A Montague-based model of Generative Lexical Semantics

Bruno Mery, Christian Bassac, Christian Retoré

A Montague-based model of Generative Lexical Semantics

Workshop on New Directions in Type-Theoretic Grammars
NDTTG – part of ESSLLI 2007, Dublin (EI)

Bruno Mery Christian Bassac Christian Retoré

SIGNES group, LaBRI, INRIA, ERSS
Université de Bordeaux
351 cours de la Libération
F-33405 Talence Cédex, France

Abstract. Computational semantics has long relied upon the Montague correspondance between syntax and semantics, which is not by itself well suited for the computing of some phenomena, such as logical polysemy, addressed by recent advances in lexical semantics. Our aim is to integrate the results of lexical semantics studies such as the Generative Lexicon Theory in a straightforward way with the existing computing of logical forms, in order to form a Montagovian framework for lexical semantics. In addition, we will outline a way to integrate other kinds of semantic information.

Computational semantics has long relied upon Montague's logical syntax to semantics system, or its variants. While the computing of logical forms from syntax is well known and has recently been formalized in type theory (e.g., by [Ranta, 2004]), the process is not fine-grained enough for a large part of language. Specifically, the Generative Lexicon Theory detailed in [Pustejovsky, 1995] and Transfers of Meanings introduced in [Nunberg, 1993] both present convincing data in favor of co-compositionality of meaning, the fact that the same lexical items often convey different yet related senses depending upon what other terms they are applied to. [Pustejovsky, 1995] also provides a complete description of the mechanisms of lexical semantics, sufficient to express the processes behind composition and co-composition.

Our goal is to present an efficient way to express those principles in a generic Montagovian computational semantics framework. In this paper, we first introduce the Generative Lexicon Theory, and explore the assumed link with Montague-style logical forms. We then detail our model and its variants, including how to extend it to other theories and phenomena, and finish with a discussion of the generality and possibilities of use of such a system.

1 The Generative Lexicon Theory

Pustejovsky's Generative Lexicon Theory (GL) gives a strongly-motivated model for many cases of logical polysemy, together with a rich structuration of the meaning of concepts. It uses an inheritance-based hierarchy of types, each corresponding to a lexical concept. Each lexical entry (associated with each type) also includes:

- the number and types of arguments needed (for a predicate),
- the characterization of the event structure associated with the concept, if any, and
- the associated *qualia*, or modes of explanation of the concept: what its properties are (*formal*), what it is made of / part of (*constitutive*), what it can be used for (*telic*), what can cause it to come into being (*agentive*)... the idea being that a word can, under certain conditions, refer to any of its qualia (e.g., "ship" can be derived from "sail").

Some of this type of information might be *underspecified*, i.e. not defined in the lexicon, and filled in during the computation of the meaning of an utterance.

In addition, some lexical entries are of a so-called *complex* (or *dot*) type, expressing two or more *aspects* of different, hierarchically not comparable types (none is the subtype of the other). For instance, if one supposes that there are physical objects of type P and informational contents of type I, then the lexical item *book* would be of type $I \bullet P$.

2 Linking GL with Montague semantics

2.1 Compositional semantics

The compositional and co-compositional mechanisms introduced by GL presupposes that a very basic kind of structure is available, such as a syntactic tree. This structure indicates which terms are actually applied to which in an utterance, i.e., which are the arguments of which predicates.

The basic Montagovian notion of application and abstraction then applies, considering that the lexicon is responsible for the typing of the term (the argument structures defining types for the term and its arguments).

In certain situations where normal applications would result in a type clash, though, the other mechanisms of GL are employed, and it is understood that a certain number of *type coercion* operations are licensed :

- if a predicate needs an argument of a type α, and the actually selected argument is of type α', with α and α' compatible (i.e., one is a subtype of the other in the type hierarchy), then the application is valid by *type accommodation*;
- if a predicate needs an argument of a type α, and the actually selected argument is of type β, with α being part of a certain quale of β, then the application is valid by *exploitation* over said quale;
- if a predicate needs an argument of a type α, and the actually selected argument is of type $\alpha \bullet \beta$, then the application is valid by $\bullet$-*exploitation*.

