Hitachi at MRP 2020: Text-to-Graph-Notation Transducer

Hiroaki Ozaki[*], **Gaku Morio**[*], [†]**Yuta Koreeda, Terufumi Morishita** and **Toshinori Miyoshi**
Research and Development Group, Hitachi, Ltd., Japan
[†]Research and Development Group, Hitachi America, Ltd., USA
{hiroaki.ozaki.yu, gaku.morio.vn, terufumi.morishita.wp, toshinori.miyoshi.pd}@hitachi.com, [†]yuta.koreeda@hal.hitachi.com

Abstract

This paper presents our proposed parser for the shared task on Meaning Representation Parsing (MRP 2020) at CoNLL, where participant systems were required to parse five types of graphs in different languages. We propose to unify these tasks as a text-to-graph-notation transduction in which we convert an input text into a graph notation. To this end, we designed a novel Plain Graph Notation (PGN) that handles various graphs universally. Then, our parser predicts a PGN-based sequence by leveraging Transformers and biaffine attentions. Notably, our parser can handle any PGN-formatted graphs with fewer framework-specific modifications. As a result, ensemble versions of the parser tied for **1st** place in both cross-framework and cross-lingual tracks.

1 Introduction

This paper introduces the proposed parser of the Hitachi team for the CoNLL 2020 Cross-Framework Meaning Representation Parsing (MRP 2020) shared task (Oepen et al., 2020). Different from the previous MRP 2019 shared task (Oepen et al., 2019), there are two tracks. The first is a *cross-framework* track that aims at parsing English sentences to five different meaning representation graphs, i.e., EDS (Oepen and Lønning, 2006), PTG (Hajič et al., 2012), UCCA (Abend and Rappoport, 2013; Hershcovich et al., 2017), AMR (Banarescu et al., 2013), and DRG (Van Der Sandt, 1992; Bos et al., 2017). The other is a *cross-lingual* track that targets four different frameworks and three languages, i.e., German UCCA and DRG, Chinese AMR (Li et al., 2016), and Czech PTG. In both tracks, the goal was to design a unified parser for all graphs.

[*] Contributed equally. Ozaki mainly developed PGN parser. Morio mainly developed neural models.

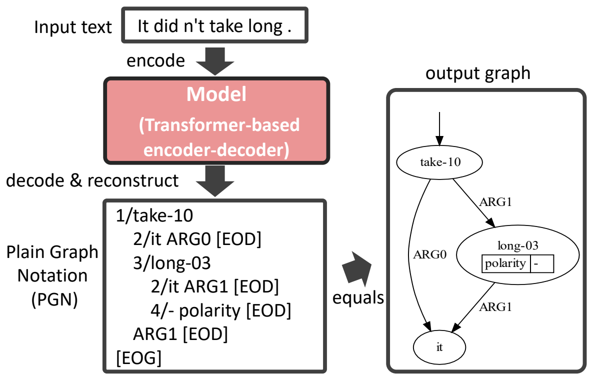


Figure 1: Given an input text, we tokenize and encode the text by pre-trained Transformer encoder (e.g., BERT). Then, Transformer decoder is applied to produce a Plain Graph Notation (PGN) that is convertible into a general graph.

In this paper, we propose a novel parser to unify graph predictions across all frameworks and languages. To this end, we introduce a *text-to-graph-notation* transduction. The overview of our parser is shown in Figure 1. Our parser utilizes sequence-to-sequence Transformer architectures (Vaswani et al., 2017) to generate a plain graph notation (PGN), which we newly designed as a context-free language. PGN is a simplified notation based on the PENMAN notation (Matthiessen and Bateman, 1991) that is generally used for AMR graphs. However, PGN is tailored for direct generation by sequence-to-sequence architecture.

Our parser is expected to combine both strengths of the neural graph-based and transition-based parsers (McDonald et al., 2005; Yamada and Matsumoto, 2003; Kulmizev et al., 2019; Ma et al., 2018). This is because the Transformer decoder directly draws attentions like a graph-based parser does, as well as handles higher-level effects of graph structures by sequence prediction like a transition-based parser does. Moreover, our parser is practically able to parse most of the graph variants in a unified manner. For example, our parser is able to predict directed acyclic graphs, disconnected graphs, directed multigraphs, reentrancy edges (Vilares and Gómez-Rodríguez, 2018), and source-side anchors without complicated language-dependent architectures.

Consequently, ensemble versions of our parser officially tied for 1st place in both the cross-framework track and cross-lingual track, achieving the top performances for English EDS, PTG, and AMR graphs. We also summarize other contributions as follows:

Alignment Free: PGN generation allows us to achieve completely alignment-free parsing.

Action Design Free: Compared to a transition-based parser, there is no need to design a complex transition strategy.

Fast Training: Since we leverage attentions, train speed is faster than a transition-based parser.

2 Related Work

2.1 Previous Systems for Cross-Framework Meaning Representation Parsing

MRP 2019 (Oepen et al., 2019) brought various parsing techniques together. According to Oepen et al. (2019), MRP systems can be characterized into three broad families of approaches: transition-, factorization-, or composition-based architectures. For example, the winning technique of HIT-SCIR (Che et al., 2019) at MRP 2019 used a transition-based parser based on a BERT encoder (Devlin et al., 2019). SUDA-Alibaba (Zhang et al., 2019c) proposed a graph-based approach with BERT. They used biaffine attention (Dozat and Manning, 2018) for the edge prediction. Donatelli et al. (2019) employed a compositional approach that represents each graph with its compositional tree structure (Lindemann et al., 2019).

2.2 Comparison with Other Systems

Like Zhang et al. (2019a), we model a context-free language instead of a sequence of transition actions, and parser states can be regarded as being implicitly materialized inside BERT's memory. However, our parser jointly generates nodes and edges based on PGN, making the system consider higher-level effects of graph structures.

The work most closely related to our study is (Zhang et al., 2019b), where the authors provided an encoder-decoder architecture to predict a sequence of semantic relations, employing a target node-, relation type-, and source node-module in the decoder. Similar to our study, Zhang et al. (2019b) encoded node and edge representation with the modules. While they jointly predicted node and edge labels, our parser outputs node and edge labels separately. In addition, they provided LSTMs whereas we provide Transformers that can draw attentions from both past node and edge representations in the decoder. Also, while Zhang et al. (2019b) solved reentrancies by producing the same node ID, we solve them with a biaffine classifier, making our parser solve reentrancies with attentions.

The biggest difference between the transition-based architectures for MRP (Che et al., 2019) and our work is that we have designed PGN such that it unifies all graph generation processes and eliminates the need to design framework-specific actions. In addition, Che et al. (2019) relied on explicit alignment between input tokens and nodes, whereas our model utilizes biaffine attention for anchoring only when it is necessary, allowing our model to be alignment-free.

```
graph      = {target "[EOG]"};
edge       = node_id {dep "[EOD]"};
dep        = target edge_label;
target     = node_id | edge;
node_id    = {digit};
edge_label = {letter};
```

Figure 2: PGN grammar described in EBNF. Essentially, a graph can be represented by a set of edges. However, to support floating nodes, we defined a graph as a set of edges and floating nodes.

