ccg2lambda: A Compositional Semantics System

Pascual Martínez-Gómez[4]
pascual.mg@aist.go.jp

Koji Mineshima[1]
mineshima.koji@ocha.ac.jp

Yusuke Miyao[2]
yusuke@nii.ac.jp

Daisuke Bekki[1,2]
bekki@is.ocha.ac.jp

[1]Ochanomizu University
Tokyo, Japan

[2]NII
Tokyo, Japan

[4]AIRC, AIST
Tokyo, Japan

Abstract

We demonstrate a simple and easy-to-use system to produce logical semantic representations of sentences. Our software operates by composing semantic formulas bottom-up given a CCG parse tree. It uses flexible semantic templates to specify semantic patterns. Templates for English and Japanese accompany our software, and they are easy to understand, use and extend to cover other linguistic phenomena or languages. We also provide scripts to use our semantic representations in a textual entailment task, and a visualization tool to display semantically augmented CCG trees in HTML.

1 Introduction

We are motivated by NLP problems that benefit from any degree of computer language understanding or semantic parsing. Two prominent examples are Textual Entailment and Question-Answering, where the most successful approaches (Abzianidze, 2015; Berant et al., 2013) require symbolic representations of the semantics of sentences. We are inspired by the theoretical developments in the formal semantics literature, where higher-order logical (HOL) formulas are used to derive meaning representations (MR); despite what is typically believed in the NLP community, Mineshima et al. (2015) demonstrated that HOL can be used effectively at a reasonable speed.

In this paper, we describe `ccg2lambda`, our system to obtain MRs given derivations (trees) of a Combinatory Categorial Grammar (CCG) (Steedman, 2000). In order to obtain the MRs, our system is guided by the combinatory characteristics of CCG derivations and a list of manually designed semantic templates. The linguistic intuitions behind the design of those semantic templates and the evaluation of the MRs that they produce is detailed in Mineshima et al. (2015), and is not repeated here. In that paper, we tackled a Textual Entailment task, where the meanings of premises and conclusions were represented symbolically, and their entailment relation was judged with a theorem prover of higher-order logics. With this system, we obtained a state-of-the-art performance on the FraCaS dataset (Cooper et al., 1994).

`ccg2lambda` and the accompanying semantic templates are open source[1]. Semantic templates are already available for English and Japanese, and they are easily extensible to other linguistic phenomena and other languages for which CCG parsers are available. Here we describe how to use `ccg2lambda` and how to specify semantic templates for other researchers to extend our work.

2 Related Work

The most similar system to ours is Boxer (Bos et al., 2004), which outputs first order formulas given CCG trees. Our system can additionally produce higher-order formulas, which are more expressive and potentially accurate (Mineshima et al., 2015).

There are three prominent textbook systems for computational semantics, that of Bird et al. (2009), Blackburn and Bos (2005) and van Eijck and Unger (2010). These three systems, together with the Lambda Calculator[2] (Champollion et al., 2007) are excellent educational resources that are very accessible to beginner linguists in general, and semanticists in particular. The development of `ccg2lambda` is inspired by these systems, in that we aimed to produce a software that is easy to understand, use and extend with only basic knowledge of formal semantics and lambda calcu-

[1] `https://github.com/mynlp/ccg2lambda`
[2] `http://lambdacalculator.com/`

Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics—System Demonstrations, pages 85–90,
Berlin, Germany, August 7-12, 2016.

lus. However, these systems are mainly developed for educational purposes and are not connected to fully fledged parsers, hence not immediately usable as a component of larger NLP systems.

We have developed `ccg2lambda` to process trees that are produced by wide-coverage CCG parsers (e.g. C&C and Jigg[3]). Other semantic parsers such as those developed by Bos et al. (2004), Abzianidze (2015) and Lewis and Steedman (2013) also connect to wide-coverage CCG parsers, but they do not emphasize easy accessibility or extensibility. NL2KR (Vo et al., 2015) is an interactive system with powerful generalization capabilities, but it does not allow fine-grained lexicon specifications (only CCG categories) and does not output machine readable semantics. Instead, `ccg2lambda` produces XML machine-readable MRs, which make our system easy to integrate in larger logic or statistical NLP systems.

