Lessons Learned from a Prototype Implementation of Montagovian Lexical Semantics

Bruno Mery

Lessons Learned from a Prototype Implementation of Montagovian Lexical Semantics

Bruno Mery
bruno.mery@u-bordeaux.fr

LaBRI & Université de Bordeaux
Bordeaux, France.

Abstract. We present a general-purpose implementation of the process of lexical semantics analysis theorised in the Montagovian Generative Lexicon ΛTY_n (hereafter MGL). The prototype software itself serves as a proof of concept of the MGL theory. The implementation process, the data structures and algorithms, also provide valuable results as to the expressive power required by MGL. While the implementation of terms and types for the purpose of meaning assembly assumed by MGL is in itself straightforward, some lexical phenomena imply additional mechanisms in order to process the logical representation using implicit knowledge. We therefore also present a minimal architecture for knowledge representation, and how it can be applied to different phenomena.

Keywords: Lexical Semantics, Prototype Software, Montagovian Generative Lexicon, Knowledge Representation for Natural Language Semantics.

1 Theories and Implementations of Lexical Semantics

Formal lexical semantics theories aim to integrate to the toolbox of compositional analysis of natural language developed since Montague considerations of (logical) polysemy. Based on original studies such as [4,8], then on a theory thoroughly developed in [22], there have been many formulations that build upon powerful type-theoretic foundations, with a generative, dynamic account of the lexicon at their heart. Such recent type-theoretic accounts of lexical meaning include Type Composition Logic (TCL) presented in [1], Dynamic Type Semantics (DTS) presented in [3], Type Theory with Records (TTR) presented in [7], Unified Type Theory (UTT) presented in [12], and the Montagovian Generative Lexicon (MGL) presented in [24].

Several partial or complete implementations of those theories have been provided for demonstration purposes, using logical or functional programming, or theorem provers such as Coq ([6] is an example among many). Concerning MGL, however, one of the stated goals was (paraphrasing slightly [24]) to provide an integrated treatment from syntax to semantics extending existing analysers based on Montagovian semantics such as [19] with mechanisms for lexical semantics that *are easily implemented in a typed functional programming language like Haskell*. Our goal in this publication is to present an actual prototype implementation (using functional and object programming in Scala) of the lexical semantics of that framework.

We detail some of the necessary data structures and algorithms used, what we learned from this implementation on the underlying logic properties of MGL, and sketch an architecture for simple knowledge representation that is necessary for the representation of certain lexical phenomena. The demonstrably functioning prototype illustrates both the validity of type-theoretic formulations of lexical meaning, and the deep interaction of lexical meaning with at least some sort of knowledge representation already evoked in [5].

2 A MGL Prototype

2.1 The Montagovian Generative Lexicon

MGL makes use of ΛTY_n (an adaptation of the many-sorted logic TY_n proposed in [21] in second-order λ-calculus, given in the syntax of System-F). The idea is to perform an usual Montague analysis (performing syntax analysis via proof-search and substituting semantic main λ-terms to syntactic categories). Lexical mechanisms are then implemented in the meaning-assembly phase via a rich system of types based on ontologically different sorts and optional λ-terms that model lexical adaptation. The mechanisms, given in Fig. 1, can be summarised as follows:

- First, the input utterance is super-tagged and analysed using categorial-grammar mechanisms, which is the only step of proof-search of the process, yielding a syntactic term whose components are syntactic categories. The lexicon is then used in standard Montagovian fashion to substitute λ-terms, yielding a main semantic term, typed with many sorts and the type $\mathbf{t}$ for propositions.
- Second, as a many-sorted logic is used, some type mismatches might (and should) occur, allowing mechanisms of lexical semantics to disambiguate between terms. The lexicon provides *optional* λ-terms that are used as *lexical transformations*. These optional terms are inserted depending on their typing and yield a λ-term with no type mismatches.
- Finally, β-reduction yields a normal, η-long λ-term of type $\mathbf{t}$ (the type of propositions), i. e. a logical formula that can be used in any usual semantics, such as model-theoretic or game-theoretic semantics.

As the first step is already well-studied and implemented, the object of concern is the second step: given a term reflecting the syntactic structure of an utterance, to construct a semantic λ-term in a many-sorted logic, making use of available transformations, and yielding a suitable formula. This is the object of our prototype implementation.

2.2 Modelling Types and Terms

The data structures and algorithms responsible for implementing the terms and types of ΛTY_n are the core mechanisms of the software. They are given as two Scala sealed abstract classes, `TermW` and `TypeW`, with a flat hierarchy of case classes implementing the various possible terms and types; this simple categorisation allows us to easily construct and detect patterns of objects.

