When the Generative Lexicon meets Computational Semantics

Renaud Marlet
SIGNES group, LaBRI / INRIA
351 cours de la Libération
33405 Talence cedex
France
Renaud.Marlet@inria.fr

Abstract

Computational semantics and lexical semantics have so far mostly been studied separately. As a result, computational semantics often constructs meanings with poor or little lexical sense, while lexical semantics generally only apply to simple and small phrases (as opposed to complete sentences). We present here a general framework to incorporate lexical semantics information originating from a generative lexicon into a standard analysis of computational semantics. This framework is illustrated on two examples, covering type coercion and selective binding. In this framework, the coupling between the analysis and the lexicon is kept low to facilitate separate evolutions and to adapt to the partial availability of lexical semantics information.

1 Introduction

The semantic analysis of text relies on two key issues: the syntax-semantics interface and the lexicon-semantics interface. However, to some extent, these two issues have mostly been studied separately.

On the one hand, computational semantics (CS) proposes semantic analyses that specify how to map a sentence, usually modeled as a syntax tree, into a meaning, commonly expressed as a logical formula (Benthem and Meulen, 1997). These approaches are often based on the composition of typed λ-terms, along the lines of Montague semantics (Montague, 1974). The range of syntactic constructions that they cover is substantial and growing: quantifying determiners, negation, referential pronouns and ellipsis, subordinate clauses, questions, etc. Different kinds of underlying logics are also studied, such as intensional logic to express belief, modal logic to express possibility and necessity, temporal logic to express past and future, etc. However, these approaches generally make little or no use of lexical semantic information. Lexemes are taken as "opaque" predicates and the relevance of produced formulas is routinely poor because of a lack of support for polysemy, and in particular metonymy. For instance, analysing "John begins a book" as $\exists x\, \mathit{begin}(\mathit{John}, x) \wedge \mathit{book}(x)$ is mostly useless, if not wrong, because it tells nothing about the actual action being performed (reading, writing, etc.) nor about the nature of "book" (physical object or information).

On the other hand, the Generative Lexicon Theory (GL) provides a framework for the creative composition of lexical meanings (Pustejovsky, 1995). Various phenomena of polysemy can be explained in this setting, as well as their contextual disambiguation. Rich lexical semantic information and powerful composition mechanisms obviate the need to explicitly list numerous individual meanings for different lexeme associations, as must be the case for "real" collocations. However, the GL focuses on the composition of "referential" lexemes, mainly verbs, nouns and adjectives. It provides

little support for other parts of speech such as determiners, pronouns or conjunctions. As a result, it is usually not possible to perform the semantic analysis of a complete sentence within the GL alone.

On way to bridge the gap between these two complementary visions is to incorporate the features of one into the other. It seems unreasonable to duplicate, within the GL, the work already done in CS. This would be a huge task, for which the GL offers no particular advantage. The modeling of "purely syntactic" phenomena might in fact be cluttered when forced to be formulated with GL entities.

Conversely, it is not obvious how to integrate the GL into a CS analysis. Indeed, the lexicon only plays a minor role in these approaches. It is used only at the beginning of the analysis process, to provide a basic semantic function or predicate for each lexeme in the sentence. It is totally useless afterwards: the terms are simply composed according to their type and to the syntactic structure of the sentence. Only β-reduction delves into the basic semantic terms, but without refining their meaning. Even if the GL can nonetheless be successfully integrated into a semantic analysis, a number of questions are raised. Would it put constraints on the GL that would reduce its power and autonomy? Would the integration be sensitive to evolutions in the GL model? Besides, as there are many proposals for such semantic analyses, which CS approaches are best suited for a GL integration? And what amount of work can be reused when integrating the GL into several such CS analyses?

Our goal in this paper is to present a general framework to incorporate GL-related information into a CS analysis. We first present a typical CS analysis and illustrate cases where it does not construct appropriate interpretations (§2). Then we describe how these cases can be fixed using GL-related information (§3 and §4). Finally, we generalise on these examples and propose a comprehensive framework that integrates GL compositions and CS interpretation, while limiting their coupling (§5). Given any well-defined CS analysis and any well-defined GL entries and composition processes, this provides an effective mechanisation[1] of the semantic analysis of complete sentences that takes into account lexical semantics.

2 Computational Semantics Interpretation

Many approaches in computational semantics define the meaning of a sentence as a type-driven compositional construction. Each element in the sentence is interpreted as a function, and represented by a λ-term. These semantic functions are combined according to the (binary) tree structure of a syntactic analysis of the sentence, and also depending on their type. Typical combination operations are functional application and functional composition. At the top level, the resulting term, after β-reduction, is a logical formula representing the meaning of the whole sentence, in some given logic. Let us consider the following example.

(1) Every linguist drinks a glass.

