Transition-based DRS Parsing Using Stack-LSTMs

Kilian Evang
University of Düsseldorf
evang@hhu.de

Abstract

We present our submission to the IWCS 2019 shared task on semantic parsing, a transition-based parser that uses explicit word-meaning pairings, but no explicit representation of syntax. Parsing decisions are made based on vector representations of parser states, encoded via stack-LSTMs (Ballesteros et al., 2017), as well as some heuristic rules. Our system reaches 70.88% f-score in the competition.

1 Introduction

Anyone who complains about arguing over "semantics" has never seen how boring it can be to argue over syntax. —isaacs (@izs). 2013-07-05. Tweet.

A spectrum is haunting semantic parsing—the spectrum ranging from traditional semantic grammars on one end to recent sequence-to-sequence methods on the other. Examples of the former include the LKB system for Minimal Recursion Semantics (Copestake, 2002), and Boxer (Bos, 2008) for Discourse Representation Theory (DRT). Examples of the latter include van Noord and Bos (2017) for Abstract Meaning Representations (AMR) and Liu et al. (2018); van Noord et al. (2018) for DRT. The approach in the present paper aims to occupy a useful middle ground on this spectrum. On the one hand, we emphasize the usefulness of an explicitly specified lexicon of word-meaning pairs, amenable to tweaking by linguists and engineers, and to interfacing with rule-based components. On the other hand, we aim to minimize the amount of grammar engineering required, and rely on neural networks to learn to assemble word meanings into sentence meanings.

We describe a system that follows this approach and apply it to the IWCS 2019 shared task on DRS parsing (Abzianidze et al., 2019). The challenge is to map raw input sentences (plain text, not tokenized or otherwise annotated) to discourse representation structures (DRSs). DRSs represent meaning as a hierarchy of nested boxes containing *referents* and *conditions*. They can be represented as a flat set of *clauses*, where referent identity and special conditions encode the structure. For example, in Figure 1 (top right), all clauses belonging in box **`b2`** are marked with the **`b2`** prefix, and that the referent **`e1`** is introduced by box **`b2`** is expressed by the special condition **`b2`** `REF` **`e1`**.

Our system is inspired by the AMR parser of Ballesteros and Al-Onaizan (2017) and, by extension, the non-projective dependency parsing algorithm of Nivre (2009): it uses a transition sytem to process tokens from left to right, and stack-LSTMs to create vector representations of parser states to make transition decisions. To apply this approach to DRT, we replace atomic node labels by *lexical clause lists* (LCLs) and edge labels by sets of *referent address pairs* (RAPs), which encode decisions to unify specific discourse referents. We also factor the lexicon to address data sparseness and apply various preprocessing and postprocessing steps to ease learning.

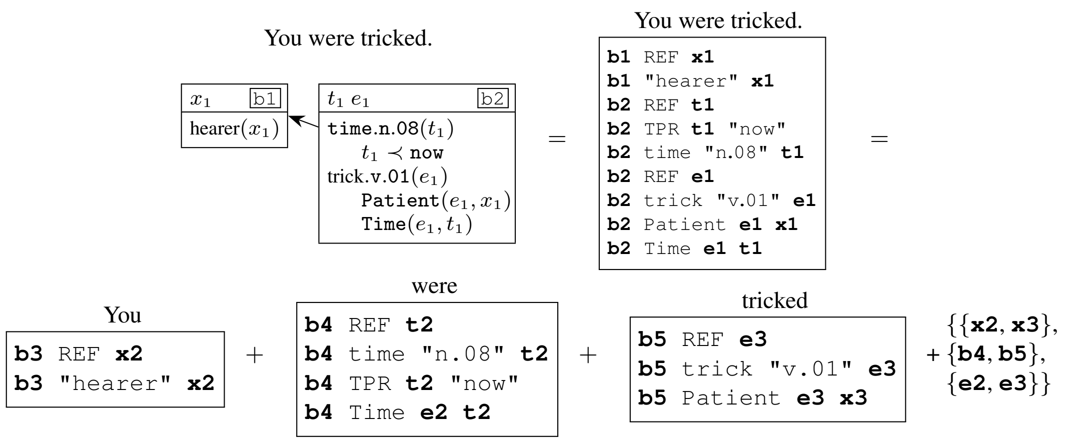

Figure 1: An example DRS in box notation (top left), clause notation (top right), and decomposed into three lexical clause lists (LCLs) and a binding set (bottom row).

