Accurate polyglot semantic parsing with DAG grammars

Federico Fancellu Akos Kádár Ran Zhang Afsaneh Fazly
Samsung AI Centre Toronto (SAIC Toronto)
{federico.f, ran.zhang, a.fazly}@samsung.com

Abstract

Semantic parses are directed acyclic graphs (DAGs), but in practice most parsers treat them as strings or trees, mainly because models that predict graphs are far less understood. This simplification, however, comes at a cost: there is no guarantee that the output is a well-formed graph. A recent work by Fancellu et al. (2019) addressed this problem by proposing a graph-aware sequence model that utilizes a DAG grammar to guide graph generation. We significantly improve upon this work, by proposing a simpler architecture as well as more efficient training and inference algorithms that can *always* guarantee the well-formedness of the generated graphs. Importantly, unlike Fancellu et al., our model does not require language-specific features, and hence can harness the inherent ability of DAG-grammar parsing in multilingual settings. We perform monolingual as well as multilingual experiments on the Parallel Meaning Bank (Abzianidze et al., 2017). Our parser outperforms previous graph-aware models by a large margin, and closes the performance gap between string-based and DAG-grammar parsing.

1 Introduction

Semantic parsers map a natural language utterance into a machine-readable meaning representation, thus helping machines understand and perform inference and reasoning over natural language data. Various semantic formalisms have been explored as the target meaning representation for semantic parsing, including dependency-based compositional semantics (Liang et al., 2013), abstract meaning representation (AMR, Banarescu et al., 2013), minimum recursion semantics (MRS, Copestake et al., 2005), and discourse representation theory (DRT, Kamp, 1981). Despite meaningful differences across formalisms or parsing models, a representation in any of these formalisms can be expressed as a *directed acyclic graph* (DAG).

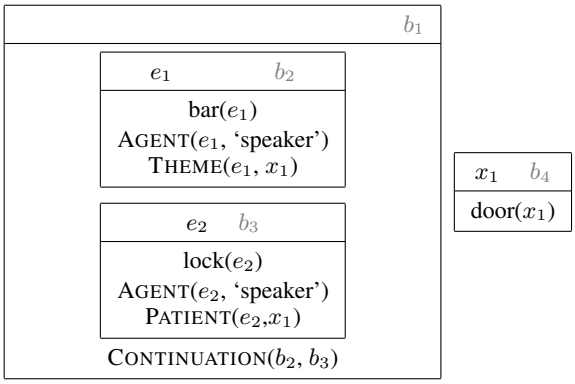


Figure 1: The discourse representation structure for '*We barred the door and locked it*'. For ease of reference in later figures, each box includes a variable corresponding to the box itself, at top right in gray.

Consider for instance the sentence '*We barred the door and locked it*', whose meaning representation as a Discourse Representation Structure (DRS) is shown in Figure 1. A DRS is usually represented as a set of nested boxes (e.g. b_1), containing variable-bound discourse referents (e.g. 'lock(e_2)'), semantic constants (e.g. 'speaker'), predicates (e.g. AGENT) expressing relations between variables and constants, and discourse relations between the boxes (e.g. CONTINUATION). This representation can be expressed as a DAG by turning referents and constants into vertices, and predicates and discourse relations into connecting edges, as shown in Figure 2.

How can we parse a sentence into a DAG? Commonly-adopted approaches view graphs as strings (e.g. van Noord and Bos, 2017; van Noord et al., 2018), or trees (e.g. Zhang et al., 2019a; Liu et al., 2018), taking advantage of the linearized graph representations provided in annotated data (e.g. Figure 3, where the graph in Figure 2 is represented in PENMAN notation (Goodman, 2020)).

Findings of the Association for Computational Linguistics: EMNLP 2020, pages 3567–3580

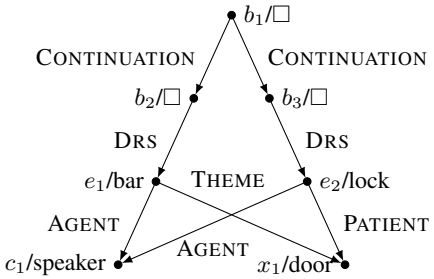


Figure 2: The DRS of Figure 1 expressed as a DAG.

```
(b1/□
  :CONTINUATION1(b2/□
    :DRS(e1/bar
      :AGENT(c1/speaker)))
      :THEME(x1/door^p)))
  :CONTINUATION2(b3/□
    :DRS(e2/ lock
      :AGENT c1
      :PATIENTx1))
```

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

An advantage of these linearized representations is that they allow for the use of well-understood sequential decoders and provide a general framework to parse into any arbitrary formalism. However, these representations are unaware of the overall graph strucure they build as well as of reentrant semantic relations, such as coordination, coreference, and control, that are widespread in language. Parsers such as Zhang et al. (2019b) although able to generate reentrancies in their output, they do so by simply predicting pointers back to already generated nodes.

Parsing directly into DAGs, although desirable, is less straightforward than string-based parsing. Whereas probabilistic models of strings and trees are ubiquitous in NLP, at present, it is an active problem in modern formal language theory to develop formalisms that allow to define probability distributions over DAGs of practical interest.[1] A successful line of work derives semantic graphs using *graph grammars* that allow to generate a graph by rewriting non-terminal symbols with graph fragments. Among these, *hyperedge replacement grammar* (HRG) has been explored for parsing into semantic graphs (Habel, 1992; Chiang et al., 2013). However, parsing with HRGs is not practical due to its complexity and large number of possible derivations per graph (Groschwitz et al., 2015). Thus, work has looked at ways of constraining the space of possible derivations, usually in the form of alignment or syntax (Peng et al., 2015). For example, Groschwitz et al. (2018) and Donatelli et al. (2019) extracted fine-grained typed grammars whose productions are aligned to the input sentence and combined over a dependency-like structure. Similarly, Chen et al. (2018) draw on constituent parses to combine together HRG fragments.

Björklund et al. (2016) show that there exists a restricted subset of HRGs, Restricted DAG grammar (RDG), that provides a *unique derivation per graph*. A unique derivation means that a graph is generated by a unique sequence of productions, which can then be predicted using sequential decoders, without the need of an explicit alignment model or an underlying syntactic structure. Furthermore, the grammar places hard constraints on the rewriting process, which can be used to guarantee the well-formedness of output graphs during decoding. Drawing on this result, a recent work by Fancellu et al. (2019) introduces *recurrent neural network RDGs*, a sequential decoder that models graph generation as a rewriting process with an underlying RDG. However, despite the promising framework the approach in FA19[2] falls short in several aspects.

