Text Generation from Abstract Meaning Representation

Lisa Jin

Area Committee:
Prof. Daniel Gildea, Chair
Prof. Lenhart Schubert
Prof. Chenliang Xu

Abstract. Given the minimal nature of semantic structures, generating text from them is a fairly unconstrained task. The crux lies in the need for linguistically fluent output that is drawn from a sparse, high-dimensional vocabulary. Learning to flexibly map from semantic structures to these output strings can involve ambiguity. This paper surveys a continuum of statistical and neural approaches for such natural language generation (NLG). It further introduces a neural decoder for NLG from AMR that is constrained by a transition-based parser. The model jointly predicts transition system actions and English tokens to more directly learn word-order traversals of AMR graphs.

1 Introduction

Tasks transforming between semantic representations and natural language usually include text generation as a subroutine. This applies to subareas such as machine translation, summarization, and question-answering. With respect to MT, for instance, it may be possible to derive a common semantic structure between source and target strings. Translation would thus involve predicting a target string from the hypothesized semantic representation.

Text generation often involves complex linguistic inference. A successful NLG system produces fluent strings that still capture the meaning of a given representation. These meaning representations are often structurally dense (e.g., tree- or graph-like) and consist of only the necessary semantic details. In addition, there may be a myriad of choices in terms of co-reference, named entities, verb-argument dependencies, etc. The spare input coupled with a vast search space burdens the generator with ambiguity; it must make informed syntactic and lexical decisions based on abstract semantics. Below is an Abstract Meaning Representation (AMR) (Banarescu et al., 2013) annotation from *The Little Prince* corpus[1] of the sentence, "Try to be happy".

```
(t / try-01 :mode imperative
    :ARG0 (y / you)
    :ARG1 (h / happy-01
              :ARG1 y))
```

[1] https://amr.isi.edu/download/amr-bank-struct-v1.6.txt

AMRs are rooted, directed graphs that model propositional meaning. In the above example, the AMR is written using nested subgraphs (enclosed in parentheses) with colon-prefixed edge labels, single-letter instance variables, and concept labels. The concepts, or vertices, are denoted by variable names and/or full concept labels. The relations, or edges, are each initialized via an edge label and pair of concept endpoints. Below is an AMR as a logical conjunction of triples.

$\exists$ t, y, h:
instance(t, try-01) $\wedge$ instance(y, you) $\wedge$ instance(h, happy-01) $\wedge$
mode(t, imperative) $\wedge$ arg0(t, y) $\wedge$ arg1(t, h) $\wedge$ arg1(h, y)

Note that the concept `(y / you)` is reused as the child of two distinct parents—a case of graph re-entrancy that distinguishes graphs from trees. The concept acts as the pronoun "your" for the implicit "happiness" and as the subject "you" of the full sentence. This is a case where the input AMR encodes semantics that lack lexical realization in the sentence. An NLG system must clearly be robust to both missing and extra input information relative to its goal. A more complete treatment of AMR will follow in Section 3.2.

1.1 Generation in Terms of Parsing

Since AMR and other semantic structures represent logical meaning, extracting English text in a principled way is nontrivial. However, it is possible to identify links—or alignments—between input concepts and output words as a starting point. Though AMR lacks ground-truth alignments between concepts and words, there exist probabilistic models for approximating them (Flanigan et al., 2014). The quality of such alignments has proved to be crucial in the context of AMR parsing (Lyu and Titov, 2018). Therefore, it is easy to see that such concept-word mappings may be useful in the reverse problem of AMR-to-text generation.

To build a graph from word-aligned concepts, transition-based parsing is attractive due to the search space constraints imposed by transition rules. Namely, these parsers are *incremental* in that inputs are processed individually from left-to-right. Traditional shift-reduce parsers begin with an ordered buffer of vertices, shifting them one-by-one onto the stack upon execution. After each shift operation, the system considers building an edge between the current buffer vertex and each stack element. This class of parsers is *deterministic*, as parse actions are predicted based on the 'current' parser configuration. With running time $\mathcal{O}(n^2)$, a shift-reduce parser can construct graphs of arbitrary complexity. In this manner, the input string maps to parser actions that can be applied sequentially to build its graph.

If AMR parsing extracts concepts from words and links them into a graph, then generation can be seen as the inverse of these two operations. These two stages correspond to concept and relation identification. For the parsing problem, even if the set of concepts is fixed, there are an exponential number of possible graphs from this set. The analogous problem in generation may be 'linearizing' an AMR into its string by visiting its concepts in an order aligned to the output string. With this perspective, the bottleneck in generation would be finding graph traversals that correspond to word order. Since parsers map between input word order and topology of the graph, they provide valuable insight for learning such traversals.

1.2 Cache Transition System

A specific type of parser called the *cache transition system* (Gildea et al., 2018) uses a fixed-size cache to leverage highly interconnected subgraphs during parsing. Compared to the shift-reduce parser that builds edges exhaustively, the cache transition system restricts its attention to a working set of vertices. This economy in memory allows for improved efficiency, enabling the parser to run in time $\mathcal{O}(n)$.

Tree Decomposition. The cache transition system operates on the theoretical notion of *treewidth* for a graph's tree decomposition. Intuitively, a tree decomposition defines an overlapping partition of vertices into minimally connected sets. For a graph $G = (V, E)$, vertex sets X_i called *bags* and a set of *arcs* F between them make up a tree decomposition $TD(G) = (\{X_i \mid i \in I\},\ T = (I, F))$ with the following properties.

- *Vertex cover*: Every graph vertex $v \in V$ exists in some bag X_i.
- *Edge cover*: Endpoints of every edge $(u, v) \in E$ exist in a particular bag.
- *Running intersection*: Bags containing a specific vertex are connected in a subtree of T; if j uniquely connects i and k in T, $X_i \cap X_k \subseteq X_j\ \forall i, j, k \in I$.

A tree decomposition's *width* is determined by its largest bag, or $\max_i |X_i| - 1$, and a graph's treewidth is the lowest such width among all its tree decompositions. At the extremes, trees have treewidth 1 (due to acyclicity) while fully connected graphs have treewidth $n - 1$ for n total vertices. Fig. 1 has $n = 3$ vertices and is a fully connected graph with treewidth $n - 1 = 2$.

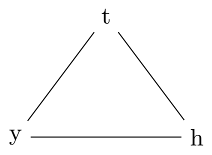

Fig. 1. Simplified semantic graph for the sentence, "You try to be happy". The vertex-word mappings are {y: *you*, t: *try*, h: *happy*}.

Relative to a traversal order of vertices, a cache transition system is only able to parse graphs of maximal width $m - 1$ for cache size m. This constraint limits the class of graphs that can be successfully parsed, offering a deeper mapping between parser actions and a graph's underlying structure.

Parser Components and Execution. Formally, a cache transition parser consists of a stack, cache, buffer, and edge set; its configuration can be written as $c = (\sigma, \eta, \beta, E)$. As in a shift-reduce parser, the buffer β is initialized with a sequence of vertices. The cache η is a random-access data structure of vertices that are active candidates for edge endpoints. At any point in time, edges may only be formed between the rightmost cache vertex and all other cache elements.

