Exploring Graph-Algebraic CCG Combinators for Syntactic-Semantic AMR Parsing

Sebastian Beschke
University of Hamburg, Germany
beschke@informatik.uni-hamburg.de

Abstract

We describe a new approach to semantic parsing based on Combinatory Categorial Grammar (CCG). The grammar's semantic construction operators are defined in terms of a graph algebra, which allows our system to induce a compact CCG lexicon. We introduce an expectation maximisation algorithm which we use to filter our lexicon down to 2500 lexical templates. Our system achieves a semantic triple (Smatch) precision that is competitive with other CCG-based AMR parsing approaches.

1 Introduction

Parsing sentences to formal meaning representations, known as *Semantic Parsing*, is a task at the frontier of Natural Language Understanding. Abstract Meaning Representation (AMR) is a meaning representation language that represents sentence semantics in the form of graphs. Research on AMR parsing systems has been very productive in recent years with many competing approaches.

Current AMR parsers vary regarding the extent to which they rely on a formal grammar. Some of the most successful systems generate AMRs through an end-to-end neural architecture, with no explicit symbolic derivations (Zhang et al., 2019). Other parsers employ transition systems with limited explanatory power (Peng et al., 2018). Constructing grammar-based semantic analyses that can be understood in terms of linguistic theory is a more difficult task than end-to-end AMR parsing because of the additional structural requirements on the output and the algorithmic constraints imposed thereby.

In this paper, we explore how semantic parsers can be built to be *interpretable* and *transparent*. Interpretability means that our system exposes rich symbolic information in the form of CCG derivations. Transparency means that it works with a compact and intuitively plausible lexicon. The lexicon is itself an artifact that can be inspected.

We achieve these goals by equipping CCG with graph-based semantics. Meaning reprentations are constructed through the operations of a simple graph algebra, which effectively constrains the search space for parsing and lexicon induction and makes the available operations and resulting lexical items easy to understand.

Technical contributions of this paper include a modified expectation-maximisation (EM) algorithm to induce compact delexicalised CCG lexica, a technique for training a syntactic-semantic supertagger with incomplete labels, and a hybrid update mechanism for training the linear parsing model.

1.1 Related Work

This work builds upon the concept of graph-algebraic CCG, which has so far been tested only in the context of lexicon induction (Beschke and Menzel, 2018). We extend the lexicon induction process by delexicalisation and EM filtering and demonstrate the first end-to-end parsing system based on graph-algebraic CCG. The idea of applying graph algebras to AMR parsing has also been applied in the context of Interpreted Regular Tree Grammar (Groschwitz et al., 2018). Furthermore, improved definitions of graph-composing CCG combinators have been proposed (Blodgett and Schneider, 2019) to cover a wider range of semantic phenomena.

Other systems that apply CCG to AMR parsing use an encoding of AMR graphs to λ-calculus expressions (Artzi et al., 2015; Misra and Artzi, 2016). One drawback of these systems is that lexicon induction is coupled to the training loop of a

Proceedings of Recent Advances in Natural Language Processing, pages 112–121,
Varna, Bulgaria, Sep 2–4, 2019.
https://doi.org/10.26615/978-954-452-056-4_014

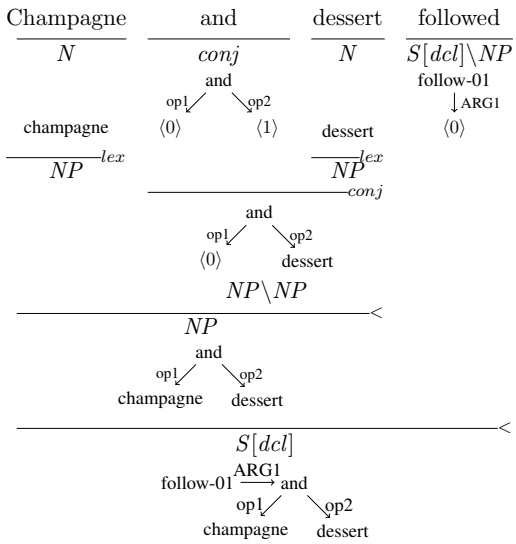

Figure 1: An example graph-algebraic CCG derivation.

parser, which makes it compute-intensive and difficult to manage. We address this issue by performing lexicon induction in a separate step.

Besides AMR parsing, CCG has also been used for joint syntactic-semantic parsing in other contexts (Krishnamurthy and Mitchell, 2014; Lewis et al., 2015).

2 Background

This paper uses Combinatory Categorial Grammar (CCG) to derive Abstract Meaning Representations (AMR) using a graph-algebraic modification of CCG's syntax-semantics interface. These concepts are briefly introduced in this below.

