Semantic Graph Parsing with Recurrent Neural Network DAG Grammars

Citation for published version:

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
Semantic Graph Parsing with Recurrent Neural Network DAG Grammars

Federico Fancellu\textsuperscript{\textregistered} and Sorcha Gilroy\textsuperscript{\textregistered} and Adam Lopez\textsuperscript{\textregistered} and Mirella Lapata\textsuperscript{\textregistered}
\textsuperscript{\textregistered}Samsung AI Centre (SAIC), Toronto, Canada
\textsuperscript{\textregistered}Peak AI, Manchester, United Kingdom
\textsuperscript{\textregistered}University of Edinburgh, Edinburgh, United Kingdom

Abstract

Semantic parses are directed acyclic graphs (DAGs), so semantic parsing should be modeled as graph prediction. But predicting graphs presents difficult technical challenges, so it is simpler and more common to predict the linearized graphs found in semantic parsing datasets using well-understood sequence models. The cost of this simplicity is that the predicted strings may not be well-formed graphs. We present recurrent neural network DAG grammars, a graph-aware sequence model that ensures only well-formed graphs while sidestepping many difficulties in graph prediction. We test our model on the Parallel Meaning Bank—a multilingual semantic graphbank. Our approach yields competitive results in English and establishes the first results for German, Italian and Dutch.

1 Introduction

Semantic parsing is the task of mapping natural language to machine interpretable meaning representations, which in turn can be expressed in many different formalisms, including lambda calculus (Montague, 1973), dependency-based compositional semantics (Liang et al., 2011), frame semantics (Baker et al., 1998), abstract meaning representations (AMR; Banarescu et al. 2013), minimal recursion semantics (MRS; Copestake et al. 2005), and discourse representation theory (DRT; Kamp 1981).

 Explicitly or implicitly, a representation in any of these formalisms can be expressed as a directed acyclic graph (DAG). Consider the sentence “Every ship in the dock needs a big anchor”. Its meaning representation, expressed as a Discourse Representation Structure (DRS, Kamp 1981), is shown in Figure 1.\textsuperscript{1} A DRS is drawn as a box with two parts: the top part lists variables for discourse referents (e.g. $x_1$, $e_1$) and the bottom part can contain unary predicates expressing the type of a variable (e.g. ship, need), binary predicates specifying relationships between variables (e.g. PARTOF, TOPIC), logical operators expressing relationships between nested boxes (e.g. $\Rightarrow$, $\neg$), or binary discourse relations (e.g. RESULT, CONTRAST). To express a DRS as a graph, we represent each box as a node labeled $\square$; each variable as a node labeled by its associated unary predicate; and each binary predicate, logical operator, or discourse relation as an edge from the first argument to the second (Figure 2). To fully realize the representation as a DAG, additional transformations are sometimes necessary: in DRS, when a box represents a presupposition, as box $b_4$ does, the label of the node corresponding to the presupposed variable is marked (e.g. $x_2$/dock\textsuperscript{\textregistered}); and edges can be reversed (e.g. TOPIC($s_1$, $x_3$) becomes TOPICOF($s_1$, $x_3$)).

Since meaning representations are graphs, semantic parsing should be modeled as graph prediction. But how do we predict graphs? A popular approach is to predict the linearized graph—that is, the string representation of the graph found in most semantic graphbanks. Figure 3 illus-

---

\textsuperscript{1}For simplicity, our examples do not show time representations, though these are consistently present in our data.

![Figure 1: The discourse representation structure for “Every ship in the dock needs a big anchor”. For ease of reference in later figures, each box includes a variable corresponding to the box itself, at top right in gray.](image_url)
graph represents one style of linearization using PENMAN notation, in which graphs are written as well-bracketed strings which can also be interpreted as trees—note the correspondence between the tree-like structure of Figure 2 and the string in Figure 3. Each subtree is a bracketed string starting with a node variable and its label (e.g. \( b_2 / \square \)), followed by a list of relations corresponding to the outgoing edges of the node. A relation consists of the edge label prefixed with a colon (:), followed by either the subtree rooted at the target node (e.g. :DRS(\( x_1 / \text{ship} \)), \( :\text{PARTOF}(x_2 / \text{dock}^p) \)), or a reference to the target node (e.g. :PIVOT \( x_1 \)). By convention, if a node is the target of multiple edges, then the leftmost one is written as a subtree, and the remainder are written as references. Hence, every node is written as a subtree exactly once.

