Parsing noun phrases with Interpreted Regular Tree Grammars

Evelin Ács[1], Ákos Holló-Szabó[1], Gábor Recski[2]

[1] Department of Automation and Applied Informatics
Budapest University of Technology and Economics
[2] Apollo.AI

Abstract. Several common tasks in natural language processing (NLP) involve graph transformation, in particular those that handle syntactic trees, dependency structures such as Universal Dependencies (UD) [1], or semantic graphs such as AMR [2] and 4lang [3]. Interpreted Regular Tree Grammars (IRTGs) [4] encode the correspondence between sets of such structures and have in recent years been used to perform both syntactic and semantic parsing. In this paper we introduce our tool that is capable of automatic IRTG generation. Our IRTG covers 83% of noun phrases (NPs) from the Wall Street Journal section of the Penn Treebank and a pilot experiment had also been made for retrieving surface realizations from UD graphs using independent data. We also describe this generated IRTG which allows for simultaneous generation of structures of various types and can be used for semantic parsing, generation, and semantics-driven translation.

1 Introduction

One of the most limiting factors in common tasks in NLP, such as machine translation, question answering and natural language inference, is the absence of high-quality deep semantic parsing. The state-of-the-art tools are mostly based on deep learning, which encode the meaning of words in multidimensional vector spaces, and the understanding of the structures of these representations is very limited. Another approach is using semantic representations based on concept networks. The automatic generation of these representations is also limited, but they facilitate more explicit analysis of tasks close to general artificial intelligence, such as natural language inference or machine comprehension. Syntactic parsers for natural language are key components for most processing pipelines within human language technologies (HLT). A common approach taken by modern HLT systems is dependency parsing, which maps raw text to directed acyclic graphs over words of each input sentence. One of the multiple dependency parsing mechanisms implemented in the Stanford Parser [5] creates dependency graphs by applying rule-based transformations on constituency structures output by a probabilistic context-free grammar (PCFG) parser. The dep_to_4lang component of the 4lang library builds graphs of syntax-independent concepts from the output of any dependency parser that conforms to the Universal Dependencies

format, also using simple template-matching. In this end-to-end semantic parser pipeline all components implement a form of graph transformation so its functionality can be unified in a single graph grammar. We use Interpreted Regular Tree Grammars [4] to simultaneously encode transformations between strings, phrase structure trees and UD and 4lang graphs. Section 2 provides an overview of the used or related tools and technologies, including the dependency parsing component of the Stanford Parser, Universal Dependencies (supported by the Stanford Parser), the 4lang formalism, and the IRTGs. In Section 3 we present a regular tree grammar with four interpretations, corresponding to strings, phrase structure trees, UD graphs, and 4lang graphs. Our grammar is non-deterministic, which means that given an input structure, it can generate more than one derivations, resulting in many different output structure configurations regarding all interpretations. Such a grammar allows converting from any of the above algebraic structures to any or all of the others, e.g. generating English text from dependency graphs. It can also be trained on correspondences between grammatical and semantic structures and surface realizations. The grammar discussed in this paper covers 83% of NPs of the Wall Street Journal section of the Penn Treebank and a pilot experiment was also executed on a small independent (i. e. not used for generating the grammar) test data for generating English text from UD graphs, which resulted in a limited number of successful parses, and also revealed some problems that need further investigation (discussed in Section 3.2).

2 Background

2.1 Dependency parsing with the Stanford Parser

The Stanford parser includes a component for dependency parsing [5] , which consists of two phases: dependency extraction and dependency typing. After parsing the phrase structure tree of a sentence, semantic heads need to be identified, rather than syntactic ones, to be more useful for semantic dependency analysis (extractions), i.e. choose content words as heads (rather than auxiliaries, complementizers, etc.) and other words as dependents. Ambiguous structures or multi-headed structures (represented as flat structures in the Penn Treebank) also need to be resolved. For dependency labeling, one or more patterns are defined over the phrase structure tree using the `tregex` tree expression syntax [6]. For example, `"ADJP !< CC|CONJP < (JJ|NNP $ JJ|NNP=target)"` describes an ADJP not dominating a CC or CONJP, and dominating a JJ or NP with a sister JJ or NNP. This is one of the patterns which describe the UD relation `amod`. In theory, every node is matched against every pattern and from the matching patterns, the most specific relation decides the type of the dependency.

