Journal of Logic, Language and Information (2019) 28:287–307
https://doi.org/10.1007/s10849-019-09290-7

Natural Language Semantics and Computability

Richard Moot[1] · **Christian Retoré**[1]

Published online: 19 April 2019

Abstract
This paper is a reflexion on the computability of natural language semantics. It does not contain a new model or new results in the formal semantics of natural language: it is rather a computational analysis, in the context for type-logical grammars, of the logical models and algorithms currently used in natural language semantics, defined as a function from a grammatical sentence to a (non-empty) set of logical formulas—because a statement can be ambiguous, it can correspond to multiple formulas, one for each reading. We argue that as long as we do not explicitly compute the interpretation in terms of possible world models, one can compute the semantic representation(s) of a given statement, including aspects of lexical meaning. This is a very generic process, so the results are, at least in principle, widely applicable. We also discuss the algorithmic complexity of this process.

Keywords Categorial grammar · Complexity · Proof theory

1 Introduction

In the well-known Turing test for artificial intelligence, a human interrogator needs to decide, via a question answering session with two terminals, which of his two interlocutors is a man and which is a machine (Turing 1950). Although early systems like Eliza based on matching word patterns may seem clever at first sight, they clearly do not pass the test. One often forgets that, in addition to reasoning and access to knowledge representation, passing the Turing test presupposes automated natural language analysis and generation which, despite significant progress in the field, have not yet been fully achieved. These natural language processing components of the Turing test are of independent interest and used in computer programs for question answering and translation—however, since both of these tasks are generally assumed to be AI-

✉ Richard Moot
Richard.Moot@lirmm.fr

[1] LIRMM, Montpellier University, CNRS, 161 Rue Ada, 34095 Montpellier Cedex 5, France

complete it is unlikely that a full solution for these problems would be simpler than a solution for the Turing test itself.

If we define the interpretation function of a (sequence of) sentence(s) σ as the mapping to a representation $\phi(\sigma)$ (it *semantics*) that can be used by a machine for natural language processing tasks, two very different ideas of semantics come to mind.

1. One notion of semantics describes what the sentence(s) speaks about. Approaches using word embedding or distributional semantics fall in this group. The dominant model for this type of semantics represents meaning using word vectors (only involving referential/full words nouns, adjectives, verbs, adverbs, …and not grammatical words) which represent what σ speaks about. This is clearly computable. One standard way of achieving this is to fix a thesaurus of n words that acts as a vector basis. Usually words not in the thesaurus or basis are expanded into their definition with words in the thesaurus. By counting occurrences of words from the thesaurus in the text (substituting words not in the thesaurus with their definition) and turning this into a n-dimensional vector reduced to be of Euclidian norm 1, we obtain word meanings in the form of n-dimensional vectors. This notion of semantics provides a useful measure of semantic similarity between words and texts, using measures such as the cosine between different vectors; typical applications include exploring Big Data and finding relevant pages on the internet. This kind of semantics models what a word (or a text) speaks about.
2. The other notion of semantics, the one this paper is about, is of a logical nature. It models what is asserted by the sentences. According to this view, computational semantics is the mapping of sentence(s) to logical formula(s). This is usually done compositionally, according to Frege's principle *"the meaning of a compound expression is a function of the meaning of its components"* to which Montague added *"and of its syntactic structure"*. This paper focuses on this logical and compositional notion of semantics and its extension (by us and others) to lexical semantics; these extensions allow us to conclude from a sentence like "I started a book" that the speaker started *reading* (or, depending on the context, writing) a book.

We should comment that, in our view, the semantic interpretation is a (computable) function from sentence(s) to logical formulas representing their different meanings, since this viewpoint is not so common in linguistics.

– Cognitive sciences also consider the language faculty as a computational device and insist on the computations involved in language analysis and production. Actually there are two different views of this cognitive and computational view: one view, promoted by authors such as Pinker (1994), claims that there is a specific cognitive function for language, a "language module" in the mind, while others, like Langacker (2008), think that our language faculty is just our general cognitive abilities applied to language.
– In linguistics and above all in philosophy of language many people think that sentences cannot have any meaning without a context, such a context involving both linguistic and extra-linguistic information. Thus, according to this view, the input of our algorithm should include context. Our answer is firstly that linguistic context is partly taken into account since we are able to produce, in addition to

formulas, discourse structures. Regarding the part of context that we cannot take into account, be it linguistic or not, our answer is that it is not part of *semantics*, but rather an aspect of *pragmatics*. And, as argued by Corblin (2013), if someone is given a few sentences on a sheet of paper without any further information, he starts imagining situations, may infer other statements from what he reads, and such thoughts can be called the semantics of the sentence.

– The linguistic tradition initiated by Montague (1974) lacks some coherence regarding computability. On the one hand, Montague gives an algorithm for parsing sentences and for computing their meaning as a logical formula (Montague 1974); Partee (2001) calls this the *indirect interpretation*. On the other hand, he asserts that the meaning of a sentence is the interpretation of the formula in possible worlds, the corresponding *direct interpretation* (Montague 1970; Partee 2001). According to the direct interpretation perspective, each intermediate step, including the intensional/modal formulas should be forgotten, and the semantics is defined as the set of possible worlds in which the semantic formula is true: but this cannot even be finitely described,[1] except by these intermediate formulas; a fortiori it cannot be computed. We place ourselves in the tradition of indirect interpretation, for at least three reasons, from the weakest to the strongest:

  – Models for higher order logic, as in Montague, are not as simple as is sometimes assumed, and they do not quite match the formulas: completeness fails.[2] This means that a model and even all models at once contains less information than the formula itself.
  – We do not want to be committed to any particular interpretation. Indeed, there are alternative relevant interpretations of formulas, as the following non exhaustive list shows: dialogical interpretations (that are the sets of proofs and/or refutations), game theoretic semantics and ludics (related to the former style of interpretation), set of consequences of the formula, structures inhabited by their normal proofs as in intuitionistic logic,…
  – Interpreting the formula(s) is no longer related to linguistics, although some interpretations might be useful for some applications. Indeed, once you have a formula, interpreting it in your favourite way is a purely logical question. Deciding whether it is true or not in a model, computing all its proofs or all its refutations, defining game strategies, computing its consequences or the corresponding structure has nothing to do with the particular natural language statement you started with.

[1] The models produced by Montague are uncountably infinite, since they start from countably infinite sets, then use the powerset operation (Montague 1970; Thomason 1974, p. 194).

[2] This is true already for second-order logic and its standard, set-theoretic models (van Dalen 2013; Shapiro 1991). As Shapiro (1991, p. vii) puts it succinctly: “there can be no complete effective deductive system for second-order logic”.

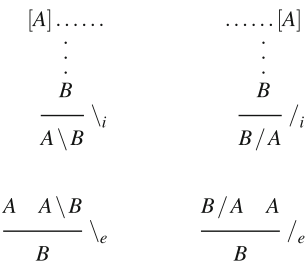

Fig. 1 Natural deduction proof rules for the Lambek calculus

2 Computational Semantics à la Montague

We shall first present the general algorithm that maps sentences to logical formulas, returning to lexical semantics in Sect. 3. The first step is to compute a syntactic analysis that is rich and detailed enough to enable the computation of the semantics (in the form of logical formulas). The second step is to incorporate the lexical lambda terms and to reduce the obtained lambda term—this step possibly includes the choice of some lambda terms from the lexicon that fix the type mismatches.