Table 1: The most frequent bind actions generated from the shared task gold training data.

rank	action	count	rank	action	count
1	bind({($\mathbf{x}_1$, $\mathbf{x}_1$)})	6 969	6	bind({($\mathbf{b}_1$, $\mathbf{b}_1$),($\mathbf{x}_2$, $\mathbf{x}_1$)})	1 126
2	bind({($\mathbf{b}_1$, $\mathbf{b}_1$)})	3 435	7	bind({($\mathbf{b}_2$, $\mathbf{b}_1$)})	865
3	bind({($\mathbf{x}_2$, $\mathbf{x}_1$)})	2 526	8	bind({($\mathbf{b}_2$, $\mathbf{b}_1$),($\mathbf{s}_1$, $\mathbf{s}_1$)})	644
4	bind({($\mathbf{b}_1$, $\mathbf{b}_1$),($\mathbf{e}_1$, $\mathbf{e}_1$))}	2 187	9	bind({($\mathbf{x}_1$, $\mathbf{x}_2$)})	629
5	bind({($\mathbf{b}_1$, $\mathbf{b}_1$),($\mathbf{x}_1$, $\mathbf{x}_1$))}	1 730	10	bind({($\mathbf{e}_1$, $\mathbf{e}_2$)})	380

2 Parsing Algorithm

Words. They mean things. —The Linguist Llama

For training, we assume tokenized sentences, each paired with a DRS in the form of a clause list, each clause aligned to 0, 1, or more tokens. We decompose this clause list into one *lexical clause list* (LCL) per token, plus a *binding set* $\mathcal{B}$, as shown in the bottom row of Figure 1. Each LCL contains only the clauses aligned to the corresponding token, and referents are replaced by fresh ones unique to that LCL. $\mathcal{B}$ contains all unordered pairs of referents that replaced the same original referent. We say that a referent has an *address* T_n in an LCL if it is the n-th referent of type T to occur in the LCL. For example, **e2** has address $\mathbf{e}_1$ in the LCL for *were*, and **e3** has the same address in the LCL for *tricked*. We write $\mathit{ref}(L, T_n)$ for the referent that has address T_n in L.

The parser uses three data structures: a *stack*, initially empty, a *buffer*, initially containing all tokens of the sentence, and a *result clause list*, initially empty. Until both stack and buffer are empty, the parser repeatedly chooses an *action* that manipulates the contents of the data structures. The correct action sequence for our example is shown in Figure 2. For training, we determine the correct action sequence (also called the *oracle*) as follows: if the rightmost stack element is a token, choose confirm and replace the token with the corresponding LCL. Otherwise, if the rightmost stack element does not contain any referent that still occurs in $\mathcal{B}$, choose reduce, add its clauses to the result clause list, and remove it from the stack. Otherwise, if there are at least two elements on the stack, consider the two rightmost ones; let them be called L and R. Compute the set B of RAPs (referent address pairs) (T_l, T_r) so that $\{\mathit{ref}(L, T_l), \mathit{ref}(R, T_r)\} \in \mathcal{B}$. If B is nonempty, choose bind(B), unify the corresponding referents, and remove referent sets that are now singleton from $\mathcal{B}$. If B is empty and L and R are still in their original order, choose swap and move L to the left end of the buffer. Otherwise, choose shift and move the leftmost buffer element to the rightmost position on the stack.

RAP sets can be seen as an automatically induced approximation to *arguments* in semantic grammars, in that they define the interface between two lexical meaning representations. Table 1 shows the most

