Semantic Graph Parsing with Recurrent Neural Network DAG Grammars

Citation for published version:
Fancellu, F, Gilroy, S, Lopez, A & Lapata, M 2019, Semantic Graph Parsing with Recurrent Neural Network DAG Grammars. in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing.* Association for Computational Linguistics, pp. 2769–2778, 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing, Hong Kong, Hong Kong, 3/11/19. https://doi.org/10.18653/v1/D19-1278

Digital Object Identifier (DOI):
10.18653/v1/D19-1278

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

Semantic Graph Parsing with Recurrent Neural Network DAG Grammars

Federico Fancellu[⊙] and **Sorcha Gilroy**[☽] and **Adam Lopez**[⊕] and **Mirella Lapata**[⊕]
[⊙]Samsung AI Centre (SAIC), Toronto, Canada
[☽]Peak AI, Manchester, United Kingdom
[⊕]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.[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. ⇒, ¬), or binary discourse relations (e.g., RESULT, CONTRAST). To express a DRS as a graph, we represent each box as a node labeled □; 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^P); and edges can be reversed (e.g. TOPIC(s_1, x_3) becomes TOPICOF(s_1, x_3)).

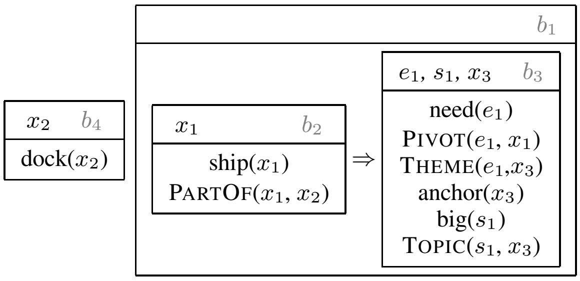

Figure 1: The discourse representation structure for "*Every ship in the dock needs a big anchor*". For ease of reference in later figures, each box includes a variable corresponding to the box itself, at top right in gray.

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-

[1]For simplicity, our examples do not show time representations, though these are consistently present in our data.

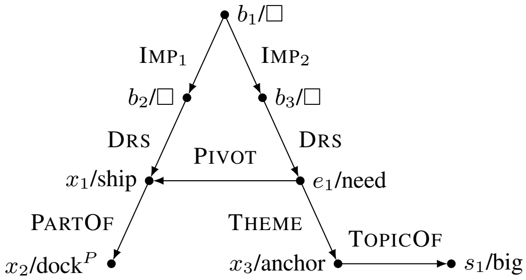

Figure 2: The DRS of Figure 1 expressed as a DAG.

```
(b1/□
   :IMP1(b2/□
      :DRS(x1/ship
         :PARTOF(x2/dockp)))
   :IMP2(b3/□
      :DRS(e1/ need
         :PIVOT x1
         :THEME(x3/ anchor
            :TOPICOF(s1/ big)))))
```

Figure 3: The DAG of Figure 2 expressed as a string.

trates 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.[2] Each subtree is a bracketed string starting with a node variable and its label (e.g. $b_2/\Box$), 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/ship :PARTOF(x_2/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 (Gu et al., 2016; Jia and Liang, 2016; van Noord et al., 2018) 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.

[2]Although PENMAN notation is now closely associated with AMR, it can represent quite arbitrary graphs as strings. Our actual implementation does not use PENMAN notation, but we use it here for expository purposes since it is relatively familiar; the underlying ideas are unchanged.

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 Erdös 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. We evaluate our model on the Parallel Meaning Bank (PMB, Abzianidze et al. 2017), a multilingual corpus of sentences paired with DRS representations. Our model performs competitively on English, and better than sequence models in German, Italian, and Dutch.

2 Graph-aware string rewriting

We use a grammar to model the process of rewriting. Formally, our grammar is a *graph* *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 Σ 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 α is a function of rank i over $\$1, \ldots, \$i$; and β 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 β may replace one of the variable references in α. All variable references in a production must appear at least twice. Hence every variable reference in α must appear at least once in β, and variables that do not appear in α must appear at least twice in β.

