The Montagovian Generative Lexicon ΛTy_n: a Type Theoretical Framework for Natural Language Semantics*

Christian Retoré

LaBRI, Université de Bordeaux / IRIT-CNRS, Toulouse
33405 Talence cedex, France
`christian.retore@labri.fr`

Abstract

We present a framework, named the Montagovian generative lexicon, for computing the semantics of natural language sentences, expressed in many-sorted higher order logic. Word meaning is described by several lambda terms of second order lambda calculus (Girard's system F): the principal lambda term encodes the argument structure, while the other lambda terms implement meaning transfers. The base types include a type for propositions and many types for sorts of a many-sorted logic for expressing restriction of selection. This framework is able to integrate a proper treatment of lexical phenomena into a Montagovian compositional semantics, like the (im)possible arguments of a predicate, and the adaptation of a word meaning to some contexts. Among these adaptations of a word meaning to contexts, ontological inclusions are handled by coercive subtyping, an extension of system F introduced in the present paper. The benefits of this framework for lexical semantics and pragmatics are illustrated on meaning transfers and coercions, on possible and impossible copredication over different senses, on deverbal ambiguities, and on "fictive motion". Next we show that the compositional treatment of determiners, quantifiers, plurals, and other semantic phenomena is richer in our framework. We then conclude with the linguistic, logical and computational perspectives opened by the Montagovian generative lexicon.

1998 ACM Subject Classification F.4.1 Mathematical Logic, I.2.7 Natural Language Processing, I.2.4 Knowledge Representation Formalisms and Methods, D.1.1 Applicative (Functional) Programming

Keywords and phrases type theory, computational linguistics

Digital Object Identifier 10.4230/LIPIcs.TYPES.2013.202

1 Introduction: word meaning and compositional semantics

The study of natural language semantics and its automated analysis, known as computational semantics, is usually divided into *formal semantics*, usually *compositional*, which has strong connections with logic and with philosophy of language, and *lexical semantics* which rather concerns word meaning and their interrelations, derivational morphology and knowledge representation. Roughly speaking, given an utterance, formal semantics tries to determine *who does what* according to this utterance, while lexical semantics analyses the concepts under discussions and their interplay i.e. *what it speaks about*.

* This work supported by the projects ANR LOCI and POLYMNIE, and was done during my CNRS-sabbatical at IRIT.

19th International Conference on Types for Proofs and Programs (TYPES 2013).
Editors: Ralph Matthes and Aleksy Schubert; pp. 202–229

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

(1) a. *A sentence:* Some club defeated Leeds.

b. *Its formal semantics:* $\exists x : \mathtt{e}\ (\mathtt{club}(x)\ \&\ \mathtt{defeated}(x, \mathtt{Leeds}))$

(2) *Lexical semantics as found in a dictionary:* `defeat:`

a. overcome in a contest, election, battle, etc.; prevail over; vanquish

b. to frustrate; thwart.

c. to eliminate or deprive of something expected

Although applications in computational linguistics require both aspects of semantics, some applications rather focus on formal and compositional semantics, e.g. man machine dialogue, non statistical translation, text generation while other applications like information retrieval, classification, statistical translation rather stress lexical semantics.

Herein we refine compositional semantics with a treatment of some of lexical semantics issues, in particular for selecting the right word meaning in a given context. Of course any sensible analyser, including human beings, or Moot's Grail parser [49] combines both the predicate argument structures and the relations between lexical meanings to build a semantic representation and to understand an utterance.

We define a framework named *the Montagovian generative lexicon*, written $\Lambda\mathsf{Ty}_n$as it extends Ty_n of Muskens [55] with the second order Λ operator and the corresponding quantified Π types. It is based on Montague view of formal and compositional semantics [46], but we provide a faithful and computable account of some phenomena of lexical semantics, which have been addressed in particular by Pustejovsky and Asher [62, 5, 6]: correctness, polysemy, adaptation of word meaning to the context, copredication over different senses of a given expression. Our framework $\Lambda\mathsf{Ty}_n$ also suggests a finer grained analysis of some formal and compositional semantic issues such as determiners, quantification, or plurals.

Compositional semantics is usually described within simply typed lambda calculus: therefore its implementation is rather straightforward in any typed functional programming language like ML, CaML or Haskell. The computational framework for natural language semantics that we present in this paper, as well as the precise description of some semantics constructions, is defined in a subsystem of system F (second order lambda calculus), which does not go beyond the type systems of the afore mentioned functional programming languages. Hence our proposals can easily be implemented in such a language, for instance in Haskell along the lines of the good and recent book by van Eijck and Unger on *computational semantics with functional programming* [73].

1.1 The syntax of compositional semantics

As opposed to many contributions to the domain of linguistics known as "formal semantics" the present paper neither deals with reference nor with truth in a given situation: we only build a logical formula (first order or higher order, single sorted or many-sorted) that can be thereafter interpreted as one wants, if he wishes to. Hence we are not committed to any particular kind of interpretation like truth values, possible worlds, game semantics,...

In the traditional Montagovian view, the process of semantic interpretation of a sentence, consists in computing from syntax and word meanings, a logical formula (which possibly includes logical modalities and intensional operators), and in interpreting this formula in possible world semantics. Although Montague thought that intermediate representations, including the logical formulae, should be regarded as unimportant, and should be wiped off just after computing truth values and references, in this paper we precisely focus on the intermediate representations, in particular on the logical formulae, which can be called the

logical forms of sentences, with particular attention to the way they are computed – for the time being, we leave out the interpretation of these formulae. A reason for doing so is that we can encompass subtle questions, like vague predicates, generalised and vague quantifiers, for which standard notions of truth and references are inadequate: possibly some interactive interpretation would be better suited, as that proposed by Lecomte and Quatrini [33] or by Abrusci and Retoré [1].

1.2 A brief reminder on Montague semantics

Let us briefly remind the reader how one computes the logical forms according to the Montagovian view. Assume for simplicity that a syntactic analysis is a tree specifying how subtrees apply one to the other – the one that is applied is called the function while the other is called its argument. A semantic lexicon provides a simply typed λ-term $[w]$ for each word w. The semantics of a leaf (hence a word) w is $[w]$ and the semantic $[t]$ of a sub syntactic tree $t = (t_1, t_2)$ is recursively defined as $[t] = ([t_1]\ [t_2])$ that is $[t_1]$ *applied to* $[t_2]$, in case $[t_1]$ is the function and $[t_2]$ the argument – and as $[t] = ([t_2]\ [t_1])$ otherwise, i.e. when $[t_2]$ is the function and $[t_1]$ the argument. In addition to these functional applications, the tree could possibly include some λ-expressions, for instance if the syntactic structure is computed with a categorial grammar that includes hypothetical reasoning like Lambek calculus and its extensions, see e.g. [53, Chapter 3].

The typed λ-terms from the lexicon are given in such a way that the function always has a semantic type of the shape $\mathsf{a} \to \mathsf{b}$ that matches the type a of the argument, and the semantics associated with the whole tree has the semantic type t, that is the type of propositions. This correspondence between syntactical categories and semantic types, which extends to a correspondence between parse structures and logical forms is crystal clear in categorial grammars, see e.g. [53, Chapter 3]. Typed λ-terms usually are defined out of two base types, e for individuals (also known as **e**ntities) and t for propositions (which have a **t**ruth value). Logical formulae can be defined in this typed λ-calculus as first observed by Church a long time ago. This early use of lambda calculus, where formulae are viewed as typed lambda terms, cannot be merged with the more familiar view of typed lambda terms as proofs. The proof such a typed lambda term corresponds to is simply the proof that the formula is well formed, e.g. that a two-place predicate is properly applied to *two* individual terms of type e and not to more or fewer objects, nor to objects of a different type etc. This initial vision of lambda calculus was designed for a proper handling of substitution in deductive systems à la Hilbert. One needs constants for the logical quantifiers and connectives:

Quantifier	Constant	Type
there exists	$\exists$	$(\mathsf{e} \to \mathsf{t}) \to \mathsf{t}$
for all	$\forall$	$(\mathsf{e} \to \mathsf{t}) \to \mathsf{t}$

Connective	Constant	Type
and	$\&$	$\mathsf{t} \to \mathsf{t} \to \mathsf{t}$
or	$\vee$	$\mathsf{t} \to \mathsf{t} \to \mathsf{t}$
implies	$\supset$	$\mathsf{t} \to \mathsf{t} \to \mathsf{t}$

Constant	Type
`defeated`	$\mathsf{e} \to \mathsf{e} \to \mathsf{t}$
`won, voted`	$\mathsf{e} \to \mathsf{t}$
`Liverpool, Leeds`	e
...	...

as well as predicates for the precise language to be described – a binary predicate like `won` has the type $\mathsf{e} \to \mathsf{e} \to \mathsf{t}$ – as usual the type $\mathsf{a} \to \mathsf{b} \to \mathsf{c} \to \mathsf{u}$ stands for $\mathsf{a} \to (\mathsf{b} \to (\mathsf{c} \to \mathsf{u}))$ and the term $h\ t\ s\ r$ stands for $(((h\ t)\ s)\ r)$ (h being a function of arity at least 3).

word	***semantic type*** u^* ***semantics : λ-term of type*** u^* x^v *the variable or constant* x *is of type* v
some	$(\mathsf{e} \to \mathsf{t}) \to (\mathsf{e} \to \mathsf{t}) \to \mathsf{t}$ $\lambda P^{\mathsf{e}\to\mathsf{t}}.\ \lambda Q^{\mathsf{e}\to\mathsf{t}}.\ (\exists^{(\mathsf{e}\to\mathsf{t})\to t}\ (\lambda x^{\mathsf{e}}.\ (\&^{\mathsf{t}\to\mathsf{t}\to\mathsf{t}}(P\ x)(Q\ x))))$
club	$\mathsf{e} \to \mathsf{t}$ $\lambda x^{\mathsf{e}}.\ (\mathtt{club}^{\mathsf{e}\to\mathsf{t}}\ x)$
defeated	$\mathsf{e} \to \mathsf{e} \to \mathsf{t}$ $\lambda y^{\mathsf{e}}.\ \lambda x^{\mathsf{e}}.\ ((\mathtt{defeated}^{\mathsf{e}\to\mathsf{e}\to\mathsf{t}}\ x)y)$
Leeds	e $\mathtt{Leeds}$

Figure 1 A simple semantic lexicon.