2.1 CCG for Semantic Parsing

Combinatory Categorial Grammar (CCG) describes syntax and semantics as part of the same derivation process (Steedman, 2000). CCG derivations are trees where every node is annotated with both a syntactic and a semantic category. The categories at the leaves of the tree are drawn from a *lexicon*, while categories at the inner nodes result from the application of *combinatory rules* to the child nodes' categories. The syntax-semantics interface in CCG is *transparent*, meaning that the same rule is always applied to syntactic and semantic categories.

In CCG, categories are understood as n-ary *functions*. *Syntactic categories* essentially express the type of the associated semantic category by specifying the types of constituents that can be accepted as arguments, either to the right of the constituent or to the left. This directionality is expressed by forward and backward slashes. E.g., given the atomic syntactic categories S for sentences and NP for noun phrases, the complex category $(S\backslash NP)/NP$ represents a transitive verb, accepting first an NP to the right and then another NP to the left to produce a sentence.

Semantic categories contain building blocks for sentential meaning representations. Traditionally, λ-calculus is used to represent the compositionality of semantic categories, while the object language that is being composed is a logical representation of sentence meaning. This paper deviates from that tradition by using a graph representation for semantic categories which is defined Section 2.3.

2.2 Abstract Meaning Representation

The Abstract Meaning Representation (AMR; Banarescu et al., 2013) is a meaning representation language that underlies much recent work in semantic parsing. In AMR, meaning is annotated on the sentence level in the form of a labeled, directed graph. While the nodes of the graph represent instances of concepts, edges represent roles that these entities play with respect to each other.

2.2.1 Evaluation of AMRs

AMR parsers are usually evaluated with respect to the *Smatch* metric (Cai and Knight, 2013), which measures precision and recall of semantic triples in an AMR graphs with respect to a gold standard graph. The computation of Smatch relies on finding an optimal alignment between the two graphs, which is usually approximated.

2.3 Graph-Algebraic CCG

Graph algebras are an established means to model the derivation of AMRs (Koller, 2015). A modification of CCG that applies graph-algebraic operators to semantic categories has first been presented by Beschke and Menzel (2018). They define a set of semantic operators that apply to *s-graphs*, which contain specially marked *source nodes*, which are consecutively indexed starting from 0. They also define three semantic operators:

- *Application*, which 1) merges the root of an argument graph with the highest-indexed source node of the function graph and 2) merges all source nodes that have the same index.

Combinator	Left Operand	Right Operand	Result
$>$	$X/Y : \bigcirc\!\!\rightarrow \langle 0\rangle$	$Y : \diamond$	$\Rightarrow$ $X : \bigcirc\!\!\rightarrow\!\diamond$
$<$	$Y : \diamond$	$X\backslash Y : \bigcirc\!\!\rightarrow \langle 0\rangle$	$\Rightarrow$ $X : \bigcirc\!\!\rightarrow\!\diamond$
$\mathbf{B}>$	$X/Y : \bigcirc\!\!\rightarrow \langle 0\rangle$	$Y/Z : \diamond\!\!\rightarrow \langle 0\rangle$	$\Rightarrow$ $X/Z : \bigcirc\!\!\rightarrow\!\diamond\!\!\rightarrow \langle 0\rangle$
$\mathbf{B}_\times<$	$Y/Z : \diamond\!\!\rightarrow \langle 0\rangle$	$X\backslash Y : \bigcirc\!\!\rightarrow \langle 0\rangle$	$\Rightarrow$ $X/Z : \bigcirc\!\!\rightarrow\!\diamond\!\!\rightarrow \langle 0\rangle$
$\mathbf{B}^2>$	$X/Y : \bigcirc\!\!\rightarrow \langle 0\rangle$	$(Y/Z_1)/Z_2 : \diamond\!\!\rightarrow \langle 0\rangle,\ \searrow \langle 1\rangle$	$\Rightarrow$ $(X/Z_1)/Z_2 : \bigcirc\!\!\rightarrow\!\diamond\!\!\rightarrow \langle 0\rangle,\ \searrow \langle 1\rangle$
conj	conj : *conj* $\rightarrow \langle 0\rangle,\ \searrow \langle 1\rangle$	$X : \bigcirc\!\!\rightarrow \langle 0\rangle$	$\Rightarrow$ $X\backslash X$: *conj* $\rightarrow \langle 1\rangle,\ \searrow\bigcirc\!\!\rightarrow \langle 0\rangle$
rp	$X : \diamond$	. : ϵ	$\Rightarrow$ X : $\diamond$
lp	. : ϵ	$X : \diamond$	$\Rightarrow$ X : $\diamond$

Table 1: The set of binary combinators used in our system. Circles and diamonds correspond to arbitrary AMR subgraphs. X and Y represent arbitrary syntactic categories. The *conj* node represents any concept corresponding to a conjunction, such as *and* or *contrast*. Edge labels are omitted.
The combinators Forward Application ($>$), Backward Application ($<$), Forward Composition ($\mathbf{B}>$), Backward Crossed Composition ($\mathbf{B}_\times<$), and Forward Generalised Composition ($\mathbf{B}^2>$) all use the *Application* semantic operator. The Conjunction (*conj*) combinator uses the *Conjunction* semantic operator, and Left and Right Punctuation (*lp*, *rp*) use *Identity*.

