AMR Normalization for Fairer Evaluation

Michael Wayne Goodman
Nanyang Technological University
Singapore
goodmami@uw.edu

Abstract

Abstract Meaning Representation (AMR; Banarescu et al., 2013) encodes the meaning of sentences as a directed graph and Smatch (Cai and Knight, 2013) is the primary metric for evaluating AMR graphs. Smatch, however, is unaware of some meaning-equivalent variations in graph structure allowed by the AMR Specification and gives different scores for AMRs exhibiting these variations. In this paper I propose four normalization methods for helping to ensure that conceptually equivalent AMRs are evaluated as equivalent. Equivalent AMRs with and without normalization can look quite different—comparing a gold corpus to itself with relation reification alone yields a difference of 25 Smatch points, suggesting that the outputs of two systems may not be directly comparable without normalization. The algorithms described in this paper are implemented on top of an existing open-source Python toolkit for AMR and will be released under the same license.

1 Introduction

Abstract Meaning Representation (AMR; Banarescu et al., 2013) encodes the meaning of sentences in a rooted, directed acyclic graph of concepts (labeled nodes) and relations (labeled edges). It was introduced as being to semantics what the Penn Treebank (Marcus et al., 1994) was to syntax—a simple pairing of sentences and hand-authored annotations—and aimed to coalesce multiple aspects of semantic annotation that had previously been done separately, such as named entity recognition, role labeling, and coreference resolution, into one form.

Research efforts targeting AMR often use the Smatch metric (Cai and Knight, 2013) for evaluation. Smatch views AMR graphs as bags of triples and attempts to find a mapping of nodes between two AMRs that results in the highest F-score in terms of matching triples. The result is a single score for a list of AMR pairs. As AMR encodes many aspects of meaning in one graph, some have found it useful to divide up the parts of the graph that Smatch evaluates so as to inspect a parser's aptitude in each task (Damonte et al., 2017). Nevertheless, Smatch remains the primary underlying method for comparing AMRs and thus ensuring that it is a fair metric is important for the task of semantic parsing.

The AMR Specification[1] describes some features of the representation that expand its expressiveness and improve its legibility, such as reifying graph edges to nodes so that the meaning of the edge can be used by other parts of the graph, and rules for inverting edges so the graph can be linearized into the PENMAN format (Matthiessen and Bateman, 1991). The specification says that these alternations express the same meaning, but they result in different triples used by Smatch for comparison.

In this paper, I investigate the effects these differences have on comparison and propose normalization methods to aid in resolving them. Normalization is intended as a preprocessing step to evaluation and is done to both the gold and test corpus. The purpose is not to yield higher Smatch scores or to change system outputs, but to ensure that conceptually equivalent AMRs evaluate as equivalent and that no system is unfairly penalized or rewarded.

[1] https://github.com/amrisi/amr-guidelines

33rd Pacific Asia Conference on Language, Information and Computation (PACLIC 33), pages 37-46, Hakodate, Japan, September 13-15, 2019

2 Background

While AMR and its PENMAN notation are often considered one and the same, I find that distinguishing them aids the discussion of the Smatch metric, so in this section I explain all three in turn.

2.1 PENMAN Graph Notation

PENMAN notation for AMR is a variation of Sentence Plan Language (Kasper and Whitney, 1989) for the PENMAN project (Matthiessen and Bateman, 1991). The notation is applicable to graphs that are: (1) directed and acyclic (DAGs), (2) connected, (3) with a distinguished root called the *top*, and (4) with labeled nodes and edges.[2] The basic syntax for nodes and edges is as follows:

⟨*node*⟩ ::= '(' ⟨*id*⟩ '/' ⟨*node-label*⟩ ⟨*edge*⟩* ')'

⟨*edge*⟩ ::= ':'⟨*edge-label*⟩ (⟨*const*⟩|⟨*id*⟩|⟨*node*⟩)

The recursion of nodes as targets of edges can only capture projective structures such as trees. In order to encode multiple roots (besides the top node), edges are inverted so the source becomes the target by appending `-of` to the edge label. For reentrancies, node identifiers, also and hereafter called *variables*, are reused.[3] Figure 1 shows an example PENMAN serialization, with all the above features, along with the graph it describes.

```
(n1 / A
   :attr "value"
   :edge1 (n2 / B)
   :edge2-of (n3 / C
      :edge3 n2))
```

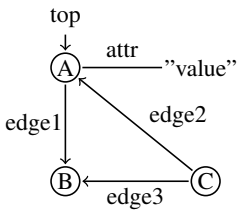

Figure 1: PENMAN notation and the equivalent graph

This paper uses the relative terms *parent* and *child* for the nodes of an edge in the tree structure and *source* and *target* for nodes in the directed graph edges (i.e., such that parent=source in regular edges and parent=target in inverted edges). Edges whose target is a constant are *attributes*. The place where a node specifies its label is the *node definition*.

