Saarland at MRP 2019: Compositional parsing across all graphbanks

Lucia Donatelli, Meaghan Fowlie*, Jonas Groschwitz, Alexander Koller, Matthias Lindemann, Mario Mina, Pia Weißenhorn
Department of Language Science and Technology, Saarland University
* Department of Linguistics, Utrecht University
{donatelli|jonasg|koller|mlinde|mariom|piaw}@coli.uni-saarland.de
m.fowlie@uu.nl

Abstract

We describe the Saarland University submission to the shared task on Cross-Framework Meaning Representation Parsing (MRP) at the 2019 Conference on Computational Natural Language Learning (CoNLL).

1 Introduction

In this paper, we describe the semantic parser submitted by Saarland University to the MRP shared task (Oepen et al., 2019)[1]. This task consists in learning to accurately map English sentences to graph-based meaning representations across five different graphbanks.

There has been substantial previous work on graph parsing for each of the graphbanks in MRP, including DM and PSD (Peng et al., 2017; Dozat and Manning, 2018), EDS (Buys and Blunsom, 2017; Chen et al., 2018), AMR (Flanigan et al., 2014; Buys and Blunsom, 2017; Lyu and Titov; Zhang et al., 2019), and UCCA (Hershcovich et al., 2017, 2018; Jiang et al., 2019). One advantage of our parser is that it works accurately across all graphbanks at the same time.

Instead of learning to map directly from sentences to graphs, our parser learns to map sentences to *AM dependency trees*. Each AM dependency tree consists of a graph for the lexical meaning of each token in the sentence, along with a dependency tree that specifies the words that fill each semantic role of a given predicate. An AM dependency tree can be deterministically evaluated to a graph via the *AM Algebra* (Groschwitz et al., 2017). Thus, the parser compositionally maps sentences to graphs, with the AM dependency trees describing the compositional structure of the meaning representation. We will sketch the background on AM dependency trees in Section 2.

[1] http://mrp.nlpl.eu

In earlier work, we showed how to accurately predict AM dependency trees for AMR using a neural dependency parser and supertagger (Groschwitz et al., 2018). We extended this parser from AMR to the DM, PAS, PSD, and EDS graphbanks and obtained state-of-the-art results across all of these graphbanks (Lindemann et al., 2019); we will call this system the *ACL-19 parser* throughout this paper. Earlier semantic parsers were only available for one or two families of closely related graphbanks; our system was the first to parse accurately across a range of different graphbanks. We took this parser as the starting point of our MRP submission; we explain the minor tweaks that were needed for the MRP flavors of DM, PSD, EDS, and AMR in Section 3.

The one MRP graphbank which was not directly supported by the ACL-19 parser is UCCA (Abend and Rappoport, 2013). We thus implemented heuristics for converting UCCA annotations into AM dependency graphs. Certain design decisions in UCCA made this more difficult than for the other graphbanks; we worked around some of these in preprocessing. We describe the details in Section 4.

We present detailed evaluation results in Section 5. We also describe a few post-deadline improvements, which bring our parser up to an MRP f-score of 71.6 on AMR and 70.1 on UCCA.

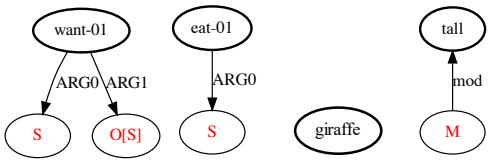
2 AM dependency parsing

We start by describing the ACL-19 parser (Lindemann et al., 2019). This parser is trained to map sentences into *AM dependency trees*, which are then deterministically evaluated to graphs in the *AM algebra*.

2.1 AM Algebra

The Apply-Modify Algebra (AM algebra; Groschwitz et al. (2017)) builds graphs from graph

Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 CoNLL, pages 66–75
Hong Kong, November 3, 2019.
https://doi.org/10.18653/v1/K19-2006



(a) G_{want} (b) G_{eat} (c) G_{giraffe} (d) G_{tall}

Figure 1: As-graphs (= supertags) for the words in the sentence "the tall giraffe wants to eat."

fragments called *annotated s-graphs*, or *as-graphs*. Figures 1a–1d show as-graphs from which the AMR in Fig. 3c for the sentence "the tall giraffe wants to eat" can be built. An as-graph is a labeled, directed graph, some of whose nodes have been marked as *sources*. Every as-graph used in AM dependency parsing has one special root source node, indicated with a bold outline. We mark the other sources with red labels (e.g. S and O); these are nodes at which the root source node of another as-graph will be inserted.