The advantage of predicting linearized graphs is twofold. The first advantage is that graph-bank datasets usually already contain linearizations, which can be used without additional work. These linearizations are provided by annotators or algorithms and are thus likely to be very consistent in ways that are beneficial to a learning algorithm. The second advantage is that we can use simple, well-understood sequence models to model them. But this simplicity comes with a cost: sequence models can predict strings that don’t correspond to graphs—for example, strings with ill-formed bracketings or unbound variable names. While it is often possible to fix these strings with pre- or post-processing, we would prefer to model the problem in a way that does not require this.

Models that predict graphs are complex and far less well-understood than models that predict sequences. Fundamentally, this is because predicting graphs is difficult: every graph has many possible linearizations, so from a probabilistic perspective, the linearization is a latent variable that must be marginalized out (Li et al., 2018). Groschwitz et al. (2018) model graphs as trees, interpreted as the (latent) derivation trees of a graph grammar; Lyu and Titov (2018) model graphs with a conditional variant of the classic Erdős and Rényi (1959) model, first predicting an alignment for each node of the output graph, and then predicting, for each pair of nodes, whether there is an edge between them. Buys and Blunsom (2017), Chen et al. (2018), and Damonte et al. (2017) all model graph generation as a sequence of actions, each aligned to a word in the conditioning sentence. Each of these models has a latent variable—a derivation tree or alignment—which must be accounted for via preprocessing or complex inference techniques.

Can we combine the simplicity of sequence prediction with the fidelity of graph prediction? We show that this is possible by developing a new model that predicts sequences through a simple string rewriting process, in which each rewrite corresponds to a well-defined graph fragment. Importantly, any well-formed string produced by our model has exactly one derivation, and thus no latent variables.

2 Graph-aware string rewriting

We use a grammar to model the process of rewriting. Formally, our grammar is a graph

\[
\begin{align*}
(b_1 / \square) \\
: \text{IMP}_1 (b_2 / \square) \\
: \text{DRS}(x_1 / \text{ship}) \\
: \text{PARTOF}(x_2 / \text{dock}^p)) \\
: \text{IMP}_2 (b_3 / \square) \\
: \text{DRS}(e_1 / \text{need}) \\
: \text{PIVOT} \ x_1 \\
: \text{THEME}(x_3 / \text{anchor}) \\
: \text{TOPICO} \ (s_1 / \text{big})))
\end{align*}
\]
grammar, specifically a restricted DAG grammar (Björklund et al., 2016), a type of context-free graph grammar designed to model linearized DAGs. Since linearized DAGs are strings, we present it as a string-rewriting system, which can be described more compactly than a graph grammar while making the connection to sequences more explicit. The correspondence between string rewriting and graph grammars is given in the Appendix.

A grammar in our model is defined by a set \( \Sigma \) of terminal symbols consisting of all symbols that can appear in the final string—brackets, variable types, node labels, and edge labels; a set \( N \) of \( n + 1 \) nonterminal symbols denoted by \( \{L, T_0, \ldots, T_n\} \), for some maximum value \( n \); an unbounded set \( V \) of variable references \( \{\$1, \$2, \ldots\} \); and a set of productions, which are defined below.

We say that \( T_0 \) is the start symbol, and for each symbol \( T_i \in N \), we say that \( i \) is its rank, and we say that \( L \) has a rank of 0. A nonterminal of rank \( i \) can be written as a function of \( i \) variable references—for example, we can write \( T_2(\$1, \$2) \). By convention, we write the rank-0 nonterminals \( L \) and \( T_0 \) without brackets. Productions in our grammar take the form \( \alpha \rightarrow \beta \), where \( \alpha \) is a function of rank \( i \) over \( \$1, \ldots, \$i \); and \( \beta \) is a linearized graph in PENMAN format, with each of its subtrees replaced by either a function or a variable reference. Optionally, the variable name in \( \beta \) may replace one of the variable references in \( \alpha \). All variable references in a production must appear at least twice. Hence every variable reference in \( \alpha \) must appear at least once in \( \beta \), and variables that do not appear in \( \alpha \) must appear at least twice in \( \beta \).