2.2 UD

The Universal Dependencies (UD) project[1] [1] is a cross-linguistically consistent annotation system and treebanks for 60+ languages. It provides a universal inventory of categories and annotation guidelines while allowing language-specific extensions. UD has evolved from Stanford Dependencies [7] by merging it with Google universal tags [8], a revised subset of the Interset feature inventory [9], and a revised version of the CoNLL-X format [10]. The two groups of core dependencies are the clausal relations (which describe syntactic roles concerning the predicate), and the modifier relations (which categorize the ways words modify their heads). For the sake of a uniform analysis, nouns introduced by or having attached prepositions and other case markings are treated as the head of these relations. The formalism follows a lexicalist approach to enable computational use: the syntactic structure consists of lexical elements linked by binary asymmetrical, one-to-one relations as opposed to constituency, which is a one-to-one-or-more correspondence. UD also allows the cross-linguistic evaluation of dependency parsers: more than 30 teams participated in the 2017 CoNLL shared task on multilingual dependency parsing [11].

2.3 4lang

4lang [3] is a formalism which builds directed graphs for semantic representation while abstracting away from syntax: in such graphs, nodes stand for language-independent concepts, which do not have any grammatical attributes, and contain shared knowledge of competent speakers. For example, *freeze(N)*, *freeze(V)*, *freezing*, or *frozen* are not differentiated in 4lang representations, resulting in a many-to-one relation between 4lang concepts and between words of a given language.

Nodes are connected via three types of edges: 0-edge represents attribution (*flower* $\xrightarrow{0}$ *beautiful*), the IS_A relation (*flower* $\xrightarrow{0}$ *plant*) and unary predication (*flower* $\xrightarrow{0}$ *bloom*). 1 and 2-edges connect binary predicates to their arguments (*James* $\xleftarrow{1}$ *like* $\xrightarrow{2}$ *dog*). Binary (transitive) elements, that do not correspond to any word in a given phrase or sentence, are marked by UPPERCASE printnames.

There is another type of edge configuration, $w_1 \underset{0}{\overset{1}{\rightleftharpoons}} w_2$, that may appear in 4lang graphs. This is necessary to consistently represent the relation between the subject and the predicate: considering the example sentences *I'm writing* (*i* $\xrightarrow{0}$ *write*) and *I'm writing a paper* (*i* $\xleftarrow{1}$ *write* $\xrightarrow{2}$ *paper*), these representations would suggest that the relation between *I* and *write* is dependent on whether the object is specified or not. The example graph of Figure 1 represents the meaning of the sentence *John loves Mary's cat.*

The 4lang library contains tools for building directed graphs from raw text (text_to_4lang) and dictionary definitions (dict_to_4lang). The core module of the 4lang library, dep_to_4lang obtains dependency relations from text by

[1] `http://universaldependencies.org/`

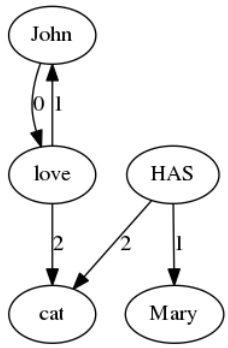


Fig. 1: 4lang graph of the sentence *'John loves Mary's cat.*

processing the output of the Stanford parser [5] and applies a simple mapping from Stanford dependencies to 4lang subgraphs.

4lang is also the name of a manually created concept dictionary [12] which contains more than 2000 definitions of language-independent concepts in four languages (Hungarian, English, Latin and Polish), hence its name.

