Deterministic Statistical Mapping of Sentences to Underspecified Semantics

Hiyan Alshawi, Pi-Chuan Chang, and Michael Ringgaard

Abstract We present a method for training a statistical model for mapping natural language sentences to semantic expressions. The semantics are expressions of an underspecified logical form that has properties making it particularly suitable for statistical mapping from text. An encoding of the semantic expressions into dependency trees with automatically generated labels allows application of existing methods for statistical dependency parsing to the mapping task (without the need for separate traditional dependency labels or parts of speech). The encoding also results in a natural per-word semantic-mapping accuracy measure. We report on the results of training and testing statistical models for mapping sentences of the Penn Treebank into the semantic expressions, for which per-word semantic mapping accuracy ranges between 79 % and 86 % depending on the experimental conditions. The particular choice of algorithms used also means that our trained mapping is deterministic (in the sense of deterministic parsing), paving the way for large-scale text-to-semantic mapping.

1 Introduction

Producing semantic representations of text is motivated not only by theoretical considerations but also by the hypothesis that semantics can be used to improve automatic systems for tasks that are intrinsically semantic in nature such as question answering, textual entailment, machine translation, and more generally any natural language task that might benefit from inference in order to more closely approximate human performance. Since formal logics have formal denotational semantics,

H. Alshawi (✉) · P.-C. Chang
Google, 1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA
e-mail: hiyan@google.com

P.-C. Chang
e-mail: pichuan@google.com

M. Ringgaard
Google, Åbogade 15, 8200 Aarhus, Denmark
e-mail: ringgaard@google.com

H. Bunt et al. (eds.), *Computing Meaning*, Text, Speech and Language Technology 47,
DOI 10.1007/978-94-007-7284-7_2,

and are good candidates for supporting inference, they have often been taken to be the targets for mapping text to semantic representations, with frameworks emphasizing (more) tractable inference choosing first order predicate logic (Stickel 1985) while those emphasizing representational power favoring one of the many available higher order logics (van Benthem 1995).

It was later recognized that in order to support some tasks, fully specifying certain aspects of a logic representation, such as quantifier scope, or reference resolution, is often not necessary. For example, for semantic translation, most ambiguities of quantifier scope can be carried over from the source language to the target language without being resolved. This led to the development of underspecified semantic representations, such as QLF (Alshawi and Crouch 1992) and MRS (Copestake et al. 2005), which are easier to produce from text without contextual inference but which can be further specified as necessary for the task being performed.

While traditionally mapping text to formal representations was predominantly rule-based, for both the syntactic and semantic components (Montague 1974; Pereira and Shieber 1987; Alshawi 1992), good progress in statistical syntactic parsing (Collins 1999; Charniak 2000) led to systems that applied rules for semantic interpretation to the output of a statistical syntactic parser (e.g. Bos et al. 2004). More recently researchers have looked at statistical methods to provide robust and trainable methods for mapping text to formal representations of meaning (Zettlemoyer and Collins 2005).

In this paper we further develop the two strands of work mentioned above, i.e. mapping text to underspecified semantic representations and using statistical parsing methods to perform the analysis. Here we take a more direct route, starting from scratch by designing an underspecified semantic representation (Natural Logical Form, or NLF) that is purpose-built for statistical text-to-semantics mapping. An underspecified logic whose constructs are motivated by natural language and that is amenable to trainable direct semantic mapping from text without an intervening layer of syntactic representation. In contrast, the approach taken by Zettlemoyer and Collins (2005), for example, maps into traditional logic via lambda expressions, and the approach taken by Poon and Domingos (2009) depends on an initial step of syntactic parsing.

In this paper, we describe a supervised training method for mapping text to NLF, that is, producing a statistical model for this mapping starting from training pairs consisting of sentences and their corresponding NLF expressions. This method makes use of an encoding of NLF expressions into dependency trees in which the set of labels is automatically generated from the encoding process (rather than being pre-supplied by a linguistically motivated dependency grammar). This encoding allows us to perform the text-to-NLF mapping using any existing statistical methods for labeled dependency parsing (Eisner 1996; Yamada and Matsumoto 2003; McDonald et al. 2005). A side benefit of the encoding is that it leads to a natural per-word measure for semantic mapping accuracy which we use for evaluation purposes. By combing our method with deterministic statistical dependency models together with deterministic (hard) clusters instead of parts of speech, we obtain a deterministic statistical text-to-semantics mapper, opening the way to feasible mapping of text-to-semantics at a large scale, for example the entire web.