- *Conjunction*, which 1) merges the root of an argument graph with the 1-indexed source node of the conjunction graph, and 2) renames the 0-indexed source node of the conjunction graph so that it becomes the highest-indexed source node in the combined graph (thus becoming accessible for application).
- *Identity*, in which the function graph is empty and the argument graph is returned unchanged.

An overview of the rules as well as how they are applied in the context of CCG derivations is given in Table 1.

An example derivation is given in Figure 2.

3 Lexicon Induction

For parsing with graph algebraic CCG, a lexicon must first be obtained. We achieve this using the *recursive splitting* algorithm by Beschke and Menzel (2018), which uses the following information to induce lexical items from an AMR-annotated sentence:

- The sentence's AMR
- A syntactic CCG parse obtained from a syntax parser
- A set of alignments linking tokens in the sentence to nodes in the meaning representation, obtained from automatic alignment tools

A set of lexical items explaining the sentence can then be obtained by walking down the syntactic parse tree, starting at the root with the full sentential meaning representation. At each binary derivation step, the meaning representation is partitioned into two subgraphs by unmerging nodes as appropriate. Each split is done in such a way that it can be reversed using a graph algebraic combinator and the token-to-node alignments are honored.

For any token, this procedure may generate several or no lexical entries. If the alignments do not uniquely specify how the meaning representation should be divided in a splitting step, all alternatives are explored. Also, splitting may abort at an inner node of the derivation if there is no combinator that satisfies the alignment constraint.

This work adds two steps to the lexicon induction process: the delexicalisation of lexical items, followed by filtering for the most probable derivation for each sentence according to EM estimates.

3.1 Delexicalisation

We achieve generalisation over content words by delexicalising lexical entries. We follow the ap-

proach from Kwiatkowski et al. (2011) which divides lexical entries into *templates* and *lexemes*. A template is a graph wherein up to one node has been replaced by a *lex* marker. A lexeme x—y is a pair of a word x and a node label y. For examples of templates and lexemes, see Table 2.

The idea of the delexicalisation algorithm is that a node in the graph which corresponds to the lexical meaning of the lexical entry is replaced by a marker, converting it into a template. Since it is not known in advance which node carries the lexical meaning, we replace every node in turn and add all resulting templates to the lexicon. Every replaced node label is associated with the token currently under consideration and stored as a lexeme.

Not all lexical entries contain a node with lexical meaning, e.g. in the case of function words. Therefore, the original meaning representation is also added to the lexicon as a template along with an empty lexeme.

Special lexemes are also added that map any word to a node labeled by the word's lemma, its surface form in quotes, or any of the propbank frame names associated with its lemma.

This process creates a large amount of superflous template/lexeme pairs. Therefore, the lexicon is subsequently filtered using Expectation Maximisation.

3.2 Expectation Maximisation

Both splitting and delexicalisation generate spurious templates and lexemes. We wish to keep only those that generalise well by being broadly applicable. In contrast, noise introduced during grammar induction should be removed.

This noise manifests itself in spurious derivations for the sentences of the training set. Expectation Maximisation (EM) is applied to identify a single most likely derivation per sentence. Every template and lexeme that in not used in at least one of these derivations is deleted.

We use a variant of the inside-outside algorithm (Baker, 1979) to estimate multinomial distributions P_t for templates and P_l for lexemes. From these, we derive a probability distribution over derivations:

$$P(d) = \prod_{(t,l) \in \text{LEX}(d)} P_t(t) P_l(l)$$

where $\text{LEX}(d)$ gives all template-lexeme pairs instantiated by the derivation.

Algorithm 1 Variation of the inside-outside algorithm to estimate parameters over CCG derivations. See Section 3.2 for function definitions.

