OIE@OIA: an Adaptable and Efficient Open Information Extraction Framework

Xin Wang, Minlong Peng, Mingming Sun, Ping Li
Cognitive Computing Lab
Baidu Research
No.10 Xibeiwang East Road, Beijing 100193, China
10900 NE 8th St. Bellevue, Washington 98004, USA
{wangxin60, pengminlong, sunmingming01, liping11}@baidu.com

Abstract

Different Open Information Extraction (OIE) tasks require different types of information, so the OIE field requires strong adaptability of OIE algorithms to meet different task requirements. This paper discusses the adaptability problem in existing OIE systems and designs a new adaptable and efficient OIE system - OIE@OIA as a solution. OIE@OIA follows the methodology of Open Information eXpression (OIX): parsing a sentence to an Open Information Annotation (OIA) Graph and then adapting the OIA graph to different OIE tasks with simple rules. As the core of our OIE@OIA system, we implement an end-to-end OIA generator by annotating a dataset (we make it open available) and designing an efficient learning algorithm for the complex OIA graph. We easily adapt the OIE@OIA system to accomplish three popular OIE tasks. The experimental show that our OIE@OIA achieves new SOTA performances on these tasks, showing the great adaptability of our OIE@OIA system. Furthermore, compared to other end-to-end OIE baselines that need millions of samples for training, our OIE@OIA needs much fewer training samples (12K), showing a significant advantage in terms of efficiency.

1 Introduction

Open Information Extraction (OIE) techniques are gradually attracting more and more attention (Christensen et al., 2011; Mausam et al., 2012; Corro and Gemulla, 2013; Angeli et al., 2015; Bhutani et al., 2016; Cui et al., 2018; Roy et al., 2019; Zhan and Zhao, 2020) as they build a bridge between language to knowledge. OIE tasks are generally designed for its information extraction requirements, which vary from verbal relations between entities (Banko et al., 2007; Etzioni et al., 2004), nominal attributes (Yahya et al., 2014; Pal and Mausam, 2016; Saha et al., 2017), and adverbial components (e.g., time) (Stanovsky et al., 2018). Even for the same type of information, the required facts may still differ. Table 1 shows the required form of facts of three popular OIE tasks: OIE2016 (Stanovsky and Dagan, 2016), Re-OIE2016 (Zhan and Zhao, 2020) and CaRB (Bhardwaj et al., 2019). The diversity of requirements is an essential feature in the field of OIE, which leads to the urgent need for OIE algorithms with strong adaptability.

The adaptability problem in the OIE field has not been well addressed. There are two primary methodologies to obtain an OIE system: the rule-based approach and the end-to-end learning-based approach. The rule-based approaches (Christensen et al., 2011; Corro and Gemulla, 2013; Angeli et al., 2015; Bhutani et al., 2016; Gashteovski et al., 2017) use human-written or bootstrap-learned rules to convert linguistic structures of sentences into target facts. The end-to-end learning approaches (Stanovsky et al., 2018; Sun et al., 2018b,a; Roy et al., 2019; Ro et al., 2020; Kolluru et al., 2020; Liu et al., 2020) first build a dataset containing <sentence, facts> pairs and then use end-to-end learning to train a neural network as the OIE system. However, these methodologies develop a specific machine for every single task. When the requirements change, one must rewrite the complex rule system or re-annotate the data and then retrain the model. These methodologies fail to meet the need for strong adaptability in the OIE field.

Recently, a concept called Open Information eXpression (OIX) was proposed by Sun et al. (2020) to address the adaptability issue of OIE algorithms. The idea of OIX is to introduce an intermediate layer between the language and OIE, which can express the sentence without information loss and be easily adapted to various OIE tasks. Sun et al. (2020) proposed a standard, called Open Information Annotation (OIA), to implement OIX. The OIA standard defines an annotation criterion of

Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics
Volume 1: Long Papers, pages 6213 - 6226

Fact	OIE2016	Re-OIE2016	CaRB
<[is], Ms. Lee, headmaster>	✗	✓	✓
<is responsible, Ms. Lee, for this>	✓	✓	✓
<told, Ms. Lee, Lily, she is responsible for this>	✓	✗	✓
<told, Ms. Lee, Jimmy, she is responsible for this>	✓	✗	✓
<told, Ms. Lee, Lily and Jimmy, she is responsible for this>	✗	✓	✗

Table 1: Facts defined in different OIE tasks, for the expression "*Ms. Lee, the headmaster, told Lily and Jimmy she is responsible for this.*". With OIA, we can easily adapt to these various OIE standards using simple rules.

natural language sentences, which aims to express all information of a sentence into a Predicate-Function-Argument structure, represented by a single-rooted DAG graph. In addition, they implemented a rule-based OIA system that generates OIA graphs from Universal Dependency graphs.

Following the methodology of OIX/OIA, we implement an adaptable and efficient OIE system - OIE@OIA. The framework of OIE@OIA shown in Figure 1 has two components. The first one is the OIA generator, which converts a sentence into an OIA graph. We annotate a large OIA dataset (containing 12,543 training samples, 2,002 development samples, and 2,077 testing samples), develop an efficient learning algorithm to learn and inference the OIA graph, and finally build an end-to-end OIA graph learner. The second component is a group of adaptors, one for each OIE task. We show three popular OIE tasks focused on in this paper in the figure. Furthermore, one can write new adaptors for new OIE tasks, which are very simple, as shown in the following sections. The OIE@OIA system achieves the SOTA (or comparable) performance on three OIE tasks: OIE2016, Re-OIE2016, and CaRB, verifying the adaptability of our OIE@OIA system. Furthermore, our OIE@OIA only needs 12K samples to train, whereas existing end-to-end OIE methods typically need millions of training samples. This verifies the efficiency of our OIE@OIA system.

The **contribution** of this work is as follows:

- An adaptable and efficient OIE system – OIE@OIA, achieving the SOTA performance on different OIE tasks.
- The first end-to-end OIA learning pipeline built on a large human-labeled OIA dataset (we make it open available) and an efficient algorithm for the OIA graph;

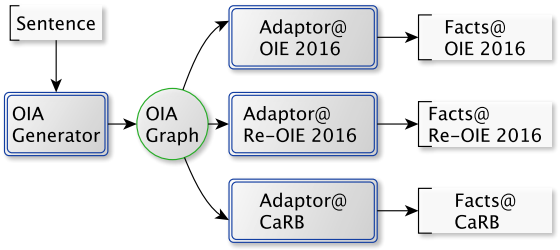


Figure 1: The framework of OIE@OIA.