To illustrate, we will use the following grammar, which can generate the string in Figure 3, assuming \( L \) can also rewrite as any node label.

\[
\begin{align*}
T_0 &\rightarrow (b/\square :\text{IMP}_1 T_1(\$1) :\text{IMP}_2 T_1(\$1)) & (r_1) \\
T_0 &\rightarrow (x/L) & (r_2) \\
T_0 &\rightarrow (s/L) & (r_3) \\
T_0 &\rightarrow (x/L :\text{TOPICOf} T_0) & (r_4) \\
T_1(\$1) &\rightarrow (b/\square :\text{DRS} T_1(\$1)) & (r_5) \\
T_1(\$1) &\rightarrow (e/L :\text{PIVOT} \$1 :\text{THEME} T_0) & (r_6) \\
T_1(x) &\rightarrow (x/L :\text{PARTOf} T_0) & (r_7)
\end{align*}
\]

Our grammar derives strings by first rewriting the start symbol \( T_0 \), and at each subsequent step rewriting the leftmost function in the partially derived string, with special handling for variable references described below. A derivation is complete when no functions remain.

We illustrate the rewriting process in Figure 4. The start symbol \( T_0 \) at step 1 is rewritten by production \( r_1 \) in step 2, and the new \( b \) variable introduced at this step is deterministically renamed to the unique name \( b_1 \). In step 3, the leftmost \( T_1(\$1) \) is rewritten by production \( r_5 \), and the new \( b \) variable is likewise renamed to the unique \( b_2 \). All productions apply in this way, simply replacing a left-hand-side function with a right-hand-side expression. These rewrites are coupled with a mechanism to correctly handle multiple references to shared variables, as illustrated in Step 4 when production \( r_7 \) is applied. In this production, the left-hand-side function applies to the \( x \) variable naming the right-hand-side node. When this production applies, \( x \) is renamed to the unique \( x_1 \) as in previous steps, but because it appears in the left-hand-side, the reference \( \$1 \) is bound to this new variable name throughout the partially-derived string. In this way, the reference to \( x_1 \) is passed through the subsequent rewrites, becoming the target of a PIVOT edge at step 10. Derivations of a DAG grammar are context-free (Figure 5).\(^3\)

Our model requires an explicit grammar like the one in \( r_1 \ldots r_7 \), which we obtain by converting each DAG in the training data into a sequence of productions. The conversion yields a single, unique sequence of productions via a simple linear-time algorithm that recursively decomposes a DAG into subgraphs (Björklund et al., 2016). Each subgraph consists of single node and its outgoing edges, as exemplified by the PENMAN-formatted right-hand-sides of \( r_1 \) through \( r_7 \). Each outgoing edge points to a nonterminal symbol representing a subgraph. If a subgraph does not share any nodes with its siblings, it is represented by \( T_0 \). But if any subgraphs share a node, then a variable reference must refer for this node in the production associated with the lowest common ancestor of all its incoming edges. For example, in Figure 3, the common ancestor of the two edges targeting \( x_1 \) is the node \( b_1 \), so production \( r_1 \) must contain two copies of variable reference \( \$1 \) to account for this. A more mathematical account can be found along with a proof of correctness in Björklund et al. (2016).

\(^3\)However, the language of derived strings may not be context-free due to variable references.
Our implementation follows their description unchanged.

3 Neural Network Realizer

We model graph parsing with an encoder-decoder architecture that takes as input a sentence $w$ and outputs a directed acyclic graph $G$ derived using the rewriting system of Section 2. Specifically, we model its derivation tree in top-down, left-to-right order as a sequence of actions $a = a_1 ... a_n$, inspired by Recurrent Neural Network Grammars (RNNG, Dyer et al., 2016). As in RNNG, we use a stack to store partial derivations.