Name	Function
`attr2name`	Append -{attr name} suffix to edge label name instead of having attributes.
`prop2node`	Make node properties independent nodes linked with edges named with properties' names.
`embed_label`	Replace node_id in PGN with {node_id}/{node_label}.

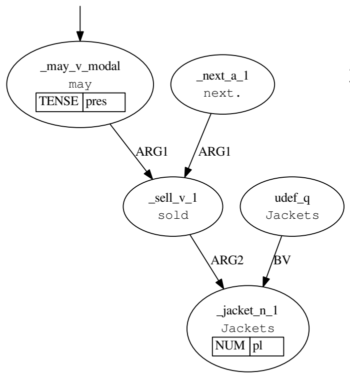
Table 1: List of PGN processors.

3 Plain Graph Notation

3.1 Format Design

To represent a graph as a text sequence, we newly designed a notation, called Plain Graph Notation (PGN), with the key principles shown below.

Simpler Format: Similar to PENMAN notation (Matthiessen and Bateman, 1991; Goodman, 2020), which is used to represent AMR graphs with text sequence, PGN is based on a context-free grammar.



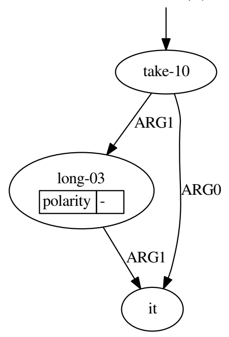
```
3/_may_v_modal
  6/pres TENSE [EOD]
  4/_sell_v_1
    5/_next_a_1 ARG1-of [EOD]
    2/_jacket_n_1
      7/pl NUM [EOD]
      1/udef_q BV-of [EOD]
    ARG2 [EOD]
  ARG1 [EOD]
[EOG]
```

(a2) PGN

```
<bos> [AS] [AE] _ may _ v _ modal [EON]
  pres [EON] TENSE [EOD]
  [AS] [AE] _ sell _ v _ 1 [EON]
    [AS] [AE] _ next _ a _ 1 [EON] ARG1-of [EOD]
    [AS] [AE] _ jacket _ n _ 1 [EON]
      pl [EON] NUM [EOD]
      udef _ q [EON] BV-of [EOD]
    ARG2 [EOD]
  ARG1 [EOD]
[EOG] <eos>
```

(a3) Prediction sequence

(a) An EDS graph and its PGN and prediction sequence for *Jackets may be sold next*



```
1/take-10
  2/it ARG0 [EOD]
  3/long-03
    2/it ARG1 [EOD]
    4/- polarity [EOD]
  ARG1 [EOD]
[EOG]
```

(b2) PGN

```
<bos> take - 10 [EON]
  it [EON] ARG0 [EOD]
  long - 03 [EON]
    [RNT] [EON] ARG1 [EOD]
    - [EON] polarity [EOD]
  ARG1 [EOD]
[EOG] <eos>
```

(b3) Prediction sequence

(b) An AMR graph and its PGN and prediction sequence for *it didn't take long.*

Figure 3: MRP graph examples and their PGN and prediction formatted expressions. We applied `prop2node` and `embed_label` processors to generate the PGN expressions.

However, PGN only represents a graph structure (namely, all edges in the graph) for simplicity. All node properties are omitted from PGN while we preserve the properties separately. In addition, we reduce redundant meta-tokens appearing in the notation as much as possible. Figure 2 shows the Extended Backus–Naur Form (EBNF)[1] of PGN grammar.

Tree-Like Structure: We employed an essentially tree-like structure[2] because all spanning graphs with a root can be converted into tree-like structures by flipping the directions of appropriate edges. This structure is useful when we convert graphs to PGN.

Left-to-Right Decodable: To make our parser robust, we allow it to convert a notation into a graph in a left-to-right manner. This operation makes us decode an ill-formatted sequence with a *best-effort* strategy. We briefly explain this algorithm in a later sub-section.

[1]https://www.iso.org/obp/ui/#iso:std:iso-iec:14977:ed-1:v1:en

[2]Here we define "tree-like" graph as a graph whose root node is always an ancestor of all nodes in the graph.

3.2 Graph to PGN Conversion

MRP graph to PGN conversion starts with the top nodes. We recursively apply the grammar shown in Figure 2 from parents to children by depth first search. In addition to the depth first search, at finding a next path, we select a child node in increasing order of the numbers of outgoing edges in all descendant nodes (i.e., we select *shallow* branch first) to convert. Since we assume the input graph consists of a tree-like structure, finding children is just extracting out-going edges. However, several frameworks such as EDS, DRG, and AMR may not form a tree-like structure. Thus, we provide an option using all edges instead of out-going edges with flipping edge directions of in-coming edges (we append "-of" suffix to labels of flipped edges). To deal with the reentrancy problem, our recursive search is applied when the node first appears. Here we describe various framework-specific modifications.

Floating Nodes: We found that some EDS and AMR graphs have floating nodes in which no incoming or outgoing edges are annotated. Thus, the PGN grammar supports floating nodes.

Floating Sub-Graphs: We found that some EDS graphs have floating sub-graphs that have no connection to the top. Therefore, we add temporal top nodes for floating sub-graphs to convert all sub-graphs on the basis of the following criteria.

1. First predicate node that has *frame* property, with the first priority.
2. First node that has smallest ID in a sub-graph, with the second priority.

3.3 Left-to-Right Decoding

This left-to-right decoding system consists of a stack and an input stream. Every time a token is fed from the input stream, we take an action of ADD (put the token to the stack), ARC (create an edge between the top two tokens on the stack), POP (pop out the top token from the stack), or CLEAR (pop out all tokens in the stack, and add them to the node list). These actions correspond to `node_id`, `edge_label`, `[EOD]`, and `[EOG]` in Figure 2, respectively.

Since neural networks may produce ill-formatted PGN, the left-to-right decoding finds as many edges as possible.[3] If there is an ill-formatted action such that a PGN sequence terminates with a non-empty stack, we generate additional edges according to the stack state.

3.4 PGN Processors

We define PGN processors, which are a set of invertible functions to apply small modification PGN formatted sequences. Table 1 shows all PGN processors and their description. To better understand these processors, Figure 3 (a2 and b2) shows example PGN formatted graphs of EDS and AMR. Actual PGN expressions are a list of serialized tokens, but here we add indentations for ease of reading. According to the PGN grammar, a `node_id` should be digits representing a node ID, but we insert node labels by the `embed_label` processor. In Figure 3 (a2), there are two flipped edges in the graph to form a tree-like structure, i.e., (_next_a_1, _sell_v_1, ARG1) and (udef_q, _jacket_n_1, BV). Also, Figure 3 (a2 and b2) depicts a larger number of nodes than that in original MRP graphs because node properties are converted into additional nodes by `prop2node` processor, e.g., 6/pres and 7/pl in the EDS graph.

