Generation from Abstract Meaning Representation using Tree Transducers

Jeffrey Flanigan♠ Chris Dyer♠ Noah A. Smith♥ Jaime Carbonell♠
♠School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
♥Computer Science & Engineering, University of Washington, Seattle, WA, USA
{jflanigan,cdyer,jgc}@cs.cmu.edu, nasmith@cs.washington.edu

Abstract

Language generation from purely semantic representations is a challenging task. This paper addresses generating English from the Abstract Meaning Representation (AMR), consisting of re-entrant graphs whose nodes are concepts and edges are relations. The new method is trained statistically from AMR-annotated English and consists of two major steps: (i) generating an appropriate spanning tree for the AMR, and (ii) applying tree-to-string transducers to generate English. The method relies on discriminative learning and an argument realization model to overcome data sparsity. Initial tests on held-out data show good promise despite the complexity of the task. The system is available open-source as part of JAMR at:
http://github.com/jflanigan/jamr

1 Introduction

We consider natural language generation from the Abstract Meaning Representation (AMR; Banarescu et al., 2013). AMR encodes the meaning of a sentence as a rooted, directed, acyclic graph, where concepts are nodes, and edges are relationships among the concepts.

Because AMR models propositional meaning\footnote{In essence, AMR handles semantic roles, entity types, within-sentence coreference, discourse connectives, modality, negation, and some other phenomena.} while abstracting away from surface syntactic realizations, and is designed with human annotation in mind, it suggests a separation of (i) engineering the application-specific propositions that need to be communicated about from (ii) general-purpose realization details, modeled by a generator shareable across many applications. The latter is our focus here.

Because any AMR graph has numerous valid realizations, and leaves underspecified many important details—including tense, number, definiteness, whether a concept should be referred to nominally or verbally, and more—transforming an AMR graph into an English sentence is a nontrivial problem.

To our knowledge, our system is the first for generating English from AMR. The approach is a statistical natural language generation (NLG) system, trained discriminatively using sentences in the AMR bank (Banarescu et al., 2013). It first transforms the graph into a tree, then decodes into a string using a weighted tree-to-string transducer and a language model (Graehl and Knight, 2004). The decoder bears a strong similarity to state-of-the-art machine translation systems (Koehn et al., 2007; Dyer et al., 2010), but with a rule extraction approach tailored to the NLG problem.

2 Overview

Generation of English from AMR graphs is accomplished as follows: the input graph is converted to a tree, which is input into the weighted intersection of a tree-to-string transducer (§4) with a language model. The output English sentence is the (approximately) highest-scoring sentence according to a feature-rich discriminatively trained linear model. After discussing notation (§3), we describe our approach in §4. The transducer’s rules are extracted from the limited AMR corpus and learned general-
izations; they are of four types: basic rules (§5), synthetic rules created using a specialized model (§6), abstract rules (§7), and a small number of handwritten rules (§8).

3 Notation and Definitions

AMR graphs are directed, weakly connected graphs with node labels from the set of concepts $L_N$ and edge labels from the set of relations $L_E$.

AMR graphs are transformed to eliminate cycles (details in §4); we refer to the resulting tree as a transducer input representation (TI representation). For a node $n$ with label $C$ and outgoing edges $n \xrightarrow{L_1} n_1, \ldots, n \xrightarrow{L_m} n_m$, sorted lexicographically by $L_i$ (each an element of $L_E$), the TI representation of the tree rooted at $n$ is denoted:

$$ (X \ C \ (L_1 \ T_1) \ldots (L_m \ T_m)) $$

(1)

where each $T_i$ is the TI representation of the tree rooted at $n_i$. See Fig. 1 for an example. A LISP-like textual formatting of the TI representation in Fig. 1 is:

$$ (X \ want-01 \ (ARG0 \ (X \ boy)) \ (ARG1 \ (X \ ride-01 \ (ARG0 \ (X \ bicycle \ (mod \ (X \ red))))))) $$

To ease notation, we use the function $\text{sort}[]$ to lexicographically sort edge labels in a TI representation. Using this function, an equivalent way of representing the TI representation in Eq. 1, if the $L_i$ are unsorted, is:

$$ (X \ C \ \text{sort}[(L_1 \ T_1) \ldots (L_m \ T_m)]) $$

The TI representation is converted into a word sequence using a tree-to-string transducer. The tree transducer formalism we use is one-state extended linear, non-deleting tree-to-string (1-xRLNs) transducers (Huang et al., 2006; Graehl and Knight, 2004).