AMR, described in the next section, uses PENMAN notation to serialize its graph structure. While AMR and PENMAN share a history, the graph notation is not restricted to AMR and could in principle be used for any graphs that meet its criteria. For example it has also been used to encode Dependency Minimal Recursion Semantics (DMRS; Copestake, 2009) for neural text generation (Hajdik et al., 2019) and machine translation (Goodman, 2018).

2.2 Abstract Meaning Representation

Where PENMAN notation is the serialization format, Abstract Meaning Representation (Banarescu et al., 2013) is the semantic framework. As AMR graphs encode semantic information, it refers to node labels as *concepts*, to edges as *relations*, and to edge labels as *roles*. AMR defines in the specification and annotation documentation[4] the inventories of valid concepts and roles and their usage. An AMR graph serialized in PENMAN notation, as in Fig. 2, is simply called an *AMR*, but it can also be represented as a sequence of triples, as in Fig. 3. Node labels are represented by `instance` triples[5] and the top node is indicated with the `:TOP` triple.

```
(d / drive-01
   :ARG0 (h / he)
   :manner (c / care-04
      :polarity -))
```

Figure 2: AMR for *He drives carelessly.*

```
instance(d, drive-01) ^
instance(h, he) ^
instance(c, care-04) ^
TOP(top, d) ^
ARG0(d, h) ^
manner(d, c) ^
polarity(c, -)
```

Figure 3: Triples for *He drives carelessly.*

Several PENMAN graphs may correspond to the same set of triples. A tree-structured graph as in

[2] No technical reason precludes cyclic and unlabeled graphs in PENMAN but I will consider these errors for this paper.

[3] Kasper and Whitney (1989) allowed node attributes and edges to be distributed across multiple references to the node but I will not consider this feature in this paper.

[4] `https://www.isi.edu/~ulf/amr/lib/roles.html`

[5] Some prefer `instance-of` but the choice is arbitrary; I use `instance` to avoid the ramifications of inverted edges.

Fig. 2 has limited options—the branches for `:ARG0` and `:manner` can swap positions, but that's it—but graphs with reentrancies can "rotate" on the reentrant nodes. For example, the graph in Fig. 1 could also be represented as in Fig. 4 or 26 other ways.[6] These alternative serializations do not affect the meaning as determined by the triples (used in evaluation as discussed below), but they can cause issues for systems that learn the serialized character sequences (e.g., Konstas et al., 2017; van Noord and Bos, 2017). Konstas et al. (2017) found that human annotators preferred to insert non-core and inverted relations in the same order as in the original sentence, which leaked ordering information.

```
(n1 / A
   :edge1 (n2 / B
      :edge3-of (n3 / C
         :edge2 n1))
   :attr "value")
```

Figure 4: Alternative serialization of the graph in Fig. 1

While AMR lacks a notion of scope and has no direct model theoretic interpretation,[7] it can encode partial scope information implicitly. For example, the AMRs for *the fast car is red* and *the red car is fast* would differ only by which concept, `fast-02` or `red-02`, is the top of the graph (AMR calls this "focus"). If the examples were, instead, *the fast car that is red* and *the red car that is fast*, then `car` would be the top of both and the triples would be the same, but the PENMAN serializations could differ. Furthermore, reentrancies in AMR present a choice of which occurrence of a variable gets the node definition. It would not be surprising, therefore, for annotators to prefer different PENMAN arrangements for sentences with the same triples, as in Figs. 5 and 6. Put another way, the PENMAN serialization can encode information not present in the triples.