2.1 Categorial Syntax

In order to express the process that maps a sentence to its semantic interpretation(s) in the form of logical formulas, we shall start with a categorial grammar. This is not strictly necessary: Montague (1974) used a context free grammar (augmented with a mechanism for quantifier scope), but if one reads between the lines, at some points he converts the phrase structure into a categorial derivation, so we shall, following our earlier work (Moot and Retoré 2012), directly use a categorial analysis. Although richer variants of categorial grammars are possible, and used in practice, we give here an example with Lambek grammars, and briefly comment on variants later.

Categories are freely generated from a set of base categories, for example np (noun phrase), n (common noun), S (sentence), by two binary operators: $\backslash$ and $/$. $A \backslash B$ and B / A are categories whenever A and B are categories. A category $A \backslash B$ intuitively looks for a category A to its left in order to form a B. Similarly, a category B / A combines with an A to its right to form a B. The full natural deduction rules are shown in Fig. 1.

A lexicon provides, for each word w of the language, a finite set of categories $lex(w)$. We say a sequence of words $w_1, \ldots, w_n$ is of type C whenever for each i there exists $c_i \in lex(w_i)\, c_1, \ldots, c_n \vdash C$. Figure 2 shows an example lexicon (top) and a derivation of a sentence (bottom).

2.2 From Syntactic Derivation to Typed Linear Lambda Terms

Categorial derivations, being a proper subset of derivations in multiplicative intuitionistic linear logic, correspond to (simply typed) linear lambda terms. This makes

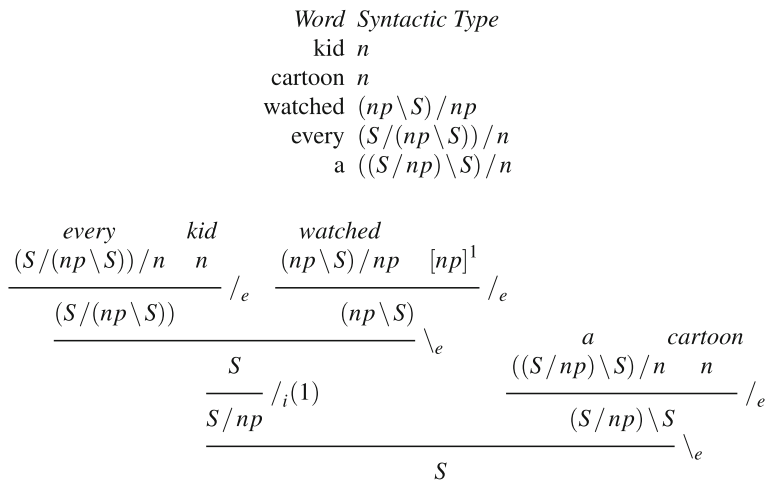

Fig. 2 Lexicon and example derivation

the connection to Montague grammar particularly transparent (van Benthem 1986; Moortgat 1997).

Denoting by e the set of entities (or individuals) and by t the type for propositions (these can be either true or false, hence the name t) one has the following mapping from syntactic categories to semantic/logical types.

(Syntactic type)* =	Semantic type
$S^* = t$	a sentence is a proposition
$np^* = e$	a noun phrase is an entity
$n^* = e \to t$	a noun is a subset of the set of entities (maps entities to propositions)
$(A\backslash B)^* = (B \,/\, A)^* = A^* \to B^*$	extends easily to all syntactic categories

Using this translation of categories into types which forgets the non-commutativity, the Lambek calculus proof of Fig. 2 is translated to the linear intuitionistic proof shown in Fig. 3; we have kept the order of the premisses unchanged to highlight the similarity with the previous proof. Such a proof can be viewed as a simply typed lambda term with the two base types e and t.

$$(a^{(e\to t)\to((e\to t)\to t)}\ cartoon^{e\to t})(\lambda y^{e}(every^{(e\to t)\to((e\to t)\to t)}\ kid^{e\to t})\ (watched^{e\to e\to t}\ y))$$

As observed by Church (1940), the simply typed lambda calculus with two types e and t is enough to express the formulas of higher order logic, provided one introduces *constants* for the logical connectives and quantifiers, that is, constants "$\exists$" and "$\forall$" of type $(e \to t) \to t$, and constants "$\wedge$", "$\vee$" and "$\Rightarrow$" of type $t \to (t \to t)$.

In addition to the syntactic lexicon, there is a semantic lexicon that maps any word to a simply typed lambda term with atomic types e and t and whose type is the translation

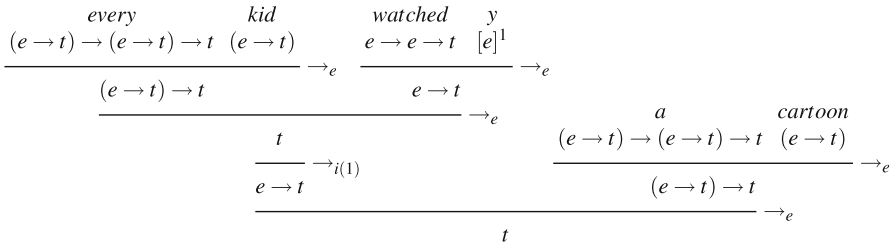

Fig. 3 The multiplicative linear logic proof corresponding to Fig. 2

word	***syntactic type*** u ***semantic type*** u^* ***semantics: λ-term of type*** u^*
every	$(S/(np\backslash S))/n$ (subject) $(e \to t) \to ((e \to t) \to t)$ $\lambda P^{e\to t}\ \lambda Q^{e\to t}\ (\forall^{(e\to t)\to t}\ (\lambda x^e(\Rightarrow^{t\to(t\to t)}\ (P\,x)(Q\,x))))$
a	$((S/np)\backslash S)/n$ (object) $(e \to t) \to ((e \to t) \to t)$ $\lambda P^{e\to t}\ \lambda Q^{e\to t}\ (\exists^{(e\to t)\to t}\ (\lambda x^e(\wedge^{t\to(t\to t)}(P\,x)(Q\,x))))$
kid	n $e \to t$ $\lambda x^e(\texttt{kid}^{e\to t}\ x)$
cartoon	n $e \to t$ $\lambda x^e(\texttt{cartoon}^{e\to t}\ x)$
watched	$(np\backslash S)/np$ $e \to (e \to t)$ $\lambda y^e\ \lambda x^e\ ((\texttt{watched}^{e\to(e\to t)}\ x)\ y)$

Fig. 4 Semantic lexicon for our example grammar

of its syntactic formula. Figure 4 presents such a lexicon for our current example. For example, the word "every" is assigned formula $(S/(np\backslash S))/n$. According to the translation function above, we know the corresponding semantic term must be of type $(e \to t) \to ((e \to t) \to t)$, as it is in Fig. 3. The term we assign in in the semantic lexicon is the following (both the type and the term are standard in a Montagovian setting).

$$\lambda P^{e\to t}\ \lambda Q^{e\to t}\ (\forall^{(e\to t)\to t}\ (\lambda x^e(\Rightarrow^{t\to(t\to t)}\ (P\,x)(Q\,x))))$$

Unlike the lambda terms computed for proof, the lexical entries in the semantic lexicon need not be linear: the lexical lambda term assigned to "every" shown above is not a linear lambda term since the single abstraction binds two occurrences of x.

Similarly, the syntactic type of "a", the formula $((S/np)\backslash S)/n$ has corresponding semantic type $(e \to t) \to ((e \to t) \to t)$ (though syntactically different, a subject and an object generalized quantifier have the same *semantic* type), and the following lexical meaning recipe.