2.4 IRTGs and s-graphs

IRTG Interpreted regular tree grammars [4] are context-free grammars in which each rule is mapped over an arbitrary number of algebras. Thus, when one rewrite rule gets applied on one of the algebras, the corresponding operations are executed on objects in each algebra. This means that an IRTG parser can accept inputs in any of the defined interpretations and can convert the input data into any of the other interpretations as output. In an example case of Figure 2, which was implemented using Alto (an open-source parser, will be introduced later in this section) [13], the two interpretations are the `string` and the `tag tree` algebras, so it can either convert a string to a phrase structure tree and vice versa. The `string algebra` has only one binary operation symbol *, which evaluates to `string concatenation`. Operations of the `tag tree algebra` will be discussed later in this section.

The rules are processed as follows: first a derivation tree is built using regular tree grammars, then a function called tree homomorphism maps the derivation tree over a term ($f(t_1, ..., t_n)$ stands for the tree with the root label f and subtrees $t_1, ..., t_n$), then the tree is evaluated over the algebra. For a formal description the reader is referred to [14].

tag tree algebra The `tag tree algebra` [15] is a simple tree manipulation language. Trees consist of single edges and nodes. Nodes marked with * are called `holes`. Subtrees can be combined with the elementary tree using `substitution` (leaving a variable in the appropriate place) and `adjunction` (using the @ op-

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

S! -> s(NP)
[string] ?1
[tree] ?1

NP -> np(DT, N_BAR)
[string] *(?1,?2)
[tree] @(?2,?1)

N_BAR -> n_bar(JJ, NN)
[string] *(?1,?2)
[tree] NP(*,?1,?2)

// terminals

DT -> a
[string] a
[tree] DT(a)

JJ -> large
[string] large
[tree] JJ(large)

NN -> dog
[string] dog
[tree] NN(dog)
```

Fig. 2: An example IRTG.

erator), as shown in Figure 3. T1 and T3 stand for elementary trees, in which T2 (the subtree, which is allowed to contain additional holes) will be inserted.

```
T1= S(*, VP(V(loves), NP(NN(Mary))))
T2= NP(NN(John))
T3= S(*, VP(V(loves), *))
@(T1,T2)= S(NP(NN(John)), VP(V(loves), NP(NN(Mary))))
@(T3, T2)= S(NP(NN(John)), VP(V(loves), NP(NN(John))))
```

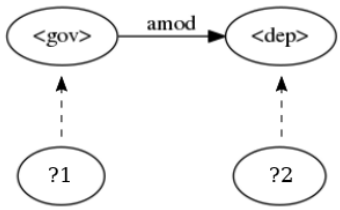
Fig. 3: An example illustrating the operations of the tag tree algebra.

Figure 3 also shows that when there are multiple holes in the tree, the binary operator @ inserts the same subtree to all of them.

s-graph The `s-graph algebra` [16] , which has been used for semantic parsing previously [17], is designed for building larger graphs from smaller ones. An s-graph's nodes are marked with `source labels` from a fixed finite set. Source labels identify which nodes should be merged when executing operations. Sources are the nodes which carry source names. An s-graph can consist of a single `root` node. The operations of the s-graph algebra are `merge` (which returns a graph that contains all the nodes and edges of its operands), `rename` (which returns a graph like the original except given source names have been changed) and `forget` (which returns a graph like the original except a given source name is removed from all nodes with that name). When used for semantic parsing [4], source names correspond to the semantic argument positions of the given grammar. The example in Figure 4 demonstrates the usage of these operators by connecting two subgraphs. ?1 and ?2 represent the subgraphs to be merged to the initial graph, which is between the quotation marks. First, the `root` label of ?2 is renamed to `dep`. Then it can be merged with the initial graph's `dep` node. After the merge the `dep` label is removed, because this node will not be used in subsequent rules. In the final step the `root` of ?1 is merged with the `root` of the initial graph.

Alto The Algebraic Language Toolkit, or `Alto`[2] [13] is an open-source parser which implements a variety of algebras for use with IRTGs, including the s-graph and the tag tree algebras. It has been used previously for graph transformations and semantic parsing as well [17], [4].

[2] `https://bitbucket.org/tclup/alto`