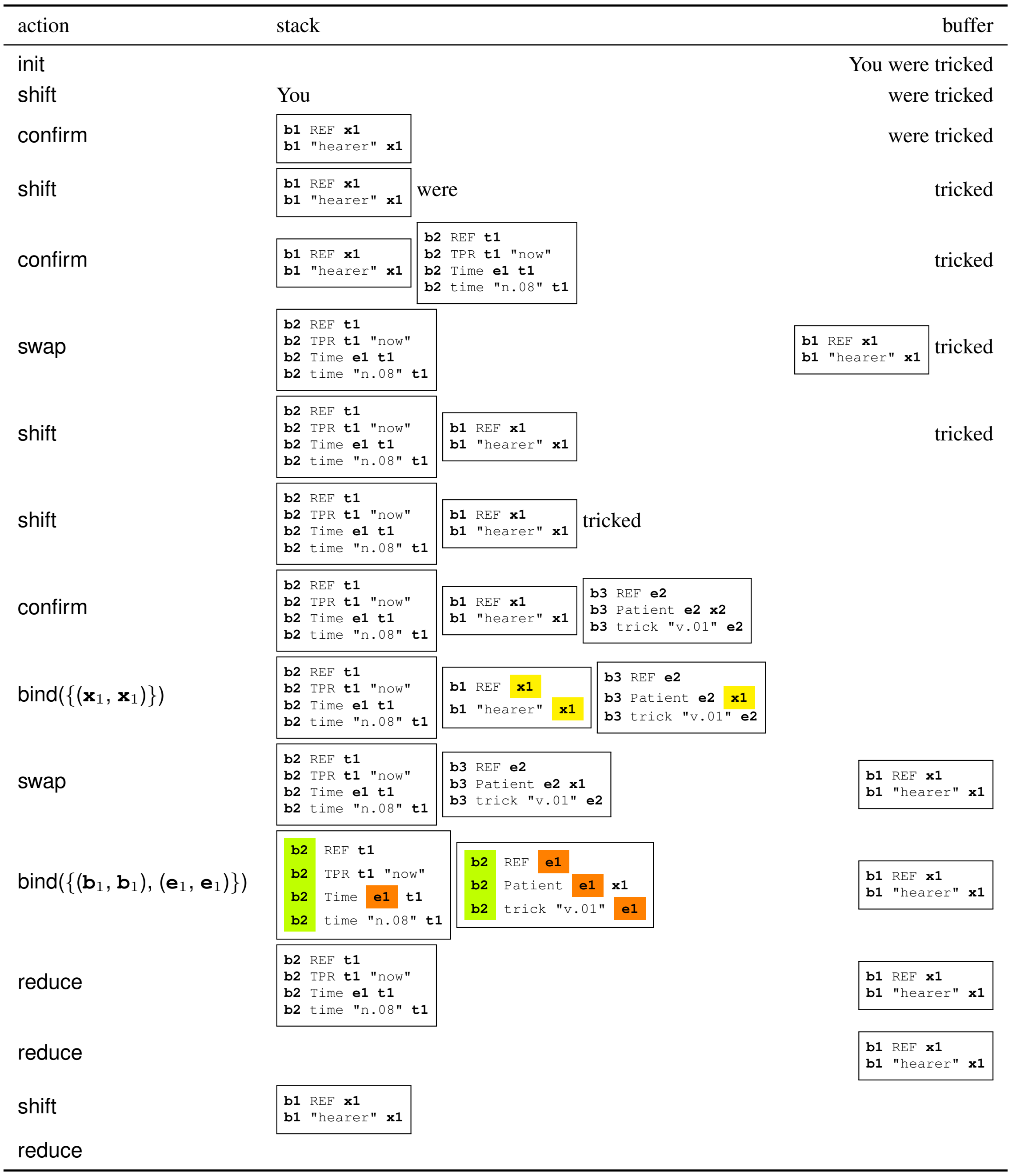
action
stack
buffer
init
You were tricked
shift
You
were tricked
confirm
b1 REF x1 b1 "hearer" x1
were tricked
shift
b1 REF x1 b1 "hearer" x1
were
tricked
confirm
b1 REF x1 b1 "hearer" x1
b2 REF t1 b2 TPR t1 "now" b2 Time e1 t1 b2 time "n.08" t1
tricked
swap
b2 REF t1 b2 TPR t1 "now" b2 Time e1 t1 b2 time "n.08" t1
b1 REF x1 b1 "hearer" x1
tricked
shift
b2 REF t1 b2 TPR t1 "now" b2 Time e1 t1 b2 time "n.08" t1
b1 REF x1 b1 "hearer" x1
tricked
shift
b2 REF t1 b2 TPR t1 "now" b2 Time e1 t1 b2 time "n.08" t1
b1 REF x1 b1 "hearer" x1
tricked
confirm
b2 REF t1 b2 TPR t1 "now" b2 Time e1 t1 b2 time "n.08" t1
b1 REF x1 b1 "hearer" x1
b3 REF e2 b3 Patient e2 x2 b3 trick "v.01" e2
bind({(x1, x1)})
b2 REF t1 b2 TPR t1 "now" b2 Time e1 t1 b2 time "n.08" t1
b1 REF x1 b1 "hearer" x1
b3 REF e2 b3 Patient e2 x1 b3 trick "v.01" e2
swap
b2 REF t1 b2 TPR t1 "now" b2 Time e1 t1 b2 time "n.08" t1
b3 REF e2 b3 Patient e2 x1 b3 trick "v.01" e2
b1 REF x1 b1 "hearer" x1
bind({(b1, b1), (e1, e1)})
b2 REF t1 b2 TPR t1 "now" b2 Time e1 t1 b2 time "n.08" t1
b2 REF e1 b2 Patient e1 x1 b2 trick "v.01" e1
b1 REF x1 b1 "hearer" x1
reduce
b2 REF t1 b2 TPR t1 "now" b2 Time e1 t1 b2 time "n.08" t1
b1 REF x1 b1 "hearer" x1
reduce
b1 REF x1 b1 "hearer" x1
shift
b1 REF x1 b1 "hearer" x1
reduce