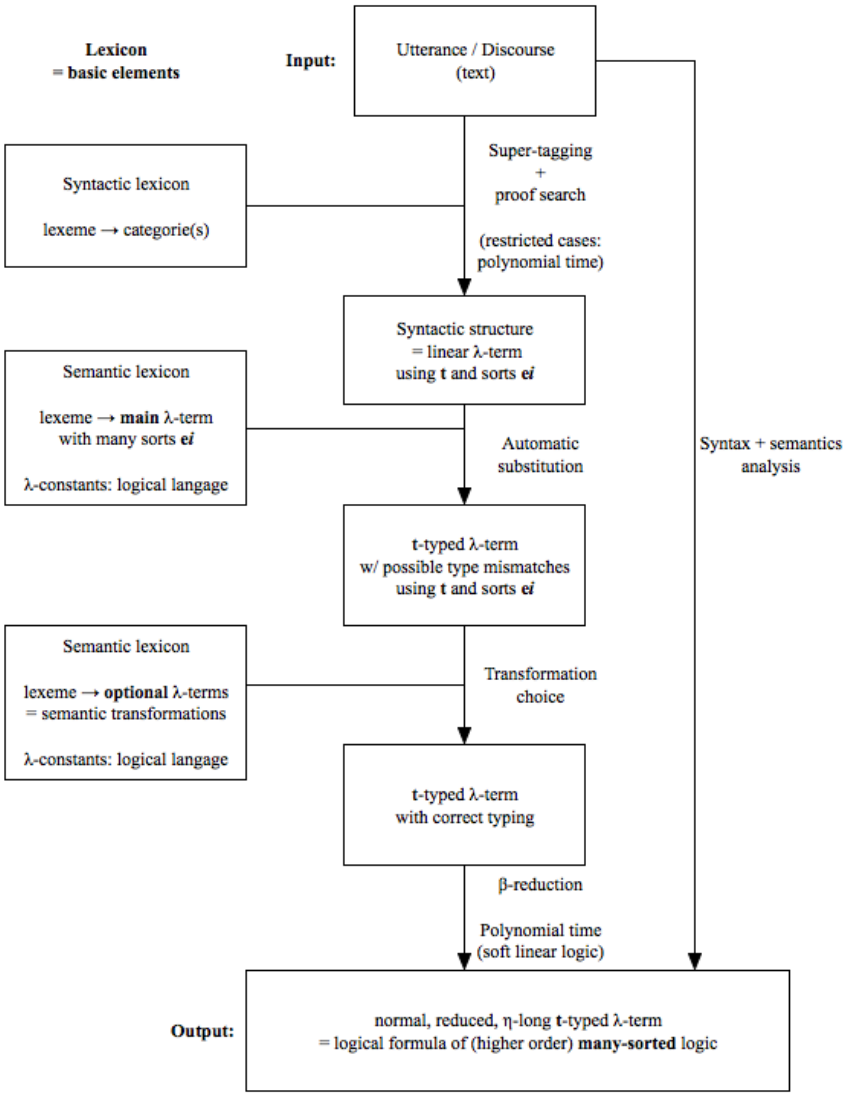

Fig. 1. MGL Process Summary

Terms and types are constructed as binary trees (abstractions and applications of more than one argument to a given term/type can be easily curryied).

For terms, leaves are `AtomicTerms` (constants), `TermVarIds` (variables) with an identifier and type, or specific `Transformations` and `Slots`, while inner nodes are `TermBindings` (λ-abstracted terms), or `TermApplications` of a predicate and argument. For types, leaves are constant `Sorts`, pre-defined objects such as `PropositionType` for **t**, or second-order variable identifiers `TypeVarIds`, while nodes are `TypeFunctions` between two types A and B modelling $A \to B$, or `TypeApplications` modelling $A\ \{B\}$.

A simplified UML class diagram presents this straightforward architecture in Fig. 2.

