Reasoning with Dependency Structures and Lexicographic Definitions using Unit Graphs

Maxime Lefrançois, Fabien Gandon

Reasoning with Dependency Structures and Lexicographic Definitions using Unit Graphs

Maxime Lefrançois and **Fabien Gandon**
Wimmics, Inria, I3S, CNRS, UNSA
2004 rte des Lucioles, BP. 93, 06902 Sophia Antipolis, France
{maxime.lefrancois,fabien.gandon}@inria.fr

Abstract

We are interested in a graph-based Knowledge Representation (KR) formalism that would allow for the representation, manipulation, query, and reasoning over dependency structures, and linguistic knowledge of the lexicon in the Meaning-Text Theory framework. Neither the semantic web formalisms nor the conceptual graphs appear to be suitable for this task, and this led to the introduction of the new Unit Graphs (UG) framework. In this paper we will overview the foundational concepts of this framework: the UGs are defined over a UG-support that contains: i) a hierarchy of unit types which is strongly driven by the actantial structure of unit types, ii) a hierarchy of circumstantial symbols, and iii) a set of unit identifiers. Based on these foundational concepts and on the definition of UGs, this paper justifies the use of a deep semantic representation level to represent meanings of lexical units. Rules over UGs are then introduced, and lexicographic definitions of lexical units are added to the hierarchy of unit types. Finally this paper provides UGs with semantics (in the logical sense), and pose the entailment problem, so as to enable the reasoning in the UGs framework.

1 Introduction

We are interested in a graph-based Knowledge Representation (KR) formalism that would allow for the representation, manipulation, query, and reasoning over dependency structures and linguistic knowledge of the Explanatory Combinatorial Dictionary (ECD), which is the lexicon at the core of the Meaning-Text Theory (MTT) (Mel'čuk, 2006).

Most past or current projects that aimed at implementing the ECD did so in a lexicographic perspective. One important example is the RELIEF project (Lux-Pogodalla and Polguère, 2011), which aims at representing a lexical system graph named RLF (Polguère, 2009), where lexical units are interlinked by paradigmatic and syntagmatic links of lexical functions (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 in the RELIEF project using the markup type that has been developed in the Definiens project (Barque and Polguère, 2008; Barque et al., 2010).

One exception is the proprietary linguistic processor ETAP-3 that implements a variety of ECD for Natural Language Processing (Apresian et al., 2003; Boguslavsky et al., 2004). Linguistic knowledge are asserted, and linguistic and grammatical rules are directly formalized in first order logic.

Adding to these formalization works, our goal is to propose a formalization from a knowledge engineering perspective, compatible with standard KR formalisms. The term *formalization* here means not only *make non-ambiguous*, but also *make operational*, i.e., *such that it supports logical operations* (e.g., knowledge manipulation, query, reasoning). We thus adopt a knowledge engineering approach applied to the domain of the MTT. The semantic web formalisms and the Conceptual Graphs formalism both seem to fit this task, but they actually present strong incompatibilities with the description of linguistic predicates (Lefrançois, 2013). These issues led to the introduction of the new graph-based Unit Graphs (UGs) KR formalism. In the UGs framework, the linguistic predicates are represented by unit types, and are described in a structure called the unit types hierarchy. Unit types specify through their

actantial structure, i.e., actant slots, the ways in which their instances may be linked to other units in a UG.

The main research question of this paper is the following: *What semantics can be attributed to UGs, and how can we define the entailment problem for UGs ?*

The rest of this paper is organized as follows. We will first overview the issues of existing graph-based KR formalisms that led to the introduction of the new UGs framework (§2). The foundational concepts of this framework are then introduced (§3). From these foundational concepts we define the lexicographic definitions of Lexical Unit Types (LexUTs) (§4). Finally we provide UGs with semantics (in the logical sense), and pose the entailment problem (§5).

2 The Unit Graphs Formalism

At first sight, two existing KR formalisms seem interesting for representing dependency structures: semantic web formalisms (RDF[1], RDFS[2], OWL[3], SPARQL[4]), and Conceptual Graphs (CGs) (Sowa, 1984; Chein and Mugnier, 2008). Both formalisms are based on directed labelled graph structures, and some research has been done towards using them to represent dependency structures and knowledge of the lexicon (OWL in (Lefrançois and Gandon, 2011; Boguslavsky, 2011), CGs at the conceptual level in (Bohnet and Wanner, 2010)). Yet authors in (Lefrançois, 2013) showed that neither of these KR formalisms can represent linguistic predicates. Let us list the main drawbacks of these existing formalisms:

- The RDF is insufficient because its semantics is limited to that of oriented labelled graphs.
- In RDFS, OWL and the CGs, there is a strong distinction between concept types and relations. Yet, a linguistic predicate may be considered both as a concept type as it is instantiated in dependency structures, and as a relation as its instances may link other instances.
- RDFS and OWL only model binary relations, which is not the case of most linguistic predicates. One would need to use reification of n-ary relations, but then no semantics is attributed to such relations.
- CGs support n-ary relations, but the inheritance mechanism of relation types is such that two relations with different arities must be incomparable. Yet the Semantic Actant Slots (SemASlots) of a lexical unit are determined by linguistic criteria and may thus differ from those of the lexical unit from which its sense derives (Mel'čuk, 2004, p. 38). One thus cannot use the natural inheritance mechanism of CGs to model the meaning specialization of predicates.

As the CGs formalism is the closest to the semantic networks, the following choice has been made to overcome these issues: *Modify the CGs formalism basis, and define transformations to the RDF syntax for sharing knowledge and publishing over the web of data.* As we are to represents linguistic units of different nature (e.g., semantic units, lexical units, grammatical units, words), term *unit* has been chosen to be used in a generic manner, and the result of this adaptation is thus *the Unit Graphs (UGs) framework*.