In the Government and Binding Theory (GB) — for instance —, this sentence is given both a surface structure and two logical forms, corresponding to two different readings (Huang, 1994). In the logical forms, the quantified determiner phrases are moved up the syntax tree, leaving indexed traces t_1 and t_2, as illustrated on Figure 1. (In the following, we only consider the interpretation where each linguist has his/her own glass.)

[1] We make a difference here between formalisation and mechanisation. Formalisation often relates to the use of formal notations as well as formal practices to provide a *better ground* for *descriptive or explanatory* theories, whereas mechanisation refers to the specification of *well-defined, operative* procedures to *systematically process* the langage (although with some approximation).

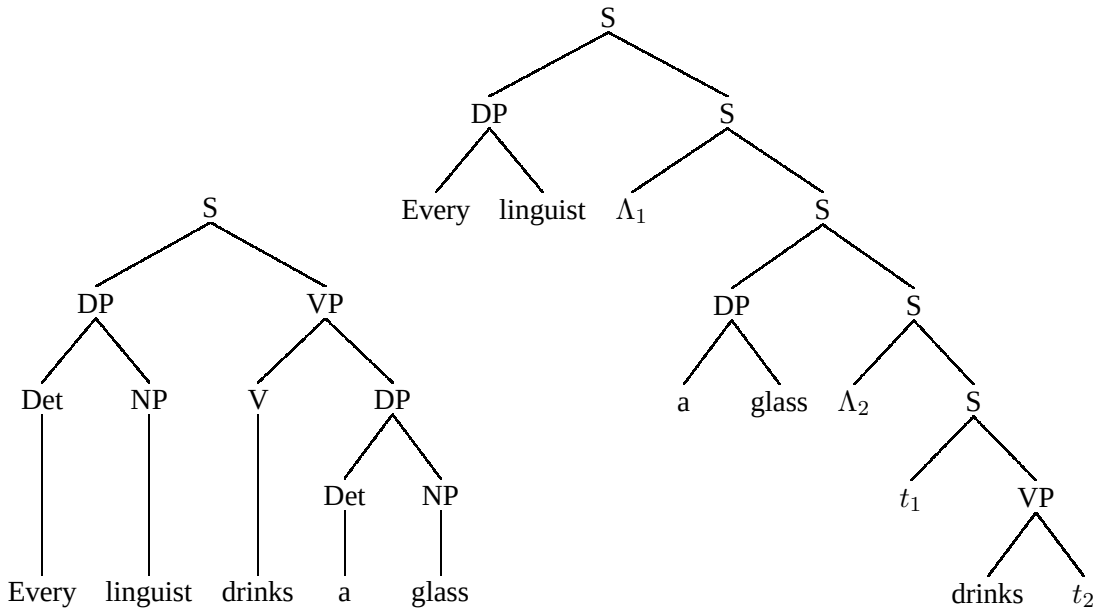

Figure 1: Surface structure example, as well as one of its two possible logical forms

For any lexeme or phrase w in the logical form of the sentence, $[\![w]\!]$ stands for the denotation of w, i.e., the semantic function that is the interpretation of w. When w is a lexeme, $[\![w]\!]$ is found in the lexicon; when it is a phrase, it is calculated from the interpretation of subphrases. The meaning of lexemes is read from a lexicon that includes both types and functional values, as illustrated on Figure 2. Following a common practice in the tradition of Montague, atomic types are just "e" (entity)

$$
\begin{array}{lll}
[\![\textbf{\textit{every}}]\!] & : \langle\langle \mathrm{e},\mathrm{t}\rangle,\langle\langle \mathrm{e},\mathrm{t}\rangle,\mathrm{t}\rangle\rangle & = \lambda P.\lambda Q.\forall x\, P(x) \Rightarrow Q(x) \\
[\![\textbf{\textit{linguist}}]\!] & : \langle \mathrm{e},\mathrm{t}\rangle & = \lambda x.\textit{linguist}(x) \\
[\![\textbf{\textit{drinks}}]\!] & : \langle \mathrm{e},\langle \mathrm{e},\mathrm{t}\rangle\rangle & = \lambda x.\lambda z.\exists e\, \textit{drink}(e,z,x) \\
[\![\textbf{\textit{a}}]\!] & : \langle\langle \mathrm{e},\mathrm{t}\rangle,\langle\langle \mathrm{e},\mathrm{t}\rangle,\mathrm{t}\rangle\rangle & = \lambda P.\lambda Q.\exists x\, P(x) \wedge Q(x) \\
[\![\textbf{\textit{glass}}]\!] & : \langle \mathrm{e},\mathrm{t}\rangle & = \lambda x.\textit{glass}(x)
\end{array}
\qquad
\begin{array}{lll}
[\![t_i]\!] & : \mathrm{e} & = x_i \\
[\![\Lambda_i]\!] & : \langle \mathrm{t},\langle \mathrm{e},\mathrm{t}\rangle\rangle & = \lambda P.\lambda x_i.P
\end{array}
$$