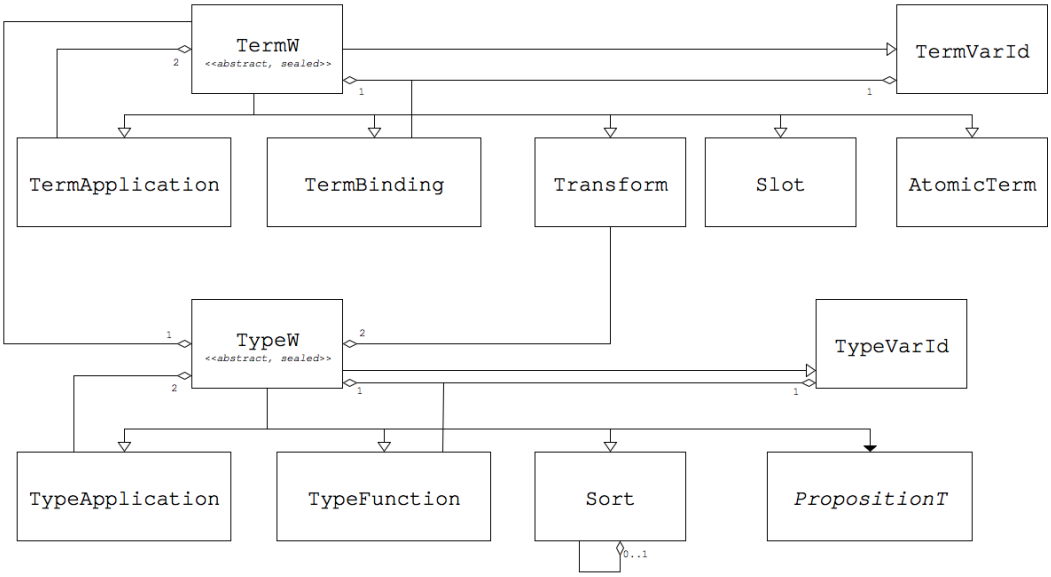

Fig. 2. Class diagram of the core package for terms and types.

Several algorithms are provided in order to work with types and terms; they are mostly simple recursive tree-walking algorithms, making the most of memoisation when possible (e. g., lists of available ressources are incrementally built as terms and types are constructed in order to minimise computations). Algorithms include the type-checking of applications, comparison between types, automated α-conversion of variables in order to prevent issues of scope, replacement of term and type variables, β-reduction, and the automated specialisation of types for polymorphic terms (i. e., a predicate with a type containing one or several type variables will be specialised to the correct types if applied to an argument with a compatible but specified type).

Most are linear in complexity (with exceptions in the adaptative mechanisms below); all algorithms are at most polynomial in time.

2.3 Explicit Adaptation

The core of MGL is to provide *transformations* as optional terms, on top of the main λ-term associated to each lexeme. A canonical example is *the book is heavy and interesting*. Supposing three basic sorts:

- R for readable materials,
- φ for physical objects,
- I for informational contents;

the book can be modelled as the_book^{R}, *heavy* as heavy$^{\varphi\rightarrow\mathbf{t}}$, *interesting* as interesting$^{I\rightarrow\mathbf{t}}$. The example utterance is a case of *co-predication*, as two predicates are simultaneously asserted on two different *facets* (with different, incompatible sorts) of a same object, and MGL will resolve this by having the lexicon provide two optional terms associated with *book* in order to access these two facets: $f^{R\rightarrow\varphi}_{phys}$ and $f^{R\rightarrow I}_{info}$. A polymorphic conjunction $\&^{\Pi} = \Lambda\alpha\Lambda\beta\lambda P^{\alpha\rightarrow\mathbf{t}}\lambda Q^{\beta\rightarrow\mathbf{t}}\Lambda\xi\lambda x^{\xi}\lambda f^{\xi\rightarrow\alpha}\lambda g^{\xi\rightarrow\beta}.(\text{and}^{\mathbf{t}\rightarrow\mathbf{t}\rightarrow\mathbf{t}}\ (P\ (f\ x))(Q\ (g\ x)))$ is needed for the co-predication. This yields, after suitable substitutions, application, and reduction, the term $(\text{and}\ (\text{heavy}(f_{phys}\ \text{book}))\ (\text{interesting}\ (f_{info}\ \text{book})))$, which is normal and of type $\mathbf{t}$.

In our implementation, there are several important differences with the theory outlined above. As we distinguish between type constants and variables, there is no need to explicitly abstract types. This is because the only second-order operation ever used in ΛTY_n is specialisation (i. e., the replacement – or instantiation – of type variables). Moreover, second-order (type) variables are all introduced by λ-bound first-order (term) variables.

We also distinguish between term variables that are necessary for the definition of an abstracted term (such as P, Q and x, the two predicates and the argument of the conjonction above) and *adaptation slots*, the places where optional λ-terms (such as f and g above) can be inserted. This is because the optional terms can be provided by various different mechanisms, and might not be provided at all if the term is well-formed (there is no lexical adaptation taking place in utterances such as *heavy and black rock*; in that case, MGL provides an useful, if slightly redundant, *id* optional polymorphic term that can be inserted in order to get the identity on any type).

