BME-UW at SR'19: Surface realization with Interpreted Regular Tree Grammars

Ádám Kovács[1,2], **Evelin Ács**[1], **Judit Ács**[1,2,3], **András Kornai**[2], **Gábor Recski**[1,4]
[1]Dept. of Automation and Applied Informatics, Budapest U of Technology
`lastname.firstname@aut.bme.hu`

[2]SZTAKI Insititute of Computer Science
`andras@kornai.com`

[3]University of Washington

[4]Sclable

Abstract

The Surface Realization Shared Task involves mapping Universal Dependency graphs to raw text, i.e. restoring word order and inflection from a graph of typed, directed dependencies between lemmas. Interpreted Regular Tree Grammars (IRTGs) encode the correspondence between generations in multiple algebras, and have previously been used for semantic parsing from raw text. Our system induces an IRTG for simultaneously building pairs of surface forms and UD graphs in the SR'19 training data, then prunes this grammar for each UD graph in the test data for efficient parsing and generation of the surface ordering of lemmas. For the inflection step we use a standard sequence-to-sequence model with a biLSTM encoder and an LSTM decoder with attention. Both components of our system are available on GitHub under an MIT license.

1 Introduction

The 'shallow' (T1) track of the Surface Realization task (Mille et al., 2019) involves mapping Universal Dependencies (UD) graphs (De Marneffe et al., 2014) to surface forms, i.e. restoring word order and inflection based on the typed grammatical dependencies among a set of lemmas. We used a hybrid method that restores word order by IRTG rules, see Section 2, induced from the training data, see Section 3, and performs inflection using a standard sequence-to-sequence model with a biLSTM encoder and an LSTM decoder with attention, see Section 4. This architecture fits well with the recent trend toward eXplainable AI (Gunning, 2017), and is not as data-hungry as end-to-end neural systems. Only 8 of the 12 teams participated on the non-English portion of the track, with BME-UW ranked second in automated, and generally in the top three in human evaluation. The IRTG based system is available under `https://github.com/adaamko/surface_realization`, the inflection system was trained using the framework under `https://github.com/juditacs/deep-morphology`

2 Rule format: IRTGs and s-graphs

Several common tasks in natural language processing involve graph transformations, in particular those that handle syntactic trees, dependency structures such as UD, or semantic graphs such as AMR (Banarescu et al., 2013) and 4lang (Kornai et al., 2015). Interpreted Regular Tree Grammars (IRTGs) (Koller, 2015) encode the correspondence between sets of such structures and have in recent years been used to perform syntactic parsing (Koller and Kuhlmann, 2012), generation (Koller and Engonopoulos, 2017), and semantic parsing (Groschwitz et al., 2015, 2018). In previous work (Ács et al., 2019) we encoded transformations between raw text, phrase structure (PS) trees, UD and 4lang semantic graphs to build a single IRTG that allows for mapping between any pair of such structures.

IRTGs are Regular Tree Grammars in which each rule is mapped to operations in an arbitrary number of algebras. Hence, derivations of an IRTG correspond to synchronous generation of objects in each of these algebras, and an IRTG parser such as alto (Gontrum et al., 2017) can be used to map from one set of objects to all others. For the word order restoration task our system constructs an IRTG operating on strings and UD graphs, simultaneously constructing sentences from words and UD graphs from nodes. Operations of the simple string algebra $(S, \cdot)$ are mapped to those of an S-graph algebra (Courcelle, 1993), a formalism also used by (Groschwitz et al., 2015) to perform semantic parsing via IRTGs. Here we give only an informal overview of s-graph algebras, see (Koller and Kuhlmann, 2011; Courcelle and Engelfriet, 2012) for a more formal explanation. S-graphs are graphs whose vertices may be labeled by one of a countable set of *sources*. The central operation of an s-graph algebra is the binary merge, which unifies pairs of s-graphs in a way that vertices with matching sources are merged, i.e. when two s-graphs G_1 and G_2 are merged, the resulting s-graph G' will contain all nodes of G_1 and G_2, and when a pair of nodes $(v_1, v_2) \in E(G_1) \times E(G_2)$ have the same source name, they will be mapped to a single node v' in G' that has all adjacent edges of v_1 and v_2. Sources can also be *renamed* or *forgotten* using the operations $ren_{a \leftrightarrow b}$ and fg_a, where a and b are sources from the set $\mathcal{A}$. Next we shall provide a small example with string and s-graph interpretations.