A small example goes as follows. Assume the syntax says that the structure of the sentence "*Some club defeated Leeds.*" is

$$(\text{some } (\text{club})) \ (\text{defeated Leeds})$$

where the function is always the term on the left. If the semantic terms are as in the lexicon in Figure 1, placing the semantical terms in place of the words yields a large λ-term that can be reduced:

$$\Big(\big(\lambda P^{\mathsf{e}\to\mathsf{t}}.\ \lambda Q^{\mathsf{e}\to\mathsf{t}}.\ (\exists^{(\mathsf{e}\to\mathsf{t})\to\mathsf{t}}\ (\lambda x^{\mathsf{e}}.\ (\&^{\mathsf{t}\to\mathsf{t}\to\mathsf{t}}(P\ x)(Q\ x))))\big)\big(\lambda x^{\mathsf{e}}.\ (\mathtt{club}^{\mathsf{e}\to\mathsf{t}}\ x)\big)\Big)$$
$$\Big(\big(\lambda y^{\mathsf{e}}.\ \lambda x^{\mathsf{e}}.\ ((\mathtt{defeated}^{\mathsf{e}\to\mathsf{e}\to\mathsf{t}}\ x)y)\big)\ \mathtt{Leeds}^{\mathsf{e}}\Big)$$
$$\downarrow \beta$$
$$\big(\lambda Q^{\mathsf{e}\to\mathsf{t}}.\ (\exists^{(\mathsf{e}\to t)\to\mathsf{t}}\ (\lambda x^{\mathsf{e}}(\&^{\mathsf{t}\to\mathsf{t}\to\mathsf{t}}(\mathtt{club}^{\mathsf{e}\to\mathsf{t}}\ x)(Q\ x))))\big)$$
$$\big(\lambda x^{\mathsf{e}}.\ ((\mathtt{defeated}^{\mathsf{e}\to\mathsf{e}\to\mathsf{t}}\ x)\mathtt{Leeds}^{\mathsf{e}})\big)$$
$$\downarrow \beta$$
$$\big(\exists^{(\mathsf{e}\to\mathsf{t})\to\mathsf{t}}\ (\lambda x^{\mathsf{e}}.\ (\&^{\mathsf{t}\to\mathsf{t}\to\mathsf{t}}(\mathtt{club}^{\mathsf{e}\to\mathsf{t}}\ x)((\mathtt{defeated}^{\mathsf{e}\to\mathsf{e}\to\mathsf{t}}\ x)\mathtt{Leeds}^{\mathsf{e}}))))\big)$$

This λ-term of type t can be called the *logical form* of the sentence. It represents the following formula of predicate calculus (admittedly more pleasant to read):

$$\exists x : \mathsf{e}\ (\mathtt{club}(x)\ \&\ \mathtt{defeated}(x, \mathtt{Leeds}))$$

The above described procedure is quite general: starting with a properly defined semantic lexicon whose terms only contain the logical constants and the predicates of the given language one always obtains a logical formula. Indeed, such λ-terms always reduce to a unique normal form and any normal λ-term of type t (preferably η long, see e.g. [53, Chapter 3]) corresponds to a logical formula.

If we closely look at the Montagovian setting described above, we observe that it is weaving two different "logics":

Logic/calculus for meaning assembly (a.k.a glue logic, metalogic,...) In our example, this is simply typed λ-calculus with two base types e and t – these terms are the proof in intuitionistic propositional logic.

Logic/language for semantic representations In our example, that is higher-order predicate logic.[1]

[1] It can be first-order logic if reification is used, but this may induce unnatural structure and exclude some readings.

The framework we present in this paper mainly concerns the extension of the metalogic and the reorganisation of the lexicon in order to incorporate some phenomena of lexical semantics, first of all restrictions of selection. Indeed, in the standard type system above nothing prevents a mismatch between the real nature of the argument and its expected nature. Consider the following sentences:[2]

(3) a. * A chair barks.
b. * Jim ate a departure
c. ? The five is fast

Although they can be syntactically analysed, they should not receive a semantical analysis. Indeed, "*barks*" requires a "*dog*" or at least an "*animate*" subject while a "*chair*" is neither of them; "*departure*" is an event, which cannot be an "*inanimate*" object that could be eaten; finally a "*number*" like "*five*" cannot do anything fast – but there are particular contexts in which such an utterance makes sense and we shall also handle these meaning transfers.

1.3 The need of integrating lexical semantics in formal semantics

In order to block the interpretation of the semantically ill formed sentences above, it is quite natural to use *types*, where the word *type* should be understood both in its intuitive and in its formal meaning. The type of the subject of barks should be "*dog*", the type of "*fast*" objects should be "*animate*", and the type of the object of "*ate*" should be "*inanimate*". Clearly, having, on the formal side a unique type $\mathtt{e}$ for all entities is not sufficient.

The traditional view with a single type $\mathtt{e}$ for entities has another related drawback. It is unable to relate predicates whose meanings are actually related, although a usual dictionary does. A common noun like "*book*" is usually viewed as a unary predicate "$\mathtt{book}$:$\mathtt{e} \to \mathtt{t}$" while a transitive verb like "*read*" is viewed as a binary predicate "$\mathtt{read}$: $\mathtt{e} \to \mathtt{e} \to \mathtt{t}$" This gives the proper argument structure of *Mary reads a book.* as $(\exists x : \mathtt{ebook}(x)\ \&\ \mathtt{reads}(Mary, x))$ but this traditional setting cannot relate the predicates $\mathtt{book}$ and $\mathtt{read}$ – while any dictionary does. With several types, as we shall have later on, we could stipulate that the object of "*read*" ought to be something that one can "*read*", and a "*book*" can be declared as something that one can "*read*", "*write*", "*print*", "*bind*", etc. Connections between a predicate like "*book*" and predicates like "*write*", "*read*", etc. allow to interpret sentences like "*I finished my book*" which usually means "*I finished to read my book*" and sometimes "*I finished to write my book*", the other possible senses being even rarer.

Hence we need a more sophisticated type theory than the one initially used by Montague to filter semantically invalid sentences. But in many cases some flexibility is needed to accept and analyse sentences in which a word type is coerced into another type. In sentence (3c), in the context of a football match, the noun "*five*" can be considered as a player i.e. a "*person*" who plays the match with the number 5 jersey, who can "*run*".

There is a vast literature on such lexical meaning transfers and coercions, starting from 1980 [11, 12, 21, 57] – see also [32, 13] for more recent surveys of some lexical theories. In those pioneering studies, the objective is mainly to classify these phenomena, to find the rules that govern them. The quest of a computational formalisation that can be incorporated into an automated semantic analyser appears with Pustejovsky's generative lexicon in 1991

[2] We use the standard linguistic notation: a "*" in front of a sentence indicates that the sentence is incorrect, a "?" indicates that the correctness can be discussed and the absence of any symbol in front means that the sentence is correct.

[61, 62]. The integration of lexical issues into compositional semantics à la Montague and type theories appears with the work by Nicholas Asher [5, 6] which led to the book [3], and differently in some works of Robin Cooper with an intensive use of records from type theory to recover frame semantics with features and attributes inside type-theoretical compositional semantics [19, 20]

1.4 Type theories for integrating lexical semantics

As the aforementioned contributions suggest, richer type systems are quite a natural framework for formal semantics à la Montague and for selectional restriction and coercions. Such a model must extend the usual ones into two directions:

1. Montague's original type system and metalogic should be enriched to encompass lexical issues (selectional restriction and coercions), and
2. the usual phenomena studied by formal semantics (quantifiers, plurals, generics) should be extended to this richer type system and so far only Cooper and us did so [19, 20, 16, 52, 41, 35, 64]

At the end of this paper, we shall provide a comparison of the current approaches, which mainly focuses on (1). Let us list right now what the current approaches are:

- The system works with type based coercions and relies on some *Modern Type Theory (MTT)* [3] – this corresponds to the work of Zhaohui Luo [38, 39, 77, 16].
- The system works with type based coercions and relies on usual typed λ-calculus extended with some categorical logic rules – this approach by Asher [5, 6] culminated in his book [3]
- The system works with term based coercions and relies on second order λ-calculus – this is our approach, first introduced with Bassac, Mery, and further developed with Mery, Moot, Prévot, Real-Coelho. [8, 51, 50, 52, 41, 34, 35, 64, 63].

In fact our approach differs from the concurrent ones mainly because of the organisation of the lexicon and of the respective rôles of types and terms. It can be said to be word-driven, as it accounts for the (numerous) idiosyncrasies of natural language in particular the different behaviour of words of the same type is coded by assigning them different terms, while others derive everything from the types.

The precise type system we use, namely system F, could be replaced by some other type theory. However, as far as the presentation of the system is concerned, it is the *simplest* of all systems, because it only contains four term building operations (two of them being the standard λ-calculus rules, the two others being their second order counterparts) and two reduction rules (one of them being the usual beta reduction and the other one being its second order counterpart). Dependent types, which are types defined from terms, are a priori not included although they could be added if necessary.

[3] This name *Modern Type Theory (MTT)* covers several variants of modern type theories, including Martin-Löf type theory, the Predicative Calculus of (Co)Inductive Constructions (pCic), the Unifying Theory of dependent Types (UTT), ... – the latter one being the closest to the system used by Zhaohui Luo

2 A Montagovian generative lexicon for compositional semantic and lexical pragmatics

We now present our solution for introducing some lexical issues in a compositional framework à la Montague.

2.1 Guidelines for a semantic lexicon

We should keep in mind that whatever the precise solution presented, the following questions must be addressed in order to obtain a computational model, so here are the guidelines of our model:

- What is the logic for semantic representation?
  We use many-sorted *higher order predicate calculus. As usual, the higher order can be reified in first order logic, so it can be first order logic, but in any case the logic has to be many-sorted. Asher [3] is quite similar on this point, while Luo use Type Theory [39].*
- What are the sorts?
  The sorts are the base types. As discussed later on these sorts may vary from a small set of ontological kinds to any formula of one variable. We recently proposed that they correspond to classifiers *in language with classifiers: this give sorts a linguistically and cognitively motivated basis [43].*
- What is the metalogic (glue logic) for meaning assembly?
  We use second order λ-calculus (Girard system F) in order to factor operations that apply uniformly to a family of types. For specific coercions, like ontological inclusions we use subtyping introduced in the present paper. Asher [3] uses simply typed λ-terms with additional categorical rules, while Luo also use Type Theory with coercive subtyping [39].
- What kind of information is associated with a word in the lexicon?
  Here it will be a finite set of λ-terms, one of them being called the principal λ-term while the other ones are said to be optional. Other approaches make use of more specific terms and rules.
- How does one compose words and constituents for a compositional semantics?
  We simply apply one λ-term to the other, following the syntactic analysis, perform some transformations corresponding to coercions and presupposition, and reduce the compound by β-reduction.
- How is the semantic incompatibility of two components rendered?
  By type mismatch, between a function of type $A \to X$ and an argument of type $B \neq A$. Most works that insert lexical considerations into compositional semantics model incompatibility by type mismatch.
- How does one allow an *a priori* impossible composition?
  By using the optional λ-terms, which change the type of at least one of the two terms being composed, the function and argument. Both the function and the argument may provide some optional lambda terms. Other approaches rather use type-driven rules.
- How does one allow or block felicitous and infelicitous copredications on various aspects of the same word?
  An aspect can be explicitly declared as incompatible with any other aspect. More recently we saw that linear types (linear system F) can account for compatibility between arbitrary subsets of the possible aspects. [42]

Each word in the lexicon is given a principal term, as well as a finite number, possibly nought, of optional terms that licence type change and implement coercions. They may be

inferred from an ordinary dictionary, electronic or not. Terms combine almost as usual except that there might be type clashes, which account for infringements of selectional restriction: in this case optional terms may be used to solve the type mismatch. In case they lead to different results these results should be considered as different possible readings – just as the different readings with different quantifier scopes are considered by formal semantics as different possible readings of a sentence.

Let us first present the type and terms and thereafter we shall come back to the composition modes.

2.2 Remarks on the type system for semantics

We use a type system that resembles Muskens Ty_n [55] where the usual type of individuals, e is replaced with a finite but large set of base types $\mathsf{e}_1, \ldots, \mathsf{e}_n$ for individuals, for instance *objects*, *concepts*, *events*,... These base types are the sorts of the many-sorted logic whose formulae express semantic representations. The set of base types as well as their interrelations can express some ontological relations as Ben Avi and Francez thought ten years ago [10].