After a $\mathsf{push}(i, C)$ transition, cache element $\eta[i]$ is displaced by $\beta[1]$ and shifted to the stack σ. Element $\eta[m]$ is linked to elements at cache indices $C \subseteq [1 \ldots m]$. A pop transition returns the topmost stack element to its original position i, shifting all right-hand cache elements right by one. A transition modifies exactly one element in the cache—each new cache configuration of size m corresponds to a bag of equal size in a tree decomposition. The unique cache states in Fig. 2 are a preorder depth-first traversal of a derivation tree (Fig. 3).

stack	cache	buffer	edges	resulting from action
[ ]	[\$, \$, \$]	[y, t, o, b, h]	$\emptyset$	—
[1, \$]	[\$, \$, y]	[t, o, b, h]	$\emptyset$	$\mathsf{push}(1, \emptyset)$
[1, \$, 1, \$]	[\$, y, t]	[o, b, h]	E_1	$\mathsf{push}(1, \{2\})$
[1, \$, 1, \$, 1, \$]	[y, t, o]	[b, h]	E_1	$\mathsf{push}(1, \emptyset)$
[1, \$, 1, \$]	[\$, y, t]	[b, h]	E_1	pop
[1, \$, 1, \$, 1, \$]	[y, t, b]	[h]	E_1	$\mathsf{push}(1, \emptyset)$
[1, \$, 1, \$]	[\$, y, t]	[h]	E_1	pop
[1, \$, 1, \$, 1, \$]	[y, t, h]	[ ]	E_2	$\mathsf{push}(1, \{1, 2\})$
[1, \$, 1, \$]	[\$, y, t]	[ ]	E_2	pop
[1, \$,]	[\$, \$, y]	[ ]	E_2	pop
[ ]	[\$, \$, \$]	[ ]	E_2	pop

Fig. 2. A cache transition system run building the Fig. 1 graph. Sets $E_1 = \{(\text{y, t})\}$ and $E_2 = E_1 \cup \{(\text{y, h}), (\text{t, h})\}$. Set {o: *to*, b: *be*} does not appear in the final graph.

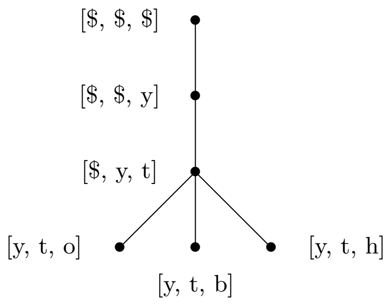

Fig. 3. Derivation tree representing the run in Fig. 2, adapted from Gildea et al. (2018).

2 Related Work

2.1 Early Approaches

Early work in text generation slowly began to favor statistical approaches over template- or rule-based ones. The former can be labeled *stochastic* and the latter *hand-crafted*. The trade-off between lexical expressiveness and grammatical validity also lead to hybrids of these approaches.

Word Lattices and Statistical LMs. One initial approach to NLG by Knight and Hatzivassiloglou (1995) was inspired by a Japanese-English machine translation system. They frame the problem as generating text that is robust to (i) imperfect semantic inputs and (ii) incomplete knowledge bases. For the former, they encode each input into a *word lattice* (e.g., Fig. 4): a directed, acyclic network of states with word-labeled transitions. This handles cases of underdefined semantic input, as linguistic uncertainty can be expressed as multiple word paths between the start and end states.

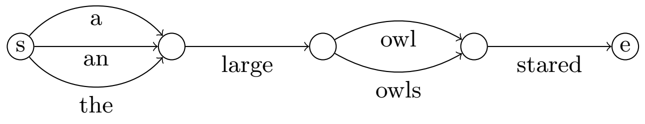

Fig. 4. Word lattice with six possible paths. Each vertex is a state, with ‘s’ marking the start and ‘e’ the end state.

A PENMAN generator then uses *lexical islands*, or non-interacting word sets, to decompose the search space. Since each lexical island corresponds to a state in the word lattice, finding the best string reduces to an efficient bottom-up search over lattice paths. To address (ii), Knight and Hatzivassiloglou augment the generator, which may be prone to disfluency, with a statistical language model that discriminates between lexical choices. Combined together, the generator runs a beam search on the word lattice and the n-gram language model ranks the top-K resulting sentences.

The Nitrogen system (Langkilde and Knight, 1998) continued to develop the use of word lattices and statistical models for NLG. This work sustains focus on handling insufficient input and KBs, but also specifies a new meaning representation[2] composed of concepts and relations. It contributes a grammar formalism and algorithm for mapping between MRs and word lattices. More concretely, the grammar formalism recursively builds a word lattice from an MR using keyword matching rules. Using an *instance rule* that acts on part of speech (e.g., noun or verb), single concepts of the MR are converted to word lattice components. If the

[2] Referred to as ‘MR’ to disambiguate with ‘AMR’.

instance rule fails to apply, a *recasting mechanism* transforms between semantically equivalent structures to add the missing keywords. This mechanism can rewrite MR components according to a range of linguistic constructs and semantic depths. Compared to the rigid template-based or purely syntactic patterns of previous NLG systems, Nitrogen employed rules that were more grounded in semantic structure.

Nitrogen's generative power and ability to encode multiple hypotheses transferred well to the problem of preserving ambiguity, a common concern in translation (Knight and Langkilde, 2000). Below is an example of such an ambiguous sentence, along with two possible interpretations.

Joan saw the man with the binoculars.
a. With the binoculars, Joan saw the man.
b. The man with binoculars was seen by Joan.

The goal from an MT standpoint is to produce a translation that retains the ambiguity of the source sentence. Due to syntactic differences, it may be nontrivial to replicate the effect of source language word order in the target string. The authors devise an algorithm to combine a pair of parse forests[3] (i.e., packed syntactic trees) such that one forest is contained in the other. Ambiguous meanings are expressed by both forests and thus present in their intersection. The routine for computing this intersection is described below.

1. Expand forests {F1, F2} into lattices {L1, L2}.
2. Rewrite both forests using a CFG.
3. Find F3 = F1 $\cap$ L2 and F4 = F2 $\cap$ L1 by using the respective forest's CFG to parse the lattice.
4. Return F5 = F3 $\cup$ F4 by merging the forests' roots.

Step 3 involves computing the intersection of an FSA (e.g., a lattice) and CFG, which can be done in polynomial-time. In this way, the internal representations of Nitrogen permit an efficient, elegant solution to ambiguity preservation.

LTAGs. On the heels of Nitrogen, Bangalore and Rambow introduced the lexicalized tree-adjoining grammar (LTAG), which they implemented in the FERGUS generation system (2000). Though Nitrogen and FERGUS both extract word lattices, the inputs to FERGUS are based on syntactic trees rather than semantic MRs. The LTAG formalism offers a powerful mapping between inputs and strings, as opposed to direct input linearization. As Langkilde and Knight acknowledged, this allows for better detection of long-range dependencies.

Theoretically, tree-adjoining grammars (TAGs) (Joshi, 1987) are akin to CFGs but rewrite trees rather than strings. TAGs build trees using two operations: *substitution*, or replacement of a leaf by a tree, and *adjunction*, or insertion of a tree at an internal node. Elementary trees that can only be substituted or adjoined are called *initial* or *auxiliary*, respectively. Each elementary

[3] The authors note that it is easier to capture long-range dependencies over trees rather than strings.

tree is associated with an anchor—or lexical item for LTAGs—that dictates how it may connect to nonterminals in an enclosing tree.