2.2 Current formulations

Type-theoritical formalizations of these semantics, however, are far from trivial. The original theory, as well as more recent formalisms such as [Asher and Pustejovsky, 2005] or [Pustejovsky, 2006], fails to remain within a standard logical calculus.

This does not mean that these formulations are inadequate, but the fact is that the level of detail for some phenomena is such that it looks difficult to integrate literally all constraints expressed in any logic type system and keep it yet simple and sound. Therefore, those formalisms struggle with the necessity of expressing the results of long studies in lexical semantics and the need for actually useable logical rules.

2.3 Our approach

We do not want to model the entirety of GL in our typing system and logical rules. Our main concern being economy, we have thought that we might keep the information present in types as simple as possible, and add the necessary data carried by the lexicon on a term-to-term basis.

Thus, we take as a basis the already complete and sound Montague logic system, with simple types, and keep the same application and abstraction rules. The one exception is that, in addition of the classical logical term, each variable will be able to convey additional lexical information by providing additional, optional terms (involving additional logical constants)that might be used when type coercion is required. We shall detail this model in the next section...

3 A model with optional terms

3.1 Basic assumptions

Our model is based upon classical Montague semantics with simple types. It supposes a hierarchy similar to an ontology, such as described in [Pustejovsky, 2006], where $\top$ is the universal type with three main subtypes, $Entity$, $Event$, $Property$, the various subtypes of which form the complete type lattice.

We shall use the standard simply-typed λ-calculus with a slightly different use of the lexicon that enhances the application of a term to another in two different ways :

- *self-adaptation*: a lexical item might provide a number of optional terms that allow a type change when the classical application would yield a type error, and
- *selection-projection*: a predicate may select for an argument of an unspecified type and attempt *afterwards* to project a certain type upon it, using a distinct set of terms that the predicate and argument might both contribute to.

This system is derived from the idea, expressed in both [Nunberg, 1993] and [Pustejovsky, 1995], that each lexical entry can licence some type shifts or coercions. It does not change overmuch the general rules for Montague semantics – as the increase in expressive power is due to extra terms, rather than extra data in the typing system that would have to be captured by adequate rules.

The main point is that each lexical item contributes *at least* a term, that *must* be consumed (as per usual), plus a (finite) number of *optional* terms that might be used.

Each use of such a term might:

- change the type of the non-optional term, in order to solve some typing mismatch;
- change the associated structure of the term, for example by specifying a quale;
- change the term itself (number of arguments, etc.);
- enable future reference for the (newly created) concept in the discourse.

In the following, type accommodation is supposed, i.e., any given type shall share the property of its supertypes, including type-coercing terms.

3.2 Self-adaptation

Each instance x of some lexically-defined type τ might provide some additional terms. Operators $f_1 \dots f_k$ are defined within τ, together with the main term (in GL terms, within the *argument structure*), and are of type $\tau \rightarrow \sigma_i$ for some $\sigma_i \neq \tau$. Thus, when needed, any $f_i(x)$ might be substituted to x.

For example, suppose that objects of type "computer", Ct, have an extensive *constitutive* quale, including at least an object of type "processing unit", CPU. Then we have an operator $f : Ct \rightarrow CPU$, which, when used on a computer x, would yield a pointer to its processor $f(x)$. Thus, supposing clock-related predicates apply only on objects of type CPU, we would have a derivation[1] :

Example 1. A 2-GHz computer
$\exists x, \lambda y^{CPU}.Clock(y)\ [x^{Ct}]$
$\exists x^{Ct}, \lambda y^{CPU}.Clock(y)\ [(f(x))^{CPU}]$

That mechanism suffices to licence most cases of qualia-exploitation (with an operator for every exploitation possible on the agent, telic or constituents), as well as some specific lexical rules such as *grinding*, where an object is subject to an irrevocable change, such as herb→food, which is modeled by a type-coercing operator, as in :

Example 2. Freshly prepared lemongrass
$\exists x, \lambda y^{F}.Fresh(y)[x^{H}]$
$\exists x^{H}, \lambda y^{F}.Fresh(y)[(f(x))^{F}]$

As every occurrence of the concerned object is changed, that use of type-coercing operator might be likened to *passing a variable by reference* in programming.

