Generating English from Abstract Meaning Representations

Nima Pourdamghani , Kevin Knight , Ulf Hermjakob
Information Sciences Institute
Department of Computer Science
University of Southern California
{damghani,knight,ulf}@isi.edu

Abstract

We present a method for generating English sentences from Abstract Meaning Representation (AMR) graphs, exploiting a parallel corpus of AMRs and English sentences. We treat AMR-to-English generation as phrase-based machine translation (PBMT). We introduce a method that learns to linearize tokens of AMR graphs into an English-like order. Our linearization reduces the amount of distortion in PBMT and increases generation quality. We report a Bleu score of 26.8 on the standard AMR/English test set.

1 Introduction

Banarescu et al. (2013) introduce Abstract Meaning Representation (AMR) graphs to represent sentence level semantics. Human annotators have created a dataset of more than $10,000$ AMR/English string pairs.

AMRs are directed acyclic graphs, where leaves are labeled with concepts, internal nodes are labeled with variables representing instances of those concepts, and edges are labeled with roles that relate pairs of concepts. For instance, the sentence *The boy wants to go* is represented as:

```
(w :instance-of want-01
   :arg0 (b :instance-of boy)
   :arg1 (g :instance-of go-01
            :arg0 b))
```

Colons discriminate roles from concepts. In this paper, *:instance-of* is our way of writing the slash (*/*) found in the AMR corpus.

Because AMR and English are highly cognate, the AMR-to-English generation problem might seem similar to previous natural language generation (NLG) problems such as bag generation (Brown et al., 1990), restoring order to unordered dependency trees (Guo et al., 2011) or generation from logical form (Corston-Oliver et al., 2002). However, AMR's deeper logic provides a serious challenge for English realization. AMR also abstracts away details of time, number, and voice, which must be inserted.

Langkilde and Knight (1998) introduced Nitrogen, which used a precursor of AMR for generating English. Recently, Flanigan et al. (2016) presented the first trained AMR-to-English generator. They generate spanning trees from AMR graphs and apply tree-to-string transducers to the trees to generate English.

We attack AMR-to-English generation using the tools of phrase-based machine translation (PBMT). PBMT has already been applied to natural language generation from simple semantic structures (Mairesse et al., 2010), but deep semantic representations such as AMR are more challenging to deal with. PBMT expects strings for its source and target languages, so we cannot work with AMR graphs as input. Therefore, we develop a method that *learns to linearize AMR graphs* into AMR strings. Our linearization strives to put AMR tokens roughly into English word order, making the transformation to English easier.

It may seem surprising that we ignore much of the structure of AMR, but we follow string-based statistical MT, which ignored much of the structure of

Proceedings of The 9th International Natural Language Generation conference, pages 21–25,
Edinburgh, UK, September 5-8 2016.

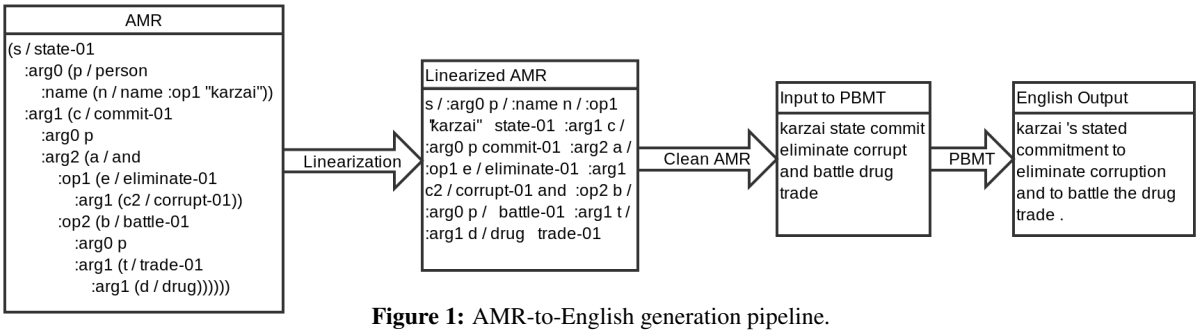

Figure 1: AMR-to-English generation pipeline.

language but nonetheless provided a strong baseline.