The Algebraic Language Toolkit, or alto[1] (Gontrum et al., 2017), is an open-source parser for IRTGs that implements a variety of algebras to use as intepretations of IRTGs, including the string algebra and s-graph algebra. An alto grammar file must declare all interpretation algebras and for each RTG rule provide mappings to operations in each of these algebras. Figure 1 shows a minimal example of an IRTG with two interpretations. The abstract RTG rule _nsubj, so named after the corresponding UD relation, has two abstract arguments, designated VERB and NOUN. The string interpretation establishes that the surface form of the second argument (NOUN) is to precede the first argument (VERB). The ud interpretation adds a directed nsubj edge between the subgraphs corresponding to each argument, by a series of rename, merge, and forget operations. Angle brackets after nodes indicate source names. In our s-graph grammars, every subgraph at every point of the derivation

[1] https://github.com/coli-saar/alto

```
interpretation string:
de.up.ling.irtg.algebra.StringAlgebra

interpretation ud:
de.up.ling.irtg.algebra.GraphAlgebra

VERB -> _nsubj(VERB, NOUN)
[string] *(?2, ?1)
[ud] f_dep1(merge(
   merge(?1, "(r<root> :nsubj d1<dep1>)
       ↪ "),
   r_dep1(?2)))

PROPN -> John
[string] John
[ud] "(John<root> / John)"

VERB -> sleeps
[string] sleeps
[ud] "(sleeps<root> / sleeps)"
```

Figure 1: Toy IRTG grammar

has exactly one node labeled with the <root> source, indicating the head of the phrase, which could be connected to a ROOT node to create a well-formed UD-graph. The intepretation in our example contains a graph literal, describing the graph r<root> $\xrightarrow{nsubj}$ d1<dep1>. This graph is first merged with the graph corresponding to the first argument, then the result is merged with the graph obtained by renaming the root source of the second argument's graph to dep. r_dep and f_dep are Alto's shorthands for renaming the root source to dep and forgetting the dep source. Terminal rules create string and UD literals. This toy grammar is therefore a representation of the parallel derivations of the sentence *John sleeps* and the UD graph sleeps $\xrightarrow{nsubj}$ John. The next section will describe our method for building such grammars automatically from UD datasets and using them for the word order restoration step of the Surface Realization task.

3 Rule induction

As seen already in the example IRTG in the previous section, we represent the correspondence between strings and UD graphs as synchronized generations in two algebras. Since our goal is to learn rules of such a grammar using UD datasets containing sentences and corresponding UD graphs, we need a method to assign derivations to UD graphs in the s-graph algebra, i.e. a series of steps that build the UD graph from its nodes, through subgraphs. We choose to represent the construc-

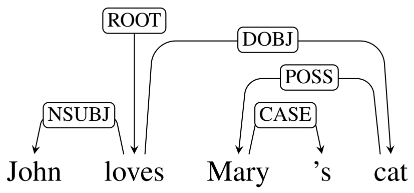

Figure 2: Sample UD analysis

```
S! -> start_VERB(VERB)
PROPN -> rule_1(PROPN,PART)
NOUN -> rule_2(NOUN,PROPN)
VERB -> rule_3(VERB,PROPN,NOUN)
```

Figure 3: sample RTG rules (interpretations omitted)

tion of UD graphs as follows: for each node in the graph we establish one generation step, which is responsible for attaching all its dependents to it. The UD graph depicted in Figure 2 would hence correspond to the RTG rules in Figure 3 (interpretations are omitted for better readability). Note that we create rules that operate at the part-of-speech level, lemmas can then be inserted by terminal rules generated separately for each sentence.

