CS447 Literature Review: AMR Parsing

Wenqi He,
wenqihe2@illinois.edu

April 30, 2023

Abstract

This paper is a brief survey of Abstract Meaning Representation (AMR) semantic parsing. AMR represents the meaning of a sentence as a directed acyclic graph of concepts and semantic relations. AMR parsers can be roughly divided into three categories: grammar-based, graph-based and transition-based. In this review, we explain the architectures of both traditional feature-based models and more recent neural models in these categories, discuss their similarities and differences, and compare their performances.

1 Introduction

One of the fundamental concepts in natural language semantics is the principle of compositionality, which says that the meaning of a complex expression is determined by the meaning of its syntactic constituents and the rules to combine them. This principle underlies our ability to generate infinite utterances and meanings with only finite primitives and rules.

Computationally, the field of lexical semantics has been dominated by vector space models. Vector representations allow for algebraic operations, including simple compositions such as compound words, but it's not obvious how we could combine them in a recursive and hierarchical way. In contrast, traditional symbolic representations such as first-order predicate logic are a natural fit for compositional semantics due to their inherent recursive structure. The task of semantic parsing is to convert natural language sentences into such logical forms.

Semantic parsing is not only interesting linguistically, but it also provides structured intermediate representations that can be used directly for many interesting tasks such as natural-language interface and natural-language programming – e.g. mapping natural language to Bash commands (Lin et al., 2018), SQL queries or general-purpose source code (Rabinovich et al., 2017). Furthermore, with semantic parsers, we could potentially build more interpretable NLP models than those relying solely on deep learning.

In this review, we will focus on one particular type of symbolic representation, Abstract Meaning Representation (AMR), and we will focus on parsing instead of application. In particular, we will discuss in detail how neural techniques such as word embeddings, RNNs and attention mechanisms can be incorporated into different types of AMR parsers, achieving increasingly better results than earlier non-neural models.

2 Background

Abstract Meaning Representation (AMR) is a neo-Davidsonian whole-sentence meaning representation. It encodes sentence meaning as a directed-acyclic graph where leaf nodes represent concepts; internal nodes (variables) represent entities, events, properties or states; and edges such as `:arg0` or `:time` represent semantics relations. Concepts in AMR are mostly based on PropBank framesets. Not only are verbs and verb phrases mapped to their corresponding frames, many adjectives and nouns, including nominalization of verbs and "-er" nouns, are also represented using verb frames.

As an example, the meaning of "an attractive spy" can be represented in AMR as:

```
(s/spy
  :arg0-of (a/attract-01
    :arg1 (w/woman)))
```

In this AMR graph, `s`, `a` and `w` are variables; `spy`, `attract-01` and `woman` are concepts; and `/` (meaning "instance of"), `arg0-of`, and `arg1` are semantic relations. Equivalently, this graph can be expressed in first-order predicate logic as a conjunction of propositions, where each semantic relation is encoded as a binary predicate:

$$
\begin{aligned}
&\texttt{instance(s, spy)} \wedge \\
&\texttt{instance(a, attract-01)} \wedge \\
&\texttt{instance(w, woman)} \wedge \\
&\texttt{arg0-of(s, a)} \wedge \\
&\texttt{arg1(a, w)}
\end{aligned}
$$

There are two main challenges to AMR parsing. First, AMR is syntax agnostic, which means that it doesn't specify how each concept in the graph corresponds to words in the source text, therefore an AMR parser must learn to translate word spans into concepts without any alignment annotation in the training data. Second, AMR uses reentrant nodes to model non-local dependencies such as coreferences, which makes AMR a graph rather than a tree. Because of this reentrancy, we can't simply use naive shift-reduce parsers, as they can only handle projective dependencies.

The standard metric for evaluating AMR parsers is the Smatch score (Cai and Knight, 2013), which compares the parser output with the gold graph. Formally, the Smatch score is defined as the maximum F_1 score that can be achieved using any one-to-one variable (i.e. node) mapping between the two graphs. Precision is defined as the percentage of edges in the parser output that are present in the gold graph, and recall is defined as the percentage of edges in the gold graph that are also identified by the parser, and

$$F_1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

3 Overview

There are three main categories of AMR parsers:

- **Grammar-based parsing** is essentially an extension of combinatory categorial grammar (CCG) chart parsing with a lexicon induced from the training corpus rather than fixed. Chart parsing relies on bottom-up dynamic programming algorithms, such as the Viterbi algorithm

and the Cocke–Younger–Kasami (CKY) algorithm. Since combinatory rules are predetermined, this approach focuses on learning an effective lexicon from the training data. To this end, a scoring function for partial derivations is trained, both for performing greedy or beam search at inference time and for deciding which categories to assign to each word at training time. With grammar-based parsing, word-concept alignments are captured by the lexicon, and reentrancy can be handled in a second pass.

- **Graph-based parsing** in general constructs AMR graphs in two passes. The first pass translates the source sentence into a set of concepts, using either an aligner or an encoder-decoder neural model. The second pass generates edges, using a learned scoring function for edges and an adapted maximum spanning arborescence/tree algorithm, such as Kruskal's algorithm or Chu-Liu/Edmonds' algorithm. As an optimization, nodes and edges could be generated in one pass by simply duplicating reentrant nodes and labeling them with the same ID, as with transition-based parsing.