Figure 1 shows our pipeline for generating English from AMR. Our contributions are:

1. We present a strong baseline method for AMR-to-English generation.
2. We introduce a method that learns to linearize AMR tokens into an order resembling English.
3. We obtain a Bleu score of 26.8 on the standard AMR/English test set, which is 4.9 points higher than previous work.

2 Method

Given a set of AMR/English pairs, divided into train, development, and test sets, we follow these steps:

Construct token-level alignments: We use the method proposed in (Pourdamghani et al., 2014) to construct alignments between AMR and English tokens in the training set.

Extend training data: We use special realization components for names, dates, and numbers found in the dev/test sets, adding their results to the training corpus.

Linearize AMR graphs: We learn to convert AMR graphs into AMR strings in a way that linearized AMR tokens have an English-like order (Section 3).

Clean AMR strings: We remove variables, quote marks, and sense tags from linearized AMRs. We also remove **-quantity* and **-entity* concepts, plus these roles: *:op*, :snt*, :arg0, :arg1, :arg2, :name, :quant, :unit, :value, :year, :domain-of*.

Phrase-Based Machine Translation: We use Moses (Koehn et al., 2007) to train and tune a PBMT system on string/string training data. We then use this system to produce English realizations from linearized development and test AMRs.

3 Linearization

When we linearize AMR, we would like—at a minimum—for semantically-related tokens to stay close together. A straightforward, pre-order depth first search (DFS) accomplishes this (Pourdamghani et al., 2014). For instance, linearizing

```
(w :instance-of want-01
   :arg0 (b :instance-of boy)
   :arg1 (g :instance-of go-01
            :arg0 b))
```

yields "*w :instance-of* ***want-01*** *:arg0 b :instance-of* ***boy*** *:arg1 g :instance-of* ***go-01*** *:arg0 b*".

Of course, we are free to visit AMR sister nodes in any order. For instance, if we visit sisters in order *(:arg0, :instance-of, :arg1)*, we get this string instead: "*w :arg0 b :instance-of* ***boy*** *:instance-of* ***want-01*** *:arg1 g :instance-of* ***go-01*** *:arg0 b*" , which more resembles English word order.

We therefore induce an ordering function that takes any set of edge labels as input and produces a permutation of those labels. We call this the *linearization function*.

The input to this function is a sequence consisting of the concept under the *:instance-of* edge (e.g., *want-01*) followed by the other edges sorted alphabetically (e.g., *:arg0 :arg1*). The output is a permutation of the input (e.g., *(2, 1, 3)*).

Because *:instance-of* concepts often have no equivalent in English, e.g.:

```
(n :instance-of name
   :op1 "Pierre"       -> Pierre Vinken
   :op2 "Vinken")
```

we additionally allow the first component of the output to be "-1", indicating deletion.

Our linearization function therefore has the following form:

$$p : \{c, r_1, r_2, ..., r_{k-1}\} \rightarrow (\pi_1, \pi_2, ..., \pi_k) \quad (1)$$

where c is a concept token, r_i are role tokens, $\pi_{i>1} \in \{1, 2, ..., k\}$ and $\pi_1 \in \{-1, 1, 2, ..., k\}$.

Here are sample input/output pairs for the linearization function:

```
(want-01, :arg0, :arg1) -> (2, 1, 3)
(name, :op1, :op2) -> (-1, 1, 2)
(and, :op1, :op2) -> (2, 1, 3)
(area-quantity, :quant, :unit) -> (-1, 1, 2)
(win-01, :arg0, :arg1, :time) -> (2, 1, 3, 4)
```

Our overall objective is to minimize the number of crossings in the alignment links after linearization. We use our token-aligned AMR/English data to produce training examples for the function (1). We assign each outgoing AMR edge a position equal to the median of the alignment points of all tokens in its subtree, including the edge itself. We assign -1 to an edge if none of its subtree tokens are aligned. Then we extract all sets of sibling edges in the AMR graph, and sort them based on these numbers. We use these sorted sets to create training instances.