The AM algebra defines two operations for combining as-graphs: Apply, which combines a head with a semantic argument, and Modify, which combines a head with a modifier. Fig. 2a shows a term using these operations that evaluates to the AMR in Fig. 3c.

The result of the Apply-O operation $\text{APP}_{\text{O}}\,(G_{\text{want}}, G_{\text{eat}})$ is shown in Fig. 3a, where the root of the argument G_{eat} is inserted into the O-source of the head G_{want}. The annotation "[S]" at this O-source means that the O-argument must still have an S-source, as is the case for G_{eat}. When two graphs that share a source name are combined, the shared sources automatically merge, creating a re-entrancy. In our example this occurs for the S-source, creating a shared subject slot for G_{want} and G_{eat}.

Fig. 3b shows the result of the Modify-M operation $\text{MOD}_{\text{M}}\,(G_{\text{giraffe}}, G_{\text{tall}})$. The M-source of the modifier G_{tall} is merged with the root of the head G_{giraffe}, which has the effect of adding the modifier to G_{giraffe}; the operation leaves the root of G_{giraffe} where it was. Modify is defined only when it adds no new sources to the head.

Finally, the APP_{S} operation at the root of the term combines the two graphs we built so far, plugging the graph for "tall giraffe" into the S source of the combined want-eat graph. This yields the full graph in Fig. 3c. From a linguistic perspective, a term over the AM algebra serves as a compositional derivation (Montague, 1973) of the graph to which it evaluates.

For this last operation, too, a restriction applies: if a source has no annotation, like the S-source in Fig. 3a, the graph inserted there must have no remaining non-root sources (as is the case here). Thus, both Apply and Modify have restrictions on when they can be used. A term over the AM algebra that satisfies all these restrictions is called *well-typed*.

2.2 AM Dependency Parsing

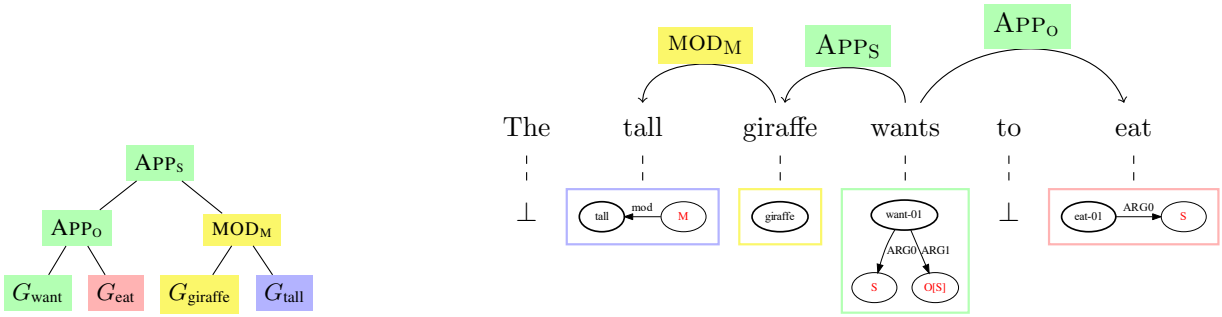
Note that in a term over the AM algebra, such as in Fig. 2a, the root source of the resulting graph is always inherited from the left child; i.e. the left child is always the head. For example, after $\text{APP}_{\text{O}}\,(G_{\text{want}}, G_{\text{eat}})$, the head is still G_{want}. We can track the heads through the term, as indicated by the colors in the example term. This allows us to read terms over the AM algebra as *AM dependency trees* in the following manner. Each operation between two graphs is encoded as a dependency edge from the head to the argument (or modifier respectively), and the edge is labeled with the relevant operation. By aligning the graph fragments to the words in the sentence, we get a dependency tree over the sentence. As a result, the term in Fig. 2a can be unambiguously encoded as the dependency tree in Fig. 2b (Groschwitz et al., 2018).

We can now perform *AM dependency parsing* by training models for the following two tasks: (i) a supertagger to predict the as-graphs for the individual word tokens (such as G_{want}) and (ii) a dependency parser to predict the dependency tree. Together, these two components predict an AM dependency tree, which then evaluates to a graph in the AM algebra as explained above.