This paper concentrates on the text-to-semantics mapping which depends, in part, on some properties of NLF. We will not attempt to defend the semantic representation choices for specific constructions illustrated here. NLF is akin to a variable-free variant of QLF or an MRS in which some handle constraints are determined during parsing. For the purposes of this paper it is sufficient to note that NLF has roughly the same granularity of semantic representation as these earlier underspecified representations.

We outline the steps of our text-to-semantics mapping method in Sect. 2, introduce NLF in Sect. 3, explain the encoding of NLF expressions as formal dependency trees in Sect. 4, and report on experiments for training and testing statistical models for mapping text to NLF expressions in Sect. 5.

2 Direct Semantic Mapping

Our method for mapping text to natural semantics expressions proceeds as follows:

1. Create a corpus of pairs consisting of text sentences and their corresponding NLF semantic expressions.
2. For each of the sentence-semantics pairs in the corpus, align the words of the sentence to the tokens of the NLF expressions.
3. "Encode" each alignment pair as an ordered dependency tree in which the labels are generated by the encoding process.
4. Train a statistical dependency parsing model with the set of dependency trees.
5. For a new input sentence S, apply the statistical parsing model to S, producing a labeled dependency tree D_S.
6. "Decode" D_S into a semantic expression for S.

For step 1, the experiments in this paper (Sect. 5) obtain the corpus by converting an existing constituency treebank into semantic expressions. However, direct annotation of a corpus with semantic expressions *is* a viable alternative, and indeed we are separately exploring that possibility for a different, open domain, text corpus.

For steps 4 and 5, any method for training and applying a dependency model from a corpus of labeled dependency trees may be used. As described in Sect. 5, for the experiments reported here we use an algorithm similar to that of Nivre (Nivre 2003).

For steps 2, 3 and 6, the encoding of NLF semantic expressions as dependency trees with automatically constructed labels is described in Sect. 4.

3 Semantic Expressions

NLF expressions are by design amenable to facilitating training of text-to-semantics mappings. For this purpose, NLF has a number of desirable properties:

Fig. 1 Example of an NLF semantic expression

```
[acquired
     /stealthily
     :[in, ^, 2002],
 Chirpy+Systems,
 companies.two
     :profitable
     :[producing,
       ^,
       pet+accessories]]
```

1. Apart from a few built-in logical connectives, all the symbols appearing in NLF expressions are natural language words.
2. For an NLF semantic expression corresponding to a sentence, the word tokens of the sentence appear exactly once in the NLF expression.
3. The NLF notation is variable-free.

Technically, NLF expressions are expression of an underspecified logic, i.e. a semantic representation that leaves open the interpretation of certain constructs (for example the scope of quantifiers and some operators and the referents of terms such as anaphora, and certain implicit relations such as those for compound nominals). NLF is similar in some ways to Quasi Logical Form, or QLF (Alshawi 1992), but the properties listed above keep NLF closer to natural language than QLF, hence *natural* logical form.[1] There is no explicit formal connection between NLF and Natural Logic (van Benthem 1986), though it may turn out that NLF is a convenient starting point for some Natural Logic inferences.

In contrast to statements of a fully specified logic in which denotations are typically taken to be *functions* from possible worlds to truth values (Montague 1974), denotations of a statement in an underspecified logic are typically taken to be *relations* between possible worlds and truth values (Alshawi and Crouch 1992; Alshawi 1996). Formal denotations for NLF expressions are beyond the scope of this paper and will be described elsewhere.

3.1 Connectives and Examples

A NLF expression for the sentence *In 2002, Chirpy Systems stealthily acquired two profitable companies producing pet accessories* is shown in Fig. 1.