In this paper, we address these shortcomings, and propose an *accurate, efficient, polyglot model* for Neural RDG parsing. Specifically, our contributions are as follows:

Grammar: In practice, RDGs extracted from training graphs can be large and sparse. We show a novel factorization of the RDG production rules that reduces the sparsity of the extracted grammars. Furthermore, we make use of RDGs extracted on fully human annotated training data to filter out samples from a larger noisy machine-generated dataset that cannot be derived using such grammars. We find that this strategy not only drastically reduces the size of the grammar, but also improves the final performance.

Model: FA19 use a syntactic parsing inspired architecture, a stackLSTM, trained on a gamut of syntactic and semantic features. We replace this with a novel architecture that allows for batched input, while adding a multilingual transformer encoder that relies on word-embedding features only.

Constrained Decoding: We identify a limitation in the decoding algorithm presented by FA19, in that it only partially makes use of the well-

[1] See Gilroy (2019) for an extensive review of the issue.

[2] For the sake of brevity, we refer to Fancellu et al. (2019) as FA19 throughout the paper.

formdness constraints of an RDG. We describe the source of this error, implement a correction and show that we can guarantee well-formed DAGs.

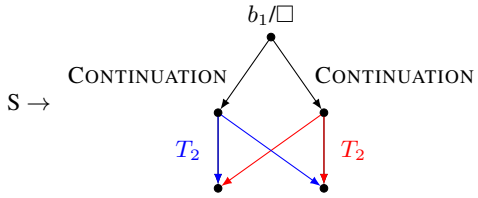
Multilinguality: Training data in languages other than English is often small and noisy. FA19 addressed this issue with cross-lingual models using features available only for a small number of languages, but did not observe improvements over monolingual baselines in languages other than English. We instead demonstrate the flexibility of RDGs by extracting a joint grammar from graph annotations in different languages. At the same time, we make full use of our multilingual encoder to build a *polyglot* model that can accept training data in any language, allowing us to experiment with different combinations of data. Our results tell a different story where models that use combined training data from multiple languages always substantially outperform monolingual baselines.

We test our approach on the Parallel Meaning Bank (PMB, Abzianidze et al., 2017), a multilingual graphbank. Our experimental results demonstrate that our new model outperforms that of FA19 by a large margin on English while fully exploiting the power of RDGs to always guarantee a well-formed graph. We also show that the ability of simultaneously training on multiple languages substantially improves performance for each individual language. Importantly, we close the performance gap between graph-aware parsing and state-of-the-art string-based models.

2 Restricted DAG Grammar

We model graph generation as a process of graph rewriting with an underlying grammar. Our grammar is a *restricted DAG grammar* (RDG, Björklund et al., 2016), a type of context-free grammar designed to model linearized DAGs. For ease of understanding, we represent fragments in grammar productions as strings. This is shown in Figure 4, where the right-hand-side (RHS) fragment can be represented as its left-to-right linearization, with reentrant nodes flagged by a dedicated $ symbol.

An RDG is a tuple $\langle P, N, \Sigma, S, V \rangle$ where P is a set of **productions** of the form $\alpha \rightarrow \beta$; N is the set of **non-terminal symbols** $\{L, T_0, \cdots, T_n\}$ up to a maximum number of n; Σ is the set of **terminal symbols**; S is the **start symbol**; V is an unbounded set of **variable references** {$1, $2,...}, whose role is described below.



S → (b_1/□ :CONTINUATION T_2($1, $2) :CONTINUATION T_2($1, $2))

Figure 4: An example production for a grammar. The graph fragment on the right-hand side can be replaced with a string representing its depth-first traversal.

The left-hand-side (LHS) α of a production $p \in P$ is a function $T_i \in N$ (where i is the **rank**) that takes i variable references as arguments. Variable references are what ensure the well-formedness of a generated graph in an RDG, by keeping track of how many reentrancies are expected in a derivation as well as how they are connected to their neighbouring nodes. Rank, in turn, is an indication of how many reentrancies are present in a graph derivation. For instance, in the graph fragment in Figure 4, given that there are two variable references and a non-terminal of rank 2, we are expecting two reentrant nodes at some point in the derivation. The RHS β is a *typed* fragment made up of three parts: a variable v describing the *semantic* type[3], a label non-terminal L, and a list of tuples $\langle e, s \rangle$ where e is an edge label from a set of labels E and s is either a non-terminal function T or a variable reference. The non-terminal L can only be rewritten as a terminal symbol $l \in \Sigma$. If a node is reentrant, we mark it with a superscript * over v. Variable references are percolated down the derivation and are replaced once a reentrant variable v^* is found on the RHS.

Following FA19, we show a complete derivation in Figure 5 that reconstructs the graph in Figure 2. Our grammar derives strings by first rewriting the start symbol S, a non-terminal function T_0. At each subsequent step, the leftmost non-terminal function in the partially derived string is rewritten, with special handling for variable references described below. A derivation is complete when no non-terminals remain.

Variable references are resolved when applying a production that maps a reentrant variable name

[3]where $v \in \{e, x, t, s, c, b\}$ (e=event, x=individual, s=state, t=time, c=constant, b=box). Note that type is optional: in other formalisms like AMR variable do not have a semantic type, hence one may name all variables with the same letter.

