obj \in \boldsymbol{\omega}(\langle\text{move}\rangle)$, 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 $\boldsymbol{\omega}$-valency of a CUT)**.** The set of *PSlots of CUTs* is defined through a mapping $\boldsymbol{\omega}^\cap$ from $\boldsymbol{T}^\cap$ to $2^{\boldsymbol{S}_\mathcal{T}}$ Such that $\forall t^\cap \in \boldsymbol{T}^\cap, \boldsymbol{\omega}^\cap(t^\cap) \stackrel{\text{def}}{=} \bigcup_{t\in t^\cap} \boldsymbol{\omega}(t)$. The number of PSlots of a CUT $t^\cap$ is denoted the $\boldsymbol{\omega}$*-valency* of $t^\cap$, i.e., $valency_\omega(t^\cap) \stackrel{\text{def}}{=} |\boldsymbol{\omega}^\cap(t^\cap)|$.

Now, consider CUT $t^\cap = \{\langle\text{move}\rangle, \langle\text{rain}\rangle\}$. PSlot obj is hidden for $\langle\text{rain}\rangle$, but is not for $\langle\text{move}\rangle$. 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 $\overline{\boldsymbol{\alpha}}^\cap$ from $\boldsymbol{T}^\cap$ to $2^{\boldsymbol{S}_\mathcal{T}}$ Such that $\forall t^\cap \in \boldsymbol{T}^\cap, \overline{\boldsymbol{\alpha}}^\cap(t^\cap) \stackrel{\text{def}}{=} \bigcup_{t\in t^\cap} \overline{\boldsymbol{\alpha}}(t)$.

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* $t^\cap \in \mathbf{T}^\cap$, $\overline{\boldsymbol{\alpha}}^\cap(t^\cap)$ *is a subset of* $\boldsymbol{\omega}^\cap(t^\cap)$.

Proof. Let $t^\cap \in \boldsymbol{T}^\cap$ and $s \in \overline{\boldsymbol{\alpha}}^\cap(t^\cap)$.
By definition 3.4, $\exists t \in t^\cap, s \in \overline{\boldsymbol{\alpha}}(t)$.
From proposition 2.7, $s \in \boldsymbol{\omega}(t)$.
By definition 3.3, $s \in \boldsymbol{\omega}^\cap(t^\cap)$. □

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 $\boldsymbol{\alpha}$-valency of a CUT)**.** The set of *ASlots of CUTs* is defined through a mapping $\boldsymbol{\alpha}^\cap$ from $\boldsymbol{T}^\cap$ to $2^{\boldsymbol{S}_\mathcal{T}}$ Such that $\forall t^\cap \in \boldsymbol{T}^\cap, \boldsymbol{\alpha}^\cap(t^\cap) \stackrel{\text{def}}{=} \boldsymbol{\omega}^\cap(t^\cap) - \overline{\boldsymbol{\alpha}}^\cap(t^\cap)$. The number of ASlots of a CUT $t^\cap$ is denoted the $\boldsymbol{\alpha}$*-valency* of $t^\cap$, i.e., $valency_\alpha(t^\cap) \stackrel{\text{def}}{=} |\boldsymbol{\alpha}^\cap(t^\cap)|$.

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

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

Proof. From proposition 4.6,
$\forall t^\cap \in \boldsymbol{T}^\cap, \boldsymbol{\alpha}^\cap(t^\cap) = \boldsymbol{\omega}^\cap(t^\cap) - \overline{\boldsymbol{\alpha}}^\cap(t^\cap) \Leftrightarrow \forall t^\cap \in \boldsymbol{T}^\cap, \boldsymbol{\omega}^\cap(t^\cap) = \boldsymbol{\alpha}^\cap(t^\cap) \cup \overline{\boldsymbol{\alpha}}^\cap(t^\cap)$. □

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*: $\{t \in t^\cap \mid obj \in \boldsymbol{\omega}(t)\}$:

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

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:*

- $\boldsymbol{\omega}^\cap(\{t\}) = \boldsymbol{\omega}(t)$.
- $\overline{\boldsymbol{\alpha}}^\cap(\{t\}) = \overline{\boldsymbol{\alpha}}(t)$.
- $\boldsymbol{\alpha}^\cap(\{t\}) = \boldsymbol{\alpha}(t)$.
- *for all* $s \in \boldsymbol{\omega}(t)$, $\boldsymbol{\varsigma}^\cap_{\{t\}}(s) = \boldsymbol{\varsigma}_t(s)$

Proof. Let $t \in \boldsymbol{T}$.
1) $\boldsymbol{\omega}^\cap(\{t\}) = \bigcup_{t \in \{t\}} \boldsymbol{\omega}(t) = \boldsymbol{\omega}(t)$;
2) $\overline{\boldsymbol{\alpha}}^\cap(\{t\}) = \bigcup_{t \in \{t\}} \overline{\boldsymbol{\alpha}}(t) = \overline{\boldsymbol{\alpha}}(t)$;
3) $\boldsymbol{\alpha}^\cap(\{t\}) = \boldsymbol{\omega}^\cap(\{t\}) - \overline{\boldsymbol{\alpha}}^\cap(\{t\}) = \boldsymbol{\omega}(t) - \overline{\boldsymbol{\alpha}}(t) = \boldsymbol{\alpha}(t)$.
4) let $s \in \boldsymbol{\omega}(t)$. $t \in \{t \in t^\cap \mid s \in \boldsymbol{\omega}(t)\}$, so $\boldsymbol{\varsigma}^\cap_{\{t\}}(s) = \bigcup_{t \in \{t\} \mid s \in \boldsymbol{\omega}(t)} \boldsymbol{\varsigma}_t(s) = \boldsymbol{\varsigma}_t(s)$. □

3.3 Pre-order over CUTs

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

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

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

- for $i = 0$, $\boldsymbol{C}^{\frown}_0 \overset{\text{def}}{=} \boldsymbol{C}^{\frown}_{\lesssim} \cup \boldsymbol{C}^{\frown}_{\top} \cup \boldsymbol{C}^{\frown}_{\bot}$, where:

$$\boldsymbol{C}^{\frown}_{\lesssim} \overset{\text{def}}{=} \{(t^{\frown}_y, t^{\frown}_x) \in {\boldsymbol{T}^{\frown}}^2 \mid \forall t_y \in t^{\frown}_y, \exists t_x \in t^{\frown}_x : t_x \lesssim t_y\} \tag{5}$$

$$\boldsymbol{C}^{\frown}_{\top} \overset{\text{def}}{=} \{(\top^{\frown}, \varnothing)\} \tag{6}$$

$$\boldsymbol{C}^{\frown}_{\bot} \overset{\text{def}}{=} \{(\bot^{\frown}, t^{\frown}) \in {\boldsymbol{T}^{\frown}}^2 \mid t^{\frown} \in \bot^{\sqcap}_{asserted}\} \tag{7}$$

- for all $i > 0$, $\boldsymbol{C}^{\frown}_i \overset{\text{def}}{=} \boldsymbol{C}^{\frown}_{i-1} \cup \boldsymbol{C}^{\frown}_{\boldsymbol{\varsigma}\,i} \cup \boldsymbol{C}^{\frown}_{+i}$ where:

$$\boldsymbol{C}^{\frown}_{\boldsymbol{\varsigma}\,i} \overset{\text{def}}{=} \{(\bot^{\frown}, t^{\frown}) \in {\boldsymbol{T}^{\frown}}^2 \mid \exists s \in \boldsymbol{\omega}^{\frown}(t^{\frown}), (\bot^{\frown}, \boldsymbol{\varsigma}^{\frown}_{t^{\frown}}(s)) \in \boldsymbol{C}^{\frown}_{i-1}\} \tag{8}$$

$$\boldsymbol{C}^{\frown}_{+i} \overset{\text{def}}{=} \{(t^{\frown}_z, t^{\frown}_x) \in {\boldsymbol{T}^{\frown}}^2 \mid \exists t^{\frown}_y \in \boldsymbol{T}^{\frown}, (t^{\frown}_z, t^{\frown}_y) \in \boldsymbol{C}^{\frown}_{i-1} \text{ and } (t^{\frown}_y, t^{\frown}_x) \in \boldsymbol{C}^{\frown}_{i-1}\} \tag{9}$$

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

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

- $\boldsymbol{C}^{\frown}_{\lesssim}$ represents the natural extension of a pre-order over a set to a pre-order over its powerset;
- $\boldsymbol{C}^{\frown}_{\top}$ is introduced to flatten the top of the powerset so that $\top^{\frown}$ is the most generic CUT;
- $\boldsymbol{C}^{\frown}_{\bot}$ 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);
- $\boldsymbol{C}^{\frown}_{\boldsymbol{\varsigma}\,i}$ represents the fact that if a signature of a CUT for a given PSlot is absurd, then that CUT is absurd;
- $\boldsymbol{C}^{\frown}_{+i}$ transitively closes the set of comparisons $\boldsymbol{C}^{\frown}$.

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

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

Proof. Reflexivity: Let $t^{\frown} \in \boldsymbol{T}^{\frown}$.
$\forall t \in t^{\frown}, \exists t'(= t) \in t^{\frown} : t' \lesssim t$.
So $(t^{\frown}, t^{\frown}) \in \boldsymbol{C}^{\frown}_{\lesssim}$.
Transitivity: Let $t^{\frown}_x, t^{\frown}_y, t^{\frown}_z \in \boldsymbol{T}^{\frown}$ such that $(t^{\frown}_z, t^{\frown}_y) \in \boldsymbol{C}^{\frown}_{\lesssim}$, and $(t^{\frown}_y, t^{\frown}_x) \in \boldsymbol{C}^{\frown}_{\lesssim}$.
$\forall t_z \in t^{\frown}_z, \exists t_y \in t^{\frown}_y : t_y \lesssim t_z$, and $\forall t'_y \in t^{\frown}_y, \exists t_x \in t^{\frown}_x : t_x \lesssim t'_y$.
Let $t'_y = t_y$, then $\forall t_z \in t^{\frown}_z, \exists t_x \in t^{\frown}_x : t_x \lesssim t_z$,
and thus $(t^{\frown}_z, t^{\frown}_x) \in \boldsymbol{C}^{\frown}_{\lesssim}$. □

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

Proposition 3.5. $\mathbf{C}^{\frown}$ *is reflexive and transitive.*

Proof. As $\boldsymbol{C}^{\frown}_{\lesssim} \subseteq \boldsymbol{C}^{\frown}$ and $\boldsymbol{C}^{\frown}_{\lesssim}$ is reflexive, then $\boldsymbol{C}^{\frown}$ is reflexive.
Transitivity is obtained from $\boldsymbol{C}^{\frown}_{+}$ in the construction process of $\boldsymbol{C}^{\frown}$. □

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

Definition 3.8 (Pre-order relation over $\boldsymbol{T}^{\frown}$)**.** $\boldsymbol{C}^{\frown}$ is a binary relation over $\boldsymbol{T}^{\frown}$which is transitive and reflexive. So it defines a pre-order relation $\hat{\lesssim}$ over $\boldsymbol{T}^{\frown}$, i.e., $t^{\frown}_x \hat{\lesssim} t^{\frown}_y$ if and only if $(t^{\frown}_y, t^{\frown}_x) \in \boldsymbol{C}^{\frown}$.