The NLF constructs and connectives are explained in Table 1. For variable-free abstraction, an NLF expression `[p, ^, a]` corresponds to $\lambda x.p(x, a)$. Note that some common logical operators are not built-in since they will appear directly as words such as *not*.[2]

[1] The term QLF is now sometimes used informally (e.g. Liakata and Pulman 2002; Poon and Domingos 2009) for any logic-like semantic representation without explicit quantifier scope.

[2] NLF does include Horn clauses, which implicitly encode negation, but since Horn clauses are not part of the experiments reported in this paper, we will not discuss them further here.

Table 1 NLF constructs and connectives

Operator	Example	Denotation	Lang. constructs
`[...]`	`[sold, Chirpy, Growler]`	predication tuple	clauses, prepositions, ...
`:`	`company:profitable`	intersection	adjectives, relative clauses, ...
`.`	`companies.two`	(unscoped) quantification	determiners, measure terms
`^`	`[in, ^, 2005]`	variable-free abstract	prepositions, relatives, ...
`_`	`[eating, _, apples]`	unspecified argument	missing verb arguments, ...
`{...}`	`and{Chirpy, Growler}`	collection	noun phrase coordination, ...
`/`	`acquired/stealthily`	type-preserving operator	adverbs, modals, ...
`+`	`Chirpy+Systems`	implicit relation	compound nominals, ...
`@`	`meeting@yesterday`	temporal restriction	bare temporal modifiers, ...
`&`	`[...] & [...]`	conjunction	sentences, ...
`|...|`	`|Dublin, Paris, Bonn|`	sequence	paragraphs, fragments, lists, ...
`%`	`met%as`	uncovered op	constructs not covered

We currently use the unknown/unspecified operator, `%`, mainly for linguistic constructions that are beyond the coverage of a particular semantic mapping model. An example that includes `%` in our converted WSJ corpus is *Other analysts are nearly as pessimistic* for which the NLF expression is

```
[are, analysts.other, pessimistic%nearly%as]
```

In Sect. 5 we give some statistics on the number of semantic expressions containing `%` in the data used for our experiments and explain how it affects our accuracy results.

4 Encoding Semantics as Dependencies

We encode NLF semantic expressions as labeled dependency trees in which the label set is generated automatically by the encoding process. This is in contrast to conventional dependency trees for which the label sets are presupplied (e.g. by a linguistic theory of dependency grammar). The purpose of the encoding is to enable training of a statistical dependency parser and converting the output of that parser

for a new sentence into a semantic expression. The encoding involves three aspects: Alignment, headedness, and label construction.

4.1 Alignment

Since, by design, each word token corresponds to a symbol token (the same word type) in the NLF expression, the only substantive issue in determining the alignment is the occurrence of multiple tokens of the same word type in the sentence. Depending on the source of the sentence-NLF pairs used for training, a particular word in the sentence may or may not already be associated with its corresponding word position in the sentence. For example, in some of the experiments reported in this paper, this correspondence is provided by the semantic expressions obtained by converting a constituency treebank (the well-known Penn WSJ treebank). For situations in which the pairs are provided without this information, as is the case for direct annotation of sentences with NLF expressions, we currently use a heuristic greedy algorithm for deciding the alignment. This algorithm tries to ensure that dependents are near their heads, with a preference for projective dependency trees. To guage the importance of including correct alignments in the input pairs (as opposed to training with inferred alignments), we will present accuracy results for semantic mapping for both correct and automatically infererred alignments.

