Challenges in the Computational Implementation of Montagovian Lexical Semantics

Bruno Mery(✉)

LaBRI, Université de Bordeaux, Bordeaux, France
bruno.mery@u-bordeaux.fr

Abstract. We present and discuss 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 constitutes a proof of concept of the MGL theory. The implementation process, as well as 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 require additional mechanisms in order to process the logical representation in order to take into account implicit common-sense world knowledge. We therefore also present a minimal architecture for knowledge representation, and how it can be applied to different phenomena. The implementation illustrates the validity of the theory, but MGL requires a stronger corpus of types and terms in order to be thoroughly tested.

Keywords: Lexical semantics · Montagovian Generative Lexicon · Knowledge representation for natural language semantics · Typed lambda calculus · Prototype software

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 [5,10,27], then on the Generative Lexicon theory thoroughly developed in [29], 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 [9], Unified Type Theory (UTT) presented in [16], and the

Many thanks to the organisers, reviewers, speakers and participants of the LENLS 13 Workshop, in the proceedings of which the first version of this publication appeared as [20]. Thanks also to the director and adjunct directors of the LaBRI for supporting this research.

© Springer International Publishing AG 2017
S. Kurahashi et al. (Eds.): JSAI-isAI 2016, LNAI 10247, pp. 90–107, 2017.
DOI: 10.1007/978-3-319-61572-1_7

framework we helped define, the Montagovian Generative Lexicon (MGL), presented in [32]. Several partial or complete implementations of those theories have been provided for demonstration purposes. Those tend to use logical or functional programming, or theorem provers such as Coq—[8] is an example.

Concerning MGL, however, no real demonstration of the computational aspect has ever been provided. One of the stated goals of MGL was (paraphrasing slightly [32]) "to provide an integrated treatment from syntax to semantics extending existing analysers based on Montagovian semantics such as [25] with mechanisms for lexical semantics that *are easily implemented in a typed functional programming language like Haskell*". Until the present publication, however, such implementations have been purely hypothetical, except for domain-specific analogues such as evoked in [26]. Our goal in this publication is to present an actual prototype implementation of the lexical semantics of that framework. For that purpose, we have used functional and object programming in Scala, but the paradigm and language of programmation are not critical elements. We also want to analyse what MGL can do using the (quite simple) computational mechanisms involved, as well as how those should be supplemented in order to provide useful treatments of semantics and pragmatics.

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 (including and not limited to MGL), and the deep interaction of lexical meaning with at least some sort of knowledge representation already evoked in [6].

2 A MGL Prototype

2.1 The Montagovian Generative Lexicon

MGL is a type-theoretic account of compositional lexical semantics that uses a calculus of semantic assembly called ΛTY_n, an adaptation of the many-sorted logic TY_n (itself proposed in [28]) for the second-order λ-calculus, given in the syntax of Girard's System-F. MGL stays close to the usual Montague analysis by first performing a syntax-based analysis via proof-search, followed by the substitution of semantic main λ-terms to syntactic categories. Afterwards, lexical mechanisms are 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, sketched in Fig. 1, are detailed in [24].

They can be roughly summarised as follows:

– First, the input utterance is super-tagged and analysed using categorial-grammar mechanisms. This is the only step of proof-search of the process, and yields a syntactic term whose components are syntactic categories. The lexicon is then used in standard Montagovian fashion to substitute semantic