2 OIE@OIA

In this section, we introduce the OIE@OIA framework, compare it with existing methodologies, and finally discuss its capability and limitation.

2.1 OIA Generator

The core of the OIE@OIA framework is an end-to-end learned OIA generator. To build the generator, we annotate a large dataset using sentences from English-EWT (version 2.4, containing 16K sentences)[1] and design a neural-based algorithm for learning the OIA graph from sentences. The data annotation procedure and the learning procedure are detailed in Section 3.

The standard OIA graph described in Sun et al. (2020) only defined three node types: Constant, Predicate, and Function. However, users may need more fine-grained type information about nodes, especially the type of predicates, to filter wanted facts. For instance, in building OIE systems based on OIA, we need to recognize verbal nodes, which act as the relationship descriptions of the OIE facts. In addition, in event logic graph construction (Ding et al., 2019), logical predicates are essential. With this consideration, we update the type field to a fine-grained version for nodes in the OIA graph according to its semantic function. The fine-grained node types are listed in Table 2.

[1] https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-2988

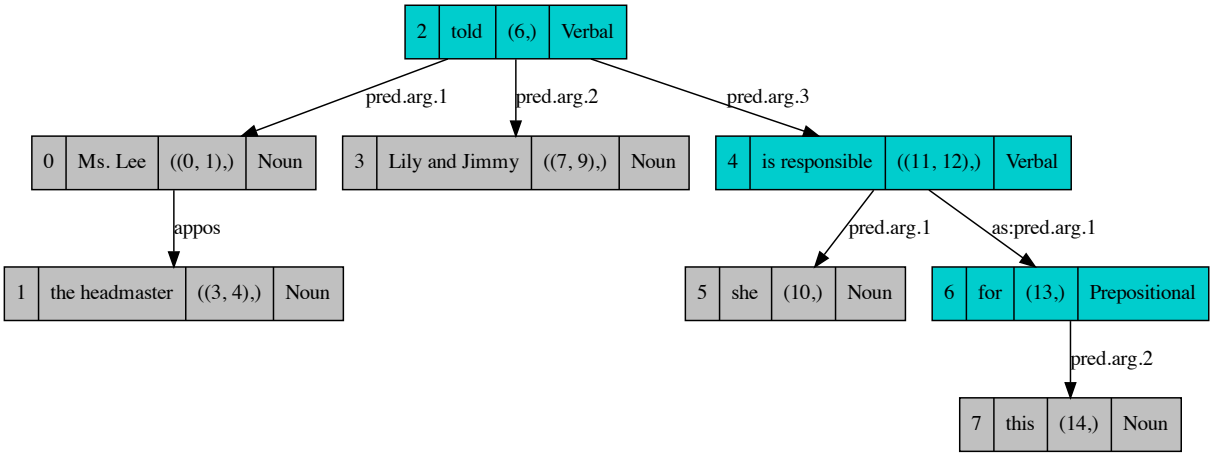


Figure 2: The OIA graph for sentence *"Ms. Lee, the headmaster, told Lily and Jimmy she is responsible for this."*

Fine-grained Type	Original Type	Example
Verbal	Predicate	*know*
Prepositional	Predicate	*to*
Logical	Predicate	*and*
Function	Function	*when*
Noun	Constant	*Chicago*
Modifier	Constant	*well*

Table 2: Node types in OIA graph. Examples are extracted from "*I know him well, and I remember when he went to Chicago*".

Figure 2 shows the OIA graph created by our OIA generator for the example sentence in Table 1, where the predicate nodes are highlighted with color cyan. We can see that all information in the sentence is expressed in the graph, and the *predicate-argument* structures concerned in OIE tasks are accurately captured.

2.2 OIE Adaptors

Next, we illustrate how to design adaptors to accomplish three popular OIE tasks: OIE2016, Re-OIE2016, and CaRB. The OIE2016 and Re-OIE2016 adaptors are built based on the relabeling principle of Zhan and Zhao (2020). The adaptor for CaRB is built based on the labeling instructions in attachment[2] of Bhardwaj et al. (2019). Note that one can always design new adaptors for new OIE tasks.

Figure 2 shows the OIA graph of the example sentence expressing the common units required by these three tasks. First, we design the following simple rules:

Verbal: For each verbal node in the OIA graph, we take the node as the predicate of the OIE fact and take each child sub-tree of the verbal node as an argument of the fact. For instance, given the sample in Figure 2, the extracted facts using the rule are <*"told", "Ms. Lee, the headmaster", "Lily and Jimmy", "she is responsible for this"*> and <*"is responsible", "she", "for this"*>.

VerbalPiP: In the OIA graph, for each verbal node with a prepositional child, we merge the child into the verbal node and apply the *Verbal* rule on the resultant OIA graph. This produces <*"is responsible for", "she", "this"*> for the sample in Figure 2 instead of <*"is responsible", "she", "for this"*>.

Appos(be): All edges like <*A, appos, B*> in OIA graphs are extracted to form the facts <*be, A, B*>.

CoordSep: The fact tuples with coordination arguments are separated into multiple fact tuples, e.g., <*told, ~, Lily and Jimmy, ~*> is separated into <*told, ~, Lily, ~*> and <*told, ~, Jimmy, ~*>.

Then, we implement the adaptors for the three tasks using the combinations of the above rules:

- Adaptor@OIE 2016 = **Verbal** + **CoordSep**;
- Adaptor@Re-OIE 2016 = **Verbal** + **Appos**([is]);
- Adaptor@CaRB = **VerbalPiP** + **Appos**(is) + **CoordSep**.

