Graph Algebraic Combinatory Categorial Grammar

Sebastian Beschke **Wolfgang Menzel**
Department of Informatics
University of Hamburg
Vogt-Kölln-Straße 30, 22527 Hamburg, Germany
{beschke,menzel}@informatik.uni-hamburg.de

Abstract

This paper describes CCG/AMR, a novel grammar for semantic parsing of Abstract Meaning Representations. CCG/AMR equips Combinatory Categorial Grammar derivations with graph semantics by assigning each CCG combinator an interpretation in terms of a graph algebra.

We provide an algorithm that induces a CCG/AMR from a corpus and show that it creates a compact lexicon with low ambiguity and achieves a robust coverage of 78% of the examined sentences under ideal conditions.

We also identify several phenomena that affect any approach relying either on CCG or graph algebraic approaches for AMR parsing. This includes differences of representation between CCG and AMR, as well as non-compositional constructions that are not expressible through a monotonic construction process. To our knowledge, this paper provides the first analysis of these corpus issues.

1 Introduction

With the release of the Abstract Meaning Representation (AMR) corpus (Knight et al., 2014), graph representations of meaning have taken centre stage in research on semantic parsing. Semantic parsing systems have to address the problem of *lexicon induction*: extracting reusable lexical items from the sentential meaning representations annotated in the AMR corpus. Since the corpus contains sentential meaning representations, but no indication of their compositional structure, derivations of the meaning representation cannot be observed. Common approaches enumerate all conceivable lexical items that may have contributed to the derivation of the meaning representation at hand (Artzi et al., 2015; Groschwitz et al., 2017). This may produce very large lexicons with high degrees of ambiguity, and therefore require large amounts of computational resources during parsing. One central contribution of this paper is a lexicon induction algorithm that produces a relatively compact lexicon.

Combinatory Categorial Grammar (CCG) uses a transparent syntax-semantic interface that constructs meaning representations using λ-calculus. This makes lexicon induction challenging, as inducing λ-calculus terms essentially requires solving higher-order unification (Kwiatkowski et al., 2010). In practice, heuristics are employed to manage the search space, but it would be preferable to make use of a less powerful mechanism which better fits the problem domain. Graph algebras such as the HR algebra (Courcelle and Engelfriet, 2012) constitute such a constrained mechanism. By combining CCG with the HR algebra, we are able to leverage the availability of syntactic parsers for CCG while making use of tailored semantic construction operations.

1.1 Related Work

There is an extensive body of work on semantic parsing of AMRs, using a range of techniques including maximum spanning tree-based parsing (Flanigan et al., 2014), transition-based parsing (Wang et al., 2015; Ballesteros and Al-Onaizan, 2017; Peng et al., 2018), machine translation (van Noord and Bos, 2017), graph grammars (Peng et al., 2015; Groschwitz et al., 2017), and CCG parsing (Artzi et al., 2015; Misra and Artzi, 2016).

The system of Artzi et al. (2015) is most similar to the present work. They induce a semantic CCG using a translation of AMR into λ-calculus. A key difference is that their training algorithm combines lexicon induction and parameter training into a single phase. New lexical items are generated during each training step and then filtered based upon the model's current parameters. In contrast, in this work we focus on lexicon in-

*Proceedings of the 7th Joint Conference on Lexical and Computational Semantics (*SEM)*, pages 54–64
New Orleans, June 5-6, 2018. ©2018 Association for Computational Linguistics

duction, with parameter training to be performed in a subsequent step.

Another related system is presented in Lewis et al. (2015), where a CCG parser is adapted to produce shallow semantic dependency graphs. In contrast, the meaning representations employed here are abstract and do not directly refer to the sentence under analysis.

Word-to-node alignments on the AMR corpus play an important role in this work. We experiment with JAMR's rule-based aligner (Flanigan et al., 2014) and the statistical ISI aligner (Pourdamghani et al., 2014).

Graph algebras have recently been applied to AMR parsing (Koller, 2015; Groschwitz et al., 2017), but not in combination with CCG. In contrast, we use syntactic CCG derivations to constrain the space of possible derivations. However, the idea of using a constrained version of the HR algebra, introduced by Groschwitz et al. (2017), is also used here, and our *Application* operator effectively subsumes their *Apply* and *Modify* operations.

1.2 Tools

Syntax parser We use EasyCCG (Lewis and Steedman, 2014) to obtain syntax derivations. For robustness, we extract the ten best derivations produced by EasyCCG based on the CCGBank-rebanked model (Honnibal et al., 2010).

Word-to-node alignments During lexicon induction, we make use of alignments between tokens in the sentence and nodes in the meaning representation. We experiment with JAMR's aligner (Flanigan et al., 2014) and the ISI aligner (Pourdamghani et al., 2014).

Other tools We use Stanford CoreNLP (Manning et al., 2014) for tokenisation.

