Parsing into Variable-in-situ Logico-Semantic Graphs

Yufei Chen[1] and **Weiwei Sun**[12]
[1]Wangxuan Institute of Computer Technology
[1]The MOE Key Laboratory of Computational Linguistics
[2]Center for Chinese Linguistics
Peking University
{yufei.chen,ws}@pku.edu.cn

Abstract

We propose variable-in-situ logico-semantic graphs to bridge the gap between semantic graph and logical form parsing. The new type of graph-based meaning representation allows us to include analysis for scope-related phenomena, such as quantification, negation and modality, in a way that is consistent with the state-of-the-art underspecification approach. Moreover, the well-formedness of such a graph is clear, since model-theoretic interpretation is available. We demonstrate the effectiveness of this new perspective by developing a new state-of-the-art semantic parser for Minimal Recursion Semantics. At the core of this parser is a novel neural graph rewriting system which combines the strengths of Hyperedge Replacement Grammar, a knowledge-intensive model, and Graph Neural Networks, a data-intensive model. Our parser achieves an accuracy of 92.39% in terms of ELEMENTARY DEPENDENCY MATCH, which is a 2.88 point improvement over the best data-driven model in the literature. The output of our parser is highly coherent: at least 91% graphs are valid, in that they allow at least one sound scope-resolved logical form.

1 Introduction

Graphs have recently become popular as a strategy for encoding sentence-level semantics, and related data-driven parsing techniques have been making rapid progress. The primary component of popular semantic graphs, e.g. Elementary Dependency Structure (EDS; Oepen and Lønning, 2006) and Abstract Meaning Representation (AMR; Banarescu et al., 2013), is the predicate–argument structure, with the predicate being a concept that takes a number of arguments. Though expressive for many applications, this predicative core does not fully match the need for logical forms that used to stand in the central area of semantic parsing. Partly due to the lack of model-theoretic semantics, it is rather difficult to add scope information related to quantification, negation and modality to a graph. Partly due to the lack of logical deduction engines, it is rather difficult to directly perform automated reasoning over graphs.

This paper proposes to express logical forms with variable-in-situ graphs for the ongoing advances in graph-centric formalisms, algorithms and neural architectures. This leads us to a novel neural graph rewriting system that combines the strengths of Hyperedge Replacement Grammar (HRG; Drewes et al., 1997) and Graph Neural Networks (Song et al., 2018a). On the one hand, it can be viewed as an improved graph embedding model that explicitly explores recursive structures that are defined by an HRG. On the other hand, it can be viewed as an enhanced graph grammar with which all nodes involved in derivations of graphs are assigned vector-based distributed encodings.

Based on our neural graph rewriting system, we develop a new parser for Minimal Recursion Semantics (MRS; Copestake et al., 2005). By means of the DeepBank (Flickinger et al., 2012) data, our parser achieves an accuracy of 92.39% in terms of ELEMENTARY DEPENDENCY MATCH, which is a 2.88 point improvement over the best data-driven model in the literature. We also consider the structural validity of logico-semantic graphs following the original design of MRS.

The output of our parser is highly coherent: at least 91% graphs are coherent, in that they allow at least one sound scope-resolved logical form.

2 Logico-Semantic Graphs

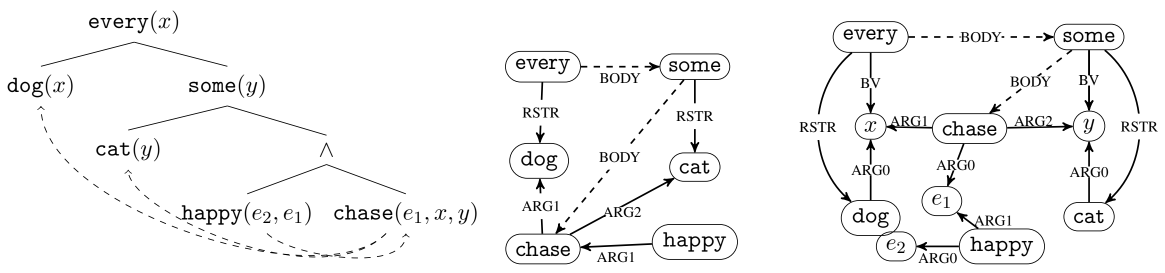
2.1 Logic-Based Meaning Representations

Classic theories of natural language semantics are based on the assumption that the core meaning of a

Source code: https://github.com/draplater/var-parser/

$\mathtt{every(x, dog(x), some(y, cat(y), chase(e_1, x, y)} \land \mathtt{happy(e_2, e_1)))}$[1]

(a) String-based representation



(b) Tree-based representation (c) Variable-reduced graph (d) Variable-in-situ graph

Figure 1: Different representations of a logical form.

sentence is captured as its truth conditions. Under this assumption, using expressions of some logical languages to encode truth conditions is the *de facto* approach in formal semantics. Classic logic, e.g. first-order predicate logic, supports precise, consistent and controlled meaning representation via truth-conditional interpretation.

A logical form can be visualized as a *pseudo* tree, as suggested by Copestake et al. (2005). For example, the formula in Fig. 1a can be encoded as the tree in Fig. 1b. However, the leaves of such a tree are not independent of each other. For instance, $\mathtt{dog(x)}$ and $\mathtt{chase(e_1, x, y)}$ share the same variable x. Transforming logical forms into trees may enlarge the distance between closely-related nodes and make it difficult for a statistical or neural model to explicitly capture such dependencies. In addition, considering syntactico-semantic similarity, this tree-structured logical form is essentially different from the corresponding syntactic tree, as shown in Fig. 2. Such a tree representation brings difficulties to develop a systematic syntax-semantics interface.

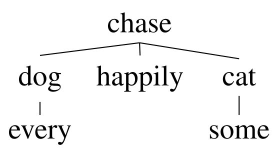


Figure 2: Dependency-based syntactic analysis.