The simplest approach to constructing a (weighted) IRTG would be to simply include all rules "observed" in the training data, along with a probablity calculated from the relative frequency of a given configuration among all occurences of a head of a particular POS-tag. In practice we prune this grammar to include only those rules that are applicable to a given sentence and that are compatible with the value of the `lin` feature (see below), and parse each UD graph using a much smaller grammar. We may also add new rules to the pruned grammar to ensure a successful parsing process (that may or may not yield the correct results).

After generating a static list of IRTG rules from the training data, we dynamically generate a reduced IRTG grammar for each sentence. In a preprocessing step we read all UD graphs that are to be parsed, and for each node and its set of dependents we check if there's a rule in our grammar covering this subgraph. If there's more than one matching rule, we check if the `lin` feature is present in the input, which allows us to identify the single matching rule. If we identify a unique rule matching the subgraph, we add one to its frequency to increase the rule's probability. In other words, sufficiently specific patterns of the test data are used as additional training data. If no rules matching a subgraph are present in our static grammar, we add binary rules for each dependent, some of which rules may already be present in the grammar, in which case we increase their frequencies. This ensures that the grammar will cover the new subgraph but will prefer to build it from subgraphs we have already seen in the training data. If the `lin` feature is not present in the input, we add two rules per dependent, corresponding to each possible word order.

When parsing individual UD graphs, we prune the grammar by deleting all rules that generate POS tags that are not present in the graph (or generate more instances of a POS tag than the tag's total frequency in the graph). We further delete all rules that contradict any `lin` features present in the input (only the $+/-$ sign of feature values is considered). This step must be skipped if it would mean deleting both of a pair of rules, e.g. because a word has punctuation both before and after it. We can then use this pruned grammar to obtain the most probable parse of the UD-graph and the corresponding string interpretation. The average parsing time of `alto` is around 2 seconds per sentence. In a few cases, sentence length would slow down parsing considerably; for all graphs that would take more than one minute to parse (less than 1.5% of the data) we fall back to a grammar that uses binary rules only, i.e. connects all edges of the graph one-by-one.

We illustrate the kind of decisions the parser must make through a simple example. Consider the sentence in Figure 4. Our system correctly predicted the word order based on the UD graph, the top parse involves attaching all dependencies of the predicate *enjoy* using the two rules in Figure 5 (s-graph interpretations are omitted for readability). The second most probable derivation applies the three rules in Figure 6 and would yield the incorrect surface realization *I enjoyed really reading it.*

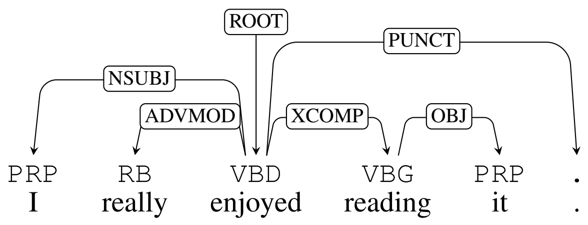

Figure 4: Example from the UD dataset

```
VBD -> rule_22276(VBD,PRP,RB,VBG,PERIOD) [8.14e-06]
[string] *(*(*(?2,?3),*(?1,?4)),?5)

VBG -> rule_615(VBG,PRP) [4.88e-05]
[string] *(?1,?2)
```

Figure 5: Most likely parse of the graph in Fig. 4

```
VBD -> rule_2004(VBD,PRP,VBG,PERIOD) [1.62e-05]
[string] *(*(?2,?1),*(?3,?4))

VBD -> rule_2698(VBD,RB) [1.22e-05]
[string] *(?1,?2)

VBG -> rule_615(VBG,PRP) [4.88e-05]
[string] *(?1,?2)
```

Figure 6: Second most likely parse

These parses illustrate a more general phenomenon: since the probabilities of individual rules are roughly similar, the system prefers derivations with fewer rules, which attach more nodes at the same time. Counterexamples with radically different rule probabilities are in principle possible, but on average the system prefers specific (more detailed) rules over generic (less detailed) ones, which makes the Elsewhere Principle (Kiparsky, 1973) an emergent, rather than an externally enforced, property of the grammar as a whole.