2.3 Comparisons

In this section, we compare OIE@OIA with existing OIE methodologies and show the difference in Table 3. The traditional rule-based OIE methods are generally based on a sentence annotation structure, such as dependency graphs or constituency graphs, and apply rules to convert the annotation structure into the OIE facts. This

[2]https://aclanthology.org/D19-1651/

Methodology	Rule-Based OIE	End-to-End OIE	OIE@OIA
Sentence Annotation	Dependency / Constituency	-	OIA
OIE Sensitiveness of Annotation	No	-	Yes
Rule Complexity	High	-	Low
Training Data	-	1 Million	12K
Training Efficiency	-	Low	High
Adaptation to New Task	Rewrite Rules	Relabel and Retrain	New Adaptor
Adaptation Cost	May Be High	May Be High	Low

Table 3: Comparisons among Rule-based OIE, End-to-End OIE , and OIE@OIA.

pipeline is similar to our OIE@OIA, where OIE@OIA uses the OIA graph as the sentence annotation structure. However, the differences between the traditional annotation and OIA make the substantial differences between the rule-based OIE and our OIE@OIA. Since the traditional annotation - dependency and constituency - is not designed for the OIE task, one needs to write a complex rule system (or construct by bootstrapping) to convert those annotation structures into the OIE facts. However, for OIE@OIA, since the OIA is designed for OIE tasks, one can accomplish the conversion with straightforward rules, just like those described in the above section. As for the end-to-end OIE algorithms, existing methods are generally built on OpenIE4 dataset (Zhan and Zhao, 2020), which contains about 1 million training samples. However, differences may exist in the forms between the training dataset and the target task, so the performance may drop when adapting to new tasks. To limit the differences, one may need many new training samples and retrain the model. Our OIE@OIA can adapt to an extensive range of new tasks by introducing new adaptors, so it has much better adaptability. In addition, our experimental studies show that OIE@OIA needs only 12K samples for training to achieve new SOTA OIE performance, so it is much more efficient to implement an OIE system.

2.4 Capability and Limitations

Besides the type of facts defined in the three popular tasks, one can extract more types of facts from OIA graphs. For example, since OIA graphs are naturally hierarchical, one can easily extract the nested facts, which can implement the target task of NestIE (Bhutani et al., 2016). One can also extract logical relationships between facts since OIA identifies the logical predicate nodes. We believe OIA can act as a general platform for various OIE tasks and provide better facts for downstream tasks based on OIE (Ding et al., 2019; Zhang et al., 2020).

However, the current version of the OIE@OIA pipeline does not separate the compound noun phrase, making it unable to extract nominal relationships between different nominals within a compound noun phrase (Yahya et al., 2014). This is because current OIA graphs are phrase-level graphs and take noun phrases as single nodes. As an example, "*the president of America*" will form a single node in our OIA graph, and it is not able to identify the relationship between "*the president*" and "*America*" based on the graph. We left this problem as one of our future work.

3 Learning the OIA Graphs

Converting a sentence into the OIA graph is the central operation of the OIE@OIA framework. We build an OIA dataset using active-learning-powered human labeling to implement such an operation. Then, we learn equivalent variants of the OIA graphs – Word-OIA graphs and convert them back to OIA-graphs, which overcomes the difficulty of learning the structures of the OIA graphs.

3.1 Dataset

We annotated sentences of English-EWT (version 2.4) for OIA. The annotation mainly follows the OIA graph definition given by Sun et al. (2020) but with some confusing or special cases being clarified. In addition, we introduce more detailed type information for the node. The obtained dataset contains 12,543 training samples, 2,002 development samples, and 2,077 testing samples. Each sample is a sentence-graph pair. On average, a graph has 7.74 nodes and 6.95 edges, and a node comprises 1.98 words. We make the updated

annotation standard and dataset open available [3].

Auxiliary Annotation System. To improve data annotation efficiency, we generate an initial OIA graph for each input sentence using the existing rule-based OIA system (Sun et al., 2020). For node types initialization, we align the phrases with the POS tags in English-EWT v2.4 and assign heuristic types based on the POS tags of the head words. Then we develop an annotation tool for the annotator to modify the adapted graphs with ease.

Active Learning. The samples in the dev set and test set are all human-labeled. For the training set, we first randomly labeled 2,000 samples, then trained a model using the proposed learning method (described in § 3.4) and started the active-learning procedure. The data labeling order of unlabeled samples was determined by the difference between the rule-generated results and the predicted results. We labeled 200 samples in each active learning iteration and stopped the iteration when the performance of the trained did not improve on the dev set. As a result, we manually annotated about 74% of the training data and treated the rule-generated results as the true labels of the rest 26% training data.

Quality Control. The data annotation was done by three postgraduate/doctoral students of linguistics. Two annotators first label each sample. If there is a disagreement, the third annotator will be involved for discussion and voting. The initial agreement ratio between the two annotators is about 80%, and the final agreement ratio after the discussion (no vote needed) is higher than 93%. The annotation of the rest 7% data is obtained by voting.

3.2 Complexity in Learning the OIA Graphs

The node of OIA graphs consists of a sequence of symbols, placeholders, and words. We call such a graph *Generalized Phrase Graph* ($G_{GPG} = (\mathbb{P}, \mathbb{S})$, where $\mathbb{P}$ is the set of generalized nodes and $\mathbb{S}$ is the edge set). Directly learning the graph is difficult due to the very large decision space caused by the complexity of OIA nodes. The decision space is large even in the simplest situation that each node consists of consecutive words (a span in the sentence without any symbol or placeholder outside the sentence). Since the target number of nodes is unknown, the number of candidate node sets to be considered is exponential to the number of words in the sentence. Due to this large decision space, it is very difficult to learn a good candidate set of nodes for OIA graphs as the first step task in a stage-wise approach. If one prefers the end-to-end approach to reduce the error propagation between tasks in each stage by jointly leaning the nodes and edges, the decision space will be even much larger.