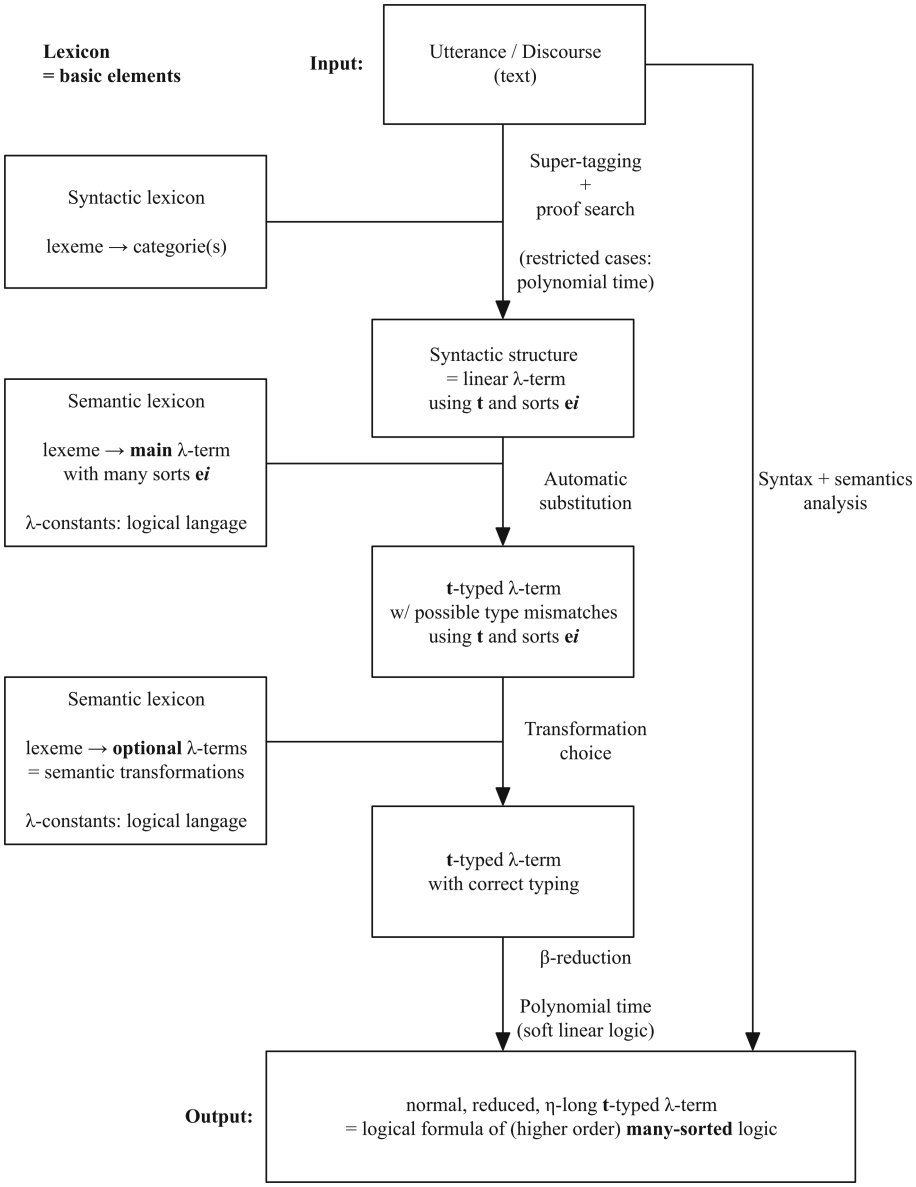

Fig. 1. MGL process summary

λ-terms to categories, yielding a main semantic term, typed with many sorts and the type **t** for propositions.

– Second, as a many-sorted logic is used, some type mismatches might occur. The direct correspondance between syntax and semantics is not guaranteed by the syntax as the arguments will be present in the correct number and

position, but will not necessary be of the sort expected by the predicate. This allows the mechanisms of lexical semantics to engage and disambiguate between terms.
To that effect, 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 **t** (the type of propositions), i.e. a logical formula that can be used in any usual semantic theories of meaning, including as model-theoretic and game-theoretic semantics.

As the first step, syntax-based analysis, is already well-studied and implemented (we use Type-Logical Grammars and Grail for this step, as given in [25]), the object of concern is the second step: given a term reflecting the syntactic structure of an utterance and a semantic lexicon, 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 term and type categories; this simple categorisation allows us to easily construct and detect patterns of objects.

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

– For terms:
  • Leaves are `AtomicTerm`s (constants), `TermVarId`s (variables) with an identifier and type, or specific constructs for MGL, `Transformation`s and `Slot`s.
  • Inner nodes are `TermBinding`s (λ-abstracted terms), or `TermApplication`s of a predicate term to an argument term.
– For types:
  • Leaves are constant `Sort`s, pre-defined objects such as `PropositionType` for **t**, or second-order variable identifiers `TypeVarId`s.
  • Inner nodes are `TypeFunction`s between two types A and B (modelling $A \rightarrow B$), or `TypeApplication`s (modelling $A\ \{B\}$).

