Robust Subgraph Generation for Abstract Meaning Representation Parsing

Keenon Werling
Stanford University
keenon@stanford.edu

Gabor Angeli
Stanford University
gabor@stanford.edu

Chris Manning
Stanford University
manning@stanford.edu

Abstract

The Abstract Meaning Representation (AMR) is a representation for open-domain rich semantics. AMR parsing is commonly divided into three challenges: alignment of training graphs to source text, generation of small semantic sub-graphs from token spans, and joining those small semantic sub-graphs to form unified representations. We propose a small set of actions to construct a sub-graphs from a span of tokens. We show that our set of construction actions both perform and generalize better than previous approaches. These actions also provide an insight for an alignment system that yields a "maximally informative" set of action labels, which we show yields good results. We improve on published state-of-the-art AMR parsing, from *0.58* smatch to *0.62* smatch on the LDC2013E117 dataset.

1 Introduction

The Abstract Meaning Representation (AMR) (Banarescu et al., 2013) is a rich language for expressing semantic understanding on both a broad domain and at a relatively deep level. We show an example AMR for the sentence "he gleefully ran to his dog Rover" in Figure 1, and give a brief tutorial on AMR in Section 2.

AMR parsing is informally defined as the task of generating an AMR graph corresponding to the meaning of an input sentence.

AMR captures many useful pieces of semantic information in a single joint representation, but this makes the state-space of possible AMR parses for a given input sentence overwhelmingly large. Previous work has observed that AMR parsing can be partitioned into two tasks: a rich lexically grounded entity detection system, which we call NER++ (see Section 2.2), and a relationship detection system, which we call SRL++ (see Section 2.3).

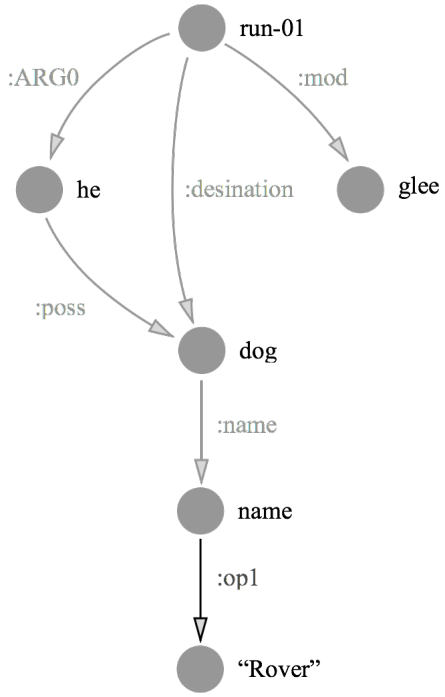

Figure 1: AMR graph for "He gleefully ran to his dog Rover". Nodes represent concepts, and arcs are relationships between concepts. The dark arc labeled "op1" is expected to be generated by NER++.

As an example of the distinction, let's briefly go over Figure 1, the AMR for "he gleefully ran to his dog Rover". To produce this parse, the NER++ task will have to do a verb sense disambiguation on "ran" to get "run-01", and a lemmatization on "gleefully" to "glee". Then NER++ will have to recognize "Rover" is a name, and generate the sub-graph (name :op1 "Rover"). It will also have to recognize that "to" is a dropped preposition, and so doesn't get a node, and "his" is a coreferant that also doesn't get a node. Then "dog" and