[3] https://github.com/sunbelbd/Open-Information-eXpression

3.3 Equivalence between OIA and Word-OIA

Fortunately, we have the following proposition connecting the *Generalized Phrase Graph* with *Word Graph*, where *Word Graph* is a graph with each graph node corresponding to one and only one word of the sentence:

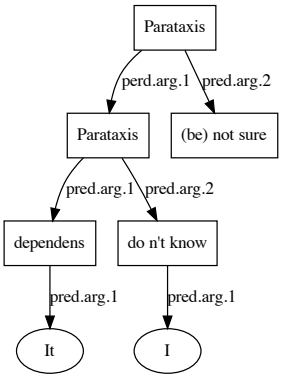
Proposition. *For any Generalized Phrase Graph $G_{GPG} = (\mathbb{P}, \mathbb{S})$, there is a one-to-one corresponding Word Graph $G_W = (\mathbb{W}, \mathbb{S}')$ in the sense that G_{GPG} and G_W can convert to each other without loss of information, where the labels of $\mathbb{S}'$ are independent to word nodes $\mathbb{W}$.*

Proof. We split each generalized phrase node into a chain of nodes of symbols, placeholders, or words, connected in order with the edge *next*, and connect all the edges from parents/children to the first node of the chain. In this new graph, each word is a single node. We replace each type of path between two words (or the virtual root) containing only symbols/placeholders into a single edge with a correspondingly designed edge label and remove the original path. The resulting graph is a valid word graph. □

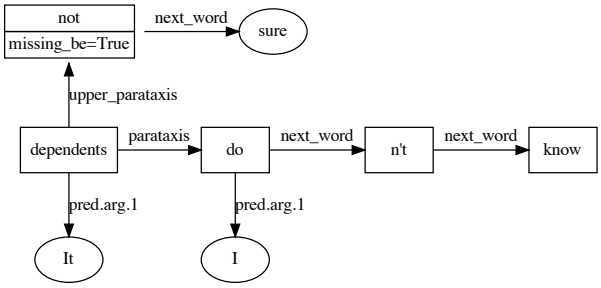
A constructive procedure to convert a *General Phrase Graph* into the *Word Graph* is shown in Appendix A. The OIA graphs are special cases of GPG, and the properties of the OIA graphs can make the procedure much simpler. We discuss these details in Appendix B.

With the above procedure, we can convert a complex OIA graph into an equivalent simple Word-OIA graph (as shown in Figure 3), which is a single-rooted DAG where each node is a word in the original sentence. Each node in the Word-OIA graph has one category attribute *type* (by sharing the type of OIA node it belongs to) and two boolean attributes *arg_whether* and *missing_be* (described in Appendix B).

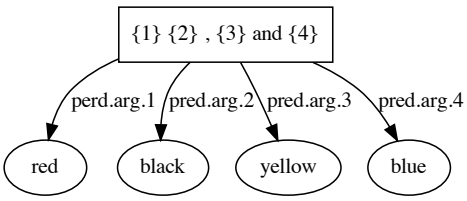
With this conversion, the learning of the OIA graph is equivalently converted into the learning



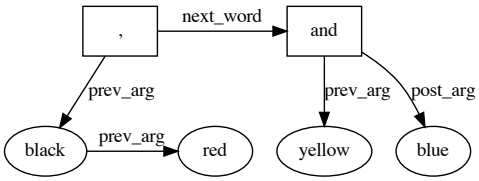
(a) The OIA graph for the sentence A.



(b) The Word-OIA graph for the sentence A.



(c) The OIA graph for the sentence B.



(d) The Word-OIA graph for the sentence B.

Figure 3: The illustration of the equivalence between the OIA graphs and the Word-OIA graphs. The example sentences are: A: "It depends, I don't know. not sure."; B: "red black, yellow and blue".

of the Word OIA graph. The node number of the Word-OIA graph is fixed N, so we only need to learn the possible $N(N-1)$ edges, and thus the learning complexity is significantly reduced.

3.4 Learning the Word-OIA Graphs

The structure of the Word-OIA graph is similar to that of the dependency graph so that the semantic dependency graph learning procedure can be applied to the learning of the Word-OIA graph. We build our learning procedure based on pre-trained BERT models (Devlin et al., 2019). Given a sentence $S = [w_1, \cdots, w_N]$, we generate the representation $\mathbf{r}_i$ of each word by:

$$R = \textbf{BERT}(S),$$

where $R = [\mathbf{r}_1, \cdots, \mathbf{r}_N]$. Then we learn the properties of nodes and the graph structures using these representations.

Node Attribute Learning. For node attribute prediction, we build a one-layer MLP classifier above $\mathbf{r}_i$ to learn each attribute a_k for word w_i:

$$\mathbf{p}_{ki}^{(node)} = p(a_k|w_i) = \text{Softmax}(\mathbf{MLP}_k^{(node)}(\mathbf{r}_i)).$$

The loss of node attribute prediction is defined as:

$$\mathcal{L}_{node} = \frac{1}{N}\sum_k \sum_i \ell_{CE}\left(\mathbf{p}_{ki}^{(node)}, y_{ki}^{(node)}\right),$$

where $y_{ki}^{(node)}$ denotes the target attribute value of the corresponding node of w_i and ℓ_{CE} denotes the cross-entropy loss function.

Edge Learning. Following the protocol of Dozat and Manning (2018), we divide the structure learning into two steps: given two nodes, we firstly determine if there is an edge between them; if so, we then determine the type of the edge. It avoids introducing an empty edge type in the second step, which will overwhelm the other edge types. For the two-step learning, we use the biaffine-based graph learning approach (Dozat and Manning, 2017, 2018). In the first step, for two words w_i and w_j, we learn the representations of each word as the start and end node of an edge:

$$\mathbf{h}_i^{(es)} = \mathbf{MLP}^{(es)}(\mathbf{r}_i), \mathbf{h}_i^{(ee)} = \mathbf{MLP}^{(ee)}(\mathbf{r}_i).$$

where es means "as the start of the edge", ee means "as the end of the edge". The probability of there

being an edge e_{ij} between w_i and w_j is:

$$\mathbf{s}_{ij}^{(edge)} = \mathbf{Biaff}^{(edge)}(\mathbf{h}_i^{(es)}, \mathbf{h}_j^{(ee)}),$$
$$\mathbf{p}_{ij}^{(edge)} = \sigma\left(\mathbf{MLP}^{(edge)}(\mathbf{s}_{ij}^{(edge)})\right).$$