4 Reinflection

In order to map sequences of lemmas to surface forms, we train a standard seq2seq (Sutskever et al., 2014) system with a bidirectional LSTM (Hochreiter and Schmidhuber, 1997) encoder and an LSTM decoder with Luong's attention (Luong et al., 2015). We include all CoNLL-U fields in the input, namely the lemma, the UPOS, the XPOS and the list of morphological tags. We also experimented with adding the position of the token in the sentence (`original_id=N`) during training time. For inference, we use the order generated by the IRTG component. This improves the performance in most, but not all languages (see Table 1). Figure 7 shows an English example of our input and output format.

We split the sentences from the train data into 80% train and 20% development sets for the inflection module. A full-scale hyperparameter search being prohibitively expensive, we only tried a few hyperparameter combinations and use the ones performing best on the dev set for the final submission. Table 1 lists the best configuration

```
Input: <L> f a m i l y </L> <P> UPOS=
    ↪ NOUN XPOS=NNS </P> <T> Number=
    ↪ Plur original_id=2 </T>
Output: f a m i l i e s
```

Figure 7: Example input and output of the inflection component.

and the word accuracy on the dev set by language. We use the Adam optimizer ($\text{lr} = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$) with early stopping based on dev accuracy and loss. Dropout is set to 0.4. Including the position of a token in the sentence (`use position`) is also a hyperparameter.

5 Evaluation

5.1 The Surface Realization Task

We participate in the 'shallow' track of the 2019 Surface Realization Shared Task (SR'19). The task involves mapping UD graphs of lemmas to surface forms in 11 languages. Training data for the task was created from the Universal Dependencies treebanks (Nivre et al., 2018) using methods described in (Mille et al., 2018) and contains UD treebanks with word forms replaced by lemmas word order information removed via scrambling. Two additional features have been added to the dataset, the `lin` feature encoding the relative order of a word and its governor and the `originalId` for reconstructing word order (in the training data only).

5.2 Results

The primary method of evaluation at SR'19 is human evaluation of two aspects of each output sentence: readability and semantic similarity to the original sentence. The detailed results are presented in (Mille et al., 2019). On 4 of the 5 datasets involved in the human evaluation scheme, our system was outperformed significantly by only two other systems in terms of readability. In terms of semantic similarity we are outperformed by only 1 or 2 systems on three of the five datasets. Automatic evaluation was performed using three metrics, described also in (Mille et al., 2019). Table 2 presents macro-average values for the top four teams, those that submitted outputs for all datasets. Our system ranks second among these four teams on two out of three metrics. On individual datasets, our system mostly performs below or around the average of all systems, with the ex-

	use position	batch size	hidden size	layers	embedding size	dev acc
ar	True	128	512	2	100	93.68
en	True	128	512	2	100	96.09
es	True	128	512	2	100	98.70
fr	True	128	512	2	100	94.59
hi	True	128	512	2	100	98.26
id	True	256	128	1	100	93.77
ja	False	32	1024	2	100	91.61
ko	True	128	512	2	100	98.43
pt	True	128	512	2	100	91.43
ru	True	128	512	2	100	97.46
zh	False	32	128	1	100	98.81

Table 1: Highest performing configurations for each language. Dev acc refers to a randomly selected subsection of the train data as the dev sets did not have gold standard inflection.

	BLEU	NIST	DIST
IMS	79.97	12.79	81.62
BME-UW	50.04	11.39	56.11
LORIA	47.67	10.32	65.78
Tilburg	45.18	10.05	56.11

Table 2: Macro-average of the top four systems across all datasets

ception of one Russian and two Korean datasets where we are outperformed by only one system (IMS).

6 Conclusions, further work

The weighting scheme described in Section 3 is in many ways similar to the way psycholinguists think about grammatical rules. Those rules that are based on fewer examples are used more rarely. In the limiting case, singleton examples are rarely abstracted into rules, they are memorized as is, and the key mechanism for such examples to override the general rules, e.g. that *mice* overrides **mouses*, is the same Elsewhere Principle (Giegerich, 2001) that we see as a derived, emergent property of the system.