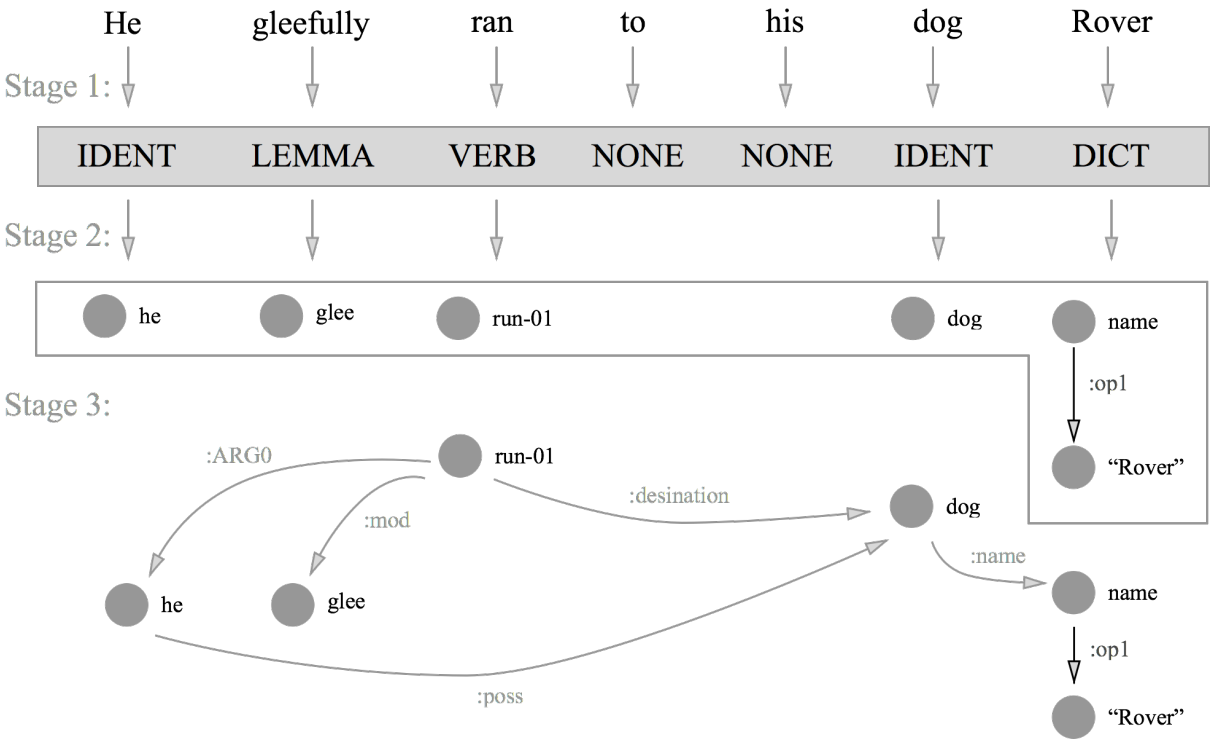

Figure 2: Derivation process for "He gleefully ran to his dog Rover". First the tokens in the sentence are labeled with derivation actions, then those actions are used to generate AMR sub-graphs, and then those sub-graphs are stitched together to form a coherent whole.

"he" will have to be generated directly as nodes. The SRL++ task is then responsible for linking together the output of the NER++ task to produce a fully-connected semantic graph.

For the NER++ task, we propose a small set of 'generative actions' that our system can take to derive an AMR sub-graph from a span of tokens (see Figure 2). For example, we have an action *VERB* that will perform a verb-sense-disambiguation on the source token, like the "ran" to "run-01" example.

We show that end to end performance in-domain is improved from **0.58** smatch to **0.64** smatch when using previously published state of the art SRL++ (Flanigan et al., 2014) to stitch together the output of our NER++ system.

In order to train our NER++ system, we need a correspondence between AMR nodes and their source tokens in the training data. For example, a node with the title "glee" will almost certainly represent the token "gleefully" if that appears in the source text. However, AMR training data is "unaligned", meaning that no effort is made to annotate which token in a sentence is being represented by a given node in an AMR graph.

Automatically inferring alignments is a source of noise in training our NER++ system. We also propose a novel alignment system inspired by our generative actions to explicitly minimize that noise. We cast AMR alignment as the task of finding the alignment of AMR graph to the source sentence that maximizes the informativeness of the implied generative actions. We define this further in Section 5.

2 A Crash-Course in AMR

AMR is a language for expressing semantic understanding that represents meaning as a directed graph, where nodes represent concepts and arcs are relationships between concepts. AMR makes no effort to have a one-to-one correspondence between nodes in a graph and tokens in the sentence whose semantics is being represented. Thus AMR is not a "semantic dependency" representation. AMR represents the relationships between objects referred to by the surface text, not merely the relationships between the words themselves. In fact, AMR will often expand single tokens into large sub-graph elements, or ignore tokens completely.

