HIT-SCIR at MRP 2020: Transition-based Parser and Iterative Inference Parser

Longxu Dou, Yunlong Feng, Yuqiu Ji, Wanxiang Che, Ting Liu
Research Center for Social Computing and Information Retrieval
Harbin Institute of Technology, China
{lxdou,ylfeng,yqji,car,tliu}@ir.hit.edu.cn

Abstract

This paper describes our submission system (HIT-SCIR) for the *CoNLL 2020 shared task: Cross-Framework and Cross-Lingual Meaning Representation Parsing*. The task includes five frameworks for graph-based meaning representations, *i.e.*, UCCA, EDS, PTG, AMR, and DRG. Our solution consists of two subsystems: transition-based parser for Flavor (1) frameworks (UCCA, EDS, PTG) and iterative inference parser for Flavor (2) frameworks (DRG, AMR). In the final evaluation, our system is ranked 3^{rd} among the seven team both in Cross-Framework Track and Cross-Lingual Track, with the macro-averaged MRP F1 score of 0.81/0.69.

1 Introduction

The goal of the CoNLL 2020 shared task (Oepen et al., 2020) is to develop a unified parsing system to process all five semantic graph banks across different languages. This task combines five frameworks for graph-based meaning representation, each with its specific formal and linguistic assumptions, including UCCA, EDS, PTG, AMR, and DRG. [1]

In the context of the shared task, the organizers distinguish different flavors of semantic graphs based on the nature of the relationship they assume between the linguistic surface string and the nodes of the graph. They call this relation *anchoring*. Therefore, the involved five frameworks could be divided into two classes: (a) **Flavor (1)**, including UCCA, EDS, and PTG, allowing arbitrary parts of the sentence as node anchors, as well as multiple nodes anchored to overlapping sub-strings, and (b) **Flavor (2)**, including AMR and DRG, not considering the correspondence between nodes and the surface tokens.

[1] See http://mrp.nlpl.eu/ for further technical details, information on how to obtain the data, and official results.

Our submitted system could be summarized in the following:

- **Transition-based Parser for Flavor (1)**

  Following Che et al. (2019), the top system in CoNLL 2019 shared task (Oepen et al., 2019), we employ the transition-based parser for Flavor (1) frameworks since it's very flexible in predicting the anchor information. We directly use their parser for UCCA and EDS. And we design a new parser for PTG.

- **Iterative Inference Parser for Flavor (2)**

  Recently, Cai and Lam (2020) proposed Graph⇔Sequence Iterative Inference system for AMR parsing, which treats parsing as a series of dual decisions on the input sequence and the incrementally constructed graph, achieving state-of-the-art results. We adopt their model for Flavor (2) frameworks (AMR, DRG).

- **Pretrained Language Model**

  Our systems benefit a lot from the pretrained language models, *i.e.*, BERT (Devlin et al., 2019), ELECTRA (Clark et al., 2020) and XLM-RoBERTa (Conneau et al., 2020).

2 Background

In the following, we will give a brief introduction to these frameworks and our corresponding solutions.

Universal Conceptual Cognitive Annotation (UCCA) is a multi-layer linguistic framework (Flavor (1)) firstly proposed by Abend and Rappoport (2013), which treats input words as terminal nodes. The non-terminal node might govern one or more nodes, which may be discontinuous. Moreover, one node can have multiple governing (parent) nodes through multiple edges, consisting of a single primary edge and other remote edges. Relationships

Proceedings of the CoNLL 2020 Shared Task: Cross-Framework Meaning Representation Parsing, pages 65–72
Online, Nov. 19-20, 2020.

between nodes are represented by edge labels. The primary edges form a tree structure, whereas the remote edges introduce reentrancy, forming directed acyclic graphs (DAGs). We directly employ the system of Che et al. (2019), which achieves the 1^{st} at CoNLL 2019 shared task.

Elementary Dependency Structure (EDS) is a graph-structured semantic representation formalism (Flavor (1)) proposed by Oepen and Lønning (2006). Che et al. (2019) introduce a neural encoder-decoder transition-based parser for the EDS graph, which extracts the node alignment (or anchoring) information effectively.