Previous study (Oepen and Lønning, 2006; Copestake, 2009) shows that there are some good engineering reasons for producing a dependency style representation (see Fig. 1c) with links between predicates: It improves readability for consumers of the representation and eases integration with distributional semantics. Exploiting this direction further, we augment such a semantic dependency graph with variables (see Fig. 1d). In fact, it is a more straightforward way to encode logical forms using graphs. Comparing the two types of graphs, we can see that the variable-in-situ representation fully specifies what there is in a logical form, while a variable-free graph may lose some information. Take Fig. 1c for example. The following logical form is also compatible with the graph, which is unfortunately a bad reading, since happy, according to its conceptual meaning, is not a scopal predicate.

(1) $\mathtt{every(x, dog(x), some(y, cat(y), happy(e_2, chase(e_1, x, y))))}$

2.2 Representing Underspecification

Natural language utterances are often ambiguous, i.e., they have more than one reading. Take scope ambiguity, an important type of ambiguity that has been receiving heightened attention by semanticists, for example. Considering the following sentence:

(2) a. Every dog happily chases some cat.

b. $\mathtt{some(y, cat(y), every(x, dog(x), happy(e_2, e_1)} \land \mathtt{chase(e_1, x, y)))}$

c. $\mathtt{every(x, dog(x), some(y, cat(y), happy(e_2, e_1)} \land \mathtt{chase(e_1, x, y)))}$

The sentence is ambiguous: it can either mean that for every dog it is the case that it chases some— potentially different— cats; or else it can mean that there is a particular group of cats which are chased by every dog. The two readings are all made up of the same set of predicates and operators, but differ in the relative scopes of certain

[1]This formula is comparable to the following first-order formula: $\forall x(\mathtt{dog(x)} \rightarrow \exists \mathtt{y}(\mathtt{cat(y)} \land (\mathtt{chase(e_1, x, y)} \land \mathtt{happy(e_2, e_1)})))$

scope bearing elements. There are some other natural language constructions that also involve scope ambiguity, e.g. negation and modality.

Underspecification is by now the standard technique to deal with semantic ambiguities in many modern semantic theories, e.g. Underspecified Discourse Representation Theory (Kamp et al., 2011) and Hole Semantics (Bos, 1996). The basic idea behind it is to derive a single compact representation that describes the set of readings for a sentence that exhibits a scope ambiguity. The individual readings can be enumerated from such an underspecified description if it is required (Koller and Thater, 2005), but it is also possible to process underspecified representations directly without enumerating the readings (Koller and Thater, 2010).

In this paper, we make our logico-semantic graph representations expressive to exhibit the complexities of human language semantics to some extent, by adopting a specific formalism for underspecification, i.e. Minimal Recursion Semantics (MRS; Copestake et al., 2005), a widely-used computational semantic framework in NLP. In addition to variables to represent individuals or events, an MRS structure use another kind of element, called *handle*, to represent out-of-scope relationships between predicates. Each node is assigned with a *label handle*, and some arguments of a concept are specified as *hole handles*. Note that a hole argument is different from an event-variable argument, as illustrated by Ex. (1). Handles can be added to current variable-in-situ graph as a new type of node. See Fig. 3 for an example. h_1, h_2, h_3, h_4 and h_5 are labels, h_2, h_5, h_7, h_9 are hole handles. The out-of-scope relationships in logical forms are converted into a set of constraints between *holes* and *labels*. For example, if we let $h_7 = h_4$ and $h_9 = h_3$, the MRS will be resolved into reading Ex. (2c); similarly, $h_4 = h_1$ and $h_7 = h_3$ for reading Ex. (2b).

To be more precise, a variable-in-situ logico-semantic graph is a graph such that,

- every node must be a predicate, handle or variable;
- every edge must be (1) between a predicate and a variable, encoding predicate–argument relation, (2) between a predicate and a handle, encoding scopal argument or (3) between a predicate and a label, encoding naming convention and tagged by "L."

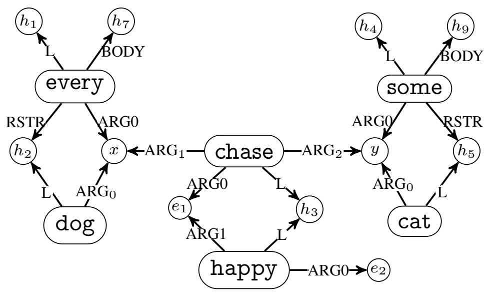


Figure 3: Underspecified logico-semantic graph. Handles associated to predicates are labeled with "L," while handles play as arguments are labeled with semantic roles, like "RSTR."

2.3 Structural Validity

Considering any type of logical form-equivalent representation, we need to be careful that our structures are well-formed. MRS provides a principled way to enumerate readings from an underspecified logical form (Niehren and Thater, 2003), showing us a way to validate the output logic structure. We thus define a valid semantic structure as an MRS in which a scope-resolved logical form is allowable. To be more precise, a variable-in-situ logico-semantic graph is valid if and only if there exists at least one fully specified logical form that satisfies all the constraints encoded by the graph.

3 Neural Graph Rewriting

Automatically constructing a semantic representation can be achieved by exploring the compositionality principle: The meaning of a complex expression is a function of the meanings of its parts and of the syntactic rules by which they are combined. In this perspective, both *meanings of its parts* and the function of *syntactic rules* can be precisely defined by graph fragments. In this paper, we investigate how to manipulate semantic graph fragments with HRG, a context-free rewriting system for generating graphs. We give a formal description of HRG, and then show how to model syntactico-semantic composition through graph rewriting. Recursive neural networks are also important for handling linguistic data. In this section, we will further augment an HRG with a hypergraph-state LSTM.