For instance, assume we have a many-sorted logic with a sort ζ for animals, a sort ϕ for physical objects and a predicate *eat* whose arguments are of respective sort ϕ and ζ the many-sorted formula $\forall z : \zeta\ \exists x : \phi\ eat(z, x)$ is rendered in type theory by the λ-term: $\forall^{\zeta}(\lambda z^{\zeta}.\ (\exists^{\phi}(\lambda x^{\phi}.\ ((eat\ x)z))))$ with *eat* a constant of type $\phi \to \zeta \to \mathsf{t}$. Observe that the type theoretic formulation requires a quantifier for each sort α of objects, that is a constant $\forall^{\alpha}$ of type $(\alpha \to \mathsf{t}) \to \mathsf{t}$.[4]

What are the base types? We have a tentative answer, but we cannot be too sure of this answer. Indeed, this is a subtle question depending on one's philosophical convictions, and also on the expected precision of the semantic representations,[5] but it does not really interfere with the formal and computational model we present here. Let us mention some natural sets of base types that have been proposed so far, from the smallest to the largest:

1. *A single base type* e *for all entities* (but as seen above it cannot account for lexical semantics).
2. *A very simple ontology* defines the base types: events, physical objects, living entities, concepts, ... (this resembles Asher's position in [3]).
3. *Classifiers.* Many Asian languages (Chinese, Japanese, Korean, Malay, Burmese, Nepali, ...) and all Sign Languages, have *classifiers* that are pronouns specific to classes of nouns (100–400) especially detailed for physical objects that can be handled, and for animals.There are almost no classifiers in European languages. Nevertheless a word like "*head*" in "*Three heads of cattle.*" can be considered as a classifier. Hence classifiers are a rather natural set of base types, or the importation of the classifiers of a language in one that does not have any [43]. But we do not claim that this is the definitive answer. For instance, for a specific task, some other set of base types may be better.
4. *A base type per common noun* (thousands of base types) as proposed by Luo in [39]).
5. *A type for every formula* with a single free variable.

Our opinion is that types should be cognitively natural classes and rich enough to express selectional restrictions. Whatever types are, there is a relation between types and properties.

[4] We do not speak about interpretations, but if one wishes to, we do not necessarily ask for the usual requirement that sorts are disjoint: this is coherent with the fact that in type theory, nothing prevents a pure term from having several types.

[5] For instance, a dictionary says that pregnant can be said of a "*woman or female animal*", but can it be said of a "*grandma*" or of a "*heifer*"?

With base types as in (5), the correspondence seems quite clear, but, because types can be used to express new many-sorted formulae, the set of types is in this case defined as a least fixed point. For other sets of base types, e.g. (4) or (2) for each type T there should be a corresponding predicate which recognises T entities among entities of a larger type. For instance, if there is a type *dog* there should be a predicate $\widehat{dog} : \alpha \to \mathsf{t}$ but what should be the type α of its argument? Should it be "*animal*", "*animate*",... the simplest solution is to assume a type of all individuals, that is Montague's e, and to say that corresponding to any base type a, there is a predicate, namely $\widehat{\mathsf{a}}$ of type $\mathsf{e} \to \mathsf{t}$.[6]

Let us make here a remark on the predicate constants in the language. If a predicate constant, say Q, has the type $\mathsf{u} \to \mathsf{t}$ with $\mathsf{u} \subsetneq \mathsf{e}$ – sometimes another type u is more natural than the standard e – then there is a canonical extension Q_{e} of Q to e which should be interpreted as false for any object that cannot be viewed as a u-object. Predicates constants from the first or higher order logical language do also have restrictions. Given a type u that is smaller than the domain q of Q one can define $Q|_{\mathsf{u}}$ which is defined as Q on $\mathsf{q} \cap \mathsf{u}$ and as false elsewhere.

2.3 $\Lambda\mathsf{Ty}_n$: many-sorted formulae in second order lambda calculus

Since we have many base types, and many compound types as well, it is quite convenient and almost necessary to define operations over family of similar terms with different types, to have some flexibility in the typing, and to have terms that act upon families of terms and types. Hence we shall extend further Ty_n into $\Lambda\mathsf{Ty}_n$ by using Girard's system F as the type system [25, 24]. System F involves quantified types whose terms can be specialised to any type.

The types of $\Lambda\mathsf{Ty}_n$are defined as follows:

- Constant types e_i and t are (base) types.
- Type variables $\alpha, \beta, \ldots$ are types.
- Whenever T and α respectively are a type and a type-variable, the expression $\Pi\alpha.\ T$ is a type. The type variable may or may not occur in the type T.
- Whenever T_1 and T_2 are types, $T_1 \to T_2$ is a type as well.

The terms of $\Lambda\mathsf{Ty}_n$, which encode proofs of quantified propositional intuitionistic logic, are defined as follows:

- A variable of type T i.e. $x : T$ or x^T is a *term*, and there are countably many variables of each type.
- In each type, there can be a countable set of constants of this type, and a constant of type T is a term of type T. Such constants are needed for logical operations and for the logical language (predicates, individuals, etc.).
- $(f\ t)$ is a term of type U whenever $t : T$ and $f : T \to U$.
- $\lambda x^T.\ t$ is a term of type $T \to U$ whenever $x : T$ and $t : U$.
- $t\{U\}$ is a term of type $T[U/\alpha]$ whenever $t : \Lambda\alpha.\ T$ and U is a type.
- $\Lambda\alpha.\ t$ is a term of type $\Pi\alpha.\ T$ whenever α is a type variable and $t : T$ is a term without any free occurrence of the type variable α in the type of a free variable of t.

The later restriction is the usual one on the proof rule for quantification in propositional logic: one should not conclude that $F[p]$ holds for any proposition p when assuming that a property $G[p]$ of p holds – i.e. when having a free hypothesis of type $G[p]$.

[6] An alternative solution, used by us and others [64, 17] would be $\Pi\alpha.\ \alpha \to \mathsf{t}$, using quantification over types to be defined in the next section.

The reduction of the terms in system F or its specialised version $\Lambda\mathsf{Ty}_n$is defined by the two following reduction schemes that resemble each other:

- $(\lambda x^{\phi}.\ t)u^{\phi}$ reduces to $t[u/x]$ (usual β reduction).
- $(\Lambda\alpha.\ t)\{U\}$ reduces to $t[U/\alpha]$ (remember that α and U are types).

As an example, we earlier said that in Ty_n we needed a first order quantifier per sort (i.e. per base type). In $\Lambda\mathsf{Ty}_n$it is sufficient to have a single quantifier $\forall$, that is a constant of type $\Pi\alpha.\ (\alpha \to \mathsf{t}) \to \mathsf{t}$. Indeed, this quantifier can be specialised to specific types, for instance to the base type ζ, yielding $\forall\{\zeta\} : (\zeta \to \mathsf{t}) \to \mathsf{t}$, or even to properties of ζ objects, which are of type $\zeta \to \mathsf{t}$, yielding $\forall\{\zeta \to \mathsf{t}\} : ((\zeta \to \mathsf{t}) \to \mathsf{t}) \to \mathsf{t}$. We actually do quantify over higher types, for instance in the examples below we respectively quantify over propositions with a human subject (Example 4), and over all propositions (Example 5):

(4) He did everything he could to stop them.

(5) And he believes whatever is politically correct and sounds good.

As Girard showed [25, 24] reduction is strongly normalising and confluent *every term of every type admits a unique normal form which is reached no matter how one proceeds.*[7] The normal forms, which can be asked to be η-long without loss of generality, can be characterised as follows (for a reference see e.g. [28]):

▶ Proposition 1. A normal Λ-term $\mathcal{N}$ of system F, β normal and η long to be precise, has the following structure:[8]

$$\mathcal{N} = \overbrace{(\ \lambda {x_i}^{X_i} \mid \Lambda X_j\ )^*}^{\text{sequence of } \lambda \text{ and } \Lambda \text{ abstractions}} \overbrace{(\cdots(\ h^{(\Pi X_k \mid X_l \to)^* Z}}^{\text{head variable}} \overbrace{(\{W_k\} \mid {t_l}^{X_l})^*\ )\cdots)}^{\text{sequence of } \{\cdots\} \text{ and } (\cdots) \text{ applications to types } W_k \text{ and normal terms } {t_l}^{X_l}}$$

This has a good consequence for computational semantics, see e.g. [53, Chapter 3]:

▶ Property 1 ($\Lambda\mathsf{Ty}_n$terms as formulae of a many-sorted logic). If the predicates, the constants and the logical connectives and quantifiers are the ones from a many-sorted logic of order n (possibly $n = \omega$) then the normal terms of $\Lambda\mathsf{Ty}_n$ of type t unambiguously correspond to many-sorted formulae of order n.

Let us illustrate how F factors uniform behaviours. Given types α, β, two predicates $P^{\alpha\to\mathsf{t}}$, $Q^{\beta\to\mathsf{t}}$, over entities of respective kinds α and β for any ξ with two morphisms from ξ to α and to β (see Figure 2), F contains a term that can coordinate the properties P, Q of (the two images of) an entity of type ξ, every time we are in a situation to do so – i.e. when the lexicon provides the morphisms.

▶ Term 1. [Polymorphic AND] is defined as

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

Such a term is straightforwardly implemented in Haskell along the lines of [73]:

[7] This is one way to be convinced of the soundness of F, which defines types depending on other types including themselves: as it is easily observed that there are no normal closed terms of type $\Pi X.\ X \equiv \bot$, the system is necessarily coherent. Another way is to construct a concrete model, for instance coherence spaces, where types are interpreted as countable sets with a binary relation (coherence spaces), and terms up to normalisation are interpreted as structure preserving functions (stable functions) [25].

[8] This structure resembles the structure of (weak) head normal form, in functional programming, but the terms inside the structure are also asked to be normal.

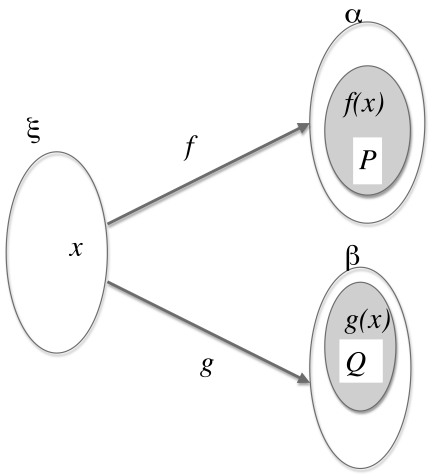

Figure 2 Polymorphic conjunction: $P(f(x))\&Q(g(x))$ with $x : \xi$, $f : \xi \to \alpha$, $g : \xi \to \beta$.

```
andPOLY :: (a -> Bool) -> (b -> Bool) -> c -> (c -> a) -> (c -> b) -> Bool
andPOLY = \ p q x f g -> p (f x) && q (g x)
```

This can apply to say, a "*book*", that can be "*heavy*" as a "*physical object*", and "*interesting*" as an "*informational content*" – the limitation of possible over-generation, that is the production or recognition of incorrect phrases or sentences, is handled by the *rigid* use of possible transformations, to be defined thereafter.