We now describe three linearization methods.

3.1 Pre-order DFS

This baseline method linearizes AMR by simple pre-order traversal, ignoring the data just described.

3.2 Majority Method

The majority method memorizes the most common order for each role set in the data. If no match is found, we use the ordering given in the original, human-annotated AMR, with the *:instance-of* edge first.

3.3 Classifier Method

The classifier method breaks the problem into learning three binary classifiers over inputs of the form $(c, r_1, r_2, ..., r_{k-1})$:

1. Should the *:instance-of* edge be dropped?
   - Features: k, c, (c, r_i), whether c is a Propbank frameset, and whether c is a "special keyword" as defined by Banarescu et al. (2013).
2. Should edge r_i appear before *:instance-of*?
   - Features: r_i, (c, r_i), (r_i, r_j) for all $j \neq i$
3. Should edge r_i appear before r_j?
   - Features: (c, r_i, r_j)

We use the toolkit of Zhang (2004) to learn a maximum entropy classifier for each task.

	AMR/English pairs	English word tokens
Train	10,313	218,021
Dev	1,368	29,848
Test	1,371	30,263

Table 1: Data for AMR-to-English generation.

After training, for a given input query, we consult the first classifier on whether or not to drop the *:instance-of* edge.

If we drop this edge, we consider the rest of the edges as one group; otherwise, we divide them into two groups each appearing on one side of the *:instance-of* edge, using the second classifier.

Next, we order the edges within each group. Let P($r_i < r_j$) be the probability—according to the third classifier—that r_i precedes r_j. For each edge r_i, we assign it a "left-leaning" score, which is the product of all P($r_i < r_j$), for all $j \neq i$. We remove the edge with the highest left-leaning score. We then recursively process the remaining edges in the group.

We were inspired by Lerner and Petrov (2013) to break the problem down this way. Because their dependencies are ordered, while our AMRs edges are not, we defined a different set of features and classifiers.

4 Experiments

We use AMR/English data from the AMR 1.0 corpus,[1] along with the provided train/development/test split (Table 1).

We implement the method of Pourdamghani et al. (2014) to construct alignments for the training set. We train the linearization function introduced in Section 3 on the aligned training set and use it to re-linearize that training set, maintaining the alignment links. This gives us aligned string-to-string training data for PBMT. We use the same trained linearization function to linearize development and test AMRs.

To measure the quality of linearization, we make calculations on the development set, using alignments to references (these alignments are used only for this experiment, and not for decoding).

A good linearization function should: (a) reduce the number of crossings in the alignment links, and (b) correctly identify concepts to be dropped.

[1] LDC Catalog number 2014T12.

	Crossings	Adj. crossings
Pre-order DFS	46671	7409
Majority Method	33772 (72%)	4850 (65%)
Classifier Method	35603 (76%)	4015 (54%)

Table 2: Total alignment crossings, and crossings between adjacent links after linearizing development AMRs with different methods. Numbers in parentheses show the reduction compared to Pre-order DFS.

	Dev Bleu	Test Bleu
1: Pre-order DFS	17.7	16.6
1a: 1 + clean AMRs	21.6	21.0
1b: 1a + name/number/date	23.5	22.5
2: Majority Method	26.5	25.6
3: Classifier Method	27.2	26.9
Flanigan et al. (2016)	22.7	22.0

Table 3: Results for AMR-to-English generation on development and test data. Experiments 2 and 3 include cleaning AMRs and name/number/date translations. Bleu scores are single-reference, case insensitive, {1..4}-grams.