Input: Data set S; scoring function SCORE^i
Output: Distributions P_T^{i+1} and P_L^{i+1}

$\mathit{count}_T[j] \leftarrow 0$ for $0 \leq j < |T|$
$\mathit{count}_L[j] \leftarrow 0$ for $0 \leq j < |L|$
for $s \in S$ **do**
  $\mathit{chart} \leftarrow \text{SPLIT}(s)$
  $\mathit{likelihood} \leftarrow \sum_{e \in \mathit{chart}[0,|s|-1]} \text{IN}^{i+1}(e)$
  for $e \in \mathit{chart}$ **do**
    $c \leftarrow \frac{\text{SCORE}^i(e)\text{OUT}^{i+1}(e)}{\mathit{likelihood}}$
    for $(t,l) \in \text{DELEX}(e)$ **do**
      $\mathit{count}_T[t] \leftarrow \mathit{count}_T[t] + c$
      $\mathit{count}_L[l] \leftarrow \mathit{count}_L[l] + c$
    end for
  end for
end for
$P_T^{i+1}(t) = \frac{\mathit{count}_T[t]}{\sum_{t' \in T} \mathit{count}_T[t']|}$ for $t \in T$
$P_L^{i+1}(l) = \frac{\mathit{count}_L[l]}{\sum_{l' \in L} \mathit{count}_L(l')|}$ for $l \in L$

Our inside-outside algorithm operates on *split charts*, which keep track of all derivation nodes created during recursive splitting. A split chart c for a sentence s contains cells $c[i,j]$ with $0 < i \leq j, j < |s|$. A cell contains a number of entries e, each of which is associated with a meaning representation $\text{MR}(e)$ and a (possibly empty) set of child pairs $(l,r) \in \text{CLD}(e)$, which are in turn entries. An entry can also have several parents $e' \in \text{PAR}(e)$, in which case it also has a neighbour $\text{NB}(e,e')$ for every parent e'.

To compute a probability for an entry, we employ a function DELEX which decomposes the entry's meaning representation into all possible template-lexeme pairs.

Inside and outside probabilities for entries are calculated recursively as follows:

$$\text{IN}^{i+1}(e) = \text{SCORE}^i(e) + \sum_{(l,r) \in \text{CLD}(e)} \text{IN}^{i+1}(l) \cdot \text{IN}^{i+1}(r)$$

$$\text{OUT}^{i+1}(e) = \sum_{e' \in \text{PAR}(e)} \text{OUT}^{i+1}(e') \cdot \text{IN}^{i+1}(\text{NB}(e,e'))$$

where

$$\textsc{Score}^i(e) = \sum_{(t,l) \in \textsc{Delex}(\textsc{MR}(e))} P_T^i(t) P_L^i(l)$$

A given meaning representation $\textsc{MR}(e)$ can be created by either instantiating a lexical entry with probability $\textsc{Score}^i(e)$, or by deriving it using any of its pairs of children (l, r) with probability $\textsc{In}^i(l)\textsc{In}^i(r)$. All of these are alternative choices; therefore, the probabilities are summed to make up the inside probability. The outside probability is composed of the entry's parents' outside probabilities and the entry's neighbours' inside probabilities.

Algorithm 3.2 describes how an updated set of parameters is estimated using these calculations.

4 Parsing

Our parser uses a CKY-style chart parsing algorithm to parse sentences to AMR. For each token, template-lexeme pairs are drawn from the lexicon. Recursively, derivation nodes are created according to CCG/AMR rules. All candidate derivation nodes are evaluated with respect to a linear model. A beam search limitation is applied, meaning that only the top n candidates from each chart cell are kept.

The flip side of using a delexicalised lexicon is that every template can now be applied to every token. To limit the number of leaves that have to be considered, we employ a supertagger which predicts the most suitable template for each token. We then limit our search to the most probable templates as predicted by the supertagger.

4.1 Supertagging

For supertagging, we use a single-layer BiLSTM. For inputs, the raw tokens and syntactic CCG categories predicted by a CCG supertagger are used. The model is then trained to predict the template instantiated by each token.

The following preprocessing steps are applied:

- Tokens are embedded using the third layer produced by ELMo (Peters et al., 2018).
- CCG supertags, as well as templates, that occur in less than two sentences are replaced by `UNK`.

To predict supertags on the *dev* and *test* sections, we train the supertagger on the entire *train* section and output the predicted token-wise supertag distributions (clipping at 99% cumulative probability). To obtain supertag predictions on the *train* section, we employ 5-way jackknifing: the data is split into five parts and predictions for each part are obtained by training on the remaining four parts.

During training, the occurrence of the correct label within the top-10 predictions for every token is monitored and training aborted when this measure stops improving (early stopping).

4.1.1 Limited Supervision

The grammar induction process as described in Section 3 attempts to find lexical items for every individual token, but may stop early if no combinatory rule fitting the alignment constraint is available. In this case, no supervision for training the tagger is available at the token level. We overcome this issue by labelling the respective tokens as `UNK` (the same label used for rare templates occurring only once) and masking `UNK` tokens in the loss function.