The AMR Specification also describes equivalent[8] variants where the triples do in fact differ. One case is the roles `:domain` and `:mod`, which are considered equivalent in the inverse (i.e., `:domain-of` is equivalent to `:mod`, etc.). The other case is reified relations, where a relation between two nodes becomes a binary node, which is useful when the relation itself interacts with other parts of the graph. These are explained further in Sections 3.1 and 3.2.

```
(b / bite-01
   :ARG0 (d / dog
      :ARG0-of (c / chase-01
         :ARG1 (b / boy)))
   :ARG1 b)
```

Figure 5: AMR for *The dog chasing the boy bit him.*

```
(b / bite-01
   :ARG0 d
   :ARG1 (b / boy
      :ARG1-of (c / chase-01
         :ARG0 (d / dog))))
```

Figure 6: AMR for *The boy chased by the dog was bit by it.*

2.3 Smatch

Smatch (Cai and Knight, 2013) is the primary metric used for AMR evaluation. It estimates the "overlap" between two AMRs by finding a mapping of variables that optimizes the number of matching triples. Precision is defined as $\frac{M}{T}$ and recall as $\frac{M}{G}$ where M is the number of matching triples, T is the number of test triples, and G is the number of gold triples,[9] and the final Smatch score is the F-score of these two. Finding an ideal mapping is an NP-complete task, so Smatch approximates it using greedy search with random restarts to avoid local optima. As regular and inverted relations in AMR are the same when presented as triples, any rearrangement of the PENMAN form for the same triples (as discussed in Section 2.2) will yield the same results as long as the top node does not change, exempting search errors.

Smatch is naïve with respect to AMR-specific interpretations of PENMAN graphs—it only considers the most direct translation of PENMAN graphs to

[6] There are 6 rotations and each rotation has 2 or 6 arrangements by swapping branch positions; more are possible when the top node is not fixed.

[7] Bos (2016) proposed a transformation to first-order logic and also found that a minor change to AMR could allow negation scope to be accurately encoded. Stabler (2017) extended this work and included tense and number features.

[8] Equivalent only by the AMR Specification, not necessarily logical equivalence by a mapping of AMR to logical forms.

[9] The Smatch utility I use (see Section 4) does not specify gold and test, only the first and second arguments. Swapping these arguments swaps precision and recall. I set the gold corpus to the second argument.

triples. It does not consider equivalent alternations where the triples do change (such as `:domain` vs `:mod` alternations and relation reifications) as equivalent, and these alternations will lead to score differences. Smatch is also not robust to subtly invalid graphs, such as inverted edges whose source (i.e., child in the tree structure) is a constant.[10] In this case, the triple will be ignored completely, leading to an inflated score.

Moreover, Smatch gives no credit for a correct role or value unless both are correct. For example, the first line in the Little Prince corpus is *Chapter 7* with the AMR `(c / chapter :mod 7)`, but all three parsers I tested failed to output the correct relation (one gave `:quant 7`, another `:li 7`, and another `:op1 7`). They are therefore all penalized in recall for missing the `:mod 7` relation and again in precision for their incorrect attempt, and none get credit for the correct value of `7`. Omitting the relation entirely (e.g., `(c / chapter)`) yields a higher score, but that's hardly ideal.

The AMR normalizations described in this paper ensure equivalent AMRs have the same triples and thus the same score. In addition, two of the normalizations involve reification which replaces a single triple with several, and this presents a tradeoff: it can allow "partial credit" for getting the role or the value correct, but getting both wrong hurts the score worse than getting a single relation wrong.

3 AMR Normalization

This section describes two meaning-preserving AMR normalizations and two meaning-augmenting normalizations. The first two include canonical role inversions and relation reification, while the latter two include attribute reification and PENMAN structure preservation.

3.1 Canonical Role Inversions

The roles of inverted relations are marked with an `-of` suffix, and generally they are deinverted by removing the suffix. AMR, however, specifies several roles whose canonical form contains the suffix `-of`, namely `:consist-of`, `:prep-on-behalf-of`, and `:prep-out-of`, and the inverse form of these therefore requires an additional suffix (e.g., `:prep-out-of-of`). In addition there is `:mod` which is equivalent to the inverse of `:domain`, and vice-versa.[11] If a gold corpus contained `:mod` while the test corpus used `:domain-of`, Smatch would not see these as equivalent and the score would drop.

By normalizing inverted roles to their canonical forms, such as `:domain-of` → `:mod`, `:consist` → `:consist-of-of`, the Smatch score will not differ for such alternations. Some may argue that normalizing invalid roles such as `:consist` in this way is meaning-altering, but as the naïve inversions of these roles are not separately defined roles in AMR there is no chance of conflation, and in this case I take the position that practicality beats purity.

