AMR Parsing with Action-Pointer Transformer

Jiawei Zhou[†] **Tahira Naseem**[*] **Ramón Fernandez Astudillo**[*] **Radu Florian**[*]
[†]Harvard University [*]IBM Research
[†]jzhou02@g.harvard.edu [*]{tnaseem,raduf}@us.ibm.com [*]ramon.astudillo@ibm.com

Abstract

Abstract Meaning Representation parsing is a sentence-to-graph prediction task where target nodes are not explicitly aligned to sentence tokens. However, since graph nodes are semantically based on one or more sentence tokens, implicit alignments can be derived. Transition-based parsers operate over the sentence from left to right, capturing this inductive bias via alignments at the cost of limited expressiveness. In this work, we propose a transition-based system that combines hard-attention over sentences with a target-side action pointer mechanism to decouple source tokens from node representations and address alignments. We model the transitions as well as the pointer mechanism through straightforward modifications within a single Transformer architecture. Parser state and graph structure information are efficiently encoded using attention heads. We show that our action-pointer approach leads to increased expressiveness and attains large gains (+1.6 points) against the best transition-based AMR parser in very similar conditions. While using no graph re-categorization, our single model yields the second best SMATCH score on AMR 2.0 (81.8), which is further improved to 83.4 with silver data and ensemble decoding.

1 Introduction

Abstract Meaning Representation (AMR) (Banarescu et al., 2013) is a sentence level semantic formalism encoding *who does what to whom* in the form of a rooted directed acyclic graph. Nodes represent concepts such as entities or predicates which are not explicitly aligned to words, and edges represent relations such as subject/object (see Figure 1).

AMR parsing, the task of generating the graph from a sentence, is nowadays tackled with sequence to sequence models parametrized with neural networks. There are two broad categories of methods that are highly effective in recent years. Transition-based approaches predict a sequence of actions given the sentence. These actions generate the graph while processing tokens left-to-right through the sentence and store intermediate representations in memories such as stack and buffer (Wang et al., 2015; Damonte et al., 2016; Ballesteros and Al-Onaizan, 2017; Vilares and Gómez-Rodríguez, 2018; Naseem et al., 2019; Astudillo et al., 2020; Lee et al., 2020). General graph-based approaches, on the other hand, directly predict nodes and edges in sequential order from graph traversals such as breath first search or depth first search (Zhang et al., 2019a,b; Cai and Lam, 2019, 2020). While not modeling the local semantic correspondence between graph nodes and source tokens, the approaches achieve strong results without restrictions of transition-based approaches, but often require graph re-categorization, a form of graph normalization, for optimal performance.

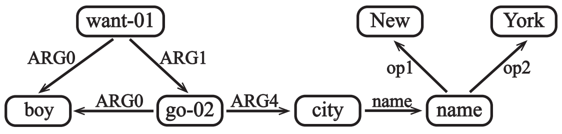


Figure 1: AMR graph expressing the meaning of the sentence *The boy wants to go to New York.*

The strong left-to-right constraint of transition-based parsers provides a form of inductive bias that fits AMR characteristics. AMR nodes are very often normalized versions of sentence tokens and locality between words and nodes is frequently preserved. The fact that transition-based systems for AMR have alignments as the core of their explanatory model also guarantees that they produce reliable alignments at decoding time, which are useful for applications utilizing AMR parses. Despite these advantages, transition-based systems still suffer in situations when multiple nodes are best explained as aligned to one sentence token or none. Furthermore, long distance edges in AMR, e.g. re-entrancies, require excessive use of SWAP or

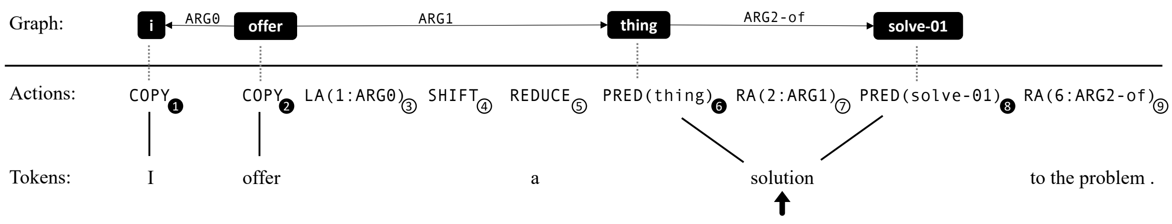


Figure 2: Source tokens, target actions and AMR graph for the sentence *I offer a solution to the problem* (partially parsed). The black arrow marks the current token cursor position. The circles contain the action indices (used as ids), black circles indicate node creating actions. Only these actions are available for edge attachments. Notice that the edge actions (at steps 3, 7 and 9) explicitly refer to past nodes using the id of the action that created the node. The other participant of the edge action is implicitly assumed to be the most recently created graph node.

equivalent actions, leading to very long action sequences. This in turn affects both a model's ability to learn and its decoding speed.

In this work, we propose the Action-Pointer Transition (APT) system which combines the advantages of both the transition-based approaches and more general graph-generation approaches. We focus on predicting an action sequence that can build the graph from a source sentence. The core idea is to put the target action sequence to a dual use – as a mechanism for graph generation as well as the representation of the graph itself. Inspired by recent progress in pointer-based parsers (Ma et al., 2018a; Fernández-González and Gómez-Rodríguez, 2020), we replace the stack and buffer by a cursor that moves from left to right and introduce a pointer network (Vinyals et al., 2015) as mechanism for edge creation. Unlike previous works, we use the pointer mechanism on the target side, pointing to past node generation actions to create edges. This eliminates the node generation and attachment restrictions of previous transition-based parsers. It is also more natural for graph generation, essentially resembling the generation process in the graph-based approaches, but keeping the graph and source aligned.

We model both the action generation and the pointer prediction with a single Transformer model (Vaswani et al., 2017). We relate target node and source token representations through masking of cross-attention mechanism, similar to Astudillo et al. (2020) but simply with monotonic action-source alignment driven by cursor positions, rather than stack and buffer contents. Finally we also embed the AMR graph structural information in the target decoder by re-purposing edge-creating steps, and propose a novel step-wise incremental graph message passing method (Gilmer et al., 2017) enabled by the decoder self-attention mechanism.

Experiments on AMR 1.0, AMR 2.0, and AMR 3.0 benchmark datasets show the effectiveness of our APT system. We outperform the best transition-based systems while using sensibly shorter action sequences, and achieve better performance than all previous approaches with similar size of training parameters.

2 AMR Generation with Action-Pointer

Figure 2 shows a partially parsed example of a source sentence, a transition action sequence and the AMR graph for the proposed transitions. Given a source sentence $\mathbf{x} = x_1, x_2, \ldots, x_S$, our transition system works by scanning the sentence from left to right using a cursor $c_t \in \{1, 2, \ldots, S\}$. Cursor movement is controlled by three actions:

SHIFT moves cursor one position to the right, such that $c_{t+1} = c_t + 1$.

REDUCE is a special SHIFT indicating that no action was performed at current cursor position.

MERGE merges tokens x_{c_t} and x_{c_t+1} and SHIFTs. Merged tokens act as a single token under the position of the last token merged.

