AMR Parsing with Action-Pointer Transformer

Anonymous

Abstract

Abstract Meaning Representation parsing belongs to a category of sentence-to-graph prediction tasks where the target graph is not explicitly linked to the sentence tokens. However, nodes or subgraphs are semantically related to subsets of the sentence tokens, and locality between words and related nodes is often preserved. Transition-based approaches have recently shown great progress in capturing these inductive biases but still suffer from 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. We model the transitions as well as the pointer mechanism using a single Transformer model. Parser state and graph structure information is efficiently encoded using attention heads. We show that our approach leads to increased expressiveness while capitalizing inductive biases and attains new state-of-the SMATCH scores on AMR 1.0 (78.5) and AMR 2.0 (81.8).

1 Introduction

Abstract Meaning Representation (AMR) (Banarescu et al., 2013) is a sentence level semantic formalism that encodes the meaning of a sentence into a rooted directed acyclic graph where nodes represent concepts and edges represent relations (see Figure 1). For AMR parsing, the task of generating the graph from a sentence, transition-based algorithms (Nivre, 2008) derived from dependency parsing are often applied. To account for the higher complexity of AMR, various extensions have been explored, such as transforming the dependency parses into AMR (Wang et al., 2015), introducing additional lookup tables for AMR concepts (Damonte et al., 2016; Vilares and Gómez-Rodríguez, 2018), dynamically adding extra tokens for additional nodes (Guo and Lu, 2018; Vilares and Gómez-Rodríguez, 2018), utilizing a SWAP action and collapsing nodes (Ballesteros and Al-Onaizan, 2017; Naseem et al., 2019; Astudillo et al., 2020) or a cache data structure (Peng et al., 2018). These approaches explicitly capture the local graph-sentence interaction and hold current state-of-the-art results (Astudillo et al., 2020; Lee et al., 2020).

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

Despite their success, the transition-based approaches still suffer from limited expressiveness because some of the design principles of the original dependency parsers do not transfer well to AMR. For instance, in AMR multiple graph nodes may correspond to one sentence token, but the transition based parsers use the tokens to refer to both themselves and the graph nodes, requiring additional workarounds for AMR. Another potential drawback of transition based AMR parsers is lengthy action sequences. Most algorithms allow arcs only between adjacent words in the stack. AMR has frequent non-projective attachments, requiring additional actions to bring non-adjacent tokens together. Long action sequences affect both a model's ability to learn and its decoding speed.

In contrast, another view of AMR parsing treats it as a graph generation problem conditioned on the source text, loosening the local semantic correspondence between graph nodes and source tokens. The graph-based approaches (Zhang et al., 2019a,b; Cai and Lam, 2020) directly tackle node generation through seq-to-seq models and edge generation with variants of attention mechanism by comparing node representations (Peng et al., 2017; Dozat and Manning, 2018). These approaches

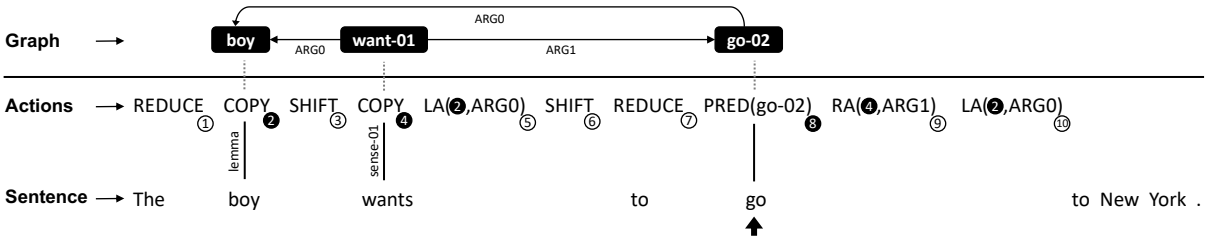

Figure 2: Source sentence, target actions and AMR graph for the sentence *The boy wants to go to New York* (partially parsed). The black arrow marks the current token cursor position. The circles contain the action indices (used as ids), black circles indicate node creating actions. Only these actions are available for edge attachments. Notice that the edge actions (at steps 5, 9 and 10) 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.

achieve competitive results without the restrictions of transition-based approaches, but require graph re-categorization, a form of graph normalization, for performance.