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 $\boldsymbol{T}^{\frown}$:

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

- ***Extension of*** $\lesssim$ *For all* $t^{\frown}_x, t^{\frown}_y \in \mathbf{T}^{\frown}$*, if* $\forall t_y \in t^{\frown}_y, \exists t_x \in t^{\frown}_x : t_x \lesssim t_y$*, then* $t^{\frown}_x \hat{\lesssim} t^{\frown}_y$*;*
- ***Top CUT*** $\varnothing \hat{\lesssim} \top^{\frown}$*;*
- ***Asserted absurd types*** *For all* $t^{\frown} \in \bot^{\sqcap}_{asserted}$*,* $t^{\frown} \hat{\lesssim} \bot^{\frown}$*;*
- ***Absurd signatures*** *For all* $t^{\frown} \in \mathbf{T}^{\frown}$*, if there exists* $s \in \boldsymbol{\omega}^{\frown}(t^{\frown})$ *such that* $\boldsymbol{\varsigma}^{\frown}_{t^{\frown}}(s) \hat{\lesssim} \bot^{\frown}$*, then* $t^{\frown} \hat{\lesssim} \bot^{\frown}$*;*

Proof. Extension of $\lesssim$*:* For all $t^{\frown}_x, t^{\frown}_y \in \boldsymbol{T}^{\frown}$, if $\forall t_y \in t^{\frown}_y, \exists t_x \in t^{\frown}_x : t_x \lesssim t_y$, then $(t^{\frown}_y, t^{\frown}_x) \in \boldsymbol{C}^{\frown}_{\lesssim} \subseteq \boldsymbol{C}^{\frown}$, so $t^{\frown}_x \hat{\lesssim} t^{\frown}_y$.
Top CUT: We know that $(\top^{\frown}, \varnothing) \in \boldsymbol{C}^{\frown}$, so $\varnothing \hat{\lesssim} \top^{\frown}$.
Asserted absurd types: For all $t^{\frown} \in \bot^{\sqcap}_{asserted}$, $(\bot^{\frown}, t^{\frown}) \in \boldsymbol{C}^{\frown}_{\bot} \subseteq \boldsymbol{C}^{\frown}$, so $t^{\frown} \hat{\lesssim} \bot^{\frown}$.
Absurd signatures: For all $t^{\frown} \in \boldsymbol{T}^{\frown}$, if there exists $s \in \boldsymbol{\omega}^{\frown}(t^{\frown})$ such that $\boldsymbol{\varsigma}^{\frown}_{t^{\frown}}(s) \hat{\lesssim} \bot^{\frown}$, then $(\bot^{\frown}, \boldsymbol{\varsigma}^{\frown}_{t^{\frown}}(s)) \in \boldsymbol{C}^{\frown}$. There exists $n \in \mathbb{N}$ such that $(\bot^{\frown}, \boldsymbol{\varsigma}^{\frown}_{t^{\frown}}(s)) \in \boldsymbol{C}^{\frown}_{n}$, and thus for $n+1$, $(\bot^{\frown}, t^{\frown}) \in \boldsymbol{C}^{\frown}_{\boldsymbol{\varsigma}\ n+1} \subseteq \boldsymbol{C}^{\frown}$. So $t^{\frown} \hat{\lesssim} \bot^{\frown}$. □

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 $\boldsymbol{T}^{\frown}$ with the inclusion relation $(\boldsymbol{T}^{\frown}, \subseteq)$ and the powerset of $\boldsymbol{T}^{\frown}$ with the specialization relation $(\boldsymbol{T}^{\frown}, \hat{\lesssim})$. The morphism f that maps one onto the other and such that $f(t^{\frown}) = t^{\frown}$ is anti-monotonic:

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

Proof. Let $t^{\frown}_x, t^{\frown}_y \in \boldsymbol{T}^{\frown}$ such that $t^{\frown}_x \subseteq t^{\frown}_y$. Then $\forall t_x \in t^{\frown}_x, t_x \in t^{\frown}_y$.
Thus $\forall t_x \in t^{\frown}_x, \exists t_y (= t_x) \in t^{\frown}_y : t_y \lesssim t_x$, so by proposition 3.6, $t^{\frown}_y \hat{\lesssim} t^{\frown}_x$. □

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

Proposition 3.8. $\top^{\frown}$ *and* $\varnothing$ *are maximal elements in* $\mathbf{T}^{\frown}$*, and* $\mathbf{T}$*,* $\bot^{\frown}$*, and* $\bot^{\sim}$ *are minimal elements in* $(\mathbf{T}^{\frown}, \hat{\lesssim})$*.*

Proof. Let $t^\cap \in \boldsymbol{T}^\cap$.
$\varnothing$: $\varnothing \subseteq t^\cap$, so using proposition 3.7: $t^\cap \mathrel{\hat{\lesssim}} \varnothing$.
$\top^\cap$: Let $t^\cap \neq \varnothing$. We know that for all $t \in \boldsymbol{T}$, $t \lesssim \top$, so for $\top$ in $\top^\cap$, $\exists t \in t^\cap : t \lesssim \top$.
So using proposition 3.6, $t^\cap \mathrel{\hat{\lesssim}} \top^\cap$.
As we moreover know that $\varnothing \mathrel{\hat{\lesssim}} \top^\cap$, then for all $t^\cap \in \boldsymbol{T}^\cap$, $t^\cap \mathrel{\hat{\lesssim}} \top^\cap$.
$\boldsymbol{T}$: $t^\cap \subseteq \boldsymbol{T}$, so using proposition 3.7: $\boldsymbol{T} \mathrel{\hat{\lesssim}} t^\cap$.
$\bot^\cap$: Let $t^\cap \neq \varnothing$. We know that for all $t \in \boldsymbol{T}$, $\bot \lesssim t$, so for all $t \in t^\cap, \bot \lesssim t$.
So using proposition 3.6, $\bot^\cap \mathrel{\hat{\lesssim}} t^\cap$.
As we moreover know that $\varnothing$ is a maximal element, $\bot^\cap \mathrel{\hat{\lesssim}} \varnothing$, then for all $t^\cap \in \boldsymbol{T}^\cap$, $\bot^\cap \mathrel{\hat{\lesssim}} t^\cap$.
$\bot^\sim$: We know that $\bot \in \bot^\sim$, so item above also applies for $\bot^\sim$, i.e.,
for $t^\cap \neq \varnothing$, $\forall t \in t^\cap, \exists t'(=\bot) \in \bot^\sim : t' \lesssim t$, and $\bot^\sim \mathrel{\hat{\lesssim}} \varnothing$
So for all $t^\cap \in \boldsymbol{T}^\cap$, $\bot^\sim \mathrel{\hat{\lesssim}} t^\cap$. □

We introduce the natural equivalence relation $\mathrel{\hat{\simeq}}$ defined by $t_x^\cap \mathrel{\hat{\simeq}} t_y^\cap \Leftrightarrow t_x^\cap \mathrel{\hat{\lesssim}} t_y^\cap$ and $t_y^\cap \mathrel{\hat{\lesssim}} t_x^\cap$. Elements in the flattened bottom of the pre-ordered set $(\boldsymbol{T}^\cap, \mathrel{\hat{\lesssim}})$ 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 $\bot^\sqcap$ and is the set: $\bot^\sqcap \stackrel{\text{def}}{=} \{t^\cap \in \boldsymbol{T}^\cap \mid t^\cap \mathrel{\hat{\simeq}} \bot^\cap\}$.

Naturally, the prime absurd CUT, $\bot^\cap$, is absurd.

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

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

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 $\mathrel{\hat{\lesssim}}$ enables PUT $\{\langle\text{rain-hail}\rangle\}$ to be absurd as it should be.

Finally, if one asserts that $\top^\cap \in \bot^\sqcap_{asserted}$, then the whole hierarchy collapses and $\bot^\sqcap = \boldsymbol{T}^\cap$. Same goes if $t^\cap \in \boldsymbol{T}^\cap$ such that $\top^\cap \mathrel{\hat{\lesssim}} t^\cap$, and $s \in \boldsymbol{\omega}^\cap(t^\cap)$ such that $\boldsymbol{\varsigma}^\cap_{t^\cap}(s) \mathrel{\hat{\lesssim}} \bot^\cap$. Then from proposition 3.6, $t^\cap \mathrel{\hat{\lesssim}} \bot^\cap$, and by the pre-order transitivity, $\bot^\sqcap = \boldsymbol{T}^\cap$. 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* $\mathcal{T}^{\cap} = (T_{declared}, C_{asserted}, \bot^{\sqcap}_{asserted}, \boldsymbol{S}_{\mathcal{T}}, root, Hiders, \{\boldsymbol{\varsigma}_t\}_{t \in \boldsymbol{T}})$, is composed of:

- $T_{declared}$ a set of declared PUTs;
- $C_{asserted}$ a set of asserted comparisons;
- $\bot^{\sqcap}_{asserted}$ a set of asserted absurd CUTs;
- $\boldsymbol{S}_{\mathcal{T}}$ 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;
- $\{\boldsymbol{\varsigma}_t\}_{t \in \boldsymbol{T}}$ 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 $t_x^\frown, t_y^\frown \in \boldsymbol{T}^\frown$ and according to the construction of the pre-order over CUTs, we introduce in §4.1 the necessary condition to have $t_x^\frown \mathrel{\hat{\lesssim}} t_y^\frown$.

Second, now that the pre-order $\mathrel{\hat{\lesssim}}$ 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 $\mathrel{\hat{\lesssim}}$ 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 $t_x^\frown, t_y^\frown \in \boldsymbol{T}^\frown$ comparable may be stated as follows:

If $t_x^\frown \mathrel{\hat{\lesssim}} t_y^\frown$, then at least one of the following is true:

- $t_x^\frown$ is absurd;
- $t_x^\frown$ is the empty set (and thus $t_y^\frown$ is a maximal element of $\boldsymbol{T}^\frown$);
- $t_x^\frown$ is naturally more specific than $t_y^\frown$, 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 $t_x^\frown, t_y^\frown \in \mathbf{T}^\frown$. if $t_x^\frown \mathrel{\hat{\lesssim}} t_y^\frown$, then at least one of the following is true:*

- $t_x^\frown \mathrel{\hat{\simeq}} \bot^\frown$;
- $t_x^\frown = \varnothing$ *(and thus $\top^\frown \mathrel{\hat{\simeq}} t_y^\frown$);*
- $\forall t_y \in t_y^\frown, \exists t_x \in t_x^\frown : t_x \lesssim t_y$ *(i.e., $(t_y^\frown, t_x^\frown) \in \mathbf{C}_\lesssim^\frown$).*

Proof. Let $t_x^\frown, t_y^\frown \in \boldsymbol{T}^\frown$.
As $\top^\frown$ (resp. $\bot^\frown$) is a maximal (resp. minimal) element of $\boldsymbol{T}^\frown$,
$\top^\frown \mathrel{\hat{\lesssim}} t_y^\frown$ implies $\top^\frown \mathrel{\hat{\simeq}} t_y^\frown$ (resp. $t_x^\frown \mathrel{\hat{\lesssim}} \bot^\frown$ implies $t_x^\frown \mathrel{\hat{\simeq}} \bot^\frown$).
So it is sufficient to prove that one of the following is true:
i) $t_x^\frown \mathrel{\hat{\lesssim}} \bot^\frown$, ii) $t_x^\frown = \varnothing$ (and thus $\top^\frown \mathrel{\hat{\lesssim}} t_y^\frown$), or iii) $(t_y^\frown, t_x^\frown) \in \boldsymbol{C}_\lesssim^\frown$.