3 Unit Graphs

For a specific Lexical Unit L, (Mel'čuk, 2004, p. 5) distinguishes considering L in language (i.e., in the lexicon), or in speech (i.e., in an utterance). KR formalisms and the Unit Graphs (UGs) formalism also make this distinction using types. In this paper and in the UGs formalism, there is thus a clear distinction between *units* (e.g., semantic unit, lexical unit), which will be represented in the UGs, and their *types* (e.g., semantic unit type, lexical unit type), which are roughly classes of units for which specific features are shared. It is those types that specify through actant slots how their instances (i.e., units) are to be linked to other units in a UG.

3.1 Support

Following the example of CGs, UGs are defined over a so-called *support*.

Definition 1. A UG *support* is denoted $\mathcal{S} \stackrel{\text{def}}{=} (\mathcal{T}, \mathcal{C}, \boldsymbol{M})$ and is composed of a hierarchy of unit types $\mathcal{T}$, a hierarchy of circumstantial symbols $\mathcal{C}$, and a set of unit identifiers $\boldsymbol{M}$.

Let us briefly introduce the components of $\mathcal{S}$.

[1] RDF - Resource Description Framework, c.f., `http://w3.org/RDF/`

[2] RDFS - RDF Schema, c.f., `http://www.w3.org/TR/rdf-schema/`

[3] OWL - Web Ontology Language, c.f., `http://www.w3.org/TR/owl2-overview/`

[4] SPARQL, c.f., `http://www.w3.org/TR/sparql11-overview/`

First, unit types and their actantial structure are described in a structure called *hierarchy* and denoted $\mathcal{T}$. Whether they are semantic, lexical or grammatical, unit types have *Actant Slots (ASlots)* with symbols. Moreover, ASlots may be optional, obligatory, or prohibited (Lefrançois and Gandon, 2013a). Let us briefly introduce two of the components of $\mathcal{T}$, and leave details for section 3.3.

- The core of $\mathcal{T}$ is the set of so-called *Primitive Unit Types (PUTs)*, denoted $\boldsymbol{T}$. Now a unit type may consist of several conjoint PUTs. In particular, it may be a lexical PUT and multiple grammatical PUTs, like $\{def, plur, \text{CAT}\}$. To represent this, we introduce the set $\boldsymbol{T}^{\cap}$ of possible *Conjunctive Unit Types (CUTs)* over $\boldsymbol{T}$ as the powerset[5] of $\boldsymbol{T}$.
- $\mathcal{T}$ contains a set of binary relation symbols of type predicate-argument called *Actant Symbols (ASymbols)*, and denoted $\boldsymbol{S}_{\mathcal{T}}$. $\boldsymbol{S}_{\mathcal{T}}$ contains numbers for the semantic unit types, and other "classical" symbols for the other levels under consideration (e.g, roman numerals **I** to **VI** for the MTT's Deep Syntactic level). Every unit type has an *actantial structure* that consists in a set of optional, obligatory and prohibited ASlots associated with some ASymbols. The actantial structure of a PUT specify how units of this type shall be linked to other units through so-called *actantial relations* in a dependency structure.

Second, there exists dependencies other than actantial: circumstantial relations (Mel'čuk, 2004). Circumstantial relations are considered of type instance-instance contrary to actantial relations. Example of such relations are the deep syntactic representation relations **ATTR**, **COORD**, **APPEND** of the MTT, but we may also use such relations to represent communicative dependencies for instance. Circumstantial relations are labelled by symbols chosen in a set of so-called *Circumstantial Symbols (CSymbols)*, denoted $\boldsymbol{S}_{\mathcal{C}}$, and their classes and usage are described in a hierarchy denoted $\mathcal{C}$. Section 3.4 details the hierarchy of CSymbols.

Finally, one actually needs symbols to identify units. We thus introduce a set of so-called *unit identifiers*, denoted $\boldsymbol{M}$. Every element of $\boldsymbol{M}$ identifies a specific unit, but multiple elements of $\boldsymbol{M}$ may identify the same unit.

[5]The powerset of X is the set of all subsets of X: 2^X

3.2 Definition of UGs

The UGs represent different types of dependency structures (e.g., semantic, syntactic). In a UG, unit nodes that are typed are interlinked by dependency relations that are either actantial or circumstantial.

Definition 2. A UG G defined over a support $\mathcal{S}$ is a tuple denoted $G \stackrel{\text{def}}{=} (U, \boldsymbol{l}, A, C, Eq)$ where U is the set of unit nodes, $\boldsymbol{l}$ is a labelling mapping over U, A and C are respectively actantial and circumstantial triples, and Eq is a set of asserted unit node equivalences.

Let us detail the components of G.

U is the set of *unit nodes*. Every unit node represents a specific unit, but multiple unit nodes may represent the same unit. Unit nodes are typed and marked so as to respectively specify what CUT they have and what unit they represent. The marker of a unit node is a set of unit identifiers, for mathematical reasons. The set of *unit node markers* is denoted $\boldsymbol{M}^{\cap}$ and is the powerset[5] of $\boldsymbol{M}$. If a unit node is marked by $\varnothing$, it is said to be *generic*, and the represented unit is unknown. On the other hand, if a unit node is marked $\{m_1, m_2\}$, then the unit identifiers m_1 and m_2 actually identify the same unit. $\boldsymbol{l}$ is thus a labelling mapping over U that assigns to each unit node $u \in U$ a couple $\boldsymbol{l}(u) = (t^{\cap}, m^{\cap}) \in \boldsymbol{T}^{\cap} \times \boldsymbol{M}^{\cap}$ of a CUT and a unit node marker. We denote $t^{\cap} = type(u)$ and $m^{\cap} = marker(u)$. Unit nodes are illustrated by rectangles with their label written inside.

A is the set of *actantial triples* $(u, s, v) \in U \times \boldsymbol{S}_{\mathcal{T}} \times U$. For all $a = (u, s, v) \in A$, the unit represented by v fills the ASlot s of the unit represented by u. We denote $u = governor(a)$, $s = symbol(a)$ and $v = actant(a)$. We also denote $arc(a) = (u, v)$. Actantial triples are illustrated by double arrows.