Figure 2: Semantic lexicon example, with types and functional values

and "t" (truth-value)[2], and functional type $\sigma \to \tau$ is written $\langle \sigma, \tau \rangle$. Some traits of verbs are omitted here, such as tense, mood, etc. Following Davidson, we also include an event argument e in the verb predicates[3] (Davidson, 1980), but we extend it to states too (e.g., $\textit{contain}(e,x,y)$)[4]. Additionally, there are specific rules to cope with references t_i and binders Λ_i.

[2]The truth-value type "t" is not to be confused with traces "t_i".

[3]An event argument "e" is not to be confused with the entity type "e".

[4]It is not required here that we extend it to nouns as well, e.g., $\textit{glass}(e,x)$.

The compositional meaning of sentence (1) is defined based on the syntax tree structure (the logical form depicted on Figure 1) as well as on the types and values listed on Figure 2. In this particular example, all compositions happen to be functional applications from left to right, except for the application of the VP, which is in the opposite direction:

$$\begin{aligned}&[\![\textbf{\textit{every linguist drinks a glass}}]\!]\\&\quad = ([\![\textbf{\textit{every}}]\!]\,[\![\textbf{\textit{linguist}}]\!])\,([\![\Lambda_1]\!]\,(([\![\textbf{\textit{a}}]\!]\,[\![\textbf{\textit{glass}}]\!])\,([\![\Lambda_2]\!]\,(([\![\textbf{\textit{drinks}}]\!]\,[\![t_2]\!])\,[\![t_1]\!]))))\end{aligned}$$

After β-reduction, the interpretation of sentence (1) is found to be the following.

$$\begin{aligned}&\#\ [\![\textbf{\textit{every linguist drinks a glass}}]\!]\\&\quad = ([\![\textbf{\textit{every}}]\!]\,[\![\textbf{\textit{linguist}}]\!])(\lambda x_1.([\![\textbf{\textit{a}}]\!]\,[\![\textbf{\textit{glass}}]\!])(\lambda x_2.(([\![\textbf{\textit{drinks}}]\!]\,x_2)\,x_1)))\\&\quad = \forall x_1\ \textit{linguist}(x_1) \Rightarrow (\ \exists x_2\ \textit{glass}(x_2) \wedge \exists e\ \textit{drink}(e, x_1, x_2)\ )\end{aligned}$$

This interpretation is wrong[5] because it does not convey the idea that the linguist drinks the contents of the glass. It is also incorrect in the sense that "drinks" actually expects a beverage but is provided a container. What is missing is a way to coerce the container into a beverage.

3 Supporting type coercion via the Generative Lexicon

The GL provides an account for this kind of metonymy phenomenon. Lexical entries for lexeme **drink** and **glass** are defined below. (Only qualia and features that are relevant for what follows have been mentioned here[6].)

$$\left[\begin{array}{l}\textbf{glass}\\ \text{EVENTSTR}\ \left[\text{E1} = e : \mathsf{state}\ \right]\\ \text{ARGSTR}\ \left[\begin{array}{l}\text{ARG1} = x : \mathsf{container}\\ \text{D-ARG1} = y : \mathsf{beverage}\end{array}\right]\\ \text{QUALIA}\ \left[\text{TELIC} = \textit{contain}(e, x, y)\ \right]\end{array}\right] \qquad \left[\begin{array}{l}\textbf{drink}\\ \text{EVENTSTR}\ \left[\text{E1} = e : \mathsf{process}\ \right]\\ \text{ARGSTR}\ \left[\begin{array}{l}\text{ARG1} = x : \mathsf{animated}\\ \text{ARG2} = y : \mathsf{beverage}\end{array}\right]\\ \text{QUALIA}\ \left[\text{AGENT} = \textit{drink}(e, x, y)\ \right]\end{array}\right]$$

When trying to compose entries **drink** and **glass**, there is type clash as **drink** expects a beverage but gets a container. The composition is enabled by the *type coercion* mechanism of the GL (Godard and Jayez, 1993), that somehow converts the container into a beverage, shifting the composition parameter from the true argument of **glass** (x : container) to the default argument (y : beverage). This results in a new lexical construction, with the following kind of structure.

[5] The sharp sign "#" denotes a semantically incorrect formula.