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

Several algorithms are provided as specialised methods and class constructors 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 resources are incrementally built as terms and types are constructed in order to minimise computations). Algorithms include the type-checking of

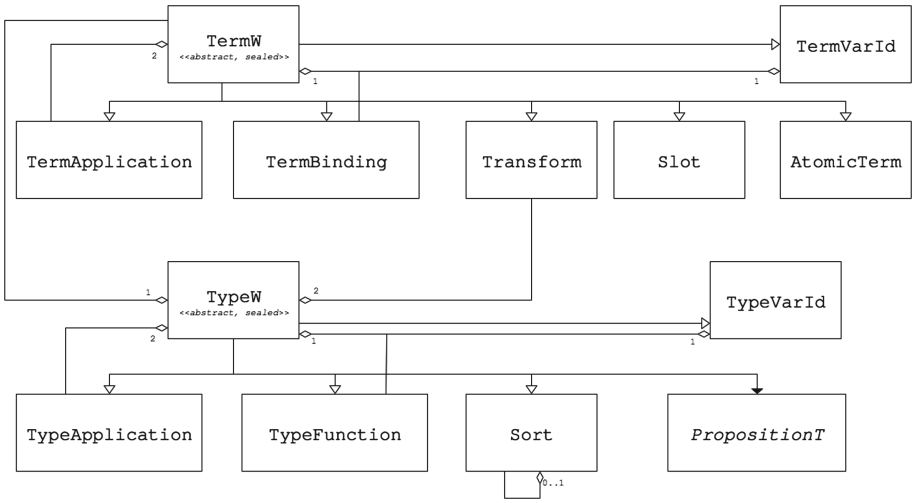

Fig. 2. Class diagram of the core package for terms and types.

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). In this prototype implementation, some degree of imperative programming is used in order to make some and optimisations easier.

It would be easy to convert the entire program to purely functional (or logical) programming. Most methods 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 principle 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*. The usual, formal MGL analysis supposes 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$^{\varphi \to \mathbf{t}}$, *interesting* as interesting$^{I \to \mathbf{t}}$. 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: $f_{phys}^{R \to \varphi}$ and $f_{info}^{R \to I}$.

In order to apply the co-predication, a specific operator, the polymorphic conjunction *and*, is required:

$$\&^{\Pi} = \Lambda\alpha\Lambda\beta\lambda P^{\alpha\to\mathbf{t}}\lambda Q^{\beta\to\mathbf{t}}\Lambda\xi\lambda x^{\xi}\lambda f^{\xi\to\alpha}\lambda g^{\xi\to\beta}.(\mathrm{and}^{\mathbf{t}\to\mathbf{t}\to\mathbf{t}}\ (P\ (f\ x))(Q\ (g\ x)))$$

This yields, after suitable substitutions, application, and reduction, the term (and (heavy (f_{phys} book)) (interesting (f_{info} book))), which is normal and of type $\mathbf{t}$.

In our implementation, there are several important differences with the formal account outlined above. As we distinguish between type constants and variables, there is no actual need to explicitly abstract (and Λ-bind) 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 conjunction above) and *adaptation slots*, the positions 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. Adding or removing adaptation slots is in fact a technical process analogue to type-raising or type-lowering in Montague semantics.