**Definition 1.** (From Huang et al., 2006.) A 1-xRLNs transducer is a tuple $(N, \Sigma, W, R)$ where $N$ is the set of nonterminals (relation labels and $X$), $\Sigma$ is the input alphabet (concept labels), $W$ is the output alphabet (words), and $R$ is the set of rules. A rule in $R$ is a tuple $(t, s, \phi)$ where:

1. $t$ is the LHS tree, whose internal nodes are labeled by nonterminal symbols, and whose frontier nodes are labeled terminals from $\Sigma$ or variables from a set $X = \{X_1, X_2, \ldots\}$;
2. $s \in (X \cup W)^*$ is the RHS string;
3. $\phi$ is a mapping from $X$ to nonterminals $N$.

A rule is a purely lexical rule if it has no variables.

As an example, the tree-to-string transducer rules which produce the output sentence from the TI representation in Fig. 1 are:

\begin{align*}
(X \text{ want-01} & \ (\text{ARG0} X_1) \ (\text{ARG1} X_2)) \rightarrow \\
\text{The } X_1 & \text{ wants to } X_2.
\end{align*}

\begin{align*}
(X \text{ ride-01} & \ (\text{ARG1} X_1)) \rightarrow \text{ride the } X_1 \\
(X \text{ bicycle (mod } X_1)) & \rightarrow X_1 \text{ bicycle} \\
(X \text{ red}) & \rightarrow \text{red} \\
(X \text{ boy}) & \rightarrow \text{boy}
\end{align*}

(2)

Here, all $X_i$ are mapped by a trivial $\phi$ to the nonterminal $X$.

The output string of the transducer is the target projection of the derivation, defined as follows:

**Definition 2.** (From Huang et al., 2006.) A derivation $d$, its source and target projections, denoted $S(d)$ and $E(d)$ respectively, are recursively defined as follows:

1. If $r = (t, s, \phi)$ is a purely lexical rule, then $d = r$ is a derivation, where $S(d) = t$ and $E(d) = s$;
2. If $r = (t, s, \phi)$ is a rule, and $d_i$ is a (sub-)derivation with the root symbol of its source projection matching the corresponding substitution node in $r$, i.e., $\text{root}(S(d_i)) = \phi(x_i)$, then $d = r(d_1, \ldots, d_m)$ is also a derivation, where $S(d) = [x_i \mapsto S(d_i)]t$ and $E(d) = [x_i \mapsto E(d_i)]s$.

The notation $[x_i \mapsto y_i]t$ is shorthand for the result of substituting $y_i$ for each $x_i$ in $t$, where $x_i$ ranges over all variables in $t$.

The set of all derivations of a target string $e$ with a transducer $T$ is denoted

$$D(e, T) = \{d \mid E(d) = e\}$$

where $d$ is a derivation in $T$.

We use a shorthand notation for the transducer rules that will be useful when discussing rule extraction and synthetic rules. Let $f_i$ be a TI representation. The TI representation has the form

$$f_i = (X \ C \ (L_1 \ T_1) \ \ldots \ (L_m \ T_m))$$

where $L_i \in L_E$ and $T_1, \ldots, T_m$ are TI representations.\(^4\) Let $A_1, \ldots, A_n \in L_E$. We use

$$(f_i, A_1, \ldots, A_n) \rightarrow r \quad (3)$$

as shorthand for the rule:

\begin{align*}
(X \ C \ \text{sort} & [(L_1 \ T_1) \ \ldots \ (L_m \ T_m)] \\
(A_1 \ X_1) & \ \ldots \ (A_n \ X_n)) \rightarrow r
\end{align*}

(4)

Note $r$ must contain the variables $X_1 \ldots X_n$. In (3) and (4), argument slots with relation labels $A_i$ have been added as children to the root node of the TI representation $f_i$.