Both of these tasks can be performed by neural models with high accuracy. We train a BiLSTM to predict a supertag for each token and use the dependency parser of Kiperwasser and Goldberg (2016) to predict dependency trees. To ensure that we obtain well-typed AM dependency trees, we use the *fixed-tree decoder* algorithm of Groschwitz et al. (2018).

2.3 Decomposition

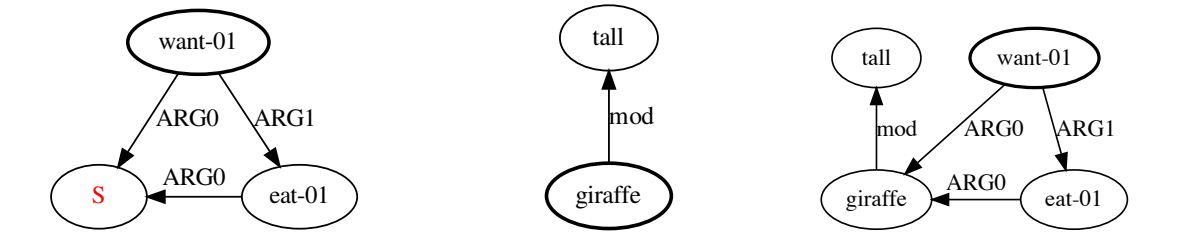
To train the neural supertagging and dependency models, we need AM dependency trees for the training set. However, the available graphbanks contain only sentences with their graph annotations. Thus we have to *decompose* the graphs in each graphbank into the corresponding AM dependency trees. We do this with handwritten heuristics, which we



(a) Term over the AM algebra.

(b) AM dependency tree.

Figure 2: Compositional derivation of the example AMR graph in Fig. 3c.



(a) $\text{APP}_\text{O}(G_\text{want}, G_\text{eat})$
(Evaluation of left subterm in Fig. 2a)

(b) MOD_M $(G_\text{giraffe}, G_\text{tall})$
(Evaluation of right subterm in Fig. 2a)

(c) AMR of *The tall giraffe wants to eat*
(Evaluation of term in Fig. 2a)

Figure 3: As-graphs to which the AM term in Fig. 2a and some of its subterms evaluate.

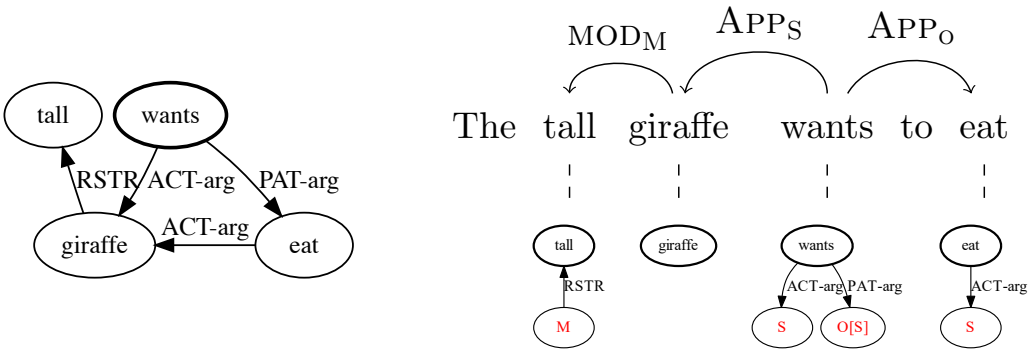


Figure 4: PSD graph (left) for *The tall giraffe wants to eat* and its AM dependency tree (right).

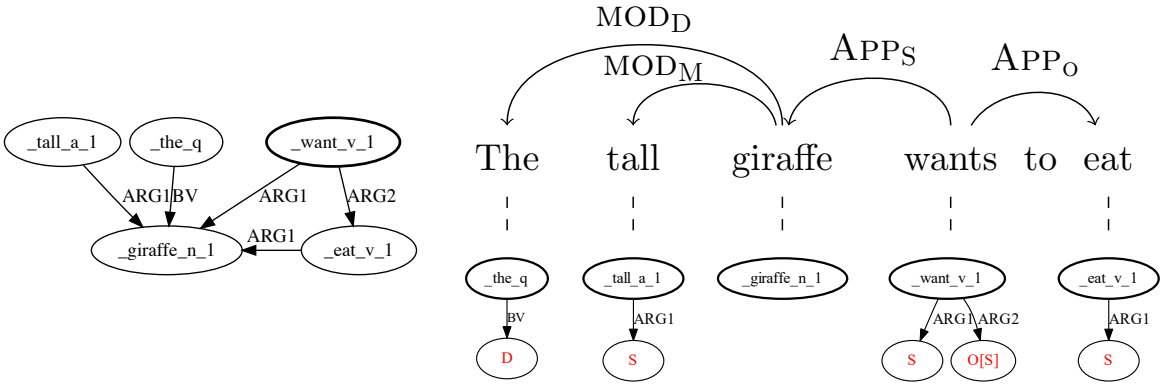


Figure 5: EDS graph (left) for *The tall giraffe wants to eat* and its AM dependency tree (right).