We provide optional terms as `Transforms`, which are distinguished from other terms. Each term has a list of available transformations, constructed recursively from the leaves (the transformations available to atomic terms should be given in the lexicon). We also distinguish `Slots` for explicit adaptations; the list of slots is maintained during the construction of the terms. Our polymorphic *and* conjunction then becomes:

```
lambda p^(B->t).lambda q^(G->t).lambda x^A.((And^{(t->(t->t))}
    (p^(B->t) (f^{(A->B)} x^A))) (q^(G->t) (g^{(A->G)} x^A)))
```

During the attempted resolution of the application of the conjunction to terms for *heavy* and *interesting*, the polymorphic *and* is specialised to sorts representing φ and I, and cannot be reduced further with the application of the argument *book*. A further algorithm is provided in order to model the choice of transformations, trying to match all available transformations to the adaptation slots. As all permutations are considered, this is potentially the most costly computation taking place. The result is a *list* of possible interpretations (given as term applications with slots filled by transformations): there might be zero, one, or finitely many. A further check on the list of terms obtained will filter those, if any, with a suitable typing, that will form the desired result(s). In the tests conduced with the input of the example, four interpretations where produced, with only the correct one of a resolvable type (**t**):

```
((And^{(t->(t->t))} (heavy^{(P->t)} (morph_R->Phy^{(R->P)}
                                                  book^{R})))
    (interesting^{(I->t)} (morph_R->I^{(R->I)} book^{R})))
```

2.4 Implicit Adaptation

Polymorphic operators such as *and*, with explicit adaptation slots, are needed for co-predications. However, most lexical adaptations can take place implicitly, simply by reacting to a type mismatch such as $(p^{A\to B}\ a^{C})$ and applying any suitable transformation to resolve the type mismatch. In order to do this automatically, there are two possibilities to resolve such type mismatches: by adapting the predicate, yielding $((f^{(A\to B)\to(C\to B)})\ p)\ a)$, or the argument, resulting in $(p\ (f^{C\to A}\ a))$. There is also a third situation to consider, that of a partial application $(\lambda x^{A}.\tau\ a^{C})$, in which the argument can be adapted as above, but the typing of the predicate might be as not be determined at the moment of the adaptation.

A procedure analyses such applications with type mismatches and no explicit adaptation slots, and inserts suitable, automatically generated adaptation slots, then proceeds as with explicit adaptations. For example, a simple term application such as `(P^{(e->t)} a^{A})`, with a transformation `f_{A->e}` available to the atomic term `a` yields the straightforward (and only felicitous) interpretation `(lambda x^A.(P^{(e->t)} (f_{A->e}^{(A->e)} x^A)) a^{A})`, that reduces to `(P^{(e->t)} (f_{A->e}^{(A->e)} a^{A})`.

Implicit adaptations are necessarily reduced to those simple cases. Trying to account automatically for co-predications would imply to try any possible permutation of types and transformations at all nodes of a term, which would be exponential in complexity; thus, the need for explicit operators such as the polymorphic *and*.

2.5 Lexicalisation

In addition to the core mechanisms, a `tecto` package provides support for a tectogrammatical/syntactic structure in the form of an unannotated binary tree of lexemes ; this serves as a factory for the input of already analysed text, and as a more streamlined form of output for adapted terms.

A `lexicon` package enables the storage of lexical entries that associate lexemes (as strings) to terms, complete with typing, transformations and ambiguities. Lexica can be merged, in order to have combine the treatment of different phenomena, treated as standalone modules, for complex sentences. Lexica also provide automated translations from a syntactic structure (a `tecto` term) to a semantic one (a `TermW` term, initially not adapted, reduced or even type-checked). Semantic terms can be presented either by a straightforward translation to syntactic terms, or printed to a string in the usual fully-parenthesed prefix notation with apparent typing (as in the examples of this article).

2.6 Phenomena Coverage

Many lexical phenomena discussed in [22,13] can be modelled using the simple mechanisms of ΛTY_n in their prototypal implementation given above; some others require additional mechanisms.

Lexical adaptations, including alternations, meaning transfers, grinding, qualia-exploitation and "Dot-type"-exploitation are all supported by the adaptation mechanisms, as given previously. *Simple predications* only require to have suitable transformations available, and to use the implicit adaptation mechanisms ; *co-predications* require explicit adaptation using polymorphic operators. Theoretical grounds have been laid in [2,24].

Constraints of application are required in order to perform co-predications correctly. As explained in [17], the simultaneous reference to different facets of a same entity can be infelicitous in some circumstances, such as the use of destructive transformations (grinding, packing) or metaphorical use of some words. Thus, the following co-predications are infelicitous to some degree : **The salmon was lighting-fast and delicious*, *? Birmingham won the championship and was split in the Brexit vote*. In order to block such co-predications, we have proposed to place constraints on transformations in order to block their usage depending on the other transformations that have been used on the same term. The first version of this system given in, e. g., [2], distinguishes between *flexible* (allowing all other facets) and *rigid* (blocking all other facets) transformations. The latest version, given in [14], proposes a $\Lambda^{-\circ}TY_n$, a system with terms of the linear intuitionistic logic as types, that (among other things) allow any arbitrary type-driven predicate to act as a constraint on the use of transformations.

