Alto: Rapid Prototyping for Parsing and Translation

Johannes Gontrum **Jonas Groschwitz**[*] **Alexander Koller**[*] **Christoph Teichmann**[*]
University of Potsdam, Potsdam, Germany / [*] Saarland University, Saarbrücken, Germany
gontrum@uni-potsdam.de / [*] {groschwitz|koller|teichmann}@coli.uni-saarland.de

Abstract

We present *Alto*, a rapid prototyping tool for new grammar formalisms. Alto implements generic but efficient algorithms for parsing, translation, and training for a range of monolingual and synchronous grammar formalisms. It can easily be extended to new formalisms, which makes all of these algorithms immediately available for the new formalism.

1 Introduction

Whenever a new grammar formalism for natural language is developed, there is a prototyping phase in which a number of standard algorithms for the new formalism must be worked out and implemented. For monolingual grammar formalisms, such as (probabilistic) context-free grammar or tree-adjoining grammar, these include algorithms for chart parsing and parameter estimation. For synchronous grammar formalisms, we also want to decode inputs into outputs, and binarizing grammars becomes nontrivial. Implementing these algorithms requires considerable thought and effort for each new grammar formalism, and can lead to faulty or inefficient prototypes. At the same time, there is a clear sense that these algorithms work basically the same across many different grammar formalisms, and change only in specific details.

In this demo, we address this situation by introducing *Alto*, the Algebraic Language Toolkit. Alto is based on Interpreted Regular Tree Grammars (IRTGs; (Koller and Kuhlmann, 2011)), which separate the derivation process (described by probabilistic regular tree grammars) from the interpretation of a derivation tree into a value of the language. In this way, IRTGs can capture a wide variety of monolingual and synchronous grammar formalisms (see Fig. 1 for some examples). By selecting an appropriate algebra in which the values of the language are constructed, IRTGs can describe languages of objects that are not strings, including string-to-tree and tree-to-tree mappings, which have been used in machine translation, and synchronous hyperedge replacement grammars, which are being used in semantic parsing.

One advantage of IRTGs is that a variety of algorithms, including the ones listed above, can be expressed generically in terms of operations on regular tree grammars. These algorithms apply identically to all IRTG grammars and Alto offers optimized implementations. Only an algebra-specific decomposition operation is needed for each new algebra. Thus prototyping for a new grammar formalism amounts to implementing an appropriate algebra that captures new interpretation operations. All algorithms in Alto then become directly available for the new formalism, yielding an efficient prototype at a much reduced implementation effort.

Alto is open source and regularly adds new features. It is available via its website:
`https://bitbucket.org/tclup/alto`.

2 An example grammar

Let us look at an example to illustrate the Alto workflow. We will work with a synchronous string-to-graph grammar, which Alto's GUI displays as in Fig. 2. The first and second column describe a weighted regular tree grammar (wRTG, (Comon et al., 2007)), which specifies how to rewrite nonterminal symbols such as S and NP recursively in order to produce *derivation trees*. For instance, the tree shown in the leftmost panel of Fig. 3 can be derived using this grammar, starting with the start symbol S, and is assigned a weight (= probability) of 0.24. These derivation trees

Formalism	Reference	Algebra(s)
Context-Free Grammars (CFGs)	(Hopcroft and Ullman, 1979)	String
Hyperedge Replacement Grammars (HRGs)	(Chiang et al., 2013)	Graph
Tree Substitution Grammars	(Sima'an et al., 1994)	String / Tree
Tree-Adjoining Grammars (TAGs)	(Joshi et al., 1975)	TAG string / TAG tree
Synchronous CFGs	(Chiang, 2007)	String / String
Synchronous HRGs	(Chiang et al., 2013)	String / Graph
String to Tree Transducer	(Galley et al., 2004)	String / Tree
Tree to Tree Transducer	(Graehl et al., 2008)	Tree / Tree

Figure 1: Some grammar formalisms that Alto can work with.

serve as abstract syntactic representations, along the lines of derivation trees in TAG.