2 Background

The task of semantic parsing is concerned with building formal meaning representations for natural language text. While meaning representations can be elements of any formal language, in this paper we are concerned with Abstract Meaning Representations (AMRs). We use Combinatory Categorial Grammar (CCG) as an underlying framework to explain how AMRs may be derived from the surface form words. To do so, we equip CCG with graph construction operators drawn from the HR algebra. These concepts are introduced below.

2.1 Combinatory Categorial Grammar

CCG is a grammar formalism centered around a transparent syntax-semantics interface (Steedman, 2000). A CCG consists of a small set of combinatory rules, along with a lexicon of entries defining each word's syntactic and semantic interpretation.

A CCG lexical item, as used in this paper, contains one or several tokens, a syntactic category, and a semantic category. The syntactic category is a functional type defining the types of arguments expected by the words and whether they are expected to the left or right. E.g., *NP/N* expects a noun (*N*) to the right (because of the rightward-facing slash) and returns an *NP* – it is the type of determiners. *(S \NP) / NP* is the category of transitive verbs, consuming first an *NP* from the right and then from the left, and returning a sentence. See Figure 1a for an example.

CCG derivations are created by recursively applying combinators to the lexical syntactic categories, thus combining them into constituents. Besides Application, implementations of CCG also use other combinators such as Composition, as well as specialized combinators for conjunctions and punctuation.

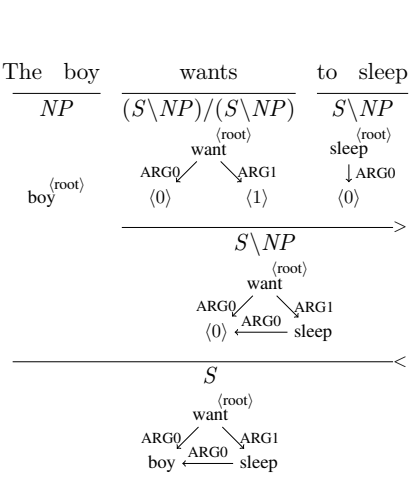
Semantic categories are represented as λ-calculus terms. A combinator is always applied to two constituents' syntactic and semantic categories at the same time, allowing semantic construction to be fully syntax-driven.

2.2 Abstract Meaning Representation

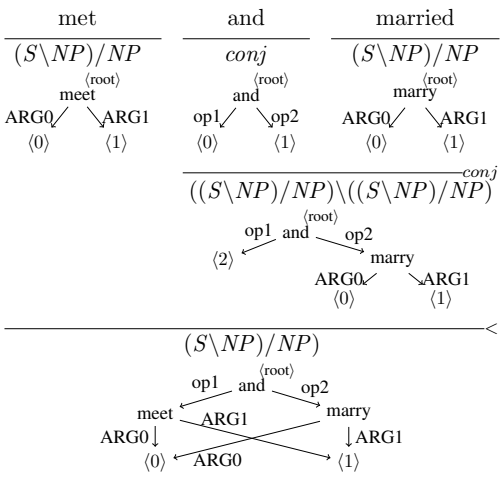
The Abstract Meaning Representation (AMR) is a semantic meaning representation language that is purposefully syntax-agnostic (Banarescu et al., 2013). The data set used in this paper, the AMR 1.0 release (Knight et al., 2014), consists of English sentences which have been directly annotated with meaning representations by human annotators.

AMR represents meaning as labeled, directed graphs. Nodes are labeled with concepts, while edges represent roles. Predicates and core roles are drawn from PropBank (Kingsbury and Palmer, 2002). In addition, a set of non-core roles has been defined, such as *mod*, *poss*, *time*, etc.

Whether it was wise to define AMR independently of any derivational process has been debated (Bender et al., 2015), and in Section 5 we will show some of the issues that arise when attempting to construct derivations for AMRs.



(a) Example for a derivation making use of two *Application* operations.
Step 1: Application. Insert *sleep* node into placeholder ⟨1⟩; merge ⟨0⟩ placeholders.
Step 2: Application. Insert *boy* into placeholder ⟨0⟩.



(b) Example for a derivation containing one *Conjunction* and one *Application* operation.
Step 1: Conjunction. Insert *marry* node into placeholder ⟨1⟩ of *and*; shift placeholder ⟨0⟩ of *and* to ⟨2⟩.
Step 2: Application of the phrase *and married* to *met*; the operands' same-numbered placeholders are merged.

Figure 1: Examples for the semantic operations *Application* and *Conjunction*.

2.3 The HR Algebra

During semantic parsing, a complete meaning representation needs to be built out of smaller components. To formalise this process, known as semantic construction, an algebra may be defined that describes the permitted operations on meaning representations. The HR algebra has first been defined in the context of graph grammars (Courcelle and Engelfriet, 2012) and has been applied to semantic construction of AMR graphs (Koller, 2015).