3 System Overview

Although our main system contribution is a semantic parser, we use the problem of textual entailment as an end-to-end task. Figure 1 schematically shows the several components of our system.

The first stage is to *parse sentences into CCG trees* (see Figure 2 for an example). Our system currently supports the C&C parser (Clark and Curran, 2004) for English, and Jigg (Noji and Miyao, 2016) for Japanese.

The second stage is the *semantic composition*, where MRs are constructed compositionally over CCG trees using lambda calculus, thus allowing higher-order logics if necessary. To this end, our system is guided by the compositional rules of the CCG tree and the semantic templates provided by the user. In Section 4 we describe in detail how these semantic templates are specified and how they control the semantic outputs. The output of this stage is a Stanford CoreNLP-style XML file (Manning et al., 2014) where each sentence has three XML nodes: `<tokens>`, `<ccg>` and `<semantics>`. Thus, sentence semantics can simply be read off the root node of the CCG tree.

In the case of recognizing textual entailment, the third stage is the *theorem construction*, definition of predicate types, and execution with a logic prover. This stage is not essential to our system, but it is added to this paper to show the usefulness of our semantic representations in an NLP task.

[3] `https://github.com/mynlp/jigg`

4 Semantic Composition

`ccg2lambda` receives CCG trees and outputs (possibly higher-order) logic formulas. To that end, we use i) the combinatory characteristics of CCG trees to guide the semantic compositions, and ii) a list of semantic templates to assign a precise meaning to CCG constituents.

See Figure 2 for an example of CCG derivation for the sentence "Some woman ordered tea", augmented with its semantics. Nodes have CCG syntactic categories (e.g. N or $S\backslash NP$), which is what our system receives as input. On the same figure, we have added the logical semantic representations (e.g. $\lambda x.\mathsf{woman}(x)$) below the syntactic categories. Our system outputs these logical formulas. For clarity, leaves also display the token base forms. The symbols $<, >$ and lex stand for left and right function application rules, and the type-shift rule in C&C, respectively. These rules and the syntactic categories guide the semantic composition, provided with semantic templates that describe the specific semantics.

4.1 Semantic templates

Semantic templates are defined declaratively in a YAML[4] file, typically by a formal semanticist after an appropriate linguistic analysis. A template applies to a node of the CCG derivation tree if certain conditions are met. Each template has two required attributes: `semantics` and (syntactic) `category`. The attribute `semantics` is a lambda term in NLTK semantics format (Garrette and Klein, 2009). In case a template applies on a CCG leaf (that is, a word), the lambda term in the template is applied on the base form of the word, and β-reduction is performed. For example, the semantic template

– `semantics` : $\backslash E.\backslash x.E(x)$

`category` : N

applying on a leaf whose base word is "woman" and its syntactic category is N, would produce the expression $(\lambda E.\lambda x.E(x))(\mathsf{woman})$ which is β-reduced to $\lambda x.\mathsf{woman}(x)$. Here, the base form "woman" substitutes all occurrences of the variable E in the `semantics` expression.

In case a template applies on a CCG inner node (a node with children), the lambda abstraction is applied on the semantics of the children, in order.

[4] `http://www.yaml.org/spec/`

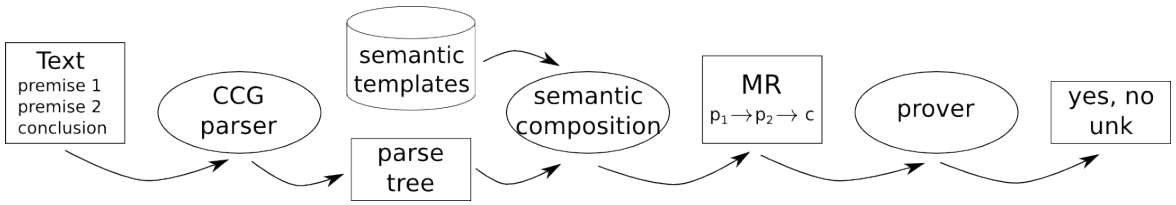

Figure 1: System pipeline for recognizing textual entailment. Syntactic structures of sentences are obtained with a CCG parser. Then, we perform the semantic composition using semantic templates. The resulting meaning representations are used to perform various logical inferences with a theorem prover.