To introduce AMR and its notation, we'll unpack the translation of the sentence "he gleefully ran to his dog Rover". We show in Figure 1 the interpretation of this sentence as an AMR graph.

Note that the root node of the graph is labeled "run-01". This is the name of a verb sense definition drawn from PropBank [citation needed] for the sense of the verb "ran" in this sentence. This distinguishes this use of the verb "run" from senses like those expressed in "he *ran* the business" or "he gave him a *run* for his money".

"run-01" has an outgoing "ARG0" arc to a node "he", with semantics (drawn from the PropBank frame) that roughly correspond to "he" being the doer of the "run-01" action. The "run-01" has an outgoing "mod" to "glee," which has the catch-all semantics that "run-01" is somehow modified by the concept "glee." "run-01" also has a "destination" arc to "dog," which draws its semantics from Vivek Srikumar's thesis chapter on preposition sense actionging [citation needed], and means that the destination of the "run-01" action is "dog". Then we have a section of the graph that is best interpreted as a unit, where all of the children of "dog" effectively mean that "dog" has the name "Rover."

2.1 Formal task definition

AMR parsing is a challenging task, defined as follows: given a sequence of tokens, generate a directed AMR graph corresponding to the sentence.

Formally, given an array of tokens $S = [s_0, \ldots, s_n]$, generate a directed AMR graph G, defined as the pair $(N = [n_0, \ldots, n_k], A \in L^{k*k})$, where N is an array of AMR nodes (which doesn't have to be the same length or have a clear correspondence to S), and A is a matrix of labels L, where $A_{i,j} = l \in L$ means that an arc exists from node n_i to node n_j with label l in the parsed graph. We include the special label "NONE" in L, corresponding to no arc existing between two nodes.

2.2 NER++

Parsing to the AMR representation demands a rich NER system, word sense disambiguation, number normalization, time parsing, and many semantic nominalizations and part of speech translations, and within-sentence coreference. We refer to these "low-level" AMR tasks collectively as NER++. NER++ is the sub-task of generating the best AMR sub-graphs ("sub-graph" is defined in Section 2.4) given the set of tokens S. This involves both partitioning the source text into spans that will be rendered as a single sub-graph in AMR (e.g. "run", "People's Republic of China", "January 1, 2008"), and then mapping each of those spans into a corresponding AMR sub-graph of maximum likelihood.

Figure 3: AMR representation of the word "sailor", which is notable for breaking the word up into a self-contained multi-node unit signifying some etymological understanding.

2.3 SRL++

Given a perfect NER++ system for an AMR parser, there remains the task of noting the verb arguments, preposition sense actionging, and doing some augmented semantic dependency parsing in order to join the disjoint NER++ output into a single AMR parse. We call this task SRL++. SRL++ is the sub-task of taking as input the disjoint sub-graphs generated by NER++, and adding the maximum likelihood set of arcs between the sub-graphs in order to have a fully connected graph.

2.4 AMR Subgraphs

AMR contains components that, while they may be composed of multiple nodes, can logically be considered the expression of a single concept. For the NER++ task, we would like to be able to generate these "single concept subgraphs" directly from spans of text.

AMR makes an attempt to capture some semantic meanings in words that are difficult to capture in a way that is not domain specific. For example, the token "sailor" in a sentence will evoke the concept graph representing a person who performs the action "sail-01", see Figure 3. This is difficult to model without resorting to memorization, because the etymological clues are so sparse. We note this as an area for further exploration.