We provide optional terms (transformations) as `Transform`s, which are distinguished from other terms. Each term has a list of available transformations, constructed recursively from the leaves. The lexicon provides a list of transformations available to each lexeme, so that each atomic term has a collection of transformations. We also distinguish `Slot`s for explicit adaptations; the list of slots is maintained during the construction of the terms. Our polymorphic *and* conjunction, also defined by the lexicon, 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)))
```

(As above, `f` and `g` are adaptation slots, distinguished from other variables and not bound, while `P` and `Q` are λ-bound predicate variables.)

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 (one for every slot/transformation permutation), 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 and applying any suitable transformation to resolve it. This requires to adapt the terms by adding correctly-typed and well-positioned adaptation slots. In the case of an application such as $(p^{A\to B}\ a^{C})$, there are two possibilities to resolve the type mismatch: by adapting the predicate, yielding $((f^{(A\to B)\to(C\to B)})\ p)\ a)$, or the argument, resulting in $(p\ (f^{C\to A}\ a))$. In the slightly different case of a partial application $(\lambda x^{A}.\tau\ a^{C})$, in which the argument can be adapted as before (yielding $(p\ (f^{C\to A}\ a))$), but the typing of the predicate might not be fully 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 lexical operators with explicit adaptation slots 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 [18,29] can be modelled using the simple mechanisms of ΛTY_n in their prototypal implementation given above; some others require additional mechanisms. The following is a short overview of how some classical phenomena are handled in MGL and our prototype.

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 (such as the higher-order conjunction "and" above). Theoretical grounds have been laid in [18,32].

Constraints of application are required in order to perform co-predications correctly. As explained in [24], 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., [18], distinguishes between *flexible* (allowing all other facets) and *rigid* (blocking all other facets) transformations, as well as relaxable constraints depending on syntactic features. The latest version, given in [19], proposes a revised calculus named $\Lambda^{-\circ}TY_n$, 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, thus allowing the complex variability of felicity of co-predications with deverbals examined by [13] and detailed for MGL in [30].

In this prototype implementation, all transformations are equipped with a member function that can be defined as an arbitrary constraint, the default being the boolean constant `true` (that simply models flexible transformations). A compatibility one-on-one check of all transformations can be performed using every constraint. As the constraint can effectively be any function, the precision is the same as in [19].

Ontological inclusion, called *type accommodation* in [29] and modelling the lexical relation of *hyponymy* (as evoked with different solutions in other sub-typing accounts, such as [4]), can be supported by tweaking the system of sorts. The theoretical and empirical basis for doing so are discussed in [23], in which we argue that *coercive sub-typing* is an accurate and helpful mechanism for resolving ontological inclusion, but no other lexical phenomena; the latter are implemented using word-level transformations. 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 in order to support sub-typing.

This is not implemented yet (it requires a refactoring of the notion of equality for sorts), 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 Sect. 3.3.

3 Layers of Lexica and Knowledge Representation

3.1 The Additional Layers