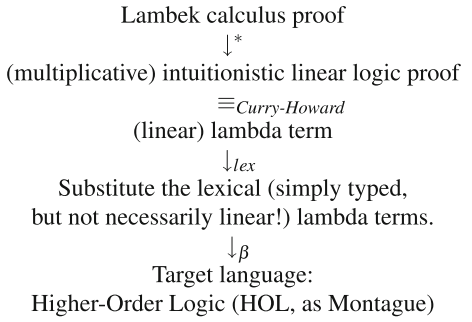

Fig. 5 The standard categorial grammar method for computing meaning

$$\lambda P^{e\to t}\ \lambda Q^{e\to t}\ (\exists^{(e\to t)\to t}\ (\lambda x^{e}(\wedge^{t\to(t\to t)}(P\ x)(Q\ x))))$$

Finally, "kid", "cartoon" and "watched" are assigned the constants $\texttt{kid}^{e\to t}$, $\texttt{cartoon}^{e\to t}$ and $\texttt{watched}^{e\to(e\to t)}$ respectively.

Because the types of these lambda terms are the same as those of the words in the initial lambda term, we can take the linear lambda term associated with the sentence and substitute the corresponding lexical meaning for each word, transforming the derivational semantics, in our case the following[3]

$$(a^{(e\to t)\to((e\to t)\to t)}\ cartoon^{e\to t})(\lambda y^{e}(every^{(e\to t)\to((e\to t)\to t)}\ kid^{e\to t})\ (watched^{e\to e\to t}\ y))$$

into an (unreduced) representation of the meaning of the sentence.

$$((\lambda P^{e\to t}\ \lambda Q^{e\to t}(\exists^{(e\to t)\to t}\ (\lambda x^{e}(\wedge^{t\to(t\to t)}(P\ x)(Q\ x)))))\ \texttt{cartoon}^{e\to t})$$
$$((\lambda y^{e}(((\lambda P^{e\to t}\ \lambda Q^{e\to t}\ (\forall^{(e\to t)\to t}\ (\lambda x^{e}(\Rightarrow^{t\to(t\to t)}\ (P\ x)(Q\ x)))))\ \texttt{kid}^{e\to t}\ x)))\ (\texttt{watched}^{e\to e\to t}\ y))$$

The above term reduces to

$$(\exists^{(e\to t)\to t}\ \lambda x^{e}(\wedge^{t\to(t\to t)}(\texttt{cartoon}\ x)(\forall^{(e\to t)\to t}\ (\lambda z^{e}(\Rightarrow^{t\to(t\to t)}\ (\texttt{kid}\ z)\ ((\texttt{watched}\ x)\ z))))))$$

that is:[4] $\exists x.\texttt{cartoon}(x) \wedge \forall z.\texttt{kid}(z) \Rightarrow \texttt{watched}(z, x)$

The full algorithm to compute the semantics of a sentence as a logical formula is shown in Fig. 5.

[3] There are *exactly* two (non-equivalent) proofs of this sentence. The second proof using the same premisses corresponds to the second, more prominent reading of the sentence whose lambda term is: $(every\ kid)(\lambda x^{e}.(a\ cartoon)(\lambda y^{e}((watched\ y)\ x))$.

[4] We use the standard convention to translate a term $((p\ y)\ x)$ into a predicate $p(x, y)$.

Lambek calculus proof
$\downarrow^{*}$
(multiplicative) intuitionistic linear logic proof
$\equiv_{Curry\text{-}Howard}$
(linear) lambda term
$\downarrow_{lex}$
Substitute the lexical ΛTy_n terms.
$\downarrow_{coercions}$
Solve type mismatches by the coercions provided by the lexicon.
$\downarrow_{\beta}$
Target language:
Higher-Order Logic (HOL, as Montague)

Fig. 6 Computing meaning in a framework with coercion

3 Adding Sorts, Coercions, and Uniform Operations

Montague (as Frege) only used a single type for entities: *e*. But it is much better to have many sorts in order to block the interpretation of some sentences:

(1) # The table barked.
(2) The dog barked.
(3) ?The sergeant barked.

As dictionaries say "*barked*" can be said of animals, usually dogs. The first one is correctly rejected: one gets $\texttt{bark}^{dog\to t}(\texttt{the table})^{artifact}$ and $dog \neq artifact$.

However we need to enable the last example $\texttt{bark}^{dog\to t}(\texttt{the sergeant})^{human}$ and in this case we use coercions (Bassac et al. 2010; Retoré 2014): the lexical entry for the verb "*barked*" which only applies to the sort of "*dogs*" provides a coercion $c : human \to dog$ from "*human*" to "*dog*". The revised lexicon provides each word with the lambda term that we saw earlier (typed using some of the several sorts/base types) and some optional lambda terms that can be used if needed to solve type mismatches.

Such coercions are needed to understand sentences like:

(4) This book is heavy.
(5) This book is interesting.
(6) This book is heavy and interesting.
(7) Washington borders the Potomac.
(8) Washington attacked Iraq.
(9) # Washington borders the Potomac and attacked Iraq.

The first two sentences will respectively use a coercion from book to physical object and a coercion from book to information. Any time an object has several related meanings, one can consider the conjunction of properties referring to those particular aspects. For these operations (and others acting uniformly on types) we exploit polymorphically typed lambda terms (system F). When the related meanings of a word are incompatible (this is usually the case) the corresponding coercions are declared to be incompatible in the lexicon (one is declared as rigid). This extended process is described in Fig. 6. Some remarks on our use of system F:

– We use it for the syntax of semantics (a.k.a. metalogic, glue logic)
– The formulas of semantics are the usual ones (many sorted as in Ty_n)
– We use polymorphic constants to model operations that act uniformly on types. Examples of this are quantifiers and the conjunction of predicates that apply to different facets of a given word.

Many other approaches to lexical semantics and coercions exist (Asher 2011; Luo 2012), but little is currently known about their formal complexity.

4 Complexity of the Syntax

As we remarked before, when computing the formal semantics of a sentence in the Montague tradition, we (at least implicitly) construct a categorial grammar proof (a syntactic analysis). Therefore, we need to study the complexity of parsing/theorem proving in categorial grammar first. The complexity generally studied in this context is the complexity of deciding about the existence of a proof (a parse) for a logical statement (a natural language sentence) as a function of the number of words in this sentence.[5]

The difference between theorem proving and parsing consists only in the inclusion of the lexicon, which may offer a choice of many different formulas for each word. Although lexical ambiguity is an important problem in practical applications, from the perspective of complexity analysis we can simply select formulas nondeterministically. This means that for logics in NP or PSPACE, the complexity of parsing and the complexity of theorem proving are the same.

Perhaps surprisingly, the simple product-free version of the Lambek calculus we have used for our examples is already NP-complete (Savateev 2009). However, there is a notion of order, which measures the level of "nesting" of the implications as defined below.

$$order(p) = 0$$
$$order(A/B) = order(B\backslash A) = \max(order(A), (order(B) + 1))$$

As an example, the order of formula $(np\backslash S)/np$ is 1, whereas the order of formula $S/(np\backslash S)$ is 2. For the Lambek calculus, the maximum order of the formulas in a grammar is a good indication of its complexity. Grammars used for linguistic purposes generally have formulas of order 3 or, at most, 4. We know that once we bound the order of formulas in the lexicon of our grammars to be less than a fixed n, parsing becomes polynomial for any choice of n, as shown by Pentus (2010).[6]

[5] For many algorithms, the complexity is a function of the number of atomic subformulas of the formulas in the sentence. Empirical estimation shows the number of atomic formulas is a bit over twice the number of words in a sentence (Moot 2015b).

[6] For the algorithm of Pentus (2010), the order appears as an exponent in the worst-case complexity: for a grammar of order n there is a multiplicative factor of $2^{5(n+1)}$. So though polynomial, this algorithm is not necessarily efficient.