3.1 Gluing Graph Fragments with an HRG

An edge-labeled, node-typed hypergraph is a tuple $H = \langle V, E, l, t, X \rangle$, where V is a finite set

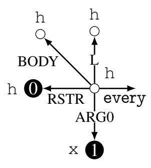
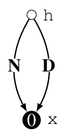
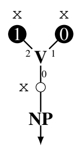
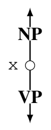
	①	②	③	④
LHS	**N**	**NP**	**VP**	**S**
Type	`hx`	`x`	`xx`	$\emptyset$
RHS (sem.)				
Syntax	every	**D + N**	**V + NP**	**NP + VP**

Table 1: Example HRG rules. Throughout this paper, we use filled black nodes to indicate external nodes, arrows to indicate single-node edges and directed arcs to indicate edges connected to two nodes. The edge labeled as **V** in Rule ③ connects more than two nodes whose orders are indicated by tiny numbers around lines. We use single-node edges with underlined terminal labels to represent predicates, e.g. `every`.

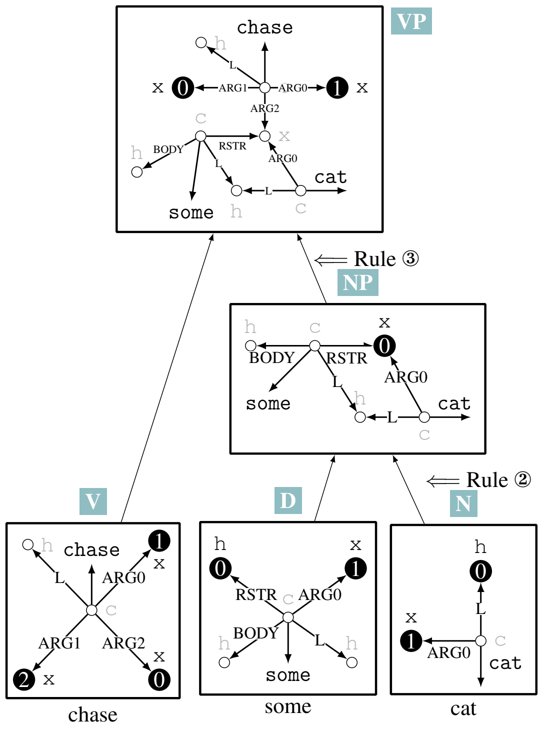


Figure 4: Semantic composition as graph gluing with the rule in Tab. 1. The top composition is according to rule ③, where the left graph that is labeled as **V** is glued with the right graph that is labeled as **NP** by combining their separated external nodes that are labeled as "0."

of nodes, and $E \subseteq V^+$ is a finite set of hyperedges. A hyperedge is an extension of a normal edge which can connect to more than two nodes or only one node. $l : E \to L$ assigns a label from a finite set L to each hyperedge. Since nodes receive no informative labels, we use single-node edges with terminal labels to represent predicates. This strategy is widely used by HRG-based NLP systems, including Chiang et al. (2013), Peng et al. (2015) and Chen et al. (2018). $X \in V^*$ defines an ordered list of nodes called **external nodes**, which specify the docking points during graph rewriting. $t : V \to T$ assigns a type from a finite set T to each node.

Different from the hypergraphs used by Chiang et al. (2013) and Chen et al. (2018), we highlight the usage of node types which has a significant impact on making parsing results logically coherent. Three node types are utilized: `h`, `x` and `c`, which indicate handle, variable and predicate respectively. During node gluing, we must make sure that the types of nodes are identical. If the type of any node is still unspecified, the type of the other node will be selected. For convenience, we define the **type of a non-terminal hyperedge** as the tuple of types of all nodes it connects to; we define the **type of a graph fragment** as the tuple of types of all external nodes in order. For example, the graph fragment of *some* in Fig. 4 is typed as (`h`, `x`), which will be denoted as `hx` for short.

A Typed Hyperedge Replacement Grammar (THRG) $G = \langle N, T, P\rangle$ is a graph rewriting system, where N and T are two disjoint finite sets of non-terminal and terminal symbols respectively. P is a finite set of production rules of the form $A \to R$, where the left hand side (LHS) $A \in N$, and the right hand side (RHS) R is a hypergraph with edge labels over $N \cup T$. The rewriting process replaces a non-terminal hyperedge with the graph fragment specified by a rule's RHS, attaching each external node to the matched node of the corresponding LHS. In the meantime, the co-related nodes in LHS and RHS must be of the same types. Tab. 1 presents four example rules. Rule ③ consists of three nodes and two hyperedges. All three nodes are of type `x`, indicating that they are variables. One hyperedge has a label **NP** and connects to one internal node; the other is labelled as **VP** and connects to one internal node and two external nodes. Fig. 4 presents the composition process for *chase some cat*, in which Rule ② and ③ are recursively called for semantic construction.

The types of a HRG rule put additional constraints to the combination of subgraphs and in this way the output graph is regularized to some extent. A failed combination is illustrated in Fig. 5.

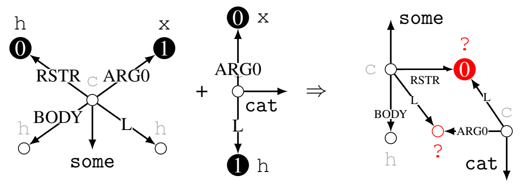


Figure 5: We deliberately swap the two external nodes of graph fragment *cat*. This combination can be blocked by type restrictions.

3.2 Recursive Hypergraph-state LSTM

Since we explicitly describe a recursive process, we are able to define a new graph embedding method—encoding graphs along with such a recursive structure. Our strategy is to assign vectors to nodes involved in the composition process in a bottom-up way. Before the application of an HRG rule $A \to R$ ($R = \langle V, E, l, t, X\rangle, V = \{n_1, n_2, ...\}, X = \{e_1, e_2, ...\}$), the external nodes of all non-terminal edges in R have been assigned vectors based on preceding composition while other newly introduced nodes are zero-initialized. The vectors assigned to all nodes in R will be updated according to a Graph Neural Network (GNN), which works by exploiting locality encoded by R. In this paper, we propose a hypergraph-state LSTM structure to do so. In what follows, we will first introduce our GNN model and then use it to equip an HRG, resulting in a recursive hypergraph-state LSTM model.