Prague Tectogrammatical Graphs (PTG) are graph-structured multi-layered semantic representation formalism (Flavor (1)) proposed by Zeman and Hajič (2020). PTG graphs essentially recast core predicate–argument structure in the form of mostly anchored dependency graphs, albeit introducing 'empty' (or generated, in FGD terminology) nodes, for which there is no corresponding surface token. We didn't find any existing parser for PTG. Thus we design a list-based arc-eager transition-based parser for PTG.

Abstract Meaning Representation (AMR), proposed by Banarescu et al. (2013), is a broad-coverage sentence-level semantic formalism (Flavor (2)) used to encode the meaning of natural language sentences. AMR can be regarded as a rooted labeled directed acyclic graph. Nodes in AMR graphs represent concepts and labeled directed edges are relations between the concepts. We directly employ state-of-the-art parser of Cai and Lam (2020).

Discourse Representation Graphs (DRG) are proposed by Abzianidze et al. (2020) (Flavor (2)), which are derived from the DRS annotations in the Parallel Meaning Bank (Bos et al., 2017; Abzianidze et al., 2017). Its concepts are represented by WordNet 3.0 (Fellbaum, 1998) senses and semantic roles by the adapted version of VerbNet (Schuler, 2006) roles. Similar to PTG, we don't find any existing parser for DRG, thus we modify the AMR parser to process the DRG.

3 Transition-based Parser for Flavor (1)

3.1 Background

A tuple (S, L, B, E, V) is used to represent the parsing state, where *S* is a stack holding processed words, *L* is a list holding words popped out of *S* that will be pushed back in the future, and *B* is a buffer holding unprocessed words. *E* is a set of labeled dependency arcs. *V* is a set of graph nodes including concept nodes and surface tokens. The initial state is $([0], [\,], [1, \cdots, n], [\,], V)$, where *V* only contains surface tokens, whereas the concept nodes will be generated during parsing. And the terminal state is $([0], [\,], [\,], E, V^{'})$. We model the *S*, *L*, *B* and action history with Stack-LSTM, which supports PUSH and POP operations. [2]

3.2 Transition Systems

For brevity, we omit the descriptions of the transition system for UCCA and EDS (Che et al., 2019). As for PTG, we propose a new arc-eager transition-based parser. To illustrate the transition-set and configuration more clearly, we list the transition process in Table 1 (UCCA), Table 2 (EDS) and Table 3 (PTG).

3.2.1 PTG

We are not aware of any parser specifically designed for PTG. But we found it is highly related to UCCA and EDS. Thus, based on the transition systems of UCCA and EDS and the definition of PTG (Čmejrek et al., 2004), we design a new system for PTG as shown in Table 3. x is the top element in the buffer and y is the top element in the stack. Moreover, y could only be a concept node (stack and list only contain concept nodes), and x could be a concept node or a surface token.

LEFT-EDGE$_X$, RIGHT-EDGE$_X$, SHIFT, DROP, REDUCE, PASS and FINISH are the same as EDS.

- SELF-EDGE$_X$ adds an arc with label X between x and itself.
- TERMINAL-NOLABEL creates new non-terminal nodes without label. These nodes have corresponding surface token(s) and their labels will be determined by rule, which will be introduced in Section 5.
- TERMINAL$_X$ creates new non-terminal nodes with label X.
- NODE$_X$ creates a new node on the buffer as a parent of the first element on the stack, with X as its label.
- NODE-ROOT$_X$ creates a new node on the buffer as a child of the root with X as its label.

[2] We recommend reading Dyer et al. (2015) for more details.