Using an LTAG, it is possible to derive a sentence from elementary trees. This derivation can also be written in derivation tree format. The input to FERGUS is a dependency tree similar to an LTAG derivation tree, but with only word-labeled nodes. With a particular TAG called an XTAG, the system instead assigns TAG trees to these nodes in a process called *supertagging.* The XTAG is then used to linearize the partial derivation trees in the form of word lattices. Finally, the highest scoring state path through the lattice is found using the Viterbi algorithm.

The underlying tree-based representation of FERGUS supported a more generalized, syntactic grammar rather than one focused on the generation task. Bangalore and Rambow claimed that this benefit, in addition to its direct modeling of morphological features, made it superior to Nitrogen.

2.2 Recent Approaches: Statistical

In recent years, there has been renewed interest in formalisms such as tree transducers and synchronous grammars. These systems offer powerful primitives for mapping between semantic structures and strings, while still supporting a probabilistic training framework. In fact, they often originate in MT due to the usefulness of hierarchical tree structures in encoding symbol-reordering patterns.

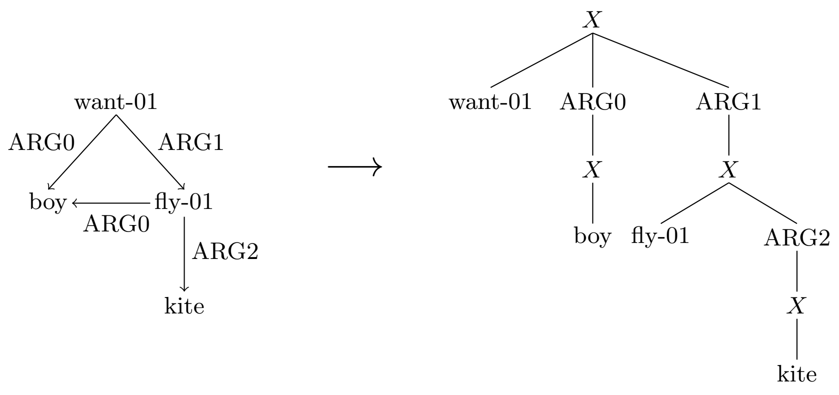

Fig. 5. AMR and transducer input tree for "The boy wants to fly the kite". The reentrant edge from 'fly-01' to 'boy' is deleted in the latter to enforce acyclicity.

Since AMR is designed primarily with human annotators in mind, converting them to trees is a challenge. A single AMR can have multiple valid realizations in terms of tense, definiteness, number, etc. Whether it is possible to compensate for underspecified graphs in downstream tree-to-string systems is unclear.

Tree Transducers. The tree transducer is a generalization of the finite-state transducer (FST) that maps source to target trees using a set of states and rules. It is easy to see that any FST-mapped pair of strings can be transposed into a pair of trees, each containing exactly one node per level. Tree transducers are thus more expressive than their string counterparts as their rules may include multi-level subtrees.

For AMR input, Flanigan et al. (2016) use tree-to-string transducers to generate English (2016). They first extract a transducer input (TI) representation or spanning tree (e.g., Fig. 5) using a BFS traversal of the AMR, where siblings are visited in lexicographic order of their relation labels. Next, the decoded string for this tree is found via a weighted intersection between a language model and tree-to-string transducer.

Flanigan et al. extend a particular type of tree-to-string transducer denoted **1-XRLNs** (Huang et al., 2006). This transducer is defined as a tuple $(N, \Sigma, W, \mathcal{R})$, where N is the set of nonterminals, Σ and W are input and output alphabets, respectively, and $\mathcal{R}$ is the set of rules. Each rule in $\mathcal{R}$ includes the LHS tree t with internal nodes in N and frontier nodes in $(\mathcal{X} \cup \Sigma)^*$, the RHS string $s \in (\mathcal{X} \cup W)^*$, and a mapping $\phi : \mathcal{X} \to N$. A derivation d, along with its source and target projections $\mathcal{S}(d)$ and $\mathcal{E}(d)$ can be defined recursively. If rule r is devoid of nonterminals, then $d = r$ and the source and target projections are precisely s and t, respectively. Otherwise, the symbols in source projection $\mathcal{S}(d)$ can be substituted by variables in t—likewise for $\mathcal{E}(d)$ and s—according to rule r. As an example, the transducer rules for the sentence in Fig. 5 are as follows.

$(X$ want-01 (ARG0 X_1) (ARG1 X_2)) $\to$ The X_1 wants to X_2.
$(X$ fly-01 (ARG1 X_1)) $\to$ fly the X_1
$(X$ kite) $\to$ kite
$(X$ boy) $\to$ boy

Rules extracted for the transducer include basic, synthetic, and abstract rules that generalize the training data as well as a small set of handwritten rules. Synthetic rules are found using a discriminative model and serve to offset data sparsity, which cannot be handled by basic rules alone. Abstract rules further generalize basic rules by using part-of-speech (POS) tags. Finally, handwritten rules help process dates, conjunctions, etc. and normalize concept labels. The routines for obtaining the first three types are briefly described below.

Basic rules Alignments between subgraph 'fragments' and words are found using the JAMR aligner (Flanigan et al., 2014). Each fragment is associated with word indices $b(i)$ and $e(i)$ that fully cover the spans of all its children. In the subsequence $\langle w_{b(i)} \dots w_{e(i)} \rangle$, rules are induced by replacing child spans with argument slots. Rules follow the general form of LHS fragments and argument slots yielding RHS word spans punctuated by nonterminals.

Synthetic rules The LHS of the rules follows that of the basic ones, or LHS = $(f, A_1, \dots, A_m)$ for TI representation f and arguments $A_1, \dots, A_m$ in the relation set. For the RHS, the rule model concatenates a *concept realization*

$\mathbf{c} \in W^*$ with *argument realizations* $R_1, \ldots, R_m$ for $R_i = \langle l_i, r_i \rangle$ in $W^* \mathcal{X} W^*$. The RHS is also described by concept position $c \in [0, m]$ and permutations $k_1, \ldots, k_m \in [1, m]$ of $R_1, \ldots, R_m$. A linear model scores RHS instances for a fixed LHS. Using dynamic programming, finding the K best RHS solutions amounts to a brute force search over c and $k_1, \ldots, k_m$.

Abstract rules A POS abstract rule table supports lookup of RHS sequences given concept realizations' POS and argument labels. The POS is effectively used as a key into this table, and all matching RHS sequences are substituted with concept realization $\mathbf{c}$.

In terms of the BLEU (Papineni et al., 2002) score for strings generated from test data, removing synthetic rules results in an over 40% decrease in points. This result suggests the benefit of using an argument realization model for extracting such rules. In particular, the basic rules depend heavily on alignments in the parallel corpora and thus may be vulnerable to data sparsity constraints.

PBMT. In phrase-based machine translation (PBMT) (Koehn et al., 2003) the units of translation are phrases that can be reordered based on the target language. This method frames the problem in a Bayesian fashion; for foreign sentence $\mathbf{f}$ and English sentence $\mathbf{e}$, $p(\mathbf{e}|\mathbf{f}) \propto p(\mathbf{f}|\mathbf{e})p(\mathbf{e})$. The sentence-level 'likelihood' $p(\mathbf{f}|\mathbf{e})$ can be factorized into the joint probability of its component phrases—subject to a separate distribution that models phrasal reordering:

$$p(\mathbf{f}|\mathbf{e}) = \prod_{i=1}^{I} \phi(\mathbf{f}_i, \mathbf{e}_i) d(\cdot),$$

where $\mathbf{f}$ is segmented into I phrases such that $\mathbf{e}_i$ is the translation of $\mathbf{f}_i$ for all $i \in I$; $\phi(\cdot)$ and $d(\cdot)$ are phrase translation and distortion probability distributions, respectively. It is possible to estimate $\phi(\cdot)$ using relative frequencies in the training data. Distribution $d(\cdot)$ can be parameterized by the $(i-1)^{th}$ end and i^{th} start indices of phrases in the foreign sentence. The 'prior' $p(\mathbf{e})$ is captured in an English language model. The objective is simply to find the highest probability translation, which can be efficiently done using beam search.