Table 2 shows the total number of crossings and number of crossings between adjacent alignment links after linearizing development AMRs with the three methods introduced in Section 3. Both advanced methods highly reduce the number of crossings. The Classifier Method reduces the number of adjacent crossings much more than the Majority Method, helping to enhance locality. End-to-end experiments (Table 3) show that the Classifier Method outperforms the Majority Method in improving Bleu score.

With respect to concept dropping, 97% of the concepts dropped by the Classifier Method are in fact not aligned, and the method correctly drops 87% of the unaligned concepts.

Next, we use the Moses (Koehn et al., 2007) system for our PBMT implementation. Phrase extraction, limited to maximum phrase length 9, yields 1.2m phrase pairs. We use a 5-gram language model trained on 1.7b tokens of Gigaword English. We use MERT for tuning, and we decode linearized AMRs into English with a maximum stack size of 1000.

Table 3 shows our results. We find that better linearization methods lead to better Bleu scores. The Majority Method outperforms Pre-order DFS by 3.1 Bleu on test data, and the Classifier Method adds another 1.2 Bleu. We also find that steps of cleaning and specialized name/number/date generators significantly improve Bleu. Compared to (Flanigan et al., 2016) our best system achives 4.5 Bleu points improvement on dev and 4.9 points improvement on test data.

Here is a small-sized input/output example from the automatic AMR-to-English generation system:

Input AMR:

```
(s / state-01
  :arg0 (p / person
    :name (n / name :op1 "fan"))
  :arg1 (c / concern-01
    :arg1 (c3 / commission)
    :arg2 (t / term
      :mod (i / invest-01
        :arg2 (c2 / country
          :name (n3/name :op1 "taiwan"))
        :time (f / future)))
    :manner (p2 / primary)))
```

Linearized, Cleaned AMR: fan state commission :manner primary concern invest taiwan :time future term

System Output: fans who have stated that the commission is primarily concerned with the terms of the investment in taiwan in the future .

Gold English: fan stated the commission is primarily concerned with the term of future investment in taiwan .

5 Conclusion

We introduce a method for learning to generate English from AMR. We use phrase-based machine translation technology and carry out experiments to compare different AMR linearization methods. We show that our method outperforms prior work by a large margin. We consider our results to form a strong baseline for future work.

References

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 *Proc. ACL Linguistic Annotation Workshop (LAW)*.

Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, John D. Laf-

ferty, Robert L. Mercer, and Paul S. Roossin. 1990. A statistical approach to machine translation. *Computational linguistics*, 16(2):79–85.

Simon Corston-Oliver, Michael Gamon, Eric Ringger, and Robert Moore. 2002. An overview of Amalgam: A machine-learned generation module. In *Proc. INLG*.

Jeffrey Flanigan, Chris Dyer, Noah A. Smith, and Jaime Carbonell. 2016. Generation from abstract meaning representation using tree transducers. In *Proc. NAACL*.

Yuqing Guo, Haifeng Wang, and Josef Van Genabith. 2011. Dependency-based n-gram models for general purpose sentence realisation. *Natural Language Engineering*, 17(4):455–483.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. 2007. Moses: Open source toolkit for statistical machine translation. In *Proc. ACL Poster and Demonstration Sessions*.

Irene Langkilde and Kevin Knight. 1998. Generation that exploits corpus-based statistical knowledge. In *Proc. ACL*.

Uri Lerner and Slav Petrov. 2013. Source-side classifier preordering for machine translation. In *Proc. EMNLP*.

François Mairesse, Milica Gašić, Filip Jurčíček, Simon Keizer, Blaise Thomson, Kai Yu, and Steve Young. 2010. Phrase-based statistical language generation using graphical models and active learning. In *Proc. ACL*.

Nima Pourdamghani, Yang Gao, Ulf Hermjakob, and Kevin Knight. 2014. Aligning English strings with Abstract Meaning Representation graphs. In *Proc. EMNLP*.

Le Zhang. 2004. Maximum entropy modeling toolkit for Python and C++. http://bit.ly/1DGnb2p.