AMR can also capture structured data, like time

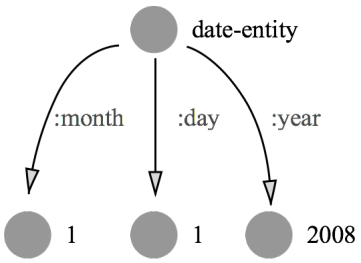

Figure 4: AMR representation of the span "January 1, 2008", an example of how AMR can represent structured data by hallucinating additional nodes like "date-entity" to signify the presence of special structure

expressions, see Figure 4. In dates, a "date-entity" node is hallucinated to signify that this cluster of nodes is part of a structured sub-component of an AMR graph, with specific semantics. Dates are a good example of a recurring pattern in AMR, which is to have an "artificial node" signify that all its immediate children are part of a structured piece of data, with some special interpretation. The most common example of this pattern is the "name" node, which signifies that its immediate children comprise the tokens of a name object.

3 Previous Work

Semantic parsing has been explored extensively. **TODO:** Cite Percy, Zettlemoyer, etc

The twin challenges of unobserved alignments and highly non-projective, potentially cyclic structures makes AMR a novel challenge. At the time of this writing, the JAMR parser (Flanigan et al., 2014) is the only published AMR parser. The crucial insight in JAMR is that AMR parsing can be broken into two relatively distinct tasks: interpreting what entities are being referred to in the text (which we call NER++), and then discovering what relationships those entities have between one another other (which we call SRL++).

For NER++, JAMR uses a simple Viterbi sequence model to directly generate AMR-subgraphs from memorized mappings of text spans to subgraphs. Then for SRL++ JAMR uses a variation of the maximum spanning tree algorithm augmented by dual decomposition to impose linguistically motivated constraints on a maximum likelihood stitching. JAMR's SRL++ component is extremely effective, and we were unable to produce a better SRL++ system in our experiments with several other structured prediction approaches.

4 NER++ Method

Our approach to improving NER++ is very simple: instead of trying to pick which of thousands of AMR sub-graphs to generate from a span of text directly, we partition the AMR sub-graph space in terms of the actions needed to derive a node from its aligned token. At test time we do a sequence labeling of input tokens with these actions, and then deterministically derive the AMR sub-graphs from spans of tokens by applying the transformation decreed by their actions. This dramatically reduces sparsity, and helps improve end-to-end performance, but is most beneficial for domain transfer. We explain in Section 4.2 how exactly we manage this partition, and explain in Section 4.4 how we create training data from existing resources to train a action-type classifier. Then we setup the classifier itself in Section 4.5.

4.1 Derivation actions

We partition the AMR sub-graph space into a set of 7 actions, each corresponding to an action that will be taken by the NER++ system if a token receives this classification.

- **VERB**: Look for the most similar PropBank frame, make that the title of the corresponding node.
- **IDENTITY**: Take the lowercased version of the token to be the title of the corresponding node.
- **VALUE**: Parse the token to an integer value, and use that as the node. AMR actually does type-check, so
- **LEMMA**: Take the lemma of the token to be the title of the corresponding node.
- **NONE**: Ignore this token in the final output.
- **NAME**: Attach a hallucinated "name" node to the top of this span, but don't add an NER action type on top of the "name" node.
- **DICT**: Look up the most probable chunk associate with this lexical span. This functions as a back off if no other actions are appropriate.

4.2 Notes on the DICT action

It's not always possible to derive an AMR sub-graph directly from tokens at test time without having memorized a mapping. For example, the parse of "sailor" as "person who sails", see Figure 3, is nearly impossible without some form of memorization. That's where the **DICT** class is important.

To implement a **DICT** class, we memorize a simple mapping from spans of text, like "sailor" to their corresponding most frequently seen AMR sub-graphs in the training data, in this case Figure 3. At test time we can do a lookup in this dictionary for any element that gets labeled with a **DICT** action. Previous approaches have been the equivalent of labeling every node with the **DICT** action, so our reduction of its use is significant. This is the distribution of actions on the LDC2014T12 proxy training data, after our automatic alignment allows us to induce actions (see Section 4.4 for how this is done).

Action	# Tokens	% Total
NONE	41538	0.371
DICT	30027	0.268
IDENTITY	19034	0.170
VERB	11739	0.104
LEMMA	5029	0.045
NAME	4537	0.04
VALUE	16	0.001

Table 1: Distribution of action types in the proxy section of the LDC2014T12 dataset, generated from automatically aligned data.