3.5 PGN to Prediction Sequence

Though existing text generation techniques are applicable to generate PGN as is, we provide further modification for PGN expressions to obtain more suitable prediction sequences for a neural decoder. Figure 3 (a3 and b3) shows example prediction sequences derived from PGN. As can be seen, we split a node label into multiple tokens (e.g., subword tokens) and add some special tokens. We add an end-of-node token (`[EON]`) just after the end of subword elements because we should know where the node label token generation terminates. Since `[EON]` is inserted as the end of node token, we can consider `[EON]` as the node's representative token, which will be used for reentrancy classifiers and property classifiers (described later). To handle anchors, we add a place-holder token for anchor starting and ending (`[AS]` and `[AE]`) before node label tokens. When our parser predicts `[AS]` and `[AE]` tokens, we resolve anchors by a biaffine classifier described later. We also add a place-holder token (`[RNT]`) to generate a reentrancy edge after all decoding steps have been completed.

[3]In practice, ill-formatted outputs were very few in experiments.

4 Model

4.1 Problem Formalization

We describe the conceptual formalization similarly to the work of Zhang et al. (2019b). Given an input sequence X (i.e., tokens in the text), we optimize an output sequence $\hat{Y} = \langle y_1, y_2, \dots y_n \rangle$, where y can be represented by a tuple $\langle y^{\mathrm{mode}}, y^{\mathrm{G}}, y^{\mathrm{E}}, y^{\mathrm{L}}, y^{\mathrm{AS}}, y^{\mathrm{AE}} \rangle$,[4] consisting of a model label (y^{mode}), mode-wise labels (y^{G}, y^{E} and y^{L}), and an index of a source-side token for anchoring (y^{AS} and y^{AE}), defined as follows:

$$\hat{Y} = \arg\max_{Y} \prod_{i}^{n} \mathrm{P}(y_i \mid y_{<i}, X).$$

4.2 Overview

To generate the prediction sequence, we provide a sequence-to-sequence model. Figure 4 illustrates an example of AMR parsing (i.e., the graph of Figure 3 (b3)). Our parser is based on a typical encoder-decoder architecture but has several proposed architectures on the decoder side. Given an input text, our parser encodes the tokens by a pre-trained language model (PLM) such as BERT (Devlin et al., 2019). At decoding, a Transformer decoder produces the prediction sequence. To effectively control the decoder, we propose a *mode switching* mechanism. At the i-th decode step, our

[4]We omit reentrancy and property outputs for simplicity.

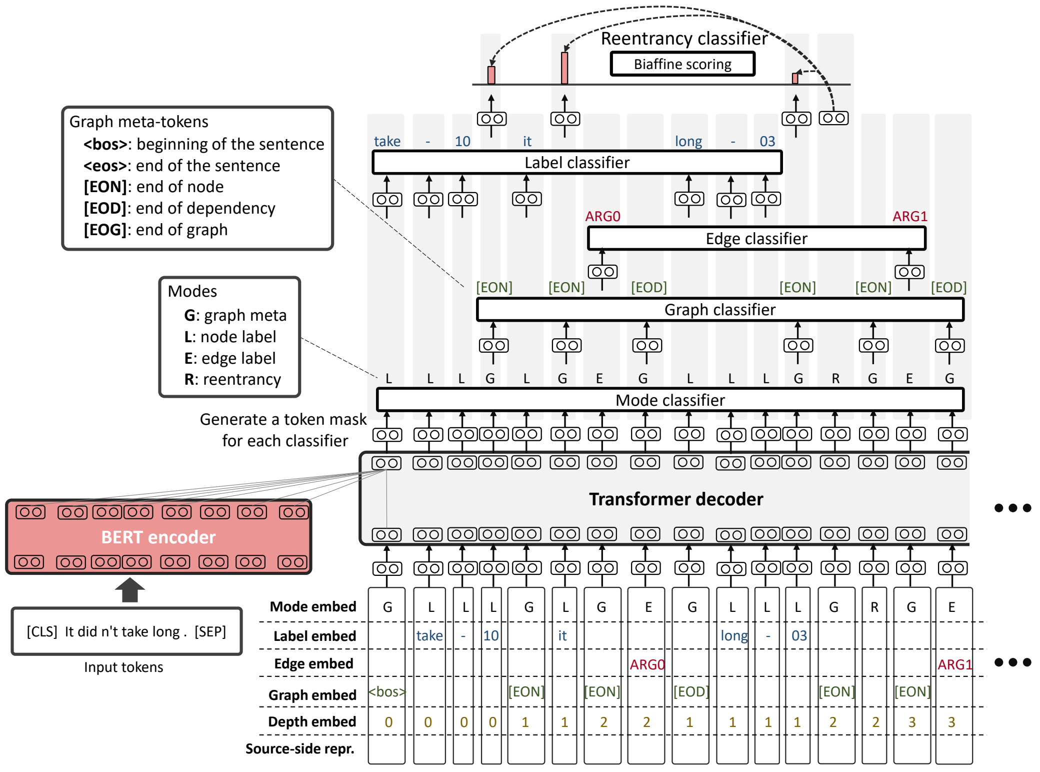


Figure 4: Overview of our proposed parser, showing an example of AMR parsing, assuming PLM = BERT. We encode input tokens that are fed into the decoder. In the decoder, each mode is embedded and the decoder produces an output label for each classifier.

parser decides which mode to execute amongst the following six modes:

G (graph) generates the meta tokens: `<bos>`, `<eos>`, `[EON]`, `[EOD]`, and `[EOG]`.

L (label) generates label tokens of a node, such as *take - 10*.

E (edge) generates an edge label such as *ARG0*.

AS (anchor start) produces an anchoring start representation corresponding to `[AS]`.

AE (anchor end) produces an anchoring end representation corresponding to `[AE]`.

R (reentrancy) generates a place-holder token `[RNT]` that is solved after all decoding steps have been completed.

Then, a classification layer of the selected mode is applied to predict the i-th output. For example, if mode E is selected, an edge classifier on the decoder is used to produce an edge label. If mode AS is selected, an anchoring classifier on the decoder is used to produce an anchor starting index in the encoder's subword tokens. If mode R is selected, we do nothing but generate a place-holder token. Instead, after all decoding steps, we apply biaffine attention, which solves the reentrancy edges for the place-holder tokens.

4.3 Encoder

Given an input text, a PLM-specific tokenizer tokenizes the text into the token sequence X. Note that we insert special tokens such as `[CLS]` and `[SEP]` according to the PLM type. To obtain PLM representations, a layer-wise attention is applied (Kondratyuk and Straka, 2019; Peters et al., 2018):

$$\mathbf{h}_{\mathrm{PLM},i} = c \sum_j \mathtt{PLM}_{ij} \cdot \mathrm{softmax}\left(\mathbf{s}\right)_j,$$

where $\mathbf{s}$ and c are parameters. Note that $\mathbf{h}_{\mathrm{PLM},i} \in \mathbb{R}^{d(\mathrm{PLM})}$, where $d(\mathrm{PLM})$ represents the number of dimensions of the PLM layers. $\mathtt{PLM}_{ij}$ is an embedding of the i-th token in the j-th PLM layer. Note that $1 \leq i \leq N$, where N is the number of tokens.