```
merge(f_dep(merge("(r<root> :amod (d<dep>))", r_dep(?2))),?1)
```

Fig. 4: An example illustrating the operations of the s-graph algebra.

3 Parsing NPs with Interpreted Regular Tree Grammars

3.1 Rule generation

In this section we present the first steps of creating a framework which encodes transformations between strings, phrase structure trees and UD and 4lang graphs. Our system supports the UD v2.1 format and is capable of automatic rule generation from Penn Treebank lines and those parsed by the Stanford parser. The code can be found at `github.com/evelinacs/semantic\_parsing\_with\_IRTGs/tree/master/code/generate\_grammar/template\_based\_grammar\_generator`.

To generate the IRTG rules, the program compares the phrase structure tree and the dependency graph of each noun phrase. The input for the phrase structure tree data is in Penn Treebank format and the dependency graph data is extracted from the output of the Stanford parser (which is generated by pattern matching). During rule generation, the program compares the relations between two words in the phrase structure tree and in the dependency graph. Given the rigidity and ordered nature of the tag tree algebra, the rule generation is based on the phrase structure of a subtree. This means that the RTG line (its left-hand side, and the arguments on the right-hand side) is derived from the phrase structure tree, and the [tree] interpretation either simply reflects the node type of the head and the number of its children or a merge operation is performed between two subtrees. To generate the [ud] and [fourlang] interpretations, the order of the nodes in the phrase structure tree must be considered, as the direction of some edges in these graphs are reversed with regards to the order of nodes in the phrase structure tree, and this must be reflected in the generated rules. The edge types in the [fourlang] interpretation are derived from the UD edge types using a predefined mapping implemented before Figure 1. This mapping also contains information regarding technical nodes in the 4lang formalism, which must be introduced for some relations in the [fourlang] interpretation to produce the correct 4lang representation of the given structure.

In our experiment we have limited our grammar to trees with an NP as a root node and any node within a tree must have at most three children. This subset makes 84,8% of all NPs of the Penn Treebank.

Dependency Edge	
advcl	$w_1 \overset{0}{\rightarrow} w_2$
advmod	
amod	
nmod	
nummod	
appos	$w_1 \underset{0}{\overset{0}{\rightleftharpoons}} w_2$
dislocated	
csubj	$w_1 \underset{0}{\overset{1}{\rightleftharpoons}} w_2$
nsubj	
ccomp	$w_1 \overset{2}{\rightarrow} w_2$
obj	
xcomp	
The treatment of `case`	
case + nmod	$w_1 \overset{1}{\leftarrow} w_3 \overset{2}{\rightarrow} w_2$
case + nsubj	
case + obl	
English subtypes	
obl:npmod	$w_1 \overset{0}{\rightarrow} w_2$
nmod:tmod	$w_1 \overset{1}{\leftarrow} \texttt{AT} \overset{2}{\rightarrow} w_2$
obl:tmod	
nmod:poss	$w_2 \overset{1}{\leftarrow} \texttt{HAS} \overset{2}{\rightarrow} w_1$

Table 1. UD-conform version of the `dep_to_4lang` mapping.

Figure 5 presents the structure of the phrase *a long way* regarding the tree and graph interpretations.

As the structures of the phrase structure tree and the UD graph are fundamentally different, their respective rules for each interpretation must be implemented to accommodate that. Appendix A presents the rules which are responsible for parsing the aforementioned sentence.

When processing this IRTG, first a derivation tree (Figure 6) is built by parsing the input according to the specified interpretation and the corresponding RTG rules. If a viable derivation tree is found, corresponding outputs are created based on all the other interpretations. For example, if the input is an UD graph, then a phrase structure tree, a string and a 4lang graph can be retrieved as outputs. The outputs are built from the leaf nodes of the derivation tree according to the rules of the interpretations' respective algebra. In the terminal rules, a word, a subtree or a subgraph is produced for the string, the phrase

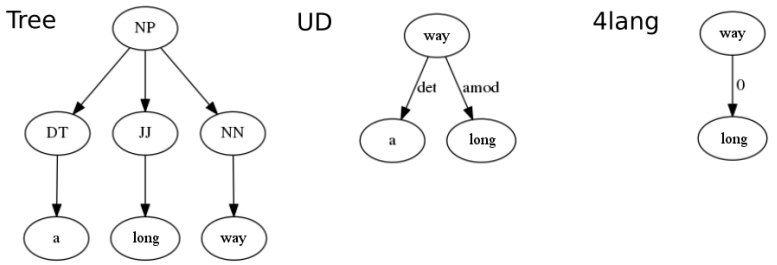


Fig. 5: Configurations of the phrase structure tree, the UD graph and the 4lang graph for the phrase *a long way*.