Each node $n_j \in V$ has a node property vector $\boldsymbol{x}_{n_j}$ to represent its own information, such as the type and the corresponding label of a concept node, and the index of an external node. And another hidden state vector $\boldsymbol{h}_j$ is employed to hopefully encode the information of its surroundings. The surrounding information of n_j is collected by multi-step information exchange between n_j and its neighbouring nodes, denoted as $\pi(n_j)$. Two nodes n_j and n_k are viewed as neighbours if there is at least one hyperedge that connects them. To keep its own information, we assume that each node has a self loop, i.e. $n_j \in \pi(n_j)$. Thus the neighbouring relation is symmetric. An optional label $l(n_j, n_k)$ can be attached to each neighboring relation.

Each node has an initial state $\boldsymbol{h}_j^0$, representing the state when information has not been updated yet. In each step of information exchange, according to $\boldsymbol{x}_j$ and its previous hidden state $\boldsymbol{h}_j^{t-1}$, the new hidden state $\boldsymbol{h}_j^t$ is calculated from the representation of itself, its neighbours $\pi(n_j)$, and the label of each relation, in a way as generally defined as follows:

$$\boldsymbol{h}_j^t = f(\{\boldsymbol{x}_k | k \in \pi(n_j)\}, \{\boldsymbol{h}_k^{t-1} | k \in \pi(n_j)\}, \{l(n_j, n_k) | n_k \in \pi(n_j)\})$$

Assume that $\boldsymbol{L}$ is a randomly initialized matrix for encoding neighbouring labels. Summation is utilized to collect information from neighbouring nodes:

$$\boldsymbol{\Pi}_{x,j} = \sum_{k \in \pi(n_j)} (\boldsymbol{x}_k \oplus \boldsymbol{L}[l(n_j, n_k)])$$
$$\boldsymbol{\Pi}_{h,j}^{t-1} = \sum_{k \in \pi(n_j)} \boldsymbol{h}_k^{t-1}$$

Introducing the LSTM gate mechanism, the state transition can be written as:

$$\begin{aligned}
\boldsymbol{i}_j^t &= \sigma(\boldsymbol{W_i}\boldsymbol{\Pi}_{x,j} + \boldsymbol{U_i}\boldsymbol{\Pi}_{h,j}^{t-1} + \boldsymbol{b}_i) \\
\boldsymbol{o}_j^t &= \sigma(\boldsymbol{W_o}\boldsymbol{\Pi}_{x,j} + \boldsymbol{U_o}\boldsymbol{\Pi}_{h,j}^{t-1} + \boldsymbol{b}_o) \\
\boldsymbol{f}_j^t &= \sigma(\boldsymbol{W_f}\boldsymbol{\Pi}_{x,j} + \boldsymbol{U_f}\boldsymbol{\Pi}_{h,j}^{t-1} + \boldsymbol{b}_f) \\
\boldsymbol{u}_j^t &= \sigma(\boldsymbol{W_u}\boldsymbol{\Pi}_{x,j} + \boldsymbol{U_u}\boldsymbol{\Pi}_{h,j}^{t-1} + \boldsymbol{b}_u) \\
\boldsymbol{c}_j^t &= \boldsymbol{f}_j^t \otimes \boldsymbol{c}_j^{t-1} + \boldsymbol{i}_j^t \otimes \boldsymbol{u}_j^t \\
\boldsymbol{h}_j^t &= \boldsymbol{o}_j^t \otimes \tanh(\boldsymbol{c}_j^t)
\end{aligned}$$

where $\boldsymbol{i}, \boldsymbol{o}, \boldsymbol{f}$ are the input, output and forget gates of LSTM. $\boldsymbol{W}$ and $\boldsymbol{U}$ are the model parameters.

Similar to the tree LSTM (Tai et al., 2015), our recursive hyperedge-state LSTM model composes the states of a graph fragment from input vectors and the representations of its subgraphs. The model alternates between two kinds of steps: (1) graph fragment encoding and (2) state propagation. The process for encoding a non-leaf graph fragment is visualized in Fig. 6. The most important feature of our graph encoding method is that the process is step-wise, making it possible to perform semantic disambiguation and graph encoding iteratively.

In a graph fragment encoding step for R, we want to get some vectors representing a specific graph fragment for further combination. This can be done by running multilayer hypergraph-state LSTM (denoted as HGS) on R:

$$[\boldsymbol{h}_{n_1}^T; \boldsymbol{h}_{n_2}^T; ...] = \text{HGS}^T([\boldsymbol{h}_{n_1}^0; \boldsymbol{h}_{n_2}^0; ...], [\boldsymbol{x}_{n_1}; \boldsymbol{x}_{n_2}; ...], R)$$