Before Transition				Transition	After Transition				Condition
Stack	**Buffer**	**Nodes**	**Edges**		**Stack**	**Buffer**	**Nodes**	**Edges**	
S	$x \mid B$	V	E	SHIFT	$S \mid x$	B	V	E	
$S \mid x$	B	V	E	REDUCE	S	B	V	E	
$S \mid x$	B	V	E	NODE$_X$	$S \mid x$	$y \mid B$	$V \cup \{y\}$	$E \cup \{(y,x)_X\}$	$x \neq \text{root}$
$S \mid y, x$	B	V	E	LEFT-EDGE$_X$	$S \mid y, x$	B	V	$E \cup \{(x,y)_X\}$	$\begin{cases} x \notin w_{1:n}, \\ y \neq \text{root}, \\ y \not\rightsquigarrow_G x \end{cases}$
$S \mid x, y$	B	V	E	RIGHT-EDGE$_X$	$S \mid x, y$	B	V	$E \cup \{(x,y)_X\}$	
$S \mid y, x$	B	V	E	LEFT-REMOTE$_X$	$S \mid y, x$	B	V	$E \cup \{(x,y)^*_X\}$	
$S \mid x, y$	B	V	E	RIGHT-REMOTE$_X$	$S \mid x, y$	B	V	$E \cup \{(x,y)^*_X\}$	
$S \mid x, y$	B	V	E	SWAP	$S \mid y$	$x \mid B$	V	E	$\text{i}(x) < \text{i}(y)$
[root]	$\emptyset$	V	E	FINISH	$\emptyset$	$\emptyset$	V	E	

Table 1: The transition set of UCCA parser. We write the **Stack** with its top to the right and the **Buffer** with its head to the left. $(\cdot,\cdot)_X$ denotes a primary X-labeled edge, and $(\cdot,\cdot)^*_X$ a remote X-labeled edge. $\text{i}(x)$ is a running index for the created nodes. In addition to the specified conditions, the prospective child in an EDGE transition must not already have a primary parent. From (Hershcovich et al., 2017).

Before Transition					Transition	After Transition					Condition
Stack	**List**	**Buffer**	**Nodes**	**Edges**		**Stack**	**List**	**Buffer**	**Nodes**	**Edges**	
S	L	$x \mid B$	V	E	SHIFT	$S \mid L \mid x$	$\emptyset$	B	V	E	concept(x)
$S \mid x$	L	B	V	E	REDUCE	S	L	B	V	E	
$S \mid x$	L	$y \mid B$	V	E	RIGHT-EDGE$_X$	$S \mid x$	L	$y \mid B$	V	$E \cup \{(x,y)_X\}$	concept(x) $\wedge$ concept(y)
$S \mid y$	L	$x \mid B$	V	E	LEFT-EDGE$_X$	$S \mid y$	L	$x \mid B$	V	$E \cup \{(x,y)_X\}$	concept(x) $\wedge$ concept(y)
$S \mid x$	L	B	V	E	PASS	S	$x \mid L$	B	V	E	
S	L	$x \mid B$	V	E	DROP	$S \mid L$	$\emptyset$	B	V	E	token(x)
S	L	$x \mid B$	V	E	TOP	S	L	$x \mid B$	$V \cup \text{Top}(x)$	E	concept(x)
S	L	$x \mid B$	V	E	NODE-START$_X$	$S \mid y$	L	$x \mid B$	$V \cup \{y_{start=x,label=X}\}$	E	token(x)
$S \mid y$	L	$x \mid B$	V	E	NODE-END	$S \mid y$	L	$x \mid B$	$V \cup \{y_{end=x}\}$	E	token(x)
[root]	$\emptyset$	$\emptyset$	V	E	FINISH	$\emptyset$	$\emptyset$	$\emptyset$	V	E	

Table 2: The transition set of EDS parser. We write the **Stack** with its top to the right, the **Buffer** with its head to the left and the **List** with its head to the left. The elements in **Stack** and **List** are all concept nodes. Indicator function token(x) means x is a token of the sentence, while concept(x) means it's a concept node. Top(x) indicates x is the top node. $y_{start=w_i,label=X,end=w_j}$ indicates the alignments of concept node y is starting at token w_i, ending at token w_j and its label is X.