4.4 Decoder

We employ a Transformer decoder to fully utilize a self-attention mechanism. The decoder includes a switching architecture of *modes* that makes the decoder explicitly learn structural representations.

Decoder Input Representation: For each decoding step i, we compute an input representation for the Transformer decoder:

$$\mathbf{e}_i = \left[\mathbf{e}_i^{\mathrm{mode}}; \mathbf{e}_i^{\mathrm{G}}; \mathbf{e}_i^{\mathrm{L}}; \mathbf{e}_i^{\mathrm{E}}; \mathbf{e}_i^{\mathrm{AS}}; \mathbf{e}_i^{\mathrm{AE}}; \mathbf{e}_i^{\mathrm{depth}}\right],$$

where ; shows a concatenate operation, and each representation is obtained as follows:

$$\begin{aligned}
\mathbf{e}_i^{\text{mode}} &= \textsc{Emb}^{(\text{mode})}\left(y_i^{\text{mode}}\right), \\
\mathbf{e}_i^{\text{G}} &= \textsc{Emb}^{(\text{G})}\left(y_i^{\text{G}}\right), \\
\mathbf{e}_i^{\text{E}} &= \textsc{Emb}^{(\text{E})}\left(y_i^{\text{E}}\right), \\
\mathbf{e}_i^{\text{L}} &= \textsc{Emb}^{(\text{L})}\left(y_i^{\text{L}}\right), \\
\mathbf{e}_i^{\text{AS}} &= W^{(\text{AS})}\mathbf{h}_{\texttt{PLM},k} + \mathbf{b}^{(\text{AS})}, \\
\mathbf{e}_i^{\text{AE}} &= W^{(\text{AE})}\mathbf{h}_{\texttt{PLM},k} + \mathbf{b}^{(\text{AE})}, \\
\mathbf{e}_i^{\text{depth}} &= \textsc{Emb}^{(\text{depth})}\left(y_i^{\text{depth}}\right),
\end{aligned}$$

where $\textsc{Emb}$ is a layer that transfers the label into a fixed sized vector, and W and $\mathbf{b}$ are parameters. The following shows detailed descriptions.

- $\mathbf{e}_i^{\text{mode}}$: This is a representation of the current mode label. $y_i^{\text{mode}} \in \{\text{G}, \text{E}, \text{L}, \text{AS}, \text{AE}, \text{R}\}$ denotes the mode label.
- $\mathbf{e}_i^{\text{G}}, \mathbf{e}_i^{\text{E}}, \mathbf{e}_i^{\text{L}}$: These are representations of a graph meta-token, edge label, and node label, respectively. In turn, y_i^{G}, y_i^{E}, and y_i^{L} represent a mode-wise label. For example, $y_i^{\text{G}} \in \{\texttt{<bos>}, \texttt{<eos>}, \texttt{[EON]}, \texttt{[EOD]}, \texttt{[EOG]}\}$. $y_i^{\text{E}} \in \{\text{ARG0}, \text{ARG1}, \dots\}$ when the target framework is AMR. Note that G, E, and L are selected exclusively and thus zero embeddings are assigned for non-selected modes: for example, if y_i^{mode} is E, $\mathbf{e}_i^{\text{G}} = \mathbf{e}_i^{\text{L}} = \mathbf{0}$.
- $\mathbf{e}_i^{\text{AS}}, \mathbf{e}_i^{\text{AE}}$: These are input embedding of source-side anchors. $W^{(\text{AS})}, W^{(\text{AE})} \in \mathbb{R}^{d(\text{PLM}) \times d(\text{PLM})}$ and $\mathbf{b}^{(\text{AS})}, \mathbf{b}^{(\text{AE})} \in \mathbb{R}^{d(\text{PLM})}$ are trainable parameters. k is an index of the anchor starting token for AS, or an index of the anchor ending token for AE. Therefore, the Transformer decoder draws attentions from the encoder representation of source-side anchored tokens. Note that AS and AE are also exclusive.
- $\mathbf{e}_i^{\text{depth}}$: This is a feature embedding to make the network consider the current depth from the top of the graph. The depth y_i^{depth} is obtained by starting from zero, adding one when `[EON]` appears, and subtracting one when `[EOD]` appears (see Figure 4).

Transformer Decoder: To leverage self-attentions throughout parsing, a multi-layered Transformer decoder (Vaswani et al., 2017) is applied to obtain an output sequence. Let $\mathbf{d}_i$ be a decoder representation at i-th step that is obtained by a multi-layered Transformer decoder where previous decoder inputs $(\mathbf{e}_1 \dots \mathbf{e}_i)$ and encoder representations $(\mathbf{h}_{\texttt{PLM},1} \dots \mathbf{h}_{\texttt{PLM},N})$ are given. We also apply position-encoding (Sukhbaatar et al., 2015; Vaswani et al., 2017; Devlin et al., 2019) for the decoder input representation. By leveraging this decoder, we can consider the entire encoder representation and all decoder inputs throughout the decoding steps.

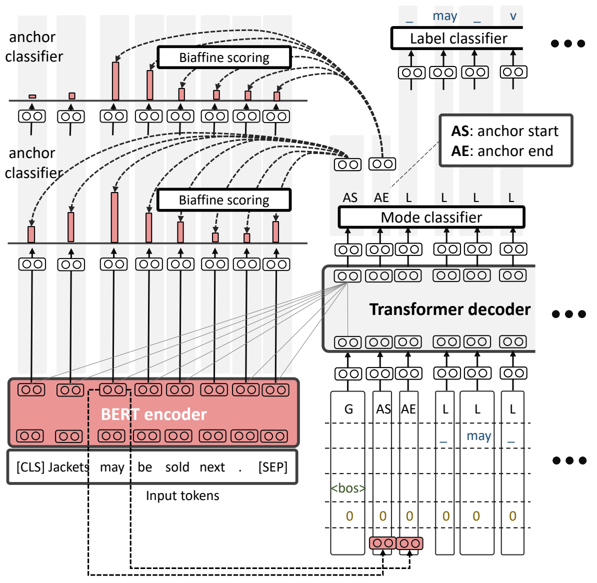


Figure 5: Overview of our proposed parser, showing an example EDS parsing with anchoring prediction.