defined for AMR in Groschwitz et al. (2018) and for DM, PAS, PSD, and EDS in Lindemann et al. (2019). The decomposition heuristics perform the following three steps:

1. **align** graph nodes to words (not necessary for graphbanks with annotated alignments between tokens and nodes),
2. **group edges** with nodes, splitting the graph into disjoint aligned fragments,
3. **assign sources** and type annotations to the argument/modification slots of each graph fragment.

These steps define the supertags, and the dependency edges follow from there. Empirically, given an assignment of supertags to tokens, there is never more than one dependency tree which evaluates to the correct graph.

While the AM algebra was originally designed for AMR, the ACL-19 parser extends it to DM, PAS, PSD and EDS as well. In fact, as the AM algebra adds a layer of abstraction on top of the original graphs, using the same parser for all graphbanks becomes easy. Conceptually, we only need a different set of graph fragment supertags for each graphbank.

The decomposition heuristics for PSD and EDS are illustrated in Fig. 4 (PSD) and Fig. 5 (EDS), both for the same sentence "the tall giraffe wants to eat" whose AMR analysis we discussed in Fig. 2b. The examples show that structural differences in the graphbanks can lead to different AM dependencies: for example, the article "the" is part of the EDS graph but not of the PSD and AMR graphs. Overall, however, the AM dependency trees are much more uniform than the underlying graphs.

In Step 2, we group argument edges with the relevant head and modifying edges with the modifier. This yields consistent supertags: for example, "giraffe" can be assigned the same supertag regardless of whether and how many times it is modified. Our heuristics form these groups based only on the edge labels. For example, in AMR, DM and EDS, we group all 'ARGx' labels with their source node. In AMR, we group 'mod' edges with their target node (the modifier), and do the same with 'RSTR' edges in PSD.

The source names are loosely inspired by (deep) syntactic relations; for example, we use the source name S for the endpoints of 'ARG0' edges in AMR, 'ACT-arg' edges in PSD, and 'ARG1' edges in EDS, because these edge labels all correspond to "deep subjects". We also add variants of source assignments to account for e.g. passive. The source annotations are obtained by matching certain patterns in the final graph. For example, the [S] annotation in G_{want} in Figure 3 is added because of the triangle structure in the final graph. Details of these heuristics can be found in Lindemann et al. (2019).

3 Changes to the ACL-19 parser

For the DM, PSD, EDS, and AMR parts of the shared task, we used the ACL-19 parser with the following minor modifications.

3.1 Decomposition heuristics

We did not change any edge attachment or source naming heuristics, but focused on complying with the rules of the shared task and accommodating changes in the graphbanks.

EDS While the ACL-19 parser only dealt with connected EDS graphs, the training corpus of the shared task also contains disconnected graphs. We handle this in the same manner as we handle disconnected graphs in DM and PSD: by introducing an additional node that has a child in each of the disconnected components. This child is chosen as the node being anchored in the highest node in a UD dependency analysis. Along with this node, we introduce a corresponding additional artificial token to the end of the sentence.

Because our decomposition heuristics require a full alignment between tokens and nodes, but the EDS annotations can anchor arbitrary subgraphs in arbitrary substrings, we have to translate EDS anchorings into node-token alignments. We refine our method from the ACL-19 paper in two ways. First, we align *implicit conjunctions* to punctuation in their anchoring span, instead of their leftmost child. Second, we include a special treatment of comparisons in subordinated clauses, where a *subord* node is grouped with a *comp* node, even though they are not immediately connected. This is illustrated in Fig. 6. The ACL-19 heuristic would have tried to group *_hard_a_for* and *subord* into one supertag, which makes it impossible to decompose the EDS graph into an AM dependency tree, because this supertag would have to have two root sources: *_hard_a_for* for the modification with *comp_too*, and *subord* for the application to