Note that **DICT** counts for around 27% of the training data, meaning that more than 72% of tokens can be generated correctly by our action type classifier even if we've never seen them before, which is a huge win.

We believe that **DICT** should count for much less than 27%, and **LEMMA** should count for much more than 4%, but issues with existing lemmatizers prevent this, see our error analysis in Section 7.

4.3 Action Informativeness Hierarchy

We define the concept of "action informativeness" of an action a as the probability of deriving the correct node from a span of tokens, given that those tokens are labeled with the action a, and a is the correct action for that span of tokens.

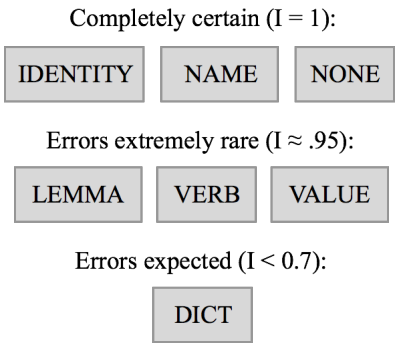

Figure 5: Informativeness hierarchy for action tags within AMR.

To provide a concrete example, our dictionary lookup classifier has a test-set accuracy of 0.67. That means that the "action informativeness" of the **DICT** action is 0.67, because given that we correctly label a token as **DICT**, there is a probability of 0.67 that we correctly generate the corresponding node.

In contrast to **DICT**, correctly labeling a node as **IDENTITY**, **NAME**, and **NONE** have action informativeness of 1.0, since there is no ambiguity in the node generation once one of those actions have been selected, and we are guaranteed (probability 1.0) to generate the correct node given the correct action.

This allows us to induce an action informativeness hierarchy, with more informative actions taking precedence over less informative actions for several important tasks. We demonstrate this hierarchy in Figure 5.

4.4 Inducing Derivation actions from Training Data

Given a set of AMR training data, in the form of (graph,sentence) pairs, we first induce alignments from the graph nodes to the sentence, see Section 5. Given an alignment, which is an annotation on the graph noting for each node N_i a token S_j that is most likely to have "generated" N_i, we can induce alignments. For concreteness, imagine the token S_j is "running", and the node N_i has the title "run-01". For each action type, we can ask whether that action type is able to take token S_j and correctly generate N_i. The two action types we find that are able to correctly generate this node are **DICT** and **VERB**. We choose the most informative action type of those available to generate the observed node. In this case, that means we choose **VERB**.

In general, our algorithm is as follows. For all S_j to which no N_i exists such that N_i aligns to S_j, assign the action **NONE** to S_j. For all pairs N_i, S_j, assign S_j the most informative action possible that could have generated N_i. **TODO:** Flesh out discussion of adjacent DICT nodes

4.5 Action Classifier

We use an extremely simple max-ent classifier to make action decisions. The classifier takes as input a pair $< i, S >$, where i is the index of the token in the input sentence, and S is a sequence tokens representing the source sentence. The output of the classifier is a action T such that the likelihood with respect to the data of token i in sentence S generating a node according to the action specified by T is maximized. See Appendix A for a list of classifier features.

4.6 Test Time Behavior

At test time, given a sequence of input tokens, we do a simple classification of each token separately, to get a sequence labeling of our input tokens. Then for each token, we apply the behavior associated with the token label, and the resulting set of sub-graphs is passed on to SRL++ for linking.

5 Automatic Alignment of Training Data

AMR training data is in the form of bi-text, where we are given a set of (sentence,graph) pairs, with no explicit alignments between them. For example, imagine we are given the graph for "He gleefully ran to his dog Rover", as shown in Figure 1. Although it's obvious to a human, the training data has no reference to the fact that the node "run-01" came from the token "ran". There is therefore a crucial task of generating these alignments prior to running training algorithms.

5.1 Alignment Task Definition

In plain english, we want a projective mapping from nodes to tokens. It is perfectly possible for multiple nodes to align to the same token. It is not possible, within our framework, to represent a single node being sourced from multiple tokens.