4.2 Headedness

The encoding requires a definition of headedness for words in an NLF expression, i.e., a head-function h from dependent words to head words. We define h in terms of a head-function g from an NLF (sub)expression e to a word w appearing in that (sub)expression, so that $g(w) = w$, and, recursively:

$$
\begin{aligned}
g\big([e_1, \ldots, e_n]\big) &= g(e_1) \\
g(e_1 : e_2) &= g(e_1) \\
g(e_1.e_2) &= g(e_1) \\
g(e_1/e_2) &= g(e_1) \\
g(e_1@e_2) &= g(e_1) \\
g(e_1\&e2) &= g(e_1) \\
g\big(|e_1, \ldots, e_n|\big) &= g(e1) \\
g\big(e_1\{e_2, \ldots, e_n\}\big) &= g(e_1) \\
g(e_1 + \cdots + e_n) &= g(e_n) \\
g(e_1\%e_2) &= g(e_1).
\end{aligned}
$$

Then a head word $h(w)$ for a dependent w is defined in terms of the smallest (sub)expression e containing w for which

$$h(w) = g(e) \neq w.$$

For example, for the NLF expression in Fig. 1, this yields the heads shown in Table 3. (The labels shown in that table will be explained in the following section.)

This definition of headedness is not the only possible one, and other variations could be argued for. The specific definition for NLF heads turns out to be fairly close to the notion of head in traditional dependency grammars. This is perhaps not surprising since traditional dependency grammars are often partly motivated by semantic considerations, if only informally.

4.3 Label Construction

As mentioned, the labels used during the encoding of a semantic expression into a dependency tree are derived so as to enable reconstruction of the expression from a labeled dependency tree. In a general sense, the labels may be regarded as a kind of formal semantic label, though more specifically, a label is interpretable as a sequence of instructions for constructing the part of a semantic expression that links a dependent to its head, given that part of the semantic expression, including that derived from the head, has already been constructed. The string for a label thus consists of a sequence of atomic instructions, where the decoder keeps track of a current expression and the parent of that expression in the expression tree being constructed. When a new expression is created it becomes the current expression whose parent is the old current expression. The atomic instructions (each expressed by a single character) are shown in Table 2.

A sequence of instructions in a label can typically (but not always) be paraphrased informally as "starting from head word w_h, move to a suitable node (at or above w_h) in the expression tree, add specified NLF constructs (connectives, tuples, abstracted arguments) and then add w_d as a tuple or connective argument."

Continuing with our running example, the labels for each of the words are shown in Table 3.

Algorithmically, we find it convenient to transform semantic expressions into dependency trees and vice versa via a derivation tree for the semantic expression in which the atomic instruction symbols listed above are associated with individual nodes in the derivation tree.

The output of the statistical parser may contain inconsistent trees with formal labels, in particular trees in which two different arguments are predicated to fill the same position in a semantic expression tuple. For such cases, the decoder that produces the semantic expression applies the simple heuristic of using the next available tuple position when such a conflicting configuration is predicated. In our experiments, we are measuring per-word semantic head-and-label accuracy, so this heuristic does not play a part in that evaluation measure.

Table 2 Atomic instructions in formal label sequences

Instruction	Decoding action
`[`, `{`, `|`	Set the current expression to a newly created tuple, collection, or sequence.
`:`, `/`, `.`, `+`, `&`, `@`, `%`	Attach the current subexpression to its parent with the specified connective.
`*`	Set the current expression to a newly created symbol from the dependent word.
`0`, `1`, ...	Add the current expression at the specified parent tuple position.
`^`, `_`	Set the current subexpression to a newly created abstracted-over or unspecified argument.
`-`	Set the current subexpression to be the parent of the current expression.

Table 3 Formal labels for an example sentence

Dependent	Head	Label
in	acquired	`[:^1-*0`
2002	in	`-*2`
Chirpy	Systems	`*+`
Systems	acquired	`-*1`
stealthily	acquired	`*/`
acquired		`[*0`
two	companies	`*.`
profitable	companies	`*:`
companies	acquired	`-*2`
producing	companies	`[:^1-*0`
pet	accessories	`*+`
accessories	producing	`-*2`

5 Experiments

5.1 Data Preparation

In the experiments reported here, we derive our sentence-semantics pairs for training and testing from the Penn WSJ Treebank. This choice reflects the lack, to our knowledge, of a set of such pairs for a reasonably sized publicly available corpus, at least for NLF expressions. Our first step in preparing the data was to convert the WSJ phrase structure trees into semantic expressions. This conversion is done by programming the Stanford treebank toolkit to produce NLF trees bottom-up from the phrase structure trees. This conversion process is not particularly noteworthy in itself (being a traditional rule-based syntax-to-semantics translation process) except