In this prototype implementation, all transformations come with a member function that can be defined as a constraint, and a compatibility check of all transformations can be performed using every constraint, the default constraint being the boolean constant `true` (that simply models flexible transformations). As the constraint can effectively be any function, the precision is the same as in [14].

Ontological inclusion, called *type accommodation* in [22] and modelling the lexical relation of hyponymy, can be supported by tweaking the system of sorts. The theoretical and empirical basis for doing so are discussed in [18], in which we argue that *coercive sub-typing* is an accurate and helpful mechanism for resolving ontological inclusion, but no other lexical phenomena.

In order to support sub-typing, each sort can be defined with an optional `parent` sort. A careful review of the typing comparison mechanism will then be enough, together with a rewriting of the equality method for sorts, in order to support sub-typing. This is not implemented yet, but does not require (much) additional processing power.

Performative lexical adaptations, such as quantification, Hilbert operators for determiners, and the alternate readings of plurals and mass nouns, are supported as far as the meaning assembly phase is concerned. However, in order to be useful, this category of lexical phenomena (as well as hypostasis and several others) require additional mechanisms in order to incorporate the knowledge gathered from the analysis of the sentence into the logical representation. The basic architecture is supported, but mechanisms of resolution remain preliminary and will be discussed next, especially in Section 3.3.

3 Layers of Lexica and Knowledge Representation

3.1 The Additional Layers

Theories of semantics deriving from [22] generally encompass some degree of common sense world knowledge: it is considered known that a *committee* (and other such group nouns) is made of several people and is a felicitous argument of predicates requiring a plural argument, and that *engine*s are part of *car*s and thus that predicates such as *powerful* or *fuel-guzzling* can apply to *car*s via their constitutive quale. It has been argued (e. g. in [9]) that such complex knowledge does not belong in a semantic lexicon; we will paraphrase Im and Lee from [10], defining semantics to be the meaning conveyed by an utterance to a competent speaker of the language in itself, excluding, for instance, the specific situation in which the utterance is made, but including any previous discourse. Thus, the full contents of a given fairy tales should be able to be described within semantics, while a political essay will probably require additional knowledge about the position of the author and the specifics of the period of writing.

From our point of view, designing a complete tool for type-theoretic lexical semantics imply the careful definition of various lexica that can convey the necessary, elementary world-knowledge for each word. A lexicon for general use will associate to all relevant lexemes their semantics (in the form of main and optional λ-terms) as can be given in a dictionary of a language. However, there are two common cases in which the general lexicon is not sufficient.

First are the specific lexica: vocabularies relevant only to a community (professional jargons, local dialects, and other linguistic constructs specific to small groups of people), and/or to a specific literary universe (fairy tales, space opera, mythology, politic speeches, etc.). Such lexica are activated on an as-needed basis, switched often, and are more specific than the general-use lexicon.

Lexical semantics also requires a lexicon used for the current enunciation. A competent speaker of any language is able to use generative mechanisms in order to introduce new lexical concepts, either by hypostasis (the use of a new word, the meaning of which can be inferred from context and morphology), or by creative use (giving a new, contextually evident meaning to an existing word).

In our view, the lexicon of the enunciation starts empty and can be augmented when the analysis of the discourse encounters words that are not present in the current active lexica. We think that such mechanisms can enable the *learning* of lexical semantic data. In addition to these lexical layers of meaning, we tend to implement different lexical phenomena using different lexica for simplicity's sake, and create a merged lexicon from every relevant one when processing text.

3.2 Individuals, Facts and Contexts

To summarise our argument in Section 3.1 above, in addition to mostly static lexical data, some sort of knowledge representation is needed to process even simple lexical phenomena such as collective and distributive readings for plurals. Namely, we need to keep track of the *individuals* mentioned in a given discourse, and of the *facts* asserted of those individuals. To be complete, we would also need to keep track of *agents*, in order to model dialogues or multiple points of view in which certain agents assert certain facts. Our implementation prototype currently supports individuals, as atomic terms of type A (for named entities: human agents, towns...) or $A \rightarrow \mathbf{t}$ (for common nouns, that can be resolved to a specific individual of type A by the means of an Hilbert-based determiner) for any sort A. We also account for facts, as predicates (`TermBindings` or atomic terms) of type $\alpha \rightarrow \mathbf{t}$ for any arbitrarily complex type α, that are used in a term application, and apply to an individual. In the analysis of a term, individuals and types are extracted and added to the context of enunciation. The hierarchy of lexical layers given above can be implemented as a hierarchy of *contexts*, some containing initial individuals and facts relevant to each lexicon; in a such complete system, the context of the real world would, to resolve the paradox mentioned in [25], include the fact that there is no King of France (and therefore that *The king of France is bald*, while grammatical, is not felicitous because there are no qualifying referents for the entities described, and thus cannot be assigned a truth value). Such contexts are specific objects (aggregating individuals, facts and a related lexicon) in our implementation.