Theories of semantics deriving from [29] generally encompass some degree of common sense world knowledge. For example, it is considered known that a *committee* (and other such group nouns) is made of several people, and is thus a felicitous argument of predicates requiring a plural argument such as *to meet*. It is also known that *engine*s are part of *car*s, and that predicates such as *powerful* or *fuel-guzzling* can apply to *car*s via their constitutive quale; all such "common-sense metaphysics" have been part of generative lexical theory from the start, as detailed by [2]. It has been argued (e.g. in [11]) that such complex knowledge does not belong in a semantic lexicon; we will ignore such claims, paraphrasing Im and Lee from [12] in 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. In this view the full contents of, for example, a given fairy tales, should be able to be described within semantics, while texts such as political essays will probably require additional knowledge about the position of the author and the specifics of the period of writing. From our point of view, the lexicon includes that minimal knowledge as part of the semantic terms and types involved. To design a complete tool for type-theoretic lexical semantics, we must first complete 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 arguments to be made for additional lexica beyond the general language lexicon.

First are the specific lexica, that detail vocabularies relevant only to a community, such as professional jargons, local dialects, and other linguistic constructs specific to small groups of people; and/or words or word uses restrained to a specific literary universe, such as fairy tales, space opera, mythology, politic speeches, etc. Such lexica are activated on an as-needed basis, 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 the use of a new word, the meaning of which can be inferred from context and morphology, or by creative use of 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 Sect. 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 supports individuals, as atomic terms of type A (for named entities: human agents, towns...), as well as some individuals as atomic terms of type $A \rightarrow \mathbf{t}$ (for common nouns, that can be resolved to a specific individual of type A by the means of an Hilbert-based determiner) for any suitable sort A. This dual typing for individual is adopted by MGL for complete compatibility with the classical Montagovian analysis, and is well-suited for the treatment of many phenomena, using operators inspired by Hilbert for the representation of determiners (as given in [22]) in order to select a specific individual from a common noun. We think that this approach is justified; however, this is not the chosen modelisation for other type-theoretic accounts of lexical semantics, and the implementation provided is easily adapted to other types for common nouns and individuals. See [24] for a detailed discussion of our choice of types and [17] for a contrasting opinion.

We also account for *facts*, as predicates (`TermBinding`s 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 [33], 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 (universal/existential, needed for some classical interpretations), counting terms (needed for plurals), logical constants (for truth-valued semantic interpretations of grammatical constraints) and Hilbert operators (used for the smooth modelling of determiners, as detailed in e.g. [31] and more recently in [22]).

Other lexica can make use of these terms in order to construct, for instance, Link-based semantics for plurals (originally given in [15]), using lexical transformations as detailed in [21]. Some functions associated to the logical lexicon then resolve the glue 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 [21]:

– *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.

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.

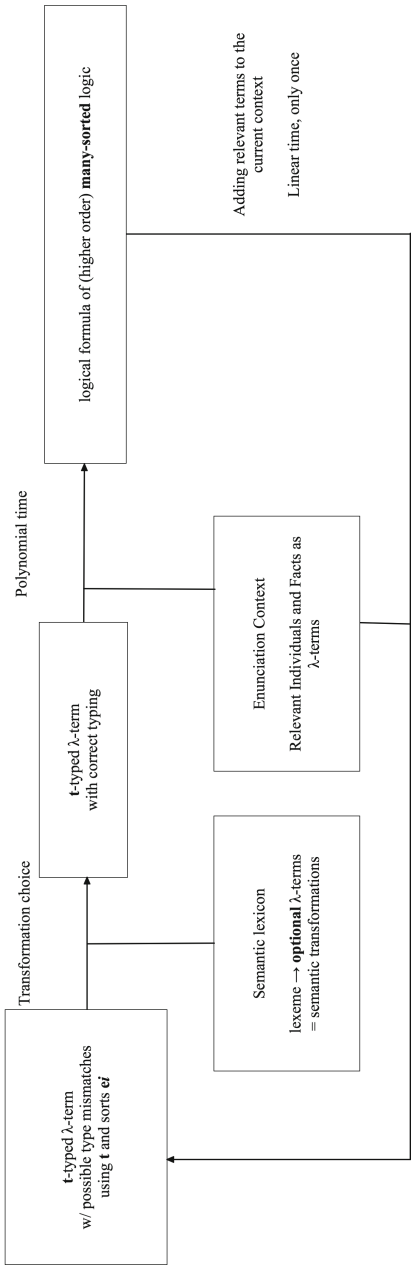

Fig. 3. Parsing-knowledge representation feedback

The second time, facts that emerge from the transformations, such as alternate distributive/collective readings for plural predicates, 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 World Sense Acquisition

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 words that are absent from the lexicon. Such words, syntactically placed in the position occupied by individuals or facts, will be added as primary entities to the lexicon, and their precise *typing* inferred from the predicates they are applied to. The typing can then be refined as the new lexeme is used again. An elementary mechanism should be enough to have a correct (if completely underspecified) 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, and simple syntax-and-typing inferences.

A first easy step is to 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. Promising automated strategies for learning more about new lexemes, or lexemes used in a creative way, are also explored in richly typed settings by [7].

3.5 Quantificational Puzzles

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...*?
  What exactly is the type of *book* in such questions?
- *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.