Here, the i-th dimension of $\mathbf{Biaff}(\mathbf{x}_1, \mathbf{x}_2)$ is:

$$\mathbf{x}_1^\top \mathbf{U}_i \mathbf{x}_2 + \mathbf{w}_i^\top (\mathbf{x}_1 \oplus \mathbf{x}_2) + b_i,$$

where $\mathbf{U}_i$, $\mathbf{w}_i$, b_i denotes a trainable matrix, vector, and scalar, respectively. $\oplus$ is the concatenation operator. The corresponding graph topology loss on S is defined as follows:

$$\mathcal{L}_{topo} = \frac{1}{N^2} \sum_{ij} \ell_{CE}\left(\mathbf{p}_{ij}^{(edge)}, y_{ij}^{(edge)}\right).$$

Then we learn the label of each edge. We learn the representations of start-node and end-node of an edge by:

$$\mathbf{h}_i^{(ls)} = \mathbf{MLP}^{(ls)}(\mathbf{r}_i), \quad \mathbf{h}_j^{(le)} = \mathbf{MLP}^{(le)}(\mathbf{r}_j).$$

The probability of the label l_{ij} is:

$$\mathbf{s}_{ij}^{(label)} = \mathbf{Biaff}^{(label)}(\mathbf{h}_i^{(ls)}, \mathbf{h}_j^{(le)}),$$
$$\mathbf{p}_{ij}^{(label)} = \text{Softmax}\left(\mathbf{MLP}^{(label)}(\mathbf{s}_{ij}^{(label)})\right),$$

and the edge prediction loss on S is defined as:

$$\mathcal{L}_{label} = \frac{\sum_{ij} \mathbb{I}(y_{ij}^{(edge)} = 1) \ell_{CE}\left(\mathbf{p}_{ij}^{(label)}, y_{ij}^{(label)}\right)}{\sum_{ij} \mathbb{I}(y_{ij}^{(edge)} = 1)},$$

where $\mathbb{I}(\cdot)$ is the indicator function.

Multi-Task Learning. We learn the Word-OIA graph in a multi-task style, that is, optimize a linear combination of the losses:

$$\mathcal{L} = \alpha \mathcal{L}_{topo} + \beta \mathcal{L}_{label} + (1 - \alpha - \beta)\mathcal{L}_{node}.$$

In the inference phase, we accomplish the following steps to generate the Word-OIA graph:

Node Attribute Prediction. For each node w_i, its type t_i is predicted by:

$$\hat{y}_{ki} = \arg\max \mathbf{p}_{ki}^{(node)}.$$

Edge Prediction. Edge and label between w_i and w_j are predicted by:

$$\hat{e}_{ij} = \mathbf{p}_{ij}^{(edge)} > 0.5,$$
$$\hat{l}_{ij} = \hat{e}_{ij} \cdot \arg\max \mathbf{p}_{ij}^{(label)}.$$

However, because the label prediction may be incorrect, constructing the graph using the above predictions may result in an invalid graph (disconnected graph, edge conflicted, etc.). So in practice, we develop a greedy search strategy to construct the graph step-by-step while maintaining the validness of the graph all the time. First, we select the edge with the highest value of $\mathbf{p}_{ij}^{(label)}$ for all edges with $\mathbf{p}_{ij}^{(edge)} > 0.5$. Then, we identify conflicted edges with the selected edges and set their corresponding values in $\mathbf{p}_{ij}^{(label)}$ to zero. The above process iterates several times until all edge types are set. In addition, the resulting graph may consist of several disconnected sub-graphs. In this case, we iteratively select the edge with the highest $\mathbf{p}_{ij}^{(label)}$ to connect it to the sub-graph to which the predicted root belongs. It guarantees that the generated Word-OIA graph is valid.

3.5 Recover OIA from Word-OIA

For a predicted Word-OIA graph, we reverse the OIA to the Word-OIA procedure to obtain the OIA graph. Specifically, we first collect nodes in Word-OIA graph chained by *next_word* and related arcs (*prev_arg*, *pos_arg*) to form the nodes in OIA graph. Then we identify the special structure such as edge *upper_parataxis* and add special node like *Parataxis* and *Missing* to OIA graph. We add *(be)* to the node span if *missing_be* is true. The *Whether* node is added as the parent of current node if *arg_whether* is true. Last, we connect the nodes using the learned arc labels in Word-OIA. The type of the phrase node in the OIA graph is set as the majority type of its constituted words in the corresponding Word-OIA graph.

4 Experiment I: OIA Learning

The experiment is conducted on the PaddlePaddle deep learning platform[4], and the pre-trained BERT model is provided by the PaddleNLP project[5]. Following Che et al. (2020), the hidden size of $\mathbf{MLP}^{(edge)}$ and $\mathbf{MLP}^{(label)}$ is set to 500 and 100, respectively. The hidden sizes of MLPs used in node attribute predictions are set to 500. The model is trained with the classifier's dropout rate being set to 0.1 and Adam optimizer with a learning rate of 10^{-5}. α and β in loss function are searched using

[4] https://www.paddlepaddle.org.cn

[5] https://github.com/PaddlePaddle/PaddleNLP

Level		Metric	Performance
Node	*type*	Acc	0.951
	arg_whether	Acc	0.998
	missing_be	Acc	0.998
Edge		P/R	0.847 / 0.851
Graph		Acc	0.528

Table 4: Performance of Word-OIA prediction, where P/R means Precision/Recall, Acc means Accuracy.

Generator	Level	Metric	Performance
Rule	Node	P/R	0.796 / 0.855
Rule	Edge	P/R	0.530 / 0.585
Rule	Graph	Acc	0.373
Neural	Node	P/R	0.893 / 0.877
Neural	Edge	P/R	0.709 / 0.688
Neural	Graph	Acc	0.525

Table 5: Performance of OIA graph prediction.

grid search on dev set and set to 0.2 and 0.4, respectively.

Evaluation Metric. In this experiment, the evaluation metrics for all measurements are based on exact match, that is, score 1 if exactly the same, otherwise 0. For nodes, we compare their node expressions. For a Word-OIA node, the node expression is the word index in the sentence; for an OIA node, the node expression is the phrase based on the word indexes it contains. For edges, we compare the triplets of <*start_node_expression, edge_label, end_node_expression*>. For graphs, we test whether the two graphs' node sets and edge sets are exactly the same.

4.1 Performance on Word-OIA

The node precision and recall of Word-OIA are always 1.0 since the nodes correspond to the words in the sentence. The performance of node attribute prediction is illustrated by the upper part of Table 4. The precision/recall of edge prediction is shown in the middle part of Table 4. We also report the accuracy of graph structure in the last part of Table 4.