To define exactly what is meant by an 'alignment', let there be a pair $G =< N, A >$ where N is a set of nodes and A is an $|N|$ by $|N|$ matrix of binary variables, representing the presence or absence of directed arcs between nodes. For example, $A_{i,j} = T$ means that an arc exists between N_i and N_j, and $A_{i,j} = F$ means that no arc exists between N_i and N_j. Let there be a set of tokens S, such that S_i is the ith token in the source sentence. We would like an array B, where $|B| = |N|$, and for all i, B_i is in the range $(1, |S|)$. For $B_i = n$, it means that token S_n generated N_i.

5.2 Previous Alignment Work

There have been two previous attempts at producing automatic AMR alignments. The first was published as a necessary component of JAMR, (Flanigan et al., 2014), and used a rule-based approach to perform alignments, which worked well on the small sample of 100 hand-labeled sentences used to develop the system. The second published approach, **TODO:** cite short paper, rendered AMR graphs as text, and then used traditional alignment techniques from machine translation to align tokens in the source text and nodes in the AMR graphs. This approach works reasonably well, but fails to take advanactione of the inherently graphical structure of AMR, and regularities within that structure like named entities and quantity values.

5.3 Intuition

Our decomposition of the AMR node generation process into a set of actions provides an interesting way to align unaligned AMR graphs. We would like an alignment of AMR nodes to the source tokens such that we maximize the "informativeness" of the actions that we use to generate the AMR nodes from the source text.

We can define the "informativeness" of a given action by the probability of generating the correct nodes given the correct sequence label. The only label with a probability of correct generation that is less than 1 (i.e. is not and immediate guaranteed win) is **DICT**, which looks up the token in a dictionary, and on our dev set less than 70% are correctly generated from a **DICT**.

That suggests a relatively simple heuristic for producing good alignments: minimize the number of **DICT** sequence labels implied by a given alignment A. We would also like to constrain nodes that are not adjacent to one another to not align to the same token, except in certain cases where hallucinated AMR node structure suggests that a contiguous segment of 3 or more nodes is plausible.

5.4 Boolean Linear Program Formulation

We can formulate the alignment problem and constraints given above as a Boolean LP.

Let Q be a matrix in $\mathcal{B}^{|N|x|S|}$ (Q is a matrix of boolean variables of size $|N|$ x $|S|$). The meaning of $Q_{i,j} = \mathbb{1}$ can be interpreted as node N_i having come from token S_j. Furthermore, let V be a matrix $\mathcal{T}^{|N|x|S|}$ (V is a matrix of derivation types, a set we call $\mathcal{T}$, of size $|N|$ x $|S|$). The matrix element $V_{i,j}$ gets the derivation type that would be implied by node N_i aligning to token S_j. **TODO:** Needs graphic Our goal can then be formulated roughly as follows:

$$\sum_{i,j} Q_{i,j} * \mathbb{1}(V_{i,j} = DICT)$$

We would like to constrain the alignment so that each node must align to exactly one token:

$$\forall i(\sum_{j} Q_{i,j} = 1)$$

It is also useful to prevent nodes that are not adjacent in the AMR graph, and do not have exactly the same title, from aligning to the same token. Let $\mathcal{J}$ be the set of all pairs (k, l) such that $k \neq l$ and N_k and N_l are not adjacent in the graph, and do not have the same title. Then we can enforce this constraint with,

$$\forall (k, l) \in \mathcal{J}(\forall j(Q_{k,j} + Q_{l,j} \leq 1))$$

We also find edit distance to be a useful encouragement for nodes to align to their correct source token, so we would like to linearly augment our goal term with another value to reflect how closely our proposed alignment follows edit distance. Let $\mathcal{E}$ be a matrix in $\mathcal{R}^{|N|x|S|}$, where $E_{i,j}$ is the Jaro-Winkler edit distance between the title of node N_i, and the sentence token S_j. Then we can augment our objective function with a linear encouragement, modulated by α, to align to the close edit-distance concepts overall. Our new augmented objective function is:

$$\sum_{i,j} Q_{i,j} * (\mathbb{1}(V_{i,j} = DICT) - \alpha \mathcal{E}_{i,j})$$

We have many choices for packages that can solve this Boolean LP efficiently. We used Gurobi [citation needed].