In this work, we propose a novel AMR parsing system, referred to as Action-Pointer system, that combines the advantages of both the transition-based approaches and more general graph-generation approaches. We model parsing as sequence-to-sequence problem, predicting graph building actions 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). Similar to Astudillo et al. (2020), we model the parser state by masking cross-attention. In our case, it is based simply on a monotonic action-source alignment indicating cursor position, rather than stack and buffer contents. Finally we also embed the AMR graph structural information in the target decoder, through a novel step-wise incremental graph message passing method (Gilmer et al., 2017) enabled by the decoder self-attention mechanism.

Experiments on both AMR 1.0 and AMR 2.0 benchmark datasets show the effectiveness of our Action-Pointer system. We establish a new state-of-the-art on both datasets for AMR parsing, surpassing even the best models trained with large amounts of silver data.

2 AMR Generation with Action-Pointer

We describe the process of generating the AMR graph from a source sentence through a sequence of actions, reducing the graph generation problem to an action sequence generation problem. Figure 2 shows a partially parsed example of the source sentence, action sequence and associated 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 is operated by actions at step t:

SHIFT moves cursor one position to the right, such that $c_{t+1} = c_t + 1$. SHIFT happens *after* all the nodes and edges spawned by the current token are completed.

REDUCE is a special case of SHIFT used to indicate that no action was performed conditioned on the token at current cursor position.

At a cursor position c_t, we can generate any number of nodes and edges through following actions:

COPY predicts an AMR node at the current token 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 for this action.

PRED(LABEL) predicts an AMR node with name LABEL from the node names seen at train time and not predictable by the COPY actions.

LA(ID,LABEL) creates an arc with LABEL from the most recently generated node to a previous node with position ID. Note that action indices are used for this pointing mechanism and we can only point to past node generating actions.

RA(ID,LABEL) creates an arc with LABEL from the node at position ID to the most recently generated node.

The node prediction actions are followed by edge creation actions. Therefore, It is always possible to establish an edge between the last predicted node and any previous node. 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.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 provides more expressiveness and flexibility than previous transition-based AMR parsers, while keeping a strong inductive bias.

The above listed basic actions can naturally generate multi-node subgraphs from single tokens. However, we found that in practice, breaking such subgraphs into individual actions improves parser performance only when such breakdown is necessary to recover the gold graph. In other words, when internal nodes of the subgraph have external attachments. Furthermore, sometimes spans of multiple tokens correspond to one or more nodes in the graph. This is mainly the case for dates and named-entities. Similarly to Ballesteros and Al-Onaizan (2017), two additional actions are used to spawn entire subgraphs from one or more tokens (see Figure 6 in Appendix for an example):

MERGE merges tokens x_{c_t} and x_{c_t+1} and moves the cursor one position to the right. Usually used before SUBGRAPH action described below.

SUBGRAPH(LABEL) produces a subgraph with a LABEL for current token(s). Since nodes are represented by actions, the whole subgraph has only one action representing it. Any future attachments can only be made to the root of the subgraph.

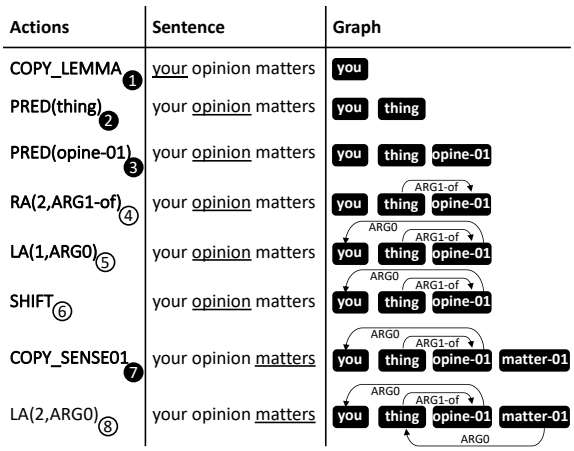

Figure 3: Step-by-step actions on the sentence *your opinion matters*. Creates subgraph from a single word (thing :ARG1-of opine-01) and allows attachment to all its nodes. The cursor is at underlined words.