Step	Production	Result
1	r_1	$(b_1/L$:CONT $T_2(\$1, \$2)$:CONT $T_2(\$1, \$2))$
2	r_2	$(b_1/\square$:CONT $(b_2/L$:DRS $T_2(\$1, \$2))$:CONT $T_2(\$1, \$2))$
3	r_3	$(b_1/\square$:CONT $(b_2/\square$:DRS $(e_1/L$:AGENT $T_1(\$1)$:THEME $T_1(\$2))))$:CONT $T_2(\$1, \$2))$
4	r_4	$(b_1/\square$:CONT $(b_2/\square$:DRS $(e_1/bar$:AGENT (c^*/L) :THEME $T_1(\$2))))$:CONT $T_2(c, \$2))$
5	r_5	$(b_1/\square$:CONT $(b_2/\square$:DRS $(e_1/bar$:AGENT $(c^*/speaker)$:THEME $(x^*/L))))$:CONT $T_2(c, x))$
6	r_2	$(b_1/\square$:CONT $(b_2/\square$:DRS $(e_1/bar$:AGENT $(c^*/speaker)$:THEME $(x^*/door^p))))$:CONT $(b_3/\square$:DRS $T_2(c, x)))$
7	r_6	$(b_1/\square$:CONT $(b_2/\square$:DRS $(e_1/bar$:AGENT $(c^*/speaker)$:THEME $(x^*/door^p))))$:CONT $(b_3/\square$:DRS $(e_2/lock$:AGENT c :PATIENT $x)$

Figure 5: A full RDG derivation for the graph in Figure 2. At each step t the leftmost non-terminal T_n (in blue) is rewritten into a fragment (underlined) and its label non-terminal L (in red) replaced with a terminal. Variable references are percolated down the derivation unless a reentrant variable v^* is found (step 4 and 5).

to a reference, as shown for production r_4, where the variable c is mapped to \$1. Once this mapping is performed, all instances of \$1 in the RHS are replaced by the corresponding variable name. In this way, the reference to c is kept track of during the derivation becoming the target of AGENT in r_6. Same applies in r_5 where x is mapped to \$2.

All our fragments are delexicalized. This is achieved by the separate non-terminal L that at every step is rewritten in the corresponding terminal label (e.g. *bar*). Delexicalization allows to reduce the size of grammar and factorize the prediction of fragments and labels separately.

However, DAG grammars can still be large due to the many combinations of how edge labels and their corresponding non-terminals can appear in a fragment. For this reason, we propose a further simplification where edge labels are replaced with placeholders $\hat{e}_1 ... \hat{e}_{|e|}$, which we exemplify using the production in Figure 4 as follows:

$$\text{S} \rightarrow (b_1/L\ \hat{e}_1\ T_2(\$1, \$2)\ \hat{e}_2\ T_2(\$1, \$2))$$

After a fragment is predicted, placeholders are then replaced with actual edge labels by a dedicated module (see § 3.2 for more details).

Comparison with Groschwitz et al. (2018)'s AM algebra. RDG is very similar to other graph grammars proposed for semantic parsing, in particular to Groschwitz et al. (2018)'s AM algebra used for AMR parsing. Groschwitz et al. (2018)'s framework relies on a fragment extraction process similar to ours where each node in a graph along with its outgoing edges makes up a fragment. However, the two grammars differ mainly in how typing and as a consequence, composition is thought of: whereas in the AM algebra both the fragments themselves and the non-terminal edges are assigned thematic types (e.g. S[ubject], O[bject], MOD[ifier]), we only place rank information on the non-terminals and assign a more generic semantic type to the fragment.

The fine-grained thematic types in the AM algebra add a level of linguistic sophistication that RDG lacks, in that fragments fully specify the roles a word is expected to fill. This ensures that the output graphs are always *semantically* well-formed; in that AM algebra behaves very similar to CCG. However this sophistication not only requires ad-hoc heuristics that are tailored to a specific formalism (AMR in this case) but also relies on alignment information with the source words.

On the other hand, our grammar is designed to predict a graph structure in sequential models. Composition is constrained by the rank of a non-terminal so to ensure that at each decoding step the model is always aware of the placement of reentrant nodes. However, we do not ensure semantic well-formedness in that words are predicted separately from their fragments and we do not rely on alignment information. In that our grammar extraction algorithm does not rely on any heuristics and can be easily applied to any semantic formalism.

3 Architecture

Our model is an encoder-decoder architecture that takes as input a sentence and generates a DAG G as a sequence of fragments with their corresponding labels, using the rewriting system in § 2. In what follows we describe how we obtain the logits for

each target prediction, all of which are normalized with the softmax function to yield probability distributions. A detailed diagram of our architecture is shown in Figure 7 in Appendix A.

3.1 Encoder

We encode the input sentence $w_1, \ldots, w_{|n|}$ using a pre-trained multilingual BERT (mBERT) model (Devlin et al., 2018).[4] The final word-level representations are obtained through mean-pooling the sub-word representations of mBERT computed using the Wordpiece algorithm (Schuster and Nakajima, 2012). We do not rely on any additional (language-specific) features, hence making the encoder polyglot. The word vectors are then fed to a two-layer BiLSTM encoder, whose forward and backward states are concatenated to produce the final token encodings $\mathbf{s}_1^{enc}, \ldots, \mathbf{s}_n^{enc}$.

3.2 Decoder

The backbone of the decoder is a two layer LSTM, with two separate attention mechanisms for each layer. Our decoding strategy follows steps similar to those in Figure 5. At each step we first predict a delexicalized fragment f_t, and substitute a terminal label l_t in place of L. We initialize the decoder LSTM with the encoder's final state $\mathbf{s_n^{enc}}$. At each step t, the network takes as input $[\mathbf{f}_{t-1}; \mathbf{l}_{t-1}]$, the concatenation of the embeddings of the fragment and its label output at the previous time step. At $t = 0$, we initialize both fragment and label encodings with a $\langle \text{START} \rangle$ token. The first layer in the decoder is responsible for predicting fragments. The second layer takes as input the output representations of the first layer, and predicts terminal labels. The following paragraphs provide details on the fragment and label predictions.

Fragment prediction. We make the prediction of a fragment dependant on the embedding of the parent fragment and the decoder history. We define as *parent fragment* the fragment containing the non-terminal the current fragment is rewriting; for instance, in Figure 5, the fragment in step 1 is the parent of the fragment underlined in step 2. Following this intuition, at time t, we concatenate the hidden state of the first layer $\mathbf{h}_t^1$ with a context vector $\mathbf{c}_t^1$ and the embedding of its parent fragment $\mathbf{u}_t$. The logits for fragment f_t are predicted with a single linear layer $\mathbf{W}^f[\mathbf{c}_t^1; \mathbf{u}_t; \mathbf{h}_t^1] + \mathbf{b}$. We compute $\mathbf{c}_t^1$ using a standard soft attention mechanism (Xu et al., 2015) as follows, where $\mathbf{s_{1:N}^{enc}}$ represents the concatenation of all encoding hidden states:

$$\mathbf{c}_t^1 = \sum_i^N \alpha_i \mathbf{s}_i^{enc} \tag{1}$$

$$\mathbf{a} = \text{MLP}^1[\mathbf{h}_t^1; \mathbf{s}_{1:N}^{enc}] \tag{2}$$

$$\alpha_i = \frac{e^{\mathbf{a}_i}}{\sum_j \mathbf{a}_j} \tag{3}$$

$$\text{MLP}^1(\mathbf{x}) = \text{ReLU}(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{4}$$