Figure 2: Actions for parsing the sentence “You were tricked.” Referent addresses (e.g., $\mathbf{b}_1$) should not be confused with referent names (e.g., **b1**).

```
        tricked
b1 REF e1                   b1 REF e1
b1 trick "v.01" e1    =     b1 work "v.00" e1     * [Patient] * trick "v.01"
b1 Patient e1 x1            b1 Role e1 x1
b1 Time e1 t1               b1 Time e1 t1
```

Figure 3: Factoring a lexical clause list (LCL) into an underspecified lexical clause list (ULCL), a rolelist, and a sense. We use `work "{n,v,a}.00"` as dummy senses.

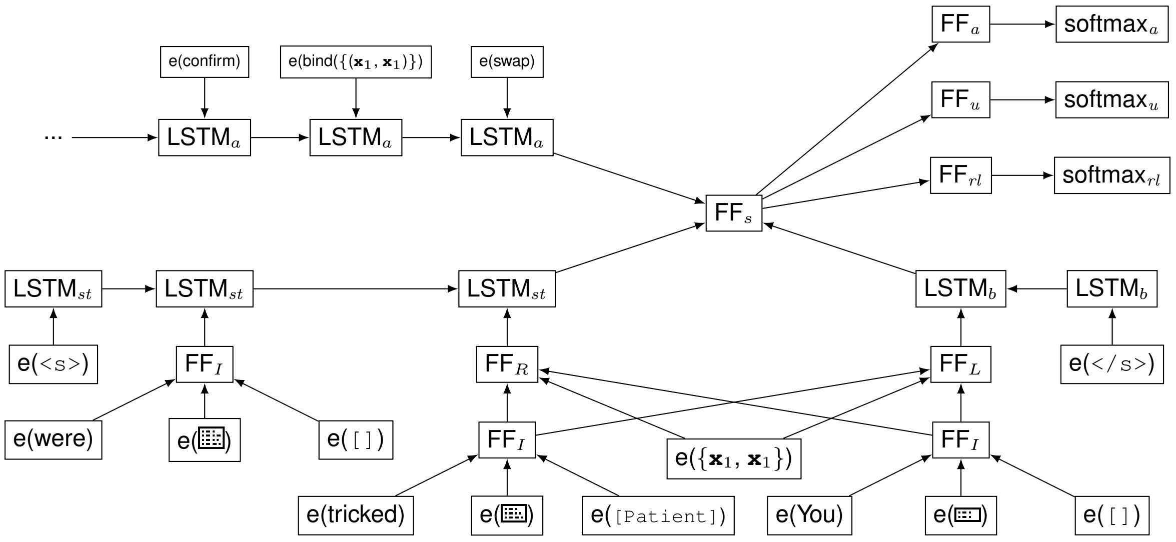

Figure 4: Configuration of the neural network after the second swap action in Figure 2. e(...) are embeddings of words, ULCLs, rolelists, RAP sets, and actions. FF_I is the interpretation function, FF_L, FF_R are the composition functions. Boxes labeled LSTM represent individual LSTM cells of the stack/buffer/action stack-LSTMs. Boxes labeled FF denote feed-forward neural networks with one hidden layer; FF_s has a RelU nonlinearity and the others have a tanh nonlinearity.

frequent bind actions. In total, 127 were induced from the training data. They may be too sparse. Our RAP set generation is not sensitive to referent names since referents are addressed by order, not name. However, it is sensitive to clause order and referent type. As a reviewer pointed out, van Noord et al. (2018) showed that normalizing clause order while conflating the referent types **x**, **t**, and **e** improved performance in their parser. We plan to investigate this in future work.