structure tree, and the UD and 4lang graph interpretations. Then the `N_BAR -> amod_JJ_NN_NBAR(JJ,NN)` rule gets applied. In the string interpretation, a concatenation of the arguments is executed. In the tree interpretation the subtrees are attached to an NP labeled head along with a placeholder node, marked by *. The UD and 4lang interpretations are almost the same in this step: first the root label in the subgraph provided as the second argument is renamed to dep, then merged with the node having the same label in the graph between the quotation marks. Next the dep label is removed, making the node with this label internal (not used in later operations). Then the subgraph provided as the first argument is merged with the root of the graph from the result of the previous operation. In the rule `NP -> det_DT_NBAR_NP(DT, N_BAR)` the same operations are applied in the string and UD interpretations. In the tree interpretation, the subtree provided as the first argument replaces the placeholder node from the previous rule. In the 4lang interpretation only the second argument is passed along, as the determiner doesn't contain much semantic information. Finally the `S! -> s(NP)` rule is simply an identity operation that is, it returns its single argument in all four interpretations.

Given the previous process, transforming the string *a long way* to a UD graph happens as the following. First, each word is mapped to the corresponding terminal rules which in turn create three subgraphs representing each word as a root labeled node. Then following the derivation tree the rule responsible for concatenating the words *long* and *way* is matched (`N_BAR -> amod_JJ_NN_NBAR(JJ,NN)`) which creates an `amod` relation between the nodes representing the words by merging them with the graph provided within the [ud] interpretation. The node label is removed from the node representing the word *long*, and the other node (representing *way*) keeps its root label so it can be used in the following rules. Next, similar to the previous step, the rule concatenating the two substrings *a* and *long way* is matched (`NP -> det_DT_NBAR_NP(DT, N_BAR)`) which creates the `det` relation between the words *a* and *way* by merging the subgraph resulting from the previous step and the node representing the word *a*. Finally the start rule of the derivation tree creation is applied which contains no transformation, so the result is the graph obtained from the previous step.

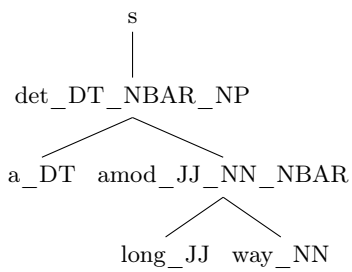


Fig. 6: The derivation tree of the phrase *a long way*.

3.2 Evaluation

Parsing the Penn Treebank To create a fully functioning IRTG, terminal nodes (e.g. the words) had to be appended to the rule file. We also had to convert from the Penn Treebank format to one that is understood by Alto. These tasks were implemented in Python and the scripts can be found in our repository.

Our grammar have been evaluated on trees with an NP as a root node and any node within a tree must have at most three children. This subset makes 84,8% of all NPs of the Wall Street Journal section of the Penn Treebank. The WSJ section contains 243 914 NPs of which 206 841 meet the aforementioned restrictions. Parsing this subset using the generated grammar resulted in 202 549 successful parses, which makes 97,9% of the test data and 83% of all NPs. We can transform structures in this subset from any interpretation to any other interpretation, including converting UD graphs to strings.

Generating text from UD graphs To test our grammar on an independent dataset (i.e. different from the one that was used for generating it), we have used a subset of the test data of the Surface Realization Shared Task [18] as well, which contains UD graphs. We extracted subgraphs corresponding to all NPs of the first 20 sentences, resulting in a small dataset of 38 noun phrases, then manually reviewed the output of parsing this dataset using the previously generated grammar. Our goal was to derive surface realizations from UD graphs. Our grammar successfully parsed 18 NPs, which makes 47% of the test data. For the remaining 20 graphs no output was generated. These either contain dependencies which are not presented in the original input (e.g. flat), or due to some bugs in the grammar (which need to be investigated), these structures cannot be recognized by the parser. Some of the successfully parsed graphs had incorrect word order. As our grammar is non-deterministic, such results are expected due to the possibility of building multiple structures for the same input data. Creating a new grammar using maximum likelihood training might resolve the problem, but the causes of these errors need further investigation.

4 Conclusion and ongoing work

In this paper we presented an IRTG which implements a mapping between four formalisms, i.e. strings, the output of the Stanford parser, UD v2.1 and 4lang. Our system covers 83% of the NPs of the Wall Street Journal section of the Penn Treebank and was capable to retrieve surface realizations from a small subset of the test data of the Surface Realization Shared Task with limited success.