We model two types of actions: GEN-FRAG rewrites $T_i$ nonterminals, while GEN-LABEL rewrites $L$ nonterminals, always resulting in a leaf of the derivation tree. A third REDUCE action is applied whenever a subtree of the derivation tree is complete, and since the number of subtrees is known in advance, it is applied deterministically. For example, when we predict $r_1$, this determines that we must rewrite an $L$ and then recursively rewrite two copies of $T_i$ ($\S 1$) and then apply REDUCE. Hence graph generation reduces to predicting rewrites only.

We define the probability of generating graph $G$ conditioned of input sentence $w$ as follows:

$$p(G|w) = p(a|w) = \prod_{i=1}^{[a]} p(a_i|a_i < i, w)$$ (1)

**Input Encoder** We represent the $i$th word $w_i$ of input sentence $w = w_1 ... w_{|w|}$ using both learned and pre-trained word embeddings ($w_i$ and $w_i^p$ respectively), lemma embedding ($l_i$), part-of-speech embedding ($p_i$), universal semantic tag (Abzianidze and Bos, 2017) embedding ($u_i$), and dependency label embedding ($d_i$). An input $x_i$ is computed as the weighted concatenation of these features followed by a non-linear projection (with vectors and matrices in bold):

$$x_i = \tanh(W(1)[w_i; w_i^p; l_i; p_i; u_i; d_i])$$ (2)

Input $x_i$ is then encoded with a bidirectional LSTM, yielding contextual representation $h^c_i$.

**Graph decoder** Since we know in advance whether the next action is GEN-FRAG or GEN-LABEL, we use different models for them.

**GEN-FRAG.** If step $t$ rewrites a $T$ nonterminal, we predict the production $y_t$ that rewrites it using context vector $c_t$ and incoming edge embedding $e_t$. To obtain $c_t$, we use soft attention (Luong et al., 2015) and weight each input hidden representation $h^c_i$ to decoding hidden state $h^d_t$:

$$\text{score}(h^c_t, h^d_i) = h^c_t W(2)^d h^d_i$$

$$\alpha_t = \frac{\exp(\text{score}(h^c_t, h^d_i))}{\sum_{i'} \exp(\text{score}(h^c_t, h^d_{i'}))}$$

$$c_t = \sum_{i} \alpha_t h^c_i$$

$$y_t = W(3)c_t + W(4)e$$

(3)

The contribution of $c$ and $e$ is weighted by matrices $W(3)$ and $W(4)$, respectively.

We then update the stackLSTM representation using the embedding of the non-terminal fragment $y_t$ (denoted as $y^e_t$), as follows:

$$h^c_{t+1} = \text{LSTM}(y^e_t, h^c_t)$$ (4)

**GEN-LABEL.** Labels $L$ can be rewritten to either semantic constants (e.g., ‘speaker’, ‘now’, ‘hearer’) or unary predicates that often correspond to the lemmas of the input words (e.g., ‘love’) or. We predict the former using a model identical to the one for GEN-FRAG. For the latter, we use a selection mechanism to choose an input lemma to copy to output. We model selection following Liu et al. (2018), assigning each input lemma a score $o_{ji}$ that we then pass through a softmax layer to obtain a distribution:

$$o_{ji} = h^c_j W(5)^f h^c_i$$

$$p^\text{copy}_i = \text{SOFTMAX}(o_{ji})$$ (5)

where $h_i$ is the encoder hidden state for word $w_i$.

We allow the model to learn whether to use soft-attention or the selection mechanism through a binary classifier, conditioned on the decoder hidden state at time $t$, $h^c_t$. Similar to Equation (4), we update the stackLSTM with the embedding of terminal predicted.

**REDUCE.** When a reduce action is applied, we use an LSTM to compose the fragments on top of

---

4Universal semantic tags are language neutral tags intended to characterize lexical semantics.