$\boldsymbol{C}^\frown$ is the limit the sequence $(\boldsymbol{C}_n^\frown)$, so let $n \in \mathbb{N}$ be such that $(t_y^\frown, t_x^\frown) \in \boldsymbol{C}_n^\frown$ and $(t_y^\frown, t_x^\frown) \notin \boldsymbol{C}_{n-1}^\frown$.
We assume that $t_x^\frown \mathrel{\hat{\not\lesssim}} \bot^\frown$, $(t_y^\frown, t_x^\frown) \notin \boldsymbol{C}_\lesssim^\frown$, and that either $\top^\frown \mathrel{\hat{\not\lesssim}} t_y^\frown$ or $t_x^\frown \neq \varnothing$.

1) If $(t_y^\frown, t_x^\frown) \in \boldsymbol{C}_{\boldsymbol{\varsigma}\,n}^\frown$, then $t_x^\frown \mathrel{\hat{\lesssim}} \bot^\frown$.
This is impossible, so $(t_y^\frown, t_x^\frown) \in \boldsymbol{C}_{+n}^\frown$ and there exists $t^\frown \in \boldsymbol{T}^\frown$ such that $(t_y^\frown, t^\frown) \in \boldsymbol{C}_{n-1}^\frown$ and $(t^\frown, t_x^\frown) \in \boldsymbol{C}_{n-1}^\frown$.
As $\boldsymbol{C}_{n-1}^\frown \subseteq \boldsymbol{C}^\frown$, $t_x^\frown \mathrel{\hat{\lesssim}} t^\frown$.
If $t^\frown \mathrel{\hat{\lesssim}} \bot^\frown$, then $t_x^\frown \mathrel{\hat{\lesssim}} \bot^\frown$ which is impossible.
So we know that $t^\frown \mathrel{\hat{\not\lesssim}} \bot^\frown$.

2) From proposition 3.4, we know that $\boldsymbol{C}^{\frown}_{\hat{\lesssim}}$ is transitive. So either 2a) $(t^{\frown}, t^{\frown}_x) \notin \boldsymbol{C}^{\frown}_{\hat{\lesssim}}$, or 2b) $(t^{\frown}_y, t^{\frown}) \notin \boldsymbol{C}^{\frown}_{\hat{\lesssim}}$.

2a) In this case, consider $m \leqslant n-1$ such that $(t^{\frown}, t^{\frown}_x) \in \boldsymbol{C}^{\frown}_m$ and $(t^{\frown}, t^{\frown}_x) \notin \boldsymbol{C}^{\frown}_{m-1}$.
We still have $t^{\frown}_x \hat{\not\lesssim} \bot^{\frown}$ and we have now $(t^{\frown}, t^{\frown}_x) \notin \boldsymbol{C}^{\frown}_{\hat{\lesssim}}$.

2b) In this case, consider $m \leqslant n-1$ such that $(t^{\frown}_y, t^{\frown}) \in \boldsymbol{C}^{\frown}_m$ and $(t^{\frown}_y, t^{\frown}) \notin \boldsymbol{C}^{\frown}_{m-1}$.
We just showed in 1) that $t^{\frown} \hat{\not\lesssim} \bot^{\frown}$ and we have now $(t^{\frown}_y, t^{\frown}) \notin \boldsymbol{C}^{\frown}_{\hat{\lesssim}}$.

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 $t^{\frown}_a$ and $t^{\frown}_b$): $(t^{\frown}_b, t^{\frown}_a) \in \boldsymbol{C}^{\frown}_0$, $t^{\frown}_a \hat{\not\lesssim} \bot^{\frown}$ so $(\bot^{\frown}, t^{\frown}_a) \notin \boldsymbol{C}^{\frown}_{\bot}$, and $(t^{\frown}_b, t^{\frown}_a) \notin \boldsymbol{C}^{\frown}_{\hat{\lesssim}}$. So $(t^{\frown}_b, t^{\frown}_a) \in \boldsymbol{C}^{\frown}_{\top}$.
Neither $\top^{\frown} \hat{\not\lesssim} t^{\frown}_b$ nor $t^{\frown}_b \neq \varnothing$ is compatible with this last statement, so we highlighted a contradiction.

As a conclusion, if $t^{\frown}_x \hat{\lesssim} t^{\frown}_y$, it is impossible that all of the following is false :
i) $t^{\frown}_x \hat{\lesssim} \bot^{\frown}$, ii) $t^{\frown}_x = \varnothing$, iii) $(t^{\frown}_y, t^{\frown}_x) \in \boldsymbol{C}^{\frown}_{\hat{\lesssim}}$. □

4.2 Properties of CUT Slots and Signatures

Now that the pre-order $\hat{\lesssim}$ 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 $t^{\frown}_x = \{(\text{fall}), (\text{past})\}$ and $t^{\frown}_y = \{(\text{move})\}$, then $t^{\frown}_x \hat{\lesssim} t^{\frown}_y$, and in every linguistic situation in which a linguistic unit of type $t^{\frown}_x$ 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 $\downarrow^{\frown} root(s)$ be the smallest lower set of $\mathbf{T}^{\frown}$ that contains $\{root(s)\}$:*

$$\downarrow^{\frown} root(s) \stackrel{def}{=} \{t^{\frown} \in \mathbf{T}^{\frown} \mid t^{\frown} \hat{\lesssim} \{root(s)\}\} \tag{10}$$

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

$$\{t^{\frown} \in \mathbf{T}^{\frown} \mid s \in \boldsymbol{\omega}^{\frown}(t^{\frown})\} \cup \bot^{\sqcap} \backslash \varnothing = \downarrow^{\frown} root(s) \backslash \varnothing \tag{11}$$

Proof. Let $s \in \boldsymbol{S}_{\mathcal{T}}$.
$\subseteq_1$: Let $t^{\frown} \in \{t^{\frown} \in \boldsymbol{T}^{\frown} \mid s \in \boldsymbol{\omega}^{\frown}(t^{\frown})\}$.
By definition 3.3, $\exists t \in t^{\frown}, s \in \boldsymbol{\omega}(t)$. So $t^{\frown} \neq \varnothing$.
On the other hand, from definition 2.6, $t \lesssim root(s)$.
By proposition 3.6, $t^{\frown} \hat{\lesssim} \{root(s)\}$ so $t^{\frown} \in \downarrow^{\frown} root(s)$.
So $t^{\frown} \in \downarrow^{\frown} root(s) \backslash \varnothing$, and $\{t^{\frown} \in \boldsymbol{T}^{\frown} \mid s \in \boldsymbol{\omega}^{\frown}(t^{\frown})\} \subseteq \downarrow^{\frown} root(s) \backslash \varnothing$.
$\subseteq_2$: Let $t^{\frown} \in \bot^{\sqcap} \backslash \varnothing$. $t^{\frown} \hat{\lesssim} \bot^{\frown}$ which is a minimum element in $\boldsymbol{T}^{\frown}$.
So $t^{\frown} \hat{\lesssim} \{root(s)\}$ and $t^{\frown} \in \downarrow^{\frown} root(s)$ and $t^{\frown} \in \downarrow^{\frown} root(s) \backslash \varnothing$.
So $\bot^{\sqcap} \backslash \varnothing \subseteq \downarrow^{\frown} root(s) \backslash \varnothing$.

$\supseteq$: Let $t^\frown \in {\downarrow^\frown} root(s) \backslash \varnothing$. $t^\frown \mathrel{\hat{\lesssim}} \{root(s)\}$.
We use proposition 4.1, so at least one of the following is true:
i) $t^\frown \mathrel{\hat{\lesssim}} \bot^\frown$: then $t^\frown \in \bot^\sqcap$. And as $t^\frown \neq \varnothing$, $t^\frown \in \bot^\sqcap \backslash \varnothing$.
ii) $t^\frown = \varnothing$: impossible case.
iii) $\exists t \in t^\frown : t \lesssim root(s)$: then using definition 2.6, $\exists t \in t^\frown : s \in \boldsymbol{\omega}(t)$, and by definition 3.3, $s \in \boldsymbol{\omega}^\frown(t^\frown)$. So $t^\frown \in \{t^\frown \in \boldsymbol{T}^\frown \mid s \in \boldsymbol{\omega}^\frown(t^\frown)\}$. □

One may have noticed that two kinds of CUTs are not considered here: the CUT $\varnothing$, and any absurd CUT $t^\frown \in \bot^\sqcap$:

- the CUT $\varnothing$: a problem would arise for instance with $s \in \boldsymbol{S}_\mathcal{T}$ such that $root(s) = \top$. By construction of the pre-order $\hat{\lesssim}$, $\varnothing \mathrel{\hat{\lesssim}} \top^\frown$, yet $\varnothing$ has no PSlot ($\boldsymbol{\omega}^\frown(\varnothing) = \bigcup_{t \in \varnothing} \boldsymbol{\omega}(t) = \varnothing$). So $s \notin \boldsymbol{\omega}^\frown(\varnothing)$ and some PSlots may not be inherited from the CUT $\varnothing$;
- any absurd CUT $t^\frown \in \bot^\sqcap$: problems arise for any $s \in \boldsymbol{S}_\mathcal{T}$ such that $s \notin \boldsymbol{\omega}^\frown t^\frown$. By construction of the pre-order $\hat{\lesssim}$, $t^\frown \mathrel{\hat{\lesssim}} \bot^\frown$, yet, from propositions 2.4 and 3.3, $\boldsymbol{\omega}^\frown(\bot^\frown) = \boldsymbol{S}_\mathcal{T}$. 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 $\varnothing$, i.e., $\forall t_x^\frown \in \mathbf{T}^\frown \backslash \varnothing - \bot^\sqcap$, and $t_y^\frown \in \mathbf{T}^\frown$,*

$$t_x^\frown \mathrel{\hat{\lesssim}} t_y^\frown \Rightarrow \boldsymbol{\omega}^\frown(t_y^\frown) \subseteq \boldsymbol{\omega}^\frown(t_x^\frown) \tag{12}$$

Proof. Let $t_x^\frown \in \boldsymbol{T}^\frown \backslash \varnothing - \bot^\sqcap$ and $t_y^\frown \in \boldsymbol{T}^\frown$ such that $t_x^\frown \mathrel{\hat{\lesssim}} t_y^\frown$.
Let $s \in \boldsymbol{\omega}^\frown(t_y^\frown)$. By definition 3.3, $\exists t \in t_y^\frown, s \in \boldsymbol{\omega}(t)$.
We use proposition 4.1, but as $t_x^\frown \notin \bot^\sqcap$ and $t_x^\frown \neq \varnothing$, $t_x^\frown \mathrel{\hat{\lesssim}} t_y^\frown$ simply implies: $\forall t_y \in t_y^\frown, \exists t_x \in t_x^\frown : t_x \lesssim t$. So $\exists t_x \in t_x^\frown : t_x \lesssim t_y$.
Using proposition 2.3, $s \in \boldsymbol{\omega}(t_x)$, and using definition 3.3, $s \in \boldsymbol{\omega}^\frown(t_x^\frown)$. □

Hidden Slots Next, consider the CUT $\{(\text{rain})\}$. PSlot *obj* is hidden for (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 $\varnothing$, and any absurd CUT $t^\frown \in \bot^\sqcap$. The following property thus holds:

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

$${\downarrow^\frown} Hiders(s) \stackrel{def}{=} \{t^\frown \in \mathbf{T}^\frown \mid \exists t_h \in Hiders(s) : t^\frown \mathrel{\hat{\lesssim}} \{t_h\}\} \tag{13}$$

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

$$\{t^\frown \in \mathbf{T}^\frown \mid s \in \overline{\boldsymbol{\alpha}}^\frown(t^\frown)\} \cup \bot^\sqcap \backslash \varnothing = {\downarrow^\frown} Hiders(s) \backslash \varnothing \tag{14}$$

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

$\subseteq_1$: Let $t^\cap \in \{t^\cap \in \boldsymbol{T}^\cap \mid s \in \overline{\boldsymbol{\alpha}}^\cap(t^\cap)\}$.
a) By definition 3.4, $\exists t \in t^\cap, s \in \overline{\boldsymbol{\alpha}}(t)$. So $t^\cap \neq \varnothing$.
b) On the other hand, from definition 2.7, $\exists t_h \in Hiders(s), t \lesssim t_h$. Thus using proposition 3.6 $t^\cap \mathrel{\hat{\lesssim}} \{t_h\}$, and $t^\cap \in \downarrow^\cap Hiders(s)$.
So $t^\cap \in \downarrow^\cap Hiders(s)\backslash\varnothing$, and $\{t^\cap \in \boldsymbol{T}^\cap \mid s \in \overline{\boldsymbol{\alpha}}^\cap(t^\cap)\} \subseteq \downarrow^\cap Hiders(s)\backslash\varnothing$.