At cursor position c_t we can generate any subgraph through following actions:

COPY creates a node by copying the word under x_{c_t}. Since AMR nodes are often lemmas or propbank frames, two versions of this action exist to copy the lemma of x_{c_t} or provide the first sense (frame -01) constructed from the lemma. This covers a large portion of the total AMR nodes. It also helps generalize for predictions of unseen nodes. We use an external lemmatizer[1] for this action.

[1]https://spacy.io/.

PRED(LABEL) creates a node with name LABEL from the node names seen at train time.

SUBGRAPH(LABEL) produces an entire subgraph indexed by label LABEL. Any future attachments can only be made to the root of the subgraph.

LA(ID,LABEL) creates an arc with LABEL from last generated node to a previous node at position ID. Note that we can only point to past node generating actions in the action history.

RA(ID,LABEL) creates an arc with LABEL to last generated node from a previous node at position ID.

Using the above actions, it is easy to derive an oracle action sequence given gold-graph information and initial word to node alignments. For current cursor position, all the nodes aligned to it are generated using SUBGRAPH(), COPY or PRED() actions. Each node prediction action is followed by edge creation actions. Edges connecting to closer nodes are generated before the farther ones. When multiple connected nodes are aligned to one token, they are traversed in pre-order for node generation. A detailed description of oracle algorithm is given in Appendix B.

The use of a cursor variable c_t decouples node reference from source tokens, allowing to produce multiple nodes and edges (see Figure 3), even the entire AMR graph if necessary, from a single token. This provides more expressiveness and flexibility than previous transition-based AMR parsers, while keeping a strong inductive bias. The only restriction is that all inbound or outbound edges between current node and all previously produced nodes need to be generated before predicting a new node or shifting the cursor. This does not limit the oracle coverage, however, for trained parsers, it leads to a small percentage of disconnected graphs in decoding. Furthermore, nodes within the SUBGRAPH() action can not be reached for edge creation. The use of SUBGRAPH() action, initially introduced in Ballesteros and Al-Onaizan (2017), is reduced in this work to cases where no such edges are expected, which is mainly the case for dates and named-entities.

Compared to previous oracles (Ballesteros and Al-Onaizan, 2017; Naseem et al., 2019; Astudillo et al., 2020), the action-pointer does not use a SWAP action. It can establish an edge between the last predicted node and *any* previous node, since edges are created by pointing to decoder representations.

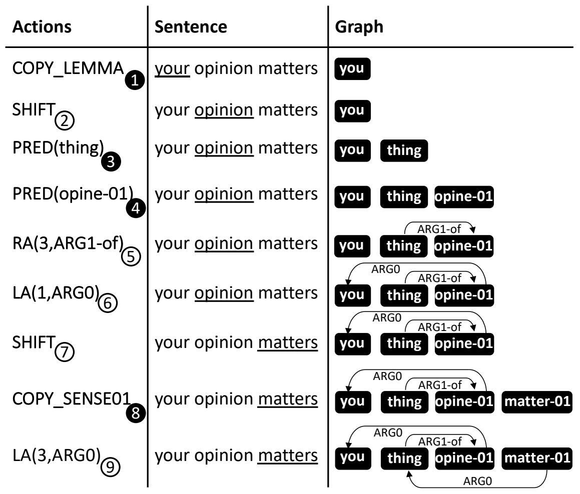


Figure 3: Step-by-step actions on the sentence *your opinion matters*. Creates subgraph from a single word (thing :ARG1-of opine-01) and allows attachment to all its nodes. Cursor is at underlined words (post-action).

This oracle is expected to work with generic AMR aligners. For this work, we use the alignments generation method of Astudillo et al. (2020), which generates many-to-many alignments. It is a combination of Expectation Maximization based alignments of Pourdamghani et al. (2014) and rule base alignments of Flanigan et al. (2014). Any remaining unaligned nodes are aligned based on their graph proximity to unaligned tokens. For more details, we refer the reader to the works of Astudillo et al. (2020) and Naseem et al. (2019).

3 Action-Pointer Transformer

3.1 Basic Architecture

The backbone of our model is the encoder-decoder Transformer (Vaswani et al., 2017), combined with a pointer network (Vinyals et al., 2015). The probability of an action sequence $\mathbf{y} = y_1, y_2, \ldots, y_T$ for input tokens $\mathbf{x} = x_1, x_2, \ldots, x_S$ is given in our model by

$$\begin{aligned} \mathbf{P}(\mathbf{y} \mid \mathbf{x}) &= \prod_{t=1}^{T} \mathbf{P}(y_t \mid \mathbf{y}_{<t}, \mathbf{x}) \\ &= \prod_{t=1}^{T} \mathbf{P}(a_t \mid \mathbf{a}_{<t}, \mathbf{p}_{<t}, \mathbf{x})\, \mathbf{P}(p_t \mid \mathbf{a}_{\leq t}, \mathbf{p}_{<t}, \mathbf{x}) \end{aligned} \quad (1)$$

where at each time step t, we decompose the target action y_t into the pointer-removed action and the pointer value with $y_t = (a_t, p_t)$. A dummy pointer $p_t = \text{null}$ is fixed for non-edge actions, so that

$$\mathbf{P}(p_t \mid \mathbf{a}_{\leq t}, \mathbf{p}_{<t}, \mathbf{x}) = [\mathbf{P}(p_t \mid \mathbf{a}_{<t}, \mathbf{p}_{<t}, \mathbf{x})]^{\gamma(a_t)}$$

where $\gamma(a_t)$ is an indicator variable set to 0 if a_t is not an edge action and 1 otherwise.

Given a sequence to sequence Transformer model with N encoder layers and M decoder layers, each decoder layer is defined by

$$\mathbf{d}_t^m = \mathrm{FF}^m(\mathrm{CA}^m(\mathrm{SA}^m(\mathbf{d}_t^{m-1}, \mathbf{d}_{\leq t}^{m-1}), \mathbf{e}^N))$$

where $\mathrm{FF}^m()$, $\mathrm{CA}^m()$ and $\mathrm{SA}^m()$ are feed-forward, multi-head cross-attention and multi-head self-attention components respectively[2]. $\mathbf{e}^N$ is the output of last encoder layer and $\mathbf{d}^{m-1}$ is the output of the previous decoder layer, with $\mathbf{d}_{\leq t}^0$ initialized to be the embeddings of the action history $\mathbf{y}_{<t}$ concatenated with a special start symbol.

The distribution over actions is given by

$$\mathbf{P}(a_t \mid \mathbf{a}_{<t}, \mathbf{p}_{<t}, \mathbf{x}) = \mathrm{softmax}\left(\mathbf{W} \cdot \mathbf{d}_t^M\right)_{a_t}$$

where $\mathbf{W}$ are the output vocabulary embeddings, and the edge pointer distribution is given by

$$\mathbf{P}(p_t \mid \mathbf{a}_{<t}, \mathbf{p}_{<t}, \mathbf{x}) = \\ \mathrm{softmax}\big((\mathbf{K}^M \cdot \mathbf{d}_{\leq t}^{M-1})^T \cdot \mathbf{Q}^M \cdot \mathbf{d}_t^{M-1}\big)_{p_t}$$

where $\mathbf{K}^M$, $\mathbf{Q}^M$ are key and query matrices of 1 head of the last decoder self-attention layer $\mathrm{SA}^M()$. The top layer self-attention is a natural choice for the pointer network, since it is likely to have high values for the nodes involved in the edge direction and label prediction. Although the edge action and its pointing value are both output at the same step, the specialized pointer head is also part of the overall self-attention mechanism used to compute the model's hidden representations, thus making actions distribution aware of the pointer distribution.