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.

$$T_0 \rightarrow (b/\Box \text{ :I}\textsc{mp}_1\ T_1(\$1) \text{ :I}\textsc{mp}_2\ T_1(\$1)) \quad (r_1)$$

$$T_0 \rightarrow (x/L) \quad (r_2)$$

$$T_0 \rightarrow (s/L) \quad (r_3)$$

$$T_0 \rightarrow (x/L \text{ :T}\textsc{opic}\text{O}\textsc{f}\ T_0) \quad (r_4)$$

$$T_1(\$1) \rightarrow (b/\Box \text{ :D}\textsc{rs}\ T_1(\$1)) \quad (r_5)$$

$$T_1(\$1) \rightarrow (e/L \text{ :P}\textsc{ivot}\ \$1 \text{ :T}\textsc{heme}\ T_0) \quad (r_6)$$

$$T_1(x) \rightarrow (x/L \text{ :P}\textsc{art}\text{O}\textsc{f}\ T_0) \quad (r_7)$$

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 $a = a_1 \dots a_{|a|}$, 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 $T_1(\$1)$ 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:

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

Input Encoder We represent the ith word w_i of input sentence $w = w_1 \dots w_{|w|}$ using both learned and pre-trained word embeddings ($\mathbf{w}_i$ and $\mathbf{w}_i^p$ respectively), lemma embedding ($\mathbf{l}_i$), part-of-speech embedding ($\mathbf{p}_i$), universal semantic tag (Abzianidze and Bos, 2017) embedding ($\mathbf{u}_i$), and dependency label embedding ($\mathbf{d}_i$).[4] An input $\mathbf{x}_i$ is computed as the weighted concatenation of these features followed by a non-linear projection (with vectors and matrices in **bold**):

$$\mathbf{x}_i = \tanh(\mathbf{W}^{(1)}[\mathbf{w}_i; \mathbf{w}_i^p; \mathbf{l}_i; \mathbf{p}_i; \mathbf{u}_i; \mathbf{d}_i]) \quad (2)$$

Input $\mathbf{x}_i$ is then encoded with a bidirectional LSTM, yielding contextual representation $\mathbf{h}_i^e$.

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

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

GEN-FRAG. If step t rewrites a T nonterminal, we predict the production y_t that rewrites it using context vector $\mathbf{c}_t$ and incoming edge embedding $\mathbf{e}_t$. To obtain $\mathbf{c}_t$ we use *soft attention* (Luong et al., 2015) and weight each input hidden representation $\mathbf{h}_i^e$ to decoding hidden state $\mathbf{h}_t^d$:

$$\begin{aligned} \text{score}(\mathbf{h}_i^e, \mathbf{h}_t^d) &= \mathbf{h}_i^e \mathbf{W}^{(2)} \mathbf{h}_t^d \\ \alpha_{ti} &= \frac{\exp(\text{score}(\mathbf{h}_i^e, \mathbf{h}_t^d))}{\sum_{i'} \exp(\text{score}(\mathbf{h}_{i'}^e, \mathbf{h}_t^d))} \\ \mathbf{c}_t &= \sum_{i=1}^{n} \alpha_{ti} \mathbf{h}_i^e \\ \mathbf{y}_t &= \mathbf{W}^{(3)} \mathbf{c}_t + \mathbf{W}^{(4)} \mathbf{e} \end{aligned} \quad (3)$$

The contribution of $\mathbf{c}$ and $\mathbf{e}$ is weighted by matrices $\mathbf{W}^{(3)}$ and $\mathbf{W}^{(4)}$, respectively.

We then update the stackLSTM representation using the embedding of the non-terminal fragment $\mathbf{y}_t$ (denoted as $\mathbf{y}_t^e$), as follows:

$$\mathbf{h}_{t+1}^d = \text{LSTM}(\mathbf{y}_t^e, \mathbf{h}_t^d) \quad (4)$$

GEN-LABEL. Labels L can be rewritten to either semantic constants (e.g., 'speaker', 'now', 'hearer') or unary predicates that often corresponds 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 distibution:

$$\begin{aligned} o_{ji} &= \mathbf{h}_j^{dT} \mathbf{W}^{(5)} \mathbf{h}_i^e \\ p_i^{copy} &= \text{SOFTMAX}(o_{ji}) \end{aligned} \quad (5)$$

where $\mathbf{h}_i$ 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, $\mathbf{h}_t^d$. Similar to Equation (4), we update the stackLSTM with the embedding of terminal predicted.[5]

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

[5]In the PMB, each terminal is annotated for sense (e.g. 'n.01', 's.01') and presupposition (e.g. for 'dock^p' 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.