Similar to prior transition-based approaches, at train time an *oracle* provides the optimal action sequence for a given sentence and graph. This oracle acts as rule-base teacher informed by pre-computed word-node alignments. We use the alignments generation method from Astudillo et al. (2020). For the proposed action set, it is easy to see that it suffices to produce all nodes aligned to a word. The oracle assumes that in such cases, the aligned nodes make a connected subgraph which is traversed in pre-order for node generation. The closer arcs are generated before the further ones.

3 Action-Pointer Transformer

3.1 Basic Architecture

The backbone of our model is the multi-layer 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

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

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, therefore we have

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

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

$$\mathbf{d}^m = \text{FF}^m(\text{CA}^m(\text{SA}^m(\mathbf{d}^{m-1}, \mathbf{d}^{m-1}), \mathbf{e}^N))$$

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

We model $\mathbf{P}(a_t \mid \mathbf{y}_{<t}, \mathbf{x})$ the standard way by projecting $\mathbf{d}_t^M$ into the action vocabulary D. We model $\mathbf{P}(p_t \mid \mathbf{y}_{<t}, \mathbf{x})$ using one self-attention head at the top layer of the decoder SA^M. This is a natural choice, since this head is likely to have high values for the nodes involved in the edge direction and label prediction. Note that the edge action and its pointing value are both output at the same step from the top decoder layer. This essentially lets the model multitask. However, 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 $\text{CA}^m()$ 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.

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

3.2 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 $\text{SA}^m()$. 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 actions $a_{t+1}, a_{t+2}, \ldots, a_{t+k}$ that connect the current node with previous nodes, pointed by $p_{t+1}, p_{t+2}, \ldots, p_{t+k}$ 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 to New York*, 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 $\text{SA}^m()$ 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 neigh-

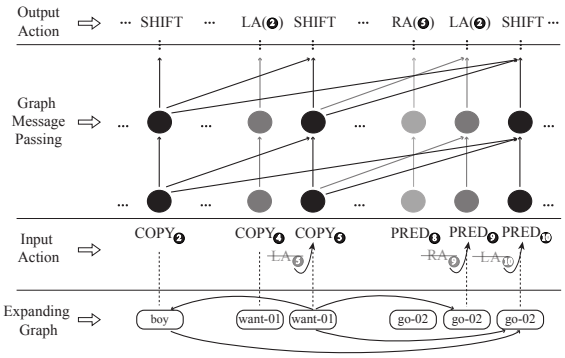

Figure 4: Incremental graph embedding in the decoder self-attention heads. Only node actions are considered, and they are attending to their neighboring nodes at different steps progressively based on the newly added edge at each step. For better illustration, We ignore the labels inside actions but only keep the edge pointers which use the latest reference of a node.

borhood information. 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 defined by taking the log of Equation (1). The valid action space, action-source alignment c_t, and the graph embedding mask at each step t are pre-calculated during 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 the 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 the Appendix.

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[2]. We also report results on the lastest 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 (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, which is supervised during training and used for decoding during testing. 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 the 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 the target action dictionary 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 $5\mathrm{e}{-4}$ 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

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

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.

size of 10 for decoding. We implement our model 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

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.