Our transition system moves the cursor c_t over the source from left to right during parsing, essentially maintaining a monotonic alignment between target actions and source tokens. We encode the alignment c_t with hard attentions in cross-attention heads $\mathrm{CA}^m()$ with $m = 1 \cdots M$ at every decoder layer. We mask one head of the cross-attention to see only the aligned source token at c_t, and augment it with another head masked to see only positions $> c_t$. This is similar to the hard attention in Peng et al. (2018) and parser state encoding in Astudillo et al. (2020).

As in prior works, we restrict the output space of our model to only allow valid actions given $\mathbf{x}, \mathbf{y}_{<t}$. The restriction is not only enforced at inference, but is also internalized with the model during training so that the model can always focus on relevant action subsets when making predictions.

3.2 Incremental Graph Embedding

Incrementally generated graphs are usually modeled via graph neural networks (Li et al., 2018), where a node's representation is updated from the collection of it's neighboring nodes' representations by message passing (Gilmer et al., 2017). However, this requires re-computation of all node representations every time the graph is modified, which is expensive, prohibiting its use in previous graph-based AMR parsing works (Cai and Lam, 2020). To better utilize the intermediate topological graph information without losing the efficient parallelization of Transformer, we propose to use the edge creation actions as updated views of each node, that encode this node's neighboring subgraph. This does not change the past computations and can be done by altering the hard masking of the self-attention heads of decoder layers $\mathrm{SA}^m()$. By interpreting the decoder layers as implementing message passing vertically, we can fully encode graphs up to depth M.

Given a node generating action $a_t = v$, it is followed by $k \geq 0$ edge generating actions $a_{t+1}, a_{t+2}, \ldots, a_{t+k}$ that connect the current node with previous nodes, pointed by $p_{t+1}, p_{t+2}, \ldots, p_{t+k}$ positions on the target side. This also defines k graph modifications, expanding the graph neighborhood on the current node. Figure 4 shows an example for the sentence *The boy wants to go*, with node prediction actions at positions $t = 2, 4, 8$, with k being 0, 1, 2, respectively. We use the steps from t to $t + k$ in the Transformer decoder to encode this expanding neighborhood. In particular, we fix the decoder input as the current node action v for these steps, as illustrated in the input actions in Figure 4. At each intermediate step $\tau \in [t, t + k]$, 2 decoder self-attention heads $\mathrm{SA}^m()$ are restricted to only attend to the direct graph neighbors of the current node, represented by previous nodes at positions $p_t, p_{t+1}, \cdots, p_\tau$ as well as the current position τ. This essentially builds sub-sequences of node representations with richer graph information step by step, and we use the last reference of the same node for pointing positions when generating new edges. Moreover, when propagating this masking pattern along m layers, each node encodes its m-hop neighborhood information.

[2]Each of these are wrapped around with residual, dropout and layer normalization operations removed for simplicity.

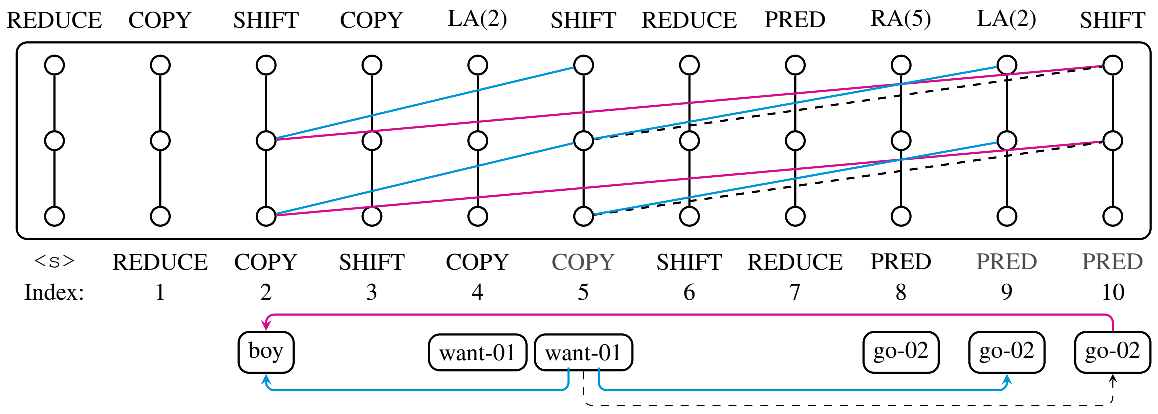


Figure 4: Encoding graph with 2 decoder layers for the sentence *The boy wants to go*. From top to bottom: target output action sequence, masked decoder self-attention, input action history and partial graph. Edge-creating action steps in the action history are used to hold updated node representations. Action labels and edge direction treatment are removed for clarity.

This defines a message passing procedure as shown in Figure 4, encoding the compositional relations between nodes. Since the edges have directions indicated by LA and RA, we also encode the direction information by separating the two heads with each only considering one direction.

4 Training and Inference

Our model is trained by maximizing the log likelihood of Equation (1). The valid action space, action-source alignment c_t, and the graph embedding mask at each step t are pre-calculated at training time. For inference, we modify the beam search algorithm to jointly search for actions and edge pointers and combine them to find the action sequence that maximizes Equation (1). We also consider hard constraints in the searching process such as valid output actions and valid target pointing values at different steps to ensure an AMR graph is recoverable. For the structural information that is extracted from the parsing state such as c_t and graph embedding masks, we compute them on the fly at each new step of decoding based on the current results, which are then used by the model for the next step decoding. We detail our search algorithm in Appendix C.

5 Experimental Setup

Data and Evaluation We test our approach on two widely used AMR parsing benchmark datasets: AMR 2.0 (LDC2017T10) and AMR 1.0 (LDC2014T12). The AMR graphs are all human annotated. The two datasets have 36521 and 10312 training AMRs, respectively, and share 1368 development AMRs and 1371 testing AMRs[3]. We also report results on the latest AMR 3.0 (LDC2020T02) dataset, which is larger in size but has not been fully explored, with 55635 training AMRs and 1722 and 1898 AMRs for development and testing set. Wiki links are removed in the preprocessing of data, and we run a wikification approach in post-processing to recover Wikipedia entries in the AMR graphs as in Naseem et al. (2019).

For evaluation, we use the SMATCH (F1) scores[4] (Cai and Knight, 2013) and further the fine-grained evaluation metrics (Damonte et al., 2016) to assess the model's AMR parsing performance.