For example, the shorthand for the transducer rules in (2) is:

$$(((X \text{ want-01}), \text{ARG0}, \text{ARG1})) \rightarrow \\
\text{The } X_1 \text{ wants to } X_2.$$ 

$$(((X \text{ ride-01}), \text{ARG1}) \rightarrow \text{ride the } X_1$$

$$(((X \text{ bicycle}), \text{mod}) \rightarrow X_1 \text{ bicycle}$$

$$((X \text{ red})) \rightarrow \text{red}$$

(5)

4 Generation

To generate a sentence $e$ from an input AMR graph $G$, a spanning tree $G'$ of $G$ is computed, then transformed into a string using a tree-to-string transducer.

**Spanning tree.** The choice of the graph $G$’s spanning tree $G'$ could have a big effect on the output, since the transducer’s output will always be a projective reordering of the tree’s leaves. Our spanning tree results from a breadth-first-search traversal, visiting child nodes in lexicographic order of the relation label (inverse relations are visited last). The edges traversed are included in the tree. This simple heuristic is a baseline which can potentially be improved in future work.

**Decoding.** Let $T = (N, \Sigma, W, \mathcal{R})$ be a tree-to-string transducer. The output sentence is the highest scoring transduction of $G'$:

$$e = \arg \max_{d \in D(G', T)} score(d; \theta)$$

(6)

\(^4\)If $f_i$ is just a single concept with no children, then $m = 0$ and $f_i = (X \ C)$.
Eq. 6 is solved approximately using the cdec decoder for machine translation (Dyer et al., 2010). The score of the transduction is a linear function (with coefficients $\theta$) of a vector of features including the output sequence’s language model log-probability and features associated with the rules in the derivation (denoted $f$; Table 1):

$$score(d; \theta) = \theta_{LM} \log(p_{LM}(E(d))) + \sum_{r \in d} \theta^T f(r)$$

The feature weights are trained on a development dataset using MERT (Och, 2003).

In the next four sections, we describe the rules extracted and generalized from the training corpus.

5 Inducing Basic Rules

The basic rules, denoted $R_B$, are extracted from the training AMR data using an algorithm similar to extracting tree transducers from tree-string aligned parallel corpora (Galley et al., 2004). In-Formally, the rules are extracted from a sentence $w = \langle w_1, \ldots, w_n \rangle$ with AMR graph $G$ as follows:

1. The AMR graph and the sentence are aligned; we use the JAMR aligner from Flanigan et al. (2014), which aligns non-overlapping subgraphs of the graph to spans of words. The subgraphs that JAMR aligns are called fragments. In JAMR’s aligner, all fragments are trees.

2. $G$ is replaced by its spanning tree by deleting relations that use a variable in the AMR annotation.

3. In the spanning tree, for each node $i$, we keep track of the word indices $b(i)$ and $e(i)$ in the original sentence that trap all of $i$’s descendants. (This is calculated using a simple bottom-up propagation from the leaves to the root.)

4. For each aligned fragment $i$, a rule is extracted by taking the subsequence $\langle w_{b(i)}, \ldots, w_{e(i)} \rangle$ and “punching out” the spans of the child nodes (and their descendants) and replacing them with argument slots.

See Fig. 2 for examples.

More formally, assume the nodes in $G$ are numbered $1, \ldots, N$ and the fragments are numbered $1, \ldots, F$. Let nodes $: \{1, \ldots, F\} \rightarrow 2^\{1,\ldots,N\}$ and root $: \{1, \ldots, F\} \rightarrow \{1, \ldots, N\}$ be functions that return the nodes in a fragment and the root of a fragment, respectively, and let children $: \{1, \ldots, N\} \rightarrow 2^\{1,\ldots,N\}$ return the child nodes of a node. We consider a node aligned if it belongs to an aligned fragment. Let the span of an aligned node $i$ be denoted by endpoints $a_i$ and $a_i'$; for unaligned nodes, $a_i = \infty$ and $a_i' = -\infty$ (depicted with superscripts in Fig. 2). The node alignments are propagated by defining $b(\cdot)$ and $e(\cdot)$ recursively, bottom up:

$$b(i) = \min(a_j, \min_{j \in \text{children}(i)} b(j))$$

$$e(i) = \max(a_j', \max_{j \in \text{children}(i)} e(j))$$

Also define functions $\tilde{b}$ and $\tilde{e}$, from fragment indices to integers, as:

$$\tilde{b}(i) = b(\text{root}(i))$$

$$\tilde{e}(i) = e(\text{root}(i))$$

For fragment $i$, let $C_i = \text{children}(\text{root}(i)) - \text{nodes}(i)$, which is the children of the fragment’s root concept that are not included in the fragment. Let $f_i$ be the TI representation for fragment $i$. If $C_i$ is empty, then the rule extracted for fragment $i$ is:

$$r_i : (f_i) \rightarrow w_{\tilde{b}(i); \tilde{e}(i)}(7)$$

Otherwise, let $m = |C_i|$, and denote the edge labels from root($i$) to elements of $C_i$ as $A_1(i) \ldots A_m(i)$. For $j \in \{1, \ldots, m\}$, let $k_j$ select the elements $c_{k_j}$ of $C_i$ in ascending order of $b(k_j)$. Then the rule extracted for fragment $i$ is:

$$r_i : (f_i, A_{k_1}(i), \ldots, A_{k_m}(i)) \rightarrow$$

$$w_{\tilde{b}(i); \tilde{e}(i)} X_1 w_{\tilde{e}(k_1); \tilde{b}(k_2)} X_2 \ldots$$

$$\ldots w_{\tilde{e}(k_{m-1}); \tilde{b}(k_m)} X_m w_{\tilde{e}(k_m); \tilde{e}(i)}(8)$$

A rule is only extracted if the fragment $i$ is aligned and the child spans do not overlap. Fig. 2 gives an example of a tree annotated with alignments, $b$ and $e$, and the extracted rules.

\(^5\)I.e., the nodes in fragment $i$, with the edges between them, represented as a TI representation.
Table 1: Rule features in the transducer. There is also an indicator feature for every handwritten rule.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule</td>
<td>1 for every rule</td>
</tr>
<tr>
<td>Basic</td>
<td>1 for basic rules, else 0</td>
</tr>
<tr>
<td>Synthetic</td>
<td>1 for synthetic rules, else 0</td>
</tr>
<tr>
<td>Abstract</td>
<td>1 for abstract rules, else 0</td>
</tr>
<tr>
<td>Handwritten</td>
<td>1 for handwritten rules, else 0</td>
</tr>
<tr>
<td>Rule given concept</td>
<td>( \log(\text{number of times rule extracted} / \text{number of times concept observed in training data}) ) (only for basic rules, 0 otherwise)</td>
</tr>
<tr>
<td>... without sense</td>
<td>same as above, but with sense tags for concepts removed</td>
</tr>
<tr>
<td>Synthetic score</td>
<td>model score for the synthetic rule (only for synthetic rules, 0 otherwise)</td>
</tr>
<tr>
<td>Word count</td>
<td>number of words in the rule</td>
</tr>
<tr>
<td>Stop word count</td>
<td>number of words not in a stop word list</td>
</tr>
<tr>
<td>Bad stop word</td>
<td>number of words in a list of meaning-changing stop words, such as “all, can, could, only, so, too, until, very”</td>
</tr>
<tr>
<td>Negation word</td>
<td>number of words in “no, not, n’t”</td>
</tr>
</tbody>
</table>

6 Modeling Synthetic Rules

The synthetic rules, denoted \( \mathcal{R}_S(G) \), are created to generalize the basic rules and overcome data sparseness resulting from our relatively small training dataset. Our synthetic rule model considers an AMR graph \( G \) and generates a set of rules for each node in \( G \). Synthetic rule’s LHS is a TI representation \( f \) with argument slots \( A_1 \ldots A_m \) (this is the same form as the LHS for basic rules). For each node in \( G \), one or more LHS are created (we will discuss this further below), and for each LHS, a set of \( k \)-best synthetic rules are produced. The simplest case of a LHS is just a concept and argument slots corresponding to each of its children.