Label prediction. Terminal labels in the output graph can either correspond to a lemma in the input sentence (e.g. 'bar', 'lock'), or to a semantic constant (e.g. 'speaker'). We make use of this distinction by incorporating a selection mechanism that learns to choose to predict either a lemma from the input or a token from a vocabulary of L. We concatenate the hidden state of the second layer $\mathbf{h}_t^2$ with the embedding of the fragment predicted at the current time-step $\mathbf{f}_t$ and the second layer context vector $\mathbf{c}_t^2$. Let us refer to this representation as $\mathbf{z}_t = [\mathbf{f}_t; \mathbf{h}_t^2; \mathbf{c}_t^2]$. The context vector for the second layer is computed in the same way as $\mathbf{c}_t^1$, but using $\mathbf{h}_t^2$ in place of $\mathbf{h}_t^1$ and separate attention MLP parameters. To compute the logits for label-prediction we apply a linear transformation to the encoder representations $\mathbf{e} = \mathbf{W}^s\mathbf{s_{1:N}^{enc}}$. We concatenate the resulting vector with the label embedding matrix $\mathbf{L}$ and compute the dot product $\mathbf{z}_t^T[\mathbf{e}; \mathbf{L}]$ to obtain the final unnormalized scores jointly for all tokens in the input and L.

In the PMB, each label is also annotated with its sense tag and information about whether it is presupposed in the context or not. We predict the former, s_t, from a class of sense tags S extracted from the training data, and the latter, p_t, a binary variable, by passing $\mathbf{z}_t$ two distinct linear layers to obtain the logits for each.

Edge factorization. In § 2, we discussed how we made grammars even less sparse by replacing the edge labels in a production fragment with placeholders. From a modelling perspective, this allows to factorize edge label prediction, where the decoder first predicts all the fragments in the graph and then predicts the edge labels $e_i ... e_{|e|}$ that substitute in place of the placeholders.

To do so, we cache the intermediate representations $\mathbf{z}_t$ over time. We use these as features, to

[4] Available through the HuggingFace Transformers library (Wolf et al., 2019) at https://huggingface.co/.

replace the edge-placeholders $\hat{e}_i$ with the corresponding true edge labels e_i. To obtain the edge-label logits we pass the second-layer representation for the child fragment $\mathbf{z}^c$ and parent fragment $\mathbf{z}^p$ to a pairwise linear layer: $\mathbf{W}^e[\mathbf{W}^c\mathbf{z}^c \odot \mathbf{W}^p\mathbf{z}^p]$.

3.3 Graph-aware decoding

At inference time, our graph decoder rewrites non-terminals left-to-right by choosing the fragment with the highest probability, and then predicts terminal and/or edge labels. The rank of a non-terminal and the variable references it takes as arguments place a hard constraint on the fragment that rewrites in its place (as shown in § 2). Only by satisfying these constraints, the model can ensure well-formedness of generated graphs.

By default, our decoder does not explicitly follow these constraints and can substitute a non-terminal with any fragment in the grammar. This is to assess whether a vanilla decoder can learn to substitute in a fragment that correctly matches a non-terminal. On top of the vanilla decoder, we then exploit these hard constrains in two different ways, as follows:

Rank prediction. We incorporate information about rank as a *soft constraint* during learning by having the model predict it at each time step. This means that the model can still predict a fragment whose rank and variable references do not match those of a non-terminal but it is guided not to do so. We treat rank prediction as a classification task where we use the same features as fragment prediction that we then pass to a linear layer: $\mathbf{r}_t = \mathbf{W}^r[\mathbf{c}_t^1; \mathbf{u}_t; \mathbf{h}_t^1] + \mathbf{b}^r$. Note that the range of predicted ranks is determined by the training grammar so it is not possible to generate a rank that has not been observed and doesn't have associated rules.

Constrained decoding. We explicitly ask the model to choose only amongst those fragments that can match the rank and variable references of a non-terminal. This may override model predictions but always ensures that a graph is well-formed. To ensure well-formedness, FA19 only checks for rank. This can lead to infelicitous consequences. Consider for instance the substitution in Figure 6. Both fragments at the bottom of the middle and right representations are of rank 2 but whereas the first allows for the edges to refer back to the reentrant nodes, the second introduces an extra reentrant node, leaving therefore one of the reentrant nodes disconnected. Checking just for rank is therefore not enough; one also needs to check whether a reentrant node that will substitute in a variable reference has already been generated. If not, any fragment of the same rank can be accepted. If such a node already exists, only fragments that do not introduce another reentrant node can be accepted. This constrained decoding strategy is what allows us to *always* generate well-formed graphs; we integrate this validation step in the decoding algorithm when selecting the candidate fragment.

Finally, we integrate these hard constraints in the softmax layer as well. Instead of normalizing the logits across all fragment types with a single softmax operation, we normalize them separately for each rank. The errors are only propagated through the subset of parameters in $\mathbf{W}^f$ and $\mathbf{b}^f$ responsible for the logits within the target rank r_t.

3.4 Training objective

Our objective is to maximize the log-likelihood of the full graph $P(G|s)$ approximated by the decomposition over each prediction task separately:

$$\sum_t \log P(f_t) + \log P(\ell_t) + \log P(r_t) + \log P(s_t) + \log P(p_t) + \sum_i \log P(e_i) \tag{5}$$

where f_t is the fragment; ℓ_t is the label; r_t is the (optional) rank of f_t; s_t and p_t are the sense and presupposition label of terminal label ℓ_t; $e_i ... e_{|e|}$ are the edge labels of f_t. To prevent our model from overfitting, rather than directly optimizing the log-likelihoods, we apply label smoothing for each prediction term (Szegedy et al., 2016).