Next, notice the column "english" in Fig. 2. This column describes how to *interpret* the derivation tree in an interpretation called "english". It first specifies a *tree homomorphism*, which maps derivation trees into terms over some algebra by applying certain rewrite rules bottom-up. For instance, the "boy" node in the example derivation tree is mapped to the term $t_1 = *(\mathsf{the}, \mathsf{boy})$. The entire subtree then maps to a term of the form $*(?1, *(\mathsf{wants}, *(\mathsf{to}, ?2)))$, as specified by the row for "wants2" in the grammar, where $?1$ is replaced by t_1 and $?2$ is replaced by the analogous term for "go". The result is shown at the bottom of the middle panel in Fig. 3. Finally, this term is *evaluated* in the underlying algebra; in this case, a simple string algebra, which interprets the symbol $\star$ as string concatenation. Thus the term in the middle panel evaluates to the string "the boy wants to go", shown in the "value" field.

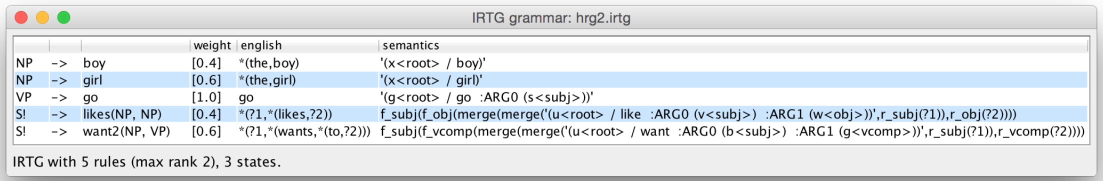
The example IRTG also contains a column called "semantics". This column describes a second interpretation of the derivation tree, this time into an algebra of graphs. Because the graph algebra is more complex than the string algebra, the function symbols look more complicated. However, the general approach is exactly the same as before: the grammar specifies how to map the derivation tree into a term (bottom of rightmost panel in Fig. 3), and then this term is evaluated in the respective algebra (here, the graph shown at the top of the rightmost panel).

Thus the example grammar is a synchronous grammar which describes a relation between strings and graphs. When we parse an input string w, we compute a *parse chart* that describes all grammatically correct derivation trees that interpret to this input string. We do this by computing a *decomposition grammar* for w, which describes all terms over the string algebra that evaluate to w; this step is algebra-specific. From this, we calculate a regular tree grammar for all derivation trees that the homomorphism maps into such a term, and then intersect it with the wRTG. These operations can be phrased in terms of generic operations on RTGs; implementing these efficiently is a challenge which we have tackled in Alto. We can compute the best derivation tree from the chart, and map it into an output graph. Similarly, we can also decode an input graph into an output string.

3 Algorithms in Alto

Alto can read IRTG grammars and corpora from files, and implements a number of core algorithms, including: automatic binarization of monolingual and synchronous grammars (Büchse et al., 2013); computation of parse charts for given input objects; computing the best derivation tree; computing the k-best derivation trees, along the lines of (Huang and Chiang, 2005); and decoding the best derivation tree(s) into output interpretations. Alto supports PCFG-style probability models with both maximum likelihood and expectation maximization estimation. Log-linear probability models are also available, and can be trained with maximum likelihood estimation. All of these functions are available through command-line tools, a Java API, and a GUI, seen in Fig. 2 and 3.