The HR algebra operates on *graphs with sources*, or *s-graphs*. Given a set of labels $\mathcal{A}$, an s-graph is a pair $(G, slab_G)$ where $G = (V_G, E_G)$ is a graph and $slab_G : V_G \to \mathcal{A}$ is a partial injective function. The nodes contained within the domain of $slab_G$ are called the *sources* of G, and the elements of $\mathcal{A}$ are called *source labels*.

The HR algebra defines three basic operations over s-graphs:

- **Parallel composition** creates the union of two disjoint s-graphs and fuses nodes that share a source label. $slab_G$ is defined to retain all source labels.
- **Forgetting** removes a source from the domain of $slab_G$, effectively deleting its label.
- **Renaming** modifies $slab_G$ to change source labels according to a specified mapping.

The HR algebra provides the building blocks for the manipulation of s-graphs. In the following section, we apply these basic operations to CCG.

3 Graph Semantics for CCG

In CCG, semantic construction is syntax-driven in that the construction operations that are applied to lexical units of meaning are determined by the combinatory operations that make up a CCG syntax tree. To define CCG derivations over graph semantics, we must therefore define the semantic construction operators invoked by each CCG combinator.

We will use a restricted version of the HR algebra, where we only consider a subset of possible s-graphs, and only some of the possible operations. To represent incomplete meaning representations, we define *CCG/AMR s-graphs*.

3.1 CCG/AMR S-Graphs

As an extension of AMR graphs, CCG/AMR s-graphs may contain unlabeled nodes called *placeholders*. In addition to the AMR labels, a source labelling function is employed, as defined in Section 2.3. Source labels used in CCG/AMR s-graphs take one of two forms:

- $\langle r, i \rangle$ with $r \in \{\text{root}, \emptyset\}$ and $i \in \mathbb{N} \cup \{\emptyset\}$, which marks a node as root if $r = \text{root}$, and assigns it an index i if $i \in \mathbb{N}$. However, we disallow $\langle \emptyset, \emptyset \rangle$ as a source label.
- $\langle \text{s} \rangle$, used temporarily to label a placeholder-argument pair.

Sources with $r = \text{root}$ are called *root-sources*, and sources with $i \in \mathbb{N}$ are called *i-sources*. For

CCG combinator	Semantic operator
(F/B) Application	(F/B) Application
(F/B) [Gen.] [Cr.] Comp.	(F/B) Application
Conjunction	F Conjunction
(Left/Right) Punctuation	(F/B) Identity

Table 1: The mapping from CCG combinators to semantic operators. *F* and *B* stand for *Forward* and *Backward*, *Gen.* stands for *Generalised*, *Cr.* stands for *Crossed*, and *Comp.* stands for composition. All variants of Composition are mapped to the Application operation.

simplicity, we abbreviate $\langle \text{root}, \emptyset \rangle$ to $\langle \text{root} \rangle$ and $\langle \emptyset, i \rangle$ to $\langle i \rangle$ for $i \in \mathbb{N}$.

The following constraints apply:

- A CCG/AMR s-graph must have exactly one root-source.
- For every $i \in \mathbb{N}$, there may be at most one i-source, and for every i-source with $i > 0$, there must also be an $(i-1)$-source.
- The i-sources of a CCG/AMR s-graph must be exactly its placeholders.

The *outermost placeholder* of a CCG/AMR s-graph is defined to be the placeholder with the highest index i.

3.2 Semantic Operators

We now define three semantic operators based on the building blocks of the HR algebra. They are binary operators, with a left and a right operand. If the operator is used in *forward* direction, we call the left operand the *function graph* and the right operand the *argument graph*. In *backward* direction, these roles are reversed.

- **Application** Relabels both the outermost placeholder of the function graph and the root of the argument graph to $\langle \text{s} \rangle$. Then performs parallel composition of both graphs. Finally, forgets $\langle \text{s} \rangle$. Requires the function graph to have at least one placeholder.
- **Conjunction** The function graph of a conjunction operator is required to have exactly two placeholders. Placeholder $\langle 1 \rangle$ and the root of the function graph are both renamed to $\langle \text{s} \rangle$. Placeholder $\langle 0 \rangle$ of the function graph is relabelled to $\langle i+1 \rangle$, where i is the index of the argument graph's outermost placeholder. Then, parallel composition is applied to the relabelled graphs and $\langle \text{s} \rangle$ forgotten.
- **Identity** A special case of Application where the function graph must consist of a single placeholder and the argument graph is therefore always returned unchanged.

Examples for the Application and Conjunction operators are given in Figure 1.

For our definition of CCG/AMR, we use the combinator set of the EasyCCG parser (Lewis and Steedman, 2014), which is small and achieves good coverage. The Application operator is sufficient to cover almost all CCG combinators, with the exception of conjunctions. The mapping of CCG combinators to semantic operators used in this paper is summarised in Table 1.