Mode Output Layers: Given the decoder representation $\mathbf{d}_i$, we produce a probability distribution of the next mode label with a softmax classifier and a feed forward network as follows:

$$\mathrm{P}\left(y_{i+1}^{\text{mode}}\right) = \textsc{Cls}^{(\text{mode})}\left(\mathbf{d}_i\right),$$

where

$$\textsc{Cls}^{(t)}(\mathbf{x}) = \mathrm{softmax}\left(W^{(t)}\textsc{Ffn}^{(t)}\left(\mathbf{x}\right) + \mathbf{b}^{(t)}\right).$$

In the equation, y_{i+1}^{mode} denotes a mode label, i.e., G, E, L, AS, AE, or R. We chose the mode y_{i+1}^{mode} based on the maximum probability. Similarly, we obtain a probability distribution for each mode-wise label as follows:

$$\begin{aligned}
\mathrm{P}\left(y_{i+1}^{\text{G}}\right) &= \textsc{Cls}^{(\text{G})}\left(\mathbf{d}_i\right), \\
\mathrm{P}\left(y_{i+1}^{\text{E}}\right) &= \textsc{Cls}^{(\text{E})}\left(\mathbf{d}_i\right), \\
\mathrm{P}\left(y_{i+1}^{\text{L}}\right) &= \textsc{Cls}^{(\text{L})}\left(\mathbf{d}_i\right),
\end{aligned}$$

After applying these layers above, we obtain output labels y_{i+1}^{mode}, y_{i+1}^{G}, y_{i+1}^{E}, y_{i+1}^{L}, and their corresponding embeddings $\mathbf{e}_{i+1}^{\text{mode}}$, $\mathbf{e}_{i+1}^{\text{G}}$, $\mathbf{e}_{i+1}^{\text{E}}$, $\mathbf{e}_{i+1}^{\text{L}}$, which are used for the next decoder input.

Anchoring Classifier: To compute source-side anchors (i.e., AS and AE), we employ biaffine attention (Dozat and Manning, 2017, 2018). The biaffine operation computes a relation for vector pairs as:

$$\textsc{Biaffine}^{(t)}(\mathbf{x},\mathbf{y}) = \mathbf{x}^\top \mathbf{U}^{(t)}\mathbf{y} + W^{(t)}[\mathbf{x};\mathbf{y}] + b^{(t)},$$

where $\mathbf{U}^{(t)}$, $W^{(t)}$, and $b^{(t)}$ are trainable parameters. We apply the biaffine operation between the decoder representation $\mathbf{d}_i$ and encoder representation $\mathbf{h}_{\mathrm{PLM},j}$ to point a range of anchoring. If $y_{i+1}^{\mathrm{mode}} = \mathrm{AS}$, the anchor starting probability can be represented as

$$\begin{aligned}
\mathbf{h}_i^{(\mathrm{A1})} &= \textsc{Ffn}^{(\mathrm{A1})}\left(\mathbf{d}_i\right),\\
\mathbf{h}_j^{(\mathrm{A2})} &= \textsc{Ffn}^{(\mathrm{A2})}\left(\mathbf{h}_{\mathrm{PLM},j}\right),\\
\mathrm{P}\left(y_{i\to j}^{\mathrm{AS}}\right) &= \sigma\left(\textsc{Biaffine}^{(\mathrm{A})}\left(\mathbf{h}_i^{(\mathrm{A1})},\mathbf{h}_j^{(\mathrm{A2})}\right)\right),
\end{aligned}$$

where σ is a sigmoid function. $\mathrm{P}\left(y_{i\to j}^{\mathrm{AS}}\right)$ represents a probability that the j-th token in the encoder is an anchor starting token. After the output layer above, we draw encoder representations by selecting $\arg\max_{j\in 1,\ldots,N} \mathrm{P}\left(y_{i\to j}^{\mathrm{AS}}\right)$ and its corresponding $\mathbf{h}_{\mathrm{PLM},j}$, which is used as the next decoder input $\mathbf{e}_{i+1}^{\mathrm{AS}}$. Also, $\mathbf{e}_{i+1}^{\mathrm{AE}}$ can be calculated in the same manner.

Figure 5 shows an example of EDS parsing (the graph of Figure 3 (a3)). This example illustrates that AS is raised at the first decoding step, applying biaffine scoring and selecting the encoder's representation of *may* token. Similarly, the AE is raised at the second step.

Reentrancy Classifier: To solve reentrancy edges, we provide another biaffine layer. Given that this is a target-side (i.e., decoder-side) operation, we apply this classifier after all decoding steps have been finished to keep the training speed fast. If $y_{i+1}^{\mathrm{mode}} = \mathrm{R}$, the probability that a reentrancy edge exists between the i and j-th decoding steps can be represented as

$$\begin{aligned}
\mathbf{h}_i^{(\mathrm{R1})} &= \textsc{Ffn}^{(\mathrm{R1})}\left(\mathbf{d}_i\right),\\
\mathbf{h}_j^{(\mathrm{R2})} &= \textsc{Ffn}^{(\mathrm{R2})}\left(\mathbf{d}_j\right),\\
\mathrm{P}\left(y_{i\to j}^{\mathrm{R}}\right) &= \sigma\left(\textsc{Biaffine}^{(\mathrm{R})}\left(\mathbf{h}_i^{(\mathrm{R1})},\mathbf{h}_j^{(\mathrm{R2})}\right)\right).
\end{aligned}$$

To restrict the search space, we only consider the end of node token (i.e., `[EON]`) for j (see Figure 4), since we assume `[EON]` is a representative token of the node.

Property Classifiers: Since we need to classify the properties of a node for PTG graphs, we provide property classifiers on the top of the decoder. Given that we consider `[EON]` as a representative token of a node, we use decoder representations of `[EON]` tokens to classify properties. Therefore, if $y_{i+1}^{\mathrm{G}} =$ `[EON]`, a probability distribution for the property is computed as

$$\mathrm{P}\left(y_{i+1}^{\mathcal{P}}\right) = \textsc{Cls}^{(\mathcal{P})}\left(\mathbf{d}_i\right),$$

where $\mathrm{P}\left(y_{i+1}^{\mathcal{P}}\right)$ represents a probability distribution of the label of property type $\mathcal{P}$ at the `[EON]` (i.e., a node representative token). Note that $\mathcal{P}$ contains a *no label* class where the node is considered to not have the property.

4.5 Loss and Decoding

Loss: We compute a mode output cross-entropy loss $\mathcal{L}_{\mathrm{mode}}$ based on $\mathrm{P}\left(y^{\mathrm{mode}}\right)$. For each mode, we compute mode-specific cross-entropy loss $\mathcal{L}_{\mathrm{G}}$, $\mathcal{L}_{\mathrm{L}}$, and $\mathcal{L}_{\mathrm{E}}$ based on $\mathrm{P}\left(y^{\mathrm{G}}\right)$, $\mathrm{P}\left(y^{\mathrm{L}}\right)$, and $\mathrm{P}\left(y^{\mathrm{E}}\right)$. Note that loss is not computed if a different mode is selected for decode step i. For example, if $y_i^{\mathrm{mode}} = \mathrm{G}$, only $\mathcal{L}_{\mathrm{G}}$ is computed, and others are ignored except the mode loss. Anchoring loss $\mathcal{L}_{\mathrm{AS}}$ and $\mathcal{L}_{\mathrm{AE}}$ are computed based on binary cross-entropy of the $\mathrm{P}\left(y^{\mathrm{AS}}\right)$ and $\mathrm{P}\left(y^{\mathrm{AE}}\right)$. Similarly, reentrancy loss is represented as $\mathcal{L}_{\mathrm{R}}$. If the graph has property tasks (e.g., PTG graphs), we compute the cross-entropy of property label $\mathcal{L}_{\mathcal{P}}$ for property type $\mathcal{P}$. The following equation describes the combined loss to be optimized:

$$\begin{aligned}
\mathcal{L} &= \lambda_{\mathrm{mode}}\mathcal{L}_{\mathrm{mode}} + \lambda_{\mathrm{G}}\mathcal{L}_{\mathrm{G}} + \lambda_{\mathrm{E}}\mathcal{L}_{\mathrm{E}} + \lambda_{\mathrm{L}}\mathcal{L}_{\mathrm{L}}\\
&+ \lambda_{\mathrm{A}}\mathcal{L}_{\mathrm{AS}} + \lambda_{\mathrm{A}}\mathcal{L}_{\mathrm{AE}} + \lambda_{\mathrm{R}}\mathcal{L}_{\mathrm{R}} + \lambda_{\mathrm{P}}\sum_{\mathcal{P}}\mathcal{L}_{\mathcal{P}}.
\end{aligned}$$

where λ are hyperparameters to adjust loss scales.

Decoding: For simplicity, we only consider greedy decoding. We also apply explicit restrictions. For example, the mode AE always comes after AS.

4.6 Ensemble

To further boost the performance of our parser, we provide an average ensemble. We apply mode-wise averaging over output probabilities. Therefore, we average probabilities for the mode layer, mode-specific layers, anchoring classifiers, reentrancy classifier, and property classifiers, respectively.

	Cross-framework					Cross-lingual			
	AMR	EDS	UCCA	PTG	DRG	PTG	UCCA	AMR	DRG
Data preparation									
No. of folds	36	24	12	52	16	52	12	16	8
Reverse edge		✓			✓				✓
`prop2node`	✓	✓						✓	
`embed_label`	✓	✓		✓	✓	✓		✓	✓
`attr2name`			✓	✓		✓	✓		
Hyperparameters									
Hugging Face PLM	roberta -large	roberta -large	roberta -large	roberta -large	roberta -large	bert-base-ml-cased	bert-base-ml-cased	chinese-roberta -wwm-ext-large	bert-base-ml-cased
Encoder dropout	.1	.1	.1	.1	.1	.1	.1	.1	.1
FFN dropout	.1	.1	.1	.1	.1	.1	.1	.1	.1
FFN activation	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU
BIAFFINE dim	400	400	400	400	400	400	400	400	400
Decoder layer, head	6, 4	6, 4	6, 4	6, 4	6, 4	6, 4	8, 4	6, 4	8 ,4
Decoder d_{ff}	256	256	256	256	256	256	256	256	256
Decoder dropout	.1	.1	.1	.1	.1	.1	.1	.1	.1
Depth embedding dim	100	100	100	100	100	100	100	100	100
λ_{mode}	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
λ_G	.137	1.0	.06	.459	.85	.45	.162	.15	.058
λ_E	.137	1.7	.09	.329	.25	.35	.058	.15	.613
λ_L	1.2	1.4	0	1.50	.4	1.5	0	1.5	.538
λ_A	0	2.0	1.73	2.0	0	2.0	2.96	3.0	0
λ_R	3.1	1.0	.50	1.5	2.0	1.5	.241	3.0	.885
λ_P	0	0	0	1.6	0	1.6	0	0	0
Encoder learning rate	1.4e-5	1e-5	1e-5	1e-5	5e-5	5e-5	1.6e-5	5e-6	5.4e-6
Decoder learning rate	5.4e-5	5e-5	5e-5	5e-5	1e-4	1e-4	1.2e-4	1e-4	8.0e-5
Adam beta1, 2	.9,.998	.9,.998	.9,.998	.9,.998	.9,.998	.9,.998	.9,.998	.9,.998	.9,.998
Warmup ratio	.01	.01	.01	.05	.05	.05	.05	.01	.05
Batch size	8	12	8	8	4	4	16	8	32
Maximum epochs	500	500	1000	500	500	500	1000	500	1000
CV ensemble	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 2: Experimental setup of submitted models: data preparation (**top**) and hyperparameter values (**bottom**).

4.7 Post-Processing

We incorporate framework-specific post-processing after reconstruction except for DRG.

For EDS, to support unknown words appearing as a named entity, we replace the `CARG` property with a node label expression extracted by anchors when the edit distance between node label expression and `CARG` is larger than 70% of the node label characters. Since the EDS frame dictionary is available[5], we correct frames by checking their arguments. When several candidates are available, we select the most frequent frame name.

For PTG, frame dictionaries for both English and Czech are also available[6], so we correct frames in the same manner.

For UCCA, we apply post-processing to follow UCCA restrictions. We remove non-anchored nodes appearing as terminal nodes. We also remove self-loop edges. We add `remote` attribute to all edges except primary edges.

For AMR, we replace `date-entity` and `url-entity` with an extracted date and URL from the input by using regular expressions.

5 Experiments

Data Preparation: We first converted all MRP formatted training data into PGN format. Table 2 (top) summarizes the PGN conversion parameters. Since PTG and UCCA utilize tree-like structures, we did not use the reverse edge option, which flips edge directions to form a tree-like structure. AMR graphs form essentially non-tree-like structures; however, given MRP data have been converted to tree-like structures, we did not apply the reverse edge option to AMR graphs. We used `attr2name` on all graphs that have edge attributes (i.e., PTG and UCCA). We also applied `prop2node` to all graphs that have node properties, except PTG graphs.

We divided training data into folds to apply cross-validation (CV) (see No. of folds in Table 2 (top)). We decided a fold size should have about a half number of graphs in the official validation data.

Implementation: We utilize BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019) models as

[5]`http://svn.delph-in.net/erg/tags/1214/etc`
[6]`http://hdl.handle.net/11234/1-1512`

Team	Mean	EDS	PTG	UCCA	AMR	DRG
Hitachi (ours)	**.8642 /1**	**.9356 /1**	**.8873 /1**	.7507 /2	**.8154 /1**	.9319 /2
ÚFAL (Samuel and Straka, 2020)	**.8639 /1**	.9273 /2	.8844 /2	**.7640 /1**	.8023 /2	**.9416 /1**
HIT-SCIR (Dou et al., 2020)	.8106 /3	.8740 /3	.8426 /3	.7476 /3	.6980 /3	.8907 /3
HUJI-KU (Arviv et al., 2020)	.6429 /4	.7968 /5	.5376 /4	.7291 /4	.5236 /5	.6275 /5
ISCAS	.4813 /5	.8586 /4	.1799 /6	.0599 /6	.6148 /4	.6935 /4
TJU-BLCU	.3016 /6	.4904 /6	.2149 /5	.1041 /5	.2996 /6	.3991 /6
JBNU (Na and Min, 2020)	.1323 /-	-	-	-	.6613 /-	-

Table 3: Official MRP results for the cross-framework track (shown as score /rank).