4 Experimental setup

4.1 Data

We evaluate our parser on the Parallel Meaning Bank (Abzianidze et al., 2017), a multilingual graph bank where sentences in four languages (English (en), Italian (it), German (de) and Dutch (nl)) are annotated with their semantic representations in the form of Discourse Representation Structures (DRS). We test on v.2.2.0 to compare with previous work, and present the first results on v.3.0 on all four languages. We also present results when training on both *gold* and *silver* data, where the latter is ∼10x larger but contains machine-generated

	# training instances	# fragments +edge label	# fragments -edge label	avg. rank
PMB2.2.0-g	4585	1196	232	1.56
PMB2.2.0-s	63960	17414	2586	2.85
PMB3-g	6618	1695	276	2.22
PMB3-s	94776	36833	6251	3.01
PMB3-it	2743	1827	378	2.32
PMB3-de	5019	4025	843	2.61
PMB3-nl	1238	1338	318	2.29

Table 1: Statistics for the grammars extracted from the PMB (*g* - gold; *s* - silver).

parses, of which only a small fraction has been manually edited. Statistics for both versions of the PMB are reported in Appendix B.

Our model requires an explicit grammar which we obtain by *automatically* converting each DAG in the training data[5] into a sequence of productions. This conversion follows the one in FA19 with minor changes; we include details in Appendix C.

Statistics regarding the grammars extracted from the PMB are presented in Table 1, where along with the number of training instances and fragments, we report average rank — an indication of how many reentrancies (on average) are present in the graphs. RDGs can be large especially in the case of silver data, where incorrect parses lead to a larger number of fragments extracted and more complex, noisy constructions, as attested by the higher average ranks. More importantly, we show that removing the edge labels from the fragments leads to a drastic reduction in the number of fragments, especially for the silver corpora.

4.2 Evaluation metrics

To evaluate our parser, we need to compare its output DRSs to the gold-standard graph structures. For this, we use the Counter tool of Van Noord et al. (2018), which calculates an F-score by searching for the best match between the variables of the predicted and the gold-standard graphs. Counter's search algorithm is similar to the evaluation system SMATCH for AMR parsing (Cai and Knight, 2013).

There might be occurrences where our graph is deemed ill-formed by Counter; we assign these graphs a score of 0. The ill-formedness is however not due to the model itself but to specific requirements placed on the output DRS by the Counter script.

[5]Our DAGs are different from the DRG (Discourse Representation Graphs) of Basile and Bos (2013); we elaborate more on this in Appendix C.

	P	R	F_1
baseline	80.0	70.9	75.2
+ rank-prediction	81.0	72.3	76.4
+ constrained-decoding	80.5	75.2	77.8
+ edge-factorization	**82.5**	**78.5**	**80.4**
ours-best + silver	**83.8**	**80.6**	**82.2**
ours-best + filtering	83.1	80.5	81.8

Table 2: Ablation results on the *dev* portion of PMB2.2.0. The top half shows results for models trained on *gold* data only. The bottom half shows results of models trained on *silver*+*gold* data.

5 Experimental Results

We first present results of ablation experiments to understand which model configuration performs best (§ 5.1). We then compare our best-performing model with several existing semantic parsers (§ 5.2), and present our model's performance in multilingual settings (§ 5.3).

5.1 Ablation experiments

Table 2 shows results for our model in various settings. Our *baseline* is trained on *gold* data alone, uses a full grammar and performs *un*constrained decoding, with and without *rank prediction*. Note that unconstrained decoding could lead to ill-formed graphs. To better understand the effect of this, we compare the performance of the baseline with a model that uses *constrained* decoding and thus always generates well-formed graphs. We train all our models on a single TitanX GPU v100. We report hyperparameters and other training details in Appendix D.

Our results are different from that of FA19, who show that a baseline model outperforms one with constrained decoding. Not only we find that constrained decoding outperforms the baseline, but we observe that without it, 26 graphs (∼4%) are ill-formed. In addition, our results show that predicting edge labels separately from fragments (edge factorization) leads to a substantial improvement in performance, while also drastically reducing the size of the grammar (as shown in Table 1).

We also train our best-performing model (ours-best) on the silver and gold data combined (+*silver*). This is to assess whether more data, albeit noisy, results in better performance. However, noisy data can lead to noisy grammar; to reduce this noise, we experiment with first extracting a grammar from

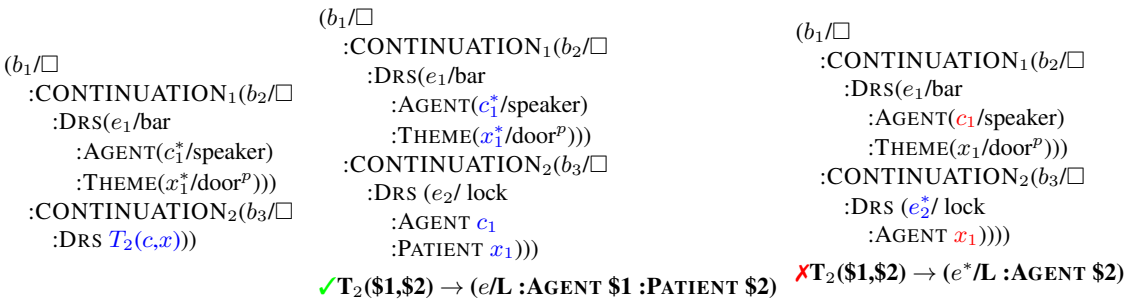


Figure 6: Example of correct (middle) and wrong (right) substitution of non-terminal function (left, in blue) during constrained decoding.

the gold training set, and use it to filter the silver set, where only instances that can be derived using the gold grammar are kept (+*filtering*). The filtering results in smaller grammar (232 vs. 2586 fragments), while at the same time sacrificing only a small percentage of training instances (10%).

van Noord et al. (2019), Liu et al. (2019) and FA19 found that models trained on silver data requires an additional training fine-tuning on gold data alone to achieve the best performance; we also follow this strategy in our experiments. Overall, results show that adding silver data improves performance, and that filtering the input silver data leads only to a slight loss in performance while keeping the size of the grammar small.

5.2 Comparison to previous work

We compare our best-performing model against previous work on PMB2.2.0. We first compare the performance on models trained solely on gold data. Besides the DAG-grammar parser of FA19, we compare with the transition-based stackLSTM of Evang (2019) that utilizes a buffer-stack architecture to predict a DRS fragment for each input token using the alignment information in the PMB; our graph parser does not make use of such information and solely relies on attention.

We then compare our best-performing model with two models trained on gold plus silver data. van Noord et al. (2019) is a seq2seq parser that decodes an input sentence into a concatenation of *clauses*, essentially a flattened version of the boxes in Figure 1. Similar to FA19, their model also uses a wide variety of language-dependent features, including part-of-speech, dependency and CCG tags, while ours relies solely on word embeddings. In this respect, our model is similar to Liu et al. (2019)'s that uses the same architecture as the model of van Noord et al. (2019) but replaces the LSTM encoder with a transformer model, without the use of additional features.