All unary combinators, including type raising and the various type-changing rules used by EasyCCG, are defined to have no effect on the semantic representation.

We add a single rule that is non-compositional in terms of the semantic operators. Since n-ary conjunctions are frequent in the AMR corpus, this rule combines two nested conjunction nodes, merging their operands into a single contiguous operand list.

3.3 Induction of CCG/AMR Lexicons

To induce a CCG/AMR lexicon, we propose a simple recursive algorithm, described in Algorithm 1. It starts with a full-sentence meaning representation, a syntax tree, and a set of word-to-node alignments. Starting from the root, it recurses down the syntax tree and splits the meaning representation into smaller parts by applying the inverse of one of the semantic operators at each node.

Constraints We impose two constraints on the generated lexical items:

1. The alignments must not be violated.
2. The number of placeholders must not exceed the arity of the lexical item's syntactic category. E. g., the meaning representation for the syntactic category $(S\backslash NP)/NP$ must not have more than two placeholders.

Soundness Algorithm 1 requires finding z_1, z_2 with $o(z_1, z_2) = z$. This equation states that a parser would be able to produce the parse observed in the data based on the induced the lexical items, thus ensuring the soundness of the induction algorithm.

Implementation We examine all ways of par-

titioning the meaning representation into a function subgraph and an argument subgraph, provided that no alignments are violated. Nodes that sit at the boundary between both subgraphs – belonging to the argument subgraph but connected by edges to the function subgraph – are *unmerged*, meaning that a placeholder is created in the function subgraph to which those edges are moved.

Phrasal items Intermediate items are emitted by the algorithm even if they can be further decomposed. The rationale behind this behaviour is that some lexical entries legitimately span multiple tokens. E.g., one could argue that a named entity such as *New York* should be kept as a lexical item.

Coreferences Since there are many cases where graphs are connected more densely than allowed by the arity constraint, we allow <coref> nodes to be created by unmerging additional nodes. They are treated just like regular nodes. In particular, they are not sources and do not count towards the function graph's placeholder count. In the experiments in this paper, we only allow creation of a single <coref> node per splitting step.

Splitting failures The algorithm may encounter situations where splitting cannot continue because it is impossible to further decompose the meaning representation without violating one of the constraints. In such cases, its output is incomplete, emitting a lexical item only for the derivation node at which the problem was encountered – which may span a long constituent of the sentence – but not for any of its sub-nodes.

4 Analysis of Lexicon Induction

We begin by analysing the statistical properties of large-scale grammar induction on the 6,603 sentences of the `proxy-train` subset of the AMR corpus, which consists of newswire texts. We also examine the influence that different alignment strategies have on the produced lexicon.

We then turn to the `consensus-dev` subset, which consists of 100 sentences taken from the Wall Street Journal corpus. Therefore, gold standard syntax parses for these sentences can be obtained from CCGBank (Hockenmaier and Steedman, 2007). In addition, Pourdamghani et al. (2014) have released gold-standard word-to-meaning alignments for these sentences. This allows us to examine the effect of tool-derived alignments and syntax parses on the induction algorithm's behaviour.

Algorithm 1 Recursive Splitting

Input: syntax derivation node y, CCG/AMR s-graph z

Definitions: OPERATOR returns the semantic operator matching a derivation node's combinator. CHILDREN returns the sub-nodes of a syntax derivation node. VALID tests whether a pair of a derivation node and a meaning representation fulfill the constraints of Section 3.3. EMIT adds an item to the lexicon. EMITTED tests whether an equivalent item is already in the lexicon.

function SPLITREC(y, z)
  if $\neg$EMITTED(y, z) $\wedge$ VALID(y, z) **then**
    EMIT(y, z)
    $y_1, y_2 \leftarrow$ CHILDREN(y)
    $o \leftarrow$ OPERATOR(y)
    for z_1, z_2 such that $o(z_1, z_2) = z$ **do**
      SPLITREC(y_1, z_1)
      SPLITREC(y_2, z_2)
    end for
  end if
end function

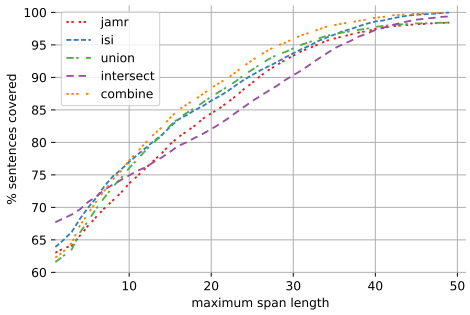
4.1 Quantitative Analysis

We run lexicon induction on the full `proxy-train` subcorpus. To limit the computational effort and reduce the extraction of poorly generalisable lexical items, we apply the following limitations:[1]

1. If more than 1,000 lexical items are extracted from a derivation, it is skipped entirely.
2. Sentences with more than ten unaligned nodes are skipped.
3. If at any derivation node, more than ten lexical items are generated, none of them are included in the lexicon (but induction may still proceed recursively from these nodes).