- **Transition-based parsing**, or "shift-reduce" parsing, constructs AMR graphs incrementally with a fixed set of actions. The model predicts the next action given a particular parser configuration: the current state of the stack, buffer, partial graph, etc. Word-concept alignments are handled by special parser actions that concatenates words and replaces spans of words with concepts. Reentrant nodes are handled with either a special "swap" action, or by generating duplicate nodes marked with the same ID.

For each of these categories, we will review several neural models as well as their feature-based non-neural precursors. There are also less sophisticated models such as Konstas et al. (2017), which simply uses a seq2seq model to translate each source sentence into a linearized AMR representation. We do not include such models in this review, as conceptually they are relatively straightforward.

4 Grammar-Based Parsing

4.1 Artzi et al. (2015)

Artzi et al. (2015) is an extension of CCG semantic parsing. The parser doesn't produce AMR representations directly, but instead uses an equivalent logic form, which is an extension of typed lambda calculus with globally scoped Skolem Terms and Skolem IDs. Variables in AMR are represented as as Skolem terms such as $\mathscr{A}_1(\lambda x.boy(x))$ or references to Skolem IDs, such as $\mathscr{R}(1)$. Semantic relations (edges) are simply represented as binary predicates, such as ARG0(x,y) and ARG0-of(x,y).

Given a learned lexicon, the parser operates in two stages. In the first stage, CKY chart parsing handles the compositional parts of AMR by producing the top k *underspecified* logical forms, where all Skolem ID references are replaced with placeholder $\mathscr{R}$(ID) and some semantic relations are replaced with placeholder REL or REL-of. In the second stage, all undetermined (co)references and semantic relations are resolved, resulting in a fully-specified logical form that can be translated back to an AMR graph. More specifically, for each underspecified logical form, the parser constructs a factor graph to score each node/edge label assignment based on features such as selectional preferences and then applies a beam search to find the best k assignments.

The predicted likelihood of parser output given the source sentence x is marginalized over all

derivations that produces the same fully-specified logical form:

$$P(z \mid x) = \sum_{d \in \mathscr{D}'(x)} \frac{\theta^\top \phi(x,d)}{\sum_{d \in \mathscr{D}(x)} \theta^\top \phi(x,d)}$$

where $\phi(x)$ is vector of indicator (binary) features. $\mathscr{D}(x)$ is the set of all possible derivations of x, and $\mathscr{D}'(x)$ is the subset that produces z.

During training, the model induces a factored lexicon (Kwiatkowski et al., 2011) consisting of lexemes and lexical templates, and simultaneously learns the parameters of the scoring function.

Given a source sentence, a two-pass algorithm is used to generate new lexemes and templates. First, new lexemes are extracted from the source sentence and paired with all existing templates. With the updated lexicon, a parse will be attempted, and if expanding lexemes is not sufficient to produce a successful parse, a recursive split algorithm will be applied to the sentence to generate new possible templates.

In each epoch, the lexicon is first tentatively expanded using the above algorithm for the entire training corpus; then each sentence is parsed and scored, and a hard gradient computed to update θ; finally, at the end of each epoch, the previously expanded lexicon is pruned, keeping only those that actually occur in successful parses. The complete training algorithm is listed in Algorithm 1

Algorithm 1 Learning Algorithm (Artzi et al., 2015)

$\Lambda \leftarrow \Lambda_0$
for each epoch **do**
  for each sample (x,z) **do**
    $\Lambda \leftarrow \Lambda \cup \textsc{GenEntries}(x,z,\theta,\Lambda)$
  end for
  for each batch **do**
    $\Delta \leftarrow 0$
    for each sample (x,z) **do**
      $\Delta \leftarrow \Delta + \textsc{HardGrad}(x,z,\theta,\Lambda)$
    end for
    $\theta \leftarrow \theta + \mu \cdot \textsc{AdaGrad}(\Delta)$
  end for
  $V \leftarrow \bigcup_{(x,z)} \textsc{BestParse}(x,z,\theta,\Lambda)$
  $\Lambda \leftarrow \bigcup_{d \in V} \textsc{Lexicon}(d)$
end for

5 Graph-Based Parsing

5.1 Flanigan et al. (2014)

JAMR (Flanigan et al., 2014), the first published AMR parser, uses a two-stage algorithm.

The first stage determines word-concept alignment. Specifically, the source sentence is segmented into word spans, each of which is mapped to a concept represented as a graph fragment, using the Viterbi algorithm with back pointers. For each prefix ending at index i, the best segmen-

tation is computed as follows:

$$S(i) = \max_{j<i} \max_{c \in C_{j:i}} \left\{ S(j) + score(w_{j:i}, j, i, c) \right\}$$

The second stage constructs a maximum spanning connected subgraph (MSCG), starting from the graph fragments (concepts) generated in the first pass. The algorithm proposed by the authors is an adaption of Kruskal's algorithm (see Algorithm 2), with various additional constraint specific to the problem of AMR parsing.