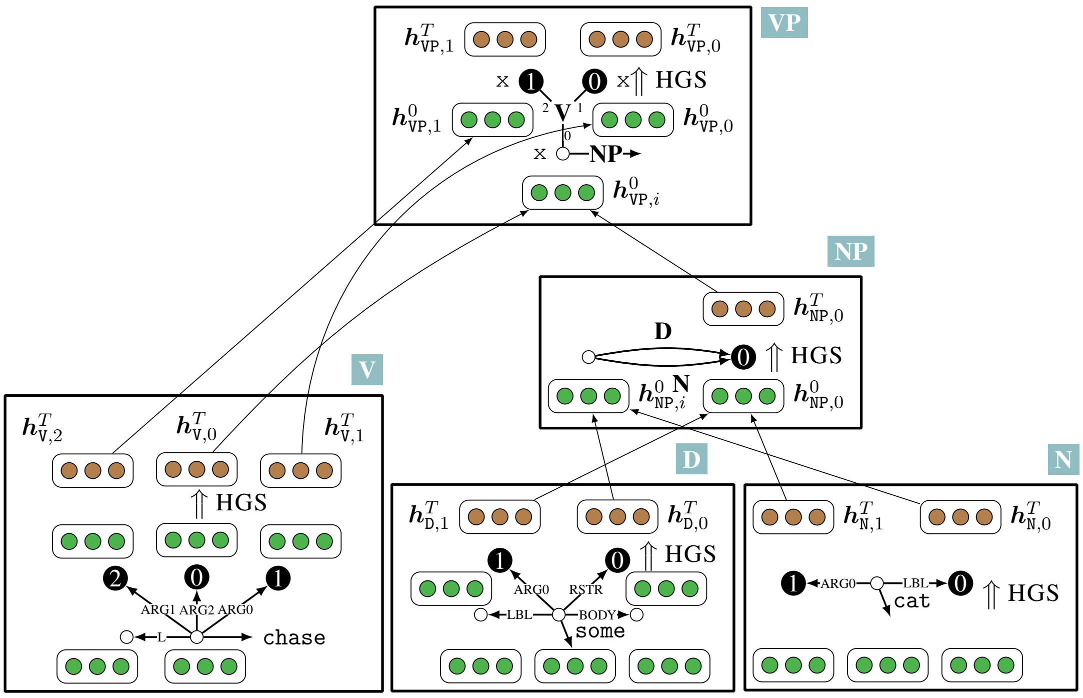


Figure 6: A graphical illustration of our recursive hypergraph-state LSTM model. "⇑ HGS" represents graph encoding with hypergraph-state LSTM. The final hidden states $\boldsymbol{h}^T$ of external nodes are used as interface vectors (brown vectors). Solid lines across the boxes denotes state propagation steps, we initialize hidden states according to the corresponding HRG rule, e.g. $\boldsymbol{h}^0_{\texttt{NP,0}} = \boldsymbol{h}^T_{\texttt{D,1}} + \boldsymbol{h}^T_{\texttt{N,1}}$ and $\boldsymbol{h}^0_{\texttt{NP,i}} = \boldsymbol{h}^T_{\texttt{D,0}} + \boldsymbol{h}^T_{\texttt{N,0}}$.

T represents the number of layers in the hypergraph-state LSTM. For a node n_j in a lexical graph fragment, we use a zero vector as $\boldsymbol{h}^0_{n_j}$. For the non-leaf case, $\boldsymbol{x}$ and $\boldsymbol{h}^0$ is acquired from preceding state propagation. Not all final states $\boldsymbol{h}^T_{n_1}, \boldsymbol{h}^T_{n_2} \dots$ should be kept for further composition. Considering the role played by external nodes in graph gluing, we use the final states of external nodes $\boldsymbol{h}^T_{e_1}, \boldsymbol{h}^T_{e_2} \dots$ to pass information and call them **interface vectors**.

State propagation is the preparatory stage of non-leaf graph fragment encoding, in which the interface vectors of its subgraph fragments are combined to calculate $\boldsymbol{x}$ and $\boldsymbol{h}^0$ for the next step. Without the loss of generality, we only discuss the case for binary rules in which R consists of two non-terminal hyperedges. It is worth noting that in non-leaf graph fragment encoding, the hypergraph-state LSTM is operated on a rule rather than the entire graph fragment. The process of encoding a non-leaf graph fragment can be seen as encoding an RHS R with special initial states originated from interface vectors. The nodes in R are of three types: unified nodes, passover nodes and newly created nodes. Newly created nodes bring new information to the combined graph fragment while the other two kinds of nodes are only used for structural connection. For a newly created node, the node property vector $\boldsymbol{x}$ is calculated from its own information, and the initial state is a zero vector. A unified node is connected by both non-terminal hyperedges, and therefore receive information from both sides. The initial state $\boldsymbol{h}^0$ of a unified node is the sum of the two corresponding interface vectors. The property vector $\boldsymbol{x}$ is redefined as the sum of the two related property vectors. A passover node is a node connected to only one non-terminal hyperedge. And its property vector and initial state are simply copied from the unique corresponding node. For example, the rule **VP** in Tab. 1 contains one unified node and two passover nodes. Denote the set of corresponding nodes of n_j as $\text{cor}(n_j)$. $|\text{cor}(n_j)|$ is 0, 1 or 2 for newly created nodes, passover nodes and unified nodes respectively. $\boldsymbol{x}_{n_j}$ and $\boldsymbol{h}^0_{n_j}$ for non-leaf graph fragment encoding can be calculated as:

$$\boldsymbol{h}^0_{n_j} = \sum_{n_i \in \text{cor}(n_j)} \boldsymbol{h}^T_{n_i}$$

$$\boldsymbol{x}_{n_j} = \sum_{n_i \in \text{cor}(n_j)} \boldsymbol{x}_{n_i} \text{ if } |\text{cor}(n_j)| \neq 0$$

4 Parsing to Variable-in-situ Graphs

Following our previous work (Chen et al., 2018), we continue to employ a synchronous grammar to build a practical parser. We integrate a CFG that expresses syntactic composition with an HRG that expresses semantic composition. Semantic construction is divided into two subtasks: syntactic parsing and semantic interpretation. When a phrase structure tree T is available, a semantic interpreter *translates* T to the derivation of graph construction by assigning corresponding HRG rules to the syntactic counterparts. At a single derivation step, there may be more than one HRG rule applicable. In this case, we need a disambiguation model to select a good one.