Before Transition					Transition	After Transition					Condition
Stack	**List**	**Buffer**	**Nodes**	**Edges**		**Stack**	**List**	**Buffer**	**Nodes**	**Edges**	
S	L	$x \mid B$	V	E	SHIFT	$S \mid L \mid x$	$\emptyset$	B	V	E	concept(x)
$S \mid x$	L	B	V	E	REDUCE	S	L	B	V	E	
$S \mid x$	L	$y \mid B$	V	E	RIGHT-EDGE$_X$	$S \mid x$	L	$y \mid B$	V	$E \cup \{(x,y)_X\}$	
$S \mid y$	L	$x \mid B$	V	E	LEFT-EDGE$_X$	$S \mid y$	L	$x \mid B$	V	$E \cup \{(x,y)_X\}$	
$S \mid y$	L	$x \mid B$	V	E	SELF-EDGE$_X$	$S \mid y$	L	$x \mid B$	V	$E \cup \{(x,x)_X\}$	
$S \mid x$	L	B	V	E	PASS	S	$x \mid L$	B	V	E	
S	L	$x \mid B$	V	E	DROP	$S \mid L$	$\emptyset$	B	V	E	token(x)
S	L	$x \mid B$	V	E	NODE$_X$	S	L	$y \mid x \mid B$	$V \cup \{y_{label=X}\}$	E	token(x)
S	L	$x \mid B$	V	E	NODE-ROOT$_X$	S	L	$y \mid x \mid B$	$V \cup \{y_{label=X}\}$	E	root(x)
S	L	$x \mid B$	V	E	TERMINAL-NOLABEL	S	L	$y \mid x \mid B$	$V \cup \{y\}$	E	
S	L	$x \mid B$	V	E	TERMINAL$_X$	S	L	$y \mid x \mid B$	$V \cup \{y_{label=X}\}$	E	
[root]	$\emptyset$	$\emptyset$	V	E	FINISH	$\emptyset$	$\emptyset$	$\emptyset$	V	E	

Table 3: The transition set of PTG parser. We write the **Stack** with its top to the right, the **Buffer** with its head to the left and the **List** with its head to the left. The elements in **Stack** and **List** are all concept nodes. Indicator function token(x) means x is a token of the sentence, while concept(x) means it's a concept node. root(x) indicates x is the top node.

The differences between TERMINAL and NODE are (a) TERMINAL generates the node ahead of NODE in the oracle transition sequence, which means the nodes generated by TERMINAL are more closer to the surface tokens, and (b) TERMINAL could generate the concept node that aligns to surface tokens(s) while NODE could only generates the node with the particular label.

4 Iterative Inference Parser for Flavor (2)

4.1 Overview

We adopt the Graph⇔Sequence Iterative Inference system proposed by Cai and Lam (2020) to parse the Flavor (2) graphs. We name it iterative inference parser in this paper. At each time step, the model performs multiple rounds of attention, reasoning, and composition that aim to answer two critical questions: (a) which part of the input sequence to abstract, and (b) where in the output graph to construct the new concept.

4.2 Implementation

This model consists of four modules:

- **Sentence Encoder** encodes the input sequence and generates a set of text memories to provide grounding for concept node generation.
- **Graph Encoder** encodes the partial graph and generate a set of graph memories to provide grounding for relation prediction.
- **Concept Solver** uses the graph hypothesis for concept node generation.
- **Graph Solver** uses the concept node hypothesis for relation prediction.

The last two components correspond to the reasoning functions $g(\cdot)$ and $f(\cdot)$ respectively.

More specifically, at the beginning of parsing, Sentence Encoder computes the text memories, while Graph Encoder constructs the graph memories incrementally.[3] During the iterative inference, a semantic representation of the current state is used to attend to both graph and text memories to locate the new concept and obtain its relations to the existing graph, both of which subsequently refine each other. Then in each step, Concept Solver generates the concept node and Relation Solver predicts the relation between the concept node with other node(s) through attention.[4]

This process could be described as follows:

$$y_t^i = g(G^i, x_t^i),$$
$$x_{t+1}^i = f(W, y_t^i),$$

where W and G^i are the input sentence and the current semantic graph respectively. $g(\cdot)$ looks for where to construct (edge prediction) and $f(\cdot)$ seeks what to abstract (node prediction) respectively. The x_t^i, y_t^i are the $t-$th graph hypothesis and the $t-$th sequence hypothesis for the $i-$th expansion step respectively.

In summary, Iterative Inference Parser uses a sentence encoder to encode the input sequence and a graph encoder to build the graph iteratively. In each step, the parser uses the graph state and the sentence representation to generate a new concept node and build the relation between the concept node and other parts of the graph.[5]

5 Pre-processing and Post-processing

In this session, we introduce the pre-processing and post-processing work. The MRP graph can be broken down into seven component pieces: top nodes, node labels, node properties, node anchoring, directed edges, edge labels, and edge attributes.