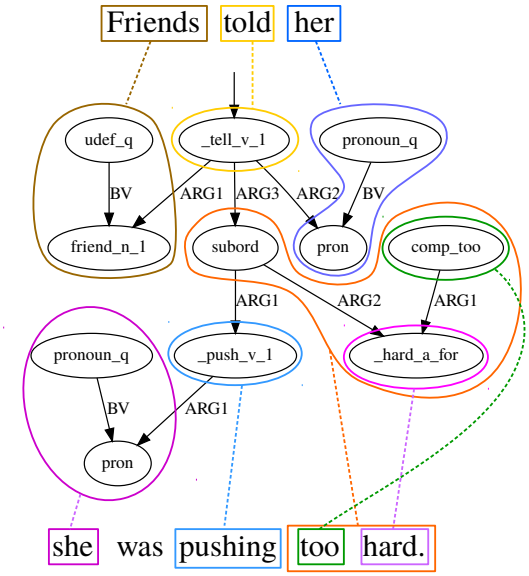


Figure 6: Anchored EDS graph for "Friends told her she was pushing too hard."

_tell_v_1. The revised heuristic instead groups *subord* and *comp_too* into one supertag, which then contains a source node into which *_hard_a_for* can be inserted via Apply.

AMR The aligner we developed for the ACL-19 parser makes non-trivial use of WordNet in order to link tokens to nodes with semantically related labels. Since WordNet was not on the white list of allowed resources, we had to replace it by ConceptNet (Speer et al., 2017). We found that this decreased the dev-set accuracy of our parser by more than a point, possibly because ConceptNet does not distinguish between word senses and thus offers a much larger variety of "hypernyms" than WordNet does.

3.2 Pre- and postprocessing

Unlike earlier versions of the graphbanks and their evaluation metrics, the MRP shared task makes a clear distinction between edges (which link two nodes) and attributes (which attach an atomic value to a node). For instance, information such as polarity and the parts of a named entity are represented as attributes in MRP-style AMR, and parsers are penalized for confusing edges with attributes.

Because our parser uses as-graphs internally, which have node and edge labels but no attributes, we encode attributes into as-graphs. For most graphbanks, we encode attribute information in the node labels and unpack them again in postprocessing. For AMR, we found a considerable amount of noise in the distinction of edges and attributes in the data. We therefore chose to read attributes as edges and restore the distinction heuristically in postprocessing (see appendix).

EDS Since EDS nodes can be anchored in entire phrases but our parser only provides anchoring for tokens to subgraphs, we applied our ACL-19 heuristics to restore such non-trivial anchorings. Where this failed, we marked the node to be anchored in the entire sentence. The ACL-19 parser deleted unanchored subgraphs for evaluation with EDM (Dridan and Oepen, 2011).

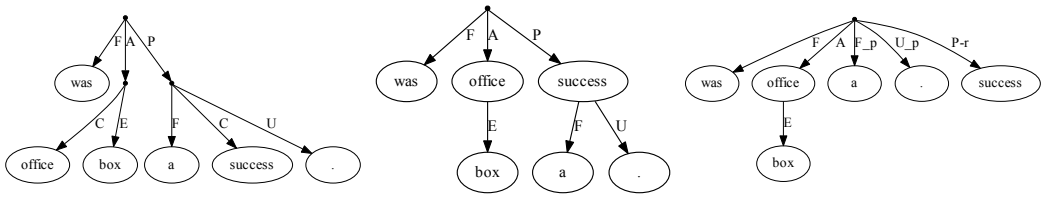
AMR We fixed a postprocessing bug which occasionally resulted in invalid labels in the graph, originating from our procedure for handling rare words.

4 UCCA

For the shared task, we extended the AM dependency parser to UCCA. This was harder than expected. Unlike the other graphbanks, UCCA takes a phrase-structure-like perspective on semantic graphs, in which one terminal node can recursively be the head of several non-terminal nodes (see Fig. 7a). This introduces two challenges for our decomposition heuristic.

First, semantic arguments and modifiers can attach to nodes at any level of the "phrase structure". The graph in Fig. 7a predicates that "office" is an (A)rgument of "success"; these nodes only come together at the root of the UCCA graph. At the same time, the (F)unction word "a" modifies "success" at a lower level of the graph. The obvious decomposition heuristic, which would put the "success" leaf and all the nodes that dominate it into the same supertag, would fail because both of these nodes would have to be root sources, which is not allowed.