The simplest disambiguation model is a **count-based model**: Given a coherent derivation tree, together with corresponding rule types, it simply selects the most frequent rule in the training data. This model provides baseline performance for reference. Chen et al. (2018) showed that disambiguation can be significantly improved when a classifier is introduced. In particular, they proposed a **feature engineering-based classifier**, in which manually defined sparse vectors are utilized. This is not suitable for our purpose because a variable-in-situ graph is much more complex in that much more external nodes are involved. With the neural graph rewriting system introduced in §3.2, we propose a **subgraph-based model** which can handle the above problem by automatically learning vector representations for graphs.

More concretely, assume that we have built the left and right subgraphs, denoted by H_l and H_r, for further composition. Usually, multiple rules, viz. $r_1, r_2, ..., r_M$, are applicable to combine H_l and H_r. Let the possible merged graphs be denoted by $\mathcal{H} = \{H_1, H_2, \ldots, H_M\}$. To build a high-quality graph, we need to rank $H_1, H_2, ...$ according to some score functions that reflect their *goodness*. Formally, we have an optimization problem:

$$\hat{H} = \arg \max_{H_m \in \mathcal{H}} \textsc{score}(H_m)$$

To calculate the score for H_m, we consider both syntactic and semantic contexts. To reflect the syntactic information, we use a vector-based encoding, denoted by $\boldsymbol{s}_{i,j}$, of the corresponding phrase/span (i, j) that can be calculated by a sequence-based model, such as LSTM or Transformer. Graph fragment H_m with n external nodes can be encoded by the neural graph rewriting system: running a recursive hypergraph-state LSTM on the RHS R_m of an HRG rule where the interface vectors of H_l and H_r are consumed as initial states. After that we get n new interface vectors related to H_m (denoted as $\boldsymbol{u}_{m,k}, 0 \le k < n$). Taking advantage of the recursive structure, the common parts H_l and H_r of graph fragments $H_1, H_2, ...$ are encoded only once, avoiding redundant computation. We use an attention mechanism to get a single vector representation $\boldsymbol{t}_m$ for the graph fragment H_m:

$$w_{m,k} = (\boldsymbol{u}_{m,k})^\top \boldsymbol{W} \boldsymbol{s}_{i,j}$$
$$\boldsymbol{t_m} = \sum_{0 \le k < n} (\boldsymbol{u}_{m,k} \cdot w_{m,k})$$

We use the similarity between $\boldsymbol{t}_m$ and $\boldsymbol{s}_{i,j}$ as the score of this graph fragment. For training, we use the cross-entropy function as loss.

$$\textsc{score}(H_m, i, j) = (\boldsymbol{t}_m)^\top \boldsymbol{W}_2 \boldsymbol{s}_{i,j}$$

5 Experiments

5.1 Data Setup

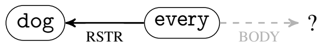
DeepBank (Flickinger et al., 2012) is a deep linguistic resource that covers the Wall Street Journal section of Penn TreeBank (PTB; Marcus et al., 1993). All annotations are governed by English Resource Grammar (ERG; Flickinger, 2000). We use the DeepBank v1.1 data, and split it into training, development and test sets along with previous work (Oepen et al., 2014, 2015; Buys and Blunsom, 2017; Chen et al., 2018) to make sure that the numeric performance can be directly compared to the results in the literature.

5.2 Evaluation Metrics

Token-wise Evaluation for Accuracy The semantic annotations in DeepBank are presented as variable-in-situ MRS style originally. It is a non-trivial problem to measure the similarity between different logical forms accordingly. Copestake (2009) provides a method to reversibly translate them into variable-reduced semantic graphs, namely dubbed Dependency MRS (DMRS), in an information-equivalent fashion, which is widely used by previous studies. We convert our outputs to DMRS, and re-use the evaluation metrics for variable-reduced graph representations, including Elementary Dependency Match (EDM;

Dridan and Oepen, 2011) and SMATCH (Cai and Knight, 2013) to perform evaluation.

Search-Based Evaluation for Coherence Another dimension for parser evaluation—the coherence of the output structures—is as essential as accuracy, since we also emphasize on the logical nature. Under the framework of underspecification, the coherence of a semantic structure entails that there must be at least one fully specified, i.e. scope-resolved logical form, which satisfies all the constraints encoded by that structure. The following shows a by-design incoherent semantic graph:



every has two scopal arguments, corresponding to the restriction and body domains respectively, but there is not enough predicates to fill in them.

Niehren and Thater (2003) proved that figuring out whether an MRS structure is coherent is NP-hard. Accordingly, we use exhaustive search to find the first scope-resolved logical form if there is any. Practically, our implementation is efficient enough to cover all graphs produced by our parser.

5.3 Inducing a Synchronous Grammar

#E	EDS	MRS	#E	EDS	MRS
1	**89.59%**	23.42%	4	0.27%	3.53%
2	8.57%	**54.98%**	5+	0.09%	0.40%
3	1.48%	17.55%			

Table 2: Statistics of HRG rule instances. "#E" indicates the number of external nodes. EDS represents the variable-free framework, while MRS represents the variable-in-situ framework.

We conduct automatic grammar induction following our previous method (Chen et al., 2018). Tab. 1 shows some rule examples, while Tab. 2 presents some statistics of the related grammars. There is a big difference between the rule distributions of the grammars for variable-reduced and variable-in-situ semantic graphs. For comparison, we report results on Elementary Dependency Structure (EDS; Oepen and Lønning, 2006). Rules for the latter one have more external nodes on average.

More external nodes bring in a new problem for grammar induction — determining the order of external nodes. Consider the rule related to chase in Fig. 4. chase has three external nodes: the endpoints of ARG0, ARG1 and ARG2. A grammar induction algorithm needs to decide which one is taken as the first external node and which one the second, etc. We find that a good order is important to the performance of a parser. In our experiments, we use the **syntactic attachment order** to decide the order of an external node. The attachment order reflects when a node is being glued to another graph fragment. For example, the ARG2 of chase connects to the graph fragment of cat firstly, since *cat* is the syntactic object; secondly, the ARG0 connects to the graph fragment of happy, because *happily* as a adjunct stands in between object and subject. As a result, we take the ARG2 and ARG0 endpoints as the first and second external nodes. This method not only makes the grammar more regular, but also endows the order of external nodes with semantic meaning.

