AMR-to-text Generation with Synchronous Node Replacement Grammar

Linfeng Song, Xiaochang Peng, Yue Zhang, Zhiguo Wang and Daniel Gildea
Department of Computer Science, University of Rochester, Rochester, NY 14627
IBM T.J. Watson Research Center, Yorktown Heights, NY 10598
Singapore University of Technology and Design

Abstract

This paper addresses the task of AMR-to-text generation by leveraging synchronous node replacement grammar. During training, graph-to-string rules are learned using a heuristic extraction algorithm. At test time, a graph transducer is applied to collapse input AMRs and generate output sentences. Evaluated on a standard benchmark, our method gives the state-of-the-art result.

1 Introduction

Abstract Meaning Representation (AMR) (Banarescu et al., 2013) is a semantic formalism encoding the meaning of a sentence as a rooted, directed graph. AMR uses a graph to represent meaning, where nodes (such as “boy”, “want-01”) represent concepts, and edges (such as “ARG0”, “ARG1”) represent relations between concepts. Encoding many semantic phenomena into a graph structure, AMR is useful for NLP tasks such as machine translation (Jones et al., 2012; Tamchyna et al., 2015), question answering (Mitra and Baral, 2015), summarization (Takase et al., 2016) and event detection (Li et al., 2015).

AMR-to-text generation is challenging as function words and syntactic structures are abstracted away, making an AMR graph correspond to multiple realizations. Despite much literature so far on text-to-AMR parsing (Flanigan et al., 2014; Wang et al., 2015; Peng et al., 2015; Vanderwende et al., 2015; Pust et al., 2015; Artzi et al., 2015; Groschwitz et al., 2015; Goodman et al., 2016; Zhou et al., 2016; Peng et al., 2017), there has been little work on AMR-to-text generation (Flanigan et al., 2016; Song et al., 2016; Pourdamghani et al., 2016).

Flanigan et al. (2016) transform a given AMR graph into a spanning tree, before translating it to a sentence using a tree-to-string transducer. Their method leverages existing machine translation techniques, capturing hierarchical correspondences between the spanning tree and the surface string. However, it suffers from error propagation since the output is constrained given a spanning tree due to the projective correspondence between them. Information loss in the graph-to-tree transformation step cannot be recovered. Song et al. (2016) directly generate sentences using graph-fragment-to-string rules. They cast the task of finding a sequence of disjoint rules to transduce an AMR graph into a sentence as a traveling salesman problem, using local features and a language model to rank candidate sentences. However, their method does not learn hierarchical structural correspondences between AMR graphs and strings.

We propose to leverage the advantages of hierarchical rules without suffering from graph-to-tree errors by directly learning graph-to-string rules. As shown in Figure 1, we learn a synchronous node replacement grammar (NRG) from a corpus of aligned AMR and sentence pairs. At test time, we apply a graph transducer to collapse input
AMR graphs and generate output strings according to the learned grammar. Our system makes use of a log-linear model with real-valued features, tuned using MERT (Och, 2003), and beam search decoding. It gives a BLEU score of 25.62 on LDC2015E86, which is the state-of-the-art on this dataset.

2 Synchronous Node Replacement Grammar

2.1 Grammar Definition

A synchronous node replacement grammar (NRG) is a rewriting formalism: \( G = \langle N, \Sigma, \Delta, P, S \rangle \), where \( N \) is a finite set of nonterminals, \( \Sigma \) and \( \Delta \) are finite sets of terminal symbols for the source and target sides, respectively. \( S \in N \) is the start symbol, and \( P \) is a finite set of productions. Each instance of \( P \) takes the form \( X_i \to (\langle F, E \rangle, \sim) \), where \( X_i \in N \) is a nonterminal node, \( F \) is a rooted, connected AMR fragment with edge labels over \( \Sigma \) and node labels over \( N \cup \Sigma \), \( E \) is a corresponding target string over \( N \cup \Delta \) and \( \sim \) denotes the alignment of nonterminal symbols between \( F \) and \( E \). A classic NRG (Engelfriet and Rozenberg, 1997, Chapter 1) also defines \( C \), which is an embedding mechanism defining how \( F \) is connected to the rest of the graph when replacing \( X_i \) with \( F \) on the graph. Here we omit defining \( C \) and allow arbitrary connections.\(^1\)

Following Chiang (2005), we use only one nonterminal \( X \) in addition to \( S \), and use subscripts to distinguish different non-terminal instances.