$\subseteq_2$: Let $t^\cap \in \bot^\sqcap\backslash\varnothing$.
$t^\cap \mathrel{\hat{\lesssim}} \bot^\cap$ which is a minimum element in $\boldsymbol{T}^\cap$.
So $\forall t_h \in Hiders(s), t^\cap \mathrel{\hat{\lesssim}} \{t_h\}$, i.e., $t^\cap \in \downarrow^\cap Hiders(s)$.
As we chose that $t^\cap$ can't be $\varnothing$, then $t^\cap \in \downarrow^\cap Hiders(s)\backslash\varnothing$.
So $\bot^\sqcap\backslash\varnothing \subseteq \downarrow^\cap Hiders(s)\backslash\varnothing$.

$\supseteq$: Let $t^\cap \in \downarrow^\cap Hiders(s)\backslash\varnothing$.
$\exists t_h \in Hiders(s), t^\cap \mathrel{\hat{\lesssim}} \{t_h\}$.
We use proposition 4.1, so at least one of the following is true:
i) $t^\cap \mathrel{\hat{\cong}} \bot^\cap$: then $t^\cap \in \bot^\sqcap$ and $t^\cap \in \bot^\sqcap\backslash\varnothing$.
ii) $t^\cap = \varnothing$: impossible case.
iii) $\exists t \in t^\cap : t \lesssim t_h$: then by definition 2.7 $\exists t \in t^\cap : s \in \overline{\boldsymbol{\alpha}}(t)$, and by definition 3.4, $s \in \overline{\boldsymbol{\alpha}}^\cap(t^\cap)$.
So $t^\cap \in \{t^\cap \in \boldsymbol{T}^\cap \mid s \in \overline{\boldsymbol{\alpha}}^\cap(t^\cap)\}$
So $t^\cap \in \{t^\cap \in \boldsymbol{T}^\cap \mid s \in \overline{\boldsymbol{\alpha}}^\cap(t^\cap)\} \cup \bot^\sqcap\backslash\varnothing$. □

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* $\varnothing$*, i.e.,* $\forall t_x^\cap \in \mathbf{T}^\cap\backslash\varnothing - \bot^\sqcap$ *and* $t_y^\cap \in \mathbf{T}^\cap$*,*

$$t_x^\cap \mathrel{\hat{\lesssim}} t_y^\cap \Rightarrow \overline{\boldsymbol{\alpha}}^\cap(t_x^\cap) \subseteq \overline{\boldsymbol{\alpha}}^\cap(t_y^\cap) \tag{15}$$

Proof. Let $t_x^\cap \in \boldsymbol{T}^\cap\backslash\varnothing - \bot^\sqcap$ and $t_y^\cap \in \boldsymbol{T}^\cap$ such that $t_x^\cap \mathrel{\hat{\lesssim}} t_y^\cap$.
Let $s \in \overline{\boldsymbol{\alpha}}^\cap(t_y^\cap)$. By definition 3.4, $\exists t \in t_y^\cap, s \in \overline{\boldsymbol{\alpha}}(t)$.
From proposition 4.1 and the facts that $t_x^\cap \notin \bot^\sqcap$ and $t_x^\cap \neq \varnothing$, we know that $t_x^\cap \mathrel{\hat{\lesssim}} t_y^\cap$ 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, $s \in \overline{\boldsymbol{\alpha}}(t_x)$, and using definition 3.4, $s \in \overline{\boldsymbol{\alpha}}^\cap(t_x^\cap)$. □

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

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

$$\begin{aligned}
&\{t^\cap \in \mathbf{T}^\cap \mid s \in \boldsymbol{\alpha}^\cap(t^\cap)\} \cup \bot^\sqcap\backslash\varnothing \\
&= (\downarrow^\cap root(s)\backslash\varnothing - \downarrow^\cap Hiders(s)\backslash\varnothing) \cup \bot^\sqcap\backslash\varnothing \\
&= \left\{t^\cap \in \mathbf{T}^\cap \mid t^\cap \mathrel{\hat{\lesssim}} \{root(s)\} \text{ and } \forall t_h \in Hiders(s), t^\cap \mathrel{\hat{\not\lesssim}} \{t_h\}\right\} \cup \bot^\sqcap\backslash\varnothing
\end{aligned}$$

Proof. We use propositions 2.10, 4.2 and 4.4:

$$
\begin{aligned}
&\{t^\frown \in \boldsymbol{T}^\frown \mid s \in \boldsymbol{\alpha}^\frown(t^\frown)\} \cup \bot^\sqcap \backslash \varnothing \\
&= \{t^\frown \in \boldsymbol{T}^\frown \mid s \in \boldsymbol{\omega}^\frown(t^\frown) - \overline{\boldsymbol{\alpha}}^\frown(t^\frown)\} \cup \bot^\sqcap \backslash \varnothing \\
&= (\{t^\frown \in \boldsymbol{T}^\frown \mid s \in \boldsymbol{\omega}^\frown(t^\frown)\} - \{t^\frown \in \boldsymbol{T}^\frown \mid s \in \overline{\boldsymbol{\alpha}}^\frown(t^\frown)\}) \cup \bot^\sqcap \backslash \varnothing \\
&= ((\{t^\frown \in \boldsymbol{T}^\frown \mid s \in \boldsymbol{\omega}^\frown(t^\frown)\} \cup \bot^\sqcap \backslash \varnothing) - (\{t^\frown \in \boldsymbol{T}^\frown \mid s \in \overline{\boldsymbol{\alpha}}^\frown(t^\frown)\} \cup \bot^\sqcap \backslash \varnothing)) \cup \bot^\sqcap \backslash \varnothing \\
&= (\downarrow^\frown root(s) \backslash \varnothing - \downarrow^\frown Hiders(s) \backslash \varnothing) \cup \bot^\sqcap \backslash \varnothing
\end{aligned}
$$

□

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* $t_x^\frown \in \mathbf{T}^\frown \backslash \varnothing - \bot^\sqcap$, $t_y^\frown \in \mathbf{T}^\frown$ *and* $s \in \mathbf{S}_\mathcal{T}$ *such that* $s \in \boldsymbol{\omega}^\frown(t_y^\frown)$,

$$t_x^\frown \mathrel{\hat{\lesssim}} t_y^\frown \Rightarrow \boldsymbol{\varsigma}_{t_x^\frown}^\frown(s) \mathrel{\hat{\lesssim}} \boldsymbol{\varsigma}_{t_y^\frown}^\frown(s) \tag{16}$$

Proof. Let $t_x^\frown \in \boldsymbol{T}^\frown \backslash \varnothing - \bot^\sqcap$ and $t_y^\frown \in \boldsymbol{T}^\frown$ and $s \in \boldsymbol{S}_\mathcal{T}$ such that $t_x^\frown \mathrel{\hat{\lesssim}} t_y^\frown$ and $s \in \boldsymbol{\omega}^\frown(t_y^\frown)$.
1) Let $t_a \in \boldsymbol{\varsigma}_{t_y^\frown}^\frown(s)$. Using definition 3.6, $\exists t_b \in t_y^\frown : s \in \boldsymbol{\omega}(t_b)$ and $t_a \in \boldsymbol{\varsigma}_{t_b}(s)$.
2) We use $t_x^\frown \mathrel{\hat{\lesssim}} t_y^\frown$ and proposition 4.1.
As $t_x^\frown \notin \bot^\sqcap$ and $t_x^\frown \neq \varnothing$, we know that $\forall t_y \in t_y^\frown, \exists t_x \in t_x^\frown : t_x \lesssim t_y$.
We restrict those t_y to be such that $s \in \boldsymbol{\omega}(t_y)$.
We use definitions 2.9 and proposition 3.6: $\forall t_y \in t_y^\frown$ such that $s \in \boldsymbol{\omega}(t_y)$, there exists $t_x \in t_x^\frown$ such that $\forall t' \in \boldsymbol{\varsigma}_{t_y}(s), \exists t \in \boldsymbol{\varsigma}_{t_x}(s) : t \lesssim t'$
3) This is rewritten as: $\forall t_y \in t_y^\frown$ such that $s \in \boldsymbol{\omega}(t_y), \forall t' \in \boldsymbol{\varsigma}_{t_y}(s), \exists t_x, t$ such that $t_x \in t_x^\frown, t \in \boldsymbol{\varsigma}_{t_x}(s)$ and $t \lesssim t'$
4) combining 1) and 3) with $t_y = t_b$, and $t' = t_a$, one obtains:
$\exists t_b \in t_y^\frown$ such that $s \in \boldsymbol{\omega}(t_b)$ and $t_a \in \boldsymbol{\varsigma}_{t_b}(s), \exists t_x, t$ such that $t_x \in t_x^\frown, t \in \boldsymbol{\varsigma}_{t_x}(s)$ and $t \lesssim t_a$
5) So there exists t and t_x such that $t_x \in t_x^\frown$ and $t \in \boldsymbol{\varsigma}_{t_x}(s)$ and $t \lesssim t_a$,
6) using definition 3.6, $t \in \boldsymbol{\varsigma}_{t_x^\frown}^\frown(s)$. So $\forall t_a \in \boldsymbol{\varsigma}_{t_y^\frown}^\frown(s), \exists t \in \boldsymbol{\varsigma}_{t_x^\frown}^\frown(s) : t \lesssim t_a$, and using proposition 3.6, $\boldsymbol{\varsigma}_{t_x^\frown}^\frown(s) \mathrel{\hat{\lesssim}} \boldsymbol{\varsigma}_{t_y^\frown}^\frown(s)$. □

4.3 CUT Equivalence Class Sets

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

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

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

Proposition 4.8. *$\top^\sqcap$ and $\bot^\sqcap$ are respectively the greatest element and the least element of $\mathbf{T}^\sqcap$.*