Step	Action	Production	Result
1	START	start	$\underline{T_0}$
2	GEN-FRAG	r_1	$(b_1/\square$:IMP$_1$ $\underline{T_1(\$1)}$:IMP$_2$ $T_1(\$1))$
3	GEN-FRAG	r_5	$(b_1/\square$:IMP$_1$ $(b_2/\square$:DRS $\underline{T_1(\$1)})$:IMP$_2$ $T_1(\$1))$
4	GEN-FRAG	r_7	$(b_1/\square$:IMP$_1$ $(b_2/\square$:DRS $(x_1/\underline{L}$:PARTOF $T_0)$:IMP$_2$ $T_1(x_1))$
5	GEN-LABEL	$L \to$ ship	$(b_1/\square$:IMP$_1$ $(b_2/\square$:DRS $(x_1/$ship :PARTOF $\underline{T_0})$:IMP$_2$ $T_1(x_1))$
6	GEN-FRAG	r_2	$(b_1/\square$:IMP$_1$ $(b_2/\square$:DRS $(x_1/$ship :PARTOF $(x_2/\underline{L}))$:IMP$_2$ $T_1(x_1))$
7	GEN-LABEL	$L \to$ dock^p	$(b_1/\square$:IMP$_1$ $(b_2/\square$:DRS $(x_1/$ship :PARTOF $(x_2/$dock$^p))$:IMP$_2$ $\underline{T_1(x_1)})$
8	REDUCE	$--$	$=$
9	REDUCE	$--$	$=$
8	GEN-FRAG	r_5	$(b_1/\square$:IMP$_1$ $(b_2/\square$:DRS $(x_1/$ship :PARTOF $(x_2/$dock$^p))$:IMP$_2$ $(b_3/\square$:DRS $\underline{T_1(x_1)}))$
9	GEN-FRAG	r_6	$(b_1/\square$:IMP$_1$ $(b_2/\square$:DRS $(x_1/$ship :PARTOF $(x_2/$dock$^p))$:IMP$_2$ $(b_3/\square$:DRS $(e_1/\underline{L}$:PIVOT x_1 :THEME $T_0)))$
10	GEN-LABEL	$L \to$ need	$(b_1/\square$:IMP$_1$ $(b_2/\square$:DRS $(x_1/$ship :PARTOF $(x_2/$dock$^p))$:IMP$_2$ $(b_3/\square$:DRS $(e_1/$need :PIVOT x_1 :THEME $\underline{T_0})))$

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).

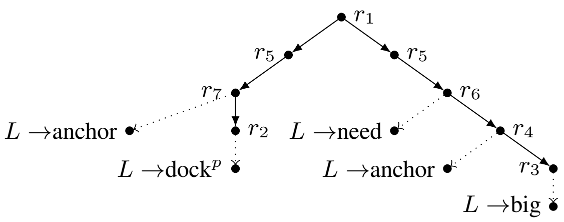

Figure 5: A derivation tree corresponding to Figure 4. Solid edges rewrite T_n nonterminals, while dotted rewrite L nonterminals.

the stack. Using the derivation tree in Figure 5 as reference, let $[\mathbf{c}_1, ..., \mathbf{c}_n]$ 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 u to update the stack LSTM as follows:

$$\mathbf{i} = [\mathbf{c}_1 ... \mathbf{c}_n, \mathbf{p}_u]$$
$$\overrightarrow{\mathbf{u}} = LSTM(\mathbf{i}_t, \mathbf{h}^c_t)$$
$$\mathbf{h}^d_{t+1} = LSTM(\mathbf{u}, \mathbf{h}^d_t)$$

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

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

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 orignal 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/RikVN/DRS_parsing`; for the other languages, we used the officially released data available at `http://pmb.let.rug.nl/data.php`

Gold	#inst.	train #frags	avg. rank	dev #inst.	test #inst.
en	4,574	1,105	1.64	682	650
it	—	—	—	387	386
de	—	—	—	736	735
nl	—	—	—	356	355

Silver	#inst.	train #frags	avg. rank
en	52,120	11,206	1.90
it	2,546	1,051	1.52
de	3,274	1,781	1.54
nl	874	789	1.45

Table 1: Data statistics of PMB v.2.2.0.

Statistics on the data and the grammar extracted from v.2.2.0 are reported in Table 1.

4.1 Converting DRSs to Graphs

In this section we discuss how DRSs are converted to *acyclic*, *single-rooted*, and *fully-instantiated* graphs (i.e., how to translate Figure 1 to Figure 2). In general, box structures are converted to acyclic graphs by rendering boxes and lexical predicates as nodes, while conditions, operators, and discourse relations become edge labels between these nodes.

