

A framework for inductive learning of typed-unification grammars

Liviu Ciortuz

CS Department, University of York, UK. E-mail: ciortuz@cs.york.ac.uk**

Abstract. LIGHT, the parsing system for typed-unification grammars [3], was recently extended so to allow the automate learning of attribute/feature paths values. Motivated by the logic design of these grammars [2], the learning strategy we adopted is inspired by Inductive Logic Programming [4]; we proceed by searching through hypothesis spaces generated by logic transformations of the input grammar. Two procedures — one for generalisation, the other for specialisation — are in charge with the creation of these hypothesis spaces.

The work on *typed-unification grammars* can be traced back to the seminal paper on the PATR-II system [6]. Basically, the design of such grammars is underlined by two simple ideas: *i.* context-free rules may be augmented with constraints, generalising the grammar symbols to attribute-value (or: feature) structures; *ii.* feature structures (FSs) may be organised into hierarchies, a very convenient way to describe in a concise manner classes of rules and lexical entries.

Different *logical perspectives* on such a grammar design were largely studied; see [2] for a survey. For the LIGHT system we adopted the Order-Sorted Feature (OSF) constraint logic framework [1].

The *aim* of our work is to explore the usefulness of a logic-based learning approach — namely Inductive Logic Programming (ILP) — to improve the coverage of a given typed-unification grammar: either generalise some (automatically identified) rule or lexical type feature structures (FSs) in the grammar so to make it accept a given sentence/parse tree, or try to specialise a certain type in the grammar so to reject a (wrongly accepted) parse.

The *main idea* of ILP-based learning is to generate (and then evaluate) new clauses starting from those defined in the input grammar. The validation of new “hypotheses” is done using a set of examples and an evaluation function, to rate (and choose among) different hypotheses. The creation of new type hypotheses is done either by generalisation or specialisation. For typed-unification grammars, *generalisation* of FSs amounts to relaxing/removing one or more atomic OSF-constraints, while *specialisation* adds new such constraints to a type FS.

To illustrate a *type feature structure*, Figure 1 presents `satisfy_HPSG_principles`, the parent of the two (binary) rules used in a simple Head-driven Phrase Structure Grammar (HPSG, [5]) appearing in [7]. The `satisfy_HPSG_principles` type encodes three HPSG principles: the Head Feature Principle, the Subcategorization Principle, and the Saturation Principle.³ The knowledge embodied in the `satisfy_HPSG_principles` will be inherited into two rules: `leftHeaded_phrase` and `rightHeaded_phrase`.⁴

The *architecture* of the system we implemented for learning attribute path values in typed-unification grammars is designed in Figure 2. The initial grammar is

** This paper was published in the Proceedings of the International Colloquium on Grammatical Inference (ICGI 2002), Amsterdam, 23–25 September 2002, as LNAI vol. 2484, Springer-Verlag, pages 299–301.

³ The feature constraint encoding of the three principles is respectively: the Head Feature Principle: `satisfy_HPSG_principles.CAT ≐ #1 ≐ satisfy_HPSG_principles.HEAD.CAT`, the Subcategorization Principle: `satisfy_HPSG_principles.HEAD.SUBCAT ≐ #3#2 ≐ satisfy_HPSG_principles.COMP.CAT|SUBCAT`, and the Saturation Principle: `satisfy_HPSG_principles.COMP.SUBCAT ≐ nil`.

⁴ Actually, the constraints specific to these rules only impose that the *head* argument is on the left, respectively the right position inside a phrase PHON feature value.