Proof. $\top^\sqcap$: From proposition 3.8, $\top^\frown$ is a maximal element for $\hat{\lesssim}$.
So $\forall t^\frown \in \boldsymbol{T}^\frown, t^\frown \hat{\lesssim} \top^\frown$. So $\forall t^\sqcap \in \boldsymbol{T}^\sqcap, t^\sqcap \hat{\lesssim}^\sqcap \top^\sqcap$, and $\top^\sqcap$ is the greatest element of $\boldsymbol{T}^\sqcap$.
$\bot^\sqcap$: From proposition 3.8, $\bot^\frown$ is a minimal element for $\hat{\lesssim}$. So $\forall t^\frown \in \boldsymbol{T}^\frown, \bot^\frown \hat{\lesssim} t^\frown$.
So $\forall t^\sqcap \in \boldsymbol{T}^\sqcap, \bot^\sqcap \hat{\lesssim}^\sqcap t^\sqcap$, so $\bot^\sqcap$ is the least element of $\boldsymbol{T}^\sqcap$. □

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

Proposition 4.9. *Let $t^\sqcap \in \mathbf{T}^\sqcap$, and $t_x^\frown, t_y^\frown \in t^\sqcap$. Then $t_x^\frown \cup t_y^\frown \in t^\sqcap$.*

Proof. Let $t^\sqcap \in \boldsymbol{T}^\sqcap$, and $t_x^\frown, t_y^\frown \in t^\sqcap$. We will prove that $t_x^\frown \cup t_y^\frown \hat{\simeq} t_x^\frown$.
$\hat{\lesssim}$: We know from proposition 3.7 that $t_x^\frown \subseteq t_x^\frown \cup t_y^\frown$ implies $t_x^\frown \cup t_y^\frown \hat{\lesssim} t_x^\frown$.
$\hat{\gtrsim}$: We know that $t_x^\frown \hat{\lesssim} t_y^\frown$. From proposition 3.6, at least one of the followings holds:
i) $t_x^\frown \hat{\lesssim} \bot^\frown$: then $t_x^\frown \cup t_y^\frown \hat{\simeq} t_x^\frown \hat{\simeq} \bot^\frown$;
ii) $t_x^\frown = \varnothing$: then $t_x^\frown \cup t_y^\frown = t_y^\frown \hat{\simeq} t_x^\frown$;
iii) $\forall t_y \in t_y^\frown, \exists t_x \in t_x^\frown : t_x \lesssim t_y$: So $\forall t \in t_x^\frown \cup t_y^\frown, \exists t_x \in t_x^\frown : t_x \lesssim t$, and $t_x^\frown \hat{\lesssim} t_x^\frown \cup t_y^\frown$. □

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

Lemma 4.10. *Let $t^\frown \in \mathbf{T}^\frown$ and $t \in \mathbf{T}$ such that $\exists t' \in t^\frown, t' \lesssim t$. Then $t^\frown \cup \{t\} \hat{\simeq} t^\frown$.*

Proof. Let $t^\frown \in \boldsymbol{T}^\frown$ and $t \in \boldsymbol{T}$ such that $\forall t' \in t^\frown, t' \lesssim t$.
$\hat{\lesssim}$: $t_y^\frown \subseteq t_y^\frown \cup \{t\}$ so $t^\frown \cup \{t\} \hat{\lesssim} t^\frown$;
$\hat{\gtrsim}$: $t^\frown \neq \varnothing$. Let $t' \in t^\frown \cup \{t\}$.
If $t' \in t^\frown$, then there exists $t''(= t') \in t^\frown : t'' \lesssim t'$.
If $t' = t$, then there exists $t'' \in t^\frown t'' \lesssim t'$.
So $\forall t' \in t^\frown \cup \{t\}, \exists t'' \in t^\frown : t'' \lesssim t'$, and $t^\frown \cup \{t\} \hat{\gtrsim} t^\frown$. □

4.4 Maximal CUTs

CUTs enable a first inference mechanism, which is type inference. If a CUT $t^{\frown}_x$ is lower than $t^{\frown}_y$, then any instance of $t^{\frown}_x$ is also an instance of every PUT in $t^{\frown}_y$. 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 $t^{\frown}$ the union of all CUTs in $[t^{\frown}]$:

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

The set of maximal CUTs $\boldsymbol{T}^{\frown}_{max}$ is then denoted $\boldsymbol{T}^{\frown}_{max}$, and defined as follows:

$$\boldsymbol{T}^{\frown}_{max} = \{\Uparrow t^{\frown} \mid t^{\frown} \in \boldsymbol{T}^{\frown}\} \tag{17}$$

In the rest of this section we will list useful facts about the maximization operator $\Uparrow$ and the set of maximal CUTs $\boldsymbol{T}^{\frown}_{max}$.

First, the maximization operator preserves the equivalence class. This means that a CUT $t^{\frown}$ and its maximized CUT $\Uparrow t^{\frown}$ 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 $t^{\sqcap} \in \mathbf{T}^{\sqcap}$, if $t^{\frown} \in t^{\sqcap}$, then $\Uparrow t^{\frown} \in t^{\sqcap}$. Or equivalently, $\forall t^{\frown} \in \mathbf{T}^{\frown}, \Uparrow t^{\frown} \mathrel{\hat{\simeq}} t^{\frown}$.*

Proof. Let $t^{\sqcap} \in \boldsymbol{T}^{\sqcap}$, and $t^{\frown} \in t^{\sqcap}$. By definition, $\Uparrow t^{\frown}$ is the union of all CUT in $t^{\sqcap}$.
We know from proposition 4.9 that $t^{\sqcap}$ is closed under the union operation, so $\Uparrow t^{\frown} \in t^{\sqcap}$. □

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 $\mathrel{\hat{\simeq}}$, i.e., if $t^{\frown}_x \mathrel{\hat{\simeq}} t^{\frown}_y$, then $\Uparrow t^{\frown}_x = \Uparrow t^{\frown}_y$*

Proof. By definition, if $t^{\frown}_x \mathrel{\hat{\simeq}} t^{\frown}_y$, then $[t^{\frown}_x] = [t^{\frown}_y]$, and $\Uparrow t^{\frown}_x = \Uparrow t^{\frown}_y$ □

Thus, there is a one-to-one correspondence between the set of a set of equivalence CUTs class sets $\boldsymbol{T}^{\sqcap}$ and the set of maximal CUTs $\boldsymbol{T}^{\frown}_{max}$. 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.,*

$$t^{\frown}_x \mathrel{\hat{\lesssim}} t^{\frown}_y \Leftrightarrow \Uparrow t^{\frown}_x \mathrel{\hat{\lesssim}} \Uparrow t^{\frown}_y \tag{18}$$

Proof. Let $t^{\frown}_x, t^{\frown}_y \in \boldsymbol{T}^{\frown}$. From proposition 4.11 we know that $t^{\frown}_x \mathrel{\hat{\simeq}} \Uparrow t^{\frown}_x$ and $t^{\frown}_y \mathrel{\hat{\simeq}} \Uparrow t^{\frown}_y$.
$\Rightarrow$: If $t^{\frown}_x \mathrel{\hat{\lesssim}} t^{\frown}_y$, then $t^{\frown}_x \mathrel{\hat{\simeq}} \Uparrow t^{\frown}_x \mathrel{\hat{\lesssim}} \Uparrow t^{\frown}_y \mathrel{\hat{\simeq}} t^{\frown}_y$. So $\Uparrow t^{\frown}_x \mathrel{\hat{\lesssim}} \Uparrow t^{\frown}_y$.
$\Leftarrow$: If $\Uparrow t^{\frown}_x \mathrel{\hat{\lesssim}} \Uparrow t^{\frown}_y$, then $\Uparrow t^{\frown}_x \mathrel{\hat{\simeq}} t^{\frown}_x \mathrel{\hat{\lesssim}} t^{\frown}_y \mathrel{\hat{\simeq}} \Uparrow t^{\frown}_y$. So $t^{\frown}_x \mathrel{\hat{\lesssim}} t^{\frown}_y$. □

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

$$\uparrow t^{\frown} \stackrel{\text{def}}{=} \{t \in \boldsymbol{T} \mid \exists t' \in t^{\frown}, t' \lesssim t\} \tag{19}$$

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

- *if $t^{\frown} \in \mathbf{T}^{\frown} \backslash \varnothing - \bot^{\sqcap}$, then $\Uparrow t^{\frown} = \uparrow t^{\frown}$;*
- *if $t^{\frown} \in \bot^{\sqcap}$, then $\Uparrow t^{\frown} = \mathbf{T}$;*

- $\Uparrow \varnothing = \Uparrow \top^\cap$;

Proof. Let $t^\cap \in \boldsymbol{T}^\cap$:
1) if $t^\cap \in \boldsymbol{T}^\cap \backslash \varnothing - \bot^\sqcap$:
$\subseteq$: Let $t \in \Uparrow t^\cap$. By definition, $\exists t_x^\cap \in [t^\cap], t \in t_x^\cap$. So $t^\cap \mathrel{\hat{\cong}} t_x^\cap$ and $t^\cap \mathrel{\hat{\lesssim}} t_x^\cap$.
From proposition 3.6, at least one of the followings holds:
i) $t^\cap \mathrel{\hat{\lesssim}} \bot^\cap$ (impossible from our hypothesis);
ii) $t^\cap = \varnothing$ (impossible from our hypothesis);
iii) $\forall t_x \in t_x^\cap, \exists t' \in t^\cap : t' \lesssim t_x$. So for $t \in t_x^\cap$, $\exists t' \in t^\cap : t' \lesssim t$. And thus $t \in \uparrow t^\cap$.
$\supseteq$: Let $t \in \uparrow t^\cap$. Then $\exists t' \in t^\cap, t' \lesssim t$. From lemma 4.10, $t^\cap \cup \{t\} \mathrel{\hat{\cong}} t^\cap$, and thus $t^\cap \cup \{t\} \in [t^\cap]$, and thus $t \in \Uparrow t^\cap$.
2) if $t^\cap \in \bot^\sqcap$, then $\boldsymbol{T} \in [t^\cap]$ and $\Uparrow t^\cap = \boldsymbol{T}$;
3) if $\varnothing \mathrel{\hat{\cong}} \top^\cap$, then from proposition 4.12, $\Uparrow \varnothing = \Uparrow \top^\cap$. □

Figure 5 illustrates the most interesting case in proposition 4.14 which is the case where $t^\cap \notin \bot^\sqcap$ and $t^\cap \neq \varnothing$: black dots represent PUTs of $t^\cap \in \boldsymbol{T}^\cap$, 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 dots, then the gray zone represents the set of PUTs that belong to $\Uparrow t^\cap$.

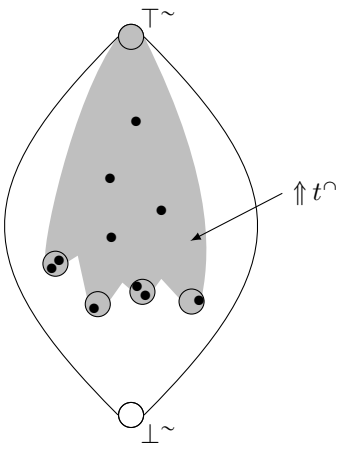


Figure 5: Illustration of a maximal CUT $\Uparrow t^\cap$ with $t^\cap \in \boldsymbol{T}^\cap \backslash \varnothing - \bot^\sqcap$.