Table 4 Datasets used in experiments

Dataset	Null labels?	Auto align?	WSJ sections	Sentences
Train+Null-AAlign	yes	no	2–21	39213
Train-Null-AAlign	no	no	2–21	24110
Train+Null+AAlign	yes	yes	2–21	35778
Train-Null+AAlign	no	yes	2–21	22611
Test+Null-AAlign	yes	no	23	2416
Test-Null-AAlign	no	no	23	1479

perhaps to the extent that the closeness of NLF to natural language perhaps makes the conversion somewhat easier than, say, conversion to a fully resolved logical form.

Since our main goal is to investigate trainable mappings from text strings to semantic expressions, we only use the WSJ phrase structure trees in data preparation: the phrase structure trees are not used as inputs when training a semantic mapping model, or when applying such a model. For the same reason, in these experiments, we do not use the part-of-speech information associated with the phrase structure trees in training or applying a semantic mapping model. Instead of parts-of-speech we use word cluster features from a hierarchical clustering produced with the unsupervised Brown clustering method (Brown et al. 1992); specifically we use the publicly available clusters reported by Koo et al. (2008).

Constructions in the WSJ that are beyond the explicit coverage of the conversion rules used for data preparation result in expressions that include the unknown/unspecified (or 'Null') operator `%`. We report on different experimental settings in which we vary how we treat training or testing expressions with `%`. This gives rise to the data sets in Table 4 which have +Null (i.e., including `%`), and -Null (i.e., not including `%`) in the data set names.

Another attribute we vary in the experiments is whether to align the words in the semantic expressions to the words in the sentence automatically, or whether to use the correct alignment (in this case preserved from the conversion process, but could equally be provided as part of a manual semantic annotation scheme, for example). In our current experiments, we discard non-projective dependency trees from training sets. Automatic alignment results in additional non-projective trees, giving rise to different effective training sets when auto-alignment is used: these sets are marked with +AAlign, otherwise -AAlign. The training set numbers shown in Table 4 are the resulting sets after removal of non-projective trees.

5.2 Parser

As mentioned earlier, our method can make use of any trainable statistical dependency parsing algorithm. The parser is trained on a set of dependency trees with

Table 5 Per-word semantic accuracy when training with the correct alignment

Training	Test	Accuracy (%)
+Null-AAlign	+Null-AAlign	81.2
-Null-AAlign	+Null-AAlign	78.9
-Null-AAlign	-Null-AAlign	86.1
+Null-AAlign	-Null-AAlign	86.5

Table 6 Per-word semantic accuracy when training with an auto-alignment

Training	Test	Accuracy (%)
+Null+AAlign	+Null-AAlign	80.4
-Null+AAlign	+Null-AAlign	78.0
-Null+AAlign	-Null-AAlign	85.5
+Null+AAlign	-Null-AAlign	85.8

formal labels as explained in Sects. 2 and 4. The specific parsing algorithm we use in these experiments is a deterministic shift reduce algorithm (Nivre 2003), and the specific implementation of the algorithm uses a linear SVM classifier for predicting parsing actions (Chang et al. 2010). As noted above, hierarchical cluster features are used instead of parts-of-speech; some of the features use coarse (6-bit) or finer (12-bit) clusters from the hierarchy. More specifically, the full set of features is:

- The words for the current and next input tokens, for the top of the stack, and for the head of the top of the stack.
- The formal labels for the top-of-stack token and its leftmost and rightmost children, and for the leftmost child of the current token.
- The cluster for the current and next three input tokens and for the top of the stack and the token below the top of the stack.
- Pairs of features combining 6-bit clusters for these tokens together with 12-bit clusters for the top of stack and next input token.

5.3 Results

Tables 5 and 6 show the *per-word semantic accuracy* for different training and test sets. This measure is simply the percentage of words in the test set for which both the predicted formal label and the head word are correct. In syntactic dependency evaluation terminology, this corresponds to the labeled attachment score.