3.2 Relation Reifications

Some specific relations in AMR can be reified into concepts with separate relations for the original relation's source and target. For example, Fig. 7 is equivalent to Fig. 2 with `:manner` reified to `have-manner-91`. While its possible to reify every eligible relation, in practice all are collapsed unless it is necessary to have the node, so Fig. 2 would generally be preferred over Fig. 7.

```
(d / drive-01
   :ARG0 (h / he)
   :ARG1-of (h2 / have-manner-91
      :ARG2 (c / care-04
         :polarity -))
```

Figure 7: AMR for *He drives carelessly* with `:manner` reified to `have-manner-91`

```
(d / drive-01
   :ARG0 (h / he)
   :ARG1-of (h2 / have-manner-91
      :ARG2 (c / care-04)
      :polarity -))
```

Figure 8: AMR for *He doesn't drive carefully.*

There are three situations where reification is useful: (1) when the meaning of the relation itself is the focus or the argument of another concept instance; (2) when it breaks a cycle in the graph; and (3) when

[10] Only nodes, not constants, may specify relations. These invalid graphs occur occasionally in the output of some parsers.

[11] The specification suggests that `:mod` *is* the inverse of `:domain`, but that could not be true as `:mod` appears in attribute relations and a relation's source cannot be a constant.

Role	Concept	Source	Target
:degree	have-degree-92	:ARG1	:ARG2
:manner	have-manner-91	:ARG1	:ARG2
:purpose	have-purpose-91	:ARG1	:ARG2

Table 1: Sample of reification definitions

an annotator uses a "shortcut" role in a relation. Situation (1) is the only case that is strictly necessary. For example, Fig. 8 is used to express *He doesn't drive carefully*, where the have-manner property is negated rather than the manner itself. The breaking of cycles in situation (2) is possible because reification replaces an edge with a node and two outgoing edges, thus becoming a new root (but not necessarily the graph's top). These kinds of reifications ensure that the graph remains a DAG—a property that may be useful for some applications. The "shortcut" roles of situation (3) are a feature of the AMR Editor (Hermjakob, 2013) provided as a convenience to annotators. They are always reified automatically by the editor and therefore might be considered not part of the official role inventory in the AMR framework. Annotators not using the editor, however, might use them as they are listed in the specification, so it is still useful to reify these in normalization.

In implementation, reification is not complicated. The process uses a defined mapping of roles to AMR fragments containing the reified concept and the roles that capture the original relation's source and target. A sample of these definitions is shown in Table 1; the full list is given in Appendix A. Reification uses this mapping to replace some relation `(a :<role> b)` with `(a :<source>-of (c / <concept> :<target> b))` for regular relations and `(a :<target>-of (c / <concept> :<source> b))` for inverted relations. Reification used in normalization will always have one inverted edge as the original AMR would not have had any way to focus the pre-reified relation.

Collapsing, or dereifying, nodes to edges is slightly more complicated because there are more restrictions on when it can be applied. A node can only be collapsed if it does not participate in relations (including the `:TOP` relation) other than those resulting from reification.[12] For example, `have-manner-91` in Fig. 7 can be collapsed but it cannot be in Fig. 8 because in the latter it is involved in the `:polarity` relation. The change to the graph itself is just the opposite of reification: `(a :<source>-of (c / <concept> :<target> b))` becomes `(a :<role> b)` and `(a :<target>-of (c / <concept> :<source> b)` becomes `(a :<role>-of b)`.

There are additional complexities when the reification mapping is not one-to-one; that is, when it maps multiple relations to the same concept or a single relation to multiple alternative concepts. For the first case, normalization always introduces a new node for each reified relation, even when multiple relations on the same node are mapped to the same concept. This case only occurs with the shortcut roles `:employed-by`/`:role` and `:subset`/`:superset`. For the second case the relations will not be reified because it is undecidable which of the competing concepts should be used, and likewise in dereification information would be lost by collapsing both concepts to the same relation. This case occurs with `:poss` reifying to either `own-01` or `have-03`, and `:beneficiary` reifying to either `benefit-01` and `receive-01`.