5In the PMB, each terminal is annotated for sense (e.g. ‘n.01’, ‘s.01’) and presupposition (e.g. for ‘docks’ in Figure 3) as well. We predict both the sense tag and whether a terminal is presupposed or not independently conditioned on the current stackLSTM state and the embedding of the main terminal labels but are not used to update the state of the stackLSTM.
<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
<th>Production</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>START</td>
<td>start</td>
<td>$T_0$</td>
</tr>
<tr>
<td>2</td>
<td>GEN-FRAG</td>
<td>$r_1$</td>
<td>$(b_1/\square : \text{IMP}_1 T_1($1) : \text{IMP}_2 T_1($1))$</td>
</tr>
<tr>
<td>3</td>
<td>GEN-FRAG</td>
<td>$r_2$</td>
<td>$(b_1/\square : \text{IMP}_1 (b_2/\square : \text{DRS} T_1($1)) : \text{IMP}_2 T_1($1))$</td>
</tr>
<tr>
<td>4</td>
<td>GEN-FRAG</td>
<td>$r_3$</td>
<td>$(b_1/\square : \text{IMP}_1 (b_2/\square : \text{DRS} (x_1/L : \text{PARTOf} T_0) : \text{IMP}_2 T_1(x_1))$</td>
</tr>
<tr>
<td>5</td>
<td>GEN-LABEL</td>
<td>$L \rightarrow \text{ship}$</td>
<td>$(b_1/\square : \text{IMP}_1 (b_2/\square : \text{DRS} (x_1/\text{ship} : \text{PARTOf} T_0) : \text{IMP}_2 T_1(x_1))$</td>
</tr>
<tr>
<td>6</td>
<td>GEN-FRAG</td>
<td>$r_4$</td>
<td>$(b_1/\square : \text{IMP}_1 (b_2/\square : \text{DRS} (x_1/\text{ship} : \text{PARTOf} (x_2/L) : \text{IMP}_2 T_1(x_1))$</td>
</tr>
<tr>
<td>7</td>
<td>GEN-LABEL</td>
<td>$L \rightarrow \text{dock}^p$</td>
<td>$(b_1/\square : \text{IMP}_1 (b_2/\square : \text{DRS} (x_1/\text{ship} : \text{PARTOf} (x_2/dock^p)) : \text{IMP}_2 T_1(x_1))$</td>
</tr>
<tr>
<td>8</td>
<td>REDUCE</td>
<td>--</td>
<td>=</td>
</tr>
<tr>
<td>9</td>
<td>REDUCE</td>
<td>--</td>
<td>=</td>
</tr>
<tr>
<td>10</td>
<td>GEN-FRAG</td>
<td>$r_5$</td>
<td>$(b_1/\square : \text{IMP}_1 (b_2/\square : \text{DRS} (x_1/\text{ship} : \text{PARTOf} (x_2/dock^p)) : \text{IMP}_2 (b_1/\square : \text{DRS} T_1(x_1))$</td>
</tr>
<tr>
<td>11</td>
<td>GEN-FRAG</td>
<td>$r_6$</td>
<td>$(b_1/\square : \text{IMP}_1 (b_2/\square : \text{DRS} (x_1/\text{ship} : \text{PARTOf} (x_2/dock^p)) : \text{IMP}_2 (b_1/\square : \text{DRS} (e_1/L : \text{Pivot} x_1 : \text{THEME} T_0))$</td>
</tr>
</tbody>
</table>

Figure 4: A partial derivation of the string in Figure 3. The stack operations follow closely each step in the derivation, where GEN-FRAG and GEN-LABEL are invoked when rewriting a non-terminal $T$ and a terminal $L$ respectively. In the result of each step, the leftmost function is underlined, and is rewritten in the fragment in blue in the next step. On the other hand, a REDUCE operation is invoked when a generated fragment does not contain non-terminals $T$ to expand further (in this partial derivation, this is the case of the result of production $r_2$).

Figure 5: A derivation tree corresponding to Figure 4. Solid edges rewrite $T_n$ nonterminals, while dotted rewrite $L$ nonterminals.

the stack. Using the derivation tree in Figure 5 as reference, let $[c_1, ..., c_n]$ denote the embeddings of one or more sister nodes $r_i$ and $p_u$ the embedding of their parent node, which we refer to as children and parent fragments respectively. A reduce operation runs an LSTM over the children fragments and the parent fragment in order and then uses the final state $v$ to update the stack LSTM as follows:

$$i = [c_1, ..., c_n, p_u]$$
$$\overline{u} = \text{LSTM}(i_t, h_t)$$
$$h_{t+1}^d = \text{LSTM}(u, h_t^d)$$