TH	AO	Span	EDM$_P$	EDM$_A$	EDM	SMATCH
Count-Based						
N	Y	91.98	94.41	65.68	80.52	80.79
Y	N	91.80	94.41	75.35	84.91	85.42
Y	Y	91.76	94.57	87.28	90.91	91.52
Subgraph-Based						
N	Y	91.98	94.86	83.59	89.22	89.72
Y	N	91.80	94.77	89.50	92.11	92.72
Y	Y	91.76	94.85	90.27	92.54	93.39

Table 3: Accuracies on the development data. "TH" indicates whether to use type restriction; "AO" indicates whether the attachment order strategy is applied. "Y/N" is short for "yes/no." "Span" indicates the performance (evalb F-score) of syntactic parsing.

Model	EDM$_P$	EDM$_A$	EDM	SMATCH
Buys and Blunsom	87.54	80.10	84.16	86.69
ACE	92.08	86.77	89.64	93.50
Chen et al.	93.11	86.01	89.51	89.77
Ours (−ELMo)	93.08	88.10	90.56	91.54
Ours (+ELMo)	94.56	90.27	92.39	93.06

Table 4: Accuracies on the test set.

TH	Dataset	SV (%)
N	Devel.	29.91
Y	Devel.	91.71
Y	Test	92.13

Table 5: Results of structural validation (SV).

5.4 Model Setup

We implement a syntactic parser according to Kitaev and Klein (2018), which contains an 8-layer transformer to extract dense vector representations for candidate phrases. ELMo (Peters et al., 2018) is used as pretrained contextualized word embeddings. In addition to the CFG rules, our syntactic parser also predicts the types of synchronous rules. If a phrase NP has a semantic part of type x, it is labeled as NP#x. A CKY decoder is employed to make sure that the output of the syntactic parser is coherent for semantic interpretation. Tab. 3 presents the accuracy of syntactic parsing.

When syntactic trees are ready, the semantic interpreter selects an HRG rule for each tree node. We apply greedy search to complete this translating process. In subgraph-based model, the span features $\boldsymbol{s}_{i,j}$ obtained by the syntactic parser are also used to perform disambiguation. The word embedding and transformer are fixed in this step.

5.5 Results and Analyses

Tab. 3 summarizes the parsing results with different set-ups. There is a significant gap between the typed and untyped HRG with respect to EDM scores. Note that the performance of syntactic parsing is comparable. This demonstrates the necessity to explicitly control the structural coherence of the semantic outputs.

An interesting observation is that the performance also drops significantly without a proper order of external nodes in the count-based model. But the gap narrows after introducing the neural model. It reveals that using the syntactic attachment order makes the grammar more regular, giving it more ability of semantic disambiguation. The recursive hypergraph-state LSTM model is robust. Its strong disambiguation ability can make up for the weakness of the grammar.

Tab. 4 shows the results on test set. Our parser achieves an accuracy of 92.39% in terms of EDM, which is a 2.88 point improvement over the best data-driven model in the literature. For fair competition, we remove the ELMo to match the experiment set-up of previous models. The result shows that we still outperform the previous best model by 1.05 points. We test the well-formedness of the output MRS and present the result in Tab. 5. With type restrictions, the output of our parser is highly coherent: at least 91% MRS allow at least one sound scope-resolved logic form.

6 Related Work

It has been a long time since researchers manipulated semantic construction following the principle of compositionality. Different formalisms have been developed to express the syntactic-semantic interface in natural language utterances. To manipulate compositional construction, HRG is a popular framework to define a graph-structured syntax-semantics interface (Peng et al., 2015; Chen et al., 2018). AM algebra (Koller, 2015; Groschwitz et al., 2017) is another formalism to handle graph construction which has been successfully explored to build semantic parsers (Groschwitz et al., 2018; Lindemann et al., 2019).

Compositional vector representation is also widely studied in recent years. Kiperwasser and Goldberg (2016) encodes syntactic dependency trees with a recursive recurrent neural network, which acts as the core of a bottom-up dependency parser. Dyer et al. (2016) introduced Recurrent Neural Network Grammar, a probabilistic model of sentences with explicit phrase structure. A recursive syntactic composition function is used to compute an embedding of a completed phrase-structure subtree.

Modeling discrete structures with principled neural networks has received an increasing interest. Kipf and Welling (2017) proposed Graph Convolution Network to classify nodes in graphs. DAG-structured LSTM is a natural extension to tree LSTM which treats nodes as basic states (Zhu et al., 2016). Graph-state LSTM can be used in both generation task (Song et al., 2018a) and relation extraction (Song et al., 2018b).

7 Conclusion

Graph-structured meaning representations provide an effective way to encode rich semantic information of natural language sentences and have been extensively studied recently. We enriched the discussion by studying an alternative graph-based representation for underspecified logical forms. In particular, we introduced a novel neural graph rewriting system and developed a new state-of-the-art semantic parser for variable-in-situ graphs.

Acknowledgments

This work is supported in part by the National Hi-Tech R&D Program of China (No. 2018YFC0831900). Weiwei Sun is the corresponding author.

References

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*. Association for Computational Linguistics, Sofia, Bulgaria, pages 178–186. `http://www.aclweb.org/anthology/W13-2322`.

Johan Bos. 1996. Predicate logic unplugged .

Jan Buys and Phil Blunsom. 2017. Robust incremental neural semantic graph parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Vancouver, Canada, pages 1215–1226. `http://aclweb.org/anthology/P17-1112`.