Second, under such a decomposition heuristic, the correct supertag for a given word depends on the circumstances. The unmodified word "office" should simply correspond to an as-graph with a single node labeled "office". However, in a sentence where "office" is modified, the correct as-graph consists of "office" with an extra parent node, which is linked to the "office" leaf node with a (C)enter edge (see Fig. 7a). Modifier edges can then attach to this new parent node. This increases lexical ambiguity for our parser, which now has to predict the correct supertag for a word from a larger class of possible supertags.



(a) the original graph (b) after C-edge contraction (c) after edge raising

Figure 7: Fragment of UCCA graph of the sentence *A Few Good Men was released in 1992 and was a box office success*

We address these issues in preprocessing, which we explain below. Edge attachment and source naming heuristics are in the appendix.

4.1 C-edge contraction

We tackle the second problem by contracting C-edges. Whenever we observe a C-edge in the training data, we delete the C-edge and replace its origin node (a nonterminal node) with the node to which the C-edge points (see Fig. 7b). As an exception, we do not contract C-edges for the conjuncts of a coordination, i.e. those C-edges that have a sister C-edge. This decreases the number of nonterminals in the UCCA graph, reduces lexical ambiguity, and increases the proportion of UCCA training graphs which we can decompose.

At test time, the parser predicts UCCA graphs with contracted C-edges, as in Fig. 7b. We uncontract these by creating an outgoing C-edge from all non-leaf nodes that have node labels, changing these nodes into nonterminal nodes. At uncontraction time, we keep the outgoing edges attached to the nonterminal node.

4.2 Edge raising

C-edge contraction is insufficient to completely solve the first problem. For instance, in Fig. 7b, the as-graph for "success" still has two nodes at which other graphs attach: the U and F edge attach to the "success" node with Modify operations, and the "was" node attaches to a non-terminal node with Modify as well. As above, this means that both "success" and this non-terminal node must be root-sources, which is not allowed.

In order to ensure that only one root-source node is required, we flatten the as-graph for "success" by raising the edges out of the lower node to the upper node, as illustrated in Fig. 7c. This means that all modifiers attach to the same node, which becomes the root-source. We train the semantic parser on these flattened UCCA graphs, and then lower the edges again in postprocessing.

Our objective when applying edge lowering on the graph is to redistribute the edges we had previously raised as they were before pre-processing. The initial idea was to make use of the edge labels and only allow lowering an edge from an upper to a lower node if they are connected by another edge with a specific label; however, we found instances where there were multiple outgoing edges with the same label, which resulted in an ambiguity regarding along which edge to lower. Thus, when we raised an edge from the lower node to the upper node, we also marked the edge that connects them with "-r" (for "raised"), and then lowered along the marked edge.

However, we encountered examples where edge lowering was still ambiguous. We found this to occur when edges were raised from multiple lower nodes to the same upper node, resulting in multiple outgoing edges of that upper node bearing the -r mark. Consequently, we had no way of determining which raised edge belonged to which lower node. To remedy this problem, we added a subindex on each of the raised edges indicating the edge over which we had raised the node (see Figure 7c for the subindices). This means for post-processing only lower a given edge to a node through another edge if the label of the former edge matched the subindex of the latter edge. For example, in Fig. 7c, we can only lower the edges with the labels U_p and F_p through another edge with the label P, which in this case implies that we can only lower these edges to the node "success". This procedure results in unambiguous lowering in most cases.

The edge raising and lowering procedure was *not* part of the submitted system. However, it is part of the improved system.

4.3 H-edge removal

An H-edge represents a scene evoked by a Process or State. These edges are normally outgoing edges

of the top node in UCCA. If an H-edge appears in a given graph, it is either unique or accompanied by other H-edges representing multiple parallel scenes and an L-edge to link these scenes, i.e. from the top node there is a single outgoing H-edge in the former case and multiple outgoing H-edges as well as one or more L-edges in the latter. In order to simplify our decomposition heuristics, we remove the H-edge in former case and add it again in post-processing, and only include heuristics for the latter case, rather than distinguish between the two cases.

4.4 Remote edges

We found that removing remote edges drastically helps decomposability. Since this gives us more training data, we decided to remove them and thereby improved decomposability from 34% to 47% in the submitted system.

4.5 Node-token alignments

The UCCA annotation aligns the leaf nodes of the UCCA graph with the tokens in the string; our parser requires an alignment of *all* nodes with their corresponding tokens. We project the aligned tokens upwards from the leaf nodes using a simple set of head percolation rules (see appendix for details).