![Figure 2: Example deduction procedure](image)

**Algorithm 1: Rule extraction**

\[
\text{Data: training corpus } C \\
\text{Result: rule instances } R \\
1 \quad R \leftarrow \{ \}; \\
2 \quad \text{for } (\text{Sent}, AMR, \sim) \text{ in } C \text{ do} \\
3 \quad \quad R_{\text{corr}} \leftarrow \text{FRAGMENTS} \text{EXTRACT}(\text{Sent}, AMR, \sim); \\
4 \quad \quad \text{for } r_i \text{ in } R_{\text{corr}} \text{ do} \\
5 \quad \quad \quad \text{if } r_i \text{ contains } \sim \text{ then} \\
6 \quad \quad \quad \quad r_{ij} \leftarrow r_i \text{ COLLAPSE}(r_j); \\
7 \quad \quad \quad \text{end} \\
8 \quad \quad \text{end} \\
9 \quad \text{end} \\
10 \quad \text{end} \\
11 \]

\[^1\]This may over generate, but does not affect our case, as in our bottom-up decoding procedure (section 3) when \( F \) is replaced with \( X_i \), nodes previously connected to \( F \) are reconnected to \( X_i \).

\[^2\]\( \sim \) denotes alignment between words and AMR labels.
contains only terminal symbols in both $F$ and $E$. As a next step, we match between pairs of initial rules $r_i$ and $r_j$, and generate $r_{ij}$ by collapsing $r_i$ with $r_j$, if $r_i$ contains $r_j$ (Line 6-8). Here $r_i$ contains $r_j$, if $r_j.F$ is a subgraph of $r_i.F$ and $r_j.E$ is a sub-phrase of $r_i.E$. When collapsing $r_i$ with $r_j$, we replace the corresponding subgraph in $r_i.F$ with a new non-terminal node, and the sub-phrase in $r_i.E$ with the same non-terminal. For example, we obtain rule (b) by collapsing (d) with (a) in Table 1. All initial and generated rules are stored in a rule list $R$ (Lines 5 and 9), which will be further normalized to obtain the final induced rule set.

2.3 Concept Rules and Glue Rules

In addition to induced rules, we adopt concept rules (Song et al., 2016) and graph glue rules to ensure existence of derivations. For a concept rule, $F$ is a single node in the input AMR graph, and $E$ is a morphological string of the node concept. A concept rule is used in case no induced rule can cover the node. We refer to the verbalization list\(^3\) and AMR guidelines\(^4\) for creating more complex concept rules. For example, one concept rule created from the verbalization list is “(k / keep-01 :ARG1 (p / peace)) ||| peacekeeping”.

Inspired by Chiang (2005), we define graph glue rules to concatenate non-terminal nodes connected with an edge, when no induced rules can be applied. Three glue rules are defined for each type of edge label. Taking the edge label “ARG0” as an example, we create the following glue rules:

<table>
<thead>
<tr>
<th>ID</th>
<th>$F$</th>
<th>$E$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1$</td>
<td>$(X1 / #X1# :ARG0 (X2 / #X2#))$</td>
<td>$#X1# #X2#$</td>
</tr>
<tr>
<td>$r_2$</td>
<td>$(X1 / #X1# :ARG0 (X2 / #X2#))$</td>
<td>$#X2# #X1#$</td>
</tr>
<tr>
<td>$r_3$</td>
<td>$(X1 / #X1# :ARG0 X1)$</td>
<td>$#X1#$</td>
</tr>
</tbody>
</table>

where for both $r_1$ and $r_2$, $F$ contains two non-terminal nodes with a directed edge connecting them, and $E$ is the concatenation the two non-terminals in either the monotonic or the inverse order. For $r_3$, $F$ contains one non-terminal node with a self-pointing edge, and $E$ is the non-terminal. With concept rules and glue rules in our final rule set, it is easily guaranteed that there are legal derivations for any input AMR graph.

3 Model

We adopt a log-linear model for scoring search hypotheses. Given an input AMR graph, we find the highest scored derivation $t^*$ from all possible derivations $t$:

$$t^* = \arg\max_t \exp \sum_i w_i f_i(g, t), \quad (1)$$

where $g$ denotes the input AMR, $f_i(\cdot, \cdot)$ and $w_i$ represent a feature and the corresponding weight, respectively. The feature set that we adopt includes phrase-to-graph and graph-to-phrase translation probabilities and their corresponding lexicalized translation probabilities (section 3.1), language model score, word count, rule count, re-ordering model score (section 3.2) and moving distance (section 3.3). The language model score, word count and phrase count features are adopted from SMT (Koehn et al., 2003; Chiang, 2005).