4.2 Performance on OIA

We report the performances of the nodes, edges, and the whole graph structure of the recovered OIA graphs in the lower half of the Table 5. Note that the graph-structure accuracy of the OIA graph is slightly lower than that of the Word-OIA graph. It is because the graph structure of the OIA graph is related to the node attributes *arg_whether* and *missing_be*. Since there are tiny proportions of bad cases in predicting these two attributes, the graph-structure accuracy of the OIA graph is lower.

As a comparison, we report the performances of the rule baseline (Sun et al., 2020) in upper part of Table 5. We can see that the proposed method achieves significant improvement over the rule baseline, e.g., improving the graph structure accuracy by 15.2%. We believe the learning-based approach solves several problems in the rule-based approach: 1) limitation of expressiveness of Universal Dependency, 2) mistakes in Universal Dependency and Enhanced++ (Schuster and Manning, 2016) parsers, and 3) failure of rules to cover the complex combination of situations.

We also evaluate the accuracy of the node type of the recovered OIA graphs. Among the 89.3% correctly identified nodes, 96.4% of them are labeled with the correct node types by the voting of nodes in the Word-OIA graphs.

4.3 Error Analysis

We reviewed the error cases to find the limitations in the graph generation process. We find several common issues that lead to incorrect graphs.

Long Tail Words and Edges. About 33% errors are caused by the long tail words and edge labels. The out-of-vocabulary words lead to problematical word representations. Rarely used edge labels (e.g., discourse) tend to be predicted as other frequent edge labels.

Granularity Issue. The granularity or boundary of the node may be controversial in prediction results. For example, the phrase 'turn out to be' can be a predicate, but it also makes sense that 'turn out' and 'to be' form a nested relation. Such granularity issues cause about 25% errors in both predicate node and constant node. Mining idioms can further clarify the boundary of expression with refined strategy. This belongs to our future work.

Ambiguous Modification. A prepositional phrase can be used to modify either a noun or a verb in its context. This ambiguity leads to about 17% of graph-level errors. For example, in sentence *I love all the roles in this play*, prepositional phrase *in this play* is the modifier of *all the roles*. Thus, they should be in the same noun node of the ground-truth OIA graph. However, in the predicted graph,

	Systems	OIE2016		Re-OIE2016		CaRB	
		AUC	F1	AUC	F1	AUC	F1
Rule Based	Stanford (Angeli et al., 2015)	7.9	13.6	11.5	16.7	13.4	23.0
	OLLIE (Mausam et al., 2012)	20.2	38.6	31.3	49.5	22.4	41.1
	NestIE (Bhutani et al., 2016)	37.7	43.8	32.1	42.2	19.4	31.1
	PropS (Stanovsky and Dagan, 2016)	32.0	54.4	43.3	64.2	12.6	31.9
	MinIE (Gashteovski et al., 2017)	35.0	41.0	45.5	47.8	28.1	41.3
	ClausIE (Corro and Gemulla, 2013)	36.4	58.0	46.4	64.2	22.4	44.9
	OIE@RuleOIA	37.3	54.6	63.3	75.0	32.4	45.6
Learning Based	OpenIE4 (Christensen et al., 2011)	40.8	58.8	50.9	68.3	27.2	48.8
	BIO (Zhan and Zhao, 2020)	46.2	68.6	71.9	80.3	27.7	46.6
	SpanOIE (Zhan and Zhao, 2020)	48.9	68.7	65.8	77.0	30.0	49.4
	BiLSTM + BERT (Ro et al., 2020)	-	-	72.1	81.3	30.6	50.6
	Multi2OIE (BERT) (Ro et al., 2020)	-	-	74.6	83.9	32.6	**52.3**
	OIE@OIA (BERT)	**54.3**	**71.6**	**76.9**	**85.3**	**33.9**	51.1

Table 6: OIE performance on OIE2016, Re-OIE2016 and CaRB. Note that BiLSTM+BERT and Multi2OIE Ro et al. (2020) use OIE2016 as validation set, so the performances are not listed.

in this play may become the sub-tree of verb *love*. It is a common error in parsing tasks that a model may incorrectly choose the headword for a modifier. We believe better language modeling will ameliorate this problem.

5 Experiment II: OIE@OIA

We further evaluate the applicability of OIA as an intermediate layer between language and OIE, i.e., OIE@OIA, on three tasks: OIE2016, Re-OIE2016, CaRB. We compared OIE@OIA with six rule-based systems and five learning-based systems.

Evaluation Metric. The performances of baseline systems on OIE2016 are from Zhan and Zhao (2020) while that on Re-OIE2016 and CaRB are from Ro et al. (2020), except that NestIE (Bhutani et al., 2016) is implemented using the code from the author. We evaluate the extraction results of the proposed methods, OIE based on rule OIA and NestIE with metric AUC and optimal F1 following the setting of the released codes[6].

The performance of our OIE@OIA system is shown in Table 6. We observe that OIE@OIA achieves better performance than most existing baselines, including learning-based methods trained on millions of samples. This result justifies the effectiveness of OIE@OIA for OIE.

We think the reason for this phenomenon is three-fold: *Firstly*, compared with the annotation of OIA, the annotation of a single OIE task is sparse. Given a sentence, it will only annotate phrases with interesting relationships. For the other phrases and relationships, it will not annotate. In contrast, in OIA, all the phrases and relationships will be annotated. So based on a single sample, the annotation in OIA is much more informative than that in any single OIE task. *Secondly*, if treating the recognition of a type of facts as a task, learning of OIA can be seen as a multi-task learning scenario, and different tasks can augment each other. *Thirdly*, OIA is designed as an intermediate layer between language and OIE. Thus, during the standard design of OIA, it has considered the compatibility between OIA and OIE, which makes it easier to adapt OIA to different OIE tasks.

6 Conclusion

We introduce an adaptable and efficient Open Information Extract system called OIE@OIA. To implement OIE@OIA, we annotate and release an OIA dataset containing about 16K sentences, design an efficient learning algorithm, and build an easy-to-implement rule system to adapt OIA graphs to different OIE tasks. Empirical studies on three popular OIE tasks show that our OIE@OIA system can achieve new SOTA performances on these tasks, using only 12K training sentences. It verifies the great advantage of our system in both effectiveness and efficiency over the previous learning-based baselines, which usually require millions of training samples to achieve comparable performance.