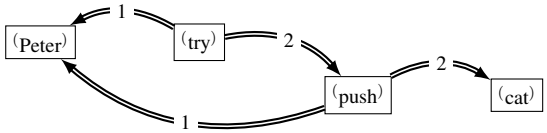
C is the set of *circumstantial triples* $(u, s, v) \in U \times \boldsymbol{S}_{\mathcal{C}} \times U$. For all $c = (u, s, v) \in C$, the unit represented by u governs the unit represented by v with respect to s. We denote $u = governor(c)$, $s = symbol(c)$ and $v = circumstantial(c)$. We also denote $arc(c) = (u, v)$. Circumstantial triples are illustrated by simple arrows.

$Eq \subseteq U^2$ is the set of so-called *asserted unit node equivalences*. For all couple (u_1, u_2) in Eq, u_1 and u_2 represent the same unit. The Eq relation is not an equivalence relation over unit nodes[6]. We thus distinguish explicit and implicit

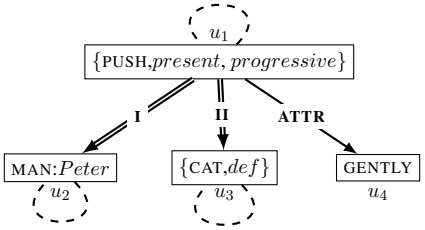
[6]An equivalent relation is a reflexive, symmetric, and

knowledge. Asserted unit node equivalences are illustrated by dashed arrows.

For instance, figure 1a is a semantic representation of sentence *Peter tries to push the cat.* in which units are typed by singletons and ASymbols are numbers, in accordance with the MTT. Figure 1b is a simplified deep syntactic representation of *Peter is gently pushing the cat.* In this figure unit nodes u_2 and u_4 are typed by singletons, and only unit node u_2 is not generic and has a marker: $\{Peter\}$. P is composed of $(u_1, \mathbf{I}, u_2)$ and $(u_1, \mathbf{II}, u_3)$, where **I** and **II** are ASymbols. C is composed of $(u_1, \mathbf{ATTR}, u_4)$ where **ATTR** is a CSymbol. In the relation Eq there is (u_1, u_1), (u_2, u_2), and (u_3, u_3).



(a) Semantic representation of sentence *Peter tries to push the cat.*



(b) Deep syntactic representation of sentence *Peter is gently pushing the cat.*

Figure 1: Examples of Unit Graphs.

UGs so defined are the core dependency structures of the UG mathematical framework. Before we present what they may be used for in more details, let us look more closely into components of the support on which UGs are defined: the unit types hierarchy and the CSymbols hierarchy.

3.3 Unit Types Hierarchy

As already stated in section 3.1, unit types and their actantial structure is described in the unit types hierarchy which is denoted $\mathcal{T}$.

Definition 3. A hierarchy of unit types, denoted $\mathcal{T}$, is a tuple $\mathcal{T} \stackrel{\text{def}}{=} (T_D, \boldsymbol{S}_\mathcal{T}, \boldsymbol{\gamma}, \boldsymbol{\gamma_1}, \boldsymbol{\gamma_0}, C_A, \bot_A^\sqcap, \{\boldsymbol{\varsigma}_t\}_{t\in\boldsymbol{T}})$ that enables to construct a pre-ordered[7] set of Conjunctive Unit Types (CUTs) $\boldsymbol{T}^\cap$ with optional, obligatory, or prohibited ASlots.

This structure has been thoroughly described in (Lefrançois and Gandon, 2013a; Lefrançois, 2013). Let us overview its components.

T_D is a set of *declared Primitive Unit Types (PUTs)*. This set is partitioned into linguistic PUTs of different nature (e.g., deep semantic, semantic, lexical). $\boldsymbol{S}_\mathcal{T}$ is a set of Actant Symbols (ASymbols). $\boldsymbol{\gamma}$ (resp1. $\boldsymbol{\gamma_1}$, resp2. $\boldsymbol{\gamma_0}$) assigns to every $s \in \boldsymbol{S}_\mathcal{T}$ its radix[8] (resp1. obligat[9], resp2. prohibet[10]) unit type $\boldsymbol{\gamma}(s)$ (resp1. $\boldsymbol{\gamma_1}(s)$, resp2. $\boldsymbol{\gamma_0}(s)$) that introduces (resp1. makes obligatory, resp2. makes prohibited) an Actant Slot (ASlot) of symbol s.

The set of *Primitive Unit Types (PUTs)* is denoted $\boldsymbol{T}$ and defined as the disjoint union of T_D, the range of $\boldsymbol{\gamma}$, $\boldsymbol{\gamma_1}$ and $\boldsymbol{\gamma_0}$, plus the *prime universal PUT* $\top$ and the *prime absurd PUT* $\bot$. $\boldsymbol{T}$ is then pre-ordered by a relation $\lesssim$ which is computed from the set of asserted PUTs comparisons $C_A \subseteq \boldsymbol{T}^2$. $t_1 \lesssim t_2$ models the fact that the PUT t_1 is more specific than the PUT t_2. A unit type may consist of several conjoint PUTs. We introduce the set $\boldsymbol{T}^\cap$ of possible *Conjunctive Unit Types (CUTs)* over $\boldsymbol{T}$ as the powerset[5] of $\boldsymbol{T}$. The set $\bot_A^\sqcap$ is the set of declared absurd CUTs that can not be instantiated.

The actantial structure of a unit type $t^\cap$ is a set of ASlots, whose symbols are chosen in the set $\boldsymbol{S}_\mathcal{T}$, and that may be optional, obligatory, or prohibited. Moreover, ASlots are signed. The signature of $t^\cap$ for one of its ASlots s is denoted $\boldsymbol{\varsigma}_{t^\cap}^\cap(s)$, and characterises the type of the unit that fills this slot. The set of signatures of $t^\cap$ is computed from the set of PUTs signatures $\{\boldsymbol{\varsigma}_t\}_{t\in\boldsymbol{T}}$.