We measure results by examining the distribution of *maximum span lengths* over the corpus. The maximum span length of a sentence is defined as the length of its longest subspan which the induction algorithm was not able to split any further. Ideally, we would like to achieve a maximum span length of 1 for every sentence, meaning that each individual token is assigned at least one lexical item.

[1] Constraints 1 and 2 will tend to penalise longer sentences more frequently. While this skews our results towards shorter sentences, we have found them necessary to keep the runtime of the algorithm manageable.



(a) Lexical induction coverage. For each alignment strategy, the plot shows the percentage of sentences having at most a given maximum span length.

Alignments	Lexicon Size	Skipped
jamr	314,299	104
isi	387,932	2
union	275,418	103
intersect	429,278	41
combine	286,129	3

(b) Lexicon sizes and skipped sentences by alignment strategy. Sentences were skipped if they produced more than 1,000 lexical items.

Figure 2: Comparison of alignment strategies for lexical induction on the `proxy-train` set.

4.1.1 Alignment Strategies

We first examine the impact of several alignment strategies. The two alignment tools investigated here, the JAMR aligner (Flanigan et al., 2014) and the ISI aligner (Pourdamghani et al., 2014), use different approaches and thus produce alignments with different characteristics. We therefore explore different strategies of combining the alignments produced by the two tools:

- **jamr/isi:** Use only the output from one of the tools. In the case of the ISI aligner, alignments to edges are dropped.
- **union/intersect:** Take the union / intersection of both aligners' outputs.
- **combine:** Take the union of both aligners' outputs. However, if a node has been aligned to different tokens by the two aligners, drop alignments for this node altogether.

Two strategies exhibit the most interesting properties (Figure 2). The *intersect* strategy achieves a maximum span length of 1 on the most sentences, but its produced alignments are too sparse, causing many lexical items to be dropped due to ambiguity. From a maximum span length of 8, the *combine* strategy is more successful, while still maintaining a small lexicon size and a low number of skipped sentences. Intuitively, it increases the node coverage of either method, while also allowing the correction of errors made by one of the tools.

4.1.2 Lexicon Statistics

The lexicon resulting from the *combine* strategy has 286,129 entries. In comparison, this is less than a fifth the lexicon size of 1.6 million entries reported by Artzi et al. (2015).

Of the 6,603 sentences, the algorithm skipped three because they would have produced more than 1,000 lexical items. Each of the remaining sentences contributes on average 3.29 lexical items per token (median 2.91).

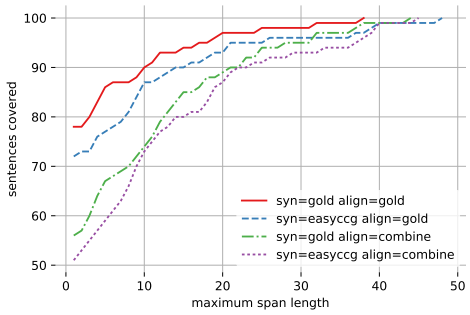
The lexicon is essentially a non-unique mapping from sequences of tokens to pairs of syntactic and semantic categories. By counting the number of distinct entries for each token sequence, we can assess the ambiguity of the lexicon. For the token sequences represented in the lexicon, the mean ambiguity is 10.6 (median: 4). There is a small number of entries with very high ambiguities: 73 tokens have an ambiguity of more than 100, the most ambiguous one being *in* with 1,798 lexical items. In general, prepositions dominate among the most ambiguous items, because they occur frequently, tend not to have any alignments, and also have a high degree of legitimate polysemy. However, high-frequency words specific to the corpus also appear, such as *government* or *security*.

Of the 10,600 unique tokens in the training corpus, 23% (2,458) are not assigned a lexical item at all because the induction algorithm was not able to fully decompose the meaning representation, or because more than ten candidates resulted from every occurrence. They are covered by multi-token items.

4.2 Impact of Tool-Derived Annotations

To examine the induction algorithm's sensitivity to errors propagated from external tools, we compare them with the gold standard annotations available for the `consensus-dev` set. The results are shown in Figure 3.

Not surprisingly, gold annotations perform better than tool-derived annotations. It can be seen that alignments impact grammar induction performance more strongly than syntax parses, with a gap of 21% of perfectly split sentences between gold-standard and tool-derived alignment annota-



(a) Lexical induction coverage with either gold-standard or tool-derived annotations. Tool-derived syntax is from EasyCCG, tool-derived annotations are JAMR/ISI alignments processed using the *combine* strategy.