4.6 Tops

We mark nodes with no incoming edges as top nodes. In an improved version, when more than one top is found, rather than include all of them, we select an arbitrary one.

5 Evaluation

5.1 Experimental setup

We trained one single-task model per graphbank and made use of a concatenation of BERT (Devlin et al., 2019) and Elmo (Peters et al., 2018) embeddings, without any finetuning. We tweaked some hyperparameters of the neural network compared to the ACL-19 parser (see appendix for details).

For DM, PSD and EDS, we use the usual train/dev split. We take a random sample of 3% of all graphs as development data for AMR and 20% for UCCA since there is much less training data.

During parsing, we use the fixed-tree decoder described in Groschwitz et al. (2018) with the six highest-scoring supertags per token. Because the search for a well-typed AM dependency tree is NP-complete, we set a timeout for each graphbank; when the parsing time for a single sentence exceeds a certain limit, we back off to a smaller number of supertags per token and restart parsing. We used a timeout of 30 minutes for DM, PSD and EDS, a timeout of 5 minutes for UCCA and 15 minutes for AMR. We ensured that every sentence was parsed using at least the highest scoring supertag.

In the ACL-19 parser, we used named entity tags as additional input to the neural network for all graphbanks. Here, we only do so for AMR, whose graphs contain very detailed named entity information. We use the Illinois Named Entity Tagger (Ratinov and Roth, 2009). We make use of the tokenization, POS tags and lemmas provided in the MRP companion data. Our code is publicly available at `https://github.com/coli-saar/am-parser`.

5.2 Results

Table 1 ("submitted") shows the official results of our parser in the shared task. Our parser achieved the highest accuracy on PSD and did very well on DM and EDS. It did much worse on AMR than we expected based on earlier results (Lindemann et al., 2019).

Table 2 shows a more detailed evaluation of the system on the development sets. First, we observe that not all graphs in the development sets can be decomposed by the heuristics described above. This is especially striking for EDS (which frequently requires graphs with multiple sources, see the discussion in Lindemann et al. (2019)) and UCCA, where the edge contraction and raising heuristics were still insufficient to decompose all graphs. The distinction between decomposable and non-decomposable graphs also has a clear effect on development f-score: the f-scores on the decomposable subset of each devset are noticeably higher than on the full devset.

Second, we report the accuracy of the two component parts of our parser: dependency parsing (reported as UAS and LAS) and supertagging (reported as 1-best and 6-best supertagging accuracy). It is noticeable that the errors in some graphbanks (e.g. PSD) are dominated by the supertagger, whereas others are hard for the dependency parser (e.g. UCCA). For most graphbanks, low supertagging accuracy goes together with a large supertag set, and low dependency accuracy with a large set of edge labels. For UCCA, accuracy is low across the board, which may be because the decomposable part of the UCCA training set is so small (47%).

	DM	PSD	EDS	UCCA	AMR	Average
Submitted	94.7	91.3	89.1	67.6	66.7	81.9
Improved	94.7	91.3	89.1	70.1	71.0	83.2
Improved + WordNet/Stanford	94.7	91.3	89.1	70.1	71.6	83.4

Table 1: Results of a single run on official test set (MRP cross-framework f-score).

	DM	PSD	EDS	UCCA	AMR	Average
F-score, complete	96.6	92.7	91.1	65.6	72.0	83.6
F-score, decomposable	96.9	92.8	92.0	74.6	73.5	86.0
Decomposability	93.2	97.2	82.0	48.6	91.3	82.5
UAS	95.4	95.7	94.6	74.7	75.2	87.1
LAS	94.6	91.8	93.4	68.1	69.2	83.4
Supertagging Accuracy (1-best)	96.6	88.6	93.9	74.5	75.2	85.8
Supertagging Accuracy (6-best)	99.8	98.8	99.2	94.2	94.2	97.2
Number supertags	424	1566	2739	298	4705	1946.4
Number edge labels	32	42	34	22	48	35.6

Table 2: Detailed dev set results of the submitted system. All rows except the first and third are based on the decomposable subsets. The last section contains statistics about the decomposed training set.

5.3 Improvements

After the shared task submission deadline, we implemented some further improvements.

AMR We fixed a bug in the post-processing of named entities, which improved the MRP f-score by 0.5 points on the dev set and by 4.3 points on the test set ("improved" in table 1).