As the quantifiers are adapted to the transformed and original entities, maintaining expected truth values for straightforward sentences poses a challenge. This is also a work in progress.

4 Results

4.1 A Fragment of Second-Order

With this prototype software, 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 steps detailed in Fig. 1, and the complete process of parsing is polynomial in time, as explained in [24].

MGL accounts such as [32] point out that the whole expressive power of second-order λ-calculus is not used, and that every possible operation 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.

Formally, it has been pointed out by Ph. de Groote (pc.) that the smallest type calculus encompassing the features provided by ΛTY_n is a simply-typed calculus supporting type collections and sub-typing, such as the typing system used by OCaML (http://ocaml.org/). Moreover, the many efforts made by proponents of other related type-theoretic accounts of generative lexical semantics to provide implementations of their own theories reinforce our belief in the computational feasibility of such analyses. The chain of treatment sketched in Fig. 1 and detailed in [24], and the computational steps provided in the present paper, are very similar to, for instance, the process detailed in [14] for Dependent

Type Semantics. The many differences between the theories and representations notwithstanding, the operational steps used to analyse similar phenomena are nearly identical (despite the research process on both accounts being parallel and candid, and the presentation of the results having occurred on the same day). Such spontaneous convergence between theories is, in our view, an indication of the pertinence of both accounts.

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 this process should still be 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 stands, 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. The completion of this software is not, however, an end in itself.

In fact, what MGL (and other related accounts of lexical semantics) really requires in order to be useful is a large-cover library of types and terms. The analyses, whether formal or computational, are justified by toy linguistic examples or on domain-restricted phenomena; without a significant step in the definition of a system of sorts and types, and subsequently of a large, semantically

rich lexicon, type-theoretic proposals have very little value compared to, e.g., "deep" neural techniques trained using massive corpora. The prototype software presented in this publication is a nice illustration of the possibilities provided by MGL, but our main hope is that it can help to build software that can learn types and lexemes.

The first step will not be on the software side but, as suggested in [24], the establishment of a linguistically motivated kernel system of sorts.

References

1. Asher, N.: Lexical Meaning in Context: A Web of Words. Cambridge University Press, Cambridge (2011)
2. Asher, N., Pustejovsky, J.: Word meaning and commonsense metaphysics. Semantics Archive, August 2005
3. Bekki, D.: Dependent type semantics: an introduction. In: Christoff, Z., Galeazzi, P., Gierasimczuk, N., Marcoci, A., Smet, S. (eds.) Logic and Interactive RAtionality (LIRa) Yearbook 2012, vol. 1, pp. 277–300. University of Amsterdam, Amsterdam (2014)
4. Bekki, D., Asher, N.: Logical polysemy and subtyping. In: Motomura, Y., Butler, A., Bekki, D. (eds.) JSAI-isAI 2012. LNCS, vol. 7856, pp. 17–24. Springer, Heidelberg (2013). doi:10.1007/978-3-642-39931-2_2
5. Bierwisch, M.: Wördliche Bedeutung - eine pragmatische Gretchenfrag. In: Grewendorf, G. (ed.) Sprechakttheorie und Semantik, pp. 119–148. Surkamp, Frankfurt (1979)
6. Bosch, P.: The bermuda triangle: natural language semantics between linguistics, knowledge representation, and knowledge processing. In: Herzog, O., Rollinger, C.-R. (eds.) Text Understanding in LILOG. LNCS, vol. 546, pp. 241–258. Springer, Heidelberg (1991). doi:10.1007/3-540-54594-8_64. http://dl.acm.org/citation.cfm?id=647415.725191
7. Breitholtz, E.: Are widows always wicked? Learning concepts through enthymematic reasoning. In: Cooper, R., Retoré, C. (eds.) ESSLLI Proceedings of the TYTLES Workshop on TYpe Theory and LExical Semantics, Barcelona, August 2015
8. Chatzikyriakidis, S., Luo, Z.: Natural language inference in Coq. J. Log. Lang. Inf. **23**(4), 441–480 (2014). http://dx.doi.org/10.1007/s10849-014-9208-x
9. Cooper, R.: Copredication, dynamic generalized quantification and lexical innovation by coercion. In: Fourth International Workshop on Generative Approaches to the Lexicon (2007)