The directed edges, edge labels, and node id form the standard input of our system. For node anchoring, we directly derive the anchoring information through segmentation from companion data. For other elements, such as top nodes, are a bit different among the frameworks. We will introduce these framework-specific work in the following.

5.1 UCCA

Top Nodes There is only one top node for each sentence in UCCA, which is used to initialize the stack. Meanwhile, the top node is the protected symbol of the stack (which will never be popped out).

Edge Properties The edge property in UCCA is used as the sign for remote edges. We treat remote edges in the same way as primary edges, except for those with a special star (*) symbol.

[3] In the beginning, we represent the empty graph with a special symbol: BOG (begin of graph).

[4] The iterative process between concept node generation and relation prediction will last until the concept solver predicts a special symbol: EOG (end of graph).

[5] More details could be found in the original paper (Cai and Lam, 2020).

Node Anchoring Referring to the original UCCA framework design, we link the node in the foundational layer to the surface token with the edge label 'Terminal'. In post-processing, we combine surface tokens and foundational layer nodes via collapsing 'Terminal' edge to extract the anchor information.

5.2 EDS

Top Nodes The TOP operation will set the first concept node in the buffer as top node.

Node Labels We train a tagger to predict the node labels. The tagger is directly adopted from AllenNLP. Although there are thousands of node labels, the result shows our system performs well on this.

Node Properties We count the co-occurrence of node_label, upos, dep and property value in the training dataset, and select the property based on the co-occurrence statistics in the predicting procedure. If the triple (node_label, upos, dep) is not found, we backoff to the tuple (upos, dep).[6]

Node Anchoring We obtain alignment information through NODE_START and NODE_END operation.

5.3 PTG

The original PTG is not a directed acyclic graph (DAG). We find that all the cycles are caused by *coref.gram* edges. Thus we reverse these edges to convert PTG to DAG to avoid this problem.

Top Nodes The top node of PTG is the root of the stack, which is used to initialize the stack.

Node Labels We obtain the node label through a rule-based method: (1) collecting the tokens that node aligns to, and (2) setting the node label as the lemma of one of the tokens. Which token to be chosen is determined by a rule considering the pos-tag, dependency tree.

Node Properties Similar to EDS, we used the statistic-based method to compute the properties.

Node Anchoring We obtain alignment information through NODE or TERMINAL operation.

[6]The 'upos' and 'dep' are obtained from corresponding companion data, which refers to the upos-tag of the token(s) that the concept node aligns to, and the label of the edge between the token(s) and its parents in the dependency tree.

5.4 AMR

The original definition of AMR consists of TOP NODES, NODE LABELS and NODE PROPERTIES. Thus we directly used the system's output.

5.5 DRG

The DRG parsing system is based on the parser of AMR (Cai and Lam, 2020). Before training, we need to convert DRG to AMR. After prediction, we recover it reversely. More specifically, we attach the *unreal* label to the unlabeled node of DRG to satisfy the requirement of AMR that all nodes must have a label. The labeled node of DRG can be divided into three types:

- **Abstract Node** Such nodes are representing comparison relations, *e.g.*, EQU, which is usually uppercased.
- **Labeled Node with property**: We split the node into the label part and property parts. For example, impossible.a.01 could be spilt into label part (impossible) and property parts (a and 01). Then we add two edges with the label 'op' from label part to property parts.
- **Labeled Node without property**: Such node are covered by double quotes, like "now". We simply copy the node label to the AMR graph ignoring the double quotes. We will add them back during post-processing.

6 Experiments

In this section, we will show the basic model setup and overall evaluation results.

6.1 Model Setup

Transition-based Parser Based on the system of Che et al. (2019), we build the transition-based parser for UCCA, EDS, and PTG. We split parameters into two groups, *i.e.*, BERT parameters and other parameters (base parameters). The two parameter groups differ in the learning rate. For training, we use the Adam optimizer (Kingma and Ba, 2015).

Iterative Inference Parser Based on the system of Cai and Lam (2020), we build the parser for AMR and DRG.[7]

Code for our parser and model weights are available at https://github.com/DreamerDeo/HIT-SCIR-CoNLL2020.