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 $t^\cap$ is maximal if and only if one of the following is true:*

- $t^\cap = \mathbf{T}$,
- $t^\cap \in \mathbf{T}^\cap \backslash \varnothing - \bot^\sqcap$, *and $t^\cap$ is a upper set of* $\mathbf{T}$.

Proof. Let us prove both implications:
$\Rightarrow$: If $t^\cap$ is maximal, then there exists $t^{\cap\prime} \in \boldsymbol{T}^\cap, t^\cap = \Uparrow t^{\cap\prime}$.
i) if $t^{\cap\prime} \in \bot^\sqcap$, then $t^\cap = \boldsymbol{T}$;
ii) if $t^{\cap\prime} = \varnothing$, then $t^\cap = \Uparrow \bot^\cap$, and $\Uparrow \bot^\cap$ is either $\boldsymbol{T}$ or a non-empty upper set of $\boldsymbol{T}$;
iii) if $t^{\cap\prime} \in \boldsymbol{T}^\cap \backslash \varnothing - \bot^\sqcap$, then $t^\cap$ is an non-empty upper set of $\boldsymbol{T}$.

$\Leftarrow$:
i) if $t^\frown = \boldsymbol{T}$, then for all $t^{\frown\prime} \in \bot^\sqcap$, $t^\frown = \Uparrow t^{\frown\prime}$ so $t^\frown$ is maximal;
ii) if $t^\frown \notin \bot^\sqcap$, and $t^\frown$ is a non-empty upper set of $\boldsymbol{T}$, then using propositions 4.12 and 4.11, $\Uparrow t^\frown = t^\frown$, so $t^\frown$ is maximal. □

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

The last result of this section is the fact the operator $\Uparrow$ defines a closure operator on $\boldsymbol{T}$.

Proposition 4.16. *$\Uparrow$ is a closure operator on $\mathbf{T}$, i.e., it satisfies the following conditions for all CUTs $t^\frown_x, t^\frown_y \in \mathbf{T}^\frown$:*

- $t^\frown_x \subseteq \Uparrow t^\frown_x$, *($\Uparrow$ is extensive);*
- $t^\frown_x \subseteq t^\frown_y \implies \Uparrow t^\frown_x \subseteq \Uparrow t^\frown_y$, *($\Uparrow$ is increasing with respect to $\subseteq$);*
- $\Uparrow \Uparrow t^\frown_x = \Uparrow t^\frown_x$, *($\Uparrow$ is idempotent);*

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

2) *Increasing w.r.t. $\subseteq$:* Let $t^\frown_x, t^\frown_y \in \boldsymbol{T}^\frown$ such that $t^\frown_x \subseteq t^\frown_y$.
First, we know from proposition 3.7 that $t^\frown_y \mathrel{\hat{\lesssim}} t^\frown_x$.
i) If $t^\frown_y \in \bot^\sqcap$, then $\Uparrow t^\frown_y = \boldsymbol{T}$ (prop. 4.14) and $\Uparrow t^\frown_x \subseteq \Uparrow t^\frown_y$.
ii) If $t^\frown_x \in \bot^\sqcap$, then so is $t^\frown_y$, and $\Uparrow t^\frown_x = \Uparrow t^\frown_y = \boldsymbol{T}$ (prop. 4.14).
iii) If $t^\frown_y = \varnothing$, then $t^\frown_x = t^\frown_y = \varnothing$, and $\Uparrow t^\frown_x = \Uparrow t^\frown_y$.
iv) If $t^\frown_y \notin \bot^\sqcap$, $t^\frown_y \neq \varnothing$, $t^\frown_x \notin \bot^\sqcap$, and $t^\frown_x \neq \varnothing$: Let $t \in \Uparrow t^\frown_x = \uparrow t^\frown_x$. We know that $\exists t' \in t^\frown_x, t' \lesssim t$. As $t^\frown_x \subseteq t^\frown_y$, then $\exists t' \in t^\frown_y, t' \lesssim t$, and $t' \in \uparrow t^\frown_y = \Uparrow t^\frown_y$.
v) If $t^\frown_x = \varnothing$, then $\Uparrow t^\frown_x = \Uparrow \top^\frown$. $\top$ is a maximal element of $\boldsymbol{T}$, so using lemma 4.10, $t^\frown_y \mathrel{\hat{\simeq}} t^\frown_y \cup \top^\frown$, and from definition 4.2, $\Uparrow t^\frown_y = \Uparrow t^\frown_y \cup \top^\frown$. Using items i) to iv), we know that $\Uparrow t^\frown_x = \Uparrow \top^\frown \subseteq \Uparrow t^\frown_y \cup \top^\frown = \Uparrow t^\frown_y$.
So $\Uparrow t^\frown_x \subseteq \Uparrow t^\frown_y$, and $\Uparrow$ is increasing.

3) *Idempotent:* Let $t^\frown \in \boldsymbol{T}^\frown$.
i) if $t^\frown \in \bot^\sqcap$, then $\Uparrow t^\frown = \boldsymbol{T}$, and as we know that $\boldsymbol{T} \in \bot^\sqcap$, $\Uparrow \Uparrow t^\frown = \Uparrow t^\frown = \boldsymbol{T}$.
ii) if $t^\frown \notin \bot^\sqcap$ and $t^\frown \neq \varnothing$, then:
$\supseteq$: We know from extensivity of $\Uparrow$ that $\Uparrow t^\frown \subseteq \Uparrow \Uparrow t^\frown$.
$\subseteq$: Let $t \in \Uparrow \Uparrow t^\frown$. $\Uparrow t^\frown \mathrel{\hat{\simeq}} t^\frown$ so (prop: 4.9 $\Uparrow t^\frown \notin \bot^\sqcap$, and we also know that $t^\frown \subseteq \Uparrow t^\frown$ so $\Uparrow t^\frown \neq \varnothing$. Using upper-set properties: $\Uparrow \Uparrow t^\frown = \uparrow \Uparrow t^\frown = \uparrow \uparrow t^\frown = \uparrow t^\frown = \Uparrow t^\frown$.
iii) if $t^\frown = \varnothing$, then $\Uparrow t^\frown = \Uparrow \bot^\frown$, and using ii), $\Uparrow t^\frown = \Uparrow \bot^\frown = \Uparrow \Uparrow \bot^\frown = \Uparrow \Uparrow t^\frown$. □

Thus a maximal CUTs $t^\frown_\Uparrow$ is also maximal in the sense that any CUT of its equivalence class is contained in $t^\frown_\Uparrow$.

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 $t^\cap = \bot^\cap$.
- else, $\forall t \in t^\cap, \nexists t' \in t^\cap$ such that $t' < t$ and $\forall t, t' \in t^\cap$, $t \not\simeq t'$.

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 $t^\cap$.

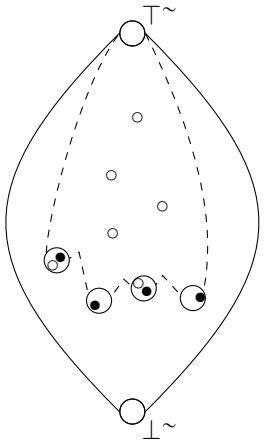


Figure 6: Let the union of black and white dots represent a CUT $t^\cap$. The set of black dots represents a concise CUT that is a subset of $t^\cap$.

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 $\boldsymbol{S_C}$.

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 $\boldsymbol{S_C}$)**.** The CSymbols set is pre-ordered by a relation $\overset{c}{\lesssim}$, which is induced by a set of *asserted comparisons* $\boldsymbol{C_{S_C}} \subseteq \boldsymbol{T}^{\frown}$. $(\boldsymbol{S_C}, \boldsymbol{C_{S_C}})$ is a directed graph on $\boldsymbol{S_C}$. The pre-order $\overset{c}{\lesssim}$ is equal to $\boldsymbol{C^*_{S_C}}$ the reflexo-transitive closure of $\boldsymbol{C_{S_C}}$, i.e., $\forall s, s' \in \boldsymbol{S_C}, s' \overset{c}{\lesssim} s$ iif $(s, s') \in \boldsymbol{C^*_{S_C}}$, 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 trough a relation having this symbol.

Definition 5.3 (Signature of CSymbols)**.** The set of signatures of CSymbols $\{\boldsymbol{\sigma}_s\}_{s \in \boldsymbol{S_C}}$ is a set of couples in $\boldsymbol{T}^{\frown 2}$. For every dependency relation symbol s, $\boldsymbol{\sigma}_s \overset{\text{def}}{=} (domain(s), range(s))$. The set of signatures $\{\boldsymbol{\sigma}_s\}_{s \in \boldsymbol{S_C}}$ must be such that for all $s_1, s_2 \in \boldsymbol{S_C}$ such that $s_1 \overset{c}{\lesssim} s_2$, $domain(s_1) \overset{\frown}{\lesssim} domain(s_2)$ and $range(s_1) \overset{\frown}{\lesssim} range(s_2)$.

We finally introduce the full hierarchy of CSymbols.

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

- $\boldsymbol{S_C}$ a set of Circumstantial Dependency Symbols (CSymbols);
- $\boldsymbol{C_{S_C}}$ a set of asserted comparisons;
- $\mathcal{T}^{\cap} = (T_{declared}, C_{asserted}, \bot^{\sqcap}_{asserted}, \boldsymbol{S_{\mathcal{T}}}, root, Hiders, \{\boldsymbol{\varsigma}_t\}_{t \in \boldsymbol{T}})$ a Conjunctive Unit Types (CUTs) hierarchy;
- $\{\boldsymbol{\sigma}_s\}_{s \in \boldsymbol{S_C}}$ 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 $\mathcal{S} \stackrel{\text{def}}{=} (\mathcal{T}^{\cap}, \mathcal{C}, \boldsymbol{M})$ where:

- $\mathcal{T}^{\cap}$ is a *Conjunctive Unit Types (CUTs) hierarchy*;
- $\mathcal{C}$ is a *Circumstantial Dependency Symbols (CSymbols) hierarchy*;
- $\boldsymbol{M}$ is the set of so-called *unit markers*. $\boldsymbol{M}^{\cap} = 2^{\boldsymbol{M}}$ is the set of unit node markers, with $\varnothing$ 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 $\boldsymbol{M}$, 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 $\boldsymbol{M}$ which is denoted $\boldsymbol{M}^{\cap}$: $\boldsymbol{M}^{\cap} \stackrel{\text{def}}{=} 2^{\boldsymbol{M}}$, 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 $\varnothing$ (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.