We have invested considerable effort into making these algorithms efficient enough for practical use. In particular, many algorithms for wRTGs in Alto are implemented in a lazy fashion, i.e. the rules of the wRTG are only calculated by need; see e.g. (Groschwitz et al., 2015; Groschwitz et al., 2016). Obviously, Alto cannot be as efficient for well-established tasks like PCFG parsing



IRTG grammar: hrg2.irtg

			weight	english	semantics
NP	->	boy	[0.4]	*(the,boy)	'(x<root> / boy)'
NP	->	girl	[0.6]	*(the,girl)	'(x<root> / girl)'
VP	->	go	[1.0]	go	'(g<root> / go :ARG0 (s<subj>))'
S!	->	likes(NP, NP)	[0.4]	*(?1,*(likes,?2))	f_subj(f_obj(merge(merge('(u<root> / like :ARG0 (v<subj>) :ARG1 (w<obj>))',r_subj(?1)),r_obj(?2))))
S!	->	want2(NP, VP)	[0.6]	*(?1,*(wants,*(to,?2)))	f_subj(f_vcomp(merge(merge('(u<root> / want :ARG0 (b<subj>) :ARG1 (g<vcomp>))',r_subj(?1)),r_vcomp(?2))))

IRTG with 5 rules (max rank 2), 3 states.

Figure 2: An example IRTG with an English and a semantic interpretation (Alto screenshot).

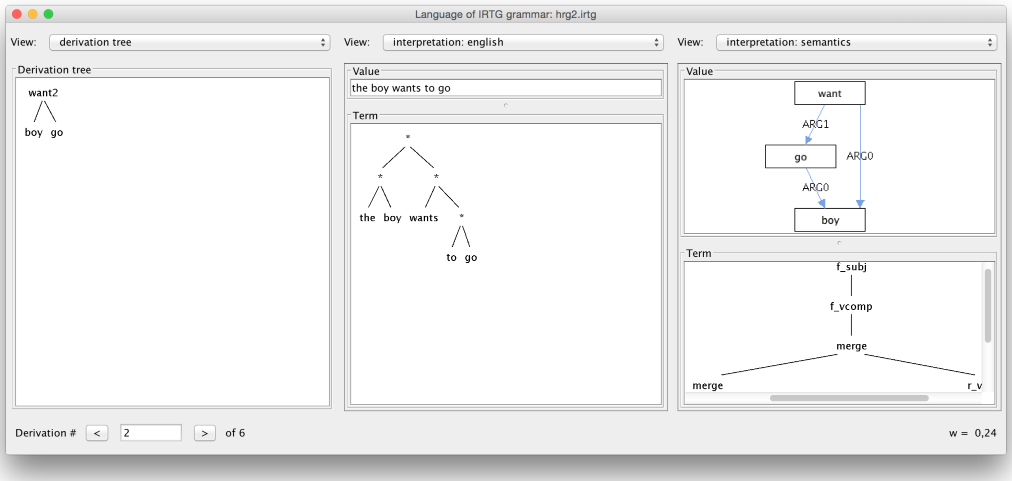


Figure 3: A derivation tree with interpreted values (Alto screenshot).

as a parser that was implemented and optimized for this specific grammar formalism. Nonetheless, Alto is fast enough for practical use with treebank-scale gramars, and for less mainstream grammar formalisms can be faster than specialized implementations for these formalisms. For instance, Alto is the fastest published parser for Hyperedge Replacement Grammars (Groschwitz et al., 2015). Alto contains multiple algorithms for computing the intersection and inverse homomorphism of RTGs, and a user can choose the combination that works best for their particular grammar formalism (Groschwitz et al., 2016).

The most recent version adds further performance improvements through the use of a number of pruning techniques, including coarse-to-fine parsing (Charniak et al., 2006). With these, Section 23 of the WSJ corpus can be parsed in a couple of minutes.

4 Extending Alto

As explained above, Alto can capture any grammar formalism whose derivation trees can be described with a wRTG, by interpreting these into different algebras. For instance, the difference between Context-Free and Tree-Adjoining Grammars in Alto is that CFGs use the simple string algebra outlined in Section 2, whereas for TAG we use a special "TAG string algebra" which defines string wrapping operations (Koller and Kuhlmann, 2012). All algorithms mentioned in Section 3 are generic and do not make any assumptions about what algebras are being used. As explained above, the only algebra-specific step is to compute decomposition grammars for input objects.