The grammar allows converting from any of the above algebraic structures to any or all of the others. A probabilistic version of this grammar (as Alto supports probabilistic IRTGs) can be trained using any parallel data. If a single probabilistic IRTG were to implement the parallel parsing of strings, syntactic constituency structures, dependency graphs and semantic graphs, it could be trained simultaneously on each of these types of gold-standard data, resulting in a single end-to-end system for semantic parsing. This might serve as a basis for semantic generation, paraphrasing or machine translation in the future.

References

1. De Marneffe, M.C., Dozat, T., Silveira, N., Haverinen, K., Ginter, F., Nivre, J., Manning, C.D.: Universal Stanford dependencies: A cross-linguistic typology. In: LREC. Volume 14. (2014) 4585–92
2. Banarescu, L., Bonial, C., Cai, S., Georgescu, M., Griffitt, K., Hermjakob, U., Knight, K., Koehn, P., Palmer, M., Schneider, N.: Abstract meaning representation for sembanking. In: Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse, Sofia, Bulgaria, Association for Computational Linguistics (2013) 178–186
3. Kornai, A., Ács, J., Makrai, M., Nemeskey, D.M., Pajkossy, K., Recski, G.: Competence in lexical semantics. In: Proceedings of the Fourth Joint Conference on Lexical and Computational Semantics (*SEM 2015), Denver, Colorado, Association for Computational Linguistics (2015) 165–175
4. Koller, A.: Semantic construction with graph grammars. In: Proceedings of the 14th International Conference on Computational Semantics (IWCS), London (2015)
5. DeMarneffe, M.C., MacCartney, W., Manning, C.: Generating typed dependency parses from phrase structure parses. In: Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC). Volume 6., Genoa, Italy (2006) 449–454
6. Levy, R., Andrew, G.: Tregex and tsurgeon: tools for querying and manipulating tree data structures. In: Proceedings of the fifth international conference on Language Resources and Evaluation, Citeseer (2006) 2231–2234
7. De Marneffe, M.C., Manning, C.D.: Stanford typed dependencies manual. Technical report, Technical report, Stanford University (2008)
8. Petrov, S., Das, D., McDonald, R.: A universal part-of-speech tagset. arXiv preprint arXiv:1104.2086 (2011)
9. Zeman, D.: Reusable tagset conversion using tagset drivers. In: LREC. Volume 2008. (2008) 28–30
10. Buchholz, S., Marsi, E.: CoNLL-X Shared Task on multilingual dependency parsing. In: Proceedings of the Tenth Conference on Computational Natural Language Learning, Association for Computational Linguistics (2006) 149–164

11. Zeman, D., Popel, M., Straka, M., Hajic, J., Nivre, J., Ginter, F., Luotolahti, J., Pyysalo, S., Petrov, S., Potthast, M., Tyers, F., Badmaeva, E., Gokirmak, M., Nedoluzhko, A., Cinkova, S., Hajic jr., J., Hlavacova, J., Kettnerová, V., Uresova, Z., Kanerva, J., Ojala, S., Missilä, A., Manning, C.D., Schuster, S., Reddy, S., Taji, D., Habash, N., Leung, H., de Marneffe, M.C., Sanguinetti, M., Simi, M., Kanayama, H., dePaiva, V., Droganova, K., Martínez Alonso, H., Çöltekin, c., Sulubacak, U., Uszkoreit, H., Macketanz, V., Burchardt, A., Harris, K., Marheinecke, K., Rehm, G., Kayadelen, T., Attia, M., Elkahky, A., Yu, Z., Pitler, E., Lertpradit, S., Mandl, M., Kirchner, J., Alcalde, H.F., Strnadová, J., Banerjee, E., Manurung, R., Stella, A., Shimada, A., Kwak, S., Mendonca, G., Lando, T., Nitisaroj, R., Li, J.: CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. In: Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, Vancouver, Canada, Association for Computational Linguistics (2017) 1–19
12. Kornai, A., Makrai, M.: A 4lang fogalmi szótár. In Tanács, A., Vincze, V., eds.: IX. Magyar Számítógépes Nyelvészeti Konferencia. (2013) 62–70
13. Gontrum, J., Groschwitz, J., Koller, A., Teichmann, C.: 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. (2017) 29–32
14. Koller, A., Kuhlmann, M.: A generalized view on parsing and translation. In: Proceedings of the 12th International Conference on Parsing Technologies (IWPT), Dublin (2011)
15. Koller, A., Kuhlmann, M.: Decomposing tag algorithms using simple algebraizations. In: Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+ 11). (2012) 135–143
16. Courcelle, B.: Graph grammars, monadic second-order logic and the theory of graph minors. Contemporary Mathematics **147** (1993) 565–565
17. Groschwitz, J., Koller, A., Teichmann, C.: Graph parsing with s-graph grammars. In: Proceedings of the 53rd ACL and 7th IJCNLP, Beijing (2015)
18. Mille, S., Belz, A., Bohnet, B., Graham, Y., Pitler, E., Wanner, L.: The first multilingual surface realisation shared task (sr'18): Overview and evaluation results. In: Proceedings of the First Workshop on Multilingual Surface Realisation. (2018) 1–12

A An example IRTG

The following is an example IRTG for the phrase *a long way*.