The NP-completeness proof of Savateev (2009) uses a reduction from SAT, where a SAT problem with c clauses and v variables produces a Lambek grammar of order $3 + 4c$, with $(2c + 1)(3v + 1)$ atomic formulas.

The notion of order therefore provides a neat indicator of the complexity: the NP-completeness proof requires formulas of order 7 and greater (that is, the term $(3 + 4c)$ from the construction of Savateev (2009) with a minimum of one clause), whereas the formulas used for linguistic modelling are of order 4 or less.

Note that there is no contradition between the NP-completeness proof and the fixed-order polynomial parsing proof of Pentus (2010), since the latter result fixes the lexicon (that is, it is a fixed recognition result) whereas the former doesn't (it is a universal recognition result). In other words, there is no fixed Lambek calculus lexicon which generates all instances of SAT (unless P = NP).

Even though the Lambek calculus is a nice and simple system, we know that the Lambek calculus generates only context-free languages (Pentus 1995), and there is good evidence that at least some constructions in natural language require a slightly larger class of languages (Shieber 1985). One influential proposal for such a larger family of language classes are the mildly context-sensitive languages (Joshi 1985), which can be characterised in various ways. One typical way to do so is as follows.

- Contains the context-free languages,
- Limited cross-serial dependencies (i.e includes $a^n b^n c^n$ but maybe not $a^n b^n c^n d^n e^n$),
- Semilinearity (a language is semilinear iff there exists a regular language to which it is equivalent up to permutation),
- polynomial fixed recognition.[7]

There are various extensions of the Lambek calculus which generate mildly context-sensitive languages while keeping the syntax-semantics interface essentially the same as for the Lambek calculus. Currently, little is known about upper bounds of the classes of formal languages generated by these extensions of the Lambek calculus. Though Moot (2002) shows that multimodal categorial grammars generate exactly the context-sensitive languages, Buszkowski (1997) underlines the difficulty of adapting the result of Pentus (1995) to extensions of the Lambek calculus.[8]

Besides problems from the point of view of formal language theory, it should be noted that the goal we set out at the start of this paper was not just to generate the right string language but rather to generate the right string-meaning pairs. This poses additional problems. Extensions of the Lambek calculus have been used to give accounts of many phenomena on the syntax-semantics interface. Examples include sentences such as the following.

(10) John left before Mary did.
(11) John studies logic and Charles, phonetics.

[7] The last two items are sometimes stated as the weaker condition "constant growth" instead of semilinearity and the stronger condition of polynomial parsing instead of polynomial fixed recognition. Since all other properties are properties of formal languages, we prefer the formal language theoretic notion of polynomial fixed recognition.

[8] We can side-step the need for a Pentus-like proof by looking only at fragments of order 1 and some order 1 fragments are known to generate mildly context-sensitive languages (de Groote and Pogodalla 2004; Wijnholds 2011).

(12) John ate more donuts than Mary bought bagels.

Where sentence 10 has a reading equivalent to 'John left before Mary *left*' and sentence 11 is equivalent to 'John studies logic and Charles *studies* phonetics'. Phenomena like these and many others have been given an analysis in the literature on type-logical grammars (Morrill et al. 2011; Kubota and Levine 2012).

However, one of the oldest and most widely studied phenomena on the syntax-semantics interface is the scope of quantifiers. In the context of complexity, this problem is particularly interesting because the combinatorics of the complexity of quantifiers scope can be assigned a precise number for its worst-case complexity. For example, a sentence with n quantified noun phrases can have $n!$ readings (Hobbs and Shieber 1987; Barker 2015): one for each of the permutations of the quantifiers (some additional readings may be available when we include cumulative readings). Although the standard notion of complexity for categorial grammars is the complexity of deciding whether or not a proof exists, formal semanticists, at least since Montague (1974), want their formalisms to generate all and only the correct readings for a sentence: we are not only interested in whether or not a proof exists but, since different natural deduction proofs correspond to different readings, also in what the different proofs of a sentence are.[9]

When we look at the examples below

(13) Every representative of a company saw most samples.
(14) Some student will investigate two dialects of every language.

they have five possible readings (instead of $3! = 6$) (Hobbs and Shieber 1987; Park 1996; Blackburn and Bos 2005; Koller and Thater 2010). For example 13, a naive implementation of Cooper storage would produce (schematically) the following six readings (the unavailable reading is marked with $*$).

1. $\forall < \exists < \mathit{most}$
2. $* \forall < \mathit{most} < \exists$
3. $\exists < \forall < \mathit{most}$
4. $\exists < \mathit{most} < \forall$
5. $\mathit{most} < \forall < \exists$
6. $\mathit{most} < \exists < \forall$

However, reading 2, when fully spelled out, would look as follows.

$$\forall x.\texttt{representative\_of}(x, y) \Rightarrow \texttt{most}(z, \texttt{sample}(z)) \Rightarrow \exists y.\texttt{company}(y) \wedge \texttt{see}(x, z)$$

Since this expression has an unbound occurrence of y (the leftmost occurrence), it is does not correspond to a valid quantifier scope, and several authors have proposed

[9] Of course, when our goal is to generate (subsets of) $n!$ different proofs rather than a single proof (if one exists), then we are no longer in NP, though it is unknown whether an algorithm exists which produces a sort of shared representation for all such subsets such that (1) the algorithm outputs "no" when the sentence is ungrammatical (2) the algorithm has a fairly trivial algorithm (say of a low-degree polynomial at worst) for recovering all readings from the shared representation (3) the shared structure is polynomial in the size of the input.

more sophisticated algorithms generating exactly the five readings required (Hobbs and Shieber 1987; Koller and Thater 2010). The Lambek calculus analysis has trouble generating reading 4, where medial quantifier "a company" outscopes first "most samples" then "every company". We can, of course, remedy this by adding new, more complex types to the quantifier "a", but this would increase the order of the formulas and there is, in principle, no bound on the number of constructions where a medial quantifier has wide scope over a sentence. We can also save the Lambek calculus by claiming, as one of the anonymous referees of this paper does, that reading 4 does not exists (although this would not help for other cases where a medial quantifier has wide scope). The important point is the following: if we want to generate exactly five readings, a simple type-logical grammar can do so without any further stipulations.

The problems with quantifier scope are more serious than is generally assumed in the literature, where it is generally shown that certain readings can only be obtained by adding further lexical entries. We can show, by a simple counting argument that, no matter how many lexical entries we add, Lambek grammars cannot generate the $n!$ readings required for quantifier scope of an n-quantifier sentence: the number of readings for a Lambek calculus proof is proportional to the Catalan numbers (the enumeration of planar proof nets, where the number of planar axiom links is in 1–1 correspondence with a binary bracketing of the atomic formulas (Moot 2007; Stanley 2015), multiplied by a number of grammar-dependent constants) and this number is in $o(n!)$;[10] in other words, given a Lambek calculus grammar, the number of readings of a sentence with n quantifiers grows much faster than the number of Lambek calculus proofs for this sentence, hence the grammar fails to generate many of the required readings.

Since the eighties, many variants and extensions of the Lambek calculus have been proposed, each with the goal of overcoming the limitations of the Lambek calculus. Extensions/variations of the Lambek calculus—which include multimodal categorial grammars (Moortgat 1997), the Displacement calculus (Morrill et al. 2011) and first-order linear logic (Moot and Piazza 2001)—solve both the problems of formal language theory and the problems of the syntax-semantics interface. For example, there are several ways of implementing quantifiers yielding exactly the five desired readings for sentences 13 and 14 without appealing to extra-grammatical mechanisms. Carpenter (1994) gives many examples of the advantages of this logical approach to scope, notably its interaction with other semantic phenomena like negation and coordination.