[6] For instance, a richer entry for **glass** would include an AGENT quale to express that a glass is an artefact (blown). Also the TELIC quale could be refined into two subqualia (Bassac and Bouillon, 2007): an "agentive of the telic" quale, expressing a precondition (the fact that somebody had to pour the beverage in the glass), and a "formal of the telic" quale, expressing a result (the fact that the glass contains the beverage). In the following, the TELIC feature is to be understood as the formal part of the telic quale. Besides, the telic of **glass** could also include an adjunct predicate $\textit{drink}(e', z, y)$ to express that the purpose of the glass is not only to contain a beverage (or liquid) but also to allow somebody to drink out of it. As for **drink**, a richer entry would also include a TELIC feature to express that the purpose of drinking is (often) to quench one's thirst.

$$\left[\begin{array}{l}\textbf{drink (a) glass}\\ \textsc{eventstr}\ \left[\begin{array}{l}\textsc{e}1 = e_1 : \textsf{process}\\ \textsc{e}2 = \boxed{1}\, e_2 : \textsf{state}\end{array}\right]\\ \textsc{argstr}\ \left[\begin{array}{l}\textsc{arg}1 = x_1 : \textsf{animated}\\ \textsc{arg}2 = \boxed{2}\, y : \textsf{beverage}\\ \textsc{arg}3 = \left[\begin{array}{l}\textbf{glass}\\ \textsc{eventstr}\ \left[\textsc{e}1 = \boxed{1}\, e_2 : \textsf{state}\right]\\ \textsc{argstr}\ \left[\begin{array}{l}\textsc{arg}1 = x_2 : \textsf{container}\\ \textsc{d-arg}1 = \boxed{2}\, y : \textsf{beverage}\end{array}\right]\\ \textsc{qualia}\ \left[\textsc{telic} = \textit{contain}(e_2, x_2, y)\right]\end{array}\right]\end{array}\right]\\ \textsc{qualia}\ \left[\textsc{agent} = \textit{drink}(e_1, x_1, y)\right]\end{array}\right]$$

Informally, if we omit the quantifiers, the meaning associated to **drink (a) glass** is thus given by the formula $\textit{glass}(x_2) \wedge \textit{contain}(e_2, x_2, y) \wedge \textit{drink}(e_1, x_1, y)$. Alternatively, the telic quale of **glass** could have been added as adjunct to the agentive quale of **drink**, yielding a similar formula.

Now the question is whether this formula can somehow be "injected" into the CS interpretation. More precisely, we are looking for a conversion (or coercion) function that could adapt the interpretation of ***glass*** when eventually composed with the interpretation of ***drinks***. In other words, we are looking for a function conv such that:

$$\begin{aligned}&\mathsf{conv}(\llbracket \textbf{\textit{a}} \rrbracket\ \llbracket \textbf{\textit{glass}} \rrbracket)\ (\lambda x_2.((\llbracket \textbf{\textit{drinks}} \rrbracket\, x_2)\, x_1))\\ &\quad = \ \exists x_2\, \exists y\, \exists e_1\, \exists e_2\, \textit{glass}(x_2) \wedge \textit{contain}(e_2, x_2, y) \wedge \textit{drink}(e_1, x_1, y)\end{aligned}$$

As can be shown after a few β-reductions and α-renamings, one solution for this equation is:

$$\mathsf{conv} = \lambda P.\lambda R.P(\lambda z.\exists y\, \exists e\, \textit{contain}(e, z, y) \wedge R(y))$$

If conv is applied at the "right" time during CS interpretation, the analysis of (1) becomes correct:

$$\begin{aligned}&\llbracket \textbf{\textit{every linguist drinks a glass}} \rrbracket\\ &\quad = (\llbracket \textbf{\textit{every}} \rrbracket\ \llbracket \textbf{\textit{linguist}} \rrbracket)(\lambda x_1.\mathsf{conv}(\llbracket \textbf{\textit{a}} \rrbracket\ \llbracket \textbf{\textit{glass}} \rrbracket)(\lambda x_2.((\llbracket \textbf{\textit{drinks}} \rrbracket\, x_2)\, x_1)))\\ &\quad = \forall z\, \textit{linguist}(z) \Rightarrow \exists x\, \textit{glass}(x) \wedge \exists e_2\, \exists y\, \textit{contain}(e_2, x, y) \wedge \exists e_1\, \textit{drink}(e_1, z, y)\end{aligned}$$

We can recognize as a subterm of conv the telic quale of **glass**, selected by the type coercion mechanism. This actually generalizes to other common type coercions.

Interestingly, the same conv operator also works for other quantifying determiners. Consider, for instance, the following sentence.

(2) A linguist drinks every glass.