The effect of reification on the Smatch score can be large. By reifying one relation to a node with two relations, the net total of triples increases by two. In the gold corpus (see Sections 4 and 5), roughly 15% of triples were reifiable, so a fully-reified corpus would contain roughly 30% more triples. The result is that Smatch will require more time and memory to compute a score, and the search for the variable mapping may become less stable because there are more nodes to search over. This normalization can affect the Smatch score by amplifying certain kinds of errors and giving partial credit for others. Table 2 shows a gold item (the top AMR for *five apples*) and several test AMRs with various differences. The Collapsed column shows the Smatch score between the gold and test AMRs when the relations are left as-is, and the Reified column shows the score when both gold and test are reified. Smatch's preference for missing versus incorrect relations becomes a dispreference unless the test AMR's role differs and is not reifiable (`:unit` in Table 2).

[12] While it is possible to pull out and collapse the information relating to the reified relation and leave in place the node and its additional relations, I do not do so here.

AMR	Collapsed	Reified
`(a / apple :quant 5)`	-	-
`(a / apple)`	0.80	0.57
`(a / apple :quant` **`1`**`)`	0.67	0.80
`(a / apple` **`:mod`** `5)`	0.67	0.80
`(a / apple` **`:mod 1`**`)`	0.67	0.60
`(a / apple` **`:unit`** `5)`	0.67	0.50
`(a / apple` **`:unit 1`**`)`	0.67	0.50

Table 2: Difference in Smatch score with and without reification (top is gold, rest are test, bold are differences)

3.3 Attribute Reification

As mentioned in Section 2.3, Smatch silently drops triples whose source is not a valid variable, leading to inflated scores. While canonical role inversions (such as `:domain-of` to `:mod`) help here, the situation can be completely averted by reifying every constant into a node with a new unique variable and with the constant as the node's concept. For example, `:mod 7` becomes `:mod (_/ 7)`. The result is not meaning-equivalent as the alternation is not provided by the AMR Specification, but it will at least allow each triple to be considered in evaluation. The effect on Smatch is that each attribute triple is replaced with a relation and a concept triple, thus increasing the number of triples by one for each constant. It also allows for partial credit, similar to reification.

3.4 PENMAN Structure Preservation

Section 2.2 described two kinds of variation in PENMAN that correspond to the same triples: the order of serialized relations on a node and which occurrence of a node contains the node definition. As discussed, these differences can be used to encode nuance or hints to the surface form that the AMR annotates. In order to preserve the information encoded by the location of node definitions, additional `:TOP` relations may be used to indicate which node is the top of the node being defined. These parallel the tree structure rather than the DAG, so they do not invert if the child of an inverted relation (i.e., the relation's source) is a node definition.[13] Inserting these relations into an AMR with n nodes results in $n-1$ new triples as one is not inserted for the top node in the graph. The effect on Smatch is a boost in the score of AMRs that define nodes in the same place.

[13]These `:TOP` relations could lead to a cyclic structure so it is not recommended as a general annotation practice.

4 Experiment Setup

For information about roles and their reifications I use the AMR 1.2.6 Specification[14] and the annotator documentation of roles as of May 1, 2019.[15] For reification I use all non-ambiguous mappings, which excludes `:beneficiary` and `:poss`, and for dereification I also exclude mappings of shortcut roles. My experiments use the training portion of the freely-available Little Prince corpus (version 1.6).[16] For reading and writing PENMAN graphs I use the open-source Penman package for Python.[17] I used JAMR (Flanigan et al., 2016),[18] CAMR (Wang et al., 2016),[19] and AMREager (Damonte et al., 2017)[20] for producing system outputs. All systems use their included models trained on the LDC2015E86 (SemEval Task 8) data, which is out-of-domain for the Little Prince corpus but the parsers then all use comparable models. For comparison I use Smatch (Cai and Knight, 2013).[21]

5 Corpus Analysis

I first inspect the corpus to understand the distribution of normalizable AMRs. Table 3 shows the number of nodes and triples in The Little Prince corpus (1,274 AMRs) for both gold annotations and system outputs. These counts are used for calculating the percentages in Tables 4 and 5.