We compare our model with existing approaches in Table 2[3]. We indicate the use of pre-trained BERT embeddings with ^{B} and graph re-categorization with ^{G}. Graph re-categorization (Lyu and Titov, 2018; Zhang et al., 2019a; Cai and Lam, 2020) 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 achieve better results than all previous approaches, both with our small and base models, establishing the new state-of-the-art parsing scores of 81.8 on AMR 2.0 and 78.5 on AMR 1.0 on average, which improves 1.6 points over the 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 AMR 1.0, which has a small training set. This indicates 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 85000 extra sentences for self-annotation (silver data). We believe our approach could also benefit from these techniques for further improvement. For the most recent AMR 3.0 dataset, we report our results for future reference.

[3]We exclude Xu et al. (2020) AMR1.0 numbers since they report 16833 train sentences, not 10312.

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,G}	75.4
	Astudillo et al. (2020)* ^{B,G}	76.9 ±0.1
	Lee et al. (2020)^{B} (85K silver)	78.2 ±0.1
	Ours small^{B}	78.2 / 78.2 ±0.0
	Ours base^{B}	**78.5 / 78.3 ±0.1**
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,G}	80.2
	Astudillo et al. (2020)* ^{B,G}	80.2 ±0.0
	Xu et al. (2020) (4M silver)	80.2
	Lee et al. (2020)^{B} (85K silver)	81.3 ±0.0
	Ours small^{B}	81.7 / 81.5 ±0.2
	Ours base^{B}	**81.8 / 81.7 ±0.1**
AMR 3.0	Lyu et al. (2020)	75.8
	Ours base^{B}	**80.4 / 80.3 ±0.1**

Table 2: SMATCH scores on AMR 1.0, 2.0, and 3.0 test sets. ^{B} indicates pre-trained BERT/RoBERTa embeddings, ^{G} use of graph re-categorization, * improved results reported in Lee et al. (2020). We report the best/average score ± standard deviation over 3 seeds.

Table 3 shows the fine-grained AMR2.0 evaluation of (Damonte et al., 2016). 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 4. The top part of the table shows

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)$^{*\ B,G}$	80.2	84.2	80.7	88.1	87.5	64.5	78.8	70.3	78.2
Ours small^{B}	81.7	85.4	82.2	**88.9**	**88.9**	67.5	78.7	70.6	80.7
Ours base^{B}	**81.8**	**85.5**	**82.3**	88.7	88.5	69.7	78.8	**71.1**	**80.8**

Table 3: Fine-grained F1 scores on the AMR 2.0 test set. ^{B} and ^{G} marks uses of pre-trained BERT 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

Table 4: 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 5: Results of model performance with different data oracles on AMR 2.0 corpus.

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 boosted 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.

The bottom part of Table 4 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 also hurts the performance, indicating that the contextual embeddings from large 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 5. 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

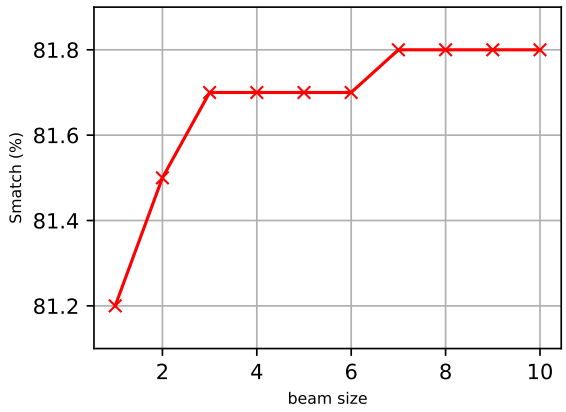

Figure 5: Effect of decoding beam size for SMATCH, with our best base model on AMR 2.0 test set.

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 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 and attaining good performances 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.

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. 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.

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.

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*.

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*.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

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*.

Hao Peng, Sam Thomson, and Noah A Smith. 2017. Deep multitask learning for semantic dependency parsing. *arXiv preprint arXiv:1704.06855*.

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.

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.

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.

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 Decoding

We outlined the decoding algorithm for our model, to combine the actions with pointers, as well as taking in parsing states and graph structures. Detailed beam search process is ignored.

C 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 12008. 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 25,718,784 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).