```
interpretation string: de.up.ling.irtg.algebra.StringAlgebra
interpretation tree: de.up.ling.irtg.algebra.TagTreeAlgebra
interpretation ud: de.up.ling.irtg.algebra.graph.GraphAlgebra
interpretation fourlang: de.up.ling.irtg.algebra.graph.GraphAlgebra

// IRTG for the phrase 'a long way'

S! -> s(NP)
[string] ?1
[tree] ?1
[ud] ?1
[fourlang] ?1

NP -> det_DT_NBAR_NP(DT, N_BAR)
[string] *(?1,?2)
[tree] @(?2,?1)
[ud] merge(f_dep(merge("(r<root> :det (d<dep>))", r_dep(?1))),?2)
[fourlang] ?2

N_BAR -> amod_JJ_NN_NBAR(JJ,NN)
[string] *(?1,?2)
[tree] NP(*,?1,?2)
[ud] merge(f_dep(merge("(r<root> :amod (d<dep>))", r_dep(?1))),?2)
[fourlang] merge(f_dep(merge("(r<root> :0 (d<dep>))", r_dep(?1))),?2)

// terminals:

DT -> a_DT
[string] a
[tree] DT(a)
[ud] "(a<root> / a)"
[fourlang] "(a<root> / a)"

JJ -> long_JJ
[string] long
[tree] JJ(long)
[ud] "(long<root> / long)"
[fourlang] "(long<root> / long)"

NN -> way_NN
[string] way
[tree] NN(way)
[ud] "(way<root> / way)"
[fourlang] "(way<root> / way)"
```