Pourdamghani et al. apply PBMT to AMR-to-text generation by first linearizing the AMR into strings that roughly follow English word order. Despite the lack of intermediate tree-like representations, they achieve a sizeable boost (4.9 BLEU points) over the previous baseline of Flanigan et al. (2016).

For linearization, they experiment with a *pre-order DFS* traversal over AMR vertices, *majority method* that memorizes orderings within shared role sets, and a *classifier method* of three binary classifiers that decide edge ordering operations. To evaluate the methods, Pourdamghani et al. compute approximate concept-word alignments and count the number of crossings between the hypothesized AMR traversals and true word order. The classifier method achieves the lowest number of edge crossings and best overall BLEU score. Empirically, they find that linearization performance and BLEU score are positively correlated.

SNRGs. Prior approaches discussed so far have skirted the issue of faulty graph-to-tree transformations. Errors caused by this stage—even before tree-to-string mapping—can be propagated to the final generated string. Song et al. (2017) apply a method that directly extracts and applies graph-fragment-to-string rules, bypassing such harmful errors in the generation pipeline.

In general, a graph transducer is akin to its tree counterpart in that it bridges input structures to output strings. A node replacement grammar (NRG) (Engelfriet and Rozenberg, 1997) includes rules to replace nonterminal nodes with subgraphs, often in a context-free manner. The replacement involves a 'mother' graph M, subgraph $S \subseteq M$, and 'daughter' graph D. Given a grammar rule, all occurrences of LHS subgraph S in M are removed to create M^-. An embedding mechanism E then governs how to attach copies of RHS daughter D to M^-, effectively replacing all S with D in M. The localized, precise way in which these substitutions occur allow the NRG to capture recursive construction of graphs.

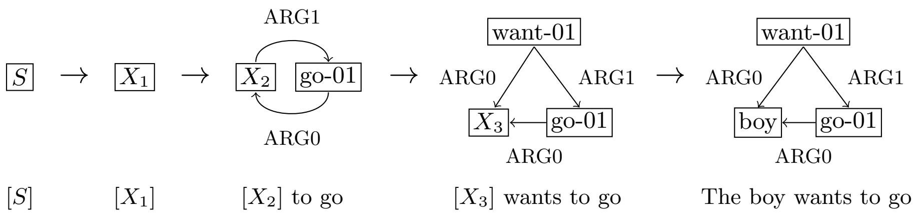

Fig. 6. Sample derivation for "The boy wants to go", where target strings are below AMR fragments—adapted from Song et al. (2017).

Song et al. use a *synchronous* NRG that differs in the sense that the RHS of a rule is not a single graph D, but a bijection-linked graph-string pair $(\langle D, T\rangle, \sim)$, where T is a target string and $\sim$ a mapping between nonterminals in D and T. The formalism must include nonterminals N, as well as source terminals Σ and target terminals Δ such that $D \in (N \cup \Sigma)^*$ and $T \in (N \cup \Delta)^*$. To solve the problem of AMR graph-to-string mapping, Song et al. train a heuristic algorithm to extract productions P during training. At test time, the graph transducer is then applied to linearize unseen AMRs.

The productions in the SNRG include *induced*, *concept*, and *graph glue rules.* The induced rules are found from training AMR-sentence pairs using an existing phrase-to-graph-fragment extraction algorithm. These rules initially contain only terminals from Σ and Δ. They are then pairwise merged when one graph-phrase pair contains the other, creating the final induced rules. Concept rules are introduced between pairs of AMR concepts and morphological strings that are not covered by induced rules. Finally, graph glue rules are added to generate all edges between nonterminals in graph-string pairs.

To find the best generated target string among all derivations, Song et al. use a log-linear model that is based on PBMT Koehn et al. (2003). The model includes ‘translation’ probabilities of AMR fragments given target strings, a reordering model, and a moving distance feature. The former two components are extended from PBMT, while the last component encodes merge distance between consecutive subgraphs. Performance-wise, the model improved over the previous state-of-the-art results of Flanigan et al. (2016) by 2.62 BLEU points.

2.3 Recent Approaches: Neural

Continuing in the vein of extending MT models to NLG, neural machine translation (NMT) has rapidly gained traction as the successor to statistical MT. Deep neural networks may have massive model complexity, yet the end-to-end nature of their training via backpropagation yields efficient parameter tuning. It is this capacity for streamlined, often highly parallel optimization that may put neural models ahead of their statistical counterparts.

Initial attempts to apply existing neural approaches to MT faced the barrier of variable-length inputs and outputs. Neural models for tasks such as image recognition or segmentation expect fixed-size inputs such that each input pixel directly maps to an output label. On the other hand, source and target sentences in MT have lengths and alignments that are not known a priori. Sequence-to-sequence models were born out of the need to handle such sequential data that exist in domains such as natural language.

Sequence-to-Sequence. Following previous work in neural sequence learning, Sutskever et al. (2014) use a pair of recurrent neural networks (RNNs) to encode and decode variable-length inputs. Specifically, they use two Long Short-Term Memory (LSTM) networks—one to compress the input sequence into a fixed dimension vector, the other to dynamically generate an output sequence. The main contribution is the use of LSTMs over vanilla RNNs in this encoder-decoder framework. LSTMs excel at retaining long-term dependencies, making them suitable for handling global patterns within long sequences. The ability of LSTM networks to ‘remember’ over many time steps is useful to (i) ensure that relevant global information of the input is stored in v_f and (ii) emit the current element e_t based on a sufficiently long history of past emissions.

In the encoder-decoder framework, the decoder must estimate the conditional probability of the output sequence of length T' given the compressed input vector v_f from the encoder of length T, decomposed as:

$$p(e_1 \dots e_{T'} \mid f_1 \dots f_T) = \prod_{t=1}^{T'} p(e_t \mid v_f, e_1 \dots e_{t-1}).$$

Each term in the product is computed as a softmax over the output vocabulary, where unseen words are mapped to an `UNK` symbol. Since output elements are predicted individually, the fact that output length T' is initially unknown is not problematic—each output sequence ends with an `EOS` symbol so that the distribution includes an option to terminate with nonzero probability. Similar to statistical decoders, finding the best output sequence involves running beam search using the decoder LSTM and using log probability of the partial output sequence to score hypotheses.

Sutskever et al. additionally find that reversing input training sequences allows for improved locality of mapping between inputs and outputs. For example, $[c, b, a] \mapsto [a', b', c']$ instead of $[a, b, c] \mapsto [a', b', c']$ results in mapping distances $[5, 3, 1]$ instead of $[3, 3, 3]$, respectively. The decoder begins by reading hidden state T from the encoder, so this ordering provides it with a 'fresher' representation of elements earlier in the input sequence.

On the WMT'14 English to French translation task, Sutskever et al. achieve a 3.2 BLEU score improvement over the previous neural baseline, but still trail behind a PBMT system by 0.5 points. Despite the model's 'depth' with an ensemble of five LSTMs, it appears to struggle with capturing sequential relationships that a simpler statistical model can.