Finally the pre-order $\lesssim$ over $\boldsymbol{T}$ is extended to a pre-order $\overset{\cap}{\lesssim}$ over $\boldsymbol{T}^\cap$ as defined by Lefrançois and Gandon (2013a). Lefrançois and Gandon (2013b) proved that in the hierarchy of unit types, if $t_1^\cap \overset{\cap}{\lesssim} t_2^\cap$ then the actantial structure of $t_1^\cap$ is more specific than that of $t_2^\cap$, except for some degenerated cases. Thus as one goes down the hierarchy of unit types, an ASlot with symbol s is introduced by the radix $\{\boldsymbol{\gamma}(s)\}$ and first defines an optional ASlot for any unit type $t^\cap$ more specific than $\{\boldsymbol{\gamma}(s)\}$, as long

transitive relation.

[7] A pre-order is a reflexive and transitive binary relation.

[8] radix is a latin word that means ⟨root⟩.

[9] obligat is the conjugated form of the latin verb obligo, 3p sing. pres., ⟨it makes mandatory⟩.

[10] prohibet is the conjugated form of the latin verb prohibeo, 3p sing. pres., ⟨it prohibits⟩.

as $t^{\cap}$ is not more specific than the obligat $\{\boldsymbol{\gamma_1}(s)\}$ (resp. the prohibet $\{\boldsymbol{\gamma_0}(s)\}$) of s. If that happens, the ASlot becomes obligatory (resp. prohibited). Moreover, the signature of an ASlot may only become more specific.

3.4 Circumstantial Symbols Hierarchy

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 dependency structures: circumstantial dependencies (Mel'čuk, 2004). Circumstantial relations are considered of type instance-instance contrary to actantial relations. Example of such relations are the deep syntactic representation relations **ATTR**, **COORD**, **APPEND** of the MTT.

We thus introduce a finite set of so-called *Circumstantial Symbols (CSymbols)* $\boldsymbol{S_C}$ which is a set of binary relation symbols. In order to classify $\boldsymbol{S_C}$ in sets and subsets, we introduce a partial order $\overset{\mathcal{C}}{\lesssim}$ over $\boldsymbol{S_C}$. $\overset{\mathcal{C}}{\lesssim}$ is the reflexo-transitive closure of a set of *asserted comparisons* $\boldsymbol{C_{S_C}} \subseteq \boldsymbol{T}^2$. Finally, to each CSymbol is assigned a signature that specifies the type of units that are linked through a relation having this symbol. The set of signatures of CSymbol $\{\boldsymbol{\sigma}_s\}_{s\in\boldsymbol{S_C}}$ is a set of couples of CUTs: $\{(domain(s), range(s))\}_{s\in\boldsymbol{S_C}}$. As one goes down the hierarchy of PUTs, we impose that the signature of a CSymbol may only become more specific. We may hence introduce the hierarchy of CSymbols:

Definition 4. The hierarchy of Circumstantial Symbols, denoted $\mathcal{C} \overset{\text{def}}{=} (\boldsymbol{S_C}, \boldsymbol{C_{S_C}}, \mathcal{T}, \{\boldsymbol{\sigma}_s\}_{s\in\boldsymbol{S_C}})$, is composed of a finite set of CSymbols $\boldsymbol{S_C}$, a set of declared comparisons of CSymbols $\boldsymbol{C_{S_C}}$, a hierarchy of unit types $\mathcal{T}$, and a set of signatures of the CSymbols $\{\boldsymbol{\sigma}_s\}_{s\in\boldsymbol{S_C}}$.

3.5 UG Homomorphisms

Unit Graphs have an underlying oriented labelled graph. It is thus convenient for reasoning applications to introduce the notion of UGs homomorphism. Recall that for non-labelled graphs, an homomorphism from H to G, is an edge-preserving mapping from nodes of H to nodes of G. We will thus introduce the notion of homomorphism of UGs, based on homomorphism of their underlying oriented labelled graphs. Let us first introduce the notion of UGs mapping. Let $H = (U^h, \boldsymbol{l}^h, A^h, C^h, Eq^h)$ and $G = (U^g, \boldsymbol{l}^g, A^g, C^g, Eq^g)$ be two UGs defined over the same support.

Definition 5. A UGs mapping f from H to G, written $f : H \to G$, corresponds to a mapping of their underlying oriented labelled graphs, i.e., a mapping $f : U^h \to U^g$ from the unit nodes of H to the unit nodes of G.

Then there is a homomorphism of UGs if there is a homomorphism of their underlying oriented labelled graphs. To define such a homomorphism, one needs to choose pre-orders over labels for unit nodes and arcs. We use inclusion for unit node markers, $\overset{\cap}{\lesssim}$ for types, and $\overset{\mathcal{C}}{\lesssim}$ for circumstantial relations.

Definition 6. There is a *homomorphism* from H to G if and only if there exists a mapping $\pi : H \to G$ such that all of the following is true:

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

4 Rules and Lexicographic Definitions

Now that we have defined the core structures of the UGs framework, and before we introduce semantics of UGs, we need to sketch some advance concepts of the UGs framework. Namely, the deep semantic representation level (§4.1), rules (§4.2), and unit types definitions (§4.3)

4.1 The deep-surface semantic interface

In the MTT, semantic ASymbols are numbers. For instance, the french lexical unit INSTRUMENT (en: instrument) has a SemASlot 1 that corresponds to the activity for which the instrument is designed. PEIGNE (en: comb) has a more specific meaning than INSTRUMENT, and also two SemASlots: 1 correspond to the person that uses the comb, and 2 is a split variable[11] that corresponds either to the hair or to the person that is to be combed.

As the specialization of PUTs implies the specialization of their actantial structure, the pre-order over semantic unit types can not correspond to a meaning specialization relation. Lefrançois and Gandon (2013a) hence defined a deeper level of representation for the MTT: the *deep semantic level*. At the deep semantic level, the pre-order

[11] See (Mel'čuk, 2004, p.43) for details about split SemASlots

over unit types may correspond to a meaning specialization relation. The *Deep Semantic Unit Type (DSemUT)* associated with a Lexical Unit Type (LexUT) L is denoted $/\text{L}\backslash$, and the set of ASymbols that is used to symbolize ASlots is a set of semantic roles (e.g., $agent$, $experiencer$, $object$). For instance, the DSemUT $/$instrument$\backslash$ associated with the LexUT INSTRUMENT may have an ASlot arbitrarily symbolized $activity$, which would be inherited by the DSemUT $/$peigne$\backslash$. Then $/$peigne$\backslash$ also introduces three new ASlots: one arbitrarily symbolized $possessor$ that corresponds to the ASlot 1 of $($peigne$)$, and two arbitrarily symbolized $combedhair$, and $combedperson$ that correspond to the ASlot 2 of $($peigne$)$.