Team	Mean	Czech PTG	German UCCA	Chinese AMR	German DRG
Hitachi (ours)	**.8505 /1**	.8735 /2	.7904 /3	**.8044 /1**	**.9336 /1**
ÚFAL (Samuel and Straka, 2020)	**.8507 /1**	**.9127 /1**	**.8101 /1**	.7817 /2	.8983 /2
HIT-SCIR (Dou et al., 2020)	.6891 /3	.7793 /3	.8002 /2	.4939 /3	.6831 /3
HUJI-KU (Arviv et al., 2020)	.6011 /4	.5849 /4	.7472 /4	.4492 /4	.6233 /4
ISCAS	-	-	-	-	-
TJU-BLCU	.2003 /5	.2171 /5	.0000 /5	.2464 /5	.3377 /5
JBNU (Na and Min, 2020)	-	-	-	-	-

Table 4: Official MRP results for the cross-lingual track (shown as score /rank).

PLMs from Hugging Face's Transformer library (Wolf et al., 2019), which is included on the official 'white-list' of legitimate resources in MRP 2020. RoBERTa large model is used for the cross-framework track because we found in our preliminary experiments that this model generally performs better. In the cross-lingual track, we utilize multi-lingual BERT (Devlin et al., 2019) except for Chinese AMR. Chinese RoBERTa (Cui et al., 2020) is used for the Chinese AMR graphs because the model is carefully tuned for Chinese.

At training, we split network parameters into two groups: one for the encoder and the other for all decoder parameters, applying discriminative fine-tuning (Kondratyuk and Straka, 2019). Each group-specific learning rate is provided and is tuned discriminatively. We select models by evaluating MRP scores on CV and official validation data.

Input texts are tokenized by the PLM-specific tokenizer. We utilized the tokenization scheme of the tokenizer for our decoder: we use the same vocabulary for the node labels y_i^{L} when decoding. The only exception is Czech PTG because node label tokens in Czech PTG graphs include accents that are removed from the vocabulary of multi-lingual BERT. Thus, we employ *character*-level decoding for Czech PTG, where the vocabulary was constructed to contain Czech characters.

Hyperparameters: Hyperparameters of the submitted models are shown in Table 2 (bottom). Adam (Kingma and Ba, 2015) was used as an optimizer, applying linear warmup scheduling. We preliminarily tuned hyperparameters for learning rates, the number of decoder layers, the number of decoder heads, and λ values.

Search ranges of hyperparameters were [1e-6, 1e-3] with log-uniform sampling for decoder learning rate, [1e-6, 1e-4] with log-uniform sampling for encoder learning rate, $[4, 8]$ for Transformer layers and heads, and $[0, 2]$ with uniform sampling for λ. We fixed $\lambda_{\text{mode}} = 1$. In terms of hidden dimensions, we did not aggressively tune them because we preliminarily found their impact on the final performance to be minuscule. In this work, we fixed the biaffine dimensions to 400 and the depth embedding dimensions to 100.

Validation: We used CV to validate our models to ensure robustness. We picked four folds from the training data in Table 2 for each framework/language.[7] For example, although we split the EDS training data into 24 folds, we only used four of these folds to validate the EDS model. We mixed official validation data with each fold to evaluate the model's performance. Then, validation performance was evaluated every 20 epochs, selecting the best model.

Through this validation, we obtained four (i.e., the number of CV folds) trained models for each framework/language with the same hyperparameters. The obtained models were then utilized for the average ensemble.

Setup for Cross-Lingual Pre-Training: Given the lower resource nature of the cross-lingual track, especially for the German UCCA and DRG graphs, we provided two-staged cross-lingual training. First, we concatenated the cross-framework (CF) (e.g., English DRG) and cross-lingual (CL)

[7]This was done to save the computation time.

	EDS	PTG	UCCA	AMR	DRG
roberta-large	**.9101**	**.8779**	**.7692**	**.7776**	**.9080**
bert-base-cased	.8964	.8589	.7473	.7634	.8675

Table 5: Comparison of MRP all-F scores between BERT base and RoBERTa large versions. Scores were evaluated with CV with no ensemble.

(e.g., German DRG) training data. Then, we applied pre-training on the concatenated data for each framework with multi-lingual BERT (Devlin et al., 2019). After that, we applied fine-tuning on only monolingual training data.

5.1 Results and Discussion

Overall Result: Table 3 shows the official cross-framework evaluation results in MRP metrics. As can be seen in the table, in terms of average MRP scores, our parser tied for 1st place: results were very close to the ÚFAL system (Samuel and Straka, 2020).[8] We achieved the top performances on EDS, PTG, and AMR, demonstrating the efficiency for these frameworks. Table 4, the official cross-lingual evaluation results, shows a similar tendency. In the cross-lingual track, we achieved a tie for 1st place, obtaining the best performance for Chinese AMR and German DRG. Notably, our parser performed well on flavor 2 graphs (Oepen et al., 2019) such as AMR and DRG, where no anchors exist in the graphs. This is because we generate node labels directly by the Transformer decoder, thus avoiding alignment errors. However, anchor-based graphs such as UCCA seem unsuitable for our parser when compared to the ÚFAL system. We presume that improving the biaffine scoring in anchoring classifiers would remedy this problem.

Comparing Pre-Trained Models: To better understand how we benefit from PLMs, we compare the `bert-base-cased` and `roberta-large` models. Table 5 shows MRP all-F scores of the cross-framework results. Note that the hyperparameters were slightly different for each model. RoBERTa large models were better than BERT small models, showing improvements ranging from one to four points.

Effectiveness of Depth Embeddings: We conduct an ablation study to examine the role of depth embedding. Table 6 shows a CV-averaged result on English DRG graphs. Note that the hyperparameters are different from Table 5. The result shows

[8]Given randomness nature of the official evaluation tool and statistical significance concerns, system ranking was considered with rounded scores.

	Label	Edge	All
w/ depth embedding	**.8661**	**.9200**	**.9011**
w/o depth embedding	.8611	.9138	.8952

Table 6: Ablation study of depth embedding for DRG's MRP metrics with CV results with no ensemble. We used BERT base in this run.

Language & Framework	Edge	All
German UCCA		
w/ cross-lingual pre-training	**.6942**	**.7791**
w/o cross-lingual pre-training	.6123	.6938
German DRG		
w/ cross-lingual pre-training	**.9479**	**.9044**
w/o cross-lingual pre-training	.9005	.8543

Table 7: Comparison of MRP scores between cross-lingual training and monolingual training. Scores were evaluated through CV by micro-averaging.

that our depth embedding is effective to boost performance. We presume this is because the decoder considers a kind of stack state in PGN, which helps the parser easily produce valid graphs.

Effectiveness of Cross-Lingual Pre-Training: Table 7 shows a comparison of F scores between CL and monolingual training. We used `bert-base-german` instead of `bert-base-ml-cased` for both monolingual trainings. CL training outperformed monolingual training. This indicates that both UCCA and DRG annotations are cross-lingually consistent, and our model can capture the consistency through the CL training. We estimate that our parser has a better transfer ability on cross-lingual graphs.