We also analyzed the impact of switching out WordNet and the Stanford NER tagger for their whitelisted replacements, ConceptNet and the Illinois NER tagger. As Table 3 shows, the use of the whitelisted resources decreased the AMR devset accuracy by almost 1.5 points. This illustrates the impact of these low-level resources on the evaluation results. Interestingly, this translates only to an improvement of 0.6 points on the test set ("Improved + WordNet/Stanford" in table 1).

We leave an investigation why the magnitude of these improvements differs so much between dev set and test set for future work.

UCCA In contrast to the submitted version, we employed edge raising and lowering and used the improved version of the top handling (see 4.6). We also fixed a bug in the node-token alignments. Overall, this resulted in 85% of the training set being decomposable as opposed to 47% in the submitted system. The results are reported in row two of table 1.

		Lexical database	
		WordNet	ConceptNet
NER tool	Stanford	73.9	72.7
	Illinois	73.7	72.5

Table 3: Comparison of MRP f-scores on our AMR development set for different NE recognizers and lexical databases, includes bugfix.

6 Conclusion

In this paper, we have described the Saarland University submission to the MRP shared task. Our system is mostly based on our compositional neural graph parser, which had already worked very well across all MRP graphbanks except for UCCA.

We found that extending the parser to UCCA was a challenge due to the radically different graph structures that UCCA uses. We aim to improve the accuracy of our parser on UCCA in future work.

One challenge our system faces is that nontrivial quantities of training data cannot be decomposed by the heuristics we used. It therefore wastes a lot of training data, especially for UCCA. In future work, we will look into better decomposition heuristics, and also into variants of the AM algebra which support multiple root-sources per as-graph.

Acknowledgments. We thank the MRP organizers for their efforts and their availability during the execution of the shared task. We thank Stephan Oepen for pointing out a bug in our AMR post-processing in an early phase of the shared task. We are grateful to Antoine Venant and Weiwei Sun for discussions and to the anonymous reviewers for their comments.

References

Omri Abend and Ari Rappoport. 2013. Universal conceptual cognitive annotation (ucca). In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*.

Jan Buys and Phil Blunsom. 2017. Robust incremental neural semantic graph parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*.

Yufei Chen, Weiwei Sun, and Xiaojun Wan. 2018. Accurate SHRG-based semantic parsing. In *Proceedings of the ACL*, Melbourne, Australia.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Timothy Dozat and Christopher D. Manning. 2018. Simpler but more accurate semantic dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*.

Rebecca Dridan and Stephan Oepen. 2011. Parser evaluation using elementary dependency matching. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 225–230.

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)*.

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 (IWCS)*.

Jonas Groschwitz, Matthias Lindemann, Meaghan Fowlie, Mark Johnson, and Alexander Koller. 2018. AMR Dependency Parsing with a Typed Semantic Algebra. In *Proceedings of the ACL*.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2017. A transition-based directed acyclic graph parser for UCCA. In *Proceedings of the ACL*, Vancouver, Canada.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2018. Multitask parsing across semantic representations. In *Proceedings of the ACL*.

Wei Jiang, Zhenghua Li, Yu Zhang, and Min Zhang. 2019. HLT@SUDA at SemEval-2019 task 1: UCCA graph parsing as constituent tree parsing. In *Proceedings of the 13th International Workshop on Semantic Evaluation*, Minneapolis, Minnesota, USA.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Matthias Lindemann, Jonas Groschwitz, and Alexander Koller. 2019. Compositional semantic parsing across graphbanks. In *Proceedings of the ACL*.

Chunchuan Lyu and Ivan Titov. AMR Parsing as Graph Prediction with Latent Alignment. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*.

Richard Montague. 1973. The proper treatment of quantification in ordinary english. In *Approaches to natural language*, pages 221–242. Springer.

Stephan Oepen, Omri Abend, Jan Hajič, Daniel Hershcovich, Marco Kuhlmann, Tim O'Gorman, Nianwen Xue, Jayeol Chun, Milan Straka, and Zdeňka Urešová. 2019. MRP 2019: Cross-framework Meaning Representation Parsing. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 1 – 27, Hong Kong, China.

Hao Peng, Sam Thomson, and Noah A. Smith. 2017. Deep Multitask Learning for Semantic Dependency Parsing. In *Proceedings of the ACL*.

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*.

Lev Ratinov and Dan Roth. 2009. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, page 147–155.

Robyn Speer, Joshua Chin, and Catherine Havasi. 2017. Conceptnet 5.5: An open multilingual graph of general knowledge.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019. AMR parsing as sequence-to-graph transduction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*.