The models are trained to minimize a cross-entropy loss objective $J$ over the sequence of gold actions $a_i$ in the derivation:

$$J = -\sum_{i=1}^{|a|} \log p(a_i)$$  \hspace{1cm} (6)

4 Experimental Setup

We evaluated our model on the Parallel Meaning Bank (PMB; Abzianidze et al. 2017), a semantic bank where sentences in English, Italian, German, and Dutch have been annotated following Discourse Representation Theory (Kamp and Reyle, 2013). Lexical predicates in PMB are in English, even for non-English languages. Since this is not compatible with our copy mechanism, we revert predicates to their original language by substituting them with the lemmas of the tokens they are aligned to. In our experiments we used both PMB v.2.1.0 and v.2.2.0\(^6\); we included the former release in order to compare against the state-of-the-art seq2seq system of van Noord et al. (2018).

\(^6\)English data is available for both releases at https://github.com/RikVNM/DRS_parsing; for the other languages, we used the officially released data available at http://pmb.let.rug.nl/data.php
In Section 2, we mentioned that given a production α → β, variable references in α should appear at least once in β (i.e., they should have the same rank). In all experiments so far, we did not model this constraint explicitly to investigate whether the model is able by default to predict rank correctly. However, in exploring model configurations we also report on whether adding this constraint leads to better performance.

7 We used word embeddings pre-trained with Glove and available at https://nlp.stanford.edu/projects/glove/.

8 Universal PoS tags, lemmas and dependency labels for all languages were obtained using pretrained UDpipe models available at http://ufal.mff.cuni.cz/udpipe#download; gold-standard universal semantic tags were extracted from the PMB release.
4.4 Cross-lingual Experiments

We conducted two sets of experiments: one monolingual (mono below) where we train and test on the same language and one cross-lingual (cross below), where we train a model on English and test it on one of the other three languages. The goal of our cross-lingual experiments is to examine whether we need data in a target language at all since the semantic representation itself is language agnostic and lexical predicates are dealt with via the copy mechanism. Most of the features mentioned above are cross-linguistic and therefore fit both mono and cross-lingual settings, with the exception of lemma and word embeddings, where we exclude the former and replaced the latter with multilingual word embeddings.

4.5 System settings

For training, we used the Adam optimizer (Kingma and Ba, 2014) with an initial learning rate of 0.001 and a decay rate of 0.1 every 10 epochs. Randomly initialized and pre-trained word embeddings have a dimensionality of 128 and 100 respectively, and all other features a dimensionality of 50. In all cross-lingual experiments, the pre-trained word embeddings have a dimensionality of 300. The LSTMs in the encoder and the decoder have a dimensionality of 150 and non-terminal and terminal embeddings during decoding have a dimensionality of 50. The system is trained for 30 epochs, with the best system chosen based on dev set performance.

4.6 Evaluation Metric

We evaluated our system by scoring the similarity between predicted and gold graphs. We used Counter (van Noord et al., 2018), an adaptation of Smatch (Cai and Knight, 2013) to Discourse Representation Structures where graphs are first transformed into a set of ‘source node – edge label – target node’ triples and the best mapping between the variables is found through an iterative hill-climbing strategy. Furthermore, Counter checks whether DRSs are well-formed in that all boxes should be connected, acyclic, with fully instantiated variables, and correctly assigned sense tags.

It is worth mentioning that there can be cases where our parser generates ill-formed graphs according to Counter; this is however not due to the model itself but to the way the graph is converted back in a format accepted by Counter.

All results shown are averages over 5 runs.

5 Results

System comparison Table 2 summarizes our results on the PMB gold data (v.2.1.0, test set). We compare our graph decoder against the system of van Noord et al. (2018) and our implementation of a seq2seq model, enhanced with a copy mechanism. Overall, we see that our graph decoder outperforms both models. Moreover, it reduces the number of ill-formed representations without any specific constraints or post-processing in order to ensure the well-formedness of the semantics of the output.