$$\dfrac{\dfrac{\dfrac{\text{Some}}{\substack{NP/N \\ \lambda F\lambda G.\exists x.(Fx \wedge Gx)}} \quad \dfrac{\text{woman}}{\substack{N \\ \lambda x.\mathsf{woman}(x)}}}{\substack{NP \\ \lambda G.\exists x(\mathsf{woman}(x) \wedge G(x))}}\,{>} \quad \dfrac{\dfrac{\text{ordered}}{\substack{(S\backslash NP)/NP \\ \lambda Q_1\lambda Q_2.Q_2(\lambda x.Q_1(\lambda y.\mathsf{order}(x,y)))}} \quad \dfrac{\dfrac{\text{tea}}{\substack{N \\ \lambda y.\mathsf{tea}(y)}}}{\substack{NP \\ \lambda F.\exists y.(\mathsf{tea}(y) \wedge F(y))}}\,\text{lex}}{\substack{S\backslash NP \\ \lambda Q_2.Q_2(\lambda x.\exists y.(\mathsf{tea}(y) \wedge \mathsf{order}(x,y)))}}\,{>}}{\substack{S \\ \exists x.(\mathsf{woman}(x) \wedge \exists y.(\mathsf{tea}(y) \wedge \mathsf{order}(x,y)))}}\,{<}$$

Figure 2: CCG derivation tree of the sentence "Some woman ordered tea", with its semantics (simplified for illustrative purposes). The actual output of `ccg2lambda` with our provided templates is in Figure 6.

For example, in Figure 2, the template

– `semantics` : $\backslash E.\backslash F.\exists y.(E(y) \wedge F(y))$

`category` : NP

`rule` : lex

produces a type-raise from N to NP, and when applied to the CCG node whose child's semantics are $\lambda y.\mathsf{tea}(y)$, it will produce, after β-reduction, the formula $\lambda F.\exists y.(\mathsf{tea}(y) \wedge F(y))$. Here, the child's semantics $\lambda y.\mathsf{tea}(y)$ substitute all occurrences of the variable E. The newly composed semantic representation $\lambda F.\exists y.(\mathsf{tea}(y) \wedge F(y))$ now expects another predicate (a verb) as an argument F (i.e. "order"), which will be filled in the next step of the composition.

The `category` attribute of a semantic template may also specify conditions on the feature structures of CCG nodes (which are provided by the CCG parser), in which case templates apply if the syntactic category *matches* and the feature structure *subsumes* that of the CCG node. For example, if the semantic template specifies a syntactic category NP[`dcl` = `true`], it matches a CCG node with a category NP[`dcl` = `true`] or a category NP[`dcl` = `true`, `adj` = `true`].

Other conditions for matching templates to CCG nodes can be specified by adding more attributes to the semantic template. In the example above, the attribute `rule` : lex is used to specify the combination rule of that inner CCG node. In practice, any XML attribute of a CCG node can be used to specify matching conditions, which means that users of `ccg2lambda` can enrich CCG trees with arbitrary annotations such as Named Entities or Events and use them as matching conditions when defining semantic templates without modifying the software. It is also possible to specify attributes of the children of the target CCG node. These conditions are always prefixed by the string `child`, followed by the branch index 0 or 1. For example, a semantic template with the attribute `child1_child0_pos` : NN matches a node whose right child's (`child1`) left child's (`child0`) POS tag is an NN. Moreover, paths to child nodes can be left unspecified, by using the keyword `child_any_X` : Y; in this case, *any* child whose attribute `X` has value Y will be matched by the template. If more than one template matches a CCG node, the first appearing template is selected.

4.2 System Usage and Output

The command for the semantic composition is:

```
  # python semparse.py ccgtrees.xml
templates.yaml semantics.xml
```

where `ccgtrees.xml` is a Jigg's XML style CCG tree, `templates.yaml` contains the semantic templates, and `semantics.xml` is the XML output of the system. We also provide a script to convert C&C XML trees into Jigg's XML style. The output of `semparse.py` follows the conventions of Stanford coreNLP (see Figure 3). However, we follow Jigg's style to represent

```
<root>
  <sentences>
    <sentence>
      <tokens>
        <token base="tea" surf="tea" pos="NN"/>
        <token .../>
      </tokens>
      <ccg>
        <span id="s1" child="s2" category="N"
              rule="lex"/>
        <span .../>
      </ccg>
      <semantics>
        <span id="s1" child="s2" sem="\y._tea(y)"
              type="_tea : Entity -> Prop"/>
        <span .../>
      </semantics>
    </sentence>
  </sentences>
</root>
```

Figure 3: XML output of the semantic composition. Span nodes of the semantics tag contain logical semantic representations of that constituent.

element characteristics as XML node attributes. For example, the base and surface forms, and the POS tag of a token are all represented as XML attributes in a `<token>` tag.