Soft Attention. More or less concurrently with Sutskever et al., Bahdanau et al. (2015) introduce a sequence-to-sequence model specialized for MT. In previous model variants, requiring the encoder to compress all inputs to vectors of fixed size still limits the model's ability to process dynamic-length inputs. They propose a *soft attention* component (Fig. 7) that lets the model learn 'soft' weightings of individual input elements based on relevance to the current target word prediction. This way, the model learns input context vectors unique to each target prediction, instead of pre-computing a representation shared among all decoder steps. Even in the absence of ground-truth alignments, a neural model may learn task-specific patterns in attending to inputs.

To more precisely describe the soft attention decoder, the context vector v_f from the model by Sutskever et al. becomes v_t for decoder step $t \in [1, T']$, where:

$$v_t = \sum_{i=1}^{T_f} \alpha_{ti} h_i,$$

$$\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_{j=1}^{T_f} \exp(e_{tj})}.$$

In the above equations, α_{ti} represents a weight of a particular hidden state h_i for encoder input $i \in [1, T_f]$. The encoder producing these hidden states is not a single LSTM, but a bidirectional RNN that encodes hidden states in left-to-right and right-to-left passes over the input. At every decoder step t, T_f such weights—one per input hidden state—feed into v_t. As the weights $\{\alpha_{ti} \mid 1 \leq i \leq T_f\}$ specify a distribution over inputs, they must be normalized to sum to one. The vector e_{ti} represents an *alignment model* for how closely input i relates to output t. It is calculated as a function of the previous decoder state s_{t-1} and an encoder state h_i. The soft attention mechanism allows a model to 'jointly align and translate' since cost function gradients can be propagated backwards in an end-to-end fashion. From another angle, this attention mechanism facilitates selective retrieval; the decoder can simultaneously rank and fetch pertinent inputs.

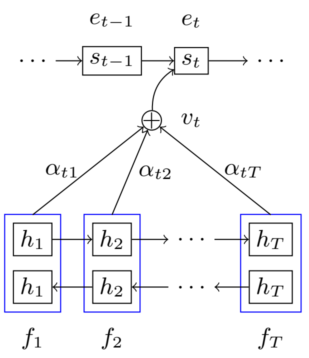

Fig. 7. Illustration of soft attention mechanism, adapted from Bahdanau et al. (2015). The $\oplus$ symbol denotes an alignment-weighted sum of hidden states stored in v_t.

Performance-wise, the system surpasses a baseline encoder-decoder model (Cho et al., 2014) both quantitatively and qualitatively. For the former, Bahdanau et al. achieve a fairly stable BLEU score across sentences of increasing length up to sixty words. The baseline model BLEU scores, however, steadily decreased when length exceeded about twenty. This supports the idea that soft attention remedies the bottleneck effect of compressing inputs into vectors of uniform length. The proposed model also outperforms baselines by 7.57–8.93 BLEU points, and it slightly overtakes a PBMT model when trained on sentences of within-vocabulary words only. This may imply that the sparse, high-dimensional vocabulary is a salient obstacle for neural models to overcome. Finally, including explicit alignment weights in the model provides transparency in aligment quality—it is possible to visualize relationships between source and target words.

Paired Parsing and Generation. As mentioned in Section 1.1, NLG models often opt to linearize AMR graphs to produce an intermediate sequence. This step creates a more canonical graph representation and is in fact a necessity to coerce graphs into input sequence format for sequence-to-sequence models.

Konstas et al. (2017) present a novel training procedure for both an AMR parser and generator that is invariant to linearization order. Their system relies on a large unlabeled corpus of sentences to help alleviate the sparsity of parallel sentence-AMR corpora. More concretely, the paired training approach involves (i) bootstrapping a parser using self-training on unlabeled data and (ii) feeding graphs parsed in the previous step to a generator as pre-training data. In addition to this routine, Konstas et al. also use a preprocessing method that simplifies AMRs and their corresponding sentences through a form of *anonymization*. In essence, anonymizing subgraphs and their aligned text spans reduces graph structure complexity and vocabulary size. This reduced complexity in the target and source space helps compensate for data sparsity constraints.

To highlight the overlap between AMR parsing and generation, Konstas et al. formulate the two problems as the respective predictions:

$$\hat{a} = \arg\max_{a} f(a \mid s; \theta_P),$$
$$\hat{s} = \arg\max_{s} f(s \mid a; \theta_G),$$

where a denotes the AMR graph and s the sentence. The predictor family f and sequence-to-sequence model architecture are common among the two tasks. The primary difference lies in the separate parameters θ_P for parsing and θ_G for generation. Interestingly, Konstas et al.'s work is inspired by a back-translation MT system (Sennrich et al., 2016) that uses automatic translations of a large, target-side monolingual corpus as added parallel data. This tactic enriches training for low-resource language pairs, and it transfers to the AMR-sentence 'translation' task as well.

In this work, the sequence-to-sequence model extends that of Bahdanau et al. (2015). Konstas et al. make two changes to the encoder: (i) concatenate forward and backward LSTMs across all time steps instead of only the relative final ones and (ii) add dropout to the first layer. They also employ a copy mechanism in the generator decoder, which copies an AMR concept label as token output when the decoder emits the `UNK` symbol.

At a high level, the paired training method leverages an unlabeled sentence set S_e from which AMR graphs are parsed and then used as training data for the generator. Empirically, the set S_e is of size 200K–20M drawn from the Gigaword corpus. The true dataset D serves as pre-training data for the parser and fine-tuning data for the generator. Below is an overview of the full training procedure.

1. Train the parser on original dataset D to learn parameters θ_P.
2. Tune the parser with N cycles of self-training, drawing $k \cdot 10^i$ samples from S_e each cycle for $i \in [1, N]$.
   (a) Parse samples A_e^i from unlabeled sentences S_e^i.
   (b) Update θ_P by training on parallel sets (A_e^i, S_e^i).
   (c) Fine-tune θ_P on the original data D.
3. Given $k \cdot 10^N$ new samples from S_e, parse A_e^N from this sampled S_e^N.
4. Train the generator on parallel sets (A_e^N, S_e^N) to learn parameters θ_G.
5. Fine-tune θ_G on original dataset D.

In addition to the above training routine, the method for preprocessing AMRs also has direct bearing on the models' performance. One technique is anonymization, which plays a key role in eliminating rarely occurring AMR entities. Konstas et al. found that certain types—including named entities, dates, and numbers—make up 31.2% of the word vocabulary with 83.4% of them occurring fewer than five times. They anonymize AMRs by collapsing subgraphs whose roots are named entities or quantities with a single categorical node, as well as normalizing format of date entities. In the parallel sentences, they find token spans aligned to these collapsed subgraphs and replace the sentence span with corresponding anonymized tokens. Besides modifying the topology and concept labels of AMRs, Konstas et al. also linearize them in a standardized way; they traverse them in a DFS order that includes edge labels of the backward pass (i.e., each edge is covered twice). In its string format, each graph is composed of its root's concept label followed by edge labels and child subgraphs—this recursive scope is captured by nested pairs of parentheses.