We consider *main* boxes (see b_2, b_3, and b_4 in Figure 1 separately from *presuppositional* boxes (see b_1), which represent instances that are presupposed in the wider discourse context (e.g., definite expressions). Using Figure 2 as an example, b_2, b_3 and b_4 become nodes in the graph and material implication (IMP stands for $\Rightarrow$) becomes an edge label. If an operator or a relation is binary, as in this case, we number the edge label so as to preserve the order of the operands.

For each node in a *main* box, we expand the graph by adding all relations and variables belonging to it. We identify the head of the first relation or the first referent mentioned as the *head variable*. These are 'ship (x_1)' for b_2, and 'need(e_1)' for b_3. We attach the head variable as a child of the box-node and follow the relations recursively to expand the subgraph. If while expanding a graph a variable in a condition is part of a presuppositional box, we introduce it as a new node and add to its label the superscript ^{p}. When expanding the DAG along the edge PartOf, since x_2 is also in the presuppositional box in Figure 1, we attach the node 'dock^p'. Graphs extracted this way are mostly acyclic except for adjectival phrases and relative clauses where state variables can be themselves root (e.g., big "a.01" s_1). We get rid of these extra roots by reversing the direction of the edge involved and adding an '-of' to the edge label to flag this change (see TOPIC-OF).

4.2 System Comparison

We compared the performance of our graph parser (seq2graph below) with a sequence-to-sequence model (enchanced with a copy mechanism) which decodes to a string linearization of the graph similar to the one shown in Figure 3 (seq2seq + copy below). We also compare against the recently proposed model of van Noord et al. (2018); they introduce a seq2seq model that generates a DRS as a concatenation of *clauses*, essentially a flat version of the standard box notation. The decoded string is made aware of the overall graph structure during preprocessing where variables are replaced by indices indicating when they were first introduced and their recency. In contrast, we model the graph structure explicitly. van Noord et al. (2018) experimented with both word and character-based models, as well as with an ensemble of both, using word embedding features. Since all our models are word-based, we compare our results with their best word model, using word embedding features only (trained using 10-fold cross validation).

4.3 Model Configurations

In addition to word embeddings[7], we also report on experiments which make use of additional features. Specifically, for each word we add information about its universal PoS tag, lemma, universal semantic tag, and dependency label.[8]

In Section 2, we mentioned that given a production $\alpha \rightarrow \beta$, 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 constrant 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.[9]

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 an 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 illformed 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*.[10]

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).

[9] We 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.

[10] The 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 sturctures yields a conversion score of 99.8%.

	P	R	F1	illformed
van Noord et al. (2018)	—	—	72.80	20%
seq2seq + copy	75.57	67.27	71.18	4.12%
seq2graph	75.51	71.69	**73.55**	**0.40%**

Table 2: Model performance (Precision, Recall, F_1) on PMB data (v.2.1.0, test set); models were trained on *gold* standard data.

	#sents	conversion score	F1	illformed
van Noord et al. (2018)	73,778	100%	82.7	1.10%
seq2seq+copy	56.694	89.58%	74.67	13.45%
seq2graph	56.694	89.58%	77.1	0.90%

Table 3: Model performance (Precision, Recall, F_1) on PMB data (v.2.1.0, test set); models were trained on *gold* and *silver* standard data combined.

We also experimented with the full gamut of additional features (+feats) as well as with ablations of individual feature classes. For comparsion, 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

	P	R	F1	illformed
seq2seq + copy	60.29	74.09	66.48	10.60%
seq2graph	**70.69**	**74.46**	**72.53**	0.10%
+restrict	70.50	74.64	72.51	0.70%
+feats	**72.51**	**76.44**	**74.42**	0.60%
-lemmas	71.53	76.28	73.83	0.32%
-pos	72.11	75.99	74.00	0.35%
-semtag	70.21	74.45	72.27	0.53%
-words	70.45	74.59	72.46	0.64%
-dep	72.16	76.40	74.22	0.50%

Table 4: Model performance (Precision, Recall, F_1) on PMB (v2.2.0, development set).

Italian	P	R	F1	ill
s2s + copy mono-silver	61.33	72.42	66.41	14.80
s2g mono-silver	74.50	77.27	75.86	0.10
s2g cross-silver	70.35	71.91	71.12	0.00
German	P	R	F1	ill
s2s + copy mono-silver	56.86	65.40	60.83	10.67
s2g mono-silver	66.44	69.34	67.86	0.80
s2g cross-silver	66.14	65.72	63.50	0.40
Dutch	P	R	F1	ill
s2s + copy mono-silver	54.27	64.17	58.81	10.67
s2g mono-silver	63.50	68.37	65.84	0.86
s2g cross-silver	62.94	67.32	65.06	0.50

Table 5: Model performance across languages (Precision, Recall, F_1). Results are reported on the PMB test set (v.2.2.0) for each language.