It has the following, correct interpretation, when conv is "injected" at the same time as above.

$$\begin{aligned}&\llbracket \textbf{\textit{A linguist drinks every glass}} \rrbracket\\ &\quad = (\llbracket \textbf{\textit{a}} \rrbracket\ \llbracket \textbf{\textit{linguist}} \rrbracket)(\lambda x_1.\mathsf{conv}(\llbracket \textbf{\textit{every}} \rrbracket\ \llbracket \textbf{\textit{glass}} \rrbracket)(\lambda x_2.((\llbracket \textbf{\textit{drinks}} \rrbracket\, x_2)\, x_1)))\\ &\quad = \exists z\, \forall x\, \textit{linguist}(z) \wedge (\textit{glass}(x) \Rightarrow (\exists e_2\, \exists y\, \textit{contain}(e_2, x, y) \wedge \exists e_1\, \textit{drink}(e_1, z, y)))\end{aligned}$$

The same "fixed" interpretation also works for the following sentences:

(3) A linguist drinks a glass.

(4) Every linguist drinks every glass.

It actually is not really surprising because the quantifier applying to ***linguist*** comes into play after the VP is fully interpreted. Knowing that the "fixed" interpretation works for ***drinks a glass*** (1) and ***drinks every glass*** (2), it consequently also works for (3) and (4).

4 Supporting selective binding via the Generative Lexicon

Another classical example is the "fast typist". In a typical CS interpretation, this phrase is analysed as follows.

$$[\![\textbf{\textit{fast}}]\!] : \langle(\langle \mathrm{e}, \mathrm{t}\rangle), (\langle \mathrm{e}, \mathrm{t}\rangle)\rangle = \lambda P.\lambda x.P(x) \wedge \mathit{fast}(x)$$

$$[\![\textbf{\textit{typist}}]\!] : \langle \mathrm{e}, \mathrm{t}\rangle = \lambda x.\mathit{typist}(x)$$

$$\# \quad [\![\textbf{\textit{fast typist}}]\!] = [\![\textbf{\textit{fast}}]\!]([\![\textbf{\textit{typist}}]\!]) = \lambda x.\mathit{typist}(x) \wedge \mathit{fast}(x)$$

This makes little sense because of a type clash: *fast* actually expects an event when *typist* expects an individual. The *selective binding* rule of the GL (Pustejovsky, 1995) is devised to handle such a case: when an entry γ_1 of type $\langle \tau_1, \tau_1 \rangle$ is combined with an entry γ_2 of type τ_2, if the qualia structure of γ_2 has a quale q that involves a type τ_1, then γ_1 and γ_2 can be composed into an entry of type τ_2, based on γ_2, where γ_1 applies to quale q. In our example, the event expected by **fast** can be found in the telic quale of **typist**.

$$\left[\begin{array}{l} \textbf{typist} \\ \textsc{eventstr}\ \left[\mathrm{E1} = e : \mathsf{process}\ \right] \\ \textsc{argstr}\ \left[\textsc{arg}1 = x : \mathsf{human}\ \right] \\ \textsc{qualia}\ \left[\textsc{telic} = \mathit{type}(e, x)\ \right] \end{array}\right]$$

Classically, the resulting GL composition is then as follows.

$$\left[\begin{array}{l} \textbf{fast typist} \\ \textsc{eventstr}\ \left[\mathrm{E1} = e : \mathsf{process}\ \right] \\ \textsc{argstr}\ \left[\textsc{arg}1 = x : \mathsf{human}\ \right] \\ \textsc{qualia}\ \left[\textsc{telic} = \mathit{type}(e, x) \wedge \mathit{fast}(e)\ \right] \end{array}\right]$$

Although the telic role expresses a persistent property, the corresponding event is only to be understood as a possible (Busa, 1997), as opposed to systematic, i.e., realised in all circumstances. This corresponds to a modal interpretation, which may be represented by the possibility operator "$\Diamond$" (Bouillon, 1997). This leads to the following semantics for "fast typist". (We omit here additional constraints expressing that x is human, etc.)

$$[\![\textbf{\textit{fast typist}}]\!] = \lambda x.\mathit{typist}(x) \wedge \exists e\, \Diamond(\mathit{type}(e, x) \wedge \mathit{fast}(e))$$

The question again is how to fix the wrong CS interpretation using the GL mechanism, but staying at the CS level.

One possible answer is, like in Section 3, to apply a conversion operator before combining the semantics of the noun with that of the adjective:

$$\mathsf{conv}' = \lambda P.\lambda R.\lambda x.R(x) \wedge \exists e.\Diamond P(\lambda e'.\textit{type}(e', x))(e)$$

$$[\![\textbf{\textit{fast typist}}]\!] = (\mathsf{conv}'[\![\textbf{\textit{fast}}]\!])([\![\textbf{\textit{typist}}]\!]) = \lambda x.\textit{typist}(x) \wedge \exists e\, \Diamond(\textit{type}(e, x) \wedge \textit{fast}(e))$$

We can also recognise as a subterm of conv' the telic quale of **typist**.