This allows the tagger to fill in the gaps with reasonable templates that are in the lexicon. However, it also means that not every sentence from the train set can be perfectly parsed any more, because it is possible that its meaning representation cannot be constructed using the induced token-level lexical entries.

5 Training

To drive the parser, we train a linear model over graph algebraic CCG derivations. Since we do not observe derivations in the data, this is an instance of latent variable learning and a supervision signal must be generated. We take a dual approach by combining two weak supervision signals:

1. An oracle is used to heuristically generate silver-standard derivations, which can then be used for training.
2. The derivations found by the parser are evaluated and used for cost-sensitive parameter updates.

5.1 Model

We train a linear model using a structured perceptron algorithm (Collins, 2002) with Adadelta updates (Zeiler, 2012). We use features over paths in the graph as well as the identities of invoked templates, lexemes, and combinators.

Template	Lexeme	Combined
N : *lex*	weapons—weapon	N : weapon
NP/N : $\langle 0 \rangle$	the—$\emptyset$	NP/N : $\langle 0 \rangle$
N/N : $\langle 0 \rangle \xrightarrow{\text{mod}}$ *lex*	nuclear—nucleus	N/N : $\langle 0 \rangle \xrightarrow{\text{mod}}$ nucleus
$(S\backslash NP)/NP$: *lex* $\xrightarrow{\text{ARG0}}$ $\langle 0 \rangle$, $\xrightarrow{ARG1}$ $\langle 1 \rangle$	said—say-01	$(S\backslash NP)/NP$: say-01 $\xrightarrow{\text{ARG0}}$ $\langle 0 \rangle$, $\xrightarrow{ARG1}$ $\langle 1 \rangle$
NP : country → name → name → op1 → *lex*	Iran—"Iran"	NP : country → name → name → op1 → "Iran"

Table 2: Selected templates and lexemes from the induced lexicon. The templates are among the 20 most highly scored according to EM parameters; the lexemes among the top 50.