The PMB (v.2.1.0) contains a large number of silver standard annotations which have been only partially manually corrected (see Table 1). Following van Noord et al. (2018), we also trained our parser on both silver and gold standard data combined. As shown in Table 3, increasing the training data improves performance but the difference is not as dramatic as in van Noord et al. (2018). We found that this is because our parser requires graphs that are fully instantiated – all unary predicates (e.g. ship(x)) need to be present for the graph to be fully connected, which is often not the case for silver graphs. Our model is at a disadvantage since it could exploit less training data; during grammar extraction we could not process around 20K sentences and in some cases could not reconstruct the whole graph, as shown by the conversion score.

Model configurations Table 4 reports on various ablation experiments investigating which features and combinations thereof perform best. The experiments were conducted on the development set (PMB v2.2.0). We show a basic version of our seq2graph model with word embeddings, to which we add information about rank (+restrict).

---

9We experimented with embeddings obtained with iterative procrustes (available at https://github.com/facebookresearch/MUSE) and with Guo et al. (2016)’s ‘robust projection’ method where the embedding of non-English words is computed as the weighted average of English ones. We found the first method to perform better on cross-lingual word similarity tasks and used it in our experiments.

10The conversion score is computed using Counter where we consider the converted graph representation as a ‘predicted’ DRS and the original DRS structure as gold data. As a comparison, converting the gold DRS structures yields a conversion score of 99.8%.
Table 2: Model performance (Precision, Recall, F1) on PMB data (v.2.1.0, test set); models were trained on gold standard data.

<table>
<thead>
<tr>
<th>Model</th>
<th>P</th>
<th>R</th>
<th>F1</th>
<th>illformed</th>
</tr>
</thead>
<tbody>
<tr>
<td>van Noord et al. (2018)</td>
<td>—</td>
<td>72.80</td>
<td>—</td>
<td>20%</td>
</tr>
<tr>
<td>seq2seq + copy</td>
<td>75.57</td>
<td>67.27</td>
<td>71.18</td>
<td>4.12%</td>
</tr>
<tr>
<td>seq2graph</td>
<td>75.51</td>
<td>71.69</td>
<td>73.55</td>
<td>0.40%</td>
</tr>
</tbody>
</table>

Table 3: Model performance (Precision, Recall, F1) on PMB data (v.2.1.0, test set); models were trained on gold and silver standard data combined.

<table>
<thead>
<tr>
<th>Model</th>
<th>#sents</th>
<th>conversion</th>
<th>score</th>
<th>F1</th>
<th>illformed</th>
</tr>
</thead>
<tbody>
<tr>
<td>van Noord et al. (2018)</td>
<td>73,778</td>
<td>100%</td>
<td>82.7</td>
<td>1.10%</td>
<td></td>
</tr>
<tr>
<td>seq2seq+copy</td>
<td>56.694</td>
<td>89.58%</td>
<td>74.67</td>
<td>13.45%</td>
<td></td>
</tr>
<tr>
<td>seq2graph</td>
<td>56.694</td>
<td>89.58%</td>
<td>77.1</td>
<td>0.90%</td>
<td></td>
</tr>
</tbody>
</table>

We also experimented with the full gamut of additional features (+feats) as well as with ablations of individual feature classes. For comparison, we also show the performance of a graph-to-string model (seq2seq+copy).

As can be seen, all linguistic features seem to improve performance. Restricting fragment selection by rank does not seem to improve the overall result showing that our baseline model is already able to predict fragments with the correct rank throughout the derivation. Subsequent experiments report results with this model using all linguistic features, unless otherwise specified.

Cross-lingual experiments Results on DRT parsing for languages other than English are reported in Table 5. There is no gold standard training data for non-English languages in the PMB (v.2.2.0). We therefore trained our parser on silver standard data but did use the provided gold standard data for development and testing (see Table 1). We present two versions of our parser, one where we train and test on the same language (s2g mono-silver) and another one where a model is trained on English but tested on the other languages (s2g cross-silver). We also show the results of a sequence-to-sequence model enhanced with a copy mechanism.