	P	R	F_1
Fancellu et al. (2019)	-	-	73.4
Evang (2019)	-	-	74.4
ours-best	84.5	81.3	**82.9**
van Noord et al. (2019)	-	-	**86.8**
Liu et al. (2019)	85.8	84.5	85.1
ours-best + silver	86.1	83.6	84.9

Table 3: Comparison with previous work on the *test* portion of PMB2.2.0. Results in the top half are for models trained on *gold* data, whereas bottom half shows results for models trained on *silver+gold* data.

Results are summarized in Table 3. When trained on gold data alone, our model outperforms previous models by a large margin, without relying on alignment information or extra features besides word embeddings. When trained on silver+gold, we close the performance gap with state-of-the-art models that decode into strings, despite relying solely on multilingual word embeddings.

5.3 Multilingual experiments

Table 4 shows the results on languages other than English. In our multilingual experiments, we first train and test monolingual models in each language. In addition, we perform zero-shot experiments by training a model on English and testing it on other languages (*cross-lingual*). We also take full advantage of the fact that our models rely solely on multilingual word embeddings, and experiment with two other multilingual settings: The *bilingual* models are trained on data in English plus data in a target language (tested on the target language). The *polyglot* models combine training data of all four languages (tested on each language). Parameters for all languages in the *bilingual* and *polyglot* mod-

PMB2.2.0				
	en	de	nl	it
FA19 (monolingual)	-	67.9	65.8	75.9
FA19 (cross-lingual)	-	63.5	65.1	72.1
ours (cross-lingual)	-	**73.4**	**73.9**	**76.9**
ours-best (various) trained and tested on PMB3				
monolingual	**80**	64.2	60.9	71.5
cross-lingual	-	**73.2**	74.1	75.2
bilingual	-	71.8	**76.0**	77.7
polyglot	79.8	72.5	74.1	**77.9**

Table 4: Results for the multilingual experiments on the test sets for PMB2.2.0 (top half) and PMB3.0 (bottom half). For the sake of brevity, we report only F_1 scores here, and refer the reader to Table 6 in Appendix E for Precision and Recall values.

els are fully shared.

FA19 only experiment with a cross-lingual model trained with additional language-dependent features, some of which available only for a small number of languages (on PMB2.2.0). We therefore compare our cross-lingual models with theirs on PMB2.2.0. We then introduce the first results on PMB3, where we experiment with the other two multilingual settings.

Our results tell a different story from FA19, where all of our multilingual models (bilingual, polyglot and cross-lingual) outperform the corresponding monolingual baselines. We hypothesize this is mainly due to the fact that for languages other than English, only small silver training data are available and adding a large gold English data might help dramatically with performance. This hypothesis is also reinforced by the fact that a cross-lingual model training on English data alone can reach a performance comparable to the other two models.

6 Conclusions

In this paper, we have introduced a graph parser that can fully harness the power of DAG grammars in a seq2seq architecture. Our approach is efficient, fully multilingual, always guarantees well-formed graphs and can rely on small grammars, while outperforming previous graph-aware parsers in English, Italian, German and Dutch by large margin. At the same time, we close the gap between string-based and RDG-based decoding. In the future, we are planning to extend this work to other semantic formalisms (e.g. AMR, UCCA) as well as testing on other languages, so to encourage work in languages other than English.

Acknowledgments

We thank three anonymous reviewers for their useful comments. Research was conducted at Samsung AI Centre Toronto and funded by Samsung Research, Samsung Electronics Co.,Ltd.

References

Lasha Abzianidze, Johannes Bjerva, Kilian Evang, Hessel Haagsma, Rik Van Noord, Pierre Ludmann, Duc-Duy Nguyen, and Johan Bos. 2017. The parallel meaning bank: Towards a multilingual corpus of translations annotated with compositional meaning representations. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of the 7th linguistic annotation workshop and interoperability with discourse*, pages 178–186.

Valerio Basile and Johan Bos. 2013. Aligning formal meaning representations with surface strings for wide-coverage text generation. In *Proceedings of the 14th European Workshop on Natural Language Generation*, pages 1–9, Sofia, Bulgaria. Association for Computational Linguistics.

Henrik Björklund, Frank Drewes, and Petter Ericson. 2016. Between a rock and a hard place–uniform parsing for hyperedge replacement DAG grammars. In *Proceedings of the International Conference on Language and Automata Theory and Applications*, pages 521–532. Springer.

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*, 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)*, pages 408–418.

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)*, pages 924–932.

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT 2019*, pages 4171–4186.

Lucia Donatelli, Meaghan Fowlie, Jonas Groschwitz, Alexander Koller, Matthias Lindemann, Mario Mina, and Pia Weißenhorn. 2019. Saarland at MRP 2019: Compositional parsing across all graphbanks. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 66–75.

Kilian Evang. 2019. Transition-based DRS parsing using stack-LSTMs. In *Proceedings of the IWCS Shared Task on Semantic Parsing*, Gothenburg, Sweden. Association for Computational Linguistics.

Federico Fancellu, Sorcha Gilroy, Adam Lopez, and Mirella Lapata. 2019. Semantic graph parsing with recurrent neural network DAG grammars. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pages 2769–2778.

Sorcha Gilroy. 2019. Probabilistic graph formalisms for meaning representations.

Michael Wayne Goodman. 2020. Penman: An open-source library and tool for amr graphs. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 312–319.

Jonas Groschwitz, Alexander Koller, and Christoph Teichmann. 2015. Graph parsing with s-graph grammars. 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)*, pages 1481–1490.

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)*.

Annegret Habel. 1992. *Hyperedge replacement: grammars and languages*, volume 643. Springer Science & Business Media.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, Melbourne, Australia. Association for Computational Linguistics.

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

Percy Liang, Michael I Jordan, and Dan Klein. 2013. Learning dependency-based compositional semantics. *Computational Linguistics*, 39(2):389–446.

Jiangming Liu, Shay B Cohen, and Mirella Lapata. 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.

Jiangming Liu, Shay B Cohen, and Mirella Lapata. 2019. Discourse representation structure parsing with recurrent neural networks and the transformer model. In *Proceedings of the IWCS Shared Task on Semantic Parsing*.