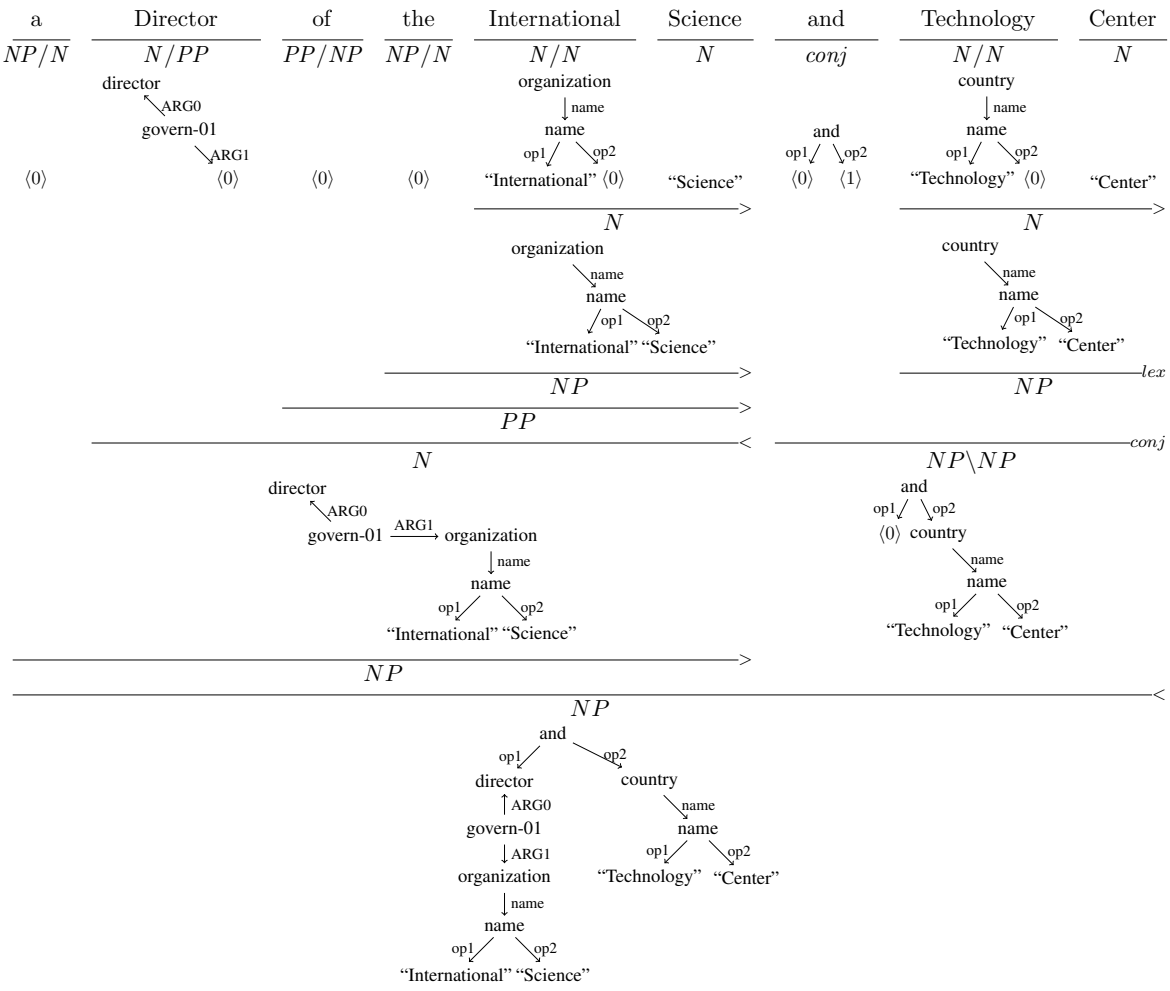

Figure 2: A derivation for a subsequence of `PROXY_NYT_ENG_20020406_0118.25`, as produced by our parser.

5.2 Oracle Parsing

In latent variable learning in structured prediction settings, the challenge is to obtain an unobserved derivation for a known gold-standard result. In this case, gold-standard sentential AMRs are annotated in the AMR corpus, but they are not related to the sentence by a grammatical derivation.

A common approach to this challenge is *forced decoding* (Artzi et al., 2015): the parser is used to construct derivations which lead to the correct result by pruning all hypotheses from the search space which deviate from the gold-standard AMR. E.g., all AMRs that contain elements not present in the gold-standard could be pruned.

However, as noted in Section 4.1, not every gold-standard AMR can be reconstructed perfectly using the induced lexicon due to the incompleteness of the splitting algorithm, which defies finding correct parses using forced decoding.

Instead, we train the parser using an oracle driven by a heuristic scoring function which scores the correctness and completeness of an intermediate hypothesised AMR. We parse the sentence using CKY with beam search, ranking intermediate results according to the harmonic mean of the following values:

- **Triple precision:** the proportion of node-edge-node triples in the intermediate result that also occur in the gold standard meaning representation.
- **Alignment recall:** the proportion of node labels that are linked by an alignment edge to one of the intermediate result's tokens that also occur in the intermediate result.

This scoring function is designed to rank results in proportion to their deviation from the gold standard, achieving a soft form of pruning.

Having obtained a set of derivations using oracle parsing, we finally re-rank these derivations by their Smatch f1 scores and use the best derivation to perform a parameter update using an early update strategy (Collins and Roark, 2004).

5.3 Cost-Sensitive Update

Another approach to training with weak supervision for structured prediction are cost-sensitive updates. While the gold-standard to update towards is unknown, an evaluation metric is available for the AMR that results from a specific derivation. Cost-sensitive updates let the parser search for complete derivations and enforce a margin between the best derivations in the beam and all the others. We follow Singh-Miller and Collins (2007) by implementing a cost-sensitive perceptron algorithm which weights hypotheses according to their Smatch f1 score.

5.4 Combined Update Strategy

While early updates are efficient, our oracle is imperfect. To allow the parser to improve over oracle parses, we use a cost-sensitive update whenever a parse has been found whose Smatch f1 score surpasses that of the oracle parse.

6 Experimental Setup

We evaluate our parser[1] on the *proxy* section of the AMR 1.0 corpus (LDC2014T12; Knight et al., 2014). This section consists of newswire texts.

Sentences are tokenised and lemmatised using Stanford NLP (Manning et al., 2014). We use EasyCCG to obtain CCG parses and supertags (Lewis and Steedman, 2014). Token-to-AMR alignments are obtained by combining outputs generated by the JAMR aligner (Flanigan et al., 2014) and the ISI aligner (Pourdamghani et al., 2014), as described by Beschke and Menzel (2018).

First, we induce a CCG/AMR lexicon from the entire *proxy-training* section, delexicalise the entries, and filter for the best derivations using EM, as described in Section 3. We perform 100 iterations of EM. Sentences longer than 100 tokens are filtered out. The resulting lexicon contains 15630 templates and 10504 lexemes.

Next, we extract template tag sequences and train our suppertagger on them. First, tags for the training data are predicted using 5-way jackknifing. Then, a model is trained on the entire training section and used to predict tags for the *dev* and *test* sections of the corpus. Since only templates are predicted that occur in at least two training sentences, a set of 2453 templates is used for prediction. The top-10 recall of the annotated supertags is 96.4% on a randomly chosen held-out portion of the training set.