Following the paired training and preprocessing steps described above, Konstas et al. markedly improve upon parsing and generation baselines. Using 20M external unlabeled sentences, the parser surpasses a prior sequence-to-sequence model (Peng et al., 2017) by 10.1 Smatch (Cai and Knight, 2013) F1 points. This improvement dips to 3.5 points without external data, proving the merits of careful preprocessing alone. With the same amount of external data, the generator outperforms systems of Pourdamghani et al. (2016) and Flanigan et al. (2016) by 6.9 and 10.8 BLEU points, respectively. Finally, Konstas et al. conduct experiments showing that the model is insensitive to linearization order. When given a random as opposed to human annotator AMR traversal, the generator's BLEU score is only 1.4 points lower in the former than in the latter case. There is ample evidence that Konstas et al.'s application of an external corpus and preprocessing boosts sequence-to-sequence model performance on AMR.

Graph-to-Sequence. Although Konstas et al. develop a model that is largely unaffected by linearization order, they fail to remedy the constraint itself. Regardless of the heuristic, flattening a graph often misrepresents its underlying structure (e.g., nodes that are neighbors may be pushed far apart). Song et al. (2018) address this limitation of sequence-to-sequence models by using a graph-based LSTM encoder instead (for generation specifically). Under this graph-to-sequence model, an AMR's graph structure is directly encoded; node information is propagated along graph edges via message passing over time steps. Using LSTM cells for recurrent propagation supports a more global distribution of local context across a graph, just as in the standard sequence-to-sequence encoder across a sequence. As per Konstas et al. (2017), the model uses a copy mechanism in the decoder to copy rare entities to the output. By representing the topology of AMR more accurately, Song et al. reach higher BLEU on generated sentences than Konstas et al. (2017).

Given an AMR graph denoted $G = (V, E)$, the node hidden states are vectors $\{h_i^t \mid v_i \in V, t \in [1, T]\}$, where the number of time steps T is a hyperparameter. Each node's hidden state is updated with node and edge label vectors from its incoming and outgoing neighbors. For vector h_i^t, its neighborhood node state v_i^t and edge state u_i^t are:

$$\begin{aligned} v_i^t &= \sum_{(i,j,l) \in E_{\mathcal{N}}(i)} h_j^{t-1}, \\ u_i^t &= \sum_{(i,j,l) \in E_{\mathcal{N}}(i)} e_{ijl}^{t-1}, \end{aligned}$$

where (i, j, l) is an edge tuple from $i \to j$ with relation label l; $E_{\mathcal{N}}$ contains edges to its neighborhood $\mathcal{N}$. This computation is done separately for incoming and outgoing edge sets (i.e., the above equations define outgoing edges and i and j are swapped in the case of incoming edges).

To compute the updated node hidden state h_i^t, an LSTM unit with a gate-wise combination of $\{v_i^t, u_i^t\}$ for each of the two edge directions (i.e., incoming and outgoing) is used. More concretely, let gate g be in $G = \{f, p, o, \widetilde{c}\}$, or forget, input, output, and candidate cell gates.

$$\begin{aligned} g_i^t &= \sigma(W_1 v_{i1}^t + W_2 v_{i2}^t + W_1' u_{i1}^t + W_2' u_{i2}^t), \\ c_i^t &= f_i^t \odot c_i^{t-1} + p_i^t \odot \widetilde{c}_i^t, \\ h_i^t &= o_i^t \odot \tanh(c_i^t), \end{aligned}$$

where g_i^t is found for each gate in G using separate weights in $\{W_1, W_2, W_1', W_2'\}$; subscripts 1 and 2 indicate incoming and outgoing directions, respectively.

The outgoing edge embeddings are initialized as:

$$e_{ijl} = W_e([x_l; x_j; h_{jc}]) + b_e,$$

where x_l is embedding of edge label l and x_j is that of v_j's node label; h_{jc} is the final hidden state of a character LSTM on v_j. The purpose behind concatenating h_{jc} is to help improve representations of out-of-vocabulary labels. Incoming edge embeddings are defined analogously to the above.

Song et al. use a bidirectional LSTM decoder with soft attention as described by Bahdanau et al. (2015). They modify alignment model vector e_{ti} to depend on final hidden state h_i^T and edge state u_i^t, as well as on a coverage vector γ_{t-1} (Tu et al., 2016). The latter vector is a sum over the attention distribution:

$$\{\alpha_{t'i} \mid 1 \leq t' \leq t-1, v_i \in V\}.$$

They add a copy mechanism, which is an interpolation of (i) the per-node attention distribution and (ii) the per-word vocabulary distribution[4] (i.e., the former is weighted by θ, the latter $1 - \theta$). The decoder can more heavily weight the attention distribution in order to 'copy' a node's label. Song et al. claim that the copy mechanism is advantageous over the anonymization used by Konstas et al. (2017) since no manual preprocessing rules are required—the model learns to copy low frequency labels automatically.

In terms of results, Song et al. reach a BLEU score of 23.3 that is 1.3 points higher than that of Konstas et al.. When using an external corpus of size 200K and 2M, the improvement is 0.8 and 0.7 points over the respective models by Konstas et al. (2017). This suggests that the proposed model is also able to fully leverage large-scale amounts of data. In addition to better performance, Song et al. note that the model offers improved efficiency; reminiscent to loopy belief propagation, the node updates can occur in parallel rather than sequentially. One can visualize the information being propagated across increasing diameters of a node's neighborhood instead of left-to-right over a linearization. In addition, using a graph structure directly allows for distribution of edge-level features for richer node representations.

[4] Computed from the concatenated context vector v_t and LSTM state s_t.

3 Future Directions

After reviewing the existing literature, finding a more direct alignment between graphs and sentences seems to be a driving goal in NLG from AMR. Both statistical and neural methodologies grapple with how to represent the rich, unordered graph structure such that English generation is more tractable. As alluded to in Section 1.1, this complexity may be a bottleneck in making great strides on the NLG task for semantic graphs.

In this section, we will first compare the merits of existing work with respect to this problem and synthesize the findings. Then we will revisit transition-based parsing and its potential to simplify AMR-to-text generation. Lastly, details of a novel graph-to-sequence system will be discussed.

3.1 Synthesis

Statistical. A 'common ancestor' of several statistical approaches in Section 2.2 has been PBMT (Koehn et al., 2003). It is clear that learning to map source-target phrase pairs using joint phrase and reordering distributions performs well on string domains. When adapting this model to the generation task, the extent to which AMR graph structure is retained varies across methods. There is evidence that learning word-order graph traversals may in fact be more effective than reordering rules.

Pourdamghani et al. (2016) use a classifier-based method to minimize crossings between aligned concept-phrase pairs in a linearized graph. In contrast, Flanigan et al. (2016) convert the AMR to a spanning tree before applying a tree transducer that applies rules resulting in the best sequential ordering of tree concept and argument realizations. Although it is arguable that Flanigan et al. (2016) use a more sophisticated tree representation than the graph linearization of Pourdamghani et al. (2016), the latter work is ultimately more faithful to the graph structure. In other words, Pourdamghani et al. learn a direct graph linearization that not only avoids data loss from graph-to-tree conversion, but may even preserve added graph-related semantics. One can conclude that modifying the true AMR structure (e.g., removing re-entrancies) may negatively affect quality of the generated text.

Similar to Flanigan et al. (2016), Song et al. (2017) use a grammar-based system to map between structures and a decoder inspired by PBMT. However, their grammar rules are graph-to-string to avoid error propagation from graph-to-tree conversion. Although the rules closely capture input structure, the performance lags behind that of Pourdamghani et al. (2016). This indicates that using a PBMT-based decoder may not be practical for the generation problem—perhaps it induces loss in structural information.