3.3 The Parsing-Knowledge Loop

We use a specific lexicon to list some common semantic terms for quantifiers, counting terms, logical and Hilbert operators (detailed in e. g. [23], more recently in [16]). Other lexica can make use of these terms in order to construct, for instance, Link-based semantics for plurals (originally given in [11]), using lexical transformations as suggested by [20] and detailed in [15]. Some functions associated to the logical lexicon then resolve the operators, given a term and a context. This updated process of analysis is given in Fig. 3.

To explain what the analysis of plural readings in MGL entail, consider the following example from [15] : *Jimi and Dusty met* is analysed as $|\lambda y^{e}.(y = j) \vee (y = d)| > 1 \wedge meet(\lambda y^{e}.(y = j) \vee (y = d))$. One elementary issue is that the predicate *met* applies to group individuals (such as *a committee*) and constructions made of more than one individuals (such as *Jimi and Dusty*) but not to singular individuals (such as *a student*). Thus, the lexical entry for the predicate is $\lambda P^{e \rightarrow t}.|P| > 1 \wedge meet(P)$ – a logical conjunction with a cardinality operator.

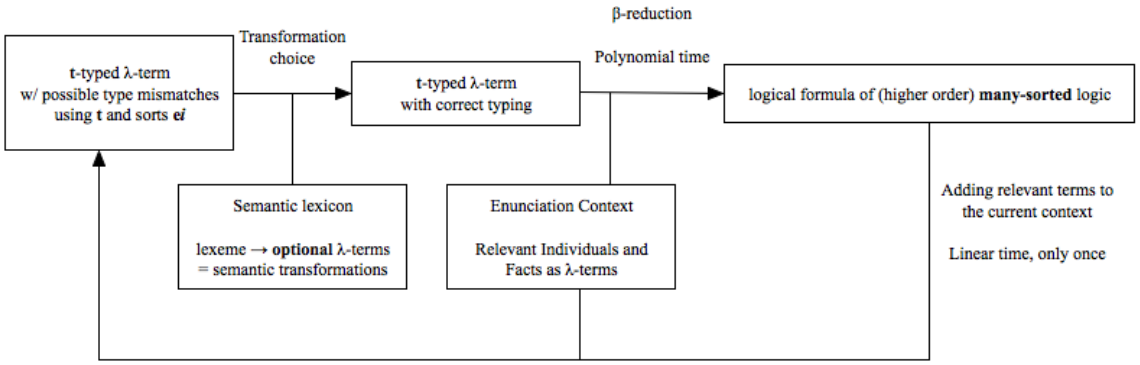

Fig. 3. Parsing-Knowledge Representation Feedback

Those two simple elements can be defined in System-F (the calculus in which ΛTY_n, the logic of MGL, is implemented). The issue is that, in order for our system to infer correctly that Jimi and Dusty are two different individuals, and thus that the above term resolves to $meet(\lambda y^e.(y = j) \vee (y = d))$, we must use processing power beyond the simple construction and reduction of terms: a minimal system of knowledge representation and logical inference. Within our architecture encompassing individuals and facts, and with a functional lexicon for logical connectives (including the logical *and* operator of that example), as well as quantification and counting (including the cardinality operator), this example can be treated.

However, this requires a given term to be parsed at least twice: the first time, the syntactic structure is converted into a semantic term and lexical transformations are applied, the second, facts that emerge from the transformations are added to the lexicon, and the logical lexicon can be used in order to process the operators that have been introduced. Our prototype implementation does not incorporate such feedback yet, as the first step can result in several different interpretations; this remains a work in progress. As a result, straightforward composition for plurals are tentatively supported (such as in the previous example), but ambiguous covering readings for plurals are not yet available.

3.4 Hypostasis and Quantificational Puzzles

An enunciation-context lexicon that is filled with individuals and facts inferred from the primary semantic analysis can serve, in a limited way, to account for *hypostasis*. Words absent from the lexicon, but syntactically placed in the position occupied by individuals, will be added as primary entities to the lexicon, and their precise typing inferred from the predicates they are applied to. An elementary mechanism should be enough to have a correct representation from Lewis Carroll's *Jabberwocky*. Of course, most competent human speakers also use morphosyntactic inference to attach at least some degree of connotative meaning to the words being proposed (e. g., *Star Wars*'s *plasteel* can be inferred as a fictive material somehow combining the characteristics of plastics and steel by any English speaker).