4.2 Rules

One question one may ask at this point is: how to represent the correspondence between the actantial structure of a DSemUT, and the actantial structure of its associated Surface Semantic Unit Type (SSemUT)? First and parallel with CGs, we will define λ-UGs that enable to distinguish some generic unit nodes of a UG.

Definition 7. A *λ-UG* $L = \{u_1, \ldots, u_n\}G$ of size n defined over a support $\mathcal{S}$ is composed of a UG $G = (U, \boldsymbol{I}, A, C, Eq)$, and a set of n generic unit nodes of G, $\{u_1, \ldots, u_n\}$, denoted the free nodes of L.

λ-UG are actually generalized UGs, and a UG may be considered as a λ-UG of size 0. A rule may then be simply represented by two λ-UGs and a bijection between their free nodes.

Definition 8. A rule is a triple $R \stackrel{\text{def}}{=} (H, C, \kappa)$ where H and C are two λ-UGs of the same size defined over the same support, H is denoted the *hypothesis*, C the *conclusion*, and κ is a bijection from free nodes of H to free nodes of C.

A rule is said to be *applicable* to a UG G if and only if there exists a homomorphism from H to G. Let π be such a homomorphism from H to G. The application of R on G with respect to π is the UG obtained by merging C in G with respect to $\pi \circ \kappa^{-1}$, i.e.,

1. add C to G;
2. for all $(u^c, u^g) \in \pi \circ \kappa^{-1}$, merge u^c and u^g as follows: (i) add a new node u, with $type(u) = type(u^c) \cup type(u^g)$ and $marker(u) = marker(u^c) \cup marker(u^g)$; (ii) replace u^c and u^g by u in any dependency triple in $A \cup C$ and in any element of Eq.

Among other, rules enable to represent correspondences between representations of two adjacent levels, and they shall be automatically generated from the dictionary. Let us just note two issues with the definition of rules for the moment:

- in the correspondence between actantial structures of $/$peigne$\backslash$ and $($peigne$)$, the ASlot 2 of $($peigne$)$ may correspond either to the ASlot $combedhair$ or to $combedperson$ of $/$peigne$\backslash$. One thus need to define two different correspondence rules, one that assumes slot $combedhair$ is filled, and one that assumes slot $combedperson$ is filled.
- if a LexUT has an optional SemASlot, then one would need two different correspondence rules between its associated DSemUT and SSemUT: one that assumes the ASlot is filled and one that assumes the ASlot is not filled.

These remarks are valid not only at the deep-surface semantic interface, and the number of rules would grow in case of several optional ASlots, several split ASlots, or optional split SemASlots for instance. The semantic web SPARQL query language has OPTIONAL and UNION constructors that we could draw inspiration from to extend the definition of rules so as to factorize these cases. Now for the purpose of our presentation we will hold on the simple definition of rules given above, and rely on the fact that we do have means to compute all the possible correspondence rules.

4.3 Unit Types Hierarchy with Definitions

We formalize the notion of *definition* of a Primitive Unit Type (PUT) and include a set of PUTs definitions in the definition of the unit types hierarchy. Definitions are of special interest to represent lexicographic definitions of a LexUT L, which corresponds to the definition of its associated DSemUT $/\text{L}\backslash$. Informally, a definition defines an equivalence between two λ-UG defined over the same support. One of them has a central free unit node typed with the defined PUT and some of its ASlots filled by free unit nodes. The other λ-UG is called the *expansion* of t. There is no circumstantial triple in these two λ-UG because they must not be part of the lexicographic definition of a LexUT.

Definition 9. A definition D_t of a PUT t is a triple $D_t \stackrel{\text{def}}{=} (D_t^-, D_t^+, \kappa)$ where:

- $D_t^- = \{u_t^-, v_1^-, \ldots, v_n^-\}(U^-, \boldsymbol{l}^-, A^-, \varnothing, \varnothing)$ contains only free unit nodes;
- u_t^- is called the the *central unit node* of D_t^-, and is typed $\{t\}$;
- the actantial triples of A^- are of the form (u_t^-, s_i, v_i^-) where the set of s_i is a subset of the ASlots of t;
- for all $(u_t^-, s_i, v_i^-) \in A^-$, the type of v_i^- corresponds to the signature of t for its ASlot s_i;
- $D_t^+ = \{u_t^+, v_1^+, \ldots, v_n^+\}(U^+, \boldsymbol{l}^+, A^+, \varnothing, \varnothing)$ is called the *expansion* of t;
- κ is a bijection from $\{u_t^-, v_1^-, \ldots, v_n^-\}$ to $\{u_t^+, v_1^+, \ldots, v_n^+\}$, such that $\kappa(u_t^-) = u_t^+$, and for all i, $\kappa(v_i^-) = v_i^+$;
- the type of u_t^+ is called the *genus* of t and is denoted $genus(t)$.

Figure 2 is an example of lexicographic definition of PEIGNE: an instrument that a person X uses to detangle the hair Y_1 of a person Y_2.

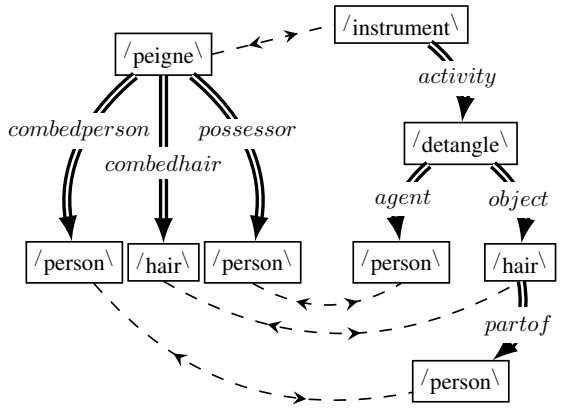


Figure 2: Lexicographic definition of PEIGNE.