For a given LHS, the synthetic rule model creates a RHS by concatenating together a string in \( W^* \) (called a concept realization and corresponding to the concept fragment) with strings in \( W^* \mathcal{X} W^* \) (called an argument realization and corresponding to the argument slots). See the top of Fig. 3 for a synthetic rule with concept and argument realizations highlighted.

Synthetic rules have the form:

\[
\begin{align*}
   & r : (f, A_1, \ldots, A_m) \rightarrow \quad (9) \\
   & l_{k_1} X_{k_1} r_{k_1} \ldots l_{k_c} X_{k_c} r_{k_c} c \\
   & l_{k_{c+1}} X_{k_{c+1}} r_{k_{c+1}} \ldots l_{k_m} X_{k_m} r_{k_m}
\end{align*}
\]

where:

- \( f \) is a TI representation.
- Each \( A_i \in L_E \).
- \( (k_1, \ldots, k_m) \) is a permutation of \( (1, \ldots, m) \).
- \( c \in W^* \) is the realization of TI representation \( f \).
- Each \( l_i, r_i \in W^* \) and \( X_i \in \mathcal{X} \). Let \( R_i = \langle l_i, r_i \rangle \) denote the realization of argument \( i \).
- \( c \in [0, m] \) is the position of \( c \) among the realizations of the arguments.

Let \( \mathcal{F} \) be the space of all possible TI representations. Synthetic rules make use of three lookup tables (which are partial functions) to provide candidate realizations for concepts and arguments: a table for concept realizations \( \text{lex} : \mathcal{F} \rightarrow 2^{W^*} \), a table for argument realizations when the argument is on the left \( \text{left}_{\text{lex}} : \mathcal{F} \times L_E \rightarrow 2^{W^*} \), and a table for argument realizations when the argument is on the right \( \text{right}_{\text{lex}} : \mathcal{F} \times L_E \rightarrow 2^{W^*} \). These tables are constructed during basic rule extraction, the details of which are discussed below.

Synthetic rules are selected using a linear model with features \( g \) and coefficients \( \phi \), which scores each RHS for a given LHS. For LHS = \( (f, A_1, \ldots, A_m) \), the RHS is specified completely by \( c, c, R_1, \ldots, R_m \) and a permutation \( k_1, \ldots, k_m \). For each node in \( G \), and for each TI representation \( f \) in the domain of \( \text{lex} \) that matches the node, a LHS is created, and a set of \( K \) synthetic rules is produced for each \( c \in \text{lex}(f) \). The rules produced are the
The boy wants to ride the bicycle.

(a) Sentence annotated with indexes, and bracketed according to $b(i)$ and $e(i)$ from the graph in (b).

(b) Tree annotated with $a_i$, $a'_i$ (superscripts) and $b(i)$, $e(i)$ (subscripts).

(c) Extracted rules.

Figure 2: Example rule extraction from an AMR-annotated sentence.

Figure 3: Synthetic rule generation for the rule shown at top. In the rule RHS, the realization for ARG0 is blue, the realization for DEST is red, and the realization for ride-01 is black. For a fixed permutation of the concept and arguments, choosing the argument realizations can be seen as a sequence labeling problem (bottom). The highlighted sequence corresponds to the rule at top.

\[ \text{arg max}_{c, k_1, \ldots, k_m} \left( \sum_{i=1}^{c} \psi^\top g(R_{k_i}, A_{k_i}, c, i, c) + \psi^\top g((\epsilon, \epsilon), *, c, c+1, c) + \sum_{i=c+1}^{m} \psi^\top g(R_{k_i}, A_{k_i}, c, i+1, c) \right) \]  

(10)

where the max is over $c \in 0 \ldots m$, $k_1, \ldots, k_m$ is any permutation of $1, \ldots, m$, and $R_i \in \text{left}_\text{lex}(A_i)$ for $i < c$ and $R_i \in \text{right}_\text{lex}(A_i)$ for $i > c$. * is used to denote the concept position. $\epsilon$ is the empty string.