10. Cruse, D.A.: Lexical Semantics. Cambridge University Press, New York (1986)
11. Fodor, J.A., Lepore, E.: The emptiness of the lexicon: reflections on James Pustejovsky's the generative lexicon. Linguist. Inq. **29**(2), 269–288 (1998)
12. Im, S., Lee, C.: A developed analysis of type coercion based on type theory and conventionality. In: Cooper, R., Retoré, C. (eds.) ESSLLI Proceedings of the TYTLES Workshop on TYpe Theory and LExical Semantics, Barcelona, August 2015
13. Jacquey, E.: Ambiguïtés lexicales et traitement automatique des langues: modélisation de la polysémie logique et application aux déverbaux d'action ambigus en français. Ph.D. thesis, Université de Nancy 2 (2001)

14. Kinoshita, E., Mineshima, K., Bekki, D.: An analysis of selectional restrictions with Dependent Type Semantics. In: Proceedings of the 13th International Workshop on Logic and Engineering of Natural Language Semantics (LENLS 2013), pp. 100–113, November 2016
15. Link, G.: The logical analysis of plurals and mass terms: a lattice-theoretic approach. In: Portner, P., Partee, B.H. (eds.) Formal Semantics - The Essential Readings, pp. 127–147. Blackwell, Oxford (1983)
16. Luo, Z.: Contextual analysis of word meanings in type-theoretical semantics. In: Pogodalla, S., Prost, J.-P. (eds.) LACL 2011. LNCS, vol. 6736, pp. 159–174. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22221-4_11
17. Luo, Z.: Common nouns as types. In: Béchet, D., Dikovsky, A. (eds.) LACL 2012. LNCS, vol. 7351, pp. 173–185. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31262-5_12
18. Mery, B.: Modélisation de la Sémantique Lexicale dans le cadre de la Théorie des Types. Ph.D. thesis, Université de Bordeaux, July 2011
19. Mery, B.: Lexical semantics with linear types. In: NLCS 2015, The Third Workshop on Natural Language and Computer Science, Kyoto, Japan, July 2015
20. Mery, B.: Lessons from a prototype implementation of Montagovian lexical semantics. In: Proceedings of the 13th International Workshop on Logic and Engineering of Natural Language Semantics (LENLS 2013), pp. 73–85, November 2016
21. Mery, B., Moot, R., Retoré, C.: Computing the semantics of plurals and massive entities using many-sorted types. In: Murata, T., Mineshima, K., Bekki, D. (eds.) JSAI-isAI 2014. LNCS, vol. 9067, pp. 144–159. Springer, Heidelberg (2015). doi:10.1007/978-3-662-48119-6_11
22. Mery, B., Moot, R., Retoré, C.: 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
23. Mery, B., Retoré, C.: Are books events? Ontological inclusions as coercive subtyping, lexical transfers as entailment. In: LENLS 2012, jSAI 2015, Kanagawa, Japan, November 2015
24. Mery, B., Retoré, C.: Classifiers, sorts, and base types in the Montagovian generative lexicon and related type theoretical frameworks for lexical compositional semantics. In: Chatzikyriakidis, S., Luo, Z. (eds.) Modern Perspectives in Type-Theoretical Semantics. SLP, vol. 98, pp. 163–188. Springer, Cham (2017). doi:10.1007/978-3-319-50422-3_7
25. Moot, R.: Wide-coverage French syntax and semantics using Grail. In: Proceedings of Traitement Automatique des Langues Naturelles (TALN), Montreal (2010)
26. Moot, R., Prévot, L., Retoré, C.: A discursive analysis of itineraries in an historical and regional corpus of travels. In: Constraints in discourse. http://passage.inria.fr/cid2011/doku.php. Ayay-roches-rouges, France, September 2011. http://hal.archives-ouvertes.fr/hal-00607691/en/
27. Moravcsik, J.M.: How do words get their meanings? J. Philos. **78**(1), 5–24 (1982)
28. Muskens, R.: Meaning and partiality. In: Cooper, R., de Rijke, M. (eds.) Studies in Logic, Langage and Information. CSLI, Stanford (1996)
29. Pustejovsky, J.: The Generative Lexicon. MIT Press, Cambridge (1995)
30. Real-Coelho, L.M., Retoré, C.: A generative Montagovian lexicon for polysemous deverbal nouns. In: 4th World Congress and School on Universal Logic - Workshop on Logic and Linguistics, Rio de Janeiro (2013)
31. Retoré, C.: Sémantique des déterminants dans un cadre richement typé (2013). CoRR abs/1302.1422. http://arxiv.org/abs/1302.1422

32. Retoré, C.: The Montagovian generative lexicon lambda Ty_n: a type theoretical framework for natural language semantics. In: Matthes, R., Schubert, A. (eds.) 19th International Conference on Types for Proofs and Programs (TYPES 2013). Leibniz International Proceedings in Informatics (LIPIcs), vol. 26, pp. 202–229. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2014). http://drops.dagstuhl.de/opus/volltexte/2014/4633
33. Russell, B.: On denoting. Mind **14**(56), 479–493 (1905). http://www.jstor.org/stable/2248381