2.4 Organisation of the lexicon and rules for meaning assembly

The lexicon associates each word w with a *principal* λ*-term* $[w]$ which basically is the Montague term reminded earlier, except that the types appearing in $[w]$ belong to a much richer typed system. In particular, the numerous base types can impose some selectional restriction. In addition to this principal term, there can be *optional* λ*-terms* also called *modifiers* or *transformations* to allow, in some cases, compositions that were initially ruled out by selectional restriction.

There are two ways to solve a type conflict using those modifiers. *Flexible modifiers* can be used without any restriction. *Rigid modifiers* turn the type, or the sense of a word, into another one which is *incompatible* with other types or senses. For a technical reason, the identity, which is always a licit modifier, is also specified to be flexible or rigid. In the lalter rigid case, it means that the original sense is incompatible with any other sense, although two other senses may be compatible. Consequently, every modifier, i.e. optional λ-term is declared, in the lexicon, to be either a rigid modifier, noted (R) or a flexible one, noted (F). More subtle compatibility relations between senses can be represented by using the linear version of system F as we did in [42].

The reader may be surprised that we repeat the morphisms in the lexical entries, rather than having general rules. For instance, one could also consider morphisms that are not anchored in a particular entry: in particular, they could implement the ontology at work in [62] as the type-driven approach of Asher does [3]. For instance, a place (type Pl) could be viewed as a physical object (type ϕ) with a general morphism $Pl2\phi$ turning places into physical objects that can be "*spread out*". We are not fully enthusiastic about a general use of

word	principal λ-term	optional λ-terms	rigid/flexible
book	$\widehat{B} : \mathtt{e} \to \mathtt{t}$	$Id_B : B \to B$	(F)
		$b_1 : B \to \phi$	(F)
		$b_2 : B \to I$	(F)
town	$\widehat{T} : \mathtt{e} \to \mathtt{t}$	$Id_T : T \to T$	(F)
		$t_1 : T \to F$	(R)
		$t_2 : T \to P$	(F)
		$t_3 : T \to Pl$	(F)
Liverpool	$\mathtt{Lpl}^T$	$Id_T : T \to T$	(F)
		$t_1 : T \to F$	(R)
		$t_2 : T \to P$	(F)
		$t_3 : T \to Pl$	(F)
spreadout	$\mathtt{spread\_out} : Pl \to \mathtt{t}$		
voted	$\mathtt{voted} : P \to \mathtt{t}$		
won	$\mathtt{won} : F \to \mathtt{t}$		

where the base types are defined as follows:

B	book	T	town
ϕ	physical objects	Pl	place
I	information	P	people
		F	football team

Figure 3 A sample lexicon.

such rules since it is hard to tell whether they are flexible or rigid. As they can be composed they might lead to incorrect copredications, while their repetition inside each entry offers a better control of incorrect and correct copredications. One can think that some meaning transfer differs although the words have the same type. An example of such a situation in French is provided by the words "*classe*" and "*promotion*", which both refer to groups of pupils. The first word "*classe*" (English: "*class*") can be coerced into the room where the pupils are taught, (the "*classroom*"), while the second, "*promotion*" (English: "*class*" or "*promotion*") cannot.

Consequently, we in general prefer word-driven coercions, i.e. modifiers that are anchored in a word. An exception are ontological inclusions that are better represented by type-driven rules: "*cars*" are "*vehicles*" which are "*artefacts*" etc. This is the reason why we also allow optional terms that are available for all words of the same type. This is done by *subtyping* and more precisely by the notion of coercive subtyping that is introduced in Section 3.4.

3 A proper account of meaning transfers

In this section we shall see that the lexicon we propose, provides a proper account of the lexical phenomena that motivated its definition: ill typed readings are rejected, coerced readings are handled, felicitous copredications are analysed while infelicitous ones are rejected. Some particular case of coerced readings are given a finer analysis as the polysemy of deverbals (nouns derived from verbs, like "*construction*"), or fictive motion. Finally we introduce coercive subtyping for system F which handles general coercions corresponding to ontological inclusion.

3.1 Coercions and copredication

One can foresee what is going to happen, using the lexicon given in Figure 3 with sentences like:

(6) a. Liverpool is spread out.
b. Liverpool voted.
c. Liverpool won.

(7) Liverpool is spread out and voted (last Sunday).

(8) # Liverpool voted and won (last Sunday).

Our purpose is not to discuss whether this or that sentence is correct, nor whether this or that copredication is felicitous, but to provide a formal and computational model which given sentences that are assumed to be correct, derives the correct readings, and which given sentences that are said to be incorrect, fails to provide a reading.

Ex. (6a) This sentence leads to a type mismatch $\texttt{spread\_out}^{Pl\to\mathsf{t}}\texttt{Lpl}^{T}$, since "*spread out*" applies to "*places*" (type Pl) and not to "*towns*" as "*Liverpool*". It is solved using the optional term ${t_3}^{T\to Pl}$ provided by the entry for "*Liverpool*", which turns a town (T) into a place (Pl) $\texttt{spread\_out}^{Pl\to\mathsf{t}}({t_3}^{T\to Pl}\texttt{Lpl}^{T})$ – a single optional term is used, the (F) / (R) difference is useless.

Ex. (6b) and (6c) are treated as the previous one, using the appropriate optional terms.

Ex. 7 In this example, the fact that "*Liverpool*" is "*spread out*" is derived as previously, and the fact that "*Liverpool voted*" is obtained from the transformation of the town into people, who can vote. The two can be conjoined by the polymorphic "*and*" defined above as term 1 ($\&^{\Pi}$) because these transformations are flexible: one can use one and the other. We can make this precise using only the rules of second order typed lambda calculus. The syntax yields the predicate $(\&^{\Pi}(\texttt{spread\_out})^{Pl\to\mathsf{t}}\texttt{voted}^{P\to\mathsf{t}})$ and consequently the type variables should be instantiated by $\alpha := Pl$ and $\beta := P$ and the exact term is $\&^{\Pi}\{Pl\}\{P\}\texttt{spread\_out}^{Pl\to\mathsf{t}}\texttt{voted}^{P\to\mathsf{t}}$ which reduces to:
$\Lambda\xi.\ \lambda x^{\xi}.\ \lambda f^{\xi\to\alpha}.\ \lambda g^{\xi\to\beta}.\ (\&^{\mathsf{t}\to\mathsf{t}\to\mathsf{t}}\ (\texttt{spread\_out}^{Pl\to\mathsf{t}}\ (f\ x))(\texttt{voted}\ (g\ x)))$.
Syntax says that this term is applied to "*Liverpool*". Consequently, the instantiation $\xi := T$ happens and the term corresponding to the sentence is, after some reduction steps,
$\lambda f^{T\to Pl}.\ \lambda g^{T\to P}.\ (\&^{\mathsf{t}\to\mathsf{t}\to\mathsf{t}}(\texttt{spread\_out}^{Pl\to\mathsf{t}}\ (f\ \texttt{Lpl}^{T}))(\texttt{voted}^{P\to\mathsf{t}}\ (g\ \texttt{Lpl}^{T})))$.
Fortunately the optional λ-terms $t_2 : T \to P$ and $t_3 : T \to Pl$ are provided by the lexicon, and they can both be used, since none of them is rigid. Thus we obtain, as expected
$(\&^{\mathsf{t}\to\mathsf{t}\to\mathsf{t}}\ (\texttt{spread\_out}Pl \to \mathsf{t}\ ({t_3}^{T\to Pl}\ \texttt{Lpl}^{T}))(\texttt{voted}^{Pl\to\mathsf{t}}\ ({t_2}^{T\to P}\ \texttt{Lpl}^{T})))$.

Ex. 8 The last example is rejected as expected. Indeed, the transformation of the town into a football club prevents any other transformation (even the identity) to be used in the polymorphic "*and*" that we defined above. We obtain the same term as above, with `won` instead of `spread_out`. The corresponding term is:
$\lambda f^{T\to Pl}.\ \lambda g^{T\to P}.\ (\&^{\mathsf{t}\to\mathsf{t}\to\mathsf{t}}\ (\texttt{won}^{F\to\mathsf{t}}\ (f\ \texttt{Lpl}^{T}))(\texttt{voted}^{P\to\mathsf{t}}\ (g\ \texttt{Lpl}^{T})))$
and the lexicon provides the two morphisms that would solve the type conflict, but the one turning the Town into its football club is rigid, i.e. we can solely use this one. Consequently the sentence is semantically invalid.

3.2 Fictive motion

A rather innovative extension is to apply this technique to what Talmy called *fictive motion* [72]. Under certain circumstances, a path may introduce a virtual traveller following the

path, as in sentences like:

(9) Path GR3 descends for two hours.

Because of the duration complement "*two hours*", one cannot consider that descends means that the altitude decreases as the curvilinear abscissa goes along the path. One ought to consider someone who follows the road. We model this by one morphism associated with the "*Path GR3*" and one with "*descends*". The first coercion turns the "*Path GR3*" from an immobile object into an object of type "*path*" that can be followed and the second one coerces "*descends*" into a verb that acts upon a "*path*" object and introduces an individual following the path downwards – this individual, which does not need to exist, is quantified, yielding a proposition that can be paraphrased as "*any individual following the path goes downwards for two hours*". [51, 50]

3.3 Deverbals

Deverbals are nouns that correspond to action verbs, as "*construction*" or "*signature*". Usually they are ambiguous between result and process. We showed that our idiosyncratic model is well adapted since their possible senses vary from one deverbal to another, even if the verbs are similar and the suffix is the same.

(10) a. The construction took three months.
b. The construction is of traditional style.
c. * The building that took three months was painted white.

(11) a. The signature was illegible.
b. The signature took three months.
c. * Although it took three months the signature was illegible.
d. Although it took one minute, the signature was illegible.

We showed that a systematical treatment of deverbal meaning as the one proposed by the type-driven approach does not properly account for the data. Indeed, the possible meanings of a deverbal are more diverse than result and event, and there are no known rules to make sure the deverbal refers to the event. Consequently, word entries in the lexicon must include lexical information such as the possible meanings of the deverbal. These meanings can be derived from the event expressed by the verb: meanings usually include the event itself (but not always), the result (but not always), and other meanings as well like the place where the event happens (e.g. English noun "*pasture*"). This lexical information can be encoded in our framework, with one principal meaning and optional terms for accessing other senses and the flexibility or rigidity of these optional terms – they are usually rigid, and copredication on the different senses of a deverbal is generally infelicitous. We successfully applied our framework and treatment to the semantic of deverbals, to the restrictions of selection (both for the deverbal and for the predicate that may apply to the deverbal), to meaning transfers, and to the felicity of copredications on different senses of a deverbal [63].

3.4 Coercive subtyping and ontological inclusions

As we said earlier on, ontological inclusions like "*Human beings are animals.*", would be better modelled by optional terms that are available for any word of the type, instead of anchoring them in words and repeating these terms for every word of this type. The model we described can take these subtyping inclusions into account as standard coercions, by

transitivity

$$\frac{A < B \qquad B < C}{A < C}$$

covariance and contravariance of implication

(if identity coercions are allowed only the left most rule is needed)

$$\frac{A < B \qquad C < D}{D \to A < C \to B} \qquad \frac{A < B}{T \to A < T \to B} \qquad \frac{A < B}{B \to T < A \to T}$$

quantification over types

$$\frac{U < T[X]}{U < \Pi X.\, T[X]}\ X \textit{ not free in } U \qquad \frac{U < \Pi X.\, T[X]}{U < T[W]}$$

Figure 4 Rules for coercive subtyping in system F.

specifying that a word like "*human being*" introduces a transformation into an "*animal*". But this is somehow heavy, since one should also say that "*human beings*" are "*living beings*" etc. Any predicate, that applies to a class, also applies to an ontologically smaller class. For instance, "*run*" that applies to "*animals*" also applies to "*human beings*", because "*human*" is a subtype of "*animals*". These subtype coercions look type-driven, and, consequently, would be more faithfully modelled with a proper notion of subtyping.