Model Configuration Our *base* setup has 6 layers and 4 attention heads for both the Transformer encoder and decoder, with model size 256 and feed-forward size 512. We also compare with a *small* model with 3 layers in encoder and decoder but identical otherwise. The pointer network is always tied with one target self-attention head of the top decoder layer. We use the cross-attention of all decoder layers for action-source alignment. For graph embedding, we use 2 heads of the bottom 3 layers for the base model and bottom 2 layers for the small model. We use contextualized embeddings extracted from the pre-trained RoBERTa (Liu et al., 2019) large model for the source sentence, with average of all layer states and BPE tokens mapped to words by averaging as in (Lee et al., 2020). The pre-trained embeddings are fixed. For

[3]Although there are annotation revisions from AMR 1.0 to AMR 2.0. Link to data: https://amr.isi.edu/download.html.

[4]There are small variations of SMATCH computation due to the stochastic nature of graph matching algorithm.

Transition system	Avg. #actions	Oracle SMATCH
Naseem et al. (2019)*	73.6	93.3
Astudillo et al. (2020)*	76.2	98.0
Ours	41.6	98.9

Table 1: Average number of actions and oracle SMATCH on AMR 2.0 training data. The average source length is 18.9. * from author correspondence.

target actions we train our own embeddings along with the model.

Implementation Details We use the Adam optimizer with β_1 of 0.9 and β_2 of 0.98 for training. Each data batch has 3584 maximum number of tokens, and the learning rate schedule is the same as Vaswani et al. (2017), where we use the maximum learning rate of $5e{-}4$ with 4000 warm-up steps. We use a dropout rate of 0.3 and label smoothing rate of 0.01. We train all the models for a maximum number of 120 epochs, and average the best 5 epoch checkpoints among the last 40 checkpoints based on the SMATCH scores on the development data with greedy decoding. We use a default beam size of 10 for decoding. We implement our model[5] with the FAIRSEQ toolkit (Ott et al., 2019). All models are trained and tested on a single Nvidia Titan RTX GPU. Training takes about 10 hours on AMR 2.0 and 3.5 hours on AMR 1.0.

6 Results and Analysis

6.1 Main Results

Oracle Actions Table 1 compares the oracle data SMATCH and average action sequence length on the AMR 2.0 training set among recent transition systems. Our approach yields much shorter action sequences due to the target-side pointing mechanism. It has also the best coverage on training AMR graphs, due to the flexibility of our transitions that can capture the majority of graph components. We chose not to tackle a number of small corner cases, such as disconnected subgraphs for a token, that account for the missing oracle performance.

Parsing Performance We compare our action-pointer transition/Transformer (APT) model with existing approaches in Table 2[6]. We indicate the use of pre-trained BERT or RoBERTa embeddings

Corpus	Model	SMATCH (%)
AMR 1.0	Pust et al. (2015)	67.1
	Flanigan et al. (2016)	66.0
	Wang and Xue (2017)^G	68.1
	Guo and Lu (2018)^G	68.3 ±0.4
	Zhang et al. (2019a)^{B,G}	70.2 ±0.1
	Zhang et al. (2019b)^{B,G}	71.3 ±0.1
	Cai and Lam (2020)^B	74.0
	Cai and Lam (2020)^{B,G}	75.4
	Astudillo et al. (2020)$^{*\,R}$	**76.9** ±0.1
	Lee et al. (2020)^R (85K silver)	**78.2** ±0.1
	APT small^R	78.2 / 78.2 ±0.0
	APT base^R	**78.5 / 78.3** ±0.1
	APT small^R p.e.	79.7
	APT base^R p.e.	**79.8**
AMR 2.0	Van Noord and Bos (2017)	71.0
	Groschwitz et al. (2018)^G	71.0
	Lyu and Titov (2018)^G	74.4 ±0.2
	Cai and Lam (2019)	73.2
	Lindemann et al. (2019)	75.3 ±0.1
	Naseem et al. (2019)^B	75.5
	Zhang et al. (2019a)^{B,G}	76.3 ±0.1
	Zhang et al. (2019b)^{B,G}	77.0 ±0.1
	Cai and Lam (2020)^B	78.7
	Cai and Lam (2020)^{B,G}	80.2
	Astudillo et al. (2020)$^{*\,R}$	80.2 ±0.0
	Bevilacqua et al. (2021)†	**83.8**
	Xu et al. (2020) (4M silver)	80.2
	Lee et al. (2020)^R (85K silver)	81.3 ±0.0
	Bevilacqua et al. (2021)† (200K silver)	**84.3**
	APT small^R	81.7 / 81.5 ±0.2
	APT base^R	**81.8 / 81.7** ±0.1
	APT small^R p.e.	82.5
	APT base^R p.e.	82.8
	APT base^R (70K Silver)	82.8 / 82.6 ±0.1
	APT base^R (70K Silver) p.e.	**83.4**
AMR 3.0	Lyu et al. (2020)	75.8
	Bevilacqua et al. (2021)†	**83.0**
	APT base^R	80.4 / 80.3 ±0.1
	APT base^R p.e.	**81.2**

Table 2: SMATCH scores on AMR 1.0, 2.0, and 3.0 test sets. APT is our model. ^{B} or ^{R} indicates pre-trained BERT or RoBERTa embeddings, ^{G} use of graph re-categorization, * improved results reported in Lee et al. (2020). † denotes concurrent work based on fine-tuning pre-trained BART large models. We report the best/average score ± standard deviation over 3 seeds. p.e. is partial ensemble decoding with 3 seed models.

(from large models) with ^{B} or ^{R}, and graph re-categorization with ^{G}. Graph re-categorization (Lyu and Titov, 2018; Zhang et al., 2019a; Cai and Lam, 2020; Bevilacqua et al., 2021) removes node senses and groups certain nodes together such as named entities in pre-processing. It reverts these back in post-processing with the help of a name entity recognizer. We report results over 3 runs for each model with different random seeds. Given that we use fixed pre-trained embeddings, it becomes computationally cheap to build a partial ensemble

[5]Available under `https://github.com/IBM/transition-amr-parser`.

[6]We exclude Xu et al. (2020) AMR 1.0 numbers since they report 16833 train sentences, not 10312.

Model	Fixed Extra Features	Trained Param.	SMATCH AMR 2.0
Zhang et al. (2019a)^{B,G}	BERT	66.1M*	76.3
Cai and Lam (2020)^{B}	BERT	27.1M	78.7
Cai and Lam (2020)^{B,G}	BERT	26.1M	80.2
Astudillo et al. (2020)^{R}	RoBERTa	21.7M	80.2
Xu et al. (2020) (4M silver)	-	239.1M	80.2
Bevilacqua et al. (2021)†	-	411.8M	83.8
APT small^{R}	RoBERTa	17.5M	81.7
APT base^{R}	RoBERTa	21.4M	81.8
APT small^{R} p.e.	RoBERTa	52.5M	82.5
APT base^{R} p.e.	RoBERTa	64.3M	82.8

Table 3: Comparison of model parametrization sizes and SMATCH scores on AMR 2.0 test set. Model sizes of previous works are obtained from their officially released pre-trained models. * is an estimate by removing BERT parameters in the released model, where a BERT base model is trained together which is different from the paper description. † denotes concurrent work based on fine-tuning pre-trained BART large models.

that uses the average probability of 3 models from different seeds which we denote as p.e.