Rik van Noord, Lasha Abzianidze, Antonio Toral, and Johan Bos. 2018. Exploring neural methods for parsing discourse representation structures. *Transactions of the Association for Computational Linguistics*, 6:619–633.

Rik van Noord and Johan Bos. 2017. Neural semantic parsing by character-based translation: Experiments with abstract meaning representations. *Computational Linguistics in the Netherlands (CLIN)*.

Rik van Noord, Antonio Toral, and Johan Bos. 2019. Linguistic information in neural semantic parsing with multiple encoders. In *Proceedings of the 13th International Conference on Computational Semantics-Short Papers*, pages 24–31.

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*, pages 32–41.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and korean voice search. In *Proceedings of the 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152. IEEE.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune BERT for text classification? In *Proceedings of the China National Conference on Chinese Computational Linguistics*, pages 194–206. Springer.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

Rik Van Noord, Lasha Abzianidze, Hessel Haagsma, and Johan Bos. 2018. Evaluating scoped meaning representations. In *Proceedings of the 11th International Conference on Language Resources and Evaluation (LREC)*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv*, abs/1910.03771.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2048–2057, Lille, France. PMLR.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019a. AMR Parsing as Sequence-to-Graph Transduction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Florence, Italy. Association for Computational Linguistics.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019b. Broad-coverage semantic parsing as transduction. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

A System architecture

An illustration of our system architecture is shown in Figure 7.

B PMB - data statistics

	train	dev	test
PMB2.2.0-g	4597 (4585)	682	650
PMB2.2.0-s	67965 (63960)	-	-
PMB3-g	6620 (6618)	885	898
PMB3-s	97598 (94776)	-	-
PMB3-it	2772 (2743)*	515	547
PMB3-de	5250 (5019)*	417	403
PMB-nl	1301 (1238)*	529	483

Table 5: Data statistics for the PMB v.2.2.0 and 3.0(g - gold; s - silver). Numbers in paranthesis are the instances we used during training that we were able to extract a derivation tree for. *: training instances for languages other than English are *silver*, whereas dev and test are *gold*

C DAG-grammar extraction

Our grammar consists of three steps:

Preprocess the DRS. First, we treat all constants as lexical elements and bind them to a variable c. For instance, in Figure 1 we bind 'speaker' to a variable c_1 and change the relations AGENT(e_1, 'speaker') and AGENT(e_2, 'speaker') into AGENT(e_1, c_1) and AGENT(e_2, c_1), respectively. Second, we deal with multiple lexical elements that map to the same variables (e.g. cat(x_1) $\wedge$ entity(x_1), where the second predicate specify the 'nature' of the first) by renaming the second variable as i and creating a dummy relation OF that maps from from the first to the second. Finally, we get rid of relations that generate cycles. We found 25 cycles in the PMB, and they are all related to the same phenomenon where the relationships 'Role' and 'Of' have inverted source and target (e.g. person(x1) - Role - mother(x4), mother(x4) - Of - person(x1)). We remove cyclicity by merging the two relations into one edge label. All these changes are then reverted before evaluation.

Converting the convert the DRS into a DAG. We convert all *main* boxes, lexical predicates and constants (now bound to a variable) to nodes whereas binary relations between predicates and boxes are treated as edges. For each box, we identify a *root* variable (if any) and attach this as child to the box-node with an edge :DRS. A root variable is defined as a variable belonging to a box that is *not* at the receiving end of any binary predicates; in Figure 1, these are e_1 and e_2 for b_2 and b_3 respectively. We then follow the binary relations to expand the graph. In doing so, we also incorporate *presuppositional* boxes in the graph (i.e. b_4 in Figure 1). Each of these boxes contain predicates that are presupposed in context (usually definite descriptions like 'the door'). To link presupposed boxes to the main boxes (i.e. to get a fully connected DAG) we assign a (boolean) presupposition feature to the root variable of the presupposed box (this feature is marked with the superscript ^{p} in Figure 2). Any descendant predicates of this root variable will be considered as part of the presupposed DRS. During post-processing, when we need to reconstruct the DRS out of a DAG, we rebuild the presupposed box around variables for which presupposition is predicted as 'True', and their descendants.

Note that Basile and Bos (2013) proposed a similar conversion to generate Discourse Representation Graphs (DRG), exemplified in Figure 8 using our working example. We argue that our representation is more compact in that: 1) we ignore 'in' edges – where each variable is explicitly marked as part of the box by means of a dedicated edge. This is possible since each box (the square nodes) has a main predicate and all its descendants belong to the box; 2) we treat binary predicates (e.g. AGENT) as edge labels and not nodes; 3) we remove presupposition boxes (in Figure 8, the subgraph rooted in a P labelled edge) and assign a (boolean) presupposition variable to the presupposed predicates.

Convert the DAGs into derivation trees. DAGs are converted into derivation trees in two passes following the algorithm in Björklund et al. (2016), which we summarize here; the reader is referred to the original paper for further details. The algorithm consists of two steps: first, for each node n we traverse the graph post-order and store information on the reentrant nodes in the subgraph rooted n. To be more precise, each outgoing edge e_i from n defines a subgraph s_i along which we extract a list of all the reentrant nodes we encounter. This list also includes the node itself, if reentrant.

We then traverse the tree depth-first to collect the grammar fragments and build the derivation tree. Each node contains information of its variable (and type), lexical predicate and features as well as a list of the labels on outgoing edges that we plug in the fragments. In order to add variable