3 Parsing Model

At test time, we have neither gold-standard LCLs nor a binding set, yet we want to end up with a result clause list that is the same as the gold standard DRS, or at least similar. We thus train a statistical model to choose the right action at each parser state, and to choose the right lexical clause list for each token. The model has three softmax classifiers, shown in the top right corner of Figure 4: the *action classifier*, the *ULCL classifier*, and the *rolelist classifer*. At each state, the action classifier chooses one out of 131 actions which were extracted from the shared task gold training data, 127 of which are bind actions with various RAP sets. The classifier only chooses among the actions which are applicable to the respective state, for example, shift requires a nonempty buffer, and bind actions require every addressed referent to exist. After each confirm action, the model chooses an LCL for the token on the stack to be replaced with.

To better handle the large variety of LCLs, we factor this into three steps, as illustrated in Figure 3: first, the *ULCL classifier* chooses one out of 548 different ULCLs (underspecified LCLs with dummy event roles, dummy senses, and dummy constants). The *rolelist classifier* then chooses from 146 lists of event roles to replace the dummy event roles with. Finally, heuristic rule-based components ("symbolizers", see below) fill in the senses and constants.

The set of ULCLs is automatically created offline from all LCLs in the training data by normalizing referent names and replacing event roles, senses, and constants with dummy values. Non-event roles are currently left intact, as are a number of very common senses and constants (`male "n.02"`, `female "n.02"`, `time "n.08"`, `person "n.01"`, `measure "n.02"`, `entity "n.01"`, `country "n.02"`, `city "n.01"`, `quantity "n.01"`, `name "n.01"`, `location "n.01"`, `"now"`, `"speaker"`, `"hearer"`).

The input to the three classifiers is a vector representation of the parser state, computed using *stack-LSTM* representations of the stack, the buffer, and the list of previous actions. Stack-LSTMs (Ballesteros et al., 2017; Ballesteros and Al-Onaizan, 2017) are LSTMs (Hochreiter and Schmidhuber, 1997) whose sequence of input vectors can change dynamically. Over the course of a parsing process, the parser grows and shrinks these input sequences as elements are added to and removed from the associated data structures. Initially, the buffer LSTM (LSTM_b) has the word embeddings of the entire input sequence. These are gradually moved to the stack LSTM (LSTM_{st}) by shift actions, where they are further transformed: when confirm occurs, the righmost hidden state of the stack LSTM is transformed by an *interpretation function* and its output then replaces the rightmost input to the stack-LSTM. When bind occurs, the two rightmost hidden states of LSTM_{st} are transformed by two separate *composition functions* and their outputs replace the two rightmost inputs to the LSTM_{st}. Inputs to LSTM_{st} can also become inputs to LSTM_b again through swap actions. Figure 4 shows one snapshot of the dynamically changing network, after the second swap action in our example.

4 Implementation

Our system is implemented in Python using DyNet (Neubig et al., 2017). We use ELMo (Peters et al., 2018) for pre-trained word embeddings. At test time, we tokenize sentences using Elephant (Evang et al., 2013) trained on a pre-release version of the Parallel Meaning Bank (Abzianidze et al., 2017).

Hyperparameters Time did not allow for extensive tuning. Where applicable, we followed the choices of Ballesteros et al. (2017). For details, see Table 2 and the source code (`https://bitbucket.org/kevang/drs_parsing`).

Preprocessing and Postprocessing In the training data, the constants `"speaker"` and `"hearer"` typically appear in clauses aligned to verbs rather than first and second person pronouns. To prevent a proliferation of verb ULCLs, our system changes this representation to the one shown in Figure 1 for training and applies an inverse transformation to its output at test time. It also creates a "main box" (a DRS containing all other DRSs) in postprocessing if none exists yet.