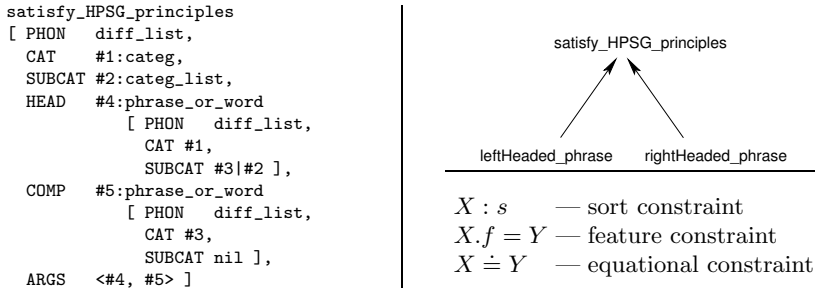


Fig. 1. A sample type feature structure, a simple sort/type hierarchy, and the atomic OSF-constraints for the logical description of feature structures.

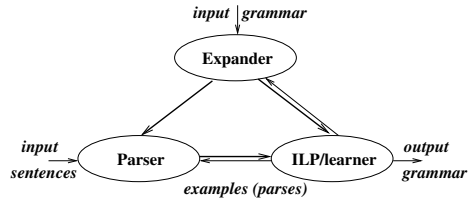


Fig. 2. The *ilp*LIGHT architecture for learning typed-unification grammars.

submitted to an Expander module, which propagates the appropriate constraints down into the type hierarchy. The Parser module uses the expanded form of (syntactic) rules and lexical descriptions to analyse input sentences. The ILP/learner module receives (maybe partial) parses produced by the Parser, and — by calling one of two hypothesis generation procedures — it infers either more specific or more general type descriptions for the considered grammar, such that the new grammar will provide a better coverage of the given sample sentences.

The bidirectional arrows in the diagram in Figure 2 are due to the bi-functionality of the Parser and Expander modules. When asked to act in “reverse” mode, the Parser takes as input a parse and tries to build up the FS associated to that parse. If the construction fails, then the LIGHT’s tracer component is able to deliver an “explanation” for the failure.⁵ This failure “explanation” will be analysed by the ILP/learner module to propose “fixes” to the grammar so that the analysed parse will get accepted. The Expander is also able to work in “reverse” mode: given as input a type name and an atomic constraint (on a certain path value) contained in the description of that type, the expander will indicate which type in the input (unexpanded) grammar is responsible for the introduction of that constraint in the expanded form of the designated type.

We conducted a series of experiments during which the *ilp*LIGHT prototype system was able to learn by specialisation each of the three HPSG principles, if the other two were present and a (rather small) set of sample sentences was provided. Equally, the system learned by specialisation lexical descriptions, and was able to recover (by generalisation) the definition of the HPSG principles if they were provided in an over-restricted form. In all cases mentioned above, learning took place in a few minutes amount of time.⁶

⁵ This explanation assembles informations on: *i.* the parse operation at which the “derivation” of the FS associated to the input parse was blocked; *ii.* the feature path along which the last (tried) unification failed; *iii.* the (atomic) unification steps which lead to the sort clash causing the unification failure.

⁶ For instance, learning the Subcategorization Principle took 4min. 25 sec. on a Pentium III PC at 933MHz running Linux Red Hat 7.1, using the sample HPSG grammar in [6] and a set of 14 training sentences.

As *further work*, we are primarily concerned with the elaboration of linguistics-based heuristics supporting the learning of attribute path values in grammars of significant size, and further improving the efficiency of the *ilp*LIGHT prototype.

Acknowledgements: This work was supported by EPSRC ROPA grant “Machine Learning for Natural Language Processing in a Computational Logic Framework”. The LIGHT system was developed at the Language Technology Lab of the German Research Center for Artificial Intelligence (DFKI), Saarbrücken.

References

1. H. Ait-Kaci and A. Podelski. Towards a meaning of LIFE. *Journal of Logic Programming*, 16:195–234, 1993.
2. B. Carpenter. *The Logic of Typed Feature Structures – with applications to unification grammars, logic programs and constraint resolution*. Cambridge University Press, 1992.
3. L. Ciortuz. LIGHT — a constraint language and compiler system for typed-unification grammars. In *KI-2002: Advances in Artificial Intelligence*, volume 2479, pages 3–17, Berlin, Germany, