The Role of Reentrancies in Abstract Meaning Representation Parsing

Ida Szubert[1]*, **Marco Damonte**[2]*†, **Shay B. Cohen**[1], **Mark Steedman**[1]
[1] University of Edinburgh, [2] Amazon Alexa, Cambridge UK
k.i.szubert@sms.ed.ac.uk, dammarco@amazon.com,
scohen@inf.ed.ac.uk, steedman@inf.ed.ac.uk

Abstract

Abstract Meaning Representation (AMR) parsing aims at converting sentences into AMR representations. These are graphs and not trees because AMR supports reentrancies (nodes with more than one parent). Following previous findings on the importance of reentrancies for AMR, we empirically find and discuss several linguistic phenomena responsible for reentrancies in AMR, some of which have not received attention before. We categorize the types of errors AMR parsers make with respect to reentrancies. Furthermore, we find that correcting these errors provides an increase of up to 5% Smatch in parsing performance and 20% in reentrancy prediction.

1 Introduction

Abstract Meaning Representation (AMR) is a semantic formalism used to annotate natural language sentences as graphs. The task of AMR parsing is to convert sentences into AMR graphs (Banarescu et al., 2013) — rooted and directed acyclic graphs where nodes represent concepts and edges represent semantic relations between them. The AMR for the sentence *I want you to believe me* is shown in Figure 1.

One of the main properties of AMR, and the reason why sentences are represented as graphs rather than trees, is the presence of nodes with multiple parents, called *reentrancies*, as demonstrated in Figure 1, where the node *I* has two parents. Reentrancies complicate AMR parsing and require the addition of specific transitions in transition-based parsing (Wang et al., 2015; Damonte et al., 2017) or of pre- and post-processing steps in sequence-to-sequence parsing (van Noord and Bos, 2017). Enabling AMR parsers to predict

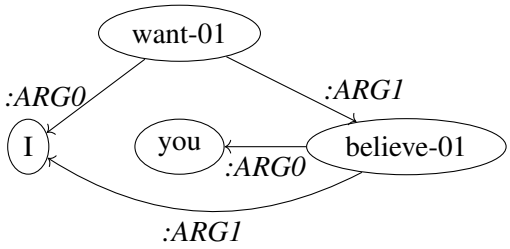


Figure 1: AMR for *I want you to believe me*.

reentrancy structures correctly is of particular importance because it separates AMR parsing from semantic parsing based on tree structures (Steedman, 2000; Liang, 2013; Cheng et al., 2017). Reentrancy is however not an AMR-specific problem (Kuhlmann and Jonsson, 2015), and other formalisms can benefit from a better understanding of how to parse such structures. Nevertheless, to our knowledge, the AMR literature lacks any detailed discussion of the types and linguistic causes of reentrant structures. We aim to fill the gap by describing the phenomena causing reentrancies and quantifying their prevalence in the AMR corpus. We identify sources of reentrancy which have not been acknowledged in the AMR literature such as adjunct control, verbalization, and pragmatics.

AMR parsers are evaluated using Smatch (Cai and Knight, 2013), which however does not explicitly assess the parsers' ability to recover reentrancies. Damonte et al. (2017) introduced a measure of reentrancy prediction, which computes the Smatch score of the AMR subgraphs containing reentrancies. It was observed that the performance of parsers at recovering reentrancy structures is generally poor. We analyze errors made by the parsers and use an oracle to demonstrate that correcting reentrancy-related errors leads to parsing score improvement. Our contributions are as follows:

- We classify the phenomena causing reentran-

* Equal contribution
† Work done while at University of Edinburgh

Findings of the Association for Computational Linguistics: EMNLP 2020, pages 2198–2207

cies, some which have been neglected so far;

- We quantify their prevalence in the AMR corpus automatically and, for a small sample of sentences, manually;
- We categorize types of reentrancy errors made by the parsers and perform oracle experiments showing that correcting these errors can lead to improvements of 20% in reentrancy prediction and 5% overall parsing (Smatch);[1]
- We establish baselines to correct the errors automatically as a post-processing step.

2 Phenomena Causing Reentrancies

AMR reentrancies reflect the fact that an entity can have more than one semantic role in the events described by a sentence. Some of the causes of reentrancies, such as control or coordination, are mentioned in the AMR guidelines and are widely recognized in the AMR literature. Here we present a more in-depth and exhaustive catalogue of reentrancy sources (Table 1) in order to shed some light on what difficult aspects of language and AMR formalism conventions we have to contend with during the task of AMR parsing.

In the analysis that follows we define an AMR node as reentrant if it is a child of more than one other node in the Penman linearization of the graph provided in the corpus. Because of the frequent use of inverse roles in AMR graphs, the directionality of the edges is not obvious. Normalizing inverse roles reverses the edge direction, which changes the parent-child relations between nodes and thus influences which nodes are reentrant, i.e. have more than one parent. As Kuhlmann and Oepen (2016) report, the percentage of reentrant nodes in the AMR corpus increases from 5% to 19% when inverse roles are normalized. For instance, in the relative clause example in Table 1 the *woman* node would be reentrant in a graph with normalized edges, but is not in the graph which follows the corpus linearization. We decided not to normalize the inverse roles for the purposes of our analysis because of the following considerations. Firstly, we assume that there is merit in accepting the edge directionality chosen by the annotator and encoded in the linearization. While different linearizations of the same graph are possible, as the AMR guidelines note, there is usually one that is sensible and reflects the intuitive understanding of which nodes should be considered reentrant. Second, most of the phenomena we discuss yield reentrancies regardless of whether the edge direction is normalized or not. Those phenomena tend to be the more linguistically interesting ones, and the reentrancies which only appear after normalization are largely formalism artefacts (such as ones resulting from using inverse roles to represent adjectives or "-er" nouns), with relative clauses admittedly being an exception.

With that in mind, we classify reentrancy triggers into three broad types: syntactic, pragmatic, and AMR-specific.

Syntactic triggers

We consider a reentrancy as syntactically triggered if the syntactic structure of a sentence forces an interpretation in which one entity performs more than one semantic role. Below we illustrate the syntactic triggers which are commonly discussed in the AMR literature: some types of *pronominal anaphora resolution* (1), prototypical *subject and object control* (3 and 4), and *coordination* (2) (Groschwitz et al., 2017; van Noord and Bos, 2017).

(1) The man$_i$ saw himself$_i$ in the mirror.

(2) She$_i$ ate and ϵ_i drank.

(3) They$_i$ want ϵ_i to believe.

(4) I asked you$_i$ ϵ_i to sing.

In addition to those, our inspection of the AMR data revealed that other kinds of control structures, primarily *adjunct control*, are frequent reentrancy triggers. In adjunct control, the clause which lacks a subject is an adjunct of the main clause, as in the following examples:

(5) I$_i$ went home before ϵ_i eating.

(6) She$_i$ left the room ϵ_i crying.

Such adjuncts express various additional information regarding the main clause, for example the goal, reason, or timing of an event. Unlike the prototypical cases of control, there is by definition no finite list of verbs associated with adjunct control.

Ellipsis is another cause of reentrancies, as in the sentence:

(7) Who can afford it and who can't.

[1] Our source code of the heuristics and the oracle is available at https://github.com/mdtux89/amr-reentrancies.

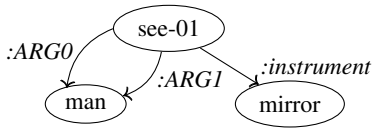
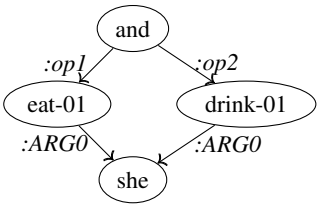
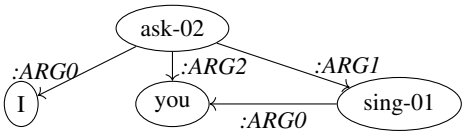
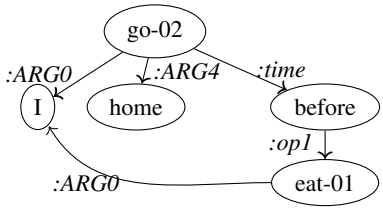
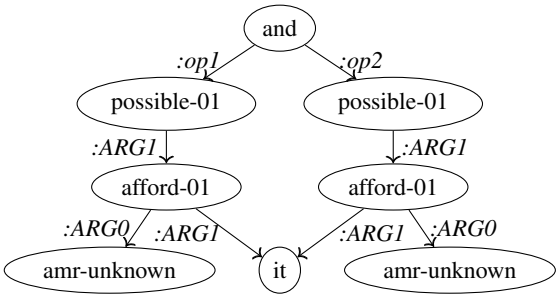
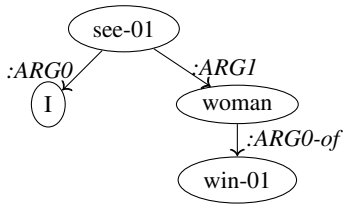
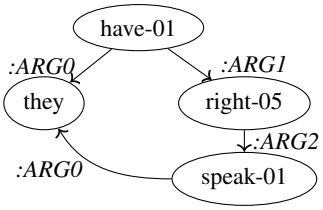
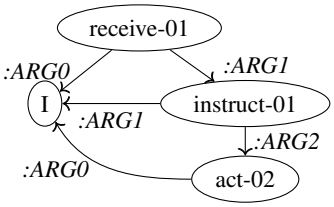
Phenomenon	Sentence	AMR
Coreference	*The man saw himself in the mirror*	
Coordination	*She ate and drank*	
Control	*I asked you to sing*	
Adjunct control	*I went home before eating*	
Ellipsis	*Who can afford it and who can't*	
Relative clause	*I saw the woman who won*	
Nominal "control"	*They have a right to speak*	
Verbalization	*I received instructions to act*	

Table 1: Several linguistic phenomena causing reentrancies in AMR.

in which the node *it* has two incoming edges, creating a reentrancy.

As mentioned before, one would expect *relative clauses* to be one of the syntactic reentrancy triggers, because the noun involved has a semantic role in both the main and relative clause:

(8) I saw the woman$_i$ who ϵ_i won.

In the example above, the woman is the object of seeing and the subject of winning. However, according to the AMR guidelines (Banarescu et al., 2013) relative clauses should be annotated as attaching to the noun with an inverse role, thereby avoiding a reentrancy (see Table 1).

Pragmatic triggers

Human annotators resolve *coreferences* even in the absence of definite syntactic clues, giving rise to pragmatically triggered reentrancies. To this class belong for instance the cases of pronominal anaphora resolution where the anaphora is not syntactically bound (unlike in 1). While coreference is, in general, a discourse phenomenon (Hobbs, 1979), it is also applicable to individual sentences such as those in the AMR corpora:

(9) The coach of FC Barcelona said the team had a good season.

It is pragmatically understood that *FC Barcelona* and *the team* refer to the same entity, even though the coach could have been talking about another team.

Another example is provided by *control-like structures* within nominal and adjectival phrases:

(10) They$_i$ have a right ϵ_i to speak freely.

(11) He$_i$ was crazy ϵ_i to trust them.

An AMR annotation will state that in example 10 the possessor of the right and the subject of speaking are the same, and in example 11 the the same person is crazy and is trusting them. The recovery of the subject of the infinitival clause in such constructions is driven by semantics or pragmatics rather than syntax (Huddleston and Pullum, 2002).

AMR conventions

Finally, the last source of reentrancies is AMR conventions. The AMR guidelines instruct annotators to use OntoNotes predicates whenever possible, regardless of the part of speech of the word. This encourages *verbalization* of elements of the sentence which would not usually be considered predicative.

(12) I received instructions to act.

(13) The opium trade finances corrupt officials.

In example 12 the plural noun *instructions* appears in the AMR graph as a predicate node *instruct-01*. This encourages explicitly annotating inferred semantic roles and so *I* becomes an object of instructing as well as of receiving, causing a reentrancy. Additionally, because of the control-like structure, *I* is also annotated as an object of acting. In example 13 the adjective *corrupt* becomes in the AMR graph a predicate whose subject are the officials.

We consider this class as separate from pragmatical triggers, because the inference made by annotators goes beyond pragmatics and is motivated by the constraints of the formalism rather than by what is actually expressed by the sentence. There are other conventions besides verbalization which introduce reentrancies, in particular if inverse roles were normalized[2]. Our choice to nor normalize edge direcionality was partially motivated by a desire to avoid including those phenomena in our analysis.

3 Quantifying Reentrancy Causes

In order to assess the prevalence of the various reentrancy triggers, we designed heuristics to assign each reentrancy in the AMR corpus to one of the above phenomena. We automatically align AMR graphs to their source sentences using JAMR (Flanigan et al., 2014) and identify the spans of words associated with re-entrant nodes.[3] Heuristics based on Universal Dependency (UD) parses (Manning et al., 2014) and automatic coreference resolution are applied to the spans and the AMR subgraphs containing the reentrancy to classify the cause.[4] We use the NeuralCoref project for coreference resolution.[5]

We recognize syntactic reentrancy triggers primarily with UD-based heuristics. For prototypical cases of control we look for common con-

[2]representation of "-er" nouns with their corresponding predicate and a person node; the convention for representing *government*; special frames for roles

[3]https://github.com/jflanigan/jamr

[4]https://stanfordnlp.github.io/CoreNLP

[5]https://github.com/huggingface/neuralcoref

Phenomenon	Frequency	
	heuristics	total
Coreference	18%	37%
Adjunct control	14%	16%
Control verbs	2%	4%
Coordination	11%	17%
Verbalization	9%	14%
Unclassified	46%	-
Pragmatic overreach	-	3%
Ellipsis	-	2%
Control-like structure	-	2%
Annotation mistakes	-	5%

Table 2: Percentage of reentrancies in the LDC2015E86 training set. The **heuristics** column reports automatically detected frequencies for the whole training set. The **total** column reports frequencies estimated by combining automatic and manual annotation. "Unclassified" are all reentrancies for which our heuristics fail to detect the cause.

trol verbs such as *want*, *try*, and *persuade*,[6] with an outgoing *xcomp* dependency. To identify other types of control, such as adjunct control, we look for *xcomp*, *ccomp* or *advcl* dependency between words aligned to parents of a re-entrant node. For coordination we only check the AMR itself, looking for coordination nodes (i.e., nodes labeled with *and*, *contrast-01*, or *or*). For coreference, we look for re-entrant nodes associated with more than one span and check if those spans corefer.

Finally, for verbalization, we look for nouns or adjectives aligned with OntoNotes predicates in the AMR graph. We tried to identify nominal control-like structures by looking for nominals with an *acl* dependent infinitive or gerund subjectless verb. However, as the precision of the rule is low, and most examples uncovered by this heuristic also fall into the verbalization category, we do not include it in our statistics.

The results of this analysis are in Table 2 in the *heuristics* column. The most common cause of reentrancy appears to be coreference. Control is almost as frequent, with adjunct control being much more common than prototypical control verbs.

We note that our heuristics cannot find the cause for 46% of all reentrancies. This can happen for several reasons. There are sources of reentrancy (ellipsis, nominal control-like structures) for which we do not have heuristics due to the difficulty of defining them in terms of UD parses. The heuristics we do define are of high precision if provided with correct input, but all of the systems we use to provide that input – AMR aligner, POS tagger, UD parse, and coreference resolution system – are in fact noisy. Moreover, what is considered to co-refer in AMR does not necessarily agree with the notion implicit in the coreference resolution system. Consider the following sentence:

(14) The countries signed an agreement that binds the signatories.

The coreference resolution system does not follow the looser definition of coreference used in the AMR annotation guidelines, where *The countries* and *the signatories* are labeled as coreferential. Finaly, some of the reentrancies unaccounted for by the heuristics are due to annotation mistakes. For example in the sentence *A nuclear team will make a visit to inspect the nuclear site*. The AMR for this sentence contains a reentrancy for the *nucleus* node, which is used to modify both the *team* and the *site*, while there should be two separate *nucleus* nodes.

To estimate the overall prevalence of reentrancy triggers, including cases for which the heuristics do not work, we manually annotated causes of unaccounted for reentrancies (79 cases) in a sample of 50 sentences. We combine the results of that manual analysis with the frequencies obtained through the use of heuristics to obtain the overall trigger frequency estimate. The results are shown in Table 2 in the *total* column.

We find that triggers not covered by heuristics account for estimated 4% of total cases, and 34% of unclassified triggers belong to categories for which we do have heuristics, which illustrated the noisiness of the systems used for the heuristic analysis. The final 3% consist of examples of what we consider to be AMR annotators overreaching in their pragmatic interpretation of the sentence. Consider the sentence:

(15) The group said the foreign broadcasters are battering their culture and that it is insulting behavior.

In its AMR, the node *insult-01* takes *group* as its *:ARG1*, making an arguably unwarranted assumption that the behavior is insulting to the group. We note that the inclusion of this type of reentrancies in AMR is controversial as it annotates be-

[6] https://en.wiktionary.org/wiki/Category:English_control_verbs

yond what semantics should represent. Finally, 5% of the unaccounted reentrancies were due to mistakes in the AMR annotations. In the following sentence, the annotator redundantly created both an edge expressing that *make-19* is the purpose of *remove-01*, as well as an edge showing that *remove-01* is *:ARG0* of *make-19*, leading to an unnecessary reentrancy for the *remove-01* node.

(16) People were removed from their homeland to make way for the base.

4 Reentrancy-related Parsing Errors

We propose a method, independent from the AMR parser used, to classify the errors that AMR parsers typically make when predicting such structures. In order to identify the errors, we compare the predicted AMR graphs with the gold standard. We use Smatch to find the best alignments between variables of the predicted and gold graph. We can then find cases where the predicted graph is either missing a reentrancy or contains an unnecessary one.

Due to the aforementioned noise in the heuristics of Section 3, we did not follow the fine-grained classification of linguistic causes. We instead follow a coarser structural classification of the errors. A typical reentrancy error involves the parser generating two nodes in place of one in the gold standard. This is often the case for reentrancies caused by coreference, as shown in Figure 2. The parser may not realize that two entity corefer, hence erroneously generating two different nodes. The opposite is also possible, where two nodes are erroneously collapsed.

Re-entrant edges can also occur between siblings. This is often the case for reentrancies caused by control verbs, as shown in Figure 3.

4.1 Oracle

We introduce corrections for reentrancy errors, implemented as actions that modify the edges and nodes of the predicted AMR. We then define an oracle, a deterministic method that, given a predicted AMR and the relative gold AMR, returns the set of actions that correct errors in the predicted AMR.

Let the predicted graph, containing n nodes, be defined as:

$$S = (V_s, E_s),$$
$$V_s = \{s_1, s_2, \ldots, s_n\},$$
$$E_s = \subseteq V_s \times V_s.$$

and the target graph, containing m nodes, be defined as:

$$T = (V_t, E_t),$$
$$V_t = \{t_1, t_2, \ldots, t_m\},$$
$$E_t = \subseteq V_t \times V_t.$$

Let $A(\cdot)$ be an alignment (computed using Smatch) that maps a node in V_s to a node in V_t, or nil if the node is not in present in V_t, and $A^{-1}(\cdot)$ be an alignment that maps a node in V_t to a node in V_s, or nil if the node is not in present in V_s. Given a source node s_i, we define $t_i = A(s_i)$. We can then define the following actions:

- ADD: A reentrancy edge is added (Figure 4a).
- ADD-ADDN A reentrancy edge and a node are added (Figure 4b).
- REMOVE A reentrancy edge is removed (Figure 4c).
- REMOVE-RMN A reentrancy edge and a node are removed (Figure 4d).
- MERGE Two nodes are merged (Figure 5a).
- MERGE-RMN Two nodes are merged and a node is removed (Figure 5b).
- SPLIT A node is split in two already existing nodes (Figure 5c).
- SPLIT-ADDN A node is split in one existing node and a new node (Figure 5d).
- ADD-SIB An edge between siblings is added (Figure 6a).
- ADD-SIB-ADDN A node is added and an edge with one of its siblings is added (Figure 6b).
- REMOVE-SIB An edge between siblings is removed (Figure 6c).
- REMOVE-SIB-RMN An edge between siblings and one of the siblings are removed (Figure 6d).

In order to identify the errors and generate the respective oracle actions, we use Smatch to align the variables of predicted and gold graphs. For instance, for the action ADD (Figure 4a), we identify three variables s_a, s_b, s_c and the aligned variable in the target graph t_a, t_b, t_c such that:

$$(s_a, s_b) \in E_s, (s_c, s_b) \notin E_s,$$
$$(t_a, t_b) \in E_t, (t_c, t_b) \in E_t.$$

When such a pattern is found, the oracle algorithm determines that an edge between the siblings has to be created:

$$E_s = E_s \cup (s_c, s_b).$$

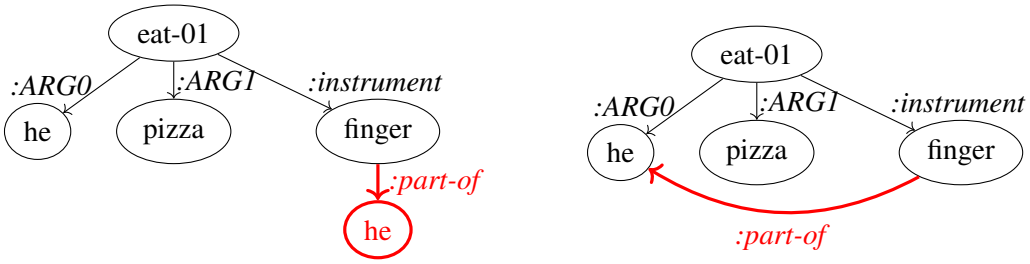


Figure 2: On the left, a coreference-related reentrancy error for the sentence *He ate the pizza with his fingers*. On the right, the correct reentrancy. The difference is highlighted in red.

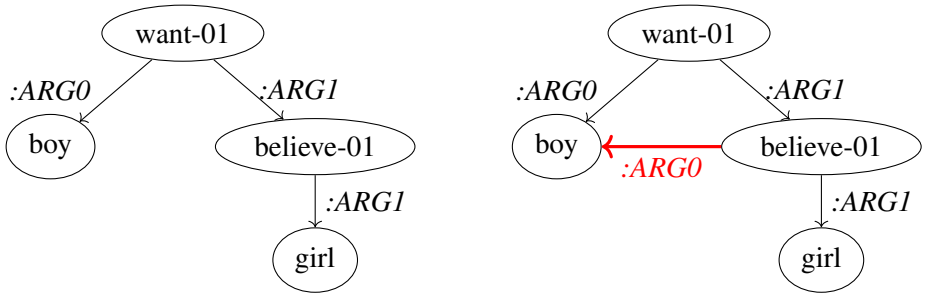


Figure 3: On the left, a control-related reentrancy error for the sentence *The boy wants to believe the girl*. On the right, the correct reentrancy.

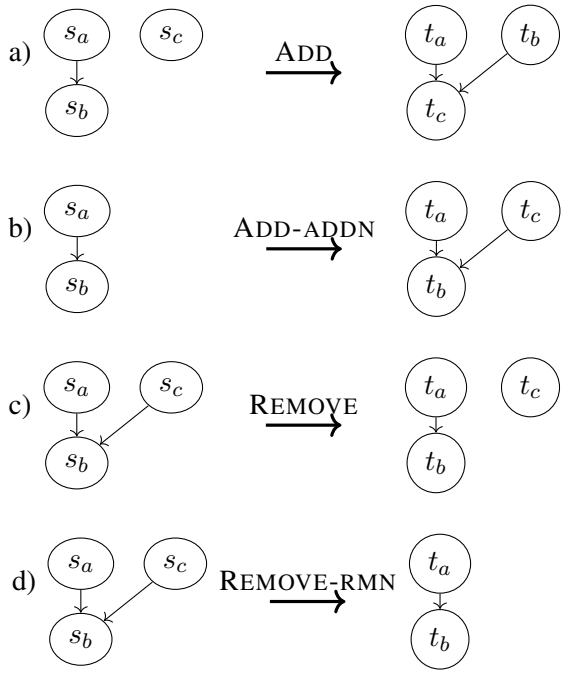


Figure 4: Actions to solve errors caused by missing or extra reentrancies.

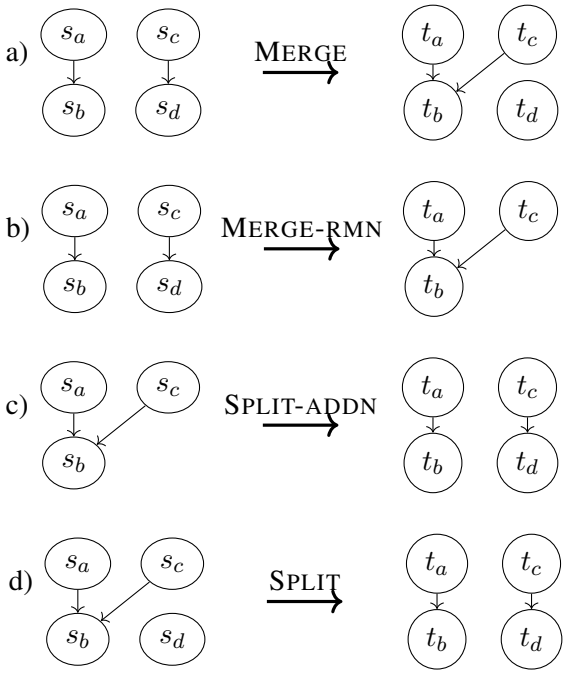


Figure 5: Actions to solve errors due to duplicated or collpased nodes.

The definition of all actions is reported in Appendix A.

We also consider the combination of all actions (ALL). We do so by correcting one error type at the time in a pre-determined order:[7] for each error type, we re-run the oracle to find all errors after the actions for the previous type were applied.

4.2 Oracle Results

We run oracle experiments to explore the impact of the error types on both overall parsing score and reentrancy prediction. For reentrancy prediction, we use the measure introduced by Damonte et al. (2017), which computes the Smatch score

[7] We sorted the actions by the reentrancy prediction score on LDC2017T10 in decreasing order.

Action	LDC2015E86 Freq.	Smatch	Reent.	LDC2017T10 Freq.	Smatch	Reent.
VANILLA	-	73.9	54.3	-	75.2	56.9
ALL	3108.3 (11.59)	+4.6	+18.8	3093.7 (10.12)	+4.4	+18.0
ADD	1292.0 (7.94)	+1.7	+10.4	1305.7 (3.21)	+1.7	+10.3
ADD-ADDN	330.0 (4.36)	+0.8	+4.2	281.3 (5.51)	+0.7	+3.1
RM	545.7 (3.06)	+0.4	-0.1	572.3 (4.04)	+0.4	-0.1
RM-RMN	217.0 (2.00)	+0.3	+0.6	224.7 (3.06)	+0.2	+0.8
MERGE	187.3 (1.53)	+0.4	+1.6	193.3 (3.06)	+0.4	+1.7
MERGE-RMN	94.3 (1.15)	+0.3	+1.0	84.0 (2.00)	+0.2	+0.9
SPLIT	574.7 (3.21)	+1.2	+1.8	541.3 (4.16)	+1.1	+1.7
SPLIT-ADDN	333.0 (1.00)	+0.9	-0.2	347.3 (3.79)	+0.9	-0.0
ADD-SIB	128.0 (1.00)	+0.2	+1.3	119.7 (1.15)	+0.1	+1.2
ADD-SIB-ADDN	99.7 (3.06)	+0.1	-0.1	104.3 (1.53)	+0.1	-0.0
RM-SIB	69.3 (0.58)	+0.1	+0.2	89.3 (0.58)	+0.0	+0.2
RM-SIB-RMN	0.0 (0.00)	+0.0	-0.1	0.0 (0.00)	+0.0	+0.0

Table 3: Relative Smatch improvements with respect to Lyu and Titov (2018) of all actions on the test split of LDC2015E86 and LDC2017T10. Freq. is the number of times the action could be applied, Smatch is the parsing score and Reent. is the reentrancies prediction score. ALL is the combination of all actions. VANILLA are the scores obtained by the original parsers. In parentheses, we report the standard deviation of the actions' frequency. The standard deviation for the Smatch and reentrancy prediction scores is less or equal than 0.12.

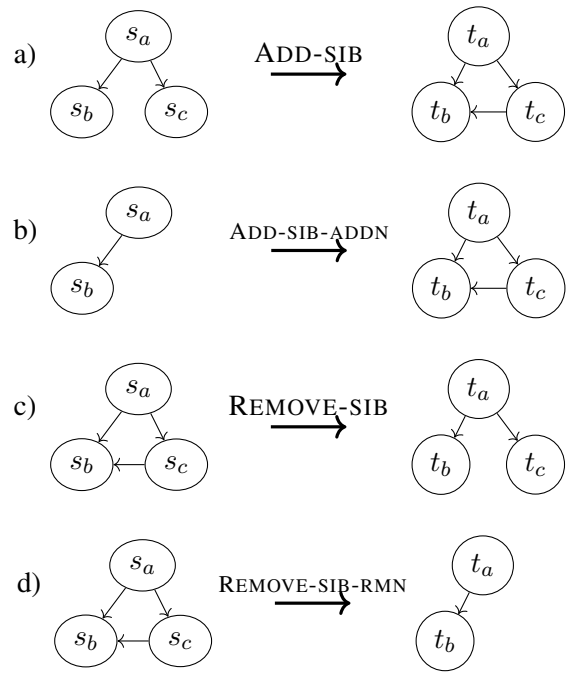


Figure 6: Actions to solve errors due to reentrancies between siblings.

of the subgraphs containing reentrancies.[8] We experiment with the parser of Lyu and Titov (2018) on the test set of LDC2015E86 and LDC2017T10. We rely on Smatch to identify the errors. Because Smatch is randomized, different runs can identify different errors to correct. To account for this, we compute the mean and standard deviation of three runs.

Results are shown in Table 3.[9] While the largest improvements are observed when correcting all error types, the most relevant single oracle action is ADD. For this action, we obtain considerable improvements for both corpora, especially for reentrancy prediction (increase by 10.4 and 10.3 points), but also for Smatch (increase by 1.7 points for both corpora). The ADD corrections provide more than half of the reentrancy score improvement provided by ALL corrections, and slightly less than half of the Smatch improvement.

Because of the use of noisy alignment in oracle action prediction, the oracle provides a lower bound estimate of the possible gains. Overall, we argue that the room for improvement is large enough to warrant more careful treatment of reentrancies, either during training or as a post-processing step.

[8] https://github.com/mdtux89/amr-evaluation

[9] To find and correct errors, we act directly on the triples, not on the PENMAN notation used by Smatch. We therefore implemented a variant of Smatch that directly read triples.

System	Reentrancies
VANILLA	56.9 (0.00)
ORACLE	+10.3 (0.00)
RANDOM	-4.2 (0.06)
SEQ2SEQ	-0.1 (0.25)

Table 4: Relative improvements in reentrancy prediction scores on the test set of LDC2017T10, obtained by the oracle and the proposed baselines. VANILLA are the scores obtained by Lyu and Titov (2018).

5 Automatic Error Correction

We further provide baseline systems that learn when to apply ADD, the most impactful action. First, we experiment with a system that randomly selects two nodes in the predicted graph that are not connected by any edge and add an edge with *ARG0*, the most frequent label. We also train a OpenNMT-py (Klein et al., 2017) sequence-to-sequence model (Bahdanau et al., 2015) with a copy mechanism (Gulcehre et al., 2016). The input sequence is the predicted graph and the output sequence is the sequence of edges to add. For each edge, the output contains three tokens: the parent node, the child node, and the edge label.

Table 4 shows that the baselines do not improve the predictions of the original parsers (VANILLA). While sequence modeling of the output is convenient, other options can be attempted. We are also only exploiting the input AMR parse but not the input sentence. We leave it to future work to address these issues and achieve better results.

6 Related Work

Our classification of phenomena causing reentrancies extends previous work in this direction (Groschwitz et al., 2017). van Noord and Bos (2017) previously attempted to improve the prediction of reentrancies in a neural parser. They experiment with several pre- and post-processing techniques and showed that co-indexing reentrancies nodes in the AMR annotations yields the best results. Transformation-based learning (Brill, 1993) inspired the idea of correcting existing parses. This approach has been mostly used for tagging (Ramshaw and Marcus, 1999; Brill, 1995; Nguyen et al., 2016) but it has also shown promises for semantic parsing (Jurčíček et al., 2009). A similar approach has been also used to add empty nodes in constituent parses (Johnson, 2002), with considerable success. The SEQ2SEQ baseline is an adaptation of the popular sequence-to-sequence modeling (Bahdanau et al., 2015).

An alternative approach to reduce reentrancy errors is to better inform training so that the errors are avoided in the first place. A recent AMR parser (Zhang et al., 2019) outperforms the previous state of the art (Lyu and Titov, 2018) by implementing a copy mechanism aimed at recovering reentrancies, confirming that reentrancies are critical for achieving good AMR parsing performance.

7 Conclusions

Building upon previous observations that AMR parsers do not perform well at recovering reentrancies, we analyzed the linguistic phenomena responsible for reentrancies in AMR. We found sources of reentrancies which have not been acknowledged in the AMR literature such as adjunct control, verbalization, and pragmatics. The inclusion of reentrancies due to pragmatics is controversial; we hope that this work can spur new discussions on the role of reentrancies. Our heuristics fail to detect the causes of many reentrancies. For a more precise estimate of the most common causes of reentrancies, it is necessary to manually annotate the reentrancies in the AMR corpora.

Our oracle experiments show that there is room for improvement in predicting reentrancies, which in turn can translate to better parsing results. Stronger baselines that can learn how to correct the errors automatically are left to future work. While the parser we experimented with no longer gives state-of-the-art results (but also not far from them), newer parsers (Zhang et al., 2019; Cai and Lam, 2020) also report relatively low accuracy on reentrancies (using the metrics from Damonte et al. 2017), and as such we believe our work is relevant to these parsers.

Acknowledgments

The authors would like to thank anoymous reviewers, Adam Lopez, Bonnie Webber, Nathan Schneider, Sameer Bansal, and Yevgen Matusevych for their help and comments. This research was supported by a grant from Bloomberg as well as by the European Union H2020 project SUMMA, under grant agreement 688139 and the project SEMANTAX, which has received funding from the European Research Council (ERC) under the European

Unions Horizon 2020 research and innovation programme, under grant agreement No. 742137.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. *Proceedings of ICLR*.

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. *Proceedings of Linguistic Annotation Workshop*.

Eric Brill. 1993. *Transformation-Based Learning*. Ph.D. thesis, PhD thesis, Univ. of Pennsylvania.

Eric Brill. 1995. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational linguistics*, 21(4):543–565.

Deng Cai and Wai Lam. 2020. Amr parsing via graph-sequence iterative inference. *arXiv preprint arXiv:2004.05572*.

Shu Cai and Kevin Knight. 2013. Smatch: an evaluation metric for semantic feature structures. *Proceedings of ACL*.

Jianpeng Cheng, Siva Reddy, Vijay Saraswat, and Mirella Lapata. 2017. Learning structured natural language representations for semantic parsing. In *Proceedings of ACL*.

Marco Damonte, Shay B Cohen, and Giorgio Satta. 2017. An incremental parser for abstract meaning representation. In *Proceedings of EACL*.

Jeffrey Flanigan, Sam Thomson, Jaime G Carbonell, Chris Dyer, and Noah A Smith. 2014. A discriminative graph-based parser for the abstract meaning representation. *Proceedings of ACL*.

Jonas Groschwitz, Meaghan Fowlie, Mark Johnson, and Alexander Koller. 2017. A constrained graph algebra for semantic parsing with amrs. In *IWCS 2017-12th International Conference on Computational Semantics-Long papers*.

Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. 2016. Pointing the unknown words. *Proceedings of ACL*.

Jerry R Hobbs. 1979. Coherence and coreference. *Cognitive science*, 3(1):67–90.

Rodney Huddleston and Geoffrey K. Pullum. 2002. *Non-finite and verbless clauses*, page 11711272. Cambridge University Press.

Mark Johnson. 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 136–143. Association for Computational Linguistics.

Filip Jurčíček, M Gašić, Simon Keizer, François Mairesse, Blaise Thomson, Kai Yu, and Steve Young. 2009. Transformation-based learning for semantic parsing. In *Tenth Annual Conference of the International Speech Communication Association*.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. 2017. Opennmt: Open-source toolkit for neural machine translation. In *Proceedings of ACL*.

Marco Kuhlmann and Peter Jonsson. 2015. Parsing to noncrossing dependency graphs. *Transactions of the Association for Computational Linguistics*, pages 559–570.

Marco Kuhlmann and Stephan Oepen. 2016. Towards a catalogue of linguistic graph banks. *Computational Linguistics*, 42(4):819–827.

Percy Liang. 2013. Lambda dependency-based compositional semantics. *arXiv preprint arXiv:1309.4408*.

Chunchuan Lyu and Ivan Titov. 2018. Amr parsing as graph prediction with latent alignment. *Proceedings of ACL*.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of ACL*.

Dat Quoc Nguyen, Dai Quoc Nguyen, Dang Duc Pham, and Son Bao Pham. 2016. A robust transformation-based learning approach using ripple down rules for part-of-speech tagging. *AI Communications*, 29(3):409–422.

Rik van Noord and Johan Bos. 2017. Dealing with coreference in neural semantic parsing. In *Proceedings of the 2nd Workshop on Semantic Deep Learning*.

Lance A Ramshaw and Mitchell P Marcus. 1999. Text chunking using transformation-based learning. In *Natural language processing using very large corpora*, pages 157–176. Springer.

Mark Steedman. 2000. The syntactic process. The MIT Press.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015. Boosting transition-based AMR parsing with refined actions and auxiliary analyzers. *Proceedings of ACL*.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019. Amr parsing as sequence-to-graph transduction. In *Proceedings of ACL*.