All tests are with respect to the correct alignment; we vary whether the correct alignment (Table 5) or auto-alignment (Table 6) is used for training to give an idea of how much our heuristic alignment is hurting the semantic mapping model. As shown by comparing the two tables, the loss in accuracy due to using the automatic alignment is only about 1 %, so while the automatic alignment algorithm can probably be improved, the resulting increase in accuracy would be relatively small.

Table 7 Per-word semantic accuracy after pruning label sets in Train-Null+AAlign (and testing with Test-Null-AAlign)

# Labels	# Train sents	Accuracy (%)
151 (all)	22611	85.5
100	22499	85.5
50	21945	85.5
25	17669	83.8
12	7008	73.4

As shown in the Tables 5 and 6, two versions of the test set are used: one that includes the 'Null' operator %, and a smaller test set with which we are testing only the subset of sentences for which the semantic expressions do not include this label. The highest accuracies (mid 80's) shown are for the (easier) test set which excludes examples in which the test semantic expressions contain Null operators. The strictest settings, in which semantic expressions with Null are not included in training but included in the test set effectively treat prediction of Null operators as errors. The lower accuracy (high 70's) for such stricter settings thus incorporates a penalty for our incomplete coverage of semantics for the WSJ sentences. The less strict Test+Null settings in which % is treated as a valid output may be relevant to applications that can tolerate some unknown operators between subexpressions in the output semantics.

Next we look at the effect of limiting the size of the automatically generated formal label set prior to training. For this we take the configuration using the TrainWSJ-Null+AAlign training set and the TestWSJ-Null-AAlign test set (the third row in Table refPerWordSemanticAccuracyAAlign for which auto-alignment is used and only labels without the NULL operator % are included). For this training set there are 151 formal labels. We then limit the training set to instances that only include the most frequent k labels, for $k = 100, 50, 25, 12$, while keeping the test set the same. As can be seen in Table 7, the accuracy is unaffected when the training set is limited to the 100 most frequent or 50 most frequent labels. There is a slight loss when training is limited to 25 labels and a large loss if it is limited to 12 labels. This appears to show that, for this corpus, the core label set needed to construct the majority of semantic expressions has a size somewhere between 25 and 50. It is perhaps interesting that this is roughly the size of hand-produced traditional dependency label sets. On the other hand, it needs to be emphasized that since Table 7 ignores beyond-coverage constructions that presently include Null labels, it is likely that a larger label set would be needed for more complete semantic coverage.

6 Conclusion and Further Work

We've shown that by designing an underspecified logical form that is motivated by, and closely related to, natural language constructions, it is possible to train a direct statistical mapping from pairs of sentences and their corresponding semantic expressions, with per-word accuracies ranging from 79 % to 86 % depending on the

strictness of the experimental setup. The input to training does not require any traditional syntactic categories or parts of speech. We also showed, more specifically, that we can train a model that can be applied deterministically at runtime (using a deterministic shift reduce algorithm combined with deterministic clusters), making large-scale text-to-semantics mapping feasible.

In traditional formal semantic mapping methods (Montague 1974; Bos et al. 2004), and even some recent statistical mapping methods (Zettlemoyer and Collins 2005), the semantic representation is overloaded to performs two functions: (i) representing the final meaning, and (ii) composing meanings from the meanings of subconstituents (e.g. through application of higher order lambda functions). In our view, this leads to what are perhaps overly complex semantic representations of some basic linguistic constructions. In contrast, in the method we presented, these two concerns (meaning representation and semantic construction) are separated, enabling us to keep the semantics of constituents simple, while turning the construction of semantic expressions into a separate structured learning problem (with its own internal prediction and decoding mechanisms).

Although in the experiments we reported here we *do* prepare the training data from a traditional treebank, we are encouraged by the results and believe that annotation of a corpus with only semantic expressions is sufficient for building an efficient and reasonably accurate text-to-semantics mapper. Indeed, we have started building such a corpus for a question answering application, and hope to report results for that corpus in the future. Other further work includes a formal denotational semantics of the underspecified logical form and elaboration of practical inference operations with the semantic expressions. This work may also be seen as a step towards viewing semantic interpretation of language as the interaction between a pattern recognition process (described here) and an inference process.