Neural. The neural models covered in Section 2.3 stem from the family of sequence-to-sequence models that have an encoder-decoder architecture. Unlike most statistical methods, these neural models are less reliant on carefully chosen features or grammar rules than on data-driven parameter optimization. As such, their abundance of parameters make them susceptible to the data sparsity common in parallel AMR-sentence corpora. To combat this issue, preprocessing routines to simplify the AMR structure have been adopted. An additional approach involves using large amounts of automatically annotated external data. Even if reducing input dimensionality or increasing data quantity solves the sparsity problem, the issue of learning graph-sentence alignments remains.

The paired training procedure between both a parser and generator by Konstas et al. (2017) leverages enough external training data that linearization order seems to have a negligible effect. However, they acknowledge that the traversal based on human-annotated AMRs still results in the highest BLEU score. It is also important to note their extensive amount of AMR preprocessing to achieve the desired amount of 'anonymization'; this amount of manual work is not only time-consuming but may result in loss of information from AMR entities. As an alternative, Song et al. (2018) use a novel graph-to-sequence model that removes the need to flatten input AMRs. To deal with out-of-vocabulary words, Song et al. (2018) use only a trainable copy mechanism instead of anonymization and post-processing. This allows the model to modulate the decision to copy concept labels instead of relying on preprocessing.

As in previous models, Song et al. (2018) use soft attention (Bahdanau et al., 2015) that permits the model to learn alignments between target words and the source graph. Although this may appear reasonable, the role of attention in the graph-to-sequence model may be more fraught than in a linearized sequence-to-sequence model. In the former, node updates in the encoder occur over possibly overlapping node-level neighborhoods from the input graph. In the latter, the predetermined AMR linearization controls the path through which node-wise information is propagated. It is possible that the node hidden states from the graph encoder contain less local, traversal-specific context than in the sequence variant. If so, the attention mechanism would need to jointly traverse the AMR concepts and align concepts to word emissions. Perhaps the overhead of both such tasks would overburden a single model component, especially given the data sparsity issue. The task that follows, then, is to find a way to decouple traversal and alignment.

3.2 Transition-Based AMR Parsing

Given the overarching issues found in Section 3.1, an ideal model will accurately reflect input structure, as in the graph encoder, while preserving the local context of a word-order graph traversal. Using an encoder-decoder model, it is sensible to retain the graph encoder by Song et al. (2018) and modify only the decoder. This decoder would need to both traverse the AMR and emit English tokens aligned to each concept. As introduced in Section 1.2, the cache transition system (Gildea et al., 2018) is capable of mapping between the input word order and bags of a graph (i.e., cache configurations) in an incremental manner. We can thus reframe the decoding problem as learning a sequence of transition-based actions parsing, or rather decomposing, the graph according to word order of the target sentence.

Properties of AMR. As introduced in Section 1, an AMR (Banarescu et al., 2013) annotates the meaning of a sentence through a graph of concepts and their connective relations. These concepts can represent verbs, nouns, or other concepts labeled by PropBank (Palmer et al., 2005) framesets (e.g., *kick.01*). Accordingly, the relations depend on the concept's semantic role (e.g., arguments *arg0* for the 'kicker' and *arg1* for the object kicked). AMR was designed to be a comprehensive sembank that is both human-readable and machine-traversable. Satisfying these properties, along with ease of annotation, comes at the cost of less specificity with respect to parallel sentences.

With more abstraction, finding subgraph-phrase alignments in AMR is not always straightforward. Specifically, the laconic nature of AMR means that single concepts may commonly map to multi-token phrases. There could also exist null alignments; a concept may not map to any words, and vice versa. We observe, then, that the local context of an AMR concept (i.e., its relations and immediate neighbors) is highly relevant in a text generation setting; concept labels in isolation may not contain enough discriminative information.

Sequence-to-Action-Sequence. Section 1.2 covered the mechanics of a cache transition system that gradually parses an input sentence into its AMR. Note that the cache size required to parse a given AMR depends on the left-to-right order of input words. More explicitly, let $\pi(T)$ be the vertex order of the preorder traversal of graph G's tree decomposition T (e.g., Fig. 3), where the newly introduced vertex of each bag is emitted. This order is precisely the word order of the input sequence. Recall that each bag of T maps to a specific cache configuration at a point in a parser's run. The minimum width over the set of all derivation trees $\{T'\}$ for G—that represents the possible parser runs—is limited by the per-tree vertex order $\pi(T')$ (Gildea et al., 2018). Due to this correspondance, we can conclude that the series of cache configurations over the course of a parser run captures information about a word-order traversal of G's vertices.

To exploit the local context present at each step of a transition-based parser run, Dyer et al. (2015) propose the stack LSTM structure for neural syntactic parsing. These stack LSTMs maintain continuous representations of a parser's state following push or pop operations. For insertion into and removal from a data structure, they maintain stack pointers that support dynamic reordering of elements (i.e., one can remove an element's back-pointer to pop it, then attach the pointer to a new element for push). This way, the current stack top contains a history-specific context of the data structure's contents. Dyer et al. apply stack LSTMs to the buffer, stack, and transition sequence to track their states. Although this work pertains to syntactic dependency parsing rather than semantic generation, it illustrates the value of sequential parser configurations for learning parser actions.

A semantic parsing method proposed by Buys and Blunsom (2017) uses a standard bidirectional LSTM encoder and *hard-attention* decoder to directly map sentences to parser transition sequences, referred to as a *sequence-to-action-sequence* approach. Broadly, the training is supervised by transition sequences from an *oracle* algorithm that deterministically parses the input. In contrast to soft attention Bahdanau et al. (2015) that learns distributions over encoded inputs, the hard-attention mechanism enforces discrete alignments between nodes and token spans using a pointer network (Vinyals et al., 2015).

Similar to Buys and Blunsom, Peng et al. (2018) use a sequence-to-action-sequence model to output cache transition parser actions for AMR. Peng et al. use two LSTM encoders that operate in tandem on respective word and concept sequences. They map word spans to anonymized AMR concepts before parsing, instead of delaying alignment until parser execution. Thus, alignment is decoupled into two stages: partitioning the sentence into phrases and phrase-concept mapping. For each transition prediction, Peng et al. use hard attention on both buffer elements and input words via a pair of per-sequence pointers sweeping from left to right. Crucially, they also use features from the current transition system configuration to provide local context to the decoder. Just as in Dyer et al. (2015), this work further supports the benefits of incremental context from a transition system.

3.3 Proposed System

Despite the fact that Section 3.2 outlines approaches for parsing, we will argue that similar systems can apply to AMR-to-text generation. Specifically, we will show how transition-based parsing aids in the goals of learning word-level AMR traversals and local node-wise context identified in Section 3.1.

Parser Transitions. On the left side of Fig. 8, a typical state diagram for an AMR cache transition parser is shown. In addition to the standard push and pop transitions, the shift and arc/noarc actions are added. The former indicates a transfer of focus to the next buffer element, while the latter is an indicator variable for whether to build an arc between two cache elements (i.e., the rightmost element and any remaining one).