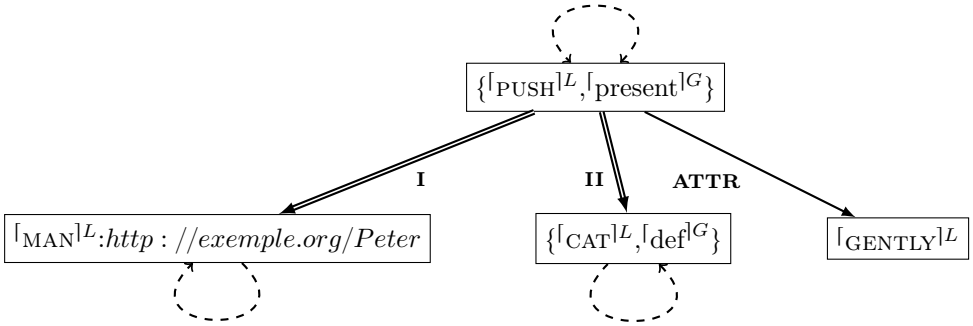


Figure 7: Example of a UG: Deep syntactic representation of sentence *Peter gently pushes the cat.*

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 $\varnothing$. Every dependency arc is labelled by a symbol in $\boldsymbol{S}_{\mathcal{T}} \cup \boldsymbol{S}_{\mathcal{C}}$ 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 $G \stackrel{\text{def}}{=} (U, \boldsymbol{l}, P, C, Eq)$ where:

- U is the set of *unit nodes*. They are illustrated by rectangles as on figure 7.
- $\boldsymbol{l}$ is a labelling mapping over U that assigns to each unit node $u \in U$ a couple $\boldsymbol{l}(u) = (t^{\frown}, m^{\frown}) \in \boldsymbol{T}^{\frown} \times \boldsymbol{M}^{\frown}$. u is thus a unit node that has type $t^{\frown}$ and marker $m^{\frown}$. We denote $t^{\frown} = type(u)$ and $m^{\frown} = marker(u)$. 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 $\{http : //exemple.org/Peter\}$.
- P is the set of *participation triples* $(u, r, v) \in U \times \boldsymbol{S}_{\mathcal{T}} \times U$. 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 $u = governor(p)$, $r = symbol(p)$ and $v = participant(p)$. We also denote $arc(p) = (u, v)$. They are illustrated by double arrows on figure 7.
- C is the set of *circumstantial dependency triples* $(u, r, v) \in U \times \boldsymbol{S}_{\mathcal{C}} \times U$. 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 $u = governor(c)$, $r = symbol(c)$ and $v = circumstant(c)$. We also denote $arc(c) = (u, v)$. 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 $graph(G)$ is a finite, oriented, labelled multigraph where:

  for all $u \in U$, u is a node of $graph(G)$ labelled by $\boldsymbol{l}(u)$;

  for all $p \in P$, $arc(p)$ is an arc of $graph(G)$ labelled by $symbol(p)$;

  for all $c \in C$, $arc(c)$ is an arc of $graph(G)$ labelled by $symbol(c)$;

  for all $(u_1, u_2) \in Eq$, (u_1, u_2) is an arc of $graph(G)$ 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 $marker(u_1) \cup marker(u_2)$ 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.

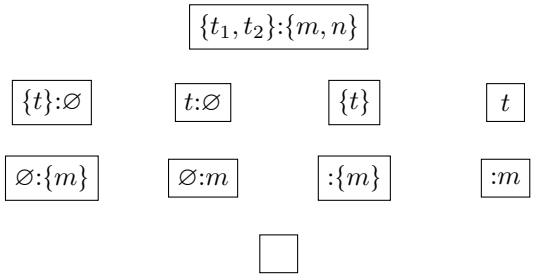


Figure 8: Graphical Representation of unit nodes and their labels:
All the unit nodes of each line share the same label.

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

- participation dependencies are drawn with a double arrow. If a participation dependency is know 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.

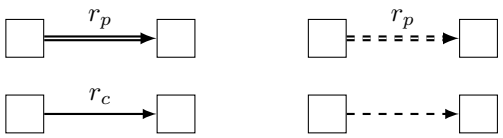


Figure 9: Graphical Representation of arcs and their labels. Top left: a participation; right: a hidden participation. Bottom left: a circumstantial dependency; right: an asserted unit nodes equivalence.

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 $(U, \boldsymbol{l}, P, C, Eq) \in \mathcal{G}(\mathcal{S})$, 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 $governor(p) = u$, $symbol(p) \in \boldsymbol{\omega}^{\cap}(type(u))$.

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 $(U, \boldsymbol{l}, P, C, Eq) \in \mathcal{G}(\mathcal{S})$, a participation triple $(u, r, v) \in P$ is said to *explicitly comply with the support* $\mathcal{S}$ if and only if: $r \in \boldsymbol{\omega}^{\cap}(type(u))$ and $\boldsymbol{\varsigma}^{\cap}_{type(u)}(r) \subseteq type(v)$.

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

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	$u \in U$	$\Uparrow type(u) \subseteq type(u)$
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	$(u_1, u_2) \in Eq$	$type(u_1) \subseteq type(u_2)$
eq-mrk-1	$(u_1, u_2) \in Eq$	$marker(u_1) \subseteq marker(u_2)$
eq-mrk-2	$u_1, u_2 \in U$ such that $marker(u_1) \cap marker(u_2) \neq \varnothing$	$(u_1, u_2) \in Eq$
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	$(u, r, v) \in P$	$root(u) \in type(u)$
p-sig	$(u, r, v) \in P$	$\varsigma_{type(u)}(r) \subseteq type(v)$
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	$(u, r, v) \in C$	$domain(r) \subseteq type(u)$
c-rng	$(u, r, v) \in C$	$range(r) \subseteq type(v)$
c-sop	$(u, r_1, v) \in C$ and $r_1 \overset{c}{\lesssim} r_2$	$(u, r_2, v) \in C$

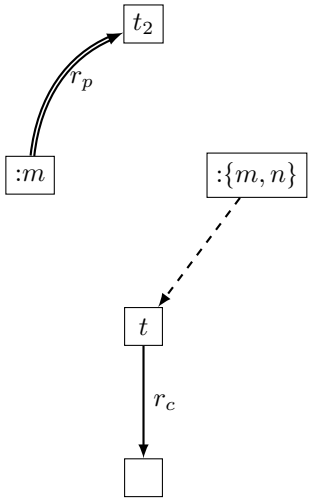
Table 1: Semantics of the UGs.

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.

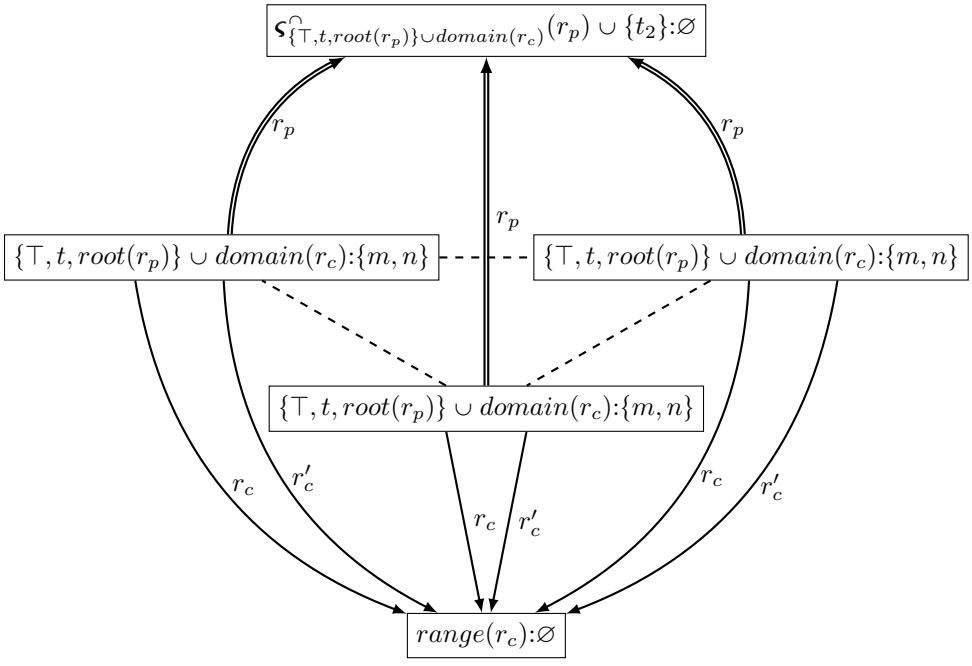
Definition 5.11 (Saturating a UG, Saturated UG)**.** Let $G \in \mathcal{G}(\mathcal{S})$. The operation of *saturating* G consists in applying sequentially all of the entailment rules of table 1 over and over, until none of them has any effect on G.
A UG for which none of the entailment rules of table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

Consider a simple support $\mathcal{S}$ made of a CUTs hierarchy $\mathcal{T}^{\cap}$ with $t, t_2 \in T_{declared}$, $r_p \in \boldsymbol{S}_{\mathcal{T}}$, a CSymbols hierarchy with $r_c, r'_c \in \boldsymbol{S}_{\mathcal{C}}$ and $(r'_c, r_c) \in \boldsymbol{C}_{\boldsymbol{S}_{\mathcal{C}}}$, and $m, n \in \boldsymbol{M}$. 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 $type(u)$, 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 $t \in T_{declared}$ and $s \in \boldsymbol{S}_{\mathcal{T}}$ such that $root(s) = t$ and $\boldsymbol{\varsigma}_t(s) = t$, a void CSymbols hierarchy, and a void set of unit markers. Consider the UG $G = (\{u\}, \boldsymbol{l}, \varnothing, \varnothing, \{(u, u)\}) \in \mathcal{G}(\mathcal{S})$, such that $marker(u) = \varnothing$ and $type(u) = \{\top, t\}$. 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.

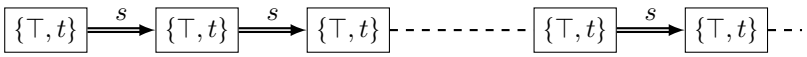


Figure 11: Illustration of an infinite closure of a simple UG

6 Mappings of UGs

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

Definition 6.1 (UG Mapping)**.** Let $G = (U^g, \boldsymbol{l}^g, P^g, C^g, Eq^g) \in \mathcal{G}(\mathcal{S}_G)$ and $H = (U^h, \boldsymbol{l}^h, P^h, C^h, Eq^h) \in \mathcal{G}(\mathcal{S}_H)$ be two UGs. A *mapping* f from H to G, written $f : H \to G$, is a mapping $f : U^h \to U^g$ 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 $\overset{\frown}{\lesssim}$ for types;
- equality for participation triples;
- pre-order $\overset{c}{\lesssim}$ for circumstantial dependency relations;
- equality for asserted equivalence relations.

Then the definition of a weak homomorphism is straightforward.

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

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

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

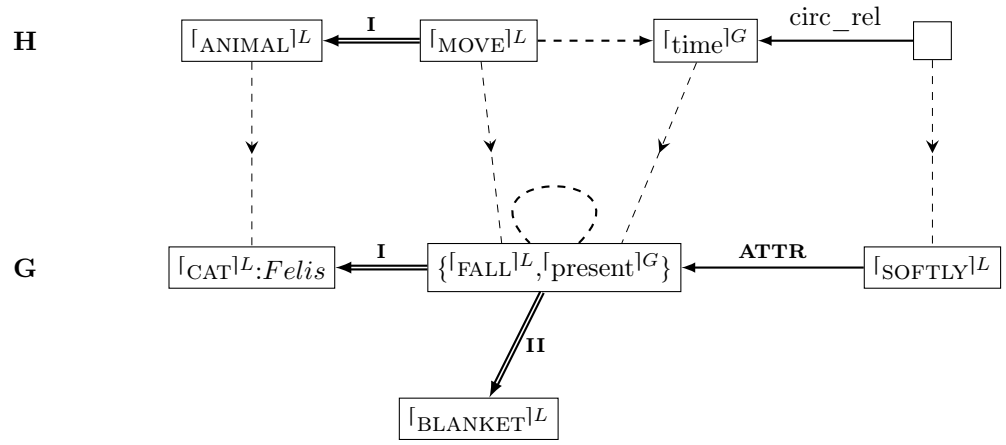


Figure 12: Illustration of a weak homomorphism from H to G.