With the exception of the recent BART-based model Bevilacqua et al. (2021), we outperform all previously published approaches, both with our small and base models. Our best single-model parsing scores are 81.8 on AMR 2.0 and 78.5 on AMR 1.0, which improves 1.6 points over the previous best model trained only with gold data. Our small model only trails the base model by a small margin and we achieve high performance on small AMR 1.0 dataset, indicating that our approach benefits from having good inductive bias towards the problem so that the learning is efficient. More remarkably, we even surpass the scores reported in Lee et al. (2020) combining various self-learning techniques and utilizing 85K extra sentences for self-annotation (silver data). For the most recent AMR 3.0 dataset, we report our results for future reference.

Additionally, the partial ensemble decoding proves to be simple and effective in boosting the model performance, which consistently brings more than 1 point gain for AMR 1.0 and 2.0. It should be noted that the ensemble decoding is only 20% slower than a single model.

We thus use this ensemble to annotate the 85K sentence set used in (Lee et al., 2020). After removing parses with detached nodes we obtained 70K model-annotated silver data sentences. Adding these for training regularly, we achieve our best score of 83.4 with ensemble on AMR 2.0.

Model Size In Table 3, we compare parameter sizes of recently published models alongside their parsing performances on AMR 2.0. Similar to our approach, most models use large pre-trained models to extract contextualized embeddings as fixed features, with the exception of Xu et al. (2020), which is a seq-to-seq pre-training approach on large amount of data, and Bevilacqua et al. (2021), which directly fine-tunes a seq-to-seq BART large (Lewis et al., 2019) model.[7] Except the large BART model, our APT small (3 layers) has the least number of trained parameters yet already surpasses all the previous models. This justifies our method is highly efficient in learning for AMR parsing. Moreover, with the small parameter size, the partial ensemble is an appealing way to improve parsing quality with minor decoding overhead. Although more performant, direct fine-tuning of pre-trained seq-to-seq models such as BART would require prohibitively large numbers to perform an ensemble.

Fine-grained Results Table 4 shows the fine-grained AMR 2.0 evaluation (Damonte et al., 2016) of APT and previous models with comparable trainable parameter sizes. Our model achieves the best scores among all sub-tasks except negations and wikification, handled by post-processing on the best performing approach. We obtain large improvement on edge related sub-tasks including SRL (ARG arcs) and Reentrancies, proving the effectiveness of our target-side pointer mechanism.

6.2 Analysis

Ablation of Model Components We evaluate the contribution of different components in our model in Table 5. The top part of the table shows effects of 2 major components that utilize parser state information and the graph structural information in the Transformer decoder. The baseline model is a free Transformer model with pointers (row 1), which is greatly increased by including the monotonic action-source alignment via hard attention (row 2) on both AMR 1.0 and AMR 2.0 corpus, and combining it with the graph embedding (row 3) gives further improvements of 0.3 and 0.2 for AMR 1.0 and AMR 2.0. This highlights that injecting hard encoded structural information in the Transformer decoder greatly helps our problem.

[7] Here we focus on trainable parameters for learning efficiency. For deployment the total number of parameters should be considered, where all the models relying on BERT/RoBERTa features would be on the similar level.

Model	SMATCH	Unlabeled	No WSD	Concepts	Named Ent.	Negations	Wikification	Reentrancies	SRL
Van Noord and Bos (2017)	71.0	74	72	82	79	62	65	52	66
Groschwitz et al. (2018)^{G}	71.0	74	72	84	78	57	71	49	64
Lyu and Titov (2018)^{G}	74.4	77.1	75.5	85.9	86.0	58.4	75.7	52.3	69.8
Cai and Lam (2019)	73.2	77.0	74.2	84.4	82.0	62.9	73.2	55.3	66.7
Naseem et al. (2019)^{B}	75.5	80	76	86	83	67	80	56	72
Zhang et al. (2019a)^{B,G}	76.3	79.0	76.8	84.8	77.9	75.2	85.8	60.0	69.7
Zhang et al. (2019b)^{B,G}	77.0	80	78	86	79	77	86	61	71
Cai and Lam (2020)^{B,G}	80.2	82.8	80.8	88.1	81.1	**78.9**	**86.3**	64.6	74.2
Astudillo et al. (2020)$^{* R}$	80.2	84.2	80.7	88.1	87.5	64.5	78.8	70.3	78.2
APT small^{R}	81.7	85.4	82.2	**88.9**	**88.9**	67.5	78.7	70.6	80.7
APT base^{R}	**81.8**	**85.5**	**82.3**	88.7	88.5	69.7	78.8	**71.1**	**80.8**

Table 4: Fine-grained F1 scores on the AMR 2.0 test set. $^{B}/^{R}$ and ^{G} marks uses of pre-trained BERT/RoBERTa embeddings and graph re-categorization processing. * We cite improved results reported in Lee et al. (2020). We report results with our single best model for fair comparison.

Model Configuration		SMATCH (%)	
Mono. Alignment	Graph embedding	AMR 1.0	AMR 2.0
		72.2 ±0.4	77.5 ±0.2
✓		78.0 ±0.1	81.5 ±0.1
✓	✓	78.3 ±0.1	81.7 ±0.1
No subspace restriction		78.0 ±0.1	80.9 ±0.1
RoBERTa base embeddings		78.0 ±0.1	81.3 ±0.1
BERT large embeddings		77.7 ±0.1	81.4 ±0.1

Table 5: Ablation study of model components. The analysis is with our base model size.

Data oracle variation	SMATCH (%) Train oracle	Model test
None	98.9	81.7 ±0.1
No subgraph breakdown	97.8	80.6 ±0.1
Create farther edges first	98.9	81.4 ±0.2
Post-order subgraph traversal	98.9	81.8 ±0.1

Table 6: Results of model performance with different data oracles on AMR 2.0 corpus.

The bottom part of Table 5 evaluates the contribution of output space restriction for target and input pre-trained embeddings for source, respectively. Removing the restriction for target output space i.e. the valid actions, hurts the model performance, as the model may not be able to learn the underlying rules that govern the target sequence restrictions. Switching the RoBERTa large embeddings to RoBERTa base or BERT large also hurts the performance (although score drops are only 0.3 $\sim$ 0.6), indicating that the contextual embeddings from large and better pre-trained models better equip the parser to capture semantic relations in the source sentence.

Effect of Oracle Setup As our model directly learns from the oracle actions, we study how the upstream transition system affects the model performance by varying transition setups in Table 6. We try three variations of the oracle. In the first setup, we measure the impact of breaking down SUBGRAPH action into individual node generation and attachment actions. We do this by using the SUBGRAPH for all cases of multi-node alignments. This degrades the parser performance and oracle SMATCH considerably, dropping by absolute 1.1 points. This is expected, since SUBGRAPH action makes internal nodes of the subgraph unattachable. In the second setup, we vary the order of edge creation actions. We reverse it so that the edges connecting farther nodes are built first. Although this does not affect the oracle score, we observe that the model performance on this oracle drops by 0.3. The reason might be that the easy close-range edge building actions become harder when pushed farther, also making easy decisions first is less prone to error propagation. Finally, we also change the order in which the various nodes connected to a token are created. Instead of generating the nodes from the root downwards, we perform a post-order traversal, where leaves are generated before parents. This also does not affect oracle score, however it gave a minor gain in parser performance.