Coercive subtyping, introduced by Luo and Soloviev [40, 70] for variants of Martin-Löf type theory, corresponds quite well to these particular transformations. One starts with *a transitive and acyclic set of coercions between base types, with at most one coercion between any two base types*, and from these given coercions rules derive coercions between complex types, preserving the property that there is at most one coercion between any two types.

This kind of subtyping seems adequate for modelling ontological inclusions. Indeed, such ontological inclusions when viewed as functions always are the identity on objects, hence there cannot be two different manners to map them in the larger type. Furthermore, other notions of subtyping that have been studied for higher order type theories are very complicated with tricky restrictions on the subtyping rules. [15, 37]

Coercive subtyping, noted $A_0 < A$, can be viewed as a short hand for allowing a predicate or a function which applies to A-objects to apply to an argument whose type A_0 is not the expected type A but a subtype A_0 of A. Hence coercive application is exactly what we were looking for:

coercive application

$$\frac{f : A \to B \quad u : A_0 \quad A_0 < A}{(f\ u) : B}$$

The subtyping judgements, which have the structure of categorical combinators, are derived with very natural rules given in Figure 4. These rules simply encode transitivity, covariance and contravariance of implicative types (arrow types), and quantification over type variables.

It should be observed that, given constants $\mathsf{c}_{i\to j}$ representing the coercions from a base type e_i to a base type e_j, any derivable coercion $T < U$ can be depicted by a linear Λ-term

$m : U$ of system F or $\Lambda\mathsf{Ty}_n$with a single occurrence of a free variable $x : T$ and occurrences of the constants $\mathsf{c}_{i\to j}$. The construction of the term according to the derivation rules is defined as follows:

- transitivity
$$\frac{x : A < t : B \qquad y : B < u : C}{x : A < u[y := t] : C}$$
- covariance and contravariance of implication
  - $$\frac{x : A < t : B \qquad z : C < u : D}{f : D \to A < \lambda z^C.\ t[x := f(u)] : C \to B}$$
  - $$\frac{x : A < t : B}{f : T \to A < \lambda w^T.\ t[x := f(w)] : T \to B}$$
  - $$\frac{x : A < t : B}{g : B \to T < \lambda x^A.\ g(t) : A \to T}$$
- quantification over types
  - $$\frac{u : U < t : T[X]}{u : U < \Lambda X.\ t : \Pi X.\ T[X]}\ X \textit{ not free in } U$$
  - $$\frac{u : U < t : \Pi X.\ T[X]}{u : U < t\{W\} : T[W]}$$

As easy induction shows that:

▶ Proposition 2. All terms derived in this system are linear, with a single occurrence of a single free variable (whose type is on the left of "<").

From this one easily concludes that:

▶ Proposition 3. Not all Λ-terms of system F can be derived in the subtyping system.

Any derivation c of $\mathsf{e}_i < \mathsf{e}_j$ for base types e_i, e_j is equivalent to a coercion $\mathsf{c}_{i\to j}$, i.e. our derivation system does not introduce new coercions between base types. This kind of result is similar to coherence in categories: given a compositional graph G, the free cartesian category over G does not contain any extra morphism between objects from the initial compositional graph. Here is the precise formulation of this coherence result:

▶ Proposition 4. Given a derivation of $\mathsf{e}_i < \mathsf{e}_j$ for base types e_i, e_j whose associated Λ-term is $\tilde{\mathcal{C}}$, the normal form $\mathcal{C}$ of $\tilde{\mathcal{C}}$ is a compound of $\mathsf{c}_{h\to k}$ applied to $x : \mathsf{e}_i$, which, because of the assumptions on coercions, must be $\mathsf{c}_{i\to j}$.

Proof. As seen above, a deduction of $T < U$ clearly corresponds to a linear Λ-term of system F, whose only free variable is $x : T$ with the $\mathsf{c}_{i\to j}$ as constants. Hence it has a normal from which also has a single free variable $x : T$ and $\mathsf{c}_{i\to j}$ as constants.

Let us show that any normal Λ-term $\mathcal{C}$ of type e_j with a single free variable $x : \mathsf{e}_i$ and constants $\mathsf{c}_{i\to j} : \mathsf{e}_i \to \mathsf{e}_j$ is a compound of $\mathsf{c}_{i\to j}$ applied to x^{e_i}, i.e. is a term of C_i defined by:

- $x^{\mathsf{e}_i} \in C_i$
- if $c^{\mathsf{e}_j} \in C_i$ then $(\mathsf{c}_{j\to k}(c))^{\mathsf{e}_k} \in C_i$

We proceed by induction on the number of occurrences of variables and constants in the normal term $\mathcal{C}$, whose form is, as said in Proposition 1:

$$\mathcal{C} \;=\; \overbrace{(\;\lambda {x_i}^{X_i} \mid \Lambda X_j\;)^*}^{\substack{\text{sequence of}\\ \lambda \text{ and } \Lambda \text{ abstractions}}} \; \overbrace{(\cdots(\;h^{(\Pi X_k \mid X_l \to)^* Z}}^{\substack{\text{head}\\ \text{variable}}} \; \overbrace{(\{W_k\} \mid {t_l}^{X_l})^*\;)\cdots)}^{\substack{\text{sequence of } \{\cdots\} \text{ and } (\cdots) \text{ applications}\\ \text{to types } W_k \text{ and normal terms } {t_l}^{X_l}}}$$

If the term $\mathcal{C}$ corresponds to a proof of $\mathsf{e}_i < \mathsf{e}_j$ there is no $(\;\lambda {x_i}^{X_i} \mid \Lambda X_j\;)$ in front, because e_j is neither of the form $U \to V$ nor of the form $\Pi X.\ T[X]$. What may be the head variable? It is either the only free variable of this term, namely x^{e_i}, or a constant i.e. some coercion $\mathsf{c}_{k\to l}$.

- If the head variable is x^{e_i} then, because of its type, the $(\;\{W_k\} \mid {t_l}^{X_l}\;)^*$ part of the term contains no application to a type nor to a term, hence $\mathsf{e}_i = \mathsf{e}_j$ and the normal form is x^{e_i}, which is in C_i
- If the head variable is is some $\mathsf{c}_{k\to l}$, which because of its type, may only be applied to a normal term ${t_l}^{X_l}$ of type e_k. This normal term is a normal term of type e_k with x^{e_i} as its single free variable and the constants $\mathsf{c}_{j\to l}$. As ${t_l}^{X_l}$ has one symbol less than $\mathcal{C}$, we can conclude that ${t_l}^{X_l}$ is in C_i hence $\mathcal{C} \in C_i$.

Hence in any case the normal form $\mathcal{C} : \mathsf{e}_j$ of the term $\tilde{\mathcal{C}} : \mathsf{e}_j$ is in C_i.

Now, given that the coercions $\mathsf{c}_{i\to j}$ enjoy $\mathsf{c}_{k\to j} \circ \mathsf{c}_{i\to j} = \mathsf{c}_{i\to k}$ (as part of our condition on base coercions) it is easily seen that the only term of type e_j in C_i is $\mathsf{c}_{i\to j}$. ◀

We think that this coherence result can be improved by showing that there is at most one normal term corresponding to a derivation $S < T$, although the proof is likely to use some variant of reducibility candidates [24, 25].

An alternative presentation. The rules for coercive subtyping given above follow a natural deduction style, as lambda terms of system F. Nevertheless, an alternative formulation of the quantifier elimination rule is possible. It requires identity axioms (whose term is identity) to derive obvious subtyping relations.

alternative quantifier elimination rule (sequent calculus style)

$$\frac{s : S[T] < t : U}{\dot{s} : \Pi X.\ S[X] < t[s := \dot{s}\{T\}]}$$

4 Compositional semantics issues: determiners, quantifiers, plurals

So far we have focused on phenomena in *lexical semantics* that are usually left out of standard models but properly accounted for by our model. However, we must also have a look at compositional semantics, that is the logical structure of a sentence, to see whether our model still properly analyses what standard compositional models do, and possibly, provide a better analysis. Fortunately, sentence structures are correctly analysed but furthermore our extended setting is quite appealing for some classical issues in *formal semantics* like determiners and quantification, or plurals, as we show in this section.

4.1 Determiners and quantifiers

The examples presented so far only involved proper names because we chose to extend the treatment of definite descriptions with the ι operator of some authors [68, 22, 75, 76], to indefinite articles and quantifiers. This slightly differs from the usual Montagovian setting, the one we used for "*some*" in subsection 1.2. This standard treatment of quantification can be adapted to many-sorted logic provided the two predicates, the common noun and the verb phrase, apply to the same type, or that the conjunction and implication respectively involved in existential and universal quantification allow some coercions, in the style of the polymorphic $\&^{\Pi}$.

We adopt the view of quantified, definite, and indefinite noun phrases as *individual terms* by using generic elements (or choice functions) [66, 65, 67] as initiated by Russell [68] and formalised by Hilbert [26], Ackerman [2], before Hilbert and Bernays provided a thorough presentation and discussion in [27]. This view of quantification has been adapted to linguistics by researchers like von Heusinger see e.g. [22, 75, 76], and is not that far from Steedman treatment of existential quantifiers by choice functions [71] although there are some differences that we shall not discuss here.

How do we tune our model, in particular the types, if instead of the proper name "*Liverpool*", the examples contain "*the town*", "*a town*", "*all towns*", or "*most towns*"? Indefinite determiners, quantifiers, generalised quantifiers,... are usually viewed as functions from two predicates to propositions, one expressing the restriction and the other the main predicate see e.g. [59]

As we said, and this is especially true in a categorial setting such as the one Moot implemented [49], the syntactic structure usually closely corresponds to the semantic structure. But the usual treatment of quantification that we saw in subsection 1.2 infringes this correspondence, since the semantic term "$(\lambda x.$ *Keith played* $x)$" in the semantic representation (12c) of example (12a) has no corresponding constituent in its syntactical structure (12b):

(12) a. *sentence:* Keith played some Beatles song.

b. *syntactical structure:* (Keith (played (some (Beatles song))))

c. *semantical structure:* `(some (Beatles song))` $(\lambda x.$ `Keith played` $x)$

Another criticism that applies to the usual treatment of quantifiers is the symmetry that it wrongly introduces between the main predicate and the class over which one quantifies. For instance, the two sentences below (13a,13b) usually have the same logical form (13c):

(13) a. Some politicians are crooks.

b. ? Some crooks are politicians.

c. $\exists x.$ `politician`(x) & `crook`(x)

Hence, in accordance with syntax, we prefer to consider that a quantified noun phrase is by itself some individual – a generic one which does not refer to a precise individual nor to a collection of individuals. As [75] we use η for indefinite determiners (whose interpretation picks up a new element) and ι for definite noun phrases[9] (whose interpretation picks up the most salient element). Regarding the deductive rules for handling these operators ι and η, both correspond to Hilbert's ϵ: only their interpretations in the discursive context differ.