Syntax	Alignments	Lexicon Size
gold	gold	7,908
easyccg	gold	11,123
gold	combine	4,435
easyccg	combine	5,401

(b) Sizes of the lexicons induced with different annotation sources.

Figure 3: Analysis of lexicon induction performance using gold-standard or tool-derived annotations.

tions. Table 3b shows an increase in lexicon size when EasyCCG syntax is used, which is likely due to added noise because the ten best derivations are considered instead of a single one. Tool-derived alignments, on the other hand, reduce the lexicon size, because alignment errors force induction to stop early on some sentences.

5 Problematic Phenomena in AMR

Is the proposed framework a good fit for analysing the AMR corpus? Figure 3 shows that even if errors from external tools are ruled out, there remains a gap of 22% of sentences that are not fully split by the lexical induction algorithm.

To assess the nature of these failures, we group them into error classes. Table 2 provides an overview of the error counts.

Broadly speaking, we identify three types of errors: Algorithmic limitations, where parameters of the algorithm prohibit the desired decomposition; mismatches where the CCG and AMR annotations choose to represent phenomena in incompatible ways; and non-compositional constructions, where certain AMRs cannot be expressed in our graph algebra.

Error Class	Label	Count
dependency mismatch	dep	15
coreference restriction	coref	3
negation	neg	2
node duplication	dup	2

Table 2: Classification of causes for lexical induction failures, using gold-standard syntax parses and word-to-node alignments, based on the `consensus-dev` data set (100 sentences). The remaining 78 sentences were split perfectly. Labels refer to the paragraphs in Section 5.

5.1 Algorithmic Limitations

Restriction of coreference node extraction (coref) In three cases, we observed more than one <coref> node to be required, as exemplified in Figure 4c.

5.2 Mismatches Between CCG and AMR

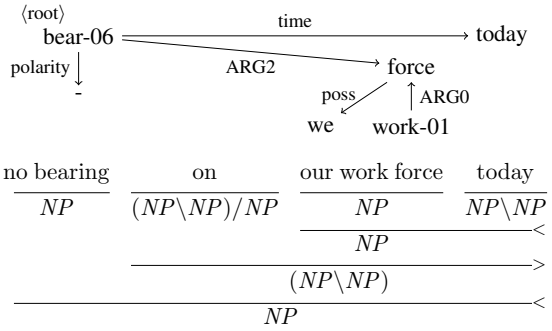
Mismatch of syntactic and semantic dependencies (dep) Since the induction algorithm walks the syntax tree and relies strongly on the existence of a parallel structure between syntax and semantics, it fails in cases where different dependency structures are present in the syntactic and the semantic annotations.

Of the 15 errors in this class, we judged 11 to be "fixable" in that an acceptable CCG derivation could be constructed matching the dependencies on the semantic level. Typically, these are related to ambiguities or annotation errors. Figure 4a shows a typical example.

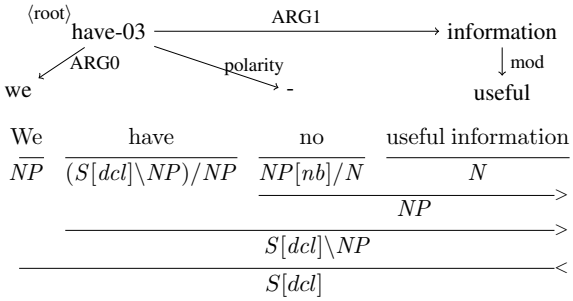
Treatment of negation (neg) Syntactically, the negator *no* can attach to a noun in cases where it semantically modifies the verb, as shown in Figure 4b. In AMR, the choice is made to attach polarity edges to the verb, which prohibits syntactic analysis of such constructions. This is a systematic difference between CCG and AMR.

5.3 Non-Compositional Features of AMR

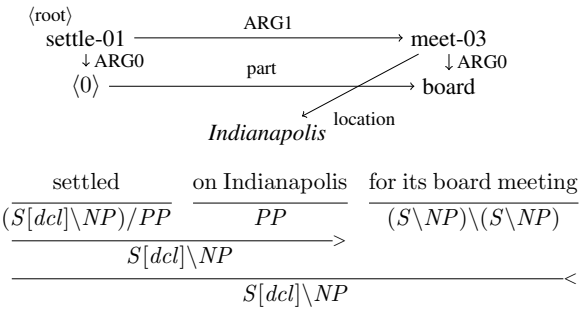
Duplication of nodes (dup) Constructions involving conjunctions or partitives can lead to a duplication of nodes in the meaning representation, as shown in Figures 4d and 4e. This behaviour is not compositional because the duplicated nodes



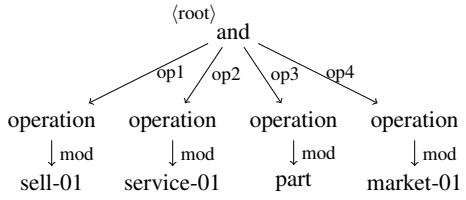
(a) Example for mismatching syntactic / semantic dependencies. Syntactically, a dependency between *work force* and *today* is annotated, but semantically, *today* is dependent on *bearing*. While a syntax derivation matching the semantic dependencies could be constructed, it has not been annotated in CCGbank. From `wsj_0003.30`.