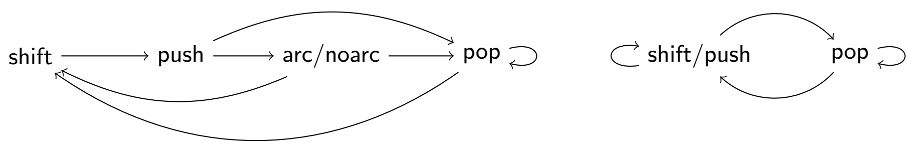

Fig. 8. State diagrams for parsing (left) and proposed generation (right) using the cache transition system.

In Section 1.1, we established that generation can be seen as the inverse of parsing—instead of creating arcs between concepts, we destroy them, emitting the words aligned to each concept upon removal from the AMR. On the right side of Fig. 8, a possible state diagram for the generation task is pictured. Note that the arc/noarc action has been eliminated since edges no longer need to be built. For generation, it is also necessary to choose the next concept to process from the unordered AMR graph, since elements are no longer arranged in a sequence. In aggregate, these choices are essentially the word-order traversal order of the AMR graph discussed in Section 3.1. Thus, the generation variant of shift must be associated with a buffer index of the subsequent element for push. Identical to the parsing variant, the push action is accompanied by a chosen cache index to evict. For simplicity, we merge the shift transition with push during generation, since the latter transition always follows the former.[5]

Model Architecture. The sequence-to-action-sequence model architecture appears suitable for parsing, and Peng et al. (2018) show that parser context can be incorporated into predictions. This property has proven helpful for generating parser actions in an incremental fashion. For the generation task, however, the following tasks still remain.

1. Selecting a word-order linearization of the AMR.
2. Producing English tokens aligned to each element of the concept sequence.

In concordance to the system by Song et al. (2018), we will use the same graph encoder in the proposed model, which will therefore become a *graph-to-action-sequence* approach. To capture both the structural context of the AMR (for task 1) and alignments between concepts and phrases (task 2), we will draw inspiration from a syntactic MT system by Wu et al. (2017). Their decoder consists of parallel word and action LSTMs, each of which produces one of the below sequences.

1. Shift-reduce parser actions to capture syntactic transformations between a sentence pair.
2. Target language tokens that are based on the current syntactic tree fragment. These are only emitted if the current parser LSTM prediction is shift.

Note that the above two sequences conform to the tasks identified earlier in the paragraph. We have also reduced our parser action vocabulary size to two, and these actions map nicely onto the shift and reduce actions of the syntactic parser used by Wu et al. (i.e., shift/push for the former and pop for the latter).

The only major discrepancy between our AMR-to-text objective and that of Wu et al. that remains is the complexity of alignments between AMR concepts and English tokens. In Wu et al. (2017), each parser action corresponds to exactly one token emitted (i.e., one-to-one alignment). As noted in Section 3.2, however, the parallel AMR-sentence corpora can contain one-to-many (and the reverse), as well as null alignments. To remedy this issue, we will allow the action LSTM

[5] The generation state diagram is also pleasantly symmetric and all state transitions are possible. This may simplify model predictions since no action sequence is 'illegal'.

to repetitively query the word LSTM until it receives a ‘stop’ symbol from the latter. This symbol will be added to the English word vocabulary prior to model execution and can be seen as an ‘end-of-phrase’ symbol for words aligned to a given concept. This flexibility in communication between the two LSTMs allows for a variety of different alignments found in the AMR corpora.

4 Conclusion

Throughout this paper, we have followed the evolution of a variety of statistical and neural approaches for AMR-to-text generation, in addition to related problems in machine translation and parsing. The most pressing barriers of the former are the structural complexity of AMR graphs and wide vocabulary of both concept labels and English tokens. To overcome these challenges, we hypothesize that it is best for a model to separately learn the (i) word-order traversal of AMR and (ii) concept-aligned English tokens. Accordingly, we propose a neural graph-to-action-sequence model that leverages both graph-specific input modeling and the reduced output space of cache transition parser sequences. The configurations of the parser provide further context relating to the input graph structure and output word order.

References

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2015.

L. Banarescu, C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight, P. Koehn, M. Palmer, and N. Schneider. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, 2013.

S. Bangalore and O. Rambow. Exploiting a probabilistic hierarchical model for generation. In *Proceedings of the 18th Conference on Computational Linguistics-Volume 1*, pages 42–48, 2000.

J. Buys and P. Blunsom. Robust incremental neural semantic graph parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1215–1226, 2017.

S. Cai and K. Knight. 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)*, pages 748–752, 2013.

K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

C. Dyer, M. Ballesteros, W. Ling, A. Matthews, and N. A. Smith. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 334–343, 2015.

J. Engelfriet and G. Rozenberg. Node replacement graph grammars. In *Handbook of Graph Grammars and Computing by Graph Transformation: Volume 1: Foundations*, pages 1–94. World Scientific, 1997.

J. Flanigan, S. Thomson, J. Carbonell, C. Dyer, and N. A. Smith. A discriminative graph-based parser for the Abstract Meaning Representation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1426–1436, 2014.

J. Flanigan, C. Dyer, N. A. Smith, and J. Carbonell. Generation from Abstract Meaning Representation using tree transducers. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 731–739, 2016.

D. Gildea, G. Satta, and X. Peng. Cache transition systems for graph parsing. *Computational Linguistics*, 44(1):85–118, 2018.

L. Huang, K. Knight, and A. Joshi. Statistical syntax-directed translation with extended domain of locality. In *Proceedings of AMTA*, pages 66–73, 2006.

A. K. Joshi. An introduction to tree adjoining grammars. *Mathematics of Language*, 1:87–115, 1987.

K. Knight and V. Hatzivassiloglou. Two-level, many-paths generation. In *Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics*, pages 252–260, 1995.

K. Knight and I. Langkilde. Preserving ambiguities in generation via automata intersection. In *AAAI/IAAI*, pages 697–702, 2000.

P. Koehn, F. J. Och, and D. Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 48–54, 2003.

I. Konstas, S. Iyer, M. Yatskar, Y. Choi, and L. Zettlemoyer. 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, 2017.

I. Langkilde and K. Knight. Generation that exploits corpus-based statistical knowledge. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1*, pages 704–710, 1998.

C. Lyu and I. Titov. AMR parsing as graph prediction with latent alignment. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 397–407, 2018.

M. Palmer, D. Gildea, and P. Kingsbury. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106, 2005.

K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

X. Peng, C. Wang, D. Gildea, and N. Xue. Addressing the data sparsity issue in neural AMR parsing. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 366–375, 2017.

X. Peng, L. Song, D. Gildea, and G. Satta. Sequence-to-sequence models for cache transition systems. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1842–1852, 2018.

N. Pourdamghani, K. Knight, and U. Hermjakob. Generating English from Abstract Meaning Representations. In *Proceedings of the 9th International Natural Language Generation Conference*, pages 21–25, 2016.

R. Sennrich, B. Haddow, and A. Birch. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, 2016.

L. Song, X. Peng, Y. Zhang, Z. Wang, and D. Gildea. AMR-to-text generation with synchronous node replacement grammar. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 7–13, 2017.

L. Song, Y. Zhang, Z. Wang, and D. Gildea. A graph-to-sequence model for AMR-to-text generation. In *Proceedings of the 56th Annual Meeting of the*

Association for Computational Linguistics (Volume 1: Long Papers), pages 1616–1626, 2018.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

Z. Tu, Z. Lu, Y. Liu, X. Liu, and H. Li. Modeling coverage for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 76–85, 2016.

O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

S. Wu, D. Zhang, N. Yang, M. Li, and M. Zhou. Sequence-to-dependency neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 698–707, 2017.