Given a first order language $\mathcal{L}$, epsilon terms and formulae are defined by mutual recursion:

[9] Actually [75] writes ϵ instead of ι. We do not follow his notation because we also use Hilbert's ϵ with its traditional meaning.

- Any constant or variable from $\mathcal{L}$ is a term.
- $f(t_1, \ldots, t_p)$ is a term provided that each t_i is a term and f is a function symbol of arity p.
- $\epsilon_x A$ and $\tau_x A$ are terms if A is a formula, x is a variable — any free occurrence of x in A is bound by ϵ_x or $\tau_x A$.
- $s = t$ is a formula whenever s and t are terms.
- $R(t_1, \ldots, t_n)$ is a formula provided each t_i is a term and R is a relation symbol of arity n.
- $A\&B$, $A \vee B$, $A \Rightarrow B$ are formulae if A and B are formulae.
- $\neg A$ is formula if A is a formula.

As the example below shows, a formula of first order logic can be recursively translated into a formula of the epsilon calculus, without surprise:

$$\forall x \exists y P(x,y) = \exists y P(\tau_x P(x,y), y) = P(\tau_x P(x, \epsilon_y P(\tau_x P(x,y), y)), \epsilon_y P(\tau_x P(x,y), y)) \quad (14)$$

Admittedly the epsilon translations of usual formulae may look quite complicated – at least we are not used to them.

The deduction rules for τ and ϵ are the usual rules for quantification:

- From $A(x)$ with x generic in the proof (no free occurrence of x in any hypothesis), infer $A(\tau_x.\ A(x))$.
- From $B(c)$ infer $B(\epsilon_x B(x))$.

The other rules can be found by duality:

- From $A(x)$ with x generic in the proof (no free occurrence of x in any hypothesis), infer $A(\epsilon_x \neg A(x))$.
- From $B(c)$ infer $B(\tau_x \neg B(x))$.

Hence we have $F(\tau_x F(x)) \equiv \forall x.F(x)$ and $F(\epsilon_x F(x)) \equiv \exists x.\ F(x)$ and because of negation, one only of these operators is needed, usually the ϵ operator is used, and the resulting logic is known as the *epsilon calculus*: $\epsilon_x A(x) = \tau_x \neg A(x)$

Hilbert in [27] turned these symbols into a mathematically satisfying deductive system that properly describes quantification. The first and second epsilon theorem basically say that this is an alternative formulation of first order logic.

First epsilon theorem When inferring a quantifier free formula C without ϵ from quantifier free formulae Γ without ϵ, the derivation can be done within quantifier free predicate calculus.

Second epsilon theorem When inferring a formula C without ϵ symbol from formulae Γ not involving the ϵ symbol the derivation can be done within usual predicate calculus.

The epsilon calculus, restricted to the translations of usual formulae with the help of the two epsilon theorems, provided the first correct proof of Herbrand theorem (much before mistakes where found and solved by Goldfarb) and a way to prove, during the same period as Gentzen worked on the same question of the consistence of Peano arithmetic with the epsilon substitution method [27]. Later, Asser [7] and Leisenring [36] worked on the epsilon calculus more specifically in order to have models and completeness. Nevertheless, as one can read on Zentralblatt (see e.g. [14, 44]) these results are misleading as well as the posterior corrections. Only the proof theoretical aspects of the epsilon calculus seem to have been further investigated with some success by Mints[10] [45] or by Moser and Zach [54].

[10] We are sorry to learn that the great logician Grigori Mints just passed away on May 29, 2014.

In a typed model, a predicate that applies to α-objects is of type $\alpha \to \mathsf{t}$. Consequently the semantic constant ι corresponding to "*the*" introducing definite descriptions, should be of type: $(\alpha \to \mathsf{t}) \to \alpha$, and, in order to have a single constant ι, its type should be $\Pi\alpha.\ (\alpha \to \mathsf{t}) \to \alpha$.[11] Therefore, if we have a predicate $\mathtt{dog}$ that applies to entities of type $\mathsf{animate}$ the term $\iota(\mathtt{dog})$ (written $\iota x.\ \mathtt{dog}(x)$ in untyped models), i.e. the semantics of "*the dog*" is of type $\mathsf{animate}.\ldots$ but we would like this term to enjoy the property $\mathtt{dog}$! How could we say so, since the predicate $\mathtt{dog}$ does not appear in ι, but only its type. Indeed, only "*animate*" entities appear in ι as an instantiation of α. We solve this by adding a presupposition[12] $P(\iota(P))$ for any P of type $\alpha \to \mathsf{t}$, as soon as some entity enjoying P is uttered.[13]

As advocated by von Heusinger and others, indefinite descriptions that are in fact existentially quantified noun phrases are processed similarly using Hilbert's ϵ instead of ι: both ι and ϵ are constants of type $\Pi\alpha.\ (\alpha \to \mathsf{t}) \to \alpha$. Determiners are modelled in our framework by such typed constants, see [66, 65, 67]. This solution avoids the problems evoked in examples (12a) and (13b). For instance, regarding the unwanted asymmetry in the semantics of (13b) the formulae $P(\epsilon_x Q(x))$ and $Q(\epsilon_x P(x))$ are not equivalent – and neither of them is equivalent to a first order formula, but $Q(\epsilon_x P(x))$, with $P(\epsilon_x P(x))$ which is added as a presupposition, entails $P\&Q(\epsilon_x P\&Q(x)) \equiv \exists x.\ P(x)\&Q(x)$.

It should be observed that generics introduced by Hilbert's operators fit better into our typed and many-sorted semantic representations. Indeed, intuitively it is easier to think of a generic "*politician*" or "*song*" than it is to think of a generic "*entity*" or "*individual*".

One can even introduce constants that model generalised quantification. They are typed just the same way, and this construct can be applied to compute the logical form of statement including the "*most*" quantifier, as exposed in [64]. It does not mean that we have the sound and complete proof rules nor a model theoretical interpretation: we simply are able to automatically compute logical forms from sentences involving generalised and vague quantifiers such as "*most*", "*many*", "*few*".

4.2 Individuals, plurals and sets in a type-theoretical framework

The organisation of the types also allows us to handle simple facts about plurals, as shown in [52, 41] – which resembles some of Partee's ideas [58]. Here are some classical examples involving plurals, exemplifying some typical readings for plurals:

(15) a. *Keith met.
b. Keith and John met. (unambiguous).

(16) a. *The student met.
b. The students met. (unambiguous, one meeting)

(17) a. The committee met. (unambiguous, one meeting)
b. The committees met. (ambiguous: one big meeting, one meeting per committee, several meetings invoking several committees)

[11] An alternative type working with any predicate $\widehat{\alpha}$ that corresponds to a type α, would be $\Pi\alpha.\ \alpha$.
[12] A presupposition is a proposition which is not explicitly stated but which is assumed by the uttered proposition and by its negation as well: "*Keith stopped smoking.*" and "*Keith did not stop smoking*" both presuppose "*Keith used to smoke.*". Observe that a typing judgement $t : \mathsf{a}$ is not easy to refute, as a presupposition: indeed after "*The dog is sleeping on the sofa.*" one can hardly answer "*It is not an animal.*" or "*It is not a dog.*" although one can say "*It is not sleeping*".
[13] If the predicate P corresponds to a type τ i.e. $P = \widehat{\tau}$, this presupposition is better written as $\iota(\widehat{\tau}) : \tau$.

q $\quad \Lambda\alpha.\ \lambda x^{\alpha}.\ \lambda y^{\alpha}.\ x = y$
$*$ $\quad \Lambda\alpha\lambda P^{\alpha\to\mathsf{t}}.\ \lambda Q^{\alpha\to\mathsf{t}}.\ \forall x^{\alpha}.\ Q(x) \Rightarrow P(x)$
$\#$ $\quad \Lambda\alpha\lambda R^{(\alpha\to\mathsf{t})\to\mathsf{t}}.\ \lambda S^{\alpha\to\mathsf{t}\to\mathsf{t}}.\ \forall P^{\alpha\to\mathsf{t}} S(P) \Rightarrow R(P)$
c $\quad \Lambda\alpha.\ \lambda R^{(\alpha\to\mathsf{t})\to\mathsf{t}}.\ \lambda P^{\alpha\to\mathsf{t}}.\ \forall x^{\alpha}.\ P(x) \Rightarrow \exists Q^{\alpha\to\mathsf{t}} Q(x) \wedge (\forall y^{\alpha} Q(y) \Rightarrow P(y)) \wedge R(Q)$

Figure 5 Some operators for plurals.

(18) a. The students wrote a paper. (unambiguous)
b. The students wrote three papers. (covering)

Such readings are derivable in our model because one can define in F operators for handling plurals. Firstly, one can add, as a constant, a cardinality operator for predicates $||\_|| : \Pi\alpha.\ (\alpha \to \mathsf{t}) \to \mathbb{N}$ where $\mathbb{N}$ are the internal integers of system F, namely $\mathbb{N} = \Pi X.\ (X \to X) \to (X \to X)$, or a predefined integer type as in Gödel system T – this might be problematic if infinitely many objects satisfied the predicate, but syntax and restriction of selection can make sure it is only applied when it makes sense. Secondly, as shown in Figure 5, we can have operators for handling plurals: q (turning an individual into a property/set, a curried version of equality), $*$ (distributivity), $\#$ (restricted distributivity from sets of sets to its constituent subsets), c (for coverings), etc. The important fact is that the computation of such readings uses exactly the same mechanisms as lexical coercion. Some combinations are blocked by their types, but optional terms coming either from the predicate or from the plural noun may allow an a priori prohibited reading. To be precise we also provide specific tools for handling groups that are singular nouns, each of which denoting a set. All these functions are easily implemented in a typed functional programming language like Haskell, in the style of [73].

5 Comparison with related work and conclusion

5.1 Variants and implementation

Some variation is possible in the above definition of the Montagovian generative lexicon without changing its general organisation. For instance, as suggested in the beginning of section 2 the set of base types can be discussed. We proposed to use classifiers as base types of a language with classifiers, because classifiers are linguistically and cognitively motivated classes of words and entities. But it is fairly possible that other sets of base types are better suited in particular for specific applications [43].

In relation to this issue, the inclusion between base types, which in our model are morphisms, can be introduced with words or as general axioms. We prefer the first solution that allows idiosyncratic behaviours, dependent on words as explained in subsection 2.4 with "*classe*" and "*promotion*". Nevertheless when dealing with ontological inclusions, or other very general coercions, we think a subtyping approach is possible and reduces the size of the lexicon, this is why we are presently exploring coercive subtyping.

The type we gave for predicates can also vary: it could be systematically $\mathsf{e} \to \mathsf{t}$, but as explained in paragraph 4, types $\mathsf{u} \to \mathsf{t}$ are possible as well – but transition from one form to another is not complicated.

An important variant is to define the very same ideas within a compositional model like λ-DRT [56] the compositional view of Discourse Representation Theory [29] which can, as its name suggests, handle discursive phenomena. Thus one can integrate the semantical and lexical issues presented here into a broader perspective. This can be done, and in fact

several applications of the model presented here are already included in the Grail parser by Richard Moot, in particular for French [49]. The *grammar* was automatically extracted from annotated corpora, but unfortunately the refined semantic terms we need can only be typed by hand. Consequently we only tested the semantic analyses described herein on a small specific lexicon. For instance, our treatment of fictive motion (cf. subsection 3.2) has been tested with a detailed lexicon for spatial semantics, but with λ-DRT [50] rather than plain lambda calculus [51]. The Grail parser is written with Prolog and as far as semantics is concerned, a functional programming language like CaML or Haskell would be better suited, as van Eijck and Unger show in [73].