In order to implement a new algebra, a user of Alto simply derives a class from the abstract base class `Algebra`, which amounts to specifying the possible values of the algebra (as a Java class) and implementing the operations of the algebra as Java methods. If Alto is also to parse objects from this algebra, the class needs to implement a method for computing decomposition grammars for the algebra's values. Alto comes with a number of algebras built in, including string algebras for Context-Free and Tree-Adjoining grammars as well as tree, set, and graph algebras. All of these can be used in parsing. By parsing sets in a set-to-string grammar for example, Alto can generate referring expressions (Engonopoulos and Koller, 2014).

Finally, Alto has a flexible system of *input and output codecs*, which can map grammars and algebra values to string representations and vice versa. A key use of these codecs is reading grammars in native input format and converting them into IRTGs. Users can provide their own codecs to maximize interoperability with existing code.

5 Conclusion and Future Work

Alto is a flexible tool for the rapid implementation of new formalisms. This flexibility is based on a division of concerns between the generation and the interpretation of grammatical derivations. We hope that the research community will use Alto on newly developed formalisms and on novel combinations for existing algebras. Further, the toolkit's focus on balancing generality with efficiency supports research using larger datasets and grammars.

In the future, we will implement algorithms for the automatic induction of IRTG grammars from corpora, e.g. string-to-graph corpora, such as the AMRBank, for semantic parsing (Banarescu et al., 2013). This will simplify the prototyping process for new formalisms even further, by making large-scale grammars for them available more quickly. Furthermore, we will explore ways for incorporating neural methods into Alto, e.g. in terms of supertagging (Lewis et al., 2016).

Acknowledgements

This work was supported by DFG grant KO 2916/2-1.

References

L. Banarescu, C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight, P. Koehn, M. Palmer, and N. Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop*.

M. Büchse, A. Koller, and H. Vogler. 2013. Generic binarization for parsing and translation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*.

E. Charniak, M. Johnson, M. Elsner, J. Austerweil, D. Ellis, I. Haxton, C. Hill, R. Shrivaths, J. Moore, M. Pozar, and T. Vu. 2006. Multilevel coarse-to-fine pcfg parsing. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the ACL*.

D. Chiang, J. Andreas, D. Bauer, K. M. Hermann, B. Jones, and K. Knight. 2013. Parsing graphs with hyperedge replacement grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*.

D. Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228.

H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi, and C. Löding. 2007. *Tree Automata Techniques and Applications*. published online - `http://tata.gforge.inria.fr/`.

N. Engonopoulos and A. Koller. 2014. Generating effective referring expressions using charts. In *Proceedings of the INLG and SIGDIAL 2014*.

M. Galley, M. Hopkins, K. Knight, and D. Marcu. 2004. What's in a translation rule? In *HLT-NAACL 2004: Main Proceedings*.

J. Graehl, K. Knight, and J. May. 2008. Training tree transducers. *Computational Linguistics*, 34(3):391–427.

J. Groschwitz, A. Koller, and C. 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*.

J. Groschwitz, A. Koller, and M. Johnson. 2016. Efficient techniques for parsing with tree automata. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

J. E. Hopcroft and J. Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Massachusetts.

L. Huang and D. Chiang. 2005. Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*.

A. K. Joshi, L. S. Levy, and M. Takahashi. 1975. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1):136–163.

A. Koller and M. Kuhlmann. 2011. A generalized view on parsing and translation. In *Proceedings of the 12th International Conference on Parsing Technologies*.

A. Koller and M. Kuhlmann. 2012. Decomposing tag algorithms using simple algebraizations. In *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+11)*.

M. Lewis, K. Lee, and L. Zettlemoyer. 2016. LSTM CCG parsing. In *Proceedings of NAACL-HLT*.

K. Sima'an, R. Bod, S. Krauwer, and R. Scha. 1994. Efficient disambiguation by means of stochastic tree substitution grammars. In *Proceedings of International Conference on New Methods in Language Processing*.