In the monolingual setting, our graph parser outperforms the seq2seq baseline by a large margin; we hypothesize this is due to the large percentage of ill-formed semantics, mostly due to training on silver data. The difference in performance between our cross-lingual parser and the monolingual parser for all languages is small, and in Dutch the two parsers perform on par, suggesting that English data and language independent features can be leveraged to build parsers in other languages when data is scarce or event absent. We also conducted various ablation studies to examine the contribution of individual features to cross-linguistic semantic parsing. Our experiments revealed that universal semantic tags are most useful, while the multilingual word embeddings that we have tested with are not. We refer the interested reader to the supplementary material for more detail on these experiments.

Error Analysis We further analyzed the output of our parser to gain insight as to what parts of meaning representation are still challenging. Table 6 shows a more detailed break-down of system output as computed by Counter, where operators (e.g., negation, implication), roles (i.e., binary relations, such as ‘Theme’), concepts (i.e., unary predicates like ‘ship’), and synsets (i.e., sense tags like ‘n.01’) are scored separately. Synsets are further broken down into into ‘Nouns’, ‘Verbs’, ‘Adverbs’, and ‘Adjectives’. We compare our best
Table 6: $F_1$-scores of fine-grained evaluation on the PMB v.2.2.0 development set; the seq2graph models trained on gold (left) and with gold and silver data combined (right) are compared.

<table>
<thead>
<tr>
<th></th>
<th>seq2graph(gold)</th>
<th>seq2graph(silver)</th>
</tr>
</thead>
<tbody>
<tr>
<td>all clauses</td>
<td>74.42</td>
<td>76.36</td>
</tr>
<tr>
<td>DRS operators</td>
<td>88.05</td>
<td>89.36</td>
</tr>
<tr>
<td>Roles</td>
<td>72.95</td>
<td>74.03</td>
</tr>
<tr>
<td>Concepts</td>
<td>71.13</td>
<td>74.42</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Synsets</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>– Nouns</td>
<td>77.13</td>
<td>81.30</td>
</tr>
<tr>
<td>– Verbs</td>
<td>55.63</td>
<td>55.49</td>
</tr>
<tr>
<td>– Adverbs</td>
<td>44.44</td>
<td>42.11</td>
</tr>
<tr>
<td>– Adjectives</td>
<td>61.67</td>
<td>58.48</td>
</tr>
</tbody>
</table>

-sense          76.91  78.93

seq2graph models (+feats) trained on the PMB v.2.2.0, gold and gold+silver data respectively.

Adding silver data helps with semantic elements (operators, roles and concepts), but does not in the case of sense prediction where the only category that benefits from additional data are is nouns. We also found that ignoring the prediction of sense tags altogether helps with the performance of both models.

6 Conclusions

In this paper we have introduced a novel graph parser that can leverage the power and flexibility of sequential neural models while still operating on graph structures. Heavy preprocessing tailored to a specific formalism is replaced by a flexible grammar extraction method that relies solely on the graph while yielding performance that is on par or better than string-based approaches. Future work should focus on extending our parser to other formalisms (AMR, MRS, etc.). We also plan to explore modelling alternatives, such as taking different graph generation orders into account (bottom-up vs. top-down) as well as predicting the components of a fragment (type, number of edges, edge labels) separately.

Acknowledgments

We thank Yova Kementchedjhieva, Andreas Gri- vas and the three anonymous reviewers for their useful comments. This work was done while Federico Fancellu was a post-doctoral researcher at the University of Edinburgh. The views expressed are his own and do not necessarily represent the views of Samsung Research. We gratefully acknowledge the support of the European Research Council (Fancellu, Lapata; award number 681760, “Translating Multiple Modalities into Text”); and the EPSRC Centre for Doctoral Training in Data Science, funded by the UK Engineering and Physical Sciences Research Council (Gilroy; grant EP/L016427/1) and the University of Edinburgh (Gilroy).

References


Abzianidze, L. and Bos, J. (2017). Towards universal semantic tagging. In IJCS 201712th International Conference on Computational SemanticsShort pa-


Baner respectively.


This figure "derivation1.PNG" is available in "PNG" format from:

http://arxiv.org/ps/1910.00051v1
This figure "derivation2.PNG" is available in "PNG" format from:

http://arxiv.org/ps/1910.00051v1
This figure "graph.PNG" is available in "PNG" format from:

http://arxiv.org/ps/1910.00051v1