Symbolizers We implemented rule-based components that replace dummy constants with proper constants for names (e.g., `"mary"` for the token `Mary`), quantities (e.g., `"1000"` for the token `"one~thousand"`), and time expressions (e.g., `"05:00"` for the token `five~o'clock`). For

Table 2: Hyperparameter settings.

hyperparameter	**value**
updated word embeddings	
unknown word probability	0.2
dimensions	40
pretrained word embeddings (ELMo)	
dimensions (average of 3 layers)	1024
other embedding dimensions	
actions	20
ULCLs	20
RAP sets	20
rolesets	20
stack-LSTMs	
input dimensions	100
hidden layers	2
hidden dimensions (per layer)	100
learning rate (simple SGD)	
initial value	0.1
decay per epoch	0.08

dummy word senses, we fill in the lemma using NLTK's WordNet lemmatizer (Bird et al., 2009) and assume sense number `01`.

Training We train with 1 batch = 1 training example, using the negative sum of the log probabilities of all correct classification decisions as loss. We train on the gold training data for 20 epochs and validate after each epoch on the gold development data using Counter (van Noord et al., 2018). We use the model with the highest validation f-score.

5 Competition Results and Discussion

For the competition, we used the best model unchanged, i.e., we did not retrain with the dev/test data included. At this point, our system had a bug where the interpretation function FF_I only took ULCL and rolelist embeddings as input, not the word embedding. It was also still lacking the quantity symbolizer. It reached 74.34% precision, 73.32% recall, and 73.83% f-score on the development data, 74.60% precision, 74.14% recall, and 74.37% on the test data, and 71.81%, 69,92% recall, and 70.88% f-score in the competition. The organizers provided five sentences for which our system's output was lowest (highest) compared to the minimum (maximum) of the other participating systems, along with all outputs. We inspected sentences and tried to identify the main reasons our system performed worse (better) than others on these examples. They are by no means guaranteed to be representative, but may serve as starting points for discussion and further investigation.

Reasons for Failure (a) The system "skipped" some tensed matrix verbs, i.e., it assigned them the empty ULCL, as it does for punctuation (sentences 522, 271, 385). This may point to failure to generalize or sparse data. (b) The system introduced many DRSs but failed to connect them by binding referents, so it defaulted to connect them with CONTINUATION discourse relations in post-processing (452, 414).

Reasons for Success (c) The system profited from the decision to leave special senses intact, which enabled it to correctly analyze relational nouns (309). (d) The system was not completely thrown off by archaic language, possibly helped by the large body of text the ELMo embeddings are trained on (163). (e) A rare adjective seemed to trip up character-based systems, but was handled correctly by our WordNet-based symbolizer (147). (f) Our system's first-sense heuristic got lucky (454). (g) Our system got lucky and agreed with an apparent error in the gold standard (138).

We further observe that our system does very poorly on some sentences that lack sentence-final punctuation, which points to hypersensitivity to diversions that is typical of current neural models (cf., e.g., Søgaard et al., 2018). Our current oracle generation algorithm treats anaphora like other long-distance dependencies, which we surmise is suboptimal. Finally, the shared task data has quite an aggressive approach to merging multi-token units into a single token, which is not handled optimally by the tokenizer we used. Beyond these specific avenues for future improvement, generic ones are applicable: architecture optimization, hyperparameter optimization, ensembles, additional features from taggers and dependency parsers, training on silver data, etc. Some of these have been shown to have a large impact on similar tasks (Ballesteros and Al-Onaizan, 2017; van Noord et al., 2018).

6 Ablations

After the competition, we improved the system by fixing the bug in the interpretation function and adding the quantity symbolizer. We then ran an ablation study to assess the contribution of some individual components. The results are shown in Table 3. Contrary to our expectations, factoring rolelists out of ULCLs does not seem to improve results, although it helps the system reach its peak performance after

	precision	recall	f-score	epochs
full system	.7562	.7460	.7511	15
- factoring rolelists out of ULCLs	.7545	.7503	.7524	18
- realigning pronouns	.7545	.7444	.7494	20
- date/time symbolizer	.7535	.7434	.7484	15
- quantity symbolizer	.7495	.7395	.7445	15

Table 3: Ablation results on the development data, with one component removed at a time. "Epochs" indicates the number of training epochs needed to reach the indicated f-score.

fewer epochs. Realigning pronouns helps a bit. The date/time symbolizer and the quantity symbolizer are clearly beneficial.