Though these modern calculi solve the problems with the Lambek calculus, they do so without excessively increasing the computational complexity of the formalism: multimodal categorial grammars are PSPACE-complete (Moot 2002), whereas most other extensions are NP-complete, like the Lambek calculus.

[10] We need to be careful here: the number of readings for a sentence with n quantifiers is $\Theta(n!)$, whereas the maximum number of Lambek calculus proofs is $O(c_0^{c_2 n} C_{c_1 c_2 n})$, for constants c_0, c_1, c_2 which depend on the grammar (c_0 is the maximum number of formulas for a single word, c_1 is the maximum number of (negative) atomic subformulas for a single formula and c_2 represents the minimum number of words needed to add a generalized quantifier to a sentence, i.e. $c_2 n$ is the number of words required to produce an n-quantifier sentence) and $O(c_0^{c_2 n} C_{c_1 c_2 n})$ is in $o(n!)$.

Even the most basic categorial grammar account of quantifier scope requires formulas of order 2, while, in contrast to the Lambek calculus, the only known polynomial fragments of these logics are of order 1. Hence the known polynomial fragments have very limited appeal for semantics.[11]

Is the NP-completeness of our logics in conflict with the condition of polynomial fixed recognition required of mildly context-sensitive formalisms? Not necessarily, since our goals are different: we are not only interested in the string language generated by our formalism but also in the string-meaning mappings. Though authors have worked on using mildly context-sensitive formalisms for semantics, they generally use one of the two following strategies for quantifier scope: (1) an external mechanism for computing quantifier scope (e.g. Cooper storage or one of its variants (Cooper 1975; Hobbs and Shieber 1987)), or (2) an underspecification mechanism for representing quantifier scope (Fox and Lappin 2010). From the perspective of type-logical grammars, both strategies shift complexity away from the syntax to the syntax-semantics interface.

For case 1 (Cooper 1975), a single syntactic structure is converted into up to $n!$ semantic readings, whereas for case 2, though we represent all possible readings in a single structure, even deciding whether the given sentence has a semantic reading *at all* becomes NP-complete (Fox and Lappin 2010), hence we simply shift the NP-completeness from the syntax to the syntax-semantics interface.[12] Our current understanding therefore indicates that NP-complete is the best we can do when we want to generate the semantics for a sentence. We do not believe this to be a bad thing, since pragmatic and processing constraints rule out many of the complex readings and enumerating all readings of sentences like sentence 13 above (and more complicated examples) is a difficult task. There is a trade-off between the work done in the syntax and in the syntax-semantics interface, where the categorial grammar account incorporates more than the traditional mildly context-sensitive formalisms. It is rather easy to set up a categorial grammar parser in such a way that it produces underspecified representations in time proportional to n^2 (Moot 2007). However, given that such an underspecified representation need not have any associated semantics, such a system would not actually qualify as a parser. We believe, following Carpenter (1994) and Jacobson (2002), that giving an integrated account of the various aspects of the syntax-semantics interface, as we are doing here, is the most promising path.

Our grammatical formalisms are not merely theoretical tools, but also form the basis of several implementations (Morrill and Valentín 2015; Moot 2015a; Morrill 2019), with a rather extensive coverage of various semantic phenomena and their interactions, including quantification, gapping, ellipsis, coordination, comparative subdeletion, etc.

[11] The known polynomial fragments of order 1 are in LOGCFL (de Groote and Pogodalla 2004; Wijnholds 2011). A counting argument similar to the one we use for the Lambek calculus can be used to show there are not enough distinct LOGCFL trees to enumerate the different quantifier scope readings required (essentially because we only have a finite number of pointers to handle n quantifiers).

[12] In addition, Ebert (2005) argues that underspecification languages are not expressive enough to capture all possible readings of a sentence in a single structure. So underspecification does not solve the combinatorial problem but, at best, reduces it.

$$\frac{!\Gamma \vdash C}{!\Gamma \vdash !C}\ P \qquad \frac{\Gamma, A \vdash C}{\Gamma, !A \vdash C}\ D$$

$$\frac{\Gamma \vdash C}{\Gamma, !A \vdash C}\ W \qquad \frac{\Gamma, !A, !A \vdash C}{\Gamma, !A \vdash C}\ C$$

Fig. 7 The exponential rules for intuitionistic linear logic

$$\frac{\Gamma \vdash C}{!\Gamma \vdash !C}\ SP \qquad \frac{\Gamma, A^n \vdash C}{\Gamma, !A \vdash C}\ M$$

Fig. 8 The exponential rules for soft linear logic

5 Complexity of Semantics

The complexity of the syntax discussed in the previous section only considered the complexity of computing unreduced lambda terms as the meaning of a sentence. Even in the standard, simply typed Montagovian framework, normalizing lambda terms is known to be of non-elementary complexity (Schwichtenberg 1982), essentially due to the possibility of recursive copying. In spite of this forbidding worst-time complexity, normalization does not seem to be a bottleneck in the computation of meaning for practical applications (Bos et al. 2004; Moot 2010).

Is there a deeper reason for this? We believe that natural language semantics uses a restricted fragment of the lambda calculus, soft lambda calculus, which we will introduce below.

5.1 Soft Linear Logic

Soft linear logic is a logic which restricts recursive copying. Cut elimination/normalization for soft linear logic has been shown to characterize the complexity class P exactly (Lafont 2004). The soft lambda calculus (Baillot and Mogbil 2004) are the lambda terms assigned to soft linear logic proofs. Therefore, soft linear logic proofs (and soft lambda terms) normalize in polynomial time.

We claim that soft linear logic is expressive enough for all of natural language semantics; that is, all lambda term meanings used in formal semantics are expressible in the soft lambda calculus. To make this claim more precise, we first introduce soft linear logic and the corresponding lambda term assignments.

Figures 7 and 8 contrast the standard exponential rules of intuitionistic linear logic with those of soft linear logic. In linear logic, a formula $!A$ is essentially a formula which can be copied any number of times. Intuitively, $!A$ indicates an unlimited amount of A formulas. The contraction rule $[C]$ in Fig. 7, read from conclusion to premiss, makes a copy of a $!A$ formula, whereas the weakening rule $[W]$ erases a $!A$ formula. As a consequence, the structural rules of contraction and weakening, which apply globally in intuitionistic logic, apply only to formulas of the form $!A$ in linear logic. The promotion and dereliction rule in linear logic are simply the sequent rules for $\Box$ in the modal logic S4. Using the dereliction rule $[D]$ allows us to use a $!A$ formula like a 'normal' A formula. Finally, the promotion rule $[P]$ states that we can derive

$$\frac{}{x : A \vdash x : A}\ Ax \qquad \frac{\Delta \vdash N : A \quad \Gamma, x : A \vdash M : C}{\Gamma, \Delta \vdash M[x := N] : C}\ Cut$$

$$\frac{\Delta \vdash N : A \quad \Gamma, y : B \vdash M : C}{\Gamma, \Delta, x : A \multimap B \vdash M[y := (xN)] : C}\ \multimap L \qquad \frac{\Gamma, x : A \vdash B}{\Gamma \vdash \lambda x.M : A \multimap B}\ \multimap R$$

$$\frac{\Gamma \vdash M : C}{!\Gamma \vdash M : !C}\ SP \qquad \frac{\Gamma, x_i : A^n \vdash M : C}{\Gamma, y : !A \vdash M[x_1 := y, \ldots, x_n := y] : C}\ M$$

Fig. 9 Term-labeled rules for soft linear logic

a formula to be a $!C$ when all context formulas can also be copied as many times as necessary (the notation $!\Gamma$ means all formulas in the antecedent Γ have ! as their main connective, that is, Γ is of the form $!A_1, \ldots, !A_n$).