[1] In the following, arguments are enclosed in square brackets for the sake of clarity.

3.3 Projection after selection

Another group of optional terms might be provided, either by an argument or predicate, in order to modify the argument after its selection. Those operators $g_1 \dots g_l$ are also lexically defined.

They might be used when the predicate is on the model of $\lambda x^{\top}.P(\Pi_{\tau}(x))$ (that is, the predicate P selects for an argument of any type and attempts to apply an optional term g_i afterwards, which would yield an output of type τ). One might think of $\Pi_{\tau}(x)$ as "x viewed as of type τ", and the construct results in a type clash if it fails.

As the semantic change of the argument is local to its selection by the concerned predicate, the object in itself remains intact, and thus, that process is like *passing a variable by value* in programming.

This mechanism can express the phenomenon known as *co-predication*: if two or more predicates select the same lexical item, they may enforce different types upon it. It happens frequently with *complex* types, i.e., objects with more than one aspect (such as *book* or *town*, where we would have a select-project operator available for every alternate type).

Recall that when *self-adaptation* is required, conversely, the item changes its type for *every occurrence*, and co-predication in sentences such as (3) is impossible :

Example 3. ?? The tuna we had yesterday night was lightning fast and delicious

For an exemple of type change after selection in a co-predicative sentence, we might have a type "town", Tn, a type "people", Pp, with an operator $g : Tn \rightarrow Pp$ that represent the relationship between a city and its inhabitants. Then we could have:

Example 4. Boston is a large city that mostly votes Democrat
$\exists x^{Tn}\ \lambda y^{Tn}.City(y)[x] \wedge \lambda z^{\top}.Vote(\Pi_{Pp}(z))[x]$
$\exists x^{Tn}\ City(x) \wedge Vote(\Pi_{Pp}(x^{Tn})$
$\exists x^{Tn}\ City(x) \wedge Vote(g(x))$

This change of type after selection, which enables to reference the newly selected aspect later on in the discourse, can be used to analyze some of the most complex quantificational puzzles in co-predicative sentences.

4 Expanding the framework beyond the lexicon

We believe our model to be sufficient to take into account most of the phenomena that are the target of GL. However, some additional areas could be further explored in the search for actual automated understanding of the language.

In this section, we will explain how our system could be extended in order to include some of those theoretical and practical points.

4.1 A possible implementation in functional programming

The simple co-compositional logical framework we have described here can very easily be implemented in functional programming such as Lisp or CaML, as functional application is the one operation needed, and the addition of optional terms corresponds to methods attached to the classes associated with the type of the variable in an object-oriented view. A translation module, which would extract the optional terms from the definitions of the types in GL, would also be quite straightforward.

4.2 Integrating multiple adjustments

The model outlined here is also quite generic in that it proposes simple adjustments for co-composition and coercion via the information encoded in each lexical entry, but can also be adapted to other kinds of linguistic and non-linguistic data. Thus, additional, optional terms might be deduced from:

- the current discourse (e.g., if someone is defined as a teacher, the associated lexical operators should be linked to the name of the person for future use),
- the situation (non-verbal signs might trigger transfers),
- some cultural assumption (the lexical definition of *village* might contain very different constituents depending on the cultural group),
- or additional pragmatic reasoning.

The system in itself remains the same, it is a simple matter of adding optional terms corresponding to the different aspects wanted, while keeping a reasonable total number of possible combinations. Of particular interest would be a translation of SDRT and λ-DRT in that framework.