5.2 Comparison with related work

There are many similarities with the contemporary work by Asher and Luo [4, 39, 16].

A first difference is the type system. Our type system, F, is quite powerful but simple: four-term building operations, and two reduction rules. Luo makes use of a version of Modern Type Theories (MTT), closed to the Unifying Theory of dependent Types (UTT), whose expressive power and computational complexity is difficult to compare: it is predicative but it includes dependent types. Hence it is not clear whether MTT better characterises the logic needed for meaning assembly. Quantification over type variables is quite comparable and admits $\forall \alpha : CN$ (CN are common nouns) which is quite convenient although it can certainly be encoded within system F using the fact that finite sums can be defined in system F, hence $x : \alpha, \alpha : CN$ can be rephrased if there are finitely many CN. This is both a positive and negative feature of system F: it can encode many things, but encodings are often dull. A possible solution, similar to [69], is to introduce predefined types F with specific reduction schemes – e.g. adding integers as in Gödel's system T.

Regarding coercions, Luo [38] makes an extensive use of coercive subtyping, which he introduced with Soloviev [70]: as said in their paper this kind of subtyping may also work well with system F. So we can say that the system of Luo is very similar. Dependent types and predicative quantification may be closer to what we wish to model, but the formal diversity of the numerous employed rules may result in an obscure formalisation. The typed system at work in Asher's view [3] is a simple type theory extended with type constructors and operations imported from category theory. The theory extends cartesian closed category with a few of the many operations that one finds in topos theory, like being a subobject. This approach is difficult to compare with the two above, since it does not belong to the same family: morphisms do not represent (quotiented) proofs of some logic, they are closer to a set theoretic interpretation.

Another ingredient of our models are base types. Asher leaves the set of base types open, but rather small (say a dozen) : e, t, physical objects, etc., with a linguistically motivated subtyping relation $\sqsubset$ defined over these types. Luo, especially in his later article [39], wants to equate base types with common nouns (also with coercions between them), and this is a possible compromise between any formula and the minimal base type system which makes it difficult to express some selectional restrictions with types. However it seems that there are too many of them, since not every common noun appears as a restriction of selection for another word in a dictionary. Dealing with classifiers as base types is a recent proposal of ours which seems cognitively and linguistically motivated. It is worth exploring this hypothesis empirically in tests over corpora.

The subtyping relation between base types are language independent in these two models, i.e. they are not triggered by words, but simply by types. We opted for a compromise in which only ontological inclusions are type-driven, using coercive subtyping, while other coercions are word-driven.

Regarding the general organisation of the lexicon and its composition modes, the same difference applies. While according to Asher and Luo, types determine the coercions, in our approach the coercions are provided by the terms in the lexicon, i.e. by the words themselves and not by their types, with an exception for ontological inclusions. The recent claim by Luo that base type should be common nouns (that are words) partly blurs the differences between on the one hand the type-driven approaches of himself and Asher and, on the other hand, ours which is more idiosyncratic being based on words and terms that are known to be arbitrary.

Finally one may wonder whether we finally derive similar logical forms. They actually are quite similar: we derive higher order many-sorted logical formulae, Asher derives formulae in a category that can be seen as an intuitionnistic set theory, which works with sorts, and Luo derives formulae of type theory. All these are more or less the same: higher order is possible, although not extensively used in examples, and there are sorts or types.

A possible difference may lie in the distance between syntax and semantics. Indeed, the effective computability of the semantic representations requires a specific treatment of the common structures in compositional semantics like determiners, quantifiers, plurals,... and to be integrated in a general analysis that also includes phenomena like time or aspect. For the time being we did more on such issues than the others, but I am pretty sure that a similar treatment is possible within the approach developed by Asher and Luo.

5.3 Perspectives

Apart from fixing up the optimal variant among the possible variants of our model, to study and develop the convergence with related work, or to pursue the implementation, there are some questions both on type theory and on linguistic modelling, both theoretical and practical, that deserve to be further studied.

The *acquisition* of the semantic lexicon has both theoretical and practical aspects. In particular, how could one acquire the optional lambda terms that represent coercions? Syntactic information on words can be automatically extracted: indeed, Moot's parser that we used to experiment our type theoretical semantic analyses was automatically acquired [48, 47]. By now there are some techniques [78] to extract the usual lambda terms of Montague semantics of subsection 1.2 that represent the argument structure of words. Machine learning (see e.g. [23]) and serious games (that are games with an outcome besides entertainment, and in particular collaborative games with a purpose [74] that have been shown to be quite efficient for constructing linguistic ressources) are also able to learn relations between words see e.g. [18, 31]. However, up to now there are no learning algorithms for acquiring a set of base type, nor for determining given a set of base type, the optional lambda terms, and our experiments with Moot parser were performed using a hand typed semantic lexicon.

On the logical side there are many intriguing questions.

- One is the relation in a type system with sorts between the (higher order) predicate calculus and the type system, exemplified by the relation between type judgements $x : T$ that, as linguistic presuppositions, cannot be denied and predicates $\widehat{T}(x)$ that can be denied.
- The Hilbert operator ϵ, which looks more natural in this typed system, deserves to be further studied. Since most of the results are false but Hilbert's original results, the study of both the deductive system and the interpretation of those operators is appealing. In particular, we are puzzled by formulae with Hilbert operators that have no corresponding formula in usual logic.

- The coercive subtyping we introduced in this paper should also be further explored, e.g. by proving that there is at most one coercion between any two types.
- It is quite clear that we do not need the full power of system F: we chose this system of variable types and quantified types for its simplicity and elegance. Nevertheless one may wonder whether there is a simple restriction that would be sufficient. Linear versions of system F both have a lower complexity [30] and allow a finer grained treatment of the constraints on sense compatibility [42].

Regarding computational linguistics, and applications to natural language processing, the way the discourse context is handled is important. In particular, the permanence and the propagation of constraints (e.g. on sense compatibilities) through linguistic structure deserves to be further studied. Observe that:

(19) a. This salmon was living nearby Scottish coast. It was delicious.
b. ? This salmon that was living nearby Scottish coast was delicious.
c. * This salmon was living nearby Scottish coast and was delicious.

We believe that the type theoretical and many-sorted view presented in this paper may shed new light on classical challenges of natural language semantics. A known difficult example is the semantics of mass nouns, like *wine*, which can be quantified:

(20) a. He drank some wine.
b. He drank all the wine.

Thanks. Special thanks to Sergeï Soloviev for his explanations on coercive subtyping during my CNRS sabbatical at IRIT.

Many thanks to Christian Bassac who introduced me to the topic, to my coauthors Bruno Mery (Bordeaux), Richard Moot (Bordeaux), Michele Abrusci (Roma), Laurent Prévot (Aix), Livy Real (Curitiba). I also thanks for helpful discussions Nicholas Asher, Zhaohui Luo, Marta Abrusan, Claire Beyssade, Heather Burnett, Sarah-Jane Conrad, Francis Corblin, Fabio Del Prete, Alda Mari, Hazel Pearson.

Finally let me thank the anonymous reviewers for providing unusually insightful comments, and the editors for their patience.

References

1 Vito Michele Abrusci and Christian Retoré. Quantification in ordinary language: from a critic of set-theoretic approaches to a proof-theoretic proposal. In Peter Schröder-Heister, editor, *14th Congress of Logic, Methodology and Philosophy of Sciences*, 2011.
2 W. Ackermann. Begründung des "tertium non datur" mittels der Hilbertschen Theorie der Widerspruchsfreiheit. *Mathematische Annalen*, 93:1–36, 1924.
3 Nicholas Asher. *Lexical Meaning in context – a web of words*. Cambridge University press, 2011.
4 Nicholas Asher and Zhaohui Luo. Formalization of coercions in lexical semantics. In Emmanuel Chemla, Vincent Homer, and Grégoire Winterstein, editors, *Sinn und Bedeutung 17*, pages 63–80, 2012. `http://semanticsarchive.net/sub2012/`.
5 Nicholas Asher and James Pustejovsky. The metaphysics of words in contexts, 2000.
6 Nicolas Asher. A type driven theory of predication with complex types. *Fundamenta Informaticae*, 84(2):151–183, 2008.
7 Gunter Asser. Theorie der logischen auswahlfunktionen. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 1957.

8 Christian Bassac, Bruno Mery, and Christian Retoré. Towards a Type-Theoretical Account of Lexical Semantics. *Journal of Logic Language and Information*, 19(2):229–245, April 2010.

9 Denis Béchet and Alexander Ja. Dikovsky, editors. *Logical Aspects of Computational Linguistics – 7th International Conference, LACL 2012, Nantes, France, July 2–4, 2012. Proceedings*, volume 7351 of *Lecture Notes in Computer Science*. Springer, 2012.

10 Gilad Ben-Avi and Nissim Francez. Categorial grammars with ontology-refined types. In *Categorial grammars – an efficient tool for natural language processing*, pages 99–113, Montpellier, June 2004. C.N.R.S.

11 Manfred Bierwisch. Wörtliche bedeutung - eine pragmatische gretchenfrage. In G. Grewendorf, editor, *Sprechakttheorie und Semantik*, pages 119–148. Surkamp, Frankfurt, 1979.

12 Manfred Bierwisch. Semantische und konzeptuelle repräsentation lexikalischer einheiten. In R. Rùžička and W. Motsch, editors, *Untersuchungen zur Semantik*, pages 61–99. Akademie-Verlag, Berlin, 1983.

13 Reinhard Blutner. Lexical semantics and pragmatics. In Fritz Hamm and Thomas Ede Zimmermann, editors, *Semantics*, volume 10 (Sonderheft), pages 27–58, Hamburg, 2002. Buske.

14 J. T. Canty. Zbl0327.02013 : review of "on an extension of Hilbert's second ϵ-theorem" by T. B. Flanagan (journal of symbolic logic, 1975). *Zentralblatt Math.*

15 Luca Cardelli, Simone Martini, John C. Mitchell, and Andre Scedrov. An extension of system F with subtyping. *Information and Computation*, 109(1/2):4–56, 1994.

16 Stergios Chatzikyriakidis and Zhaohui Luo. An account of natural language coordination in type theory with coercive subtyping. In Denys Duchier and Yannick Parmentier, editors, *7th International Workshop on Constraint Solving and Language Processing (CSLP'12). Selected and Revised Papers*, number 8114 in Lecture Notes in Computer Science. Springer, 2013.

17 Stergios Chatzikyriakidis and Zhaohui Luo. Adjectives in a modern type-theoretical setting. In Glyn Morrill and Mark-Jan Nederhof, editors, *FG*, volume 8036 of *Lecture Notes in Computer Science*, pages 159–174. Springer, 2013.

18 Philipp Cimiano and Johanna Wenderoth. Automatic acquisition of ranked qualia structures from the web. In John A. Carroll, Antal van den Bosch, and Annie Zaenen, editors, *ACL*. The Association for Computational Linguistics, 2007.

19 Robin Cooper. Copredication, dynamic generalized quantification and lexical innovation by coercion. In *Fourth International Workshop on Generative Approaches to the Lexicon*. Université de Genève, 2007.