In soft linear logic, the promotion rule $[P]$ has been replaced by the *soft promotion* rule $[SP]$. The soft promotion rule simultaneously adds a '!' connective to all antecedent formulas and to the succedent formula. As a consequence, we can no longer derive $!A \vdash !!A$.

The weakening $[W]$, contraction $[C]$, and dereliction $[D]$ rules have been replaced by the *multiplexing* rule $[M]$. The notation A^n in the premiss of the rule indicates a sequence of n occurrences of the A formula. It allows us to replace a formula $!A$ by any number of A formulas. The special case $n = 0$ corresponds to weakening and the special case $n = 1$ corresponds to dereliction. Unlike the contraction rule, the multiplexing rule doesn't allow us to make a copy of $!A$ itself. Therefore, in soft linear logic, the principle $!A \vdash !A \otimes !A$ is no longer derivable, although $!A \vdash A \otimes A$ is.

Figure 9 gives a version of soft linear logic labeled with lambda terms, a slightly simplified version of the calculus used in Baillot and Mogbil (2004) (we have simply replaced the *let* statements by explicit substitutions). In the term-labeled system $x_i : A^n$ is shorthand for $x_1 : A, \ldots x_n : A$, that is, n occurrences of A each with a distinct variable x_i.

5.2 Formal Semantics and Soft Linear Logic

If the lambda terms used in formal semantics are in the soft lambda calculus, this would explain why even naive implementations of normalization perform well in practice.

The question of whether soft linear logic suffices for our semantic parser may appear hard to answer, however, it is an obvious (although tedious) result for any explicit grammar. To show that all the semantic lambda terms can be typed in soft linear logic, we only need to verify that every lambda term in the lexicon is soft. There is a finite number of words, with only a finite number of lambda terms per word. Furthermore, words from open classes (nouns, verbs, adjectifs, manner adverbs,…in which speakers may introduce new words…about 200,000 inflected word forms) are the most numerous and all have soft and often even linear lambda terms. Thus only closed class words (grammatical words such as pronouns, conjunctions, auxiliary verbs,…and some complex adverbs, such as "too") may potentially need a non-soft semantic lambda term: there are less than 500 such words, so it is just a matter of

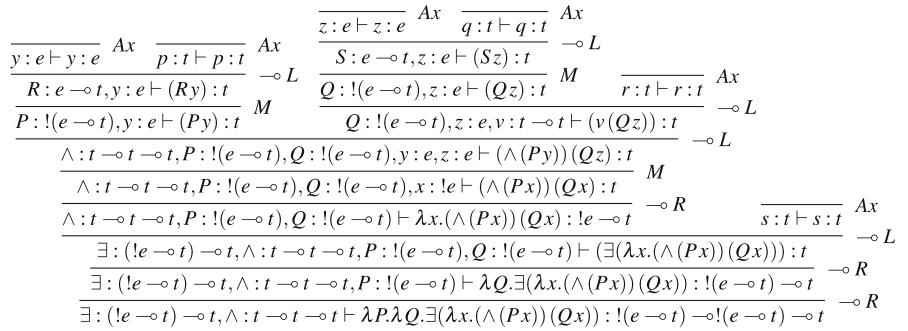

Fig. 10 Soft linear logic proof of the lambda term for "a" from Fig. 4

patience to prove they all have soft lambda terms. Of course, finding deep reasons (cognitive, linguistic) for semantic lambda terms to be soft in *any* language would be much more difficult (and much more interesting!).

As a concrete example, we can return to the semantics of a quantifier like "a". In Fig. 4, we assigned it type $(e \to t) \to ((e \to t) \to t)$. We now show that for such quantifiers, we can instead use the soft linear logic formula $!(e \multimap t) \multimap (!(e \multimap t) \multimap t)$ and use $t \multimap t \multimap t$ for the constant $\wedge$ and $(!e \multimap t) \multimap t$ for the constant $\exists$. Figure 10 then shows how the lambda term assigned to "a" is a soft lambda term.

Given that we only need a single copy of the P and Q variable (the two topmost multiplexing rules replace P and Q by a single variable of type $e \multimap t$), we could have chosen to assign the simpler formula $(e \multimap t) \multimap ((e \multimap t) \multimap t)$ to this entry and this would produce a proof requiring the multiplexing rule only once for the x formula. However, the current formula easily extends to more complex quantifiers such as "exactly two/three/…", "at least two/three/…" with each successive numeral requiring more copies of P, Q and x, as well as a constants $=$ and $\neq$ of type $e \to (e \to t)$ (technically, we want to allow multiple occurrences of the logical constants in a term and therefore should assign $=$ and $\neq$ the type $!(e \to (e \to t))$, and similarly for the other logical constants).

The reader may wonder how to decide whether a simply typed lambda term can be assigned a formula in soft linear logic. A simple solution would be to use the standard translation of intuitionistic logic into linear logic and translate intuitionistic implication as follows.

$$(A \to B)^* = !A^* \multimap B^* \tag{1}$$

We can also distinguish between positive and negative occurrences and add the "!" connective only to negative occurrences as follows, then translating constants using the negative translation and translating the lexical lambda terms using the positive translation.

$$(A \to B)^+ = !A^- \multimap B^+ \tag{2}$$

$$(A \to B)^- = A^+ \multimap B^- \tag{3}$$

This translation would produce the given types for "∃", "∧" (although we would need to add "!" as the outer connective for any connective occurring multiple times) and the type assigned to the entry for "a" itself in Fig. 10. We can then use the given lambda term to remove all non-determinism from proof search, for example by using the following algorithm for finding a soft linear logic proof of $\Gamma \vdash M : C$. We assume, without loss of generality, that M is a lambda term in long normal form. Γ initially contains the constants occurring in M and, when α is the type of M, $C = \alpha^+$, obtained using Eqs. (2) and (3) above, transforming the simple type α to the soft linear logic formula C.

1. If M is an atomic term t (a variable or a constant) then we must be in the case $t : A \vdash t : A$, and we obtain a proof using the axiom rule, or $t : !A \vdash t : A$, and we obtain a proof using first the multiplexor to create a single copy of A then use the axiom rule; if the current sequent has any other form, we fail.
2. If M is of the form $\lambda x.N$ then we are looking for a proof of $\Gamma \vdash \lambda x.N : A \multimap B$, we apply the $\multimap R$ rule and continue proof search for a proof of $\Gamma, x : A \vdash N : B$.
3. Otherwise, M must be of the form $((t\ N_1) \ldots N_n)$ with t the head term (a variable or a constant), now for t and for each free variable and constant in N_1 which is also free in $N_2, \ldots, N_n$, we count the number of occurrences of these variables and constants in M and make that many copies for each (e.g. if x occurs three times in the full term, then we replace $x : !B$ in the antecedent by $x_1 : B, x_2 : B, x_3 : B$ and the three occurrence of x in M by x_1, x_2 and x_3 respectively; this will fail if x occurs multiple times in M but is not assigned a formula $!B$). We then apply the multiplexor once for each constant and free variable followed by the $\multimap E$ rule, using the formula of the head term as the main formula and obtaining Γ from the free variables/constants of N_1 and Δ from the free variables of $N_2 \ldots N_n$, schematically as follows.

$$\frac{\Gamma \vdash N_1 : A \quad \Delta, z : B \vdash ((z\ N_2) \ldots N_n) : C}{\Gamma, \Delta, t : A \multimap B \vdash ((t\ N_1) \ldots N_n) : C} \multimap L$$

Given that multiple occurrences of the same free variable have been replaced by distinct occurrences using the multiplexor rule, this separation into Γ and Δ is unique. The term assigned to both subproofs is strictly smaller than the initial term. We then recursively continue the translation for the two subproofs.