More generally, when lexicon entries γ_1 and γ_2 are composed according to selective binding via quale q, their corresponding CS interpretation can be composed as follows.

$$\mathsf{conv}' = \lambda P.\lambda R.\lambda x.R(x) \wedge \exists e\, (P(q(\gamma_2)(x))(e)$$

$$[\![\gamma_1\ \gamma_2]\!] = (\mathsf{conv}'[\![\gamma_1]\!])([\![\gamma_2]\!])$$

In the case where $q = \text{TELIC}$, then $q(\gamma)$ is to be understood as the contents of the telic quale modified by the possibility modality.

5 Synchronising the two calculi into a general framework

We now generalise on the preceding examples and define a way to associate CS and GL. Given a CS interpretation $[\![\cdot]\!]$, as well as a lexicon L equiped with a set of GL composition mechanisms, we build a new interpretation $[\![\cdot]\!]_{\mathsf{mix}}$ that combines them. Intuitively, the *mixed interpretation* $[\![\cdot]\!]_{\mathsf{mix}}$ is constructed as an instrumented semantics[7] that modifies the CS interpretation $[\![\cdot]\!]$. The CS interpretation and the GL compositions are "synchronised" in the sense that both processes are somehow performed in parallel on the same syntactic structure, synchronizing at each node level, i.e., stopping and cooperating before pursuing. The mixed interpretation is formally defined as follows.

The domain of the mixed interpretation is the product domain: elements are pairs (φ, γ) where φ is a semantic function in the domain of $[\![\cdot]\!]$, and γ is a lexical entry in L, possibly obtained by the GL composition of other L entries.

The mixed interpretation works bottom up on the same syntactic tree structure as the original CS interpretation. We assume that movements, if any, have already been performed. (The CS interpretation actually does not have to be based on GB; it just has to be functional and compositional, e.g., based on categorial grammars.) The mixed interpretation is defined inductively as described below.

For any leaf w in the syntax tree, we define:

$$[\![w]\!]_{\mathsf{mix}} = ([\![w]\!], \mathsf{entry}(w))$$

It is important to note that the lexicon does not have to be complete. If w is absent from L, then $\mathsf{entry}(w) = \bot$, where $\bot$ denotes an undefined entry.

For any non-leaf (binary) tree w_0 in the syntax tree, w_0 results from the association of two subterms w_1 and w_2, that represent subphrases, specifier, morphemes, etc. Assuming that the mixed interpretations $[\![w_1]\!]_{\mathsf{mix}} = (\varphi_1, \gamma_1)$ and $[\![w_2]\!]_{\mathsf{mix}} = (\varphi_2, \gamma_2)$ have been calculated, we want to express $[\![w_0]\!]_{\mathsf{mix}} = (\varphi_0, \gamma_0)$.

As modeled and illustrated in the previous sections, the CS interpretation defines:

$$[\![w_0]\!] = \mathsf{compos}_{\mathsf{func}}([\![w_1]\!], [\![w_2]\!])$$

where $\mathsf{compos}_{\mathsf{func}}$ is some functional operation depending on the types of $[\![w_1]\!]$ and $[\![w_2]\!]$.

[7] as defined in the abstract interpretation framework used for programming languages (Jones and Nielson, 1994)

Regarding the GL, a lexical composition $\mathsf{compos}_{\mathsf{lex}}$ for L entries γ_1 and γ_2 may or may not be defined. There are thus two cases:

(a) If a composition $\mathsf{compos}_{\mathsf{lex}}(\gamma_1, \gamma_2)$ is defined in L, then the mixed interpretation $[\![w_0]\!]_{\mathsf{mix}} = (\varphi_0, \gamma_0)$ is defined as follows:

$$\varphi_0 = \mathsf{compos}_{\mathsf{func}}(\chi_1(\varphi_1), \chi_2(\varphi_2))$$
$$\gamma_0 = \mathsf{compos}_{\mathsf{lex}}(\gamma_1, \gamma_2)$$
$$\chi_i = \mathsf{effect}_i(\mathsf{compos}_{\mathsf{lex}})$$

where $\mathsf{effect}_i(\mathsf{compos}_{\mathsf{lex}})$ denotes the effect that the composition $\mathsf{compos}_{\mathsf{lex}}$ has on its ith argument. In other words, φ_1 and φ_2 are composed via $\mathsf{compos}_{\mathsf{func}}$ as in the original CS interpretation, except that they are first converted as specified by χ_1 and χ_2. These conversions account for the GL composition of lexical entries γ_1 and γ_2.