The best solution to Eq. 10 is found exactly by brute force search over concept position $c \in [0, m + 1]$ and the permutation $k_1, \ldots, k_m$. With fixed concept position and permutation, each $R_i$ for the arg max is found independently. To obtain the exact $K$-best solutions, we use dynamic programming with a $K$-best semiring (Goodman, 1999) to keep track of the $K$ best sequences for each concept position and permutation, and take the best $K$ sequences over all values of $c$ and $k$.

The synthetic rule model’s parameters are estimated using basic rules extracted from the training data. Basic rules are put into the form of Eq. 9 by
Table 2: Synthetic rule model features. POS is the most common part-of-speech tag sequence for c, “dist” is the string “dist”, and side is “L” if \( i < c \), “R” otherwise. + denotes string concatenation.

<table>
<thead>
<tr>
<th>Feature name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>POS + ( A_i ) + “dist”</td>
<td>(</td>
</tr>
<tr>
<td>POS + ( A_i ) + side</td>
<td>1.0</td>
</tr>
<tr>
<td>POS + ( A_i ) + side + “dist”</td>
<td>(</td>
</tr>
<tr>
<td>POS + ( A_i ) + ( R_i ) + side</td>
<td>1.0</td>
</tr>
<tr>
<td>c + ( A_i ) + “dist”</td>
<td>(</td>
</tr>
<tr>
<td>c + ( A_i ) + side</td>
<td>1.0</td>
</tr>
<tr>
<td>c + ( A_i ) + side + “dist”</td>
<td>(</td>
</tr>
<tr>
<td>c + POS + ( A_i ) + side + “dist”</td>
<td>(</td>
</tr>
</tbody>
</table>

Segmenting the RHS into the form

\[ I_1 X_1 r_1 \ldots c \ldots I_m X_m r_m \] (11)

by choosing \( c, l_i, r_i \in W^* \) for \( i \in \{1, \ldots, m\} \). An example segmentation is the rule RHS in Fig. 3.

Segmenting the RHS of the basic rules into the form of Eq. 11 is done as follows: \( c \) is the aligned span for \( f \). For the argument realizations, arguments to the left of \( c \) pick up words to their right, and arguments to the right pick up words to their left. Specifically, for \( i < c \) (\( R_i \) to the left of \( c \) but not next to \( c \)), \( I_i \) is empty and \( r_i \) contains all words between \( a_i \) and \( a_{i+1} \). For \( i = c \) (\( R_i \) directly to the left of \( c \)), \( I_i \) is empty and \( r_i \) contains all words between \( a_c \) and \( c \). For \( i > c + 1 \), \( I_i \) contains all words between \( a_{i-1} \) and \( a_i \), and for \( i = c + 1 \), \( I_i \) contains all words between \( c \) and \( a_i \).

The tables for \( \text{lex} \), \( \text{left}_{\text{lex}} \), and \( \text{right}_{\text{lex}} \) are populated using the segmented basic rules. For each basic rule extracted from the training corpus and segmented according to the previous paragraph, \( f \rightarrow c \) is added to \( \text{lex} \), and \( A_{k_i} \rightarrow \langle l_i, r_i \rangle \) is added to \( \text{left}_{\text{lex}} \) for \( i \leq c \) and \( \text{right}_{\text{lex}} \) for \( i > c \). The permutation \( k_i \) is known during extraction in Eq. 8.

The parameters \( \psi \) are trained using AdaGrad (Duchi et al., 2011) with the perceptron loss function (Rosenblatt, 1957; Collins, 2002) for 10 iterations over the basic rules. The features \( g \) are listed in Table 2.

7 Abstract Rules

Like the synthetic rules, the abstract rules \( R_A(G) \) generalize the basic rules. However, abstract rules are much simpler generalizations which use part-of-speech (POS) tags to generalize. Abstract rules make use of a POS abstract rule table, which is a table listing every combination of the POS of the concept realization, the child arguments’ labels, and rule RHS with the concept realization removed and replaced with *. This table is populated from the basic rules extracted from the training corpus. An example entry in the table is:

\((\text{VBD}, \text{ARG0}, \text{DEST}) \rightarrow X_1 (\ast) \) to the \( X_2 \)

For the LHS \((f, A_1, \ldots, A_m)\), an abstract rule is created for each member of \( c \in \text{lex}(f) \) and the most common POS tag \( p \) for \( c \) by looking up \( p, A_1, \ldots, A_m \) in the POS abstract rule table, finding the common RHS, and filling in the concept position with \( c \). The set of all such rules is returned.