Perhaps one modification that would bring the system even closer to psychological reality would be to use morphological features when restoring the id-s. While this remains future work, we consider it a strong point in favor of XAI that such questions can be raised: explainability makes it possible to leverage decades of psycholinguistic work, currently almost entirely ignored in the deep neural net paradigm which, in its laboratory pure form, pays no attention to biological or psychological evidence.

References

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria. Association for Computational Linguistics.

Bruno Courcelle. 1993. Graph grammars, monadic second-order logic and the theory of graph minors. *Contemporary Mathematics*, 147:565–565.

Bruno Courcelle and Joost Engelfriet. 2012. *Graph structure and monadic second-order logic*. Cambridge University Press.

Marie-Catherine De Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D Manning. 2014. Universal Stanford dependencies: A cross-linguistic typology. In *LREC*, volume 14, pages 4585–92.

Heinz J. Giegerich. 2001. Synonymy blocking and the elsewhere condition: lexical morphology and the speaker. *Transactions of the Philological Society*, 99(1):65–98.

Johannes Gontrum, Jonas Groschwitz, Alexander Koller, and Christoph Teichmann. 2017. Alto: Rapid prototyping for parsing and translation. In *Proceedings of the Software Demonstrations of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 29–32.

Jonas Groschwitz, Alexander Koller, and Christoph Teichmann. 2015. Graph parsing with s-graph grammars. In *Proceedings of the 53rd ACL and 7th IJCNLP*, Beijing.

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)*, pages 1831–1841, Melbourne, Australia. Association for Computational Linguistics.

David Gunning. 2017. Explainable Artificial Intelligence (XAI). Defense Advanced Research Projects Agency, DARPA/I20. `https://www.darpa.mil/attachments/XAIProgramUpdate.pdf`.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Paul Kiparsky. 1973. 'elsewhere' in phonology. In Stephen R. Anderson and Paul Kiparsky, editors, *A Festschrift for Morris Halle*, pages 93–106. Holt, Rinehart, and Winston, New York.

Alexander Koller. 2015. Semantic construction with graph grammars. In *Proceedings of the 14th International Conference on Computational Semantics (IWCS)*, London.

Alexander Koller and Nikos Engonopoulos. 2017. Integrated sentence generation using charts. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 139–143.

Alexander Koller and Marco Kuhlmann. 2011. A generalized view on parsing and translation. In *Proceedings of the 12th International Conference on Parsing Technologies (IWPT)*, Dublin.

Alexander Koller and Marco Kuhlmann. 2012. Decomposing tag algorithms using simple algebraizations. In *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+ 11)*, pages 135–143.

András Kornai, Judit Ács, Márton Makrai, Dávid Márk Nemeskey, Katalin Pajkossy, and Gábor Recski. 2015. Competence in lexical semantics. In *Proceedings of the Fourth Joint Conference on Lexical and Computational Semantics (*SEM 2015)*, pages 165–175, Denver, Colorado. Association for Computational Linguistics.

Thang Luong, Hieu Pham, and Christopher D. Manning. 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. Association for Computational Linguistics.

Simon Mille, Anja Belz, Bernd Bohnet, Yvette Graham, and Leo Wanner. 2019. The Second Multilingual Surface Realisation Shared Task (SR'19): Overview and Evaluation Results. In *Proceedings of the 2nd Workshop on Multilingual Surface Realisation (MSR), 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Hong Kong, China.

Simon Mille, Anja Belz, Bernd Bohnet, and Leo Wanner. 2018. Underspecified universal dependency structures as inputs for multilingual surface realisation. In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 199–209, Tilburg University, The Netherlands. Association for Computational Linguistics.

Joakim Nivre, Mitchell Abrams, Željko Agić, et al. 2018. Universal dependencies 2.3. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Proc. NIPS*, pages 3104–3112, Montreal, CA.

Evelin Ács, Ákos Holló-Szabó, and Gábor Recski. 2019. Parsing noun phrases with Interpreted Regular Tree Grammars. In *XV. Magyar Számítógépes Nyelvészeti Konferencia*, pages 301–313.