Algorithm 2 MSCG (Flanigan et al., 2014)

```
E ← E_concepts ∪ {e | score(e) > 0}
Q ← PriorityQueue{e | score(e) ≤ 0}
while not spanning and connected do
    e ← argmax_{e'∈Q}{score(e')}
    Pop(Q, e)
    if e connects two components then
        Add(E, e)
    end if
end while
```

The scoring functions for nodes and edges are simple linear functions: $\theta^\top \mathbf{f}(w_{j:i}, j, i, c)$ and $\psi^\top \mathbf{g}(e)$, respectively, where $\mathbf{f}$ and $\mathbf{g}$ are vectors consisting of both indicator and statistical features. The model is trained using subgradient descent:

$$\theta_i^{t+1} \leftarrow \theta_i^t - \frac{\eta}{\sum_{t'=1}^{t} s_i^{t'}} s_i^t$$

$$\psi_i^{t+1} \leftarrow \psi_i^t - \frac{\eta}{\sum_{t'=1}^{t} s_i^{t'}} s_i^t$$

5.2 Lyu and Titov (2018)

Lyu and Titov (2018) proposes a neural model based on BiLSTM encoder. As with Flanigan et al. (2014), Lyu and Titov (2018) also generates nodes and edges in two passes, but instead of mapping word spans to concepts, the problem of word-concept alignment is framed as a sequence tagging task by assuming an injective mapping from concepts to words.

In this probabilistic model, alignment $\mathbf{a}$ is modeled as a latent variable that is only involved in deriving the training objective. Formally, $\mathbf{a}_i$ is defined as an injective mapping from concept indices $i \in \{1, \ldots, m\}$ to word indices $\mathbf{a}_i \in \{1, \ldots, n\}$. At inference time, this index mapping is never computed, and the parser only needs to predict either a single concept or NULL for each input word, as predicting a concept for $\mathbf{w}_i$ is really the same as predicting concept $\mathbf{c}_j$ under the latent alignment $\mathbf{a}_j = i$.

Given word sequence $\mathbf{w}$, concept sequence $\mathbf{c}$, and latent alignment $\mathbf{a}$, the model predicts the output probability, marginalizing over all possible alignments:

$$P(\mathbf{c}, \mathbf{R} \mid \mathbf{w}) = \sum_{\mathbf{a}} P(\mathbf{c}, \mathbf{R}, \mathbf{a} \mid \mathbf{w})$$

where the joint probability can be factorized into node probabilities and edge probabilities:

$$\begin{aligned} P(\mathbf{c},\mathbf{R},\mathbf{a} \mid \mathbf{w}) &= P(\mathbf{a})P(\mathbf{c} \mid \mathbf{a},\mathbf{w})P(\mathbf{R} \mid \mathbf{a},\mathbf{w},\mathbf{c}) \\ &= P(\mathbf{a})\prod_i P(c_i \mid \mathbf{h}_{a_i})\prod_{i,j} P(r_{ij} \mid \mathbf{h}_{a_i},\mathbf{c}_i,\mathbf{h}_{a_j},\mathbf{c}_j) \end{aligned}$$

During concept generation, a probability distribution over all concepts (and NULL) is computed for each input word i, using its word embedding $\mathbf{w}_i$ and BiLSTM encoding (hidden state) $\mathbf{h}_i$:

$$\begin{aligned} P(c \mid \mathbf{h}_i,\mathbf{w}_i) &= P(\tau(c) \mid \mathbf{h}_i,\mathbf{w}_i)P(c \mid \mathbf{h}_i,\mathbf{w}_i,\tau(c)) \\ P(\tau(c) \mid \mathbf{h}_i,\mathbf{w}_i) &= \text{softmax}(\text{FFN}([\mathbf{h}_i;\mathbf{w}_i])) \\ P(c \mid \mathbf{h}_i,\mathbf{w}_i,\tau(c)) &= \frac{\delta[e_i=c]\exp(\mathbf{u}^\top\mathbf{h}_i)+\exp(\mathbf{v}_c^\top\mathbf{h}_i)}{\delta[e_i=c]\exp(\mathbf{u}^\top\mathbf{h}_i)+\sum_{c'\in\tau(c)}\exp(\mathbf{v}_{c'}^\top\mathbf{h}_i)} \end{aligned}$$

where $\tau(c)$ is a predicted concept category; e_i is a proposed candidate concept for word i (e.g. lemma given by a lemmatizer); and a concept can be generated by either picking the proposed candidate or sampling from the predicted concept category.

After nodes are generated, edges are scored using a bilinear model, where each concept argument (i.e. incident node) i is represented as the concatenation of concept embedding $\mathbf{c}_i$ and its corresponding LSTM encoding $\mathbf{h}_{a_i}$:

$$score_{ij} = (\mathbf{M}_{head}[\mathbf{h}_{a_i};\mathbf{c}_i])^\top \mathbf{C}_r \left(\mathbf{M}_{dep}[\mathbf{h}_{a_j};\mathbf{c}_j]\right)$$

Based on edges scores, the final AMR graph is constructed greedily under various constraints, e.g. certain concepts can have only one neighbor; certain predicates can only have one argument of each type; the graph must be connected, etc.

Deriving a practical objective function is somewhat tricky due to the marginalization over latent alignments. The idea is to apply a continuous relaxation to the expectation over all possible alignments, resulting in a differentiable objective that is a lower bound of the original. The new objective depends on a bilinear scoring function ψ for word-concept pairs using embedding $\mathbf{g}_i$ for concept i and BiLSTM encoding $\mathbf{h}_j$ for word j:

$$\psi(\mathbf{g}_i,\mathbf{h}_j) = \mathbf{g}_i^\top \mathbf{B}\mathbf{h}_j$$

The technical details of the relaxation method is rather mathematically involved. Interested reader may refer to Section 2 of the original paper.

5.3 Zhang et al. (2019a)

Conceptually, the model proposed by Zhang et al. (2019a) is similar to Flanigan et al. (2014), except that it uses neural models for scoring, and Chu-Liu/Edmond's algorithm instead of Kruskal's algorithm to construct spanning trees.

For concept generation, instead of using an aligner as Flanigan et al. (2014) does, Zhang et al. (2019a) utilizes an encoder-decoder architecture with a pointer-generator network (See et al., 2017), which was originally intended for text summarization. The encoder is an l-layer bidirectional LSTM, with an embedding layer that concatenates GloVe embeddings, BERT embeddings and various features vectors. The decoder is an l-layer unidirectional LSTM with an embedding layer for AMR nodes.

The purpose of pointer-generator network is to model the three possible actions to generate concepts: (i) sampling from a fixed concept vocabulary; (ii) sampling from source text and use the word (sense) as concept; (iii) sampling from output history (i.e. reusing a previously generated concept). Each newly generated node (either from concept vocabulary or source words) is assigned a new ID t, whereas copied node reuses the ID of the node it's copied from – these reused nodes will be reentrant in the output graph. At each step t, four probability distributions – vocabulary distribution P_{vocab}, source attention $\mathbf{a}^t_{enc}$, target attention $\mathbf{a}^t_{dec}$ and the switch distribution $[p_{enc}, p_{dec}, p_{gen}]$ – are computed as follows:

$$\begin{aligned}
\mathbf{a}^{enc}_t &= \text{softmax}(\text{MLP}([\mathbf{e}^l_{1:n}; \mathbf{h}^l_t])) \\
\mathbf{c}_t &= (\mathbf{a}^{enc})^\top \mathbf{e}^l \\
\tilde{\mathbf{s}}_t &= \text{FFN}([\mathbf{c}_t; \mathbf{h}^l_t]) \\
\mathbf{a}^{dec}_t &= \text{softmax}(\text{MLP}([\tilde{\mathbf{s}}_{1:t-1}, \tilde{\mathbf{s}}_t])) \\
P_{vocab} &= \text{softmax}(\text{FFN}(\tilde{\mathbf{s}}_t)) \\
[p_{enc}, p_{dec}, p_{gen}] &= \text{softmax}(\text{FFN}(\tilde{\mathbf{s}}_t))
\end{aligned}$$

The list of concepts can be generated by either a greedy search or a beam search using the mixture distribution at each step t over all possible concepts, computed as follows:

$$P(u) = p_{gen} P_{vocab}(u) + p_{enc} \sum_{w_i = u} \mathbf{a}^{enc}_i + p_{dec} \sum_{u_i = u} \mathbf{a}^{dec}_i$$

The second pass constructs a maximum spanning tree based on edge scores. For each pair of concepts (i, j), the scores for directed edge $(1, j)$ and directed edge (i, j) with label l is computed from decoder hidden states $\mathbf{h}_i$ and $\mathbf{h}_j$:

$$\begin{aligned}
score_{i,j} &= \text{BIAFFINE}(\text{MLP}^{head}(\mathbf{h}_i), \text{MLP}^{dep}(\mathbf{h}_j)) \\
score^l_{i,j} &= \text{BILINEAR}(\text{MLP}^{l_{head}}(\mathbf{h}_i), \text{MLP}^{l_{dep}}(\mathbf{h}_j))
\end{aligned}$$

where MLP stands for multilayer perceptron, and

$$\begin{aligned}
\text{BIAFFINE}(\mathbf{x}, \mathbf{y}) &= \mathbf{x}^\top \mathbf{U} \mathbf{y} + \mathbf{W}[\mathbf{x}; \mathbf{y}] + \mathbf{b} \\
\text{BILINEAR}(\mathbf{x}, \mathbf{y}) &= \mathbf{x}^\top \mathbf{U} \mathbf{y} + \mathbf{b}
\end{aligned}$$

For training, the loss function is the sum of negative log likelihoods of all the actions – i.e. generating concepts, generating directed edges and assigning labels – required to output the gold graph, as well as a penalty for repeating nodes:

$$J = -\sum_t \left[\log P(u_t) + \log P_t(u_k) + \log P_{k,t}(l) + \lambda \sum_i \min \left(\mathbf{a}^t_{src}, \sum_{t'=1}^{t-1} \mathbf{a}^{t'}_{src} \right) \right]$$

where $P_j(u_i)$, the probability that u_i is the head of u_j, and $P_{i,j}(l)$, the probability that edge (i, j) has a label l, are computed respectively with softmax:

$$\begin{aligned}
P_j(u_i) &= \frac{\exp(score_{i,j})}{\sum_{i'} \exp(score_{i',j})} \\
P_{i,j}(l) &= \frac{\exp(score^l_{i,j})}{\sum_{l'} \exp(score^{l'}_{i,j})}
\end{aligned}$$

5.4 Zhang et al. (2019b)

Zhang et al. (2019b) is an improvement upon Zhang et al. (2019a). The new model is intended as a unified semantic parser for AMR, DM and UCCA, but we will keep our focus on the AMR part in this review. As with Zhang et al. (2019a), AMR graphs are transformed into arborescences by duplicating reentrant nodes with their original IDs.

The major improvement over Zhang et al. (2019a) is in terms of efficiency: The new model no longer needs to compute a maximum spanning tree. Instead, edges are now generates on the fly, while nodes are being generated. Compared with Zhang et al. (2019a), which only uses an encoder-decoder seq2seq model for concept generation, in Zhang et al. (2019b) the entire parser becomes a single seq2seq model, where the input X is the source sentence and the output Y is a sequence of semantic relations of the form $\langle u, d^u, r, v, d^v \rangle$, with source label u, source index d^u, target label v, target index d^v and relation r. Parsing then becomes a decoding problem, i.e. finding an output sequence that maximizes output likelihood:

$$\text{argmax}_{y \in \mathscr{Y}} \prod_i P(y_i \mid y_{<i}, X)$$

Similarly to Zhang et al. (2019a), the encoder is a l-layer BiLSTM with an embedding layer that concatenates GLoVe, BERT, CharCNN embeddings and POS tags. The decoder consists of three modules that predicts the target node, source node and relation between target and source nodes, respectively, at each step t. The target node module is basically the same as in the previous model. The target index d_t^v is assigned d_j^v if the node is copied from v_j, otherwise it is assigned a new index t. The probabilities are again computed as follows:

$$\begin{aligned}
\mathbf{a}_t^{enc} &= \text{softmax}(\text{MLP}([\mathbf{e}_{1:n}^l; \mathbf{h}_t^l])) \\
\mathbf{c}_t &= (\mathbf{a}^{enc})^\top \mathbf{e}^l \\
\mathbf{z}_t &= \text{FFN}([\mathbf{h}_t^l; \mathbf{c}_t; \mathbf{r}_{t-1}; \mathbf{u}_{t-1}; \mathbf{d}_{t-1}^u]) \\
\mathbf{a}_t^{dec} &= \text{softmax}(\text{MLP}([\mathbf{z}_{1:t-1}, \mathbf{z}_t])) \\
P_{vocab} &= \text{softmax}(\text{FFN}(\mathbf{z}_t)) \\
[p_{src}, p_{tgt}, p_{gen}] &= \text{softmax}(\text{FFN}(\tilde{\mathbf{s}}_t)) \\
P(v) &= p_{gen} P_{vocab}(v) + p_{enc} \sum_{w_i = v} \mathbf{a}_i^{enc} + p_{dec} \sum_{u_i = v} \mathbf{a}_i^{dec}
\end{aligned}$$

Immediately after the generation of *each* target node (as opposed to after all nodes are generated), the same biaffine and bilinear classifiers from Zhang et al. (2019a) are used to predict a source node and a label for the directed edge from source to target. The source distribution is given by a biaffine classifier:

$$\text{softmax}(\text{BIAFFINE}(\text{MLP}^s(\mathbf{h}_t^l), \text{MLP}^e(\mathbf{h}_{1:t-1}^l)))$$

Similarly, the label distribution conditioned on source node v_s is given by a bilinear classifier:

$$\text{softmax}(\text{BILINEAR}(\text{MLP}^{src}(\mathbf{h}_s^l), \text{MLP}^{tgt}(\mathbf{h}_t^l)))$$

For decoding, either greedy search or beam search can be applied, using probabilities for source, target and relation at each step t, to recover the best sequence of semantic relations

$$\text{argmax}_{y \in \mathscr{Y}} \prod_t P(u_t) P(r_t) P(v_t)$$

Action	Description
$Swap(l_r)$	Label (β_0, σ_0) with relation l_r and insert β_0 into σ immediately after σ_0
$NextEdge(l_r)$	Label (σ_0, β_0) with relation l_r and pop β_0 from β
$Reattach(k, l_r)$	Label (k, β_0) with relation l_r and pop β_0
$Reentrance(k, l_r)$	Label (k, β_0) with relation l_r
$ReplaceHead$	Replace σ_0 and with β_0 and reset β with β_0's neighbors
$Merge$	Pop β_0 and merge it into σ_0.
$NextNode(l_c)$	Label σ_0 with l_c, pop σ_0 and reset β with σ_1's neighbors
$DeleteNode$	Pop σ_0 and reset β with σ_1's neighbors

Table 1: Wang et al. (2015)

For training, gold arborescences are linearized into gold sequences using pre-order traversal. The order is crucial because each semantic relation can only reference preceding nodes, i.e. the head of a node must be generated before the node itself. The loss function is again very similar to that of Zhang et al. (2019a):

$$J = -\sum_t \left[\log p(u_t) + \log P(r_t) + \log P(v_t) + \lambda \sum_i \min \left(\mathbf{a}^t_{src}, \sum_{t'=1}^{t-1} \mathbf{a}^{t'}_{src} \right) \right]$$

6 Transition-Based Parsing

6.1 Wang et al. (2015)

Similarly to how Artzi et al. (2015) relies on CCG parsing, Wang et al. (2015) builds AMR graphs from dependency parse trees through various actions that operate on nodes and edges. Alignment is handled by node actions that can merge word spans and replace them with concepts; reentrancy is handled by edge actions that can add new edges or flip existing edges. The full list of actions can be found in Table 1.

The parser maintains a stack σ of nodes to be processed, initialized with all nodes from the dependency parse tree, each containing a single word; a buffer β of unvisited neighbors of σ_0, which is reset every time σ is updated; and a partial graph G, initialized with the unlabeled dependency parse tree, which is eventually transformed into AMR when the parser terminates.

At each parsing step, a score is computed for each action t given the current parser state s, and the maximum-scoring action is applied to G. The score used in this paper is simply a linear function of extracted features:

$$score(t, s) = \mathbf{w}^\top \phi(t, s)$$

During training, $\mathbf{w}$ is updated at each transition by comparing the predicted action with the gold action generated from the gold graph by a deterministic oracle function; then the gold action is applied instead of the predicted one. The weights are updated by the gradient:

$$\mathbf{w} \leftarrow \mathbf{w} + \phi(t_{gold}, s) - \phi(t_{best}, s)$$

6.2 Ballesteros and Al-Onaizan (2017)

Ballesteros and Al-Onaizan (2017) builds AMR graphs from scratch without relying on dependency

Action	Description
$Shift$	$Push(\sigma, Pop(\beta))$
$Confirm$	Replace top of stack with a predicted AMR node
$Reduce$	$Pop(\sigma)$
$Merge$	Merge top two elements on the stack, i.e. $Push(\sigma, Pop(\sigma) + Pop(\sigma))$
$Entity(l_e)$	Label the top node on stack with an entity label l_e
$Dependent(l_r, n)$	Creates a node n dependent on top node on the stack
$LeftArc(l_r)$	Creates an left arc labeled l_r between top two nodes on the stack
$RightArc(l_r)$	Creates an right arc labeled l_r between top two nodes on the stack
$Swap$	$u \leftarrow Pop(\sigma); v \leftarrow Pop(\sigma); PushFront(\beta, v); Push(\sigma, u)$

Table 2: Ballesteros and Al-Onaizan (2017)

parse trees. As with all transition-based parsers, it uses a stack σ of word/concept nodes, a buffer β of words, and a partial graph G. The full list of actions can be found in Table 2.

The model is an application of the stack-LSTM (Dyer et al., 2015) architecture. Note that this is not stacked LSTM but rather LSTM with a movable "head". Specifically, at each step t, stack-LSTM uses $\mathbf{h}_{top}$ and $\mathbf{c}_{top}$ instead of $\mathbf{h}_{t-1}$ and $\mathbf{c}_{t-1}$ to compute $\mathbf{h}_t$ and $\mathbf{c}_t$, where the "top" position can be updated with either PUSH (i.e. process $\mathbf{x}_t$ and move "head" to t) or POP (i.e. move "head" one step to the left) .

The word buffer, node stack and action history are each encoded using a stack-LSTM, which at each step t computes an encoding $\mathbf{b}_t$ for β, $\mathbf{st}_t$ for σ, and $\mathbf{a}_t$ for the action history. These encodings are combined into a single vector $\mathbf{s}_t$ that encodes the entire parser state, which is then used to predict a probability distribution over all actions as well as a probability distribution over all node labels if the action is CONFIRM (see Table 2), :

$$\mathbf{s}_t = \text{ReLU}(\text{LINEAR}([\mathbf{st}_t; \mathbf{b}_t; \mathbf{a}_t]))$$
$$P(\mathbf{a} \mid \mathbf{s}_t) = \text{softmax}([\mathbf{g}_1; ...; \mathbf{g}_{|\mathscr{A}|}]^\top \mathbf{s}_t + \mathbf{h}))$$
$$P(\mathbf{n} \mid \mathbf{s}_t) = \text{softmax}([\mathbf{p}_1; ...; \mathbf{p}_{|\mathscr{N}|}]^\top \mathbf{s}_t + \mathbf{q}))$$

For training, the aligner introduced by Flanigan et al. (2014), which we described earlier, is used to map concepts to word spans in order to translate gold AMR graphs into gold action sequences.

6.3 Peng et al. (2018)

Instead of relying on greedy or beam search, the model proposed by Peng et al. (2018) generates the entire action sequence using a sequence-to-action-sequence encoder-decoder architecture with cross attention.

The parser uses a buffer β of concepts to be processed, an indexed cache η of concepts currently being processed, a stack σ to put aside cached concepts temporarily, and a partial graph G. The full list of actions can be found in Table 3.

The model uses two BiLSTM encoders, one for the word sequence, and the other for the concept sequence, which is generated using the aligner from Flanigan et al. (2016) (an improved version of Flanigan et al. (2014) that we introduced earlier).

The decoder is a unidirectional LSTM with cross attention. At each step, the model first computes a new hidden state $\mathbf{s}_t$ based on the previous hidden state $\mathbf{s}_{t-1}$, action embedding $\mathbf{e}_{t-1}$, word

Action	Description
$Shift$	No-op
$Pop(i,v)$	$(v,i) \leftarrow Pop(\sigma); Insert(\eta, i, v)$, shifting the last entry out of cache
$PushIndex(i)$	Move next concept into rightmost cache position and move (i, η_i) onto stack
$Arc(i,d,l_r)$	Make an d-arc labeled l_r between rightmost concept in cache and η_i

Table 3: Peng et al. (2018)

context μ^w_{t-1} and concept context μ^c_{t-1}, as well as features extracted from the current parser state $\mathbf{e}_f(C_t)$. Then, using cross attention for words and concepts respectively, the new word context μ^w_t and concept context μ^c_t are computed from $\mathbf{s}_t$, word encodings $\mathbf{h}^w$ and concept encodings $\mathbf{h}^c$:

$$
\begin{aligned}
\mathbf{s}_t &= \text{LSTM}(\mathbf{s}_{t-1}, [\mathbf{e}_{t-1}; \mu^w_{t-1}; \mu^c_{t-1}; \mathbf{e}_f(C_t)]) \\
\varepsilon^w_{t,i} &= \mathbf{v}_w^\top \tanh(\mathbf{W}_{w,h}\mathbf{h}^w_i + \mathbf{W}_{w,s}\mathbf{s}_t + \mathbf{b}_w) \\
\alpha^w_t &= \text{softmax}(\varepsilon^w_t) \\
\mu^w_t &= (\alpha^w_t)^\top \mathbf{h}^w \\
\varepsilon^c_{t,i} &= \mathbf{v}_c^\top \tanh(\mathbf{W}_{c,h}\mathbf{h}^c_i + \mathbf{W}_{c,s}\mathbf{s}_t + \mathbf{b}_c) \\
\alpha^c_t &= \text{softmax}(\varepsilon^c_t) \\
\mu^c_t &= (\alpha^c_t)^\top \mathbf{h}^c \\
P(a_t) &= \text{softmax}(W[\mathbf{s}_t; \mu^w_t; \mu^c_t] + \mathbf{b})
\end{aligned}
$$

The model is trained using the cross entropy loss of each gold action sequence, which is generated from the original gold AMR graph by an oracle.

6.4 Zhou et al. (2021)

Action-Pointer Transition (APT), the model proposed by Zhou et al. (2021), extends the encoder-decoder transformer model with a pointer network. The model achieve the best results among all the models we included in this survey.

Zhou et al. (2021) differs from most transition-based models in that, instead of using stack and buffer, it relies on self-attention (i.e. pointer) to form edges, and operates in-place on the source sequence using a cursor c_t (see Table 4).

Although categorized as transition-based, this model is actually quite similar to Zhang et al. (2019a) and Zhang et al. (2019b), both of which, as we described eariler, are based on pointer-generator network, which is itself an hybrid of pointer network and the basic seq2seq model with source attention. The main difference is that, in Zhang et al. (2019a) and Zhang et al. (2019b), the pointer points to a previous generated node; while in Zhou et al. (2021), the pointer points to a previous node-generating *action*. Furthermore, in Zhou et al. (2021), edges can be constructed in both directions, so the model is free from the head-before-dependent constraint that Zhang et al. (2019b) imposes.

At each step t, each decoder layer l computes a hidden state $\mathbf{d}^l_t$ with a self-attention layer SA^l, a cross-attention layer CA^l, and a feed-forward layer FF^l, i.e.

$$
\mathbf{d}^l_t = FF^l(CA^l(SA^l(\mathbf{d}^{l-1}_t, \mathbf{d}^{l-1}_{0:t}), \mathbf{e}))
$$

Action	Description
$Shift$	Increment c_t
$Reduce$	Increment c_t
$Merge$	Merge x_{c_t} into x_{c_t+1} and increment c_t
$CopyLemma$	Construct a node from the lemma of x_{c_t}
$CopySense01$	Construct a node from the first sense (frame) of the lemma of x_{c_t}
$Predict(l_c)$	Construct a non-root node with label l_c.
$Subgraph(l_c)$	Construct a subgraph root with label l_c.
$LeftArc(a_{root}, l_r)$	Creates an arc labeled l_r from current node to *root* generated by action a_{root}
$RightArc(a_{root}, l_r)$	Creates an arc labeled l_r from *root* generated by action a_{root} to current node

Table 4: Zhou et al. (2021)

The pointer network is built directly from the last self-attention layer SA^l. The action distribution is computed from the hidden state $\mathbf{d}_t^l$:

$$P(a_t \mid \mathbf{y}_{<t}, \mathbf{x}) = softmax(\mathbf{W}\mathbf{d}_t^l)$$

and the edge pointer distribution is simply the masked self-attention from (one head of) SA^l:

$$P(p_t \mid \mathbf{y}_{<t}, \mathbf{x}) = softmax\left(\left(\mathbf{W}_K^l \mathbf{d}_{0:t}^l\right)^\top \mathbf{W}_Q^l \mathbf{d}_t^l\right)$$

Based on action and edge distributions computed at each step t, the output sequence is obtained by a beam search that seeks to maximize the output probability $\prod_t P(y_t \mid \mathbf{x})$. The beam search algorithm is listed in Algorithm 3.

Algorithm 3 Beam Search (Zhou et al., 2021)

$q \leftarrow max_{p_t} P(p_t \mid \mathbf{y}_{<t}, \mathbf{x})$
$p \leftarrow argmax_{p_t} P(p_t \mid \mathbf{y}_{<t}, \mathbf{x})$
for each history $\mathbf{y}_{<t}$ and action a_t **do**
    if a_t is an edge-generating action **then**
        $y_t \leftarrow (a_t, p)$
        $P(y_t \mid \mathbf{y}_{<t}, \mathbf{x}) \leftarrow P(a_t \mid \mathbf{y}_{<t}) \cdot q$
    else
        $y_t \leftarrow (a_t, null)$
        $P(y_t \mid \mathbf{y}_{<t}, \mathbf{x}) \leftarrow P(a_t \mid \mathbf{y}_{<t})$
    end if
    $P(\mathbf{y}_{\leq t}) \leftarrow P(\mathbf{y}_{<t}) \cdot P(y_t \mid \mathbf{y}_{<t})$
end for
Select best k actions based on $P(\mathbf{y}_{\leq t})$

7 Conclusion

In this paper, we briefly reviewed grammar-based, graph-based and transition-based AMR parsers. For each model discussed above, we list in Table 5 its Smatch score for each corpus it has been tested

	LDC2013 E117	LDC2014 T12	LDC2015 E86	LDC2017 T10	LDC2020 T02
Flanigan et al. (2014)	.58	-	-	-	
Wang et al. (2015)	.63	-	-	-	-
Artzi et al. (2015)	-	.66	-	-	-
Ballesteros and Al-Onaizan (2017)	-	.63	-	-	-
Peng et al. (2018) (soft attention)	-	-	.65	-	-
Peng et al. (2018) (hard attentino)	-	-	.66	-	-
Lyu and Titov (2018)	-	-	.74	-	-
Zhang et al. (2019a)	-	.70	-	.76	-
Zhang et al. (2019b)	-	.71	-	.77	-
Zhou et al. (2021)	-	.80	-	.83	.81

Table 5: Comparison of Smatch scores

on. From the data, it is obvious that both graph-based and transition-based models have benefited from the deep learning renaissance in recent years – by incorporating word embeddings, RNNs, the encoder-decoder architecture and the attention mechanism, these neural models have achieved increasingly better results. In particular, attention-based neural models, whether transition-based (e.g. Zhou et al. (2021)) or graph-based (e.g. Zhang et al. (2019a) and Zhang et al. (2019b), significantly outperform feature-based models and previous neural models, which clearly demonstrates the broad applicability and outstanding effectiveness of the attention mechanism.

References

Yoav Artzi, Kenton Lee, and Luke Zettlemoyer. 2015. Broad-coverage CCG semantic parsing with AMR. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1699–1710, Lisbon, Portugal. Association for Computational Linguistics.

Miguel Ballesteros and Yaser Al-Onaizan. 2017. AMR parsing using stack-LSTMs. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1269–1275, Copenhagen, Denmark. Association for Computational Linguistics.

Shu Cai and Kevin Knight. 2013. 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, Sofia, Bulgaria. Association for Computational Linguistics.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. 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, Beijing, China. Association for Computational Linguistics.

Jeffrey Flanigan, Chris Dyer, Noah A. Smith, and Jaime Carbonell. 2016. CMU at SemEval-2016 task 8: Graph-based AMR parsing with infinite ramp loss. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1202–1206, San Diego, California. Association for Computational Linguistics.

Jeffrey Flanigan, Sam Thomson, Jaime Carbonell, Chris Dyer, and Noah A. Smith. 2014. 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, Baltimore, Maryland. Association for Computational Linguistics.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017. 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, Vancouver, Canada. Association for Computational Linguistics.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2011. Lexical generalization in CCG grammar induction for semantic parsing. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1512–1523, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D. Ernst. 2018. NL2Bash: A corpus and semantic parser for natural language interface to the linux operating system. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Chunchuan Lyu and Ivan Titov. 2018. 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, Melbourne, Australia. Association for Computational Linguistics.

Xiaochang Peng, Linfeng Song, Daniel Gildea, and Giorgio Satta. 2018. 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, Melbourne, Australia. Association for Computational Linguistics.

Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1139–1149, Vancouver, Canada. Association for Computational Linguistics.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015. A transition-based algorithm for AMR parsing. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 366–375, Denver, Colorado. Association for Computational Linguistics.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019a. AMR parsing as sequence-to-graph transduction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 80–94, Florence, Italy. Association for Computational Linguistics.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019b. Broad-coverage semantic parsing as transduction. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3786–3798, Hong Kong, China. Association for Computational Linguistics.

Jiawei Zhou, Tahira Naseem, Ramón Fernandez Astudillo, and Radu Florian. 2021. AMR parsing with action-pointer transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5585–5598, Online. Association for Computational Linguistics.