Finally, the induced lexicon as well as the predicted tag sequences are used to parse the *proxy-test* section of the AMR corpus. We use a beam

[1] For information on reproducing the experiments, see `https://gitlab.com/nats/gramr-ranlp19/`.

System	P	R	F
This paper	0.688	0.423	0.524
Artzi et al. (2015)	0.668	0.657	0.663
Misra and Artzi (2016)	0.681	0.642	0.661
Liu et al. (2018)	-	-	**0.731**

Table 3: Smatch results on the *proxy-test* section of LDC2014T12. Liu et al. (2018) did not report precision and recall in their paper. P stands for *precision*, R for *recall*, F for *f1 score*.

size of 15 during parsing and 20 for finding oracle derivations (see Section 5.2). Parses whose root categories do not match any of the top-10 derivations produced by EasyCCG are dropped from the parser output[2].

The `smatch` tool[3]is used to calculate Smatch precision, recall, and f1 scores for the parser output.

7 Results

We compare our system to two previous CCG-based AMR parsers (Artzi et al., 2015; Misra and Artzi, 2016), as well as the current state of the art in AMR parsing on this data set (Liu et al., 2018). The results are shown in Table 7. The system introduced in this paper achieves comparable precision to the other CCG-based systems, but lower recall.

This gap is largely, but not completely, explained by sentences that were not parsed at all: when unparsed sentences are excluded from the evaluation, our system achieves a precision of 0.701 and a recall of 0.615[4]. Oracle parsing achieves a Smatch precision of 0.886 and an f1 score of 0.706.

The evaluation set contains 823 sentences in total, of which 170 were not parsed, resulting in a coverage of 79%. Of these sentences, 68 were skipped because they were longer than 40 tokens. The remaining 102 are unparsed because the parser failed to find a complete parse.

[2]This restriction was included because the parser tended to favour interpretations of sentences as *NP* instead of *S*.

[3]`https://github.com/snowblink14/smatch`, revision `ad7e655`

[4]The precision improves when unparsed sentences are excluded because the `smatch` tool does not permit empty AMRs to be specified. Unparsed sentences are therefore represented by single-node placeholder AMRs, which are penalised in terms of precision.

7.1 Discussion

The parser output in Figure 5 shows some of the most common errors produced by our parser. Firstly, the sequence *International Science and Technology Center* is not recognised as a contiguous named entity. Additionally, *Technology Center* is misrecognised as a country. Both of these issues can be classified as supertagging errors, as they result from the templates chosen from the lexicon. In this specific case, the supertagger's behaviour could likely be improved by adding named entity features to its input. In general, the supertagging task is challenging, especially in the case of function words, which tend to be highly polysemous.

Additionally, the scopes of *and* and *of* are inverted. This can be interpreted as a weakness of the parsing model, which misjudges the probability of the respective scope assignments. Although one would hope for a semantic parser to improve precisely upon these semantically informed syntactic decisions, this behaviour is perhaps to be expected given that we train a sparse linear model with a relatively small amount of training data. Replacing the linear classifier with a neural model that computes embeddings of graph meanings, such as the architecture proposed by (Misra and Artzi, 2016), could improve the parser's judgment.

8 Conclusion

We have introduced a pipeline for training a CCG parser which jointly models syntax and semantics. A central element of our architecture are efforts to reduce the lexicon size. With 2453 delexicalised templates, our parser uses a relatively small lexicon despite the templates being induced automatically. We employ a semantic construction mechanism that is less powerful with λ-calculus, but still achieve competitive precision.

Future directions in this line of work could include applications that make use of the system's transparency, such as the interactive training of parsers without gold-standard annotations, or the application of external constraints such as contextual knowledge to the parser.

Acknowledgments

The author would like to thank Wolfgang Menzel for discussions and advice, as well as the anonymous reviewers.

References

Yoav Artzi, Kenton Lee, and Luke Zettlemoyer. 2015. Broad-coverage CCG Semantic Parsing with AMR. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Lisbon, Portugal, pages 1699–1710. http://aclweb.org/anthology/D15-1198.

J. K. Baker. 1979. Trainable grammars for speech recognition. *The Journal of the Acoustical Society of America* 65(S1):S132–S132. https://doi.org/10.1121/1.2017061.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract Meaning Representation for Sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*. Association for Computational Linguistics, Sofia, Bulgaria, pages 178–186. http://www.aclweb.org/anthology/W13-2322.

Sebastian Beschke and Wolfgang Menzel. 2018. Graph algebraic combinatory categorial grammar. In *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics*. Association for Computational Linguistics, New Orleans, Louisiana, pages 54–64. https://doi.org/10.18653/v1/S18-2006.