Shu Cai and Kevin Knight. 2013. Smatch: an evaluation metric for semantic feature structures. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. volume 2, pages 748–752.

Yufei Chen, Weiwei Sun, and Xiaojun Wan. 2018. Accurate shrg-based semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 408–418. `http://aclweb.org/anthology/P18-1038`.

David Chiang, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, Bevan Jones, and Kevin Knight. 2013. Parsing graphs with hyperedge replacement grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Sofia, Bulgaria, pages 924–932. `http://www.aclweb.org/anthology/P13-1091`.

Ann Copestake. 2009. ***Invited Talk:*** slacker semantics: Why superficiality, dependency and avoidance of commitment can be the right way to go. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*. Association for Computational Linguistics, Athens, Greece, pages 1–9. `http://www.aclweb.org/anthology/E09-1001`.

Ann Copestake, Dan Flickinger, Carl Pollard, and Ivan A. Sag. 2005. Minimal Recursion Semantics: An introduction. *Research on Language and Computation* pages 281–332.

F. Drewes, H.-J. Kreowski, and A. Habel. 1997. Handbook of graph grammars and computing by graph transformation. World Scientific Publishing Co., Inc., River Edge, NJ, USA, chapter Hyperedge Replacement Graph Grammars, pages 95–162. `http://dl.acm.org/citation.cfm?id=278918.278927`.

Rebecca Dridan and Stephan Oepen. 2011. Parser evaluation using elementary dependency matching. In *Proceedings of the 12th International Conference on Parsing Technologies*. Association for Computational Linguistics, Dublin, Ireland, pages 225–230. `http://www.aclweb.org/anthology/W11-2927`.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, San Diego, California, pages 199–209. `https://doi.org/10.18653/v1/N16-1024`.

Dan Flickinger. 2000. On building a more efficient grammar by exploiting types. *Nat. Lang. Eng.* 6(1):15–28.

Daniel Flickinger, Yi Zhang, and Valia Kordoni. 2012. Deepbank: A dynamically annotated treebank of the wall street journal. In *Proceedings of the Eleventh International Workshop on Treebanks and Linguistic Theories*. pages 85–96.

Jonas Groschwitz, Meaghan Fowlie, Mark Johnson, and Alexander Koller. 2017. A constrained graph algebra for semantic parsing with AMRs. In *IWCS 2017 - 12th International Conference on Computational Semantics - Long papers*.

Jonas Groschwitz, Matthias Lindemann, Meaghan Fowlie, Mark Johnson, and Alexander Koller. 2018. AMR dependency parsing with a typed semantic algebra. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 1831–1841. `http://aclweb.org/anthology/P18-1170`.

Hans Kamp, Josef Van Genabith, and Uwe Reyle. 2011. Discourse representation theory. In *Handbook of philosophical logic*, Springer, pages 125–394.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Easy-first dependency parsing with hierarchical tree LSTMs. *Transactions of the Association for Computational Linguistics* 4:445–461. `https://doi.org/10.1162/tacl_a_00110`.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*.

Nikita Kitaev and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 2676–2686. `http://aclweb.org/anthology/P18-1249`.

Alexander Koller. 2015. Semantic construction with graph grammars. In *Proceedings of the 11th International Conference on Computational Semantics*. Association for Computational Linguistics, London, UK, pages 228–238. https://www.aclweb.org/anthology/W15-0127.

Alexander Koller and Stefan Thater. 2005. Efficient solving and exploration of scope ambiguities. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*. Association for Computational Linguistics, Ann Arbor, Michigan, pages 9–12. https://doi.org/10.3115/1225753.1225756.

Alexander Koller and Stefan Thater. 2010. Computing weakest readings. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Uppsala, Sweden, pages 30–39. https://www.aclweb.org/anthology/P10-1004.

Matthias Lindemann, Jonas Groschwitz, and Alexander Koller. 2019. Compositional semantic parsing across graphbanks.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: the penn treebank. *Computational Linguistics* 19(2):313–330. http://dl.acm.org/citation.cfm?id=972470.972475.

Joachim Niehren and Stefan Thater. 2003. Bridging the gap between underspecification formalisms: Minimal recursion semantics as dominance constraints. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Sapporo, Japan, pages 367–374. https://doi.org/10.3115/1075096.1075143.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinková, Dan Flickinger, Jan Hajic, and Zdenka Uresová. 2015. Semeval 2015 task 18: Broad-coverage semantic dependency parsing. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajic, Angelina Ivanova, and Yi Zhang. 2014. Semeval 2014 task 8: Broad-coverage semantic dependency parsing. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*. Association for Computational Linguistics and Dublin City University, Dublin, Ireland, pages 63–72. http://www.aclweb.org/anthology/S14-2008.

Stephan Oepen and Jan Tore Lønning. 2006. Discriminant-based mrs banking. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC-2006)*. European Language Resources Association (ELRA), Genoa, Italy. ACL Anthology Identifier: L06-1214.

Xiaochang Peng, Linfeng Song, and Daniel Gildea. 2015. A synchronous hyperedge replacement grammar based approach for amr parsing. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics, Beijing, China, pages 32–41. http://www.aclweb.org/anthology/K15-1004.

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)*. Association for Computational Linguistics, pages 2227–2237. https://doi.org/10.18653/v1/N18-1202.

Linfeng Song, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2018a. A graph-to-sequence model for amr-to-text generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 1616–1626. http://aclweb.org/anthology/P18-1150.

Linfeng Song, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2018b. N-ary relation extraction using graph-state lstm. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 2226–2235. http://aclweb.org/anthology/D18-1246.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 1556–1566. https://doi.org/10.3115/v1/P15-1150.

Xiaodan Zhu, Parinaz Sobhani, and Hongyu Guo. 2016. Dag-structured long short-term memory for semantic compositionality. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, pages 917–926. https://doi.org/10.18653/v1/N16-1106.