We perform bottom-up search to transduce input AMRs to surface strings. Each hypothesis contains the current AMR graph, translations of collapsed subgraphs, the feature vector and the current model score. Beam search is adopted, where hypotheses with the same number of collapsed edges and nodes are put into the same beam.

3.1 Translation Probabilities

Production rules serve as a basis for scoring hypotheses. We associate each synchronous NRG with the same number of collapsed derivations $t$, and $g$ is a subgraph of $t$ ($\forall t \ni g$ with a set of probabilities. First, phrase-to-fragment translation probabilities are defined based on maximum likelihood estimation (MLE), as shown in Equation 2, where $c(F,E)$ is the fractional count of $\langle F, E \rangle$.

$$p(F|E) = \frac{c(F,E)}{\sum_{F'} c(F',E)} \quad (2)$$

In addition, lexicalized translation probabilities are defined as:

$$p_w(F|E) = \prod_{l \in F} \sum_{w \in E} p(l|w) \quad (3)$$

Here $l$ is a label (including both edge labels such as “ARG0” and concept labels such as “want-01”) in the AMR fragment $F$, and $w$ is a word in the phrase $E$. Equation 3 can be regarded as a “soft” version of the lexicalized translation probabilities adopted by SMT, which picks the alignment yielding the maximum lexicalized probability for each translation rule. In addition to $p(F|E)$ and $p_w(F|E)$, we use features in the reverse direction, namely $p(E|F)$ and $p_w(E|F)$, the definitions of which are omitted as they are consistent with
Equations 2 and 3, respectively. The probabilities associated with concept rules and glue rules are manually set to 0.0001.

3.2 Reordering Model

Although the word order is defined for induced rules, it is not the case for glue rules. We learn a reordering model that helps to decide whether the translations of the nodes should be monotonic or inverse given the directed connecting edge label. The probabilistic model using smoothed counts is defined as:

\[ p(M|h, l, t) = \frac{1.0 + \sum_{h} \sum_{t} c(h, l, t, M)}{2.0 + \sum_{o \in \{M, I\}} \sum_{h} \sum_{t} c(h, l, t, o)} \]  (4)

\( c(h, l, t, M) \) is the count of monotonic translations of head \( h \) and tail \( t \), connected by edge \( l \).

3.3 Moving Distance

The moving distance feature captures the distances between the subgraph roots of two consecutive rule matches in the decoding process, which controls a bias towards collapsing nearby subgraphs consecutively.

4 Experiments

4.1 Setup

We use LDC2015E86 as our experimental dataset, which contains 16833 training, 1368 dev and 1371 test instances. Each instance contains a sentence, an AMR graph and the alignment generated by a heuristic aligner. Rules are extracted from the training data, and model parameters are tuned on the dev set. For tuning and testing, we filter out sentences with more than 30 words, resulting in 1103 dev instances and 1055 test instances. We train a 4-gram language model (LM) on gigaword (LDC2011T07), and use BLEU (Papineni et al., 2002) as the evaluation metric. MERT is used (Och, 2003) to tune model parameters on \( k \)-best outputs on the dev set, where \( k \) is set 50.

We investigate the effectiveness of rules and features by ablation tests: “NoInducedRule” does not adopt induced rules, “NoConceptRule” does not adopt concept rules, “NoMovingDistance” does not adopt the moving distance feature, and “NoReorderingModel” disables the reordering model. Given an AMR graph, if “NoConceptRule” cannot produce a legal derivation, we concatenate existing translation fragments into a final translation, and if a subgraph can not be translated, the empty string is used as the output. We also compare our method with previous works, in particular JAMR-gen (Flanigan et al., 2016) and TSP-gen (Song et al., 2016), on the same dataset.

4.2 Main results

The results are shown in Table 2. First, All outperforms all baselines. NoInducedRule leads to the greatest performance drop compared with All, demonstrating that induced rules play a very important role in our system. On the other hand, NoConceptRule does not lead to much performance drop. This observation is consistent with the observation of Song et al. (2016) for their TSP-based system. NoMovingDistance leads to a significant performance drop, empirically verifying the fact that the translations of nearby subgraphs are also close. Finally, NoReorderingModel does not affect the performance significantly, which can be because the most important reordering patterns are already covered by the hierarchical induced rules. Compared with TSP-gen and JAMR-gen, our final model All improves the BLEU from 22.44 and 23.00 to 25.62, showing the advantage of our model. To our knowledge, this is the best result reported so far on the task.