For instance (cf. §3), when a composition $\mathsf{compos}_{\mathsf{lex}}$ translates into the adjunction of some quale q_2 of γ_2 to some quale q_1 of γ_1, where $q_2(\gamma_2)$ is expressed as $pred(e, x, y)$ and x is the variable denoted by γ_2, then the situation is as follows.

$$q_1(\gamma_0) = q_1(\gamma_1) \wedge q_2(\gamma_2) \quad \text{and} \quad \forall\, q \neq q_1 \;\; q(\gamma_0) = q_1(\gamma_1)$$
$$\chi_1 = \lambda x.x$$
$$\chi_2 = \lambda P.\lambda R.P(\lambda x.\exists y\, \exists e\, q(\gamma_2) \wedge R(y))$$

In other words, γ_0 has the same qualia as γ_1, except for the adjunction of $q_2(\gamma_2)$ to quale q_1. Moreover, the interpretation φ_1 is unaltered (χ_1 is the identity function) whereas the interpretation φ_2 is converted to incorporate the contents of quale q_2 of γ_2, and to apply to the third argument of $q_2(\gamma_2)$ rather than to the second one. The other arguments are existentially quantified.

(b) In case the lexicon L does not define entry γ_1 or γ_2, or if no composition mechanism $\mathsf{compos}_{\mathsf{lex}}$ is defined for γ_1 and γ_2, then the standard interpretation is used as default (χ_1 and χ_2 are identity functions). Moreover, if there is an entry in the lexicon for the lexical head of the phrase, then it is used as a default entry γ_0, which further allows possible compositions. In other words:

$$\varphi_0 = \mathsf{compos}_{\mathsf{func}}(\varphi_1, \varphi_2)$$
$$\gamma_0 = \mathsf{entry}(\mathsf{lexhead}(w_0)) \text{ if any, otherwise } \bot$$

It is the best that we can do given the limited available information.

For instance, an auxiliary verb w_1 whose lexicon entry is γ_1 can be combined with a verb w_2 whose lexicon entry is γ_2. In this case, the lexical head of $w_0 = w_1 w_2$ is the verb: $\mathsf{lexhead}(w_0) = w_2$. The lexicon entry corresponding to the composition of w_1 and w_2 is then $\gamma_0 = \mathsf{entry}(\mathsf{lexhead}(w_0)) = \mathsf{entry}(w_2) = \gamma_2$, i.e., the lexicon entry of the verb.

Similarly, when a determiner is combined with a noun to form a determiner phrase (DP), the lexicon entry of the noun (lexical head of the DP) is used as the entry for the whole DP[8]. This is actually used to correctly interprete example sentence (1): generative lexicons usually do not define any entry for determiner ***a***, let alone any composition mechanism to combine an entry for ***a*** with an entry for ***glass***. In our setting, we thus have $\gamma_1 = \mathsf{entry}(\boldsymbol{a}) = \bot$ and $\gamma_2 = \mathsf{entry}(\boldsymbol{glass})$, as well as $\mathsf{compos}_{\mathsf{lex}}(\gamma_1, \gamma_2) = \bot$. The result is then defined as $\gamma_0 = \mathsf{entry}(\mathsf{lexhead}(\boldsymbol{a\ glass})) = \mathsf{entry}(\boldsymbol{glass}) =$ **glass**. This allows entry **glass** to be appropriately combined with **drink** later in the interpretation.

[8] As far as this mixed interpretation is concerned, we are agnostic regardind DPs and NPs: if the phrase is analysed as a noun phrase (NP) rather than a determiner phrase, the lexical head is still the noun.

As a fallback, it is thus always possible to consider a GL composition as undefined, i.e., $\gamma_0 = \bot$. In the extreme case where, for all items of a given sentence, there are no applicable entries γ in the semantic lexicon, or no applicable composition mechanisms $\mathsf{compos}_{\mathsf{lex}}$, the mixed interpretation $[\![\cdot]\!]_{\mathsf{mix}}$ coincides with the standard interpretation $[\![\cdot]\!]$.

6 Conclusion

Given an arbitrary computational semantics analysis (with only a few assumptions) and an arbitrary generative lexicon (a subset of entries and composition mechanisms), as well as a small number of operations to map GL composition information into CS terms, we have defined an effective, mixed semantic analyser, that applies to the same syntactic constructs as the original CS process but produces GL-refined analyses.

Arguably, this mixed interpretation provides in some sense the best of both worlds: CS and GL are both used for what they are good, i.e., respectively syntax-semantics interface and lexicon-semantics interface. This of course does not prohibit uses of the GL to also address syntactic phenomena.

It can be noted that the coupling between CS and GL is very limited, which facilitates the separate evolution both of CS features (support of additional syntactic constructs, refinements in the target logics, etc.) and GL features (general model, composition mechanisms, actual lexicon entries, etc.).

In particular, the mixed interpretation is applicable even when only partial information is available from the lexicon, for instance when lexical entries or GL compositions are missing. This has two major advantages. First, it allows the gradual addition of lexical semantics information in a semantic analysis system. The semantic lexicon does not have to be complete before the first sentence can be processed. Second, it also allows the gradual addition of more GL composition mechanisms. This makes sense for parts of speech that are not (yet) supported by the GL, as well as for mechanisms that have been described but not yet been mechanised.