20 Robin Cooper. Copredication, quantification and frames. In Pogodalla and Prost [60], pages 64–79.

21 D.A. Cruse. *Lexical semantics*. Cambridge textbooks in linguistics. Cambridge University Press, 1986.

22 Urs Egli and Klaus von Heusinger. The epsilon operator and E-type pronouns. In Urs Egli, Peter E. Pause, Christoph Schwarze, Arnim von Stechow, and Götz Wienold, editors, *Lexical Knowledge in the Organization of Language*, pages 121–141. Benjamins, 1995.

23 Peter Flach. *Machine Learning: The Art and Science of Algorithms That Make Sense of Data*. Cambridge University Press, New York, NY, USA, 2012.

24 Jean-Yves Girard. Une extension de l'interprétation de Gödel à l'analyse et son application: l'élimination des coupures dans l'analyse et la théorie des types. In Jens Erik Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, volume 63 of *Studies in Logic and the Foundations of Mathematics*, pages 63–92, Amsterdam, 1971. North Holland.

25 Jean-Yves Girard. *The blind spot – lectures on logic*. European Mathematical Society, 2011.

26 David Hilbert. Die logischen grundlagen der mathematik. *Mathematische Annalen*, 88:151–165, 1922.

27 David Hilbert and Paul Bernays. *Grundlagen der Mathematik. Bd. 2.* Springer, 1939. Traduction française de F. Gaillard, E. Guillaume et M. Guillaume, L'Harmattan, 2001.

28 Gérard P. Huet. *Résolution d'équations dans des langages d'ordre 1,2,...,ω.* Thèse de doctorat d'état, Université Paris VII, 1976.

29 Hans Kamp and Uwe Reyle. *From Discourse to Logic.* D. Reidel, Dordrecht, 1993.

30 Yves Lafont. Soft linear logic and polynomial time. *Theoretical Computer Science*, 318(1–2):163 – 180, 2004.

31 Mathieu Lafourcade and Alain Joubert. Computing trees of named word usages from a crowdsourced lexical network. In *IMCSIT*, volume Computational Linguistics – Applications (CLA'10), pages 439–446, 2010.

32 Sven Lauer. A comparative study of current theories of polysemy in formal semantics. Master's thesis, Cognitive science Osnabrück - Computational Linguistics, 2004.

33 Alain Lecomte and Myriam Quatrini. Figures of dialogue: a view from ludics. *Synthese*, 183:59–85, 2011.

34 Anaïs Lefeuvre, Richard Moot, and Christian Retoré. Traitement automatique d'un corpus de récits de voyages pyrénéens : analyse syntaxique, sémantique et pragmatique dans le cadre de la théorie des types. In SHS Web of Conferences, editor, *Congrès mondial de linguistique française*, pages 2485–2497, 2012.

35 Anaïs Lefeuvre, Richard Moot, Christian Retoré, and Noémie-Fleur Sandillon-Rezer. Traitement automatique sur corpus de récits de voyages pyrénéens : Une analyse syntaxique, sémantique et temporelle. In *Traitement Automatique du Langage Naturel, TALN'2012*, volume 2, pages 43–56, 2012.

36 Albert C. Leisenring. *Mathematical logic and Hilbert's ϵ symbol.* University Mathematical Series. Mac Donald & Co., 1967.

37 Giuseppe Longo, Kathleen Milsted, and Sergei Soloviev. Coherence and transitivity of subtyping as entailment. *Journal of Logic and Computation*, 10(4):493–526, 2000.

38 Zhaohui Luo. Contextual analysis of word meanings in type-theoretical semantics. In Pogodalla and Prost [60], pages 159–174.

39 Zhaohui Luo. Common nouns as types. In Béchet and Dikovsky [9], pages 173–185.

40 Zhaohui Luo, Sergei Soloviev, and Tao Xue. Coercive subtyping: Theory and implementation. *Inf. Comput.*, 223:18–42, 2013.

41 Bruno Mery, Richard Moot, and Christian Retoré. Plurals: individuals and sets in a richly typed semantics. In Shunsuke Yatabe, editor, *Logic and Engineering of Natural Language Semantics 10 (LENLS 10)*, pages 143–156. Keio University, 2013. ISBN 978-4-915905-57-5.

42 Bruno Mery and Christian Retoré. Advances in the logical representation of lexical semantics. In Valeria de Paiva and Larry Moss, editors, *Natural Language and Computer Science (LICS 2013 satellite workshop)*, New-Orleans, 2013.

43 Bruno Mery and Christian Retoré. Semantic types, lexical sorts and classifiers. In B. Sharp and M. Zock, editors, *10th International Workshop on Natural Language Processing and Cognitive Science*, Marseilles, September 2013.

44 G. Mints. Zbl0381.03042: review of "cut elimination in a Gentzen-style ϵ-calculus without identity" by Linda Wessels (Z. math Logik Grundl. Math., 1977). *Zentralblatt Math.*

45 Grigori Mints. Cut elimination for a simple formulation of epsilon calculus. *Ann. Pure Appl. Logic*, 152(1-3):148–160, 2008.

46 Richard Montague. English as a formal language. In Bruno Visentini, editor, *Linguaggi nella Societa e nella Tecnica*, pages 189–224. Edizioni di Communità, Milan, Italy, 1970. (Reprinted in R. Thomason (ed) *The collected papers of Richard Montague* Yale University Press, 1974.).

47 Richard Moot. Automated extraction of type-logical supertags from the spoken dutch corpus. In Srinivas Bangalore and Aravind Joshi, editors, *The Complexity of Lexical Descriptions and its Relevance to Natural Language Processing: A Supertagging Approach*. MIT Press, 2007.

48 Richard Moot. Semi-automated extraction of a wide-coverage type-logical grammar for French. In *Proceedings of Traitement Automatique des Langues Naturelles (TALN)*, Montreal, 2010.

49 Richard Moot. Wide-coverage French syntax and semantics using Grail. In *Proceedings of Traitement Automatique des Langues Naturelles (TALN)*, Montreal, 2010.

50 Richard Moot, Laurent Prévot, and Christian Retoré. A discursive analysis of itineraries in an historical and regional corpus of travels. In *Constraints in discourse*, Ayay-roches-rouges, France, September 2011. `http://passage.inria.fr/cid2011/doku.php`.

51 Richard Moot, Laurent Prévot, and Christian Retoré. Un calcul de termes typés pour la pragmatique lexicale – chemins et voyageurs fictifs dans un corpus de récits de voyages. In *Traitement Automatique du Langage Naturel, TALN 2011*, pages 161–166, Montpellier, France, June 2011.

52 Richard Moot and Christian Retoré. Second order lambda calculus for meaning assembly: on the logical syntax of plurals. In Reinhard Muskens, editor, *Coconat: Conference on Computing Natural Reasoning*. University of Tilburg, December 2011. `http://hal.inria.fr/hal-00650644`.

53 Richard Moot and Christian Retoré. *The logic of categorial grammars: a deductive account of natural language syntax and semantics*, volume 6850 of *LNCS*. Springer, 2012.

54 Georg Moser and Richard Zach. The epsilon calculus and herbrand complexity. *Studia Logica*, 82(1):133–155, 2006.

55 Reinhard Muskens. Anaphora and the logic of change. In Jan van Eijck, editor, *JELIA*, volume 478 of *Lecture Notes in Computer Science*, pages 412–427. Springer, 1990.

56 Reinhard Muskens. Combining Montague Semantics and Discourse Representation. *Linguistics and Philosophy*, 19:143–186, 1996.

57 Geoffrey Nunberg. Transfers of meaning. *Journal of semantics*, 12(2):109–132, 1995.

58 Barbara Partee. Noun phrase interpretation and type shifting principles. In B.H. Partee and P.H. Portner, editors, *Formal Semantics: The Essential Readings*, pages 357–381. Wiley, 2008.

59 Stanley Peters and Dag Westerståhl. *Quantifiers in Language and Logic*. Clarendon Press, 2006.

60 Sylvain Pogodalla and Jean-Philippe Prost, editors. *Logical Aspects of Computational Linguistics – 6th International Conference, LACL 2011, Montpellier, France, June 29 to July 1, 2011. Proceedings*, volume 6736 of *LNCS*. Springer, 2011.

61 James Pustejovsky. The generative lexicon. *Computational Linguistics*, 17(4):409–441, 1991.

62 James Pustejovsky. *The generative lexicon*. M.I.T. Press, 1995.

63 Livy Real and Christian Retoré. Deverbal semantics and the Montagovian generative lexicon ΛTy_n. *Journal of Logic Language and Information*, 2014. 10.1007/s10849-014-9187-y.

64 Christian Retoré. Variable types for meaning assembly: a logical syntax for generic noun phrases introduced by "most". *Recherches Linguistiques de Vincennes*, 41:83–102, 2012.

65 Christian Retoré. A natural framework for natural language semantics: many sorted logic and Hilbert operators in type theory. In Mário Edmundo and Boban Velickovic, editors, *Logic colloquium*, Evora, 2013.

66 Christian Retoré. Sémantique des déterminants dans un cadre richement typé. In Emmanuel Morin and Yannick Estève, editors, *Traitement Automatique du Langage Naturel, TALN RECITAL 2013*, volume 1, pages 367–380. ACL Anthology, 2013.

67 Christian Retoré. Typed hilbert epsilon operators and the semantics of determiner phrases (invited lecture). In Glyn Morrill, Reinhard Muskens, Rainer Osswald, and Frank Richter, editors, *Proceedings of Formal Grammar 2014*, number 8612 in LNCS/FoLLI, pages 15–33. Springer, 2014. Invited lecture.

68 Bertrand Russell. On denoting. *Mind*, 56(14):479–493, 1905.

69 Sergei Soloviev and David Chemouil. Some Algebraic Structures in Lambda-Calculus with Inductive Types. In Stefano Berardi, Mario Coppo, and Ferruccio Damiani, editors, *TYPES*, volume 3085 of *Lecture Notes in Computer Science*, pages 338–354. Springer, 2003.

70 Sergei Soloviev and Zhaohui Luo. Coercion completion and conservativity in coercive subtyping. *Annals of Pure and Applied Logic*, 1-3(113):297–322, 2000.

71 Mark Steedman. *Taking Scope: The Natural Semantics of Quantifiers*. MIT Press, 2012.

72 Leonard Talmy. Fictive motion in language and "ception". In Paul Bloom, Mary A. Peterson, Lynn Nadel, and Merrill F. Garrett, editors, *Language and Space*, pages 211–276. MIT Press, 1999.

73 Jan van Eijck and Christina Unger. *Computational Semantics with Functional Programming*. Cambridge University Press, 2010.

74 Luis von Ahn. Games with a purpose. *Computer*, 39(6):92–94, 2006.

75 Klaus von Heusinger. Definite descriptions and choice functions. In S. Akama, editor, *Logic, Language and Computation*, pages 61–91. Kluwer, 1997.

76 Klaus von Heusinger. Choice functions and the anaphoric semantics of definite nps. *Research on Language and Computation*, 2:309–329, 2004.

77 Tao Xue and Zhaohui Luo. Dot-types and their implementation. In Béchet and Dikovsky [9], pages 234–249.

78 Luke S. Zettlemoyer and Michael Collins. Learning context-dependent mappings from sentences to logical form. In Keh-Yih Su, Jian Su, and Janyce Wiebe, editors, *ACL/IJCNLP*, pages 976–984. The Association for Computer Linguistics, 2009.