Figure 10 gives an example of a proof obtained using this algorithm. We comment briefly on the non-trivial applications of step 3: for the rules with conclusion $(P\ y)$ and $(Q\ z)$, the head terms P and Q are assigned a formula $!(e \multimap t)$, so we must make a single copy before applying the $\multimap L$ rule, whereas for the rule with conclusion $(\wedge\ (P\ x))\ (Q\ x) : t$, x occurs once inside of $N_1 = (P\ x)$ and once outside, therefor we need to replace it with two distinct copies before applying the $\multimap L$ rule.

As a rough estimate, even a naive implementation of this proof search algorithm will have an $O(n^3)$ worst case complexity: a naive implementation of case 3 uses traverses the term M (of size n) once for each free variable (at most n), giving an $O(n^2)$ bound for each step, with an upper limit of n for the number of steps for a total of $O(n^3)$.

Of course, it is possible that the simple typed lambda term has a proof in soft linear logic while being outside of the fragment produced by the translation of Eqs. (2) and (3), for example because it requires a subterm of the form $!!A$. Other terms cannot be assigned a proof in soft linear logic at all, for example terms requiring an exponential number of reductions since this contradicts the polynomiality of soft linear logic. When the algorithm above fails to find a proof, careful analysis is needed to find out in which of these cases we are. A more sophisticated algorithm can be found, for example one exploiting the soft promotion rule $[SP]$. We leave these questions to future research. However, we conjecture that the algorithm given above to be powerful enough for the lambda terms found in formal semantics.

When adding coercions, as in Sect. 3, the process becomes a bit more complicated. However, the system of Lafont (2004) includes the logical rules for second-order quantifiers, hence reduction stays polynomial once coercions have been chosen. Their choice (as the choice of the syntactic category) increases complexity: when there is a type mismatch $g^{A\to X}u^{B}$ one needs to chose one of the coercions of type $B \to A$ provided by the entries of the words in the analysed phrase, with the requirement that when a rigid coercion is used, all other coercions provided by the same word are blocked (hence rigid coercions, as opposed to flexible coercions decrease the number of choices for other type mismatches).

Finally, having computed a set of formulas in higher-order logic corresponding to the meaning of a sentence, though of independent interest for formal semanticists, is only a step towards using these meaning representations for concrete applications. Typical applications such as question answering, automatic summarization, etc. require world knowledge and common sense reasoning but also a method for deciding about entailment: that is, given a set of sentences, can we conclude that another sentence is true. This question is of course undecidable, already in the first-order case. However, some recent research shows that even higher-order logic formulas of the type produced by our analysis can form the basis of effective reasoning mechanisms (Chatzikyriakidis and Luo 2014; Mineshima et al. 2015) and we leave it as an interesting open question to what extent such reasoning can be applied to natural language processing tasks.

Summarizing, the complexity of computing a semantic formula for a grammatical string is dominated by the complexity of finding a proof in the type-logical grammar framework used, provided the lexical lambda terms in our lexicon can be typed using soft linear logic. This means the total complexity of this task is, depending on the logic used, either NP-complete (in the case of the Lambek calculus and of the Displacement calculus) or PSPACE-complete (in the case of multimodal categorial grammar).

6 Conclusion

It is somewhat surprising that, in contrast to the well-developed theory of the algorithmic complexity of parsing, little is known about semantic analysis, even though computational semantics is an active field, as the recurring conferences with the same title as well as the number of natural language processing applications show. In this paper we simply presented remarks on the computability and on the complexity of this process. The good news is that semantics (at least defined as a set of logical formulas)

is *computable*. This was known, but only implicitly: Montague gave a set of instructions to compute the formula (and to interpret it in a model), but he never showed that, when computing such logical formula(s):

- The process he defined stops with a normal lambda terms of type proposition (t),
- Eta-long normal lambda terms with constants being either logical connectives or constants of a first (or higher order) logical language are in bijective correspondence with formulas of this logical language (this is more or less clear in the work of Church (1940) on simple type theory).
- The complexity of the whole process has a known complexity class, in particular the beta-reduction steps which was only discovered years after his death (Schwichtenberg 1982).

A point that we did not discuss is that we considered the *worst case complexity* of producing a logical formula, viewed as a function from the *number of words in a sentence*. Both aspects of our point of view can be challenged: in practice, grammar size is at least as importance as sentence length and average case complexity, empirically determined over a large corpus of sentences, may be a more appropriate measure of real-world performance than worst case complexity. Though the high worst case complexity shows that computing the semantics of a sentence is not always efficient, we nevertheless believe, confirmed by actual practice, that statistical models of a syntactic or semantic domain improve efficiency considerably, by providing extra information (as a useful though fallible "oracle") for many of the difficult choices. Indeed, human communication and understanding are very effective in general, but, from time to time, we misunderstand each other or need to ask for clarifications. For computers, the situation is almost identical: most sentences are analysed quickly, while some require more time or even defeat the software. Even though it is quite difficult to obtain the actual probability distribution on sentence-meaning pairs, we can simply estimate such statistics empirically by randomly selecting manually annotated examples from a corpus.

The other aspect, the sentence length, is, as opposed to what is commonly assumed in complexity theory, not a very satisfactory empirical measure of performance: indeed the average number of words per sentence is around 10 in spoken language and around 25 in written language. Sentences with more than 100 words are very rare.[13] Furthermore, lengthy sentences tend to have a simple structure, because otherwise they would quickly become incomprehensible (and hard to produce as well). Experience with parsing shows that in many cases, the grammar size is at least as important as sentence length for the empirical complexity of parsing algorithms (Joshi 1997; Sarkar 2000; Gómez-Rodríguez et al. 2006). Grammar size, though only a constant factor in the complexity, tends to be a *big* constant for realistic grammars: grammars with between 10,000 and 20,000 rules are common.

We believe that the complexity of computing the semantics and of reasoning with the semantic representations are some of the most important reasons that the Turing test is presently out of reach. However we have also shown that, for many interesting

[13] To give an indication, the TLGbank (Moot 2015b) contains more than 14,000 French sentences and has a median of 26 words per sentence, 99% of sentences having less than 80 words, with outliers at 190 and at 266 (the maximum sentence length in the corpus).

type-logical grammars, computing a logical formula representing the meaning of a sentence is a problem which is NP-complete.

References

Asher, N. (2011). *Lexical meaning in context: A web of words*. Cambridge: Cambridge University Press.

Baillot, P., & Mogbil, V. (2004). Soft lambda-calculus: A language for polynomial time computation. In *Foundations of software science and computation structures* (pp. 27–41). Springer.

Barker, C. (2015). Scope. In S. Lapping & C. Fox (Eds.), *Handbook of contemporary semantic theory* (2nd ed., pp. 40–76). Hoboken: Wiley Blackwell.

Bassac, C., Mery, B., & Retoré, C. (2010). Towards a type-theoretical account of lexical semantics. *Journal of Logic, Language and Information*, *19*(2), 229–245. https://doi.org/10.1007/s10849-009-9113-x.

Blackburn, P., & Bos, J. (2005). *Representation and inference for natural language: A first course in computational semantics*. Stanford: CSLI.

Bos, J., Clark, S., Steedman, M., Curran, J. R., & Hockenmaier, J. (2004). Wide-coverage semantic representation from a CCG parser. In *Proceedings of COLING-2004* (pp. 1240–1246).