Intuitively, a definition corresponds to two reciprocal rules: (D_t^-, D_t^+, κ) and $(D_t^+, D_t^-, \kappa^{-1})$. If there is the defined PUT in a UG then one may infer its definition, and vice versa. Again, there is currently an issue with the definitions with optional ASlots. In fact, one would need two different definitions, one with the ASlot filled, and one with the ASlot not filled. For the purpose of our presentation we will hold on the simple definition of definitions given above, and simply represent multiple definitions for a given PUT.

A set of PUTs definitions $\mathcal{D}$ may thus be added to the unit types hierarchy:

Definition 10. A hierarchy of unit types, denoted $\mathcal{T}$, is a tuple $\mathcal{T} \stackrel{\text{def}}{=} (T_D, \boldsymbol{S}_\mathcal{T}, \boldsymbol{\gamma}, \boldsymbol{\gamma_1}, \boldsymbol{\gamma_0}, C_A, \perp_A^\sqcap, \{\varsigma_t\}_{t \in \boldsymbol{T}}, \mathcal{D})$ that enables to construct a pre-ordered[7] set of unit types $\boldsymbol{T}^\cap$ with their actantial structure, and with $\mathcal{D}$ being definitions of some PUTs.

5 Semantics of UGs

5.1 Closure of a UG

The UGs framework makes 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. Consider the UG $G = (U, \boldsymbol{l}, A, C, Eq)$ defined over the support $\mathcal{S}$ illustrated in figure 3a. Some knowledge in G is implicit:

1. two unit nodes u_1 and u_2 share a common unit marker $Mary$, so one may infer that they represent the same unit. (u_1, u_2) may be added to Eq.
2. every PUT is a subtype of $\top$, so one could add $\top$ to all the types of unit nodes in G.
3. there are two unit nodes v_1 and v_2 that fill the same ASlot $activity$ of the unit node typed /instrument\. So one may infer that v_1 and v_2 represent the same unit. Said otherwise, (v_1, v_2) may be added to Eq.
4. one may recognize the expansion of /peigne\ as defined in figure 2, so this type may be made explicit in the unit node typed /instrument\.

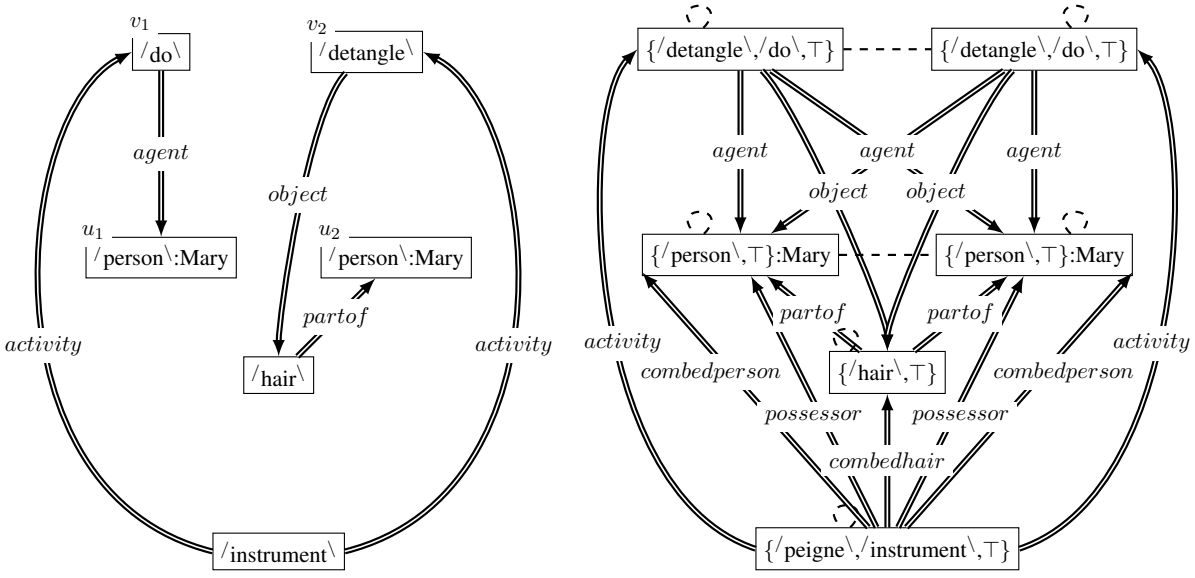
Each of the rules behind these cases explicit knowledge in G. More generally, table 1 lists a set of rules that one may use to explicit knowledge in any UG. Cases 1 to 4 respectively correspond to rules **mrk-eq**, **u-typ**, **a-fp**, and **def-**. The complete set of rules defines the axiomatization of the UGs semantics.

Definition 11 (Closing a UG)**.** The process of applying the set of rules of figure 1 on G until none of them has any effect is called *closing* G, and results in $cl(G)$.

Figure 3b illustrates the closure of G, where all of the inferable knowledge has been made explicit.