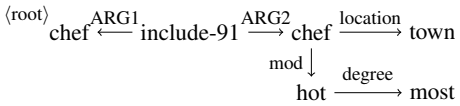
(b) Example for the incompatible treatment of negation. The polarity edge of *have-03* does not match the dependency between *no* and *information*. Simplified from `wsj_0003.9`



(c) Example for a phrase with more than one coreference. The phrase *its board meeting* contains coreferences both via the *location* and *part* edges. The *Indianapolis* node is an abbreviation for a multi-node named entity subgraph. Simplified from `wsj_0010.3`.

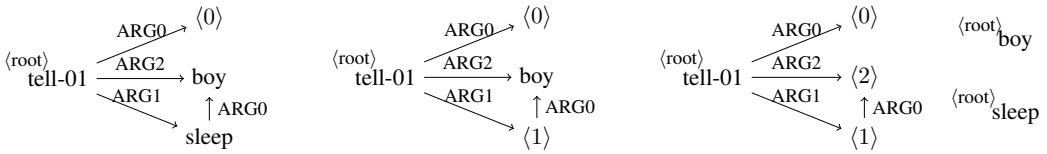


(d) Example for the duplication of nodes due to coordination. The phrase is *sales, service, parts and marketing operations*. Even though the token *operations* occurs only once, a separate *operation* node is introduced for each operand of the *and* conjunction. From `wsj_0009.2`.



(e) Example for the duplication of nodes due to a partitive. The phrase is *some of the hottest chefs in town*. The example illustrate how quantification can lead to nodes being duplicated, such as the *chef* nodes in this AMR. From `wsj_0010.17`

Figure 4: Corpus examples for the phenomena described in Section 5: mismatching dependencies, treatment of negations, number of coreferences, duplication of nodes due to coordination and partitives. Syntax annotations are taken from CCGbank, and semantic annotations are taken from the AMR corpus.

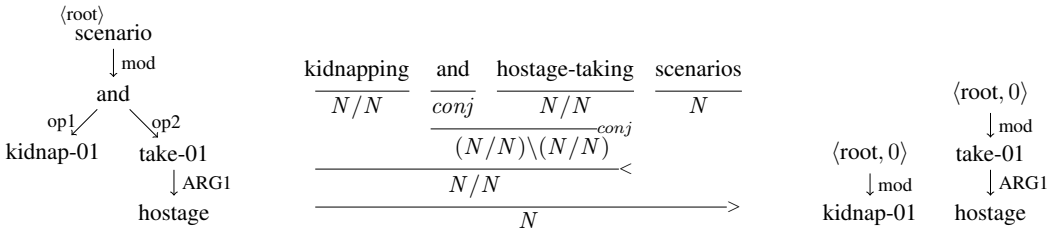


(a) Meaning representation for *told the boy to sleep*.

(b) Induced meaning representation for *told the boy*.

(c) Induced meaning representations for *told*, *boy*, and *sleep*.

Figure 5: Example for the induction of lexical items from an object control verb construction. The sentence is *[The girl] told the boy to sleep*. Since *sleep* is an argument to *tell*, it is not assigned any placeholders and is extracted as a 0-ary lexical item. The control structure is encoded in the lexical item for *tell*.



(a) AMR for the phrase *kidnapping and hostage-taking scenarios*.

(b) CCG derivation showing the lexical entries for the modifiers *kidnapping* and *hostage-taking*.

(c) The lexical entries for the modifiers *kidnapping* and *hostage-taking*.

Figure 6: Corpus example for conjunctions of modifiers. The AMR in Figure 6a cannot be compositionally constructed from the modifier interpretations of *kidnapping* and *hostage-taking* because the *mod* edges present in the lexical entries would have to be destructively modified. Example from `PROXY_AFP_ENG_20020105_0162.12`.

are introduced only once lexically, and then copied based on the syntactic context.

Coordination of modifiers[2] When modifiers are the arguments of a conjunction, the conjunction node itself is connected to the modifiee via a *mod* edge, as shown in Figure 6. Given that each of the conjoined modifiers itself has a *mod* edge, these edges need to be merged, moved, or deleted somehow.

6 Conclusion

We have presented a new variant of CCG which performs semantic construction using the operations of a graph algebra instead of combinatory operations on λ-calculus terms. This allows the grammar to construct graph representations directly without going through intermediate representations. It also restricts the set of possible operations, leading to a compact lexicon. We have demonstrated that under ideal conditions, our grammar achieves a robust coverage of 78% on WSJ sentences.