8 Handwritten Rules

We have handwritten rules for dates, conjunctions, multiple sentences, and the concept have-org-role-91. We also create pass-through rules for concepts by removing sense tags and quotes (for string literals).

9 Experiments

We evaluate on the AMR Annotation Release version 1.0 (LDC2014T12) dataset. We follow the recommended train/dev/test splits, except that we remove MT09 data (204 sentences) from the training data and use it as another test set. Statistics for this dataset and splits are given in Table 3. We use a 5-gram language model trained with KenLM (Heafield et al., 2013) on Gigaword (LDC2011T07), and use 100-best synthetic rules.

We evaluate with the Bleu scoring metric (Papineni et al., 2002) (Table 4). We report single ref-
Table 4: Uncased Bleu scores with various types of rules removed from the full system.

<table>
<thead>
<tr>
<th>Rules</th>
<th>Test</th>
<th>MT09</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>22.1</td>
<td>21.2</td>
</tr>
<tr>
<td>Full − basic</td>
<td>22.1</td>
<td>20.9</td>
</tr>
<tr>
<td>Full − synthetic</td>
<td>9.1</td>
<td>7.8</td>
</tr>
<tr>
<td>Full − abstract</td>
<td>22.0</td>
<td>21.2</td>
</tr>
<tr>
<td>Full − handwritten</td>
<td>21.9</td>
<td>20.5</td>
</tr>
</tbody>
</table>

dherence Bleu for the LCD2014T12 test set, and four-reference Bleu for the MT09 set. We report ablation experiments for different sources of rules. When ablating handwritten rules, we do not ablate pass-through rules.

The full system achieves 22.1 Bleu on the test set, and 21.2 on MT09. Removing the synthetic rules drops the results to 9.1 Bleu on test and 7.8 on MT09. Removing the basic and abstract rules has little impact on the results. This may be because the synthetic rule model already contains much of the information in the basic and abstract rules. Removing the handwritten rules has a slightly larger effect, demonstrating the value of handwritten rules in this statistical system.

10 Related Work

There is a large body of work for statistical and non-statistical NLG from a variety of input representations. Statistical NLG systems have been built for input representations such as HPSG (Nakanishi et al., 2005), LFG (Cahill and Van Genabith, 2006; Hogan et al., 2007), and CCG (White et al., 2007), as well as surface and deep syntax (Belz et al., 2011). The deep syntax representations in Bohnet et al. (2010) and Belz et al. (2011) share similarities with AMR: the representations are graphs with re-entrancies, and have an concept inventory from PropBank (Palmer et al., 2005).

The Nitrogen and Halogen systems (Langkilde and Knight, 1998; Langkilde, 2000) used an input representation that was a precursor to the modern version of AMR, which was also called AMR, although it was not the same representation as Banarescu et al. (2013).

Techniques from statistical machine translation have been applied to the problem of NLG (Wong and Mooney, 2006), and many grammar-based approaches can be formulated as weighted tree-to-string transducers. Jones et al. (2012) developed technology for generation and translation with synchronous hyperedge replacement (SHRG) grammars applied to the GeoQuery corpus (Wong and Mooney, 2006), which in principle could be applied to AMR generation.

11 Conclusion

We have presented a two-stage method for natural language generation from AMR, setting a baseline for future work. We have also demonstrated the importance of modeling argument realization for good performance. Our feature-based, tree-transducer approach can be easily extended with rules and features from other sources, allowing future improvements.

Acknowledgments

The authors would like to thank Adam Lopez and Nathan Schneider for valuable feedback, and Sam Thomson and the attendees of the Fred Jelinek Memorial Workshop in 2014 in Prague for helpful discussions. This work is supported by the U.S. Army Research Office under grant number W911NF-10-1-0533. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the view of the U.S. Army Research Office or the United States Government.

References


Liang Huang, Kevin Knight, and Aravind Joshi. 2006. Statistical syntax-directed translation with extended domain of locality. In Proc. of AMTA.