u-typ	For all $u \in U$, and $type(u) \overset{\cap}{\lesssim} t^{\cap}$	Add $t^{\cap}$	in $type(u)$
u-bot	For all $u \in U$, if $\bot \in type(u)$, ...	Error: inconsistency !	
eq-ref	For all $u \in U$	Add (u, u)	in Eq
eq-sym	For all $(u_1, u_2) \in Eq$	Add (u_2, u_1)	in Eq
eq-trans	For all (u_1, u_2) and $(u_2, u_3) \in Eq$	Add (u_1, u_3)	in Eq
eq-typ	For all $(u_1, u_2) \in Eq$	Add $type(u_1)$	in $type(u_2)$
eq-mrk	For all $(u_1, u_2) \in Eq$	Add $marker(u_1)$	in $marker(u_2)$
mrk-eq	For all $u_1, u_2 \in U$, if $marker(u_1) \cap marker(u_2) \neq \varnothing$,	Add (u_1, u_2)	in Eq
a-eq-g	For all $(u_1, s, v) \in A$ and $(u_1, u_2) \in Eq$	Add (u_2, s, v)	in A
a-eq-a	For all $(u, s, v_1) \in A$ and $(v_1, v_2) \in Eq$	Add (u, s, v_2)	in A
a-fp	For all (u, s, v_1) and $(u, s, v_2) \in A$	Add (v_1, v_2)	in Eq
a-radix	For all $(u, s, v) \in A$	Add $\boldsymbol{\gamma}(s)$	in $type(u)$
a-obl	For all $u \in U$ and $s \in \boldsymbol{S_{\mathcal{T}}}$, if $\boldsymbol{\gamma_1}(s) \in type(u)$	Ensure there exists (u, s, v) in A	
a-pro	For all $(u, s, v) \in A$, if $\boldsymbol{\gamma_0}(s) \in type(u)$,.....	Error: inconsistency !	
a-sig	For all $(u, s, v) \in A$	Add $\boldsymbol{\varsigma}^{\cap}_{type(u)}(s)$	in $type(v)$
c-eq-g	For all $(u_1, s, v) \in C$ and $(u_1, u_2) \in Eq$	Add (u_2, s, v)	in C
c-eq-c	For all $(u, s, v_1) \in C$ and $(v_1, v_2) \in Eq$	Add (u, s, v_2)	in C
c-dom	For all $(u, s, v) \in C$	Add $domain(s)$	in $type(u)$
c-rng	For all $(u, s, v) \in C$	Add $range(s)$	in $type(v)$
c-sop	For all $(u, s_1, v) \in C$ and $s_1 \overset{c}{\lesssim} s_2$	Add (u, s_2, v)	in C
def+	For all $D_t \in \mathcal{D}$, if (D_t^-, D_t^+, κ) is applicable and $(D_t^+, D_t^-, \kappa^{-1})$ is not, apply (D_t^-, D_t^+, κ)		
def-	For all $D_t \in \mathcal{D}$, if $(D_t^+, D_t^-, \kappa^{-1})$ is applicable and (D_t^-, D_t^+, κ) is not, apply $(D_t^+, D_t^-, \kappa^{-1})$		

Table 1: Semantics of the Unit Graphs.



(a) Incomplete deep semantic representation G

(b) Closure of the unit graph G

Figure 3: Closure of a UG.

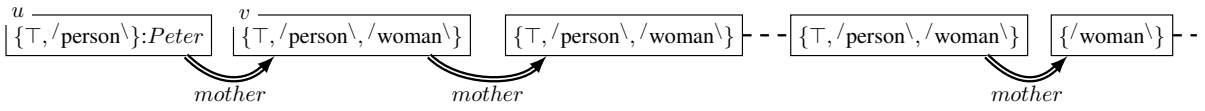


Figure 4: Illustration of an infinite closure of a simple Unit Graph

5.2 Reasoning with Homomorphisms

Now that we provided UGs with semantics and that we have means to explicit knowledge, we will define the prime decision problem of the UGs framework: *Considering two UGs G and H defined over the same support $\mathcal{S}$, does the knowledge of G entails the knowledge of H ?* The notion of *entailment* may intuitively be defined for UGs as follows: G entails H if and only if $cl(G)$ *includes* H.

There are two issues with this definition of entailment. The first is that one needs to define what is the precise meaning of *inclusion of a UG*. The second is that $cl(G)$ may be infinite, thus preventing decidability of entailment.

The answer to the first issue is straightforward as we already defined the UGs homomorphism. $cl(G)$ includes H if and only if there is a homomorphism from H to $cl(G)$.

Now, the second issue is more problematic. Indeed, the closure of a finite UG may be infinite, thus preventing decidability of the decision problem. This problem is illustrated on a simple example in figure 4, suppose one asserts that deep semantic PUTs $/\text{person}\backslash$ has a obligatory ASlot $mother$ with $\varsigma_{/\text{person}\backslash}(mother) = /\text{woman}\backslash$, and of course, $/\text{woman}\backslash \lesssim /\text{person}\backslash$. Consider the UG $G = (\{u\}, \boldsymbol{l}, \varnothing, \varnothing, \{(u,u)\})$, such that $marker(u) = Peter$ and $type(u) = \{\top, /\text{person}\backslash\}$. One knows that the unit represented by u should have a unit of type $/\text{woman}\backslash$ that fills its obligatory ASlot $mother$. So rule **add-a** is applicable and one could add a unit node v to represent that argument, with $(u, mother, v) \in A$. Rule **u-typ** will then make v be of type $\{\top, /\text{person}\backslash, /\text{woman}\backslash\}$, and rule **add-a** is again applicable on v. Thus $cl(G)$ is an infinite chain of unit nodes having type $\{\top, /\text{person}\backslash, /\text{woman}\backslash\}$ and that fill the ASlot $mother$ of one another.

In the set of inference rules of table 1, only three rules add unit nodes to the UG when triggered: **add-g**,**def-** and **def+**. An open problem is thus to find a sufficient condition on the unit types hierarchy and the set of definitions so that we are ensured that the closure of a finite UG is finite.

6 Conclusion

We studied how to formalize, in a knowledge engineering perspective, the dependency structures and the linguistic predicates, in order to represent, manipulate, query, and reason with linguistic knowledge.

We provided a rationale the introduction of the new graph-based Unit Graphs (UGs) KR formalism, and gave an overview of the foundational concepts of the UGs framework. The linguistic predicates are represented by unit types, and are described in a unit types hierarchy. Circumstantial relations are another kind of dependency relation that are described in a hierarchy, and along with a set of unit identifiers these two structures form a UGs support on which UGs may be defined. As UG have an underlying oriented labelled graph, one could introduce the notion of UG homomorphism which is useful to define the applicability of rules and the entailment problem.

The strong coherence in the unit types hierarchy justifies the introduction of a deep semantic representation level that is deeper than the semantic level, and in which one may represent the actual meaning of Lexical Unit Type (LexUT). It is at the deep semantic level that the lexicographic definitions of LexUTs shall be represented, and we gave a definition of the definition of the associated Deep Semantic Unit Type (DSemUT) of a LexUT.