Our experiments suggest that CCG/AMR is a good overall match for representing the derivation of AMRs. There remain several possibilities for improvement which we leave for future work:

- Allowing the induction algorithm to search over possible derivations and alignments would reduce the influence of both tool errors and mismatching annotations. To keep the lexicon manageable, an optimizing induction algorithm would be needed, e.g. using EM.
- An attempt could me made to more strongly identify placeholders with the argument positions of the corresponding syntactic categories. Among others, this would allow for a more canonical treatment of object control verbs, which is somewhat ad hoc, requiring an interpretation of verbs as 0-ary lexical items (see Figure 5 for an example).
- Additional rules could be introduced to deal with non-compositional phenomena such as the conjunction of modifiers. Statistically, such phenomena appear to be rare, affecting only 2% of the examined corpus.
- Other differences in representation might be resolved statistically or using heuristics. E.g., the fact that negators that attach to a noun syntactically attach to the verb in AMR could be mitigated by a rule that allows for the movement of polarity edges.

Our results represent a promising step towards a more complete grammatical treatment of AMR. Although AMR has not been designed with compositionality in mind, we have shown that it is possible to construct linguistically motivated compositional derivations.

7 Acknowledgements

We thank Alexander Koller and his group, Christine and Arne Köhn, and the anonymous reviewers for their helpful comments. Part of this work was supported by the DFG (IGK 1247).

[2]This phenomenon has not been observed in the `consensus` data set and is therefore not represented in Table 2.

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*, pages 1699–1710, Lisbon, Portugal. Association for Computational Linguistics.

Miguel Ballesteros and Yaser Al-Onaizan. 2017. AMR Parsing using Stack-LSTMs. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1269–1275, Copenhagen, Denmark. Association for Computational Linguistics.

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*, pages 178–186, Sofia, Bulgaria. Association for Computational Linguistics.

Emily M. Bender, Dan Flickinger, Stephan Oepen, Woodley Packard, and Ann Copestake. 2015. Layers of Interpretation: On Grammar and Compositionality. In *Proceedings of the 11th International Conference on Computational Semantics*, pages 239–249, London, UK. Association for Computational Linguistics.

Bruno Courcelle and Joost Engelfriet. 2012. *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*, 1st edition. Cambridge University Press, New York, NY, USA.

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*, pages 1426–1436, Baltimore, Maryland. Association for Computational Linguistics.

Jonas Groschwitz, Meaghan Fowlie, Mark Johnson, and Alexander Koller. 2017. A constrained graph algebra for semantic parsing with AMRs. In *Proceedings of the 12th International Conference on Computational Semantics*, Montpellier, France. Association for Computational Linguistics.

Julia Hockenmaier and Mark Steedman. 2007. CCGbank: a corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396.

Matthew Honnibal, James R. Curran, and Johan Bos. 2010. Rebanking CCGbank for Improved NP Interpretation. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 207–215, Uppsala, Sweden. Association for Computational Linguistics.

Paul Kingsbury and Martha Palmer. 2002. From TreeBank to PropBank. In *Proceedings of the Third International Conference on Language Resources and Evaluation*, pages 1989–1993, Las Palmas, Spain. European Language Resources Association.

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.

Alexander Koller. 2015. Semantic construction with graph grammars. In *Proceedings of the 11th International Conference on Computational Semantics*, pages 228–238, London, UK. Association for Computational Linguistics.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2010. Inducing Probabilistic CCG Grammars from Logical Form with Higher-Order Unification. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1223–1233, Cambridge, MA. Association for Computational Linguistics.

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*, pages 1444–1454, Lisbon, Portugal. Association for Computational Linguistics.

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)*, pages 990–1000, Doha, Qatar. Association for Computational Linguistics.

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.

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*, pages 1775–1786, Austin, Texas. Association for Computational Linguistics.

Rik van Noord and Johan Bos. 2017. Neural Semantic Parsing by Character-based Translation: Experiments with Abstract Meaning Representations. *Computational Linguistics in the Netherlands Journal*, 7:93–108.

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*, New Orleans, USA. Association for the Advancement of Artificial Intelligence.

Xiaochang Peng, Linfeng Song, and Daniel Gildea. 2015. A Synchronous Hyperedge Replacement

Grammar based approach for AMR parsing. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, pages 32–41, Beijing, China. Association for Computational Linguistics.

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)*, pages 425–429, Doha, Qatar. Association for Computational Linguistics.

Mark Steedman. 2000. *The Syntactic Process*. MIT Press, Cambridge, MA, USA.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015. A Transition-based Algorithm for AMR Parsing. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 366–375, Denver, Colorado. Association for Computational Linguistics.