Table 4 shows the percentage of graphs and triples that have the non-canonical `:domain-of` and `:mod-of` relations. They do not appear in the gold annotations, CAMR output, or AMREager output,

[14]https://github.com/amrisi/amr-guidelines

[15]https://www.isi.edu/~ulf/amr/lib/roles.html

[16]https://amr.isi.edu/download.html

[17]https://github.com/goodmami/penman/

[18]Semeval-2016 branch as of March 21, 2019: https://github.com/jflanigan/jamr

[19]Master branch as of February 19, 2018: https://github.com/c-amr/camr

[20]Master branch as of April 11, 2019: https://github.com/mdtux89/amr-eager

[21]https://github.com/snowblink14/smatch/

Corpus	# Nodes	# Triples
Gold	8,189	16,832
JAMR	8,115	15,509
CAMR	7,404	13,922
AMREager	7,461	15,226

Table 3: Corpus sizes

but do in the JAMR output along with a small number of `:consist` relations (not shown), and no corpus used non-canonical inversions of the `:prep-*` relations. This is not unexpected, as the gold corpus does not contain any instances of these roles, so data-driven parsers would have no examples to learn from. A parser that assembles the graph and inverts as necessary to serialize may be susceptible.

	% `:domain-of`		% `:mod-of`	
Corpus	Graphs	Triples	Graphs	Triples
Gold	0	0	0	0
JAMR	5.81	0.52	8.63	0.80
CAMR	0	0	0	0
AMREager	0	0	0	0

Table 4: Non-canonical role inversions

Table 5 shows the percentage of graphs and relations that are reifiable and the percentage of graphs and nodes that are collapsible. All systems have roughly as many reifiable graphs and relations as the gold corpus. CAMR is the only system that outputs reified relations that can be collapsed, although the number is miniscule.

	% Reifiable		% Collapsible	
Corpus	Graphs	Rels	Graphs	Nodes
Gold	78.96	15.23	0	0
JAMR	73.94	13.07	0	0
CAMR	68.68	14.22	0.16	0.03
AMREager	76.92	17.02	0	0

Table 5: Reifiable relations and collapsible nodes

Using Smatch to compare two versions of the gold corpus—one original and one with reified relations—yields an F-Score of 0.75, or a drop of 25 Smatch points. This result is an estimate of the range of score variation when a system perfectly reproduces the gold corpus but makes the opposite decision regarding reification.

6 System Evaluation

Here I test the effect the normalizations have on Smatch when evaluating system outputs to the gold corpus. Table 6 shows the results of the three systems with various normalizations. While JAMR was the only parser that output non-canonical roles, normalizing the roles did not help its score; in fact, the score dropped slightly. Some of JAMR's non-canonical roles were inverted relations to constants, so Smatch was ignoring them. Normalizing them would thus hurt the score unless the normalized relations were correct. Reification (both kinds) generally led to higher scores, meaning that most relations that were reified were fully or partially correct. One result that stands out is structure preservation; for both JAMR and AMREager it led to decreased scores but it helped CAMR, showing that CAMR is more likely to place node definitions where an annotator would. Finally, the normalization helped AMREager close the gap with JAMR, and in some configurations even surpass it.

7 Related Work

Konstas et al. (2017) normalized AMRs is a destructive way in order to reduce data sparsity for their character-based neural parser and generator. My normalization methods can also reduce sparsity but they also generally increase the size and complexity of the graph, so it's not clear if it would aid character-based models. Damonte et al. (2017) found that parsers do well on different sub-tasks, such role labeling and word-sense disambiguation, and ran Smatch on different subsets of the triples in order to highlight a parser's performance in each task. In addition, Damonte et al. also found that Smatch weighted certain error types more than others, although they looked at more application-specific error types, like the representation of proper names. In contrast, I compare using the full graphs as the goal is normalization, not specialization. My normalization methods are mostly compatible with the subtask evaluation of Damonte et al. 2017 but some the evaluation tasks look for certain roles which disappear on reification. Anchiêta et al. (2019) also noticed that Smatch gives more weight to the top node of the graph, but they reached different conclusions. Where I proposed adding `:TOP` re-