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 $G = (U^g, \boldsymbol{l}^g, P^g, C^g, Eq^g) \in \mathcal{G}(\mathcal{S})$ and $H = (U^h, \boldsymbol{l}^h, P^h, C^h, Eq^h) \in \mathcal{G}(\mathcal{S})$ 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.

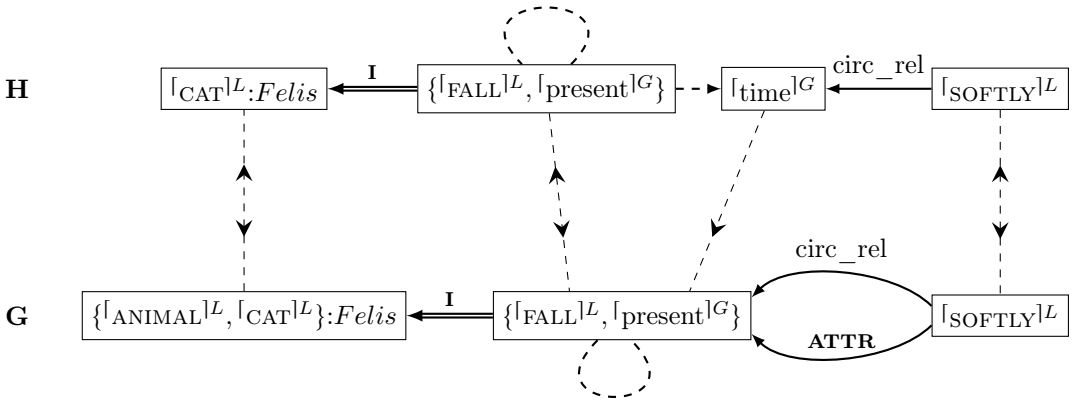


Figure 13: Illustration of a weak hom-equivalence between H and G.

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

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

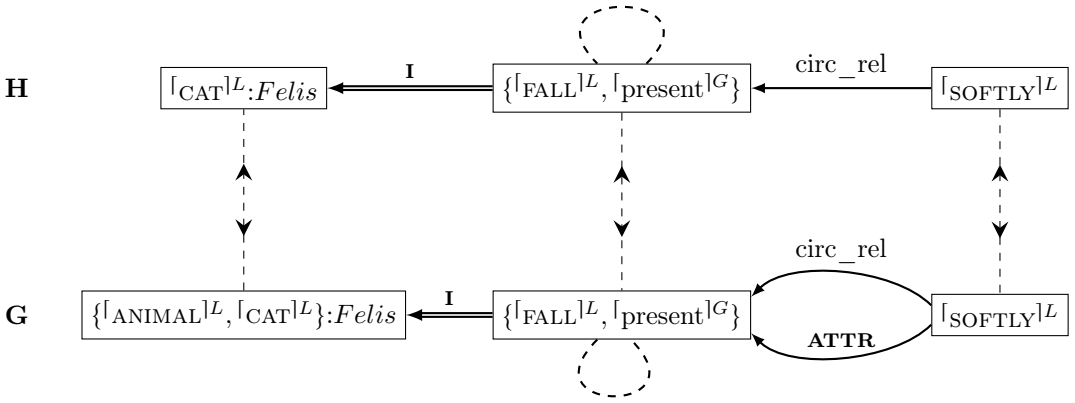


Figure 14: Illustration of a weak isomorphism between H and G.

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 $G = (U^g, \mathbb{l}^g, P^g, C^g, Eq^g) \in \mathcal{G}(\mathcal{S})$ and $H = (U^h, \mathbb{l}^h, P^h, C^h, Eq^h) \in \mathcal{G}(\mathcal{S})$ 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:

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

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

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

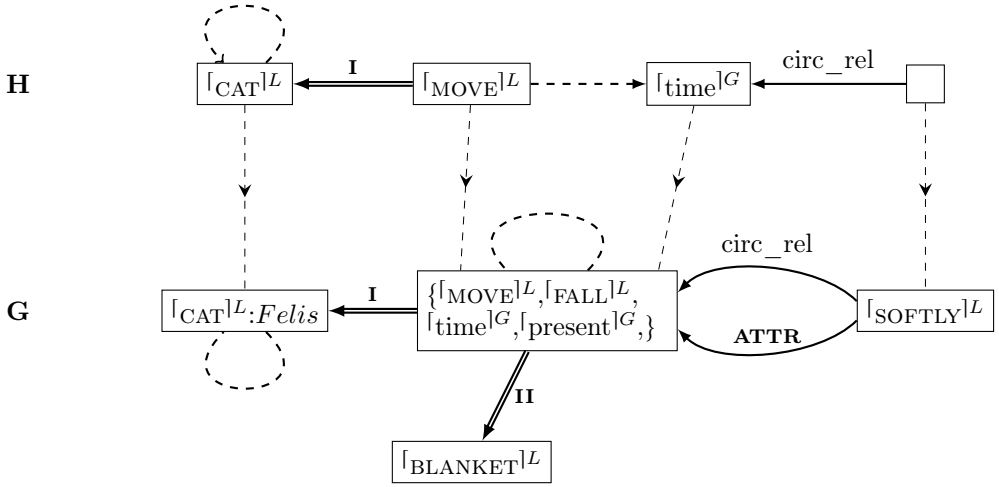


Figure 15: Illustration of a strong homomorphism from H to G.

6.5 Strong Isomorphism: Equality

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

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

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

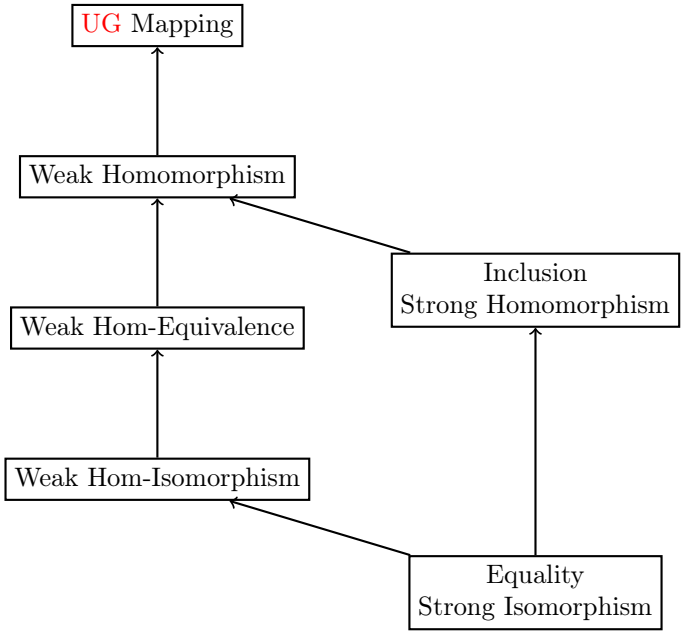


Figure 16: Illustration of the links between the different UGs mappings.

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 $G = (U, \boldsymbol{l}, P, C, Eq) \in \mathcal{G}(\mathcal{S})$. 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-typ-top** Let $u \in U$. Add $\top$ to $type(u)$;
- **u-typ** Let $u \in U$, and $t, t' \in \boldsymbol{T}$ 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-typ** 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 $marker(u_1) \cap marker(u_2) \neq \varnothing$. 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 $t \in \boldsymbol{\varsigma}^{\cap}_{type(u)}(r)$. 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 $r_1 \overset{c}{\lesssim} r_2$. Add (u, r_2, v) in C.
- **p-expl** Let $u \in U$ and $s \in \boldsymbol{\omega}(u)$. Add v in U with $type(v) = \varnothing$ and $marker(v) = \varnothing$, 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 twoUGs is a specialization operation defined by:

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

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

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

A UG may be considered as a λ-UG with $\{u_1, \ldots, u_n\} = \varnothing$.

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

1. add C' to G;
2. for all $u^c \in \{u_1^c, \ldots, u_n^c\}$, 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:

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

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 $\boldsymbol{T}$: 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 $D_t^- = \{u_t, v_1, \ldots, v_{valency_\alpha(t)}\}(U^-, \boldsymbol{l}^-, P^-, \varnothing, \varnothing)$ such that:

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

- there is a bijection $\mathbf{v}$ from $\boldsymbol{\alpha}(t)$ to $\{v_1, \ldots, v_{valency_\alpha(t)}\}$;
- $P^- = \{(u_t, s, \mathbf{v}(s))\}_{s \in \boldsymbol{\alpha}(t)}$ (i.e., u_t is explicit and any other unit node is implicit);
- for all $s \in \boldsymbol{\alpha}(t)$, $type(\mathbf{v}(s)) = \boldsymbol{\varsigma}_t(s)$;
- 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 $\bot$ respectively.

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

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:

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

(D_t^-, D_t^+, κ) and $(D_t^+, D_t^-, \kappa^{-1})$ are respectively called the $E \rightarrow P$ (expansion → print) and the $P \rightarrow E$ (print → 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 $t_1, \ldots, t_n$ 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 $t_{i+1} \in type(u)$.

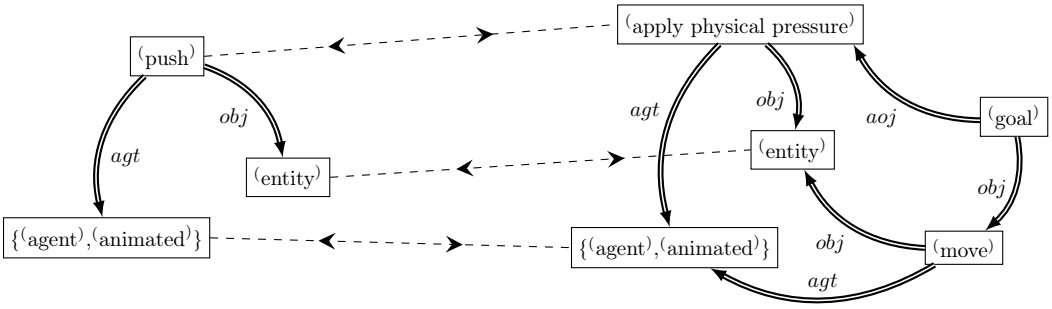


Figure 18: Definition of $\lceil \text{PUSH} \rceil^L$. On the left, the print of ⟨push⟩.

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 $\mathcal{S} \stackrel{\text{def}}{=} (\mathcal{T}^\cap, \mathcal{C}, \boldsymbol{M}, \mathcal{D})$ where:

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

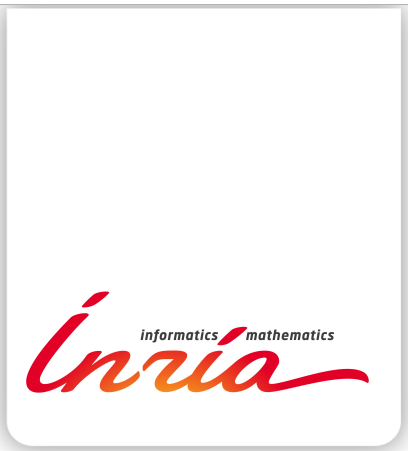
Rudolph, S. (2011). Foundations of description logics. *Reasoning Web. Semantic Technologies for the Web of Data*, pages 76–136.

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