6 Conclusion

This paper described a novel parser for the shared task on Meaning Representation Parsing 2020. We proposed a text-to-graph-notation transduction that provides a novel graph notation. Our model effectively parsed the graph-notation. Experimental results showed that our parser achieved the top performances in many frameworks. Since our parser is not limited to the five frameworks, in future work we will extend our technique for other tasks.

Acknowledgments

Computational resource of AI Bridging Cloud Infrastructure (ABCI) provided by the National Institute of Advanced Industrial Science and Technology (AIST) was used. We would like to thank the anonymous reviewers for their helpful comments. We also thank Dr. Masaaki Shimizu for the convenience of the computational resources.

References

Omri Abend and Ari Rappoport. 2013. Universal conceptual cognitive annotation (UCCA). In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*.

Ofir Arviv, Ruixiang Cui, and Daniel Hershcovich. 2020. HUJI-KU at MRP 2020: Two transition-based neural parsers. In *Proceedings of the CoNLL 2020 Shared Task: Cross-Framework Meaning Representation Parsing*, pages 73 – 82, Online.

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*.

Johan Bos, Valerio Basile, Kilian Evang, Noortje J. Venhuizen, and Johannes Bjerva. 2017. *The Groningen Meaning Bank*, pages 463–496. Springer.

Wanxiang Che, Longxu Dou, Yang Xu, Yuxuan Wang, Yijia Liu, and Ting Liu. 2019. HIT-SCIR at MRP 2019: A unified pipeline for meaning representation parsing via efficient training and effective encoding. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 76–85, Hong Kong. Association for Computational Linguistics.

Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin, Shijin Wang, and Guoping Hu. 2020. Revisiting pre-trained models for chinese natural language processing. *arXiv preprint arXiv:2004.13922*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Lucia Donatelli, Meaghan Fowlie, Jonas Groschwitz, Alexander Koller, Matthias Lindemann, Mario Mina, and Pia Weißenhorn. 2019. Saarland at MRP 2019: Compositional parsing across all graph-banks. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 66–75, Hong Kong. Association for Computational Linguistics.

Longxu Dou, Yunlong Feng, Yuqiu Ji, Wanxiang Che, and Ting Liu. 2020. HIT-SCIR at MRP 2020: Transition-based parser and iterative inference parser. In *Proceedings of the CoNLL 2020 Shared Task: Cross-Framework Meaning Representation Parsing*, pages 65 – 72, Online.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *the Fifth International Conference on Learning Representations*.

Timothy Dozat and Christopher D. Manning. 2018. Simpler but More Accurate Semantic Dependency Parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*.

Michael Wayne Goodman. 2020. Penman: An open-source library and tool for AMR graphs. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 312–319, Online. Association for Computational Linguistics.

Jan Hajič, Eva Hajičová, Jarmila Panevová, Petr Sgall, Ondřej Bojar, Silvie Cinková, Eva Fučíková, Marie Mikulová, Petr Pajas, Jan Popelka, Jiří Semecký, Jana Šindlerová, Jan Štěpánek, Josef Toman, Zdeňka Urešová, and Zdeněk Žabokrtský. 2012. Announcing Prague Czech-English dependency treebank 2.0. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation*.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2017. A transition-based directed acyclic graph parser for UCCA. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1127–1138, Vancouver, Canada. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015*.

Dan Kondratyuk and Milan Straka. 2019. 75 languages, 1 model: Parsing universal dependencies universally. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2779–2795, Hong Kong, China. Association for Computational Linguistics.

Artur Kulmizev, Miryam de Lhoneux, Johannes Gontrum, Elena Fano, and Joakim Nivre. 2019. Deep contextualized word embeddings in transition-based and graph-based dependency parsing - a tale of two parsers revisited. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2755–2768, Hong Kong, China. Association for Computational Linguistics.

Bin Li, Yuan Wen, Weiguang Qu, Lijun Bu, and Nianwen Xue. 2016. Annotating the little prince with Chinese AMRs. In *Proceedings of the 10th Linguistic Annotation Workshop held in conjunction with ACL 2016 (LAW-X 2016)*, pages 7–15, Berlin, Germany. Association for Computational Linguistics.

Matthias Lindemann, Jonas Groschwitz, and Alexander Koller. 2019. Compositional semantic parsing across graphbanks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4576–4585, Florence, 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*.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. 2018. 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.

Christian Matthiessen and John A Bateman. 1991. *Text generation and systemic-functional linguistics: experiences from English and Japanese*. Pinter Publishers.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 91–98, Ann Arbor, Michigan. Association for Computational Linguistics.

Seung-Hoon Na and Jinwoo Min. 2020. JBNU at MRP 2020: AMR parsing using a joint state model for graph-sequence iterative inference. In *Proceedings of the CoNLL 2020 Shared Task: Cross-Framework Meaning Representation Parsing*, pages 83 – 87, Online.

Stephan Oepen, Omri Abend, Lasha Abzianidze, Johan Bos, Jan Hajič, Daniel Hershcovich, Bin Li, Tim O'Gorman, Nianwen Xue, and Daniel Zeman. 2020. MRP 2020: The Second Shared Task on Cross-framework and Cross-Lingual Meaning Representation Parsing. In *Proceedings of the CoNLL 2020 Shared Task: Cross-Framework Meaning Representation Parsing*, pages 1 – 22, Online.

Stephan Oepen, Omri Abend, Jan Hajic, Daniel Hershcovich, Marco Kuhlmann, Tim O'Gorman, and Nianwen Xue, editors. 2019. *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*. Association for Computational Linguistics, Hong Kong.

Stephan Oepen and Jan Tore Lønning. 2006. Discriminant-based MRS banking. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation*.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

David Samuel and Milan Straka. 2020. ÚFAL at MRP 2020: Permutation-invariant semantic parsing in PERIN. In *Proceedings of the CoNLL 2020 Shared Task: Cross-Framework Meaning Representation Parsing*, pages 53 – 64, Online.

Sainbayar Sukhbaatar, arthur szlam, Jason Weston, and Rob Fergus. 2015. End-to-end memory networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2440–2448. Curran Associates, Inc.

Rob A. Van Der Sandt. 1992. Presupposition Projection as Anaphora Resolution. *Journal of Semantics*, 9(4):333–377.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

David Vilares and Carlos Gómez-Rodríguez. 2018. A transition-based algorithm for unrestricted AMR parsing. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 142–149, New Orleans, Louisiana. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. HuggingFace's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the Eighth International Conference on Parsing Technologies*, pages 195–206, Nancy, France.

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

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

Yue Zhang, Wei Jiang, Qingrong Xia, Junjie Cao, Rui Wang, Zhenghua Li, and Min Zhang. 2019c. SUDA-Alibaba at MRP 2019: Graph-based models with BERT. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 149–157, Hong Kong. Association for Computational Linguistics.