[7]https://github.com/jcyk/AMR-gs

System	UCCA	EDS	PTG	AMR	DRG	ALL
Hitachi	0.75	0.94	0.89	0.82	0.93	0.86
UFAL	0.76	0.93	0.88	0.80	0.94	0.86
HIT-SCIR	0.75	0.87	0.84	0.70	0.89	0.81
HUJI-KU	0.73	0.80	0.54	0.52	0.63	0.64
ISCAS	0.06	0.86	0.18	0.61	0.69	0.48
TJU-BLCU	0.10	0.49	0.21	0.3	0.40	0.30

Table 4: Evaluation results on Cross-Framework Track upon MRP F1.

System	UCCA	PTG	AMR	DRG	ALL
UFAL	0.81	0.91	0.78	0.90	0.85
Hitachi	0.79	0.87	0.80	0.93	0.85
HIT-SCIR	0.80	0.78	0.49	0.68	0.69
HUJI-KU	0.75	0.58	0.45	0.62	0.60

Table 5: Evaluation results on Cross-Lingual Track upon MRP F1.

6.2 Fine-Tuning BERT with Parser

Based on Devlin et al. (2019), fine-tuning BERT with supervised downstream task will receive the most benefit. So we choose to fine-tune BERT model together with the original parser. In our preliminary study, gradual unfreezing and slanted triangular learning rate scheduler is essential for BERT fine-tuning model.

We find it beneficial to warm up the learning rate at beginning of training progress and cool down after. With the slanted triangular learning rate scheduler, the learning rate increases linearly from $lr/ratio$ to lr during the first $num\_step \times cut\_frac$ steps and decreases linearly back to $lr/ratio$ during the left steps.

Gradual unfreezing is also used during training so in the first few ($1 \sim 5$) epochs BERT parameters are frozen. While being gradually unfrozen, the learning rate experiences a full warm-up and cool-down cycle per epoch. And then a full cycle is performed during the rest training progress once all parameters are unfrozen. Batch normalization (Sergey and Christian, 2015) is useful when avoiding the gradient exploding during training UCCA and EDS.

6.3 Hyperparameters

Pretrained Language Model The model choices for each framework across two tracks, are listed in Table 6.

Transition-based Parser Following Che et al. (2019), we adopt the hyper-parameters listed in Table 7.

Iterative Inference Parser Following Cai and Lam (2020), we perform $N = 4$ steps of iterative inference. Other hyper-parameter settings can be found in the Table 8.

6.4 Overall Evaluation Results and Analysis

We list the evaluation results on Table 4 and Table 5, which is ranked by the cross-framework metric, named macro-averaged MRP F1.[8]

For Flavor (1) framework, our transition-based parser is a local-decision model. Thus, our parser cannot effectively use global information when predicting the attributes of graph nodes, resulting in some more complex structures that can not be effectively generated. On another hand, our system in UCCA achieves nearly state-of-the-art performance but falls behind in EDS and PTG. We argue that the main reason is that we obtain the properties by the statistic-based way instead of training a specific model, which lower the overall performance in EDS and PTG.

For Flavor (2) framework, we suppose that the main reason for the performance degradation in AMR is the entity vocabulary, which is obtained in AMR2.0 from Cai and Lam (2020) and doesn't match the MRP2020-AMR very well. Based on our experience, the AMR parser will benefit a lot from a high-quality entity vocabulary.

7 Conclusion

In this paper, we describe our submission system for the CoNLL 2020 shared task. We separate

[8]Evaluation results of CoNLL 2020 shared task are available at http://bit.ly/cfmrp20.

Track	Cross-Framework	Cross-Lingual
UCCA	BERT-Base	BERT-Base German
EDS	BERT-Base	-
PTG	BERT-Base	BERT-Base Multilingual
AMR	ELECTRA-Large	ELECTRA-Large Chinese
DRG	ELECTRA-Large	XLM-RoBERTa-Large

Table 6: The pretrained language model used in each track .

.