7 Conclusions

Traditional semantic grammars are transparent, but theory-heavy and costly to adapt to new languages and domains. End-to-end systems are easy to use and performant, but opaque: if there are errors, it is hard to pinpoint the causes and fix them. Thus, either approach has problems that may make it infeasible in production, semi-automatic annotation, or education settings. We believe that our approach—lexicalist but with no need for an explicit representation of syntax—strikes an elegant balance between the two extremes. At the time of this writing, we do not know where our system ranks among the shared task participants. Previous work on similar tasks (Liu et al., 2018; van Noord et al., 2018) has reached f-scores of up to 77.5% resp. 83.6%, however, these results were obtained on different and potentially less complex test sets. And as discussed above, there are many promising avenues to further increasing the performance of our system. Thus, whatever the outcome of this competition, we believe that our approach is worth pursuing further.

Acknowledgments

The author would like to thank the three anonymous reviewers for valuable feedback. This work was carried out within the TreeGraSP project, funded by a Consolidator Grant of the European Research Council.

References

Abzianidze, L., J. Bjerva, K. Evang, H. Haagsma, R. van Noord, P. Ludmann, D.-D. Nguyen, and J. 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: Volume 2, Short Papers*, pp. 242–247. Association for Computational Linguistics.

Abzianidze, L., R. van Noord, H. Haagsma, and J. Bos (2019). The first shared task on discourse representation structure parsing. In *Proceedings of the IWCS 2019 Shared Task on Semantic Parsing*.

Ballesteros, M. and Y. Al-Onaizan (2017). AMR parsing using stack-LSTMs. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 1269–1275. Association for Computational Linguistics.

Ballesteros, M., C. Dyer, Y. Goldberg, and N. A. Smith (2017). Greedy transition-based dependency parsing with stack lstms. *Computational Linguistics, Volume 43, Issue 2 - June 2017*, 311–347.

Bird, S., E. Loper, and E. Klein (2009). *Natural Language Processing with Python*. O'Reilly Media Inc.

Bos, J. (2008). Wide-coverage semantic analysis with Boxer. In *Semantics in Text Processing. STEP 2008 Conference Proceedings*.

Copestake, A. (2002). *Implementing typed feature structure grammars*, Volume 110. Stanford: CSLI.

Evang, K., V. Basile, G. Chrupała, and J. Bos (2013). Elephant: Sequence labeling for word and sentence segmentation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1422–1426. Association for Computational Linguistics.

Hochreiter, S. and J. Schmidhuber (1997, November). Long short-term memory. *Neural Comput. 9*(8), 1735–1780.

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

Neubig, G., C. Dyer, Y. Goldberg, A. Matthews, W. Ammar, A. Anastasopoulos, M. Ballesteros, D. Chiang, D. Clothiaux, T. Cohn, K. Duh, M. Faruqui, C. Gan, D. Garrette, Y. Ji, L. Kong, A. Kuncoro, G. Kumar, C. Malaviya, P. Michel, Y. Oda, M. Richardson, N. Saphra, S. Swayamdipta, and P. Yin (2017). DyNet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*.

Nivre, J. (2009). Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pp. 351–359. Association for Computational Linguistics.

Peters, M. E., M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer (2018). Deep contextualized word representations. In *Proc. of NAACL*.

Søgaard, A., M. de Lhoneux, and I. Augenstein (2018, November). Nightmare at test time: How punctuation prevents parsers from generalizing. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, Brussels, Belgium, pp. 25–29. Association for Computational Linguistics.

van Noord, R., L. Abzianidze, H. Haagsma, and J. Bos (2018). Evaluating scoped meaning representations. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan.

van Noord, R., L. Abzianidze, A. Toral, and J. Bos (2018). Exploring neural methods for parsing discourse representation structures. *Transactions of the Association for Computational Linguistics 6*, 619–633.

van Noord, R. and J. Bos (2017). Neural semantic parsing by character-based translation: Experiments with abstract meaning representations. *Computational Linguistics in the Netherlands Journal 7*, 93–108.