This is completely beyond the power of our early software. Rather, we can have the process of meaning assembly outline which lexemes are not in the lexicon, and use human input for correcting the precise types and terms associated.

The process of counting, quantifying and selecting entities using Hilbert operators can also shed some light on the quantificational puzzles mentioned in [1] and several other related works. The issue with having universal quantification used together with co-predication on multi-faceted entities can be seen in examples such as *There are five copies of* War and Peace *and a copy of an anthology of Tolstoï's complete works on the shelf* (what is the answer to questions such as *How many books…?*, and what exactly is the type of *book* in such questions ?), or *I read, then burnt, every book in the attic* (the entities being predicated form two different sets). In order to resolve such quantificational puzzles satisfactorily, the methods for counting and quantifying must be adapted to each predicate, and only apply to individuals of the appropriate type. For our purpose, this implies a close monitoring of the entities introduced by lexical transformations and their context of appearance. This is also a work in progress.

4 Results

4.1 A Fragment of Second-Order

We have proven that MGL can actually be computationally implemented. This was not really in doubt, but the way that the combination of types and terms are implemented illustrates that the time and space complexity of most of the process is limited: the algorithms used are mostly linear tree walks, with a few quadratic worst-case operations. The most complex step is the choice of optional terms for adaptation slots, of complexity $|t| \times |s| \times n$ at worst (the product of the number of optional terms available, adaptation slots, and length of the term); the hypothesis behind MGL is that the number of available optional terms at any point remains manageable. Thus, the step not actually implemented in this prototype (but for which many implementations exist), syntactic analysis, is the costliest of the process detailed in Fig. 1 and the complete process of parsing is polynomial in time.

MGL accounts such as [24] point out that the whole expressive power of second-order λ-calculus is not used, and that all could be implemented using first-order terms if all possible adaptations were listed at each step (which is syntactically much longer to write). Indeed, our implementation only supports the single second-order operation of type specialisation (by distinguishing type variables from other types and using pattern matching to recognise and rewrite types), which is required for having polymorphic terms. There are no features of ΛTY_n that require additional power: sub-typing can be implemented by an optional `parent` field in `Sorts`, arbitrary complex on co-predications are supported by including a check on transformations that can be any arbitrary function, quantification, counting and Hilbert operators can be included…

4.2 Minimal Processing Architecture

Our prototype implementation includes the skeleton of an architecture that represents the individuals, facts and agents appearing during the semantic analysis.

This goes beyond the straightforward process of producing a logical representation for an utterance, as some of the terms of that logical representation might be analysed differently depending on the context; we argue that that process is still part of a semantic analysis. The individuals, facts and agents are stored in objects called *contexts*, organised in a hierarchy that includes the most specific context (modelling the analysis of the current discourse), universe-specific contexts (describing whether the discourse is part of a fictional, historical or activity-specific setting), dialect- and language-specific contexts, each associated to an appropriate lexicon. A complete analysis would minimally involve the construction of the logical representation of an utterance, the update of the enunciation context with individuals and facts introduced by that utterance, and a re-interpretation of the logical representation in the active contexts. This minimal processing architecture can be completed with no difficulties; our implementation includes relevant data structures and algorithms, but requires significant work on examples of performative lexica in order to be thoroughly tested.

4.3 Perspectives

This prototype implementation has already served its primary purpose: to illustrate that MGL can be computationally implemented, and that the examples usually given with the theory actually work. As it is, however, this implementation is more of a proof of concept than useful software.

To be actively used by the community, more work would be required to give it an helpful interface, both for the user and for existing analysers; we also would like to convert from and to representations of the other most active type-theoretic accounts of lexical semantics. The knowledge-representation architecture remains a work in progress, and requires solid efforts in order to correspond to our ambitions. However, what MGL really requires in order to be useful is a large-cover library of types and terms; our hope is that this prototype will help to build software that can learn those features from corpora.

References

1. Nicholas Asher. *Lexical Meaning in Context: A Web of Words*. Cambridge University Press, March 2011.
2. Christian Bassac, Bruno Mery, and Christian Retoré. Towards a Type-Theoretical Account of Lexical Semantics. *Journal of Language, Logic, and Information*, 19(2), 2010.
3. Daisuke Bekki. Dependent Type Semantics: An Introduction. In Zoé Christoff, Paolo Galeazzi, Nina Gierasimczuk, Alexandru Marcoci, and Sonja Smet, editors, *Logic and Interactive RAtionality (LIRa) Yearbook 2012*, volume I, pages 277–300. University of Amsterdam, 2014.
4. M. Bierwisch. Wördliche Bedeutung - eine pragmatische Gretchenfrag. In G. Grewendorf, editor, *Sprechakttheorie und Semantik*, pages 119–148. Surkamp, Frankfurt, 1979.
5. Peter Bosch. The bermuda triangle: Natural language semantics between linguistics, knowledge representation, and knowledge processing. In *Text Understanding in LILOG, Integrating Computational Linguistics and Artificial Intelligence, Final Report on the IBM Germany LILOG-Project*, pages 243–258, London, UK, UK, 1991. Springer-Verlag.