System	Normalization I	A	R	S	Score P	R	F
					0.60	0.56	0.58
	✓				0.60	0.55	0.57
		✓			0.61	0.56	0.58
			✓		0.63	0.57	0.60
JAMR				✓	0.59	0.55	0.57
	✓		✓		0.63	0.57	0.60
		✓	✓		0.64	0.57	0.60
	✓	✓	✓		0.64	0.57	0.60
	✓	✓	✓	✓	0.61	0.56	0.59
					0.67	0.56	0.61
	✓				0.67	0.56	0.61
		✓			0.67	0.55	0.60
			✓		**0.70**	0.57	**0.63**
CAMR				✓	0.68	**0.58**	**0.63**
	✓		✓		0.69	0.57	**0.63**
		✓	✓		**0.70**	0.56	0.62
	✓	✓	✓		**0.70**	0.56	0.62
	✓	✓	✓	✓	**0.70**	**0.58**	**0.63**
					0.57	0.52	0.55
	✓				0.57	0.52	0.55
		✓			0.57	0.53	0.55
			✓		0.61	0.57	0.59
AMREager				✓	0.59	0.54	0.56
	✓		✓		0.61	0.57	0.59
		✓	✓		0.60	**0.58**	0.59
	✓	✓	✓		0.60	**0.58**	0.59
	✓	✓	✓	✓	0.61	0.57	0.59

Table 6: Smatch results comparing gold to system outputs with the original graphs, canonical role inversions (I), attribute reification (A), relation reification (R), and structure preservation (S)

lations to all nodes to preserve the PENMAN structure, they discard the `:TOP` node, meaning that the AMRs for *the fast car is red* and *the red car is fast* are evaluated as equivalent. Barzdins and Gosko (2016) presented extensions to Smatch including a visualization of per-sentence error patterns and an ensemble selection from multiple test AMRs per gold AMR. The latter extension could in principle be combined with the normalization procedures I have described, however it would need to be augmented to allow for the normalizations of the gold corpus as well as the test corpus.

8 Conclusion and Future Work

AMR provides flexibility with the way that equivalent graphs are encoded. This flexibility can make life easier for annotators and parsers alike, but it also means that evaluation tools not aware of these allowed alternations can give unfair results. I introduced four normalization methods in this paper. Of these, canonical role inversion, relation reification, and attribute reification are intended to tame the variation that can reasonably appear in parser outputs. The fourth, PENMAN structure preservation, makes evaluation more strictly account for annotation choices which may implicitly encode subtle distinctions in meaning, like scope or nuance.

The evaluation results when comparing a normalized test corpus to the similarly normalized gold corpus are not drastically different. I think this result is a good thing, particularly because comparing a corpus to itself with and without normalization has a very large difference in scores. It suggests that normalization, done to both sides, resolves small differences. While one parser I tested, CAMR, maintained its lead with normalized outputs, the third-place parser AMREager nearly caught up to the second-place JAMR. The relative changes in evaluation scores may important for determining state-of-the-art parsers or for shared task competitions.

The normalizations may be useful not only for evaluation but for preprocessing for data-driven workflows. By removing sources of variation, data sparsity can be reduced which could benefit parser training. The increase in graph size due to the normalization, however, may counteract the benefits. I leave this question open to future research.

The code for this paper is available online at `https://github.com/goodmami/norman`.

References

Rafael Torres Anchiêta, Marco Antonio Sobrevilla Cabezudo, and Thiago Alexandre Salgueiro Pardo. 2019. Sema: an extended semantic evaluation for AMR. In *Proceedings of the 20th Computational Linguistics and Intelligent Text Processing*. Springer International Publishg.

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 Interop-*

erability with Discourse, pages 178–186, Sofia, Bulgaria. Association for Computational Linguistics.

Guntis Barzdins and Didzis Gosko. 2016. RIGA at SemEval-2016 task 8: Impact of Smatch extensions and character-level neural translation on AMR parsing accuracy. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1143–1147, San Diego, California. Association for Computational Linguistics.

Johan Bos. 2016. Expressive power of abstract meaning representations. *Computational Linguistics*, 42(3):527–535.

Shu Cai and Kevin Knight. 2013. Smatch: an evaluation metric for semantic feature structures. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 748–752.

Ann Copestake. 2009. **Invited Talk:** slacker semantics: Why superficiality, dependency and avoidance of commitment can be the right way to go. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 1–9, Athens, Greece. Association for Computational Linguistics.

Marco Damonte, Shay B. Cohen, and Giorgio Satta. 2017. An incremental parser for abstract meaning representation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 536–546, Valencia, Spain. Association for Computational Linguistics.

Jeffrey Flanigan, Chris Dyer, Noah A Smith, and Jaime Carbonell. 2016. CMU at SemEval-2016 task 8: Graph-based AMR parsing with infinite ramp loss. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1202–1206.