Buszkowski, W. (1997). Mathematical linguistics and proof theory. In J. van Benthem & A. ter Meulen (Eds.), *Handbook of logic and language, chap. 12* (pp. 683–736). Amsterdam: Elsevier.

Carpenter, B. (1994). Quantification and scoping: A deductive account. In *The proceedings of the 13th west coast conference on formal linguistics*.

Chatzikyriakidis, S., & Luo, Z. (2014). Natural language inference in Coq. *Journal of Logic, Language and Information*, *23*(4), 441–480.

Church, A. (1940). A formulation of the simple theory of types. *Journal of Symbolic Logic*, *5*(2), 56–68.

Cooper, R. (1975). Montague's semantic theory and transformational grammar. Ph.D. thesis, University of Massachusetts.

Corblin, F. (2013). *Cours de sémantique: Introduction*. Paris: Armand Colin.

de Groote, P. D., & Pogodalla, S. (2004). On the expressive power of abstract categorial grammars: Representing context-free formalisms. *Journal of Logic, Language, and Information*, *13*(4), 421–438.

Ebert, C. (2005). Formal investigations of underspecified representations. Ph.D. thesis, King's College, University of London.

Fox, C., & Lappin, S. (2010). Expressiveness and complexity in underspecified semantics. *Linguistic Analysis*, *36*(1–4), 385–417.

Gómez-Rodríguez, C., Alonso, M. A., & Vilares, M. (2006). On the theoretical and practical complexity of TAG parsers. In *Proceedings of formal grammar (FG 2006)* (pp. 87–101).

Hobbs, J. R., & Shieber, S. M. (1987). An algorithm for generating quantifier scopings. *Computational Linguistics*, *13*(1–2), 47–63.

Jacobson, P. (2002). The (dis)organization of the grammar: 25 years. *Linguistics and Philosophy*, *25*(5–6), 601–626.

Joshi, A. (1985). Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions? In D. R. Dowty, L. Karttunen & A. Zwicky (Eds.), *Natural language parsing* (pp. 206–250). Cambridge: Cambridge University Press.

Joshi, A. (1997). Parsing techniques. In R. A. Cole, J. Mariani, H. Uszkoreit, A. Zaenen, & V. Zue (Eds.), *Survey of the state of the art in human language technology, chap. 11.4* (pp. 351–356). Cambridge University Press and Giardini.

Koller, A., & Thater, S. (2010). Computing weakest readings. In *Proceedings of the 48th annual meeting of the association for computational linguistics* (pp. 30–39).

Kubota, Y., & Levine, R. (2012). Gapping as like-category coordination. In D. Béchet & A. Dikovsky (Eds.), *Logical aspects of computational linguistics. Lecture notes in computer science* (Vol. 7351, pp. 135–150). Nantes: Springer.

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

Langacker, R. (2008). *Cognitive grammar: A basic introduction*. Oxford: Oxford University Press.

Luo, Z. (2012). Formal semantics in modern type theories with coercive subtyping. *Linguistics and Philosophy*, *35*(6), 491–513.

Mineshima, K., Martınez-Gómez, P., Miyao, Y., & Bekki, D. (2015). Higher-order logical inference with compositional semantics. In *Proceedings of EMNLP* (pp. 2055–2061).

Montague, R. (1970). English as a formal language. In B. Visentini (Ed.), *Linguaggi nella societa e nella tecnica* (pp. 188–221). Edizioni di Communita.
Montague, R. (1974). The proper treatment of quantification in ordinary English. In R. Thomason (Ed.), *Formal philosophy: Selected papers of Richard Montague*. New Haven: Yale University Press.
Moortgat, M. (1997). Categorial type logics. In J. van Benthem & A. ter Meulen (Eds.), *Handbook of logic and language, chap. 2* (pp. 93–177). Amsterdam: Elsevier.
Moot, R. (2002). Proof nets for linguistic analysis. Ph.D. thesis, Utrecht Institute of Linguistics OTS, Utrecht University.
Moot, R. (2007). Filtering axiom links for proof nets. In L. Kallmeyer, P. Monachesi, G. Penn, & G. Satta (Eds.), *Proccedings of formal grammar 2007*.
Moot, R. (2010). Wide-coverage French syntax and semantics using Grail. In *Proceedings of traitement automatique des langues naturelles (TALN)*, Montreal. System Demo
Moot, R. (2015a). Linear one: A theorem prover for first-order linear logic. https://github.com/RichardMoot/LinearOne. Accessed 16 Apr 2019.
Moot, R. (2015b). A type-logical treebank for french. *Journal of Language Modelling*, *3*(1), 229–264.
Moot, R., & Piazza, M. (2001). Linguistic applications of first order multiplicative linear logic. *Journal of Logic, Language and Information*, *10*(2), 211–232.
Moot, R., & Retoré, C. (2012). *The logic of categorial grammars: A deductive account of natural language syntax and semantics*. Berlin: Springer.
Morrill, G. (2019). Parsing/theorem-proving for logical grammar *CatLog3*. *Journal of Logic, Language and Information*. https://doi.org/10.1007/s10849-018-09277-w.
Morrill, G., & Valentín, O. (2015). Computational coverage of TLG: The Montague test. In *Proceedings CSSP 2015 Le onzième Colloque de Syntaxe et Sémantique à Paris* (pp. 63–68).
Morrill, G., Valentín, O., & Fadda, M. (2011). The displacement calculus. *Journal of Logic, Language and Information*, *20*(1), 1–48.
Park, J. C. (1996). Quantifier scope, lexical semantics, and surface structure constituency. Technical report, University of Pennsylvania.
Partee, B. (2001). Montague grammar. In N. J. Smelser & P. B. Baltes (Eds.), *International encyclopedia of the social and behavioral sciences*. Oxford: Pergamon.
Pentus, M. (1995). Lambek grammars are context free. In *Proceedings of logic in computer science* (pp. 429–433).
Pentus, M. (2010). A polynomial-time algorithm for Lambek grammars of bounded order. *Linguistic Analysis*, *36*(1–4), 441–471.
Pinker, S. (1994). *The language instinct*. Penguin Science.
Retoré, C. (2014). The Montagovian generative lexicon $\Lambda T y_n$: A type theoretical framework for natural language semantics. In *Proceedings of TYPES* (pp. 202–229). 10.4230/LIPIcs.TYPES.2013.202.
Sarkar, A. (2000). Practical experiments in parsing using tree adjoining grammars. In *Proceeding of TAG+5*.
Savateev, Y. (2009). Product-free Lambek calculus is NP-complete. In *Symposium on logical foundations of computer science (LFCS)* (pp. 380–394).
Schwichtenberg, H. (1982). Complexity of normalization in the pure typed lambda-calculus. In *The L. E. J. Brouwer centenary symposium* (pp. 453–457). North-Holland.
Shapiro, S. (1991). *Foundations without foundationalism: A case for second-order logic*. Oxford: Clarendon Press.
Shieber, S. (1985). Evidence against the context-freeness of natural language. *Linguistics & Philosophy*, *8*, 333–343.
Stanley, R. P. (2015). *Catalan numbers*. Cambridge: Cambridge University Press.
Thomason, R. (Ed.). (1974). *Formal philosophy: Selected papers of Richard Montague*. New Haven: Yale University Press.
Turing, A. (1950). Computing machinery and intelligence. *Mind*, *49*, 433–460.
van Benthem, J. (1986). Categorial grammar. In *Essays in logical semantics, chap. 7* (pp. 123–150). Dordrecht: Reidel.
van Dalen, D. (2013). *Logic and structure* (5th ed.). Berlin: Springer.
Wijnholds, G. (2011). Investigations into categorial grammar: Symmetric pregroup grammar and displacement calculus. Master's thesis, Utrecht University.