Future work includes the explicit definition of more mappings from GL compositions to CS conversions (i.e., functions effect_i) as well as a comparison of the treatment of quantifiers for dotted types as in (Asher and Pustejovsky, 2000) and (Jacquey, 2001).

Also interesting would be a comparison with approaches where the lexicon-semantics and syntax-semantics interfaces are studied within the same framework, such as the Meaning-Text Theory (MTT) (Mel'čuk, 1988). In this theoretical framework, practical procedures have been defined to transform a meaning, expressed as a semanteme graph, into a textual representation (Bohnet and Wanner, 2001). Even if this differs from the above in that it addresses text generation rather than text analysis[9], and semanteme graphs rather than logical formulas, this does not preclude inspiring comparisons. The fact is that the MTT includes a rich lexicon model, the Explanatory and Combinatorial Dictionary (ECD), that has been formalized (Mel'čuk and Polguère, 1987) and that has computational instances (Altman and Polguère, 1997). In this lexicon model, there is a particular emphasis on collocation information, expressed as lexical functions, which allows in particular the selection of appropriate lexemes and forms when generating text (Polguère, 1998). However, it seems that phenomena like metonymy have to be explicited for each lexeme, although polysemy templates allow for some factorisation.

[9] MTT equative rules can in theory also be used for analysis, but in practice this direction is not as easy and as developped as generation is. Conversely, the Generative Lexicon theory as well as most Computational Semantics approaches are more geared towards analysis than generation.

Acknowledgements

We wish to thank Christian Bassac for fruitful discussions and helpful comments on earlier versions of this paper.

References

Joel Altman and Alain Polguère. 1997. La bdéf : base de définitions dérivée du dictionnaire explicatif et combinatoire. In *Proceedings of the 1st International Conference on Meaning-Text Theory*, pages 43–54.

Nicholas Asher and James Pustejovsky. 2000. The metaphysics of words in context. *Submitted to the Journal of Logic, Language and Information*.

Christian Bassac and Pierrette Bouillon. 2007. The telic relationship in compounds. In James Pustejovsky et al., editor, *Generative approaches to the lexicon*. Kluwer. To appear.

Johan van Benthem and Alice ter Meulen, editors. 1997. *Handbook of logic and language*. The MIT Press.

Bernd Bohnet and Leo Wanner. 2001. On using a parallel graph rewriting formalism in generation. In *Proceedings of the 8th European workshop on Natural Language Generation (EWNLG'01)*, pages 1–11, Morristown, NJ, USA. Association for Computational Linguistics.

Pierrette Bouillon. 1997. *Polymorphie et sémantique lexicale : le cas des adjectifs*. Ph.D. thesis, Université de Paris 7.

Federica Busa. 1997. The semantics of agentive nominals in the generative lexicon. In Patrick St. Dizier, editor, *Predicative Forms in Natural Language*. Kluwer.

Donald Davidson. 1980. *Essays on Actions and Events*. Clarendon, Oxford.

Danièle Godard and Jacques Jayez. 1993. Towards a proper treatment of coercion phenomena. In *Proceedings of the sixth conference of the European chapter of the Association for Computational Linguistics*, pages 168–177, Morristown, NJ, USA, April. ACL.

C.-T. James Huang. 1994. Logical form. In Gert Webelhuth, editor, *Government and binding theory and the minimalist program*, pages 127–173. Blackwell, Oxford.

Evelyne Jacquey. 2001. *Ambiguïtés lexicales et Traitement Automatique des Langues : Modélisation de la polysémie logique et applications aux déverbaux d'action ambigus en français*. Ph.D. thesis, Université de Nancy 2, December.

Neil D. Jones and Flemming Nielson. 1994. Abstract interpretation: a semantics-based tool for program analysis. In *Handbook of Logic in Computer Science*, pages 527–629. Oxford University Press.

Igor Mel'čuk and Alain Polguère. 1987. A formal lexicon in Meaning-Text Theory (or how to do lexica with words). *Computational Linguistics*, 13(3–4):261–275.

Igor Mel'čuk. 1988. *Dependency Syntax: Theory and Practice*. The SUNY Press.

Richard Montague. 1974. *Formal Philosophy. Selected Papers of Richard Montague*. Yale University Press. Edited and with an introduction by Richmond H. Thomason.

Alain Polguère. 1998. Pour un modèle stratifié de la lexicalisation en génération de texte. *Traitement Automatique des Langues (T.A.L.)*, 39(2):57–76.

James Pustejovsky. 1995. *The Generative Lexicon*. The MIT Press.