Michael Wayne Goodman. 2018. *Semantic Operations for Transfer-based Machine Translation*. Ph.D. thesis, University of Washington, Seattle.

Valerie Hajdik, Jan Buys, Michael Wayne Goodman, and Emily M. Bender. 2019. Neural text generation from rich semantic representations. In *Proceedings of the 2019 Conference on the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT))*, Minneapolis, Minnesota.

Ulf Hermjakob. 2013. AMR editor: A tool to build abstract meaning representations. Technical report, ISI.

Robert Kasper and Richard Whitney. 1989. SPL: A sentence plan language for text generation. *University of Southern California*.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017. Neural AMR: Sequence-to-sequence models for parsing and generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 146–157, Vancouver, Canada. Association for Computational Linguistics.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The penn treebank: Annotating predicate argument structure. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, pages 114–119, Stroudsburg, PA, USA. Association for Computational Linguistics.

Christian Matthiessen and John A Bateman. 1991. *Text generation and systemic-functional linguistics: experiences from English and Japanese*. Pinter Publishers.

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.

Edward Stabler. 2017. Reforming AMR. In *International Conference on Formal Grammar*, pages 72–87. Springer.

Chuan Wang, Sameer Pradhan, Xiaoman Pan, Heng Ji, and Nianwen Xue. 2016. CAMR at semeval-2016 task 8: An extended transition-based AMR parser. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1173–1178, San Diego, California. Association for Computational Linguistics.

A Relation Reifications

Role	Concept	Source	Target	Reifies	Dereifies	Shortcut
:accompanier	accompany-01	:ARG0	:ARG1	✓	✓	
:age	age-01	:ARG1	:ARG2	✓	✓	
:beneficiary	benefit-01	:ARG0	:ARG1			
:beneficiary	receive-01	:ARG2	:ARG0			
:cause	cause-01	:ARG1	:ARG0	✓		✓
:concession	have-concession-91	:ARG1	:ARG2	✓	✓	
:condition	have-condition-91	:ARG1	:ARG2	✓	✓	
:cost	cost-01	:ARG1	:ARG2	✓		✓
:degree	have-degree-92	:ARG1	:ARG2	✓	✓	
:destination	be-destined-for-91	:ARG1	:ARG2	✓	✓	
:domain	have-mod-91	:ARG2	:ARG1	✓	✓	
:duration	last-01	:ARG1	:ARG2	✓	✓	
:employed-by	have-org-role-91	:ARG0	:ARG1	✓		✓
:example	exemplify-01	:ARG0	:ARG1	✓	✓	
:extent	have-extent-91	:ARG1	:ARG2	✓	✓	
:frequency	have-frequency-91	:ARG1	:ARG2	✓	✓	
:instrument	have-instrument-91	:ARG1	:ARG2	✓	✓	
:li	have-li-91	:ARG1	:ARG2	✓	✓	
:location	be-located-at-91	:ARG1	:ARG2	✓	✓	
:manner	have-manner-91	:ARG1	:ARG2	✓	✓	
:meaning	mean-01	:ARG1	:ARG2	✓		✓
:mod	have-mod-91	:ARG1	:ARG2	✓	✓	
:name	have-name-91	:ARG1	:ARG2	✓	✓	
:ord	have-ord-91	:ARG1	:ARG2	✓	✓	
:part	have-part-91	:ARG1	:ARG2	✓	✓	
:polarity	have-polarity-91	:ARG1	:ARG2	✓	✓	
:poss	own-01	:ARG0	:ARG1			
:poss	have-03	:ARG0	:ARG1			
:purpose	have-purpose-91	:ARG1	:ARG2	✓	✓	
:quant	have-quant-91	:ARG1	:ARG2	✓	✓	
:role	have-org-role-91	:ARG0	:ARG2	✓		✓
:source	be-from-91	:ARG1	:ARG2	✓	✓	
:subevent	have-subevent-91	:ARG1	:ARG2	✓	✓	
:subset	include-91	:ARG2	:ARG1	✓		✓
:superset	include-91	:ARG1	:ARG2	✓		✓
:time	be-temporally-at-91	:ARG1	:ARG2	✓	✓	
:topic	concern-02	:ARG0	:ARG1	✓	✓	
:value	have-value-91	:ARG1	:ARG2	✓	✓	

Table 7: Full mapping of roles and concepts used for reification, dereification, and editor shortcuts