HYPERPARAMETER	VALUE
Hidden dimension	200
Action dimension	50
Optimizer	Adam
β_1, β_2	0.9, 0.99
Dropout	0.5
Layer dropout	0.2
Recurrent dropout	0.2
Input dropout	0.2
Batch size	16
Epochs	50
Base learning rate	1×10^{-3}
BERT learning rate	5×10^{-5}
Gradient clipping	5.0
Gradient norm	5.0
Learning rate scheduler	slanted triangular
Gradual Unfreezing	True
Cut Frac	0.1
Ratio	32

Table 7: Transition-based parser hyper-parameters settings.

HYPERPARAMETER	VALUE
lemma dimension	300
NER dimension	16
POS dimension	32
concept dimension	300
char dimension	32
CNN	
filters	256
filter size	3
output size	128
Transformer	
heads	8
hidden size	512
feed-forward hidden size	1024
Sentence Encoder transformer layers	4
Graph Encoder transformer layers	2
Concept Solver feed-forward hidden size	1024
Releation Solver feed-forward hidden size	1024
Releation Solver feed-forward heads	8
Deep biaffine classifier hidden size	100

Table 8: Iterative Inference parser hyper-parameters settings.

our solutions into two classes based on the flavor of the framework: (a) transition-based parser for Flavor (1) (UCCA, EDS, PTG), and (b) iterative inference parser for Flavor (2) (AMR, DRG). Especially, we propose a new transition-based parser for the PTG framework, which achieves comparable performance. In the final evaluation, our system positions at 3^{rd} in both tracks.

Acknowledgments

We thank the reviewers for their insightful comments and the HIT-SCIR colleagues for the coordination on the machine usage. This work was supported by the National Natural Science Foundation of China (NSFC) via grant 61976072, 61632011 and 61772153.

References

Omri Abend and Ari Rappoport. 2013. Universal conceptual cognitive annotation (UCCA). In *ACL*, pages 228–238.

Lasha Abzianidze, Johannes Bjerva, Kilian Evang, Hessel Haagsma, Rik van Noord, Pierre Ludmann, Duc-Duy Nguyen, and Johan Bos. 2017. The parallel meaning bank: Towards a multilingual corpus of translations annotated with compositional meaning representations. In *EACL*, pages 242–247.

Lasha Abzianidze, Johan Bos, and Stephan Oepen. 2020. DRS at MRP 2020: Dressing up Discourse Representation Structures as graphs. In *Proceedings of the CoNLL 2020 Shared Task: Cross-Framework Meaning Representation Parsing*, pages 23 – 32, 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*, pages 178–186.

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

Deng Cai and Wai Lam. 2020. AMR parsing via graph-

sequence iterative inference. In *ACL*, pages 1290–1301.

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 Computational Natural Language Learning*, pages 76 – 85, Hong Kong, China.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *ICLR*.

Martin Čmejrek, Jan Cuřín, and Jiří Havelka. 2004. Prague Czech-English dependency treebank: Any hopes for a common annotation scheme? In *HLT-NAACL*, pages 47–54.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *ACL*, pages 8440–8451.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, pages 4171–4186.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *ACL and IJCNLP*, pages 334–343.

Christiane Fellbaum. 1998. *WordNet: An Electronic Lexical Database*. Bradford Books.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2017. A transition-based directed acyclic graph parser for UCCA. In *ACL*, pages 1127–1138.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR*.

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 Hajič, Daniel Hershcovich, Marco Kuhlmann, Tim O'Gorman, Nianwen Xue, Jayeol Chun, Milan Straka, and Zdeňka Urešová. 2019. MRP 2019: Cross-framework Meaning Representation Parsing. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Computational Natural Language Learning*, pages 1 – 27, Hong Kong, China.

Stephan Oepen and Jan Tore Lønning. 2006. Discriminant-based MRS banking. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*.

Karin Kipper Schuler. 2006. *VerbNet: A Broad-Coverage, Comprehensive Verb Lexicon*. Ph.D. thesis.

Ioffe Sergey and Szegedy Christian. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*.

Daniel Zeman and Jan Hajič. 2020. FGD at MRP 2020: Prague Tectogrammatical Graphs. In *Proceedings of the CoNLL 2020 Shared Task: Cross-Framework Meaning Representation Parsing*, pages 33 – 39, Online.