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 scarse 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

	seq2graph(gold)	seq2graph(silver)
all clauses	74.42	**76.36**
DRS operators	88.05	**89.36**
Roles	72.95	**74.03**
Concepts	71.13	**74.42**
Synsets – Nouns	77.13	**81.80**
– Verbs	**55.63**	55.49
– Adverbs	**44.44**	42.11
– Adjectives	**61.67**	58.48
-sense	76.91	**78.93**

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.

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 oders 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 Grivas 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., Bjerva, J., Evang, K., Haagsma, H., van Noord, R., Ludmann, P., Nguyen, D.-D., and Bos, J. (2017). The parallel meaning bank: Towards a multilingual corpus of translations annotated with compositional meaning representations. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 242–247, Valencia, Spain. Association for Computational Linguistics.

Abzianidze, L. and Bos, J. (2017). Towards universal semantic tagging. In *IWCS 201712th International Conference on Computational SemanticsShort papers*.

Baker, C. F., Fillmore, C. J., and Lowe, J. B. (1998). The Berkeley FrameNet project. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 1*, pages 86–90, Montreal, Quebec, Canada.

Banarescu, L., Bonial, C., Cai, S., Georgescu, M., Griffitt, K., Hermjakob, U., Knight, K., Koehn, P., Palmer, M., and Schneider, N. (2013). Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria. Association for Computational Linguistics.

Björklund, H., Drewes, F., and Ericson, P. (2016). Between a rock and a hard place–uniform parsing for hyperedge replacement dag grammars. In *International Conference on Language and Automata Theory and Applications*, pages 521–532. Springer.

Buys, J. and Blunsom, P. (2017). Robust incremental neural semantic graph parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1215–1226.

Cai, S. and Knight, K. (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.

Chen, B., Sun, L., and Han, X. (2018). Sequence-to-action: End-to-end semantic graph generation for semantic parsing. *arXiv preprint arXiv:1809.00773*.

Copestake, A., Flickinger, D., Pollard, C., and Sag, I. A. (2005). Minimal recursion semantics: An introduction. *Research on language and computation*, 3(2-3):281–332.

Damonte, M., Cohen, S. B., and Satta, G. (2017). An incremental parser for abstract meaning representation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, volume 1, pages 536–546.

Dyer, C., Kuncoro, A., Ballesteros, M., and Smith, N. A. (2016). Recurrent neural network grammars. In *Proceedings of NAACL-HLT*, pages 199–209.

Erdös, P. and Rényi, A. (1959). On random graphs. *Publicationes Mathematicae*, 6:290–297.

Groschwitz, J., Lindemann, M., Fowlie, M., Johnson, M., and Koller, A. (2018). Amr dependency parsing with a typed semantic algebra. *arXiv preprint arXiv:1805.11465*.

Gu, J., Lu, Z., Li, H., and Li, V. O. (2016). Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1631–1640.

Guo, J., Che, W., Yarowsky, D., Wang, H., and Liu, T. (2016). A representation learning framework for multi-source transfer parsing. In *AAAI*, pages 2734–2740.

Jia, R. and Liang, P. (2016). Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12–22, Berlin, Germany.

Kamp, H. (1981). A theory of truth and semantic representation. *Formal semantics-the essential readings*, pages 189–222.

Kamp, H. and Reyle, U. (2013). *From discourse to logic: Introduction to modeltheoretic semantics of natural language, formal logic and discourse representation theory*, volume 42. Springer Science & Business Media.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. (2018). Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*.

Liang, P., Jordan, M., and Klein, D. (2011). Learning dependency-based compositional semantics. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 590–599, Portland, Oregon.

Liu, J., Cohen, S. B., and Lapata, M. (2018). Discourse representation structure parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 429–439. Association for Computational Linguistics.

Luong, T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421.

Lyu, C. and Titov, I. (2018). Amr parsing as graph prediction with latent alignment. *arXiv preprint arXiv:1805.05286*.

Montague, R. (1973). The proper treatment of quantification in ordinary English. In Hintikka, K., Moravcsik, J., and Suppes, P., editors, *Approaches to Natural Language*, volume 49 of Synthese Library, pages 221–242. Springer Netherlands.

van Noord, R., Abzianidze, L., Toral, A., and Bos, J. (2018). Exploring neural methods for parsing discourse representation structures. *arXiv preprint arXiv:1810.12579*.