Given a matrix Q that minimizes our objective, we can decode our solved alignment as follows: for each i, align N_i to the j s.t. $Q_{i,j} = 1$. By our constraints, exactly one such j must exist.

6 Results

6.1 End to end results

Our end to end results are reported by plugging the output of our NER++ into the SRL++ component of JAMR (Flanigan et al., 2014), which is able to produce final AMR graphs when given a sequence of spans and their corresponding chunks. AMR parsing accuracy is measured with a metric called smatch [citation needed], which stands for "s(emantic) match". We trained and tested on the **LDC2013E117** dataset, for which the only published result is a smatch score of **0.58** on the test set by JAMR (Flanigan et al., 2014). We report **0.64** on the same dataset, by substituting our NER++ system.

6.2 Alignment results

We hand annotated 500 sentence graph pairs with alignments. These pairs were selected to evenly represent every domain from the LDC2014T12 dataset, and were hand-annotated over a period of three weeks by a single individual, so alignment style is consistent throughout. Hallucinated nodes, like "temporal-entity", are aligned to their left-most child for consistency, with the reasoning that they will then grouped in a **DICT** action with their children during training and testing.

TODO: Report Results

7 Error Analysis

7.1 Weak lemmatization

The **DICT** class was intended to be used for things that a system cannot know without memorization, like "sailor". These don't occur nearly 25% of the time. One of the reasons that the **DICT** class is so disappointingly large is that it's stealing from **LEMMA**, because AMR will aggressively normalize words and change their part of speech to a semantic neighbor. For example, 'gleefully' gets mapped to 'glee' and not 'gleeful', which is hard to do automatically with stemming rules in the general case. We leave this as a direction for future work.

7.2 Linear Classifier

8 Future Work

8.1 Semantically equivalent POS normalization

The benefit of this approach could be increased by having a very strong stemmer tuned to AMR parsing, which currently doesn't exist.

8.2 Etymological approach to generation

There is an opportunity to create and test etymological-semantic approaches to parsing words like 'sailor' that would benefit AMR parsing domain generalization tremendously.

9 Appendix

NER++ Features
Input token
Input token word embedding
Left token
Right token
Left bigram
Right bigram
POS
Left POS
Right POS
Left POS bigram
Right POS bigram
Token's dependency parent token
Token's dependency parent POS
Token's dependency parent arc name
Bag of outgoing dependency arcs
Number of outgoing dependency arcs
Number of outgoing dependency arcs (indicator)
Max JaroWinker to any lemma in PropBank
Closest (JaroWinkler) in PropBank
Token NER
Left NER bigram
Right NER bigram
Right NER bigram
Indicator for if token is a recognized AMR NER type
Indicator for if token is capitalized
Parent arc is prep_* or appos, and parent has NER action
Indicator for token is pronoun
Indicator for token is part of a coref chain
Indicator for token pronoun and part of a coref chain

Table 2: The features for the NER++ max-ent classifiers.

Acknowledgments

The acknowledgments should go immediately before the references. Do not number the acknowledgments section. Do not include this section when submitting your paper for review.

References

Jeffrey Flanigan, Sam Thomson, Jaime Carbonell, Chris Dyer, Noah A. Smith 2014. *ACL 14*, volume 1.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. *Proc. of the Linguistic Annotation Workshop and Iteroperability with Discourse*, volume 1.

Alfred V. Aho and Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation and Compiling*, volume 1. Prentice-Hall, Englewood Cliffs, NJ.

American Psychological Association. 1983. *Publications Manual*. American Psychological Association, Washington, DC.

Association for Computing Machinery. 1983. *Computing Reviews*, 24(11):503–512.

Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. 1981. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133.

Dan Gusfield. 1997. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, Cambridge, UK.