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 $t = 1$

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;

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.

end

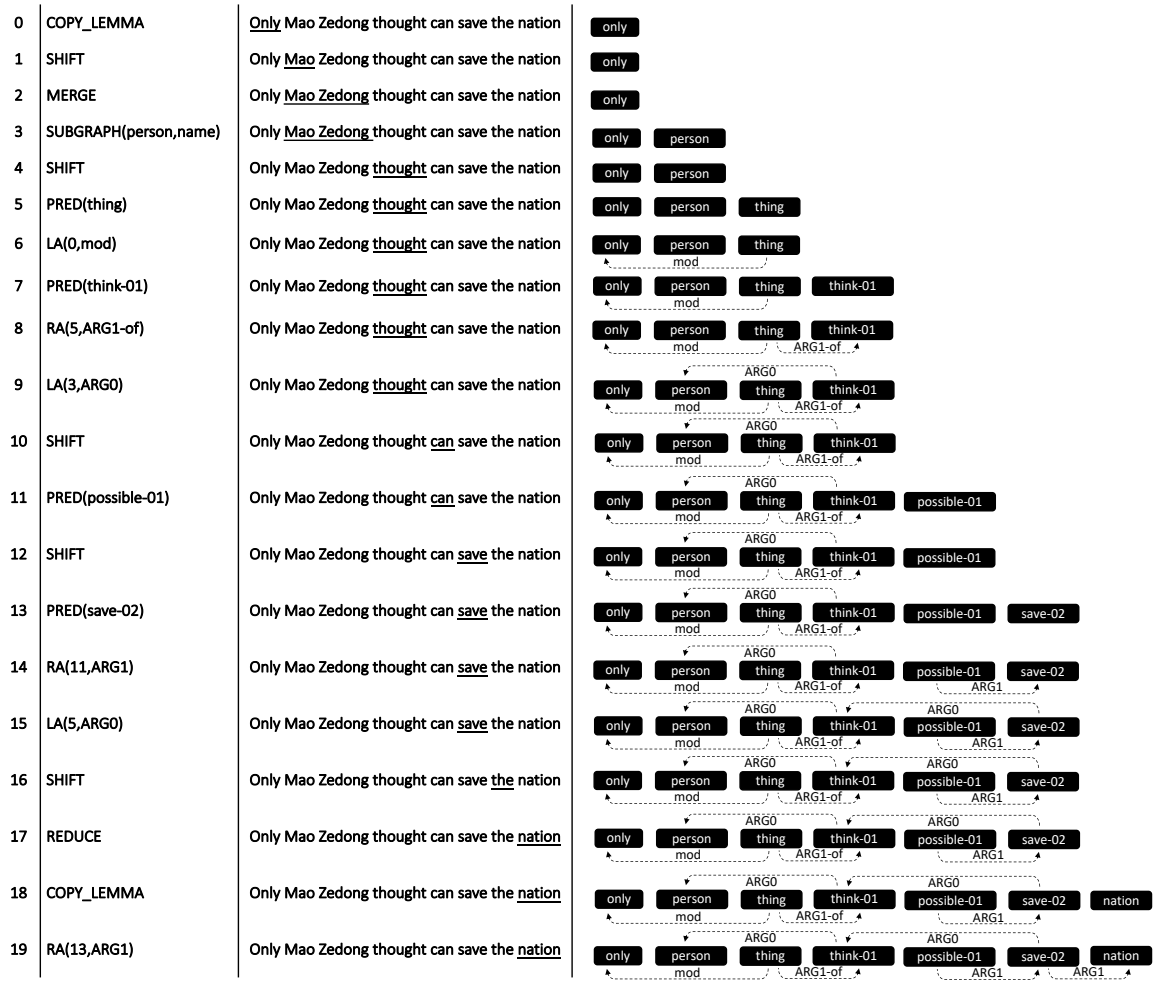

Figure 6: Step-by-step actions based on our action-pointer transition system. We illustrate the use of MERGE and SUBGRAPH with the named entity of a person's name in this example.