Once we added a set of unit types definitions in the unit types hierarchy, we introduced the semantics of UGs and we posed the entailment problem for UGs. A UGs along with the support on which it is defined represents explicit knowledge, and additional knowledge may be inferred. We introduced a set of entailment rules that one may use to compute the closure $cl(G)$ of a UG G, i.e., make explicit all of the knowledge that is implicit. We then defined the entailment problem of H by G as a directed labelled graphs homomorphism problem between H, and the closure of G: $cl(G)$. In case $cl(G)$ is finite, the entailment problem is thus NP-complete.

In this paper we also sketched two directions for future research:

- Many rules may be needed to represent correspondences between the deep semantic and the semantic representation levels in case some Semantic ASlots are optional or split. More research is needed to adapt the SPARQL OPTIONAL and UNION constructors in these cases. The same can be said about definitions of DSemUTs that have optional ASlots.
- The closure may be infinite for finite UGs. If

that occurs it makes the closure undecidable, along with the entailment problem. We are currently working of the definition of restrictions of the unit types hierarchy and the set of definitions in order to ensure that any UG has a finite closure.

References

[Apresian et al.2003] Juri Apresian, Igor Boguslavsky, Leonid Iomdin, Alexander Lazursky, Vladimir Sannikov, Victor Sizov, and Leonid Tsinman. 2003. ETAP-3 linguistic processor: A full-fledged NLP implementation of the MTT. In *First International Conference on Meaning–Text Theory (MTT'2003)*, pages 279–288.

[Barque and Polguère2008] Lucie Barque and Alain Polguère. 2008. Enrichissement formel des définitions du Trésor de la Langue Française informatisé (TLFi) dans une perspective lexicographique. *Lexique*, 22.

[Barque et al.2010] Lucie Barque, Alexis Nasr, and Alain Polguère. 2010. From the Definitions of the 'Trésor de la Langue Française' To a Semantic Database of the French Language. In Fryske Akademy, editor, *Proceedings of the XIV Euralex International Congress*, Fryske Akademy, pages 245–252, Leeuwarden, Pays-Bas. Anne Dykstra et Tanneke Schoonheim, dir.

[Boguslavsky et al.2004] Igor Boguslavsky, Leonid Iomdin, and Viktor Sizov. 2004. Multilinguality in ETAP-3: reuse of lexical resources. In Gilles Sérasset, editor, *Proc. COLING 2004 Multilingual Linguistic Ressources*, pages 1–8, Geneva, Switzerland. COLING.

[Boguslavsky2011] Igor Boguslavsky. 2011. Semantic Analysis Based on Linguistic and Ontological Resources. In Igor Boguslavsky and Leo Wanner, editors, *Proceedings of the 5th International Conference on Meaning-Text Theory (MTT'2011)*, pages 25–36, Barcelona, Spain. INALCO.

[Bohnet and Wanner2010] Bernd Bohnet and Leo Wanner. 2010. Open source graph transducer interpreter and grammar development environment. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, pages 19–21, Valletta, Malta. European Language Resources Association (ELRA).

[Chein and Mugnier2008] Michel Chein and Marie-Laure Mugnier. 2008. *Graph-based Knowledge Representation: Computational Foundations of Conceptual Graphs*. Springer-Verlag New York Incorporated.

[Kahane and Polguère2001] Sylvain Kahane and Alain Polguère. 2001. Formal foundation of lexical functions. In *Proceedings of ACL/EACL 2001 Workshop on Collocation*, pages 8–15.

[Lefrançois and Gandon2011] Maxime Lefrançois and Fabien Gandon. 2011. ILexicOn: Toward an ECD-Compliant Interlingual Lexical Ontology Described with Semantic Web Formalisms. In Igor Boguslavsky and Leo Wanner, editors, *Proceedings of the 5th International Conference on Meaning-Text Theory (MTT'2011)*, pages 155–164, Barcelona, Spain. INALCO.

[Lefrançois and Gandon2013a] Maxime Lefrançois and Fabien Gandon. 2013a. The Unit Graphs Framework: A graph-based Knowledge Representation Formalism designed for the Meaning-Text Theory. In *Proceedings of the 6th International Conference on Meaning-Text Theory (MTT'2013)*, Prague, Czech Republic.

[Lefrançois and Gandon2013b] Maxime Lefrançois and Fabien Gandon. 2013b. The Unit Graphs Mathematical Framework. Research Report RR-8212, Inria.

[Lefrançois2013] Maxime Lefrançois. 2013. Représentation des connaissances du DEC: Concepts fondamentaux du formalisme des Graphes d'Unités. In *Proceedings of the 15ème Rencontre des Étudiants Chercheurs en Informatique pour le Traitement Automatique des Langues (RECITAL'2013)*, Les Sables d'Olonne, France.

[Lux-Pogodalla and Polguère2011] Veronika Lux-Pogodalla and Alain Polguère. 2011. Construction of a French Lexical Network: Methodological Issues. In *Proceedings of the International Workshop on Lexical Resources*, Ljubljana.

[Mel'čuk1996] Igor Mel'čuk. 1996. Lexical Functions: A Tool for the Description of Lexical Relations in a Lexicon. In Leo Wanner, editor, *Lexical Functions in Lexicography and Natural Language Processing*, pages 37–102. Benjamins Academic Publishers, Amsterdam/Philadelphia.

[Mel'čuk2004] Igor Mel'čuk. 2004. Actants in Semantics and Syntax I: Actants in Semantics. *Linguistics*, 42(1):247–291.

[Mel'čuk2006] Igor Mel'čuk. 2006. Explanatory combinatorial dictionary. *Open Problems in Linguistics and Lexicography*, pages 225–355.

[Polguère2009] Alain Polguère. 2009. Lexical systems: graph models of natural language lexicons. *Language resources and evaluation*, 43(1):41–55.

[Sowa1984] John F Sowa. 1984. *Conceptual structures: information processing in mind and machine*. System programming series. Addison-Wesley Pub., Reading, MA.