Our semantic composition produces the `<semantics>` tag, which has as many children nodes (`<span>`) as the CCG tree, the same span identifiers and structure. However, semantic spans also have a "sem" attribute encoding the semantics (using NLTK's semantics format) that have been composed for that constituent. An example of a resulting semantic logic formula in NLTK semantics format is:

$$\texttt{\textbackslash F.exists}\ y.\ (\texttt{\_tea}(y)\ \&\ F(y))$$

Note that predicates are prefixed with an underscore to avoid collisions with reserved predicates in NLTK semantics format or in a potential prover.

Semantic spans also provide the type of single predicates (attribute "type"). For instance, the type of the predicate `_tea` is a function that receives an entity as an argument, and produces a proposition:

$$\texttt{\_tea} : \texttt{Entity} \rightarrow \texttt{Prop}$$

Types are automatically inferred using NLTK semantics functionality. However, it is possible to force arbitrary types in a semantic template by adding the attribute "coq_type". For example, we can specify the type for a transitive verb as:

– `semantics` : ...

  `category` : $(\mathrm{S}\backslash\mathrm{NP})/\mathrm{NP}$

  `coq_type` : $\mathrm{Entity} \rightarrow \mathrm{Entity} \rightarrow \mathrm{Prop}$

We can activate these types with the flag `--arbi-types` in the call to `semparse.py`.

5 Textual Entailment

The logical formulas that `ccg2lambda` outputs can be used in a variety of applications. In this demonstration, we use them to recognize textual entailment, an NLP problem that often requires precise language understanding. We assume that the user inputs a file with one sentence per line. All sentences are assumed to be premises, except the last sentence, which is assumed to be the conclusion. An entailment problem example is:
premise$_1$: All women ordered coffee or tea.
premise$_2$: Some woman did not order coffee.
conclusion: Some woman ordered tea.

Contrarily to other textual entailment systems based on logics (Angeli and Manning, 2014; MacCartney and Manning, 2007), we do not assume single-premise problems, which makes our system more general. The MRs of the problem above are:

$p_1 : \forall x.(\mathsf{woman}(x) \rightarrow \exists y.((\mathsf{tea}(y) \vee \mathsf{coffee}(y)) \wedge \mathsf{order}(x, y)))$

$p_2 : \exists x.(\mathsf{woman}(x) \wedge \neg\exists y.(\mathsf{coffee}(y) \wedge \mathsf{order}(x, y)))$

$c : \exists x.(\mathsf{woman}(x) \wedge \exists y.(\mathsf{tea}(y) \wedge \mathsf{order}(x, y)))$

We build a theorem by concatenating meaning representations of the premises $\{p_1, \ldots, p_n\}$ and the conclusion c with the implication operator, which is a convenience in theorem proving:

$$\mathrm{Theorem} : p_1 \rightarrow \ldots \rightarrow p_n \rightarrow c. \qquad (1)$$

And then, we define predicate types as:
Parameter _tea : Entity → Prop.
Parameter _order : Entity → Entity → Prop.

Finally, we pipe the theorem and type definitions to Coq (Castéran and Bertot, 2004), an interactive higher-order prover that we run fully automated with the use of some tactics (including arithmetics and equational reasoning), as described in Mineshima et al. (2015). We return the label *yes* (entailment) if the conclusion can be logically proved from the premises, *no* if the negated conclusion can be proved, and *unknown* otherwise.

The recognition of textual entailment can be performed with the following command:

```
# python prove.py semantics.xml
```

where the entailment judgment (*yes*, *no*, *unknown*) is printed to standard output. Moreover, the flag `--graph_out` allows to specify an HTML file to print a graphical visualization of the CCG tree structure of sentences, their semantic composition (every constituent annotated with a component of the formula), and the prover script. An excerpt of the visualization is shown in Section 6.