[6] www.github.com/zhanjunlang/Span_OIE, and www.github.com/youngbin-ro/Multi2OIE

References

Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D Manning. 2015. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 344–354, Beijing, China.

Michele Banko, Michael J. Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. 2007. Open information extraction from the web. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2670–2676, Hyderabad, India.

Sangnie Bhardwaj, Samarth Aggarwal, and Mausam. 2019. CaRB: A crowdsourced benchmark for open IE. 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 6261–6266, Hong Kong, China.

Nikita Bhutani, H. V. Jagadish, and Dragomir R. Radev. 2016. Nested propositions in open information extraction. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 55–64, Austin, TX.

Wanxiang Che, Yunlong Feng, Libo Qin, and Ting Liu. 2020. N-LTP: A open-source neural chinese language technology platform with pretrained models. *arXiv preprint arXiv:2009.11616*.

Janara Christensen, Mausam, Stephen Soderland, and Oren Etzioni. 2011. An analysis of open information extraction based on semantic role labeling. In *Proceedings of the 6th International Conference on Knowledge Capture (K-CAP)*, pages 113–120, Banff, Alberta, Canada.

Luciano Del Corro and Rainer Gemulla. 2013. ClausIE: clause-based open information extraction. In *Proceedings of the 22nd International World Wide Web Conference (WWW)*, pages 355–366, Rio de Janeiro, Brazil.

Lei Cui, Furu Wei, and Ming Zhou. 2018. Neural open information extraction. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 407–413.

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 (NAACL-HLT)*, pages 4171–4186, Minneapolis, MN.

Xiao Ding, Zhongyang Li, Ting Liu, and Kuo Liao. 2019. ELG: an event logic graph. *arXiv preprint arXiv:1907.08015*.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, Toulon, France.

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 (ACL)*, pages 484–490, Melbourne, Australia.

Oren Etzioni, Michael J. Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. 2004. Web-scale information extraction in knowitall: (preliminary results). In *Proceedings of the 13th international conference on World Wide Web (WWW)*, pages 100–110, New York, NY.

Kiril Gashteovski, Rainer Gemulla, and Luciano Del Corro. 2017. MinIE: Minimizing facts in open information extraction. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2630–2640, Copenhagen, Denmark.

Keshav Kolluru, Samarth Aggarwal, Vipul Rathore, Mausam, and Soumen Chakrabarti. 2020. IMoJIE: Iterative memory-based joint open information extraction. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 5871–5886, Online.

Guiliang Liu, Xu Li, Jiakang Wang, Mingming Sun, and Ping Li. 2020. Extracting knowledge from web text with monte carlo tree search. In *Proceedings of the Web Conference (WWW)*, pages 2585–2591, Taipei.

Mausam, Michael Schmitz, Stephen Soderland, Robert Bart, and Oren Etzioni. 2012. Open language learning for information extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 523–534, Jeju Island, Korea. ACL.

Harinder Pal and Mausam. 2016. Demonyms and compound relational nouns in nominal open IE. In *Proceedings of the 5th Workshop on Automated Knowledge Base Construction (AKBC@NAACL-HLT)*, pages 35–39, San Diego, CA.

Youngbin Ro, Yukyung Lee, and Pilsung Kang. 2020. Multi^{2}OIE: Multilingual open information extraction based on multi-head attention with BERT. In *Findings of the Association for Computational Linguistics (EMNLP Findings)*, pages 1107–1117, Online Event.

Arpita Roy, Youngja Park, Taesung Lee, and Shimei Pan. 2019. Supervising unsupervised open information extraction models. 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 728–737, Hong Kong, China.

Swarnadeep Saha, Harinder Pal, and Mausam. 2017. Bootstrapping for numerical open IE. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 317–323, Vancouver, Canada.

Sebastian Schuster and Christopher D. Manning. 2016. Enhanced english universal dependencies: An improved representation for natural language understanding tasks. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC)*, Portorož, Slovenia.

Gabriel Stanovsky and Ido Dagan. 2016. Creating a large benchmark for open information extraction. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2300–2305, Austin, TX.

Gabriel Stanovsky, Julian Michael, Luke Zettlemoyer, and Ido Dagan. 2018. Supervised open information extraction. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 885–895, New Orleans, LA.

Mingming Sun, Wenyue Hua, Zoey Liu, Xin Wang, Kangjie Zheng, and Ping Li. 2020. A Predicate-Function-Argument Annotation of Natural Language for Open-Domain Information eXpression. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2140–2150, Online.

Mingming Sun, Xu Li, and Ping Li. 2018a. Logician and orator: Learning from the duality between language and knowledge in open domain. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2119–2130, Brussels, Belgium.

Mingming Sun, Xu Li, Xin Wang, Miao Fan, Yue Feng, and Ping Li. 2018b. Logician: A unified end-to-end neural approach for open-domain information extraction. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining (WSDM)*, pages 556–564, Marina Del Rey, CA.

Mohamed Yahya, Steven Whang, Rahul Gupta, and Alon Y. Halevy. 2014. Renoun: Fact extraction for nominal attributes. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 325–335, Doha, Qatar.

Junlang Zhan and Hai Zhao. 2020. Span model for open information extraction on accurate corpus. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*, pages 9523–9530, New York, NY.

Jingyuan Zhang, Mingming Sun, Yue Feng, and Ping Li. 2020. Learning interpretable relationships between entities, relations and concepts via bayesian structure learning on open domain facts. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 8045–8056, Online.

A Convert Generalized Phrase Graph to Word Graph

We will limit our discussion to a well-formed GPG. First, we assume the relational symbols are not neighbors to each other. That is, they do not form any not continuous sequence. Second, we assume that the relational symbols either appear lonely as a node or connecting/separating two words or placeholders. That is, relational symbols do not appear as prefixes or suffixes in a node. Last, we assume that placeholders must appear together with elements or relational symbols in a node. Since the relational symbols are designed to express relations between nodes, elements, and placeholders, we believe the above constraints are reasonable. OIA graphs naturally satisfy these constraints.