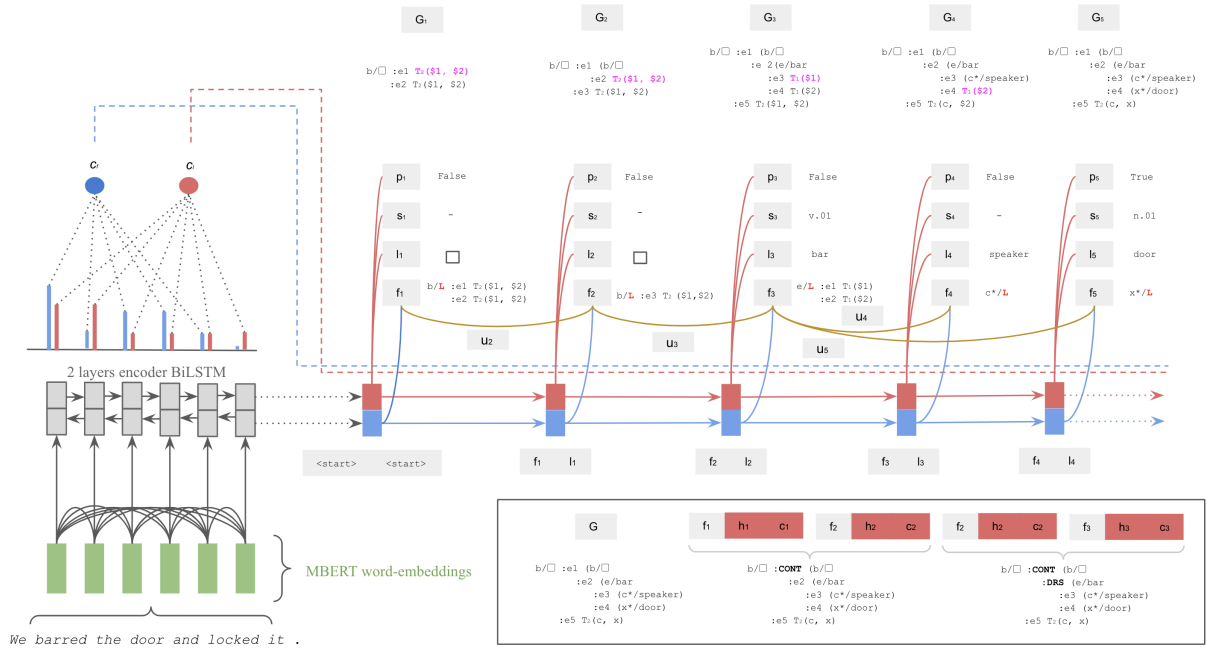


Figure 7: Overview of our architecture, following the description is § 3. Our encoder (on the left) computes multilingual word-embeddings using MBERT which then feed into a 2-layers BiLSTM. At the time step t, a 2 layers decoder LSTM (on the right) reconstructs a graph G by predicting fragment f_t and terminal label l_t. Additionally, parsing on PMB requires to predict for each label l_t a sense tag s_t and presupposition information p_t (a boolean flag). To predict f_t we use the hidden state of the decoder first layer (in blue) along with context vector c_t^f and information about the parent fragment u_t (yellow edges). All other predictions are done using the hidden state of the decoder second layer (in red) along a separate context vector c_t^l. Both context vectors are computed using soft attention over the input representations (top left). Fragments predicted are used to substitute the leftmost non-terminal in the partial graph G (in pink), as shown at the top for $G_2...G_5$. For G_1 the first fragment predicted initializes the graph (this corresponds to substituting the start symbol S). The edge labels in the fragments above are replaced with placeholders $e_1...e_{|e|}$ to display how edge factorization works. We assume here, for brevity, that G_5 is our final output graph and show the prediction of two edges that substitute in place of the placeholders (box at the bottom). For edge prediction, we use a bundle of features collected during decoding, namely the parent and children fragment embedding f_t, the second layer hidden state (in red) and the context vector c^l at time t.

	en			de			nl			it		
	P	R	F	P	R	F	P	R	F	P	R	F
monolingual	81.6	78.4	80	64.5	64	64.2	62.6	59.2	60.9	72.4	70.6	71.5
cross-lingual	-	-	-	72.8	73.6	73.2	73.4	74.9	74.1	74.2	76.2	75.2
bilingual	-	-	-	72	71.5	71.8	76.7	75.3	76	76.8	78.6	77.7
polyglot	81	78.8	79.8	72.2	72.9	72.5	74.3	73.8	74.1	78.2	77.5	77.9

Table 6: Results for the multilingual experiments on PMB v.3.0 (test set). Monolingual results (top half) are compared with different combinations of multilingual training data (bottom half).

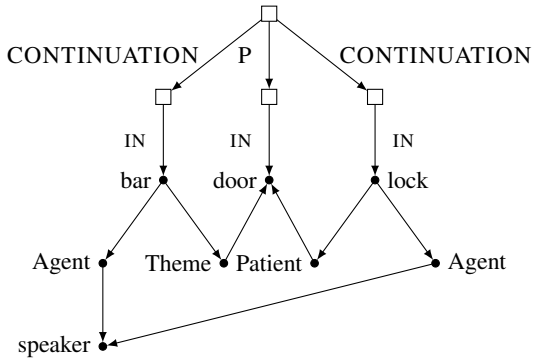


Figure 8: The DRS of Figure 2 expressed as a Discourse Representation Graph (DRG).

references, if any, we need to know whether there are any reentrant nodes that are shared across the subgraphs $s_i ... s_{|e|}$. If so, these become variable references. If the node n itself is reentrant, we flag it with * so that we know that its variable name can substitute a variable reference.

D Implementation Details

We use the pre-trained uncased multilingual BERT base model from Wolf et al. (2019). All models trained on English data, monolingual or multilingual, share the same hyper-parameter settings. Languages other than English in the PMB v3.0 have less training data, especially in the cases of Dutch and Italian. Hence, we reduce the model capacity across the board and increase dropout to avoid over-fitting. Hyperparameter settings are shown in Table. 7.

We found fine-tuning BERT model necessary to achieve good performance. Following Sun et al. (2019) and Howard and Ruder (2018), we experimented with different fine-tuning strategies, all applied after model performance plateaued:

1. setting constant learning rate for BERT layers
2. gradually unfreezing BERT layer by layer with decaying learning rate
3. *slanted-triangular learning rate scheduling* following Howard and Ruder (2018).

We have concluded that strategy 1 works best for our task, with fine-tuning learning rate of 2e-5 for English and a smaller learning rate of 1e-5 for other languages.

Model Parameters		
BERT		768
Num of Encoder Layer		2
Encoder	en	2@512
	de/nl/it	1@512
Fragment/Relation/Label	en	100
	de/nl/it	75
Edge Prediction Layer	en	100
	de/nl/it	75
Decoder	en	1024
	de/nl/it	512
Optimization Parameters		
Optimizer		ADAM
Learning Rate		0.001
Weight Decay		1e-4
Gradient Clipping		5
Label Smoothing ϵ		0.1
Bert Finetune LR	en	2e-5
	de/nl/it	1e-5
Dropout	en	0.33
	de/nl/it	0.5

Table 7: Hyper-parameter Settings

E Multilingual experiments - full results

All results for the multilingual experiments including precision and recall are shown in Table 6.