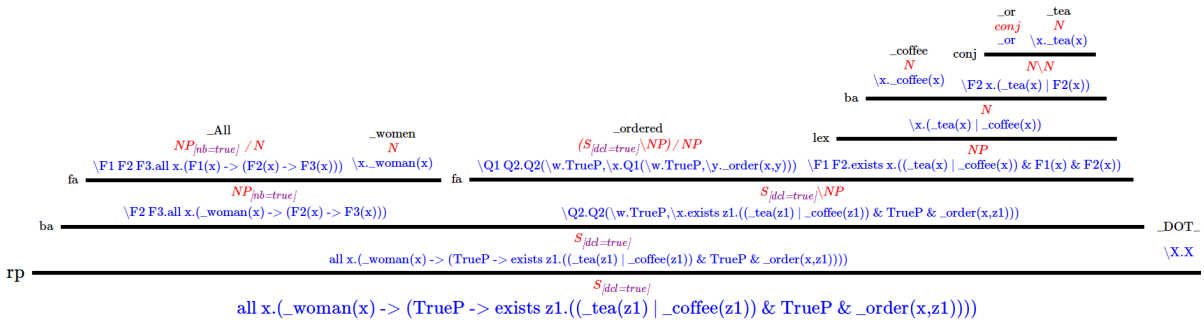

Figure 4: Visualization of the semantic output of `ccg2lambda` for the sentence "All women ordered coffee or tea." where logical semantic representations appear below their respective CCG nodes.

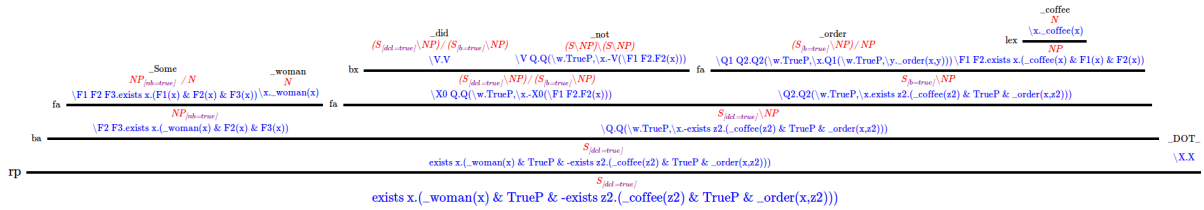

Figure 5: Visualization of the semantic output of `ccg2lambda` for the sentence "Some woman did not order coffee." where logical semantic representations appear below their respective CCG nodes.

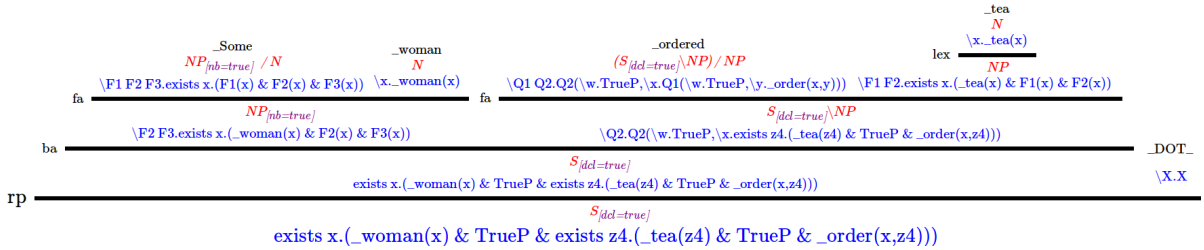

Figure 6: Visualization of the semantic output of `ccg2lambda` for the sentence "Some woman ordered tea." where logical semantic representations appear below their respective CCG nodes.

6 Visualization

For visualization purposes, we provide a separate script that can be called as:

```
# python visualize.py semantics.xml
  > semantics.html
```

which produces a file `semantics.html` with an HTML graphical representation of the CCG tree, augmented at every node with the semantics composed up to that node (see Figures 4, 5 and 6 for an excerpt). These semantic representations are obtained with the semantic templates that accompany our software and that were developed and evaluated in Mineshima et al. (2015). The trivial propositions "TrueP" have no effect and appear in the formulas in place of potential modifiers (such as adjectives or adverbs) of more complex sentences. The visualization can be configured to display the root on top, change colors and sizes of the syntactic categories, feature structures, logical formulas and base forms at the leaves.