6. Stergios Chatzikyriakidis and Zhaohui Luo. Natural language inference in coq. *J. of Logic, Lang. and Inf.*, 23(4):441–480, December 2014.
7. Robin Cooper. Copredication, dynamic generalized quantification and lexical innovation by coercion. In *Fourth International Workshop on Generative Approaches to the Lexicon*, 2007.
8. D. A. Cruse. *Lexical Semantics*. Cambridge, New York, 1986.
9. J. A. Fodor and E. Lepore. The emptiness of the lexicon : Reflections on James Pustejovsky's *The Generative Lexicon*. *Liguistic Inquiry*, 29(2), 1998.
10. Seohyun Im and Chunngmin Lee. A developed analysis of type coercion based on type theory and conventionality. In Robin Cooper and Christian Retoré, editors, *ESSLLI proceedings of the TYTLES workshop on TYpe Theory and LExical Semantics*, Barcelona, August 2015.
11. Godehard Link. The logical analysis of plurals and mass terms: A lattice-theoretic approach. In P. Portner and B. H. Partee, editors, *Formal Semantics - the Essential Readings*, pages 127–147. Blackwell, 1983.
12. Zhaohui Luo. Contextual analysis of word meanings in type-theoretical semantics. In *Pogodalla and Prost*, pages 159–174. 2011.
13. Bruno Mery. *Modélisation de la Sémantique Lexicale dans le cadre de la Théorie des Types*. PhD thesis, Université de Bordeaux, July 2011.
14. Bruno Mery. Lexical Semantics with Linear Types. In *NLCS '15, the Third Workshop on Natural Language and Computer Science*, Kyoto, Japan, July 2015.
15. Bruno Mery, Richard Moot, and Christian Retoré. Computing the Semantics of Plurals and Massive Entities using Many-Sorted Types. In Tsuyoshi Murata, Koji Mineshima, and Daisuke Bekki, editors, *New Frontiers in Artificial Intelligence JSAI-isAI 2014 Workshops, LENLS, JURISIN, and GABA, Kanagawa, Japan, October 27-28, 2014, Revised Selected Papers*, volume 9067 of *Lecture Notes in Artificial Intelligence*, page 357. Springer-Verlag Berlin Heidelberg, 2015.
16. Bruno Mery, Richard Moot, and Christian Retoré. Typed Hilbert Operators for the Lexical Semantics of Singular and Plural Determiner Phrases. In *Epsilon 2015 – Hilbert's Epsilon and Tau in Logic, Informatics and Linguistics*, Montpellier, France, June 2015.
17. Bruno Mery and Christian Retoré. Recent advances in the logical representation of lexical semantics. In *NCLS – Workshop on Natural Language and Computer Science, LiCS 2013*, Tulane University, New Orleans, June 2013.
18. Bruno Mery and Christian Retoré. Are books events ? Ontological Inclusions as Coercive Sub-Typing, Lexical Transfers as Entailment. In *LENLS '12, in jSAI 2015*, Kanagawa, Japan, November 2015.
19. Richard Moot. Wide-coverage French syntax and semantics using Grail. In *Proceedings of Traitement Automatique des Langues Naturelles (TALN)*, Montreal, 2010.
20. Richard Moot and Christian Retoré. Second order lambda calculus for meaning assembly: on the logical syntax of plurals. In *Coconat*, Tilburg, Netherlands, 2011.
21. Reinhard Muskens. Meaning and Partiality. In Robin Cooper and Maarten de Rijke, editors, *Studies in Logic, Langage and Information*. CSLI, 1996.
22. James Pustejovsky. *The Generative Lexicon*. MIT Press, 1995.
23. Christian Retoré. Sémantique des déterminants dans un cadre richement typé. *CoRR*, abs/1302.1422, 2013.
24. Christian Retoré. The Montagovian Generative Lexicon Lambda Ty_n: a Type Theoretical Framework for Natural Language Semantics. In Ralph Matthes and Aleksy Schubert, editors, *19th International Conference on Types for Proofs and Programs (TYPES 2013)*, volume 26 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 202–229, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
25. Bertrand Russell. On denoting. *Mind*, 14(56):479–493, 1905.