If a graph does not satisfy these constraints, preprocessing can be applied to adapt the graph to meet the constraints. If a continuous sequence of relation symbols exists in the graph, one can merge them into one new relational symbol. If some nodes have prefix/suffix symbols, one can add a boolean attribute to the node indicating that this node has a prefix/suffix symbol. If a node is a sequence of placeholders, one can design a new relational symbol to replace the sequence of placeholders.

Given a well-formed GPG, we can apply Algorithm 1 to convert it to a Word Graph. The sub-procedures used in the algorithm are shown in Algorithm 2. In these procedures, $[x]$ means the label of x, where x can be a node or an edge.

B Specialization for the OIA Graph

OIA graphs have the following properties/internal constraints. We use them to simplify the label system for Word-OIA.

- **Lonely Relational Symbols** In OIA, there are only two Relational Symbols: *Parataxis* and *Missing*. They can only form a node lonely with themselves, which reduce some possible combined edge labels;
- **Ordered Predicate-Argument Labels** In OIA, the predicates connect their arguments with a unified form of edge label *pred.arg.n*, where *n* denote the order of the argument. When processing the *Parataxis*, these arguments are chained by label "*parataxis*", and the *n* is omitted since the order is naturally embedded in the chain;
- **Relational Symbols Nested Up to Two Layers** Nested Relational symbol is the most complex situation for GPG. In OIA, only one symbol *Parataxis* can be nested. From the annotated dataset, we observe that the length of the nested path of *Parataxis* is 2. For this simple situation, after processing the child *Parataxis* nodes, we link the children with edges *upper_parataxis*, *peer_parataxis*, or *lower_parataxis*, according to their depth compared to the depth of the first child. Actually, only *upper_parataxis* is needed in our annotated dataset.

Beyond the above processing, we analyze the label system, remove unnecessary prefixes/suffixes, merge labels, and rename labels for better readability. These steps build a simple edge label system of Word-OIA graphs by introducing few new edge labels.

Besides the label system, when converting the OIA graphs into Word-OIA graphs, we introduce the following node attributes for each node/word to preserve the information of the original OIA graphs:

- *type*: Share the type of the origin OIA node which contains this word;
- *arg_whether*: Boolean attribute that indicate whether the original OIA node is an argument of the *Whether* function;
- *missing_be*: Boolean attribute that indicate whether the original OIA node is a predicate that misses the *be* word.

Algorithm 1: Converting General Phrase Graph to Word-Graph

Data: An input $\mathcal{G}$
Result: $y = x^n$
Add a virtual root node to $\mathcal{G}$ and connect it to the original root with edge $root$;;
while *visiting node* n *over a depth first back-track traversal* $\mathcal{G}$ **do**
  switch n **do**
    case *a single element or is virtual root* **do**
      continue
    end
    case *a sequence of elements (no symbol)* **do**
      Split the sequence into a sequence of a new node with each contains one element, and then connect them in order with edge $next\_elem$;
      Replace n with the first node in the sequence;
    end
    case n *is a mixed sequence of placeholders, relational symbols, and elements* **do**
1   Make a new node for each element;
2   Connect nodes of continuous or placeholder-separated elements with edge $next\_elem$;
3   **foreach** *continuous sequences of placeholders and corresponding edges and children* $[(p_i, e_i, c_i)]_{i=1}^{k}$ **do** `ProcessPlaceholderSequence(`$[(p_i, e_i, c_i)]_{i=1}^{k}$`);`
4   **foreach** *relational symbol* s **do** `BuildRelationalSymbolBridge(`s`);`
5   **if** *element exists in* n **then**
6   **foreach** *placeholder* p **do** `BridgePlaceholderAndElement(`p`);`
      else
        `/* Currently,` n `only has one child corresponding to first placeholder */`
7   Get the only child c with edge e;
8   Find all parents of n as $\{m_i\}$ with edges $\{e_i\}$;
9   **foreach** m_i, e_i **do** Connect m_i to c with edge $[e_i]\_sub\_arg\_[e]$;
      end
    end
    case n *is a relational symbol node* **do**
      `ProcessSymbolNode(`n`)`
    end
  end
end

Algorithm 2: Sub-Procedures

Procedure `ProcessPlaceholderSequence(`$[(p_i, e_i, c_i)]_{i=1}^{k}$`)`

Merge $[p_i]_{i=1}^{k}$ into one placeholder p, set e_1, c_1 be the corresponding edge and child of p;

foreach $i > 1$ **do** Remove e_i, connect $c_i - 1$ to c_i with edge $next\_[p_i]\_[e_i]$;

Procedure `BuildRelationalSymbolBridge(`s`)`

Identify the corresponding edge and node of the previous item as e_p, n_p and that of the next item as e_n, n_n;

if n_p *is a placeholder and* n_n *is an element* **then**

  Connect n_n to n_p with edge label $prev\_arg\_[s]\_[e_p]$, remove e_p;

else if n_p *is an element and* n_n *is a placeholder* **then**

  Connect n_p to n_n with edge label $next\_arg\_[s]\_[e_n]$, remove e_n;

else

  Connect n_p to n_n with edge label $[s]\_[e_n]$, remove e_n;

end

Procedure `BridgePlaceholderAndElement(`p`)`

Find the corresponding edge and node of p as e_p and n_p;

if *no correspondings found* **then return**;

if *The nearest previous element* n_e *exist* **then**

  Connect n_p to n_e with edge $next\_arg\_[e_p]$

else

  Find the nearest next element n_e, connect n_p to n_e with edge $prev\_arg\_[e_p]$

end

Procedure `ProcessSymbolNode(`n`)`

Get the children set $C = \{c_j\}_{j=1}^{k}$ of n and the corresponding edge set $\{e_j\}_{j=1}^{k}$;

if $|C|$ = *1* **then**

  Connect parents of n with the only child c_1 with label $[n]$; Remove n;

else

  Compute $l = \max(L[c] \text{ for } c \text{ in } C) + 1$;

  Connect c_{j-1} to $c_j, j > 1$ with edge $[n]\_l\_[e_j]$; Remove $e_j, j > 1$;

  Add suffix $\_[n]\_l\_[e_1]$ to all parent edge labels of n;

  Replace n with c_1, preserving all edge connection;

end