7 Future Work and Conclusion

As an extension to `ccg2lambda`, it would be valuable to produce (possibly scored) N-best lists of logical formulas, instead of the current single 1-best. Moreover, our current semantic templates do not cover all syntactic categories that C&C or Jigg produce, and we need a good default combination mechanism. Other minor enhancements are to produce logical formulas for each CCG derivation in an N-best list, and to allow features other than the base form to become predicates.

In this paper we have demonstrated our system to convert CCG trees to logic MRs. It operates by composing semantics bottom-up, guided by the combinatory characteristics of the CCG derivation and semantic templates provided by the user. In this release, semantic templates for English and Japanese are also included. As Mineshima et al. (2015) has shown, the MRs obtained by `ccg2lambda` are useful to recognize textual

entailment. We believe that these easy-to-produce MRs can be useful to NLP tasks that require precise language understanding or that benefit from using MRs as features in their statistical systems.

Acknowledgments

This work was supported by CREST, JST.

References

Lasha Abzianidze. 2015. A tableau prover for natural logic and language. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2492–2502, Lisbon, Portugal, September. Association for Computational Linguistics.

Gabor Angeli and Christopher D. Manning. 2014. NaturalLI: Natural logic inference for common sense reasoning. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 534–545, Doha, Qatar, October. Association for Computational Linguistics.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA, October. Association for Computational Linguistics.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*. O'Reilly Media, Inc.

Patrick Blackburn and Johan Bos. 2005. *Representation and Inference for Natural Language: A First Course in Computational Semantics*. CSLI.

Johan Bos, Stephen Clark, Mark Steedman, James R Curran, and Julia Hockenmaier. 2004. Wide-coverage semantic representations from a CCG parser. In *Proceedings of the 20th international conference on Computational Linguistics*, pages 1240–1246. Association for Computational Linguistics.

Pierre Castéran and Yves Bertot. 2004. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions.* Springer Verlag.

Lucas Champollion, Joshua Tauberer, and Maribel Romero. 2007. The Penn Lambda Calculator: Pedagogical software for natural language semantics. In Tracy Holloway King, editor, *Proceedings of the Grammar Engineering Across Frameworks (GEAF0 7) Workshop*, pages 106–127, Stanford. CSLI Publications.

Stephen Clark and James R Curran. 2004. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, pages 104–111. Association for Computational Linguistics.

Robin Cooper, Richard Crouch, Jan van Eijck, Chris Fox, Josef van Genabith, Jan Jaspers, Hans Kamp, Manfred Pinkal, Massimo Poesio, Stephen Pulman, et al. 1994. FraCaS–a framework for computational semantics. *deliverable*, D6.

Dan Garrette and Ewan Klein. 2009. An extensible toolkit for computational semantics. In *Proceedings of the Eighth International Conference on Computational Semantics*, IWCS-8 '09, pages 116–127, Stroudsburg, PA, USA. Association for Computational Linguistics.

Mike Lewis and Mark Steedman. 2013. Combining distributional and logical semantics. *Transactions of the Association for Computational Linguistics*, 1:179–192.

Bill MacCartney and Christopher D Manning. 2007. Natural logic for textual inference. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pages 193–200. Association for Computational Linguistics.

Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, Baltimore, Maryland, June. Association for Computational Linguistics.

Koji Mineshima, Pascual Martínez-Gómez, Yusuke Miyao, and Daisuke Bekki. 2015. Higher-order logical inference with compositional semantics. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2055–2061, Lisbon, Portugal, September. Association for Computational Linguistics.

Hiroshi Noji and Yusuke Miyao. 2016. Jigg: A framework for an easy natural language processing pipeline. In *Proceedings of ACL 2016 System Demonstrations*, Berlin, Germany, August. Association for Computational Linguistics.

Mark Steedman. 2000. *The Syntactic Process*. MIT Press.

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

Nguyen Vo, Arindam Mitra, and Chitta Baral. 2015. The NL2KR platform for building natural language translation systems. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 899–908, Beijing, China, July. Association for Computational Linguistics.