4.3 Metaphors and idiomatic expressions

There are two ways to account for idiomatic expressions and metaphors specific to a given language or dialect.

The most obvious is to list, for each language or fragment, every such expression and use, and to associate additional terms that would map the entire expression (as a fixed string) into its logical equivalent. This approach would be costly, both in establishing a complete lexicon containing all such information and in terms of complexity, but it would probably be useful for complete understanding and necessary for machine-driven translation.

The other approach, more economic and possibly more efficient in a day-to-day basis, is to consider that each (valid syntactic-wise) utterance is semantically correct, and to try and derive a plausible, if incomplete, interpretation. This method would introduce unspecified operators as a last resort in a type clash, and the resulting sense might later be clarified through machine-assisted dialogue.

In any case, taking into account idiomatic expressions will prove necessary in order to grasp the correct meaning of many common sentences.

4.4 Interpretation choice and scoring

During the computation of the actual meaning of a given logical form, when several operators f_i, g_i are available for use, several defeasible interpretations might be possible. A specialized module might have to *choose* between the possibilities: supposing a "town" can be either a geographical entity, the set of its inhabitants, the ruling body (either mayor or council), or the set of its civil servants, then in

Example 5. Philadelphia wants a new bridge, but the mayor is opposing it

the prominent typing (geographical entity) should be rejected, as well as the type change associated to "mayor", but any other typing operators associated to subtypes of "people" can be valid interpretations.

It is also often the case that several interpretations remain available when everything is taken into account. To help classify interpretations, a notion of "semantic cost" might be added to each type-coercing operator, and a minimal cost would indicate higher likeliness of the final logical form (even as any of the other ones could still be considered as "correct").

4.5 Generative power and complexity

As this model is a front-end to semantic and pragmatic theories rather than a generative grammar in itself, it presupposes some kind of syntactic formalism in order to generate the bare logical form (i.e., what lexical items are arguments of one another). GL, as well as most other theories that we might model, will not by itself change the generative capacity of that syntactic formalism, being intended primarily for analysis and understanding.

The semantic power, i.e., the number of correct interpretations for the same syntactically valid utterances, however, will be greater. The extent of the semantic power of the overall formalism depends upon the number of choices added by the available optional terms, and this also very much affects the computational complexity of any implementation: for the process of interpretation to remain feasible, the number of available operators to any type and, most importantly, of arguments to any predicate, should be bounded – and tightly so. The general complexity of the computing of interpretations for any given interpretations should remain, at most, polynomial in space and time in order to be useful.

In the real world, heuristics shall be employed in order to check only the most likely derivations and to use presuppositions in order to help ensure the efficiency of the process. Moreover, some moderate use of underspecification and symbolic synthetical reasoning (not deriving every ambiguous reading at each step of the calculus) should be helpful in the establishment of a balanced strategy for actual computing, for which further research is needed.

Conclusion

The model outlined herein is a simple way to implement both GL and other theories of the modifications to the semantics of lexical items. We think that a complete formalization is not so difficult to attain, and that it should at the least be an efficient formulation of GL, with possible practical applications.

References

[Asher and Pustejovsky, 2005] Asher, N. and Pustejovsky, J. (2005). Word Meaning and Commonsense Metaphysics. Semantics Archive.

[Nunberg, 1993] Nunberg, G. (1993). Transfers of meaning. In *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, pages 191–192, Morristown, NJ, USA. Association for Computational Linguistics.

[Pustejovsky, 1995] Pustejovsky, J. (1995). *The Generative Lexicon*. MIT Press.

[Pustejovsky, 2006] Pustejovsky, J. (2006). Type Theory and Lexical Decomposition. Semantics Archive.

[Ranta, 2004] Ranta, A. (2004). Computational semantics in type theory. *Mathématiques et Sciences Sociales*, 165:31–57.