4.3 Grammar analysis

We have shown the effectiveness of our synchronous node replacement grammar (SNRG) on the AMR-to-text generation task. Here we further analyze our grammar as it is relatively less studied than the hyperedge replacement grammar (HRG) (Drewes et al., 1997).

Statistics on the whole rule set

We first categorize our rule set by the number of terminals and nonterminals in the AMR fragment \( F \), and show the percentages of each type in Figure 3. Each rule contains at most 1 nonterminal, as we collapse each initial rule only once. First
of all, the percentage of rules containing nonterminals are much more than those without nonterminals, as we collapse each pair of initial rules (in Algorithm 1) and the results can be quadratic the number of initial rules. In addition, most rules are small containing 1 to 3 terminals, meaning that they represent small pieces of meaning and are easier to matched on a new AMR graph. Finally, there are a few large rules, which represent complex meaning.

**Statistics on the rules used for decoding**

In addition, we collect the rules that our well-tuned system used for generating the 1-best output on the testset, and categorize them into 3 types: (1) glue rules, (2) nonterminal rules, which are not glue rules but contain nonterminals on the right-hand side and (3) terminal rules, whose right-hand side only contain terminals. Over the rules used on the 1-best result, more than 30% are non-terminal rules, showing that the induced rules play an important role. On the other hand, 30% are glue rules. The reason is that the data sparsity for graph grammars is more severe than string-based grammars (such as CFG), as the graph structures are more complex than strings. Finally, terminal rules take the largest percentage, while most are induced rules, but not concept rules.

**Rule examples**

Finally, we show some rules in Table 4, where $F$ and $E$ are the right-hand-side AMR fragment and phrase, respectively. For the first rule, the root of $F$ is a verb (“give-01”) whose subject is a nonterminal and object is a AMR fragment “(:ARG0 X1 / #X1#)”, which means “user”. So it is easy to see that the corresponding phrase $E$ conveys the same meaning. For the second rule, “(s3 / stay-01 :accompanier (i / i))” means “stay with me”, which is also covered by its phrase.

### Table 3: Rules used for decoding.

<table>
<thead>
<tr>
<th></th>
<th>Glue</th>
<th>Nonterminal</th>
<th>Terminal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-best</td>
<td>30.0%</td>
<td>30.1%</td>
<td>39.9%</td>
</tr>
</tbody>
</table>

**Table 4: Example rules.**

<table>
<thead>
<tr>
<th>$F$:</th>
<th>$E$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ARG0 (X1 / #X1#)</td>
<td>#X1# has given users an</td>
</tr>
<tr>
<td>:ARG2 (p / person :ARG0-of (u / use-01)))</td>
<td>(X1 / #X1#)</td>
</tr>
</tbody>
</table>

**Table 5: Generation example.**

<table>
<thead>
<tr>
<th>$F$:</th>
<th>$E$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(g / give-01</td>
<td>E:</td>
</tr>
<tr>
<td>:ARG0 (X1 / #X1#)</td>
<td>#X1# staying with me</td>
</tr>
<tr>
<td>:ARG2 (p / person :ARG0-of (u / use-01)))</td>
<td></td>
</tr>
</tbody>
</table>

4.4 Generation example

Finally, we show an example in Table 5, where the top is the input AMR graph, and the bottom is the generation result. Generally, most of the meaning of the input AMR are correctly translated, such as “:example”, which means “such as”, and “thing”, which is an abstract concept and should not be translated, while there are a few errors, such as “that” in the result should be “what”, and there should be an “in” between “tmt” and “fairfax”.

5 Conclusion

We showed that synchronous node replacement grammar is useful for AMR-to-text generation by developing a system that learns a synchronous NRG in the training time, and applies a graph transducer to collapse input AMR graphs and generate output strings according to the learned grammar at test time. Our method performs better than the previous systems, empirically proving the advantages of our graph-to-string rules.

Acknowledgement

This work was funded by a Google Faculty Research Award. Yue Zhang is funded by NSFC61572245 and T2MOE201301 from Singapore Ministry of Education.
References