References

Alshawi, H. (Ed.) (1992). *The core language engine*. Cambridge: MIT Press.

Alshawi, H. (1996). Underspecified first order logics. In K. van Deemter & S. Peters (Eds.), *Semantic ambiguity and underspecification* (pp. 145–158). Stanford: CSLI.

Alshawi, H., & Crouch, R. (1992). Monotonic semantic interpretation. In *Proceedings of the 30th annual meeting of the association for computational linguistics*, Newark, Delaware (pp. 32–39).

Bos, J., Clark, S., Steedman, M., Curran, J. R., & Hockenmaier, J. (2004). Wide-coverage semantic representations from a ccg parser. In *Proceedings of the 20th international conference on computational linguistics*, Geneva, Switzerland (pp. 1240–1246).

Brown, P., Pietra, V., Souza, P., Lai, J., & Mercer, R. (1992). Class-based n-gram models of natural language. *Computational Linguistics*, *18*(4), 467–479.

Chang, Y.-W., Hsieh, C.-J., Chang, K.-W., Ringgaard, M., & Lin, C.-J. (2010). Training and testing low-degree polynomial data mappings via linear svm. *Journal of Machine Learning Research*, *11*, 1471–1490.

Charniak, E. (2000). A maximum entropy inspired parser. In *Proceedings of the 1st conference of the North American chapter of the association for computational linguistics*, Seattle, Washington (pp. 132–139).

Collins, M. (1999). *Head driven statistical models for natural language parsing*. Ph.D. thesis, University of Pennsylvania.

Copestake, A., Flickinger, D., & Sag, C. P. I. (2005). Minimal recursion semantics, an introduction. *Research on Language and Computation*, *3*, 281–332.

Eisner, J. (1996). Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th international conference on computational linguistics*, Sydney (pp. 340–345).

Koo, T., Carreras, X., & Collins, M. (2008). Simple semisupervised dependency parsing. In *Proceedings of the annual meeting of the association for computational linguistics*, Columbus, Ohio (pp. 595–603).

Liakata, M., & Pulman, S. (2002). From trees to predicate-argument structures. In *Proceedings of the 19th international conference on computational linguistics*, Taipei, Taiwan (pp. 563–569).

McDonald, R., Crammer, K., & Pereira, F. (2005). Online large-margin training of dependency parsers. In *Proceedings of the 43rd annual meeting of the association for computational linguistics*, Ann Arbor, Michigan (pp. 91–98).

Montague, R. (1974). The proper treatment of quantification in ordinary English. In R. Thomason (Ed.), *Formal philosophy: Selected papers of Richard Montague* (pp. 247–270). New Haven: Yale University Press.

Nivre, J. (2003). An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th international workshop on parsing technologies*, Nancy, France (pp. 149–160).

Pereira, F., & Shieber, S. (1987). *Prolog and natural language analysis*. Stanford: CSLI.

Poon, H., & Domingos, P. (2009). Unsupervised semantic parsing. In *Proceedings of the 2009 conference on empirical methods in natural language processing*, Singapore (pp. 1–10).

Stickel, M. (1985). Automated deduction by theory resolution. *Journal of Automated Reasoning*, *1*, 333–355.

van Benthem, J. (1986). *Essays in logical semantics*. Dordrecht: Reidel.

van Benthem, J. (1995). *Language in action: Categories, lambdas, and dynamic logic*. Cambridge: MIT Press.

Yamada, H., & Matsumoto, Y. (2003). Statistical dependency analysis with support vector machines. In *Proceedings of the 8th international workshop on parsing technologies*, Nancy, France (pp. 195–206).

Zettlemoyer, L. S., & Collins, M. (2005). Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the 21st conference on uncertainty in artificial intelligence*, Edinburgh, Scotland (pp. 658–666).

http://www.springer.com/978-94-007-7283-0

Computing Meaning
Volume 4
Bunt, H.; Bos, J.; Pulman, S. (Eds.)
2014, VIII, 260 p. 13 illus., Hardcover
ISBN: 978-94-007-7283-0