Austin Blodgett and Nathan Schneider. 2019. An improved approach for semantic graph composition with CCG. In *Proceedings of the 13th International Conference on Computational Semantics - Long Papers*. Association for Computational Linguistics, Gothenburg, Sweden, pages 55–70. https://www.aclweb.org/anthology/W19-0405.

Shu Cai and Kevin Knight. 2013. Smatch: an Evaluation Metric for Semantic Feature Structures. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, Sofia, Bulgaria, pages 748–752. http://www.aclweb.org/anthology/P13-2131.

Michael Collins. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 1–8. https://doi.org/10.3115/1118693.1118694.

Michael Collins and Brian Roark. 2004. Incremental Parsing with the Perceptron Algorithm. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*. Barcelona, Spain, pages 111–118. https://doi.org/10.3115/1218955.1218970.

Jeffrey Flanigan, Sam Thomson, Jaime Carbonell, Chris Dyer, and Noah A. Smith. 2014. A Discriminative Graph-Based Parser for the Abstract Meaning Representation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Baltimore, Maryland, pages 1426–1436. http://www.aclweb.org/anthology/P14-1134.

Jonas Groschwitz, Matthias Lindemann, Meaghan Fowlie, Mark Johnson, and Alexander Koller. 2018. AMR dependency parsing with a typed semantic algebra. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Melbourne, Australia, pages 1831–1841. http://aclweb.org/anthology/P18-1170.

Kevin Knight, Laura Baranescu, Claire Bonial, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Daniel Marcu, Martha Palmer, and Nathan Schneider. 2014. *Abstract Meaning Representation (AMR) Annotation Release 1.0 LDC2014T12*. Linguistic Data Consortium, Philadelphia. https://catalog.ldc.upenn.edu/LDC2014T12.

Alexander Koller. 2015. Semantic construction with graph grammars. In *Proceedings of the 11th International Conference on Computational Semantics*. Association for Computational Linguistics, London, UK, pages 228–238. http://www.aclweb.org/anthology/W15-0127.

Jayant Krishnamurthy and Tom M. Mitchell. 2014. Joint Syntactic and Semantic Parsing with Combinatory Categorial Grammar. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Baltimore, Maryland, pages 1188–1198. https://doi.org/10.3115/v1/P14-1112.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2011. Lexical generalization in CCG grammar induction for semantic parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. pages 1512–1523. http://dl.acm.org/citation.cfm?id=2145593.

Mike Lewis, Luheng He, and Luke Zettlemoyer. 2015. Joint A* CCG Parsing and Semantic Role Labelling. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Lisbon, Portugal, pages 1444–1454. http://aclweb.org/anthology/D15-1169.

Mike Lewis and Mark Steedman. 2014. A* CCG Parsing with a Supertag-factored Model. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, pages 990–1000. http://www.aclweb.org/anthology/D14-1107.

Yijia Liu, Wanxiang Che, Bo Zheng, Bing Qin, and Ting Liu. 2018. An AMR Aligner Tuned by Transition-based Parser. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Brussels, Belgium, pages 2422–2430. https://www.aclweb.org/anthology/D18-1264.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*. pages 55–60. http://www.aclweb.org/anthology/P/P14/P14-5010.

Kumar Dipendra Misra and Yoav Artzi. 2016. Neural Shift-Reduce CCG Semantic Parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Austin, Texas, pages 1775–1786. http://aclweb.org/anthology/D16-1183.

Xiaochang Peng, Daniel Gildea, and Giorgio Satta. 2018. AMR Parsing with Cache Transition Systems. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*. Association for the Advancement of Artificial Intelligence, New Orleans, USA. https://www.cs.rochester.edu/u/gildea/pubs/peng-aaai18.pdf.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep Contextualized Word Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Association for Computational Linguistics, New Orleans, Louisiana, pages 2227–2237. https://doi.org/10.18653/v1/N18-1202.

Nima Pourdamghani, Yang Gao, Ulf Hermjakob, and Kevin Knight. 2014. Aligning English Strings with Abstract Meaning Representation Graphs. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, pages 425–429. http://www.aclweb.org/anthology/D14-1048.

Natasha Singh-Miller and Michael Collins. 2007. Trigger-Based Language Modeling Using a Loss-Sensitive Perceptron Algorithm. *IEEE ICASSP* .

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

Matthew D Zeiler. 2012. ADADELTA: An Adaptive Learning Rate Method. *arXiv preprint arXiv:1212.5701* .

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019. AMR Parsing as Sequence-to-Graph Transduction. https://arxiv.org/abs/1905.08704.