Effect of Beam Size Figure 5 shows performance for different beam sizes. Ideally, if the model is more certain and accurate in making right predictions at different steps, the decoding performance should be less impacted by beam size. The results show that performance improves with beam size, but the gains saturate at beam size 3. This indicates that a smaller beam size can be considered

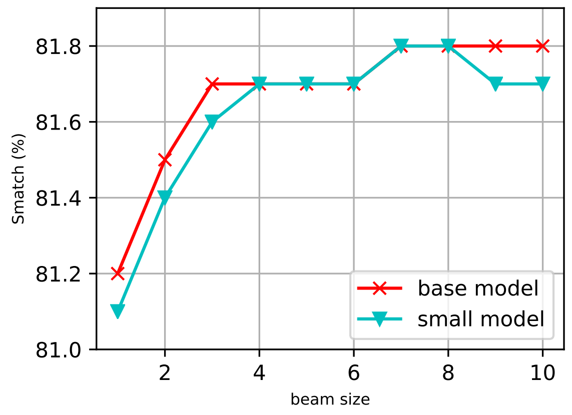


Figure 5: Effect of decoding beam size for SMATCH, with our best single models on AMR 2.0 test set.

for application scenarios with time constraints.

7 Related Work

With the exception of Astudillo et al. (2020), other works introducing stack and buffer information into sequence-to-sequence attention parsers (Liu and Zhang, 2017; Zhang et al., 2017; Buys and Blunsom, 2017), are based on RNNs and do not attain high performances. Liu and Zhang (2017); Zhang et al. (2017) tackle dependency parsing and propose modified attention mechanisms while Buys and Blunsom (2017) predicts semantic graphs jointly with their alignments and compares stack-based with latent and fixed alignments. Compared to the stack-Transformer (Astudillo et al., 2020), we propose the use of an action pointing mechanism to decouple word and node representation, remove the need for stack and buffer and model graph structure on the decoder side. We show that these improvements yield superior performance while exploiting the same inductive biases with little train data or small models.

Vilares and Gómez-Rodríguez (2018) proposed an AMR-CONVINGTON system for unrestricted non-projective AMR parsing, comparing the current word with all previous words for arc attachment as we propose. However, their comparison is done with sequential actions whereas we use an efficient pointer mechanism to parallelize the process.

Regarding the use of pointer mechanisms for arc attachment, Ma et al. (2018b) proposed the stack-pointer network to build partial graph representations, and Fernández-González and Gómez-Rodríguez (2020) adopted pointers along with the left-to-right scan of the sentence, greatly improving the efficiency. Compared with these works, we tackle a more general text-to-graph problem, where nodes are only loosely related to words, by utilizing the action-pointer mechanism. Our method is also able to build up to depth M graph representations with M decoding layers.

While not explicitly stated, graph-based approaches (Zhang et al., 2019a; Cai and Lam, 2020) generate edges with a pointing mechanism, either with a deep biaffine classifier (Dozat and Manning, 2018) or with attention (Vaswani et al., 2017). They also model inductive biases indirectly through graph re-categorization, detailed in Section 6.1, which requires a name entity recognition system at test time. Re-categorization was proposed in Lyu and Titov (2018), which reformulated alignments as a differentiable permutation problem, interpretable as another form of inductive bias.

Finally, augmenting seq-to-seq models with graph structures has been explored in various NLP areas, including machine translation (Hashimoto and Tsuruoka, 2017; Moussallem et al., 2019), text classification (Lu et al., 2020), AMR to text generation (Zhu et al., 2019), etc. Most of these works model graph structure in the encoder since the complete source sentence and graph are known. We embed a dynamic graph in the Transformer decoder during parsing. This is similar to broad graph generation approaches (Li et al., 2018) relying on graph neural networks (Li et al., 2019), but our approach is much more efficient as we do not require heavy re-computation of node representations.

8 Conclusion

We present an Action-Pointer mechanism that can naturally handle the generation of arbitrary graph constructs, including re-entrancies and multiple nodes per token. Our structural modeling with incremental encoding of parser and graph states based on a single Transformer architecture proves to be highly effective, obtaining the best results on all AMR corpora among models with similar learnable parameter sizes. An interesting future exploration is on combining our system with large pre-trained models such as BART, as directly fine-tuning on the latter shows great potential in boosting the performance (Bevilacqua et al., 2021). Although we focus on AMR graphs in this work, our system can essentially be adopted to any task generating graphs from texts where copy mechanisms or hard-attention plays a central role.

References

Ramon Fernandez Astudillo, Miguel Ballesteros, Tahira Naseem, Austin Blodgett, and Radu Florian. 2020. Transition-based parsing with stack-transformers. *arXiv preprint arXiv:2010.10669*.

Miguel Ballesteros and Yaser Al-Onaizan. 2017. Amr parsing using stack-lstms. *arXiv preprint arXiv:1707.07755*.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of the 7th linguistic annotation workshop and interoperability with discourse*, pages 178–186.

Michele Bevilacqua, Rexhina Blloshmi, and Roberto Navigli. 2021. One spring to rule them both: Symmetric amr semantic parsing and generation without a complex pipeline.

Jan Buys and Phil Blunsom. 2017. Robust incremental neural semantic graph parsing. *arXiv preprint arXiv:1704.07092*.

Deng Cai and Wai Lam. 2019. Core semantic first: A top-down approach for amr parsing. *arXiv preprint arXiv:1909.04303*.

Deng Cai and Wai Lam. 2020. Amr parsing via graph-sequence iterative inference. *arXiv preprint arXiv:2004.05572*.

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.

Marco Damonte, Shay B Cohen, and Giorgio Satta. 2016. An incremental parser for abstract meaning representation. *arXiv preprint arXiv:1608.06111*.

Timothy Dozat and Christopher D Manning. 2018. Simpler but more accurate semantic dependency parsing. *arXiv preprint arXiv:1807.01396*.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2020. Transition-based semantic dependency parsing with pointer networks. *arXiv preprint arXiv:2005.13344*.

Jeffrey Flanigan, Chris Dyer, Noah A Smith, and Jaime G 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.

Jeffrey Flanigan, Sam Thomson, Jaime G 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.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*.

Jonas Groschwitz, Matthias Lindemann, Meaghan Fowlie, Mark Johnson, and Alexander Koller. 2018. Amr dependency parsing with a typed semantic algebra. *arXiv preprint arXiv:1805.11465*.

Zhijiang Guo and Wei Lu. 2018. Better transition-based amr parsing with a refined search space. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1712–1722.

Kazuma Hashimoto and Yoshimasa Tsuruoka. 2017. Neural machine translation with source-side latent graph parsing. *arXiv preprint arXiv:1702.02265*.

Young-Suk Lee, Ramon Fernandez Astudillo, Tahira Naseem, Revanth Gangi Reddy, Radu Florian, and Salim Roukos. 2020. Pushing the limits of amr parsing with self-learning. *arXiv preprint arXiv:2010.10673*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Michael Lingzhi Li, Meng Dong, Jiawei Zhou, and Alexander M Rush. 2019. A hierarchy of graph neural networks based on learnable local features. *arXiv preprint arXiv:1911.05256*.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. 2018. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*.

Matthias Lindemann, Jonas Groschwitz, and Alexander Koller. 2019. Compositional semantic parsing across graphbanks. *arXiv preprint arXiv:1906.11746*.

Jiangming Liu and Yue Zhang. 2017. Encoder-decoder shift-reduce syntactic parsing. In *Proceedings of the 15th International Conference on Parsing Technologies*, pages 105–114, Pisa, Italy. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Zhibin Lu, Pan Du, and Jian-Yun Nie. 2020. Vgcn-bert: Augmenting bert with graph embedding for text classification. In *European Conference on Information Retrieval*, pages 369–382. Springer.

Chunchuan Lyu, Shay B Cohen, and Ivan Titov. 2020. A differentiable relaxation of graph segmentation and alignment for amr parsing. *arXiv preprint arXiv:2010.12676*.

Chunchuan Lyu and Ivan Titov. 2018. Amr parsing as graph prediction with latent alignment. *arXiv preprint arXiv:1805.05286*.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. 2018a. Stack-pointer networks for dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1403–1414, Melbourne, Australia. Association for Computational Linguistics.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. 2018b. Stack-pointer networks for dependency parsing. *arXiv preprint arXiv:1805.01087*.

Diego Moussallem, Mihael Arčan, Axel-Cyrille Ngonga Ngomo, and Paul Buitelaar. 2019. Augmenting neural machine translation with knowledge graphs. *arXiv preprint arXiv:1902.08816*.

Tahira Naseem, Abhishek Shah, Hui Wan, Radu Florian, Salim Roukos, and Miguel Ballesteros. 2019. Rewarding smatch: Transition-based amr parsing with reinforcement learning. *arXiv preprint arXiv:1905.13370*.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*.

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.

Nima Pourdamghani, Yang Gao, Ulf Hermjakob, and Kevin Knight. 2014. Aligning english strings with abstract meaning representation graphs. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 425–429.

Michael Pust, Ulf Hermjakob, Kevin Knight, Daniel Marcu, and Jonathan May. 2015. Parsing english into abstract meaning representation using syntax-based machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1143–1154.

Rik Van Noord and Johan Bos. 2017. Neural semantic parsing by character-based translation: Experiments with abstract meaning representations. *arXiv preprint arXiv:1705.09980*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Ashwin K Vijayakumar, Michael Cogswell, Ramprasath R Selvaraju, Qing Sun, Stefan Lee, David Crandall, and Dhruv Batra. 2016. Diverse beam search: Decoding diverse solutions from neural sequence models. *arXiv preprint arXiv:1610.02424*.

David Vilares and Carlos Gómez-Rodríguez. 2018. A transition-based algorithm for unrestricted amr parsing. *arXiv preprint arXiv:1805.09007*.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in neural information processing systems*, pages 2692–2700.

Chuan Wang and Nianwen Xue. 2017. Getting the most out of amr parsing. In *Proceedings of the 2017 conference on empirical methods in natural language processing*, pages 1257–1268.

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.

Dongqin Xu, Junhui Li, Muhua Zhu, Min Zhang, and Guodong Zhou. 2020. Improving amr parsing with sequence-to-sequence pre-training. *arXiv preprint arXiv:2010.01771*.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019a. Amr parsing as sequence-to-graph transduction. *arXiv preprint arXiv:1905.08704*.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019b. Broad-coverage semantic parsing as transduction. *arXiv preprint arXiv:1909.02607*.

Zhirui Zhang, Shujie Liu, Mu Li, Ming Zhou, and Enhong Chen. 2017. Stack-based multi-layer attention for transition-based dependency parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1677–1682, Copenhagen, Denmark. Association for Computational Linguistics.

Jiawei Zhou and Alexander M Rush. 2019. Simple unsupervised summarization by contextual matching. *arXiv preprint arXiv:1907.13337*.

Jie Zhu, Junhui Li, Muhua Zhu, Longhua Qian, Min Zhang, and Guodong Zhou. 2019. Modeling graph structure in transformer for better amr-to-text generation. *arXiv preprint arXiv:1909.00136*.

A A More Detailed Example of Action-Pointer Transitions

We present a step-by-step walk-through of our actions on a less trivial example for generating the AMR in Figure 6. The sentence contains a named entity which also demonstrates the MERGE and SUBGRAPH usage of our transition system.

B Action-Pointer Oracle

For a given sentence, at every oracle step, apply the actions in the order listed below. Continue until the source cursor moves past the last token.

1. **If** cursor is on a non-final token of a span aligned to a node, keep moving cursor (and collecting tokens) with MERGE till it reaches the final token of the span.

2. **If** there is a matching pattern for current token(s) in SUBGRAPH() action dictionary:

    - Apply matching SUBGRAPH() action.
    - Generate edges between the root of the sub-graph and past nodes through LA(), RA(). Generate closer edges first.

    Otherwise, for all nodes aligned to the current token, in top-down order:

    - Generate node through COPY (lemma or first sense), and if not possible then through PRED().
    - Generate edges between the last nodes and past nodes through LA(), RA(). Generate closer edges first.

3. **If** no action performed at step 2, move cursor with REDUCE **otherwise**, move cursor with SHIFT.

C Action-Pointer Decoding

We outline the decoding algorithm for our model in Algorithm 1, to combine the actions with pointers, as well as taking in parsing states and graph structures for the model during the decoding steps. Detailed beam search process is ignored. Although tacking the specific problem of AMR graph generation with pointers, our constraint decoding process is a modified beam search algorithm with different components and step-wise controls, among others (Vijayakumar et al., 2016; Zhou and Rush, 2019).

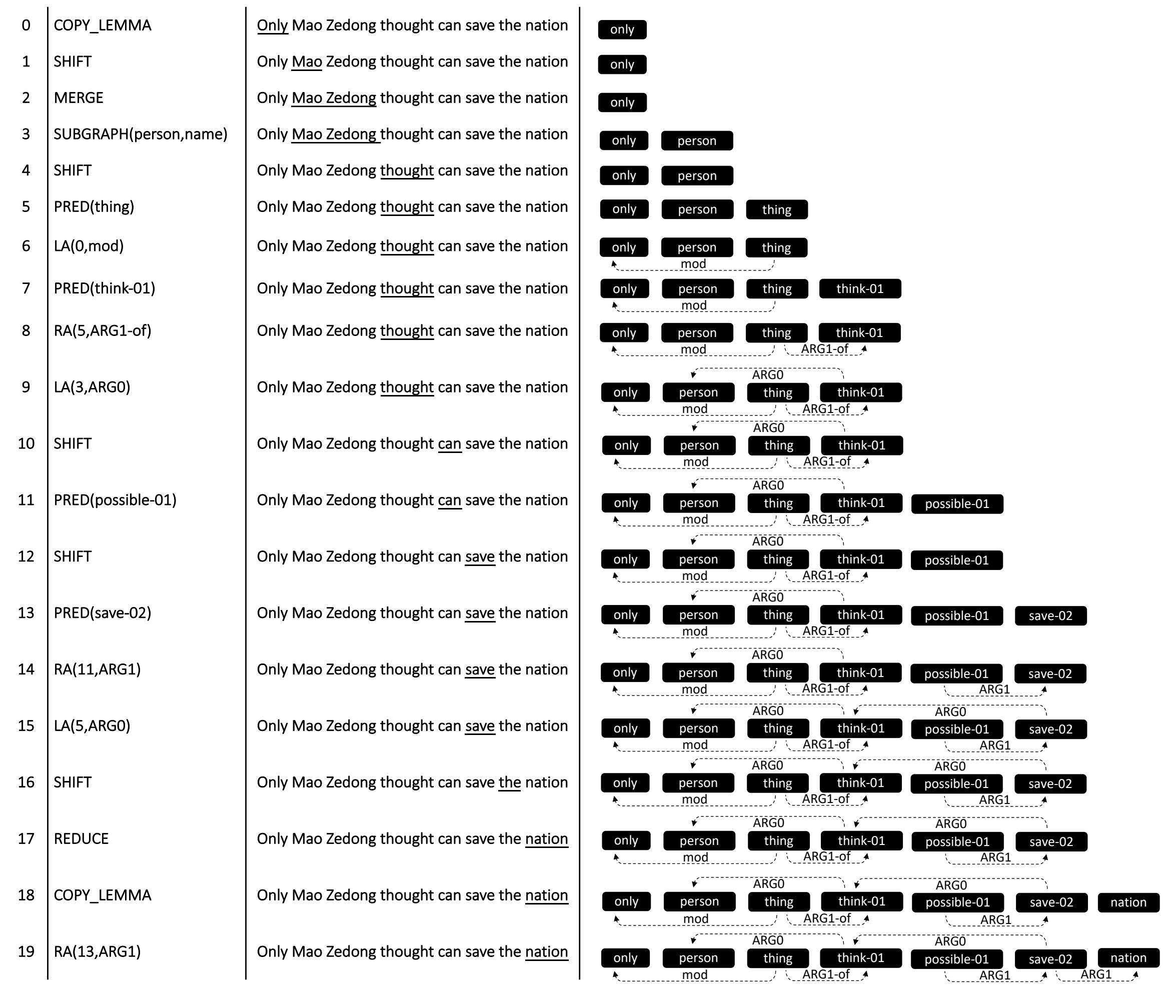
D Number of Parameters

Our model is a single Transformer (Vaswani et al., 2017) model. The pointer distribution, action-source alignment encoding from parsing state, and structural graph embedding are all contained in certain attention layers and heads, without introducing any extra parameters on original Transformer. We fix our model size and all the embedding size to be 256, and the feedforward hidden size in Transformer as 512. And they are the same for our base model with 6 layers and 4 heads and our small model with 3 layers and 4 heads, both for encoder and decoder.

We use pre-trained RoBERTa embeddings for the source token embeddings. The embeddings are extracted in pre-processing and fixed. The RoBERTa model parameters are fixed and not trained with our model. We have a projection layer to project the RoBERTa embedding size 1024/768 to our model size 256.

The target side dictionary is built from all the oracle actions without pointers on training data. The dictionary size for AMR 1.0 is 4640, for AMR 2.0 is 9288, and for AMR 3.0 is 11680. We build the target action embeddings along with the model for the action prediction on top of Transformer decoder. The dictionary embedding size is fixed at 256.

Overall, the total number of parameters for our 6 layer base model is 14,852,096 on AMR 1.0, 21,438,464 on AMR 2.0, and 25,550,848 on AMR 3.0 (difference is in target dictionary embedding size). The total number of parameter for our 3 layer small model is 10,898,432 for AMR 1.0 and 17,484,800 on AMR 2.0 (difference is in target dictionary embedding size).



0	COPY_LEMMA	Only Mao Zedong thought can save the nation
1	SHIFT	Only Mao Zedong thought can save the nation
2	MERGE	Only Mao Zedong thought can save the nation
3	SUBGRAPH(person,name)	Only Mao Zedong thought can save the nation
4	SHIFT	Only Mao Zedong thought can save the nation
5	PRED(thing)	Only Mao Zedong thought can save the nation
6	LA(0,mod)	Only Mao Zedong thought can save the nation
7	PRED(think-01)	Only Mao Zedong thought can save the nation
8	RA(5,ARG1-of)	Only Mao Zedong thought can save the nation
9	LA(3,ARG0)	Only Mao Zedong thought can save the nation
10	SHIFT	Only Mao Zedong thought can save the nation
11	PRED(possible-01)	Only Mao Zedong thought can save the nation
12	SHIFT	Only Mao Zedong thought can save the nation
13	PRED(save-02)	Only Mao Zedong thought can save the nation
14	RA(11,ARG1)	Only Mao Zedong thought can save the nation
15	LA(5,ARG0)	Only Mao Zedong thought can save the nation
16	SHIFT	Only Mao Zedong thought can save the nation
17	REDUCE	Only Mao Zedong thought can save the nation
18	COPY_LEMMA	Only Mao Zedong thought can save the nation
19	RA(13,ARG1)	Only Mao Zedong thought can save the nation

Figure 6: Step-by-step actions based on our action-pointer transition system. We illustrate the use of MERGE and SUBGRAPH with the named entity of a person's name in this example. The source cursor after the action is applied is pointing at words underlined, and the partially built graph is shown in the right-most column.

Algorithm 1: Constrained beam search for action-pointer decoding

```
Input: Initial token $a_0$ =</s>, beam size $k$,
       max step $T_{max}$, action dictionary $D$
       without pointers, model $M$ that
       outputs both distribution over $D$ and
       the pointer distribution from
       self-attention
Output: Decoded results
       $y_1 = (a_1, p_1), y_2 =$
       $(a_2, p_2), \cdots, y_T = (a_T, p_T)$
initialization: step $t = 1$, $k$ state machines
while $t <= T_{max}$ do
    1) Get the valid action dictionary
      $D_t \subset D$, previous node action
      positions $N_t \subset \{0, 1, 2, \ldots, t\}$,
      current token cursor $c_t$, and current
      graph $G_t$ (all from the corresponding
      state machines);
    2) Input prefix $a_0, a_1, \cdots, a_{t-1}$ and
      $D_t, c_t, G_t$ into model, get output
      distribution $\mathbf{P}(a_t|\mathbf{y}_{<t})$, and the
      self-attention distribution $Q(p)$ from
      pointer head with $p$ over $\{0, 1, \ldots, t\}$;
    3) Take the most likely valid pointer
      value, with $p^* = \text{argmax}_{p \in N_t} Q(p)$,
      and its score $q^* = \max_{p \in N_t} Q(p)$;
    for each possible action $a$ from $D$ do
        if $a$ is an edge action then
            combine the action probability
              with pointer probability
              $\mathbf{P}(y_t) = \mathbf{P}(a_t|\mathbf{y}_{<t}) \cdot q^*$, with
              $y_t = (a, p^*)$
        else
            set $\mathbf{P}(y_t) = \mathbf{P}(a_t|\mathbf{y}_{<t})$, with
              $y_t = (a, null)$
        end
    end
    Do beam search with $\mathbf{P}(y_t)$ over $y_t$ to
      get $k$ decoded results;
    Apply the corresponding actions with
      the $k$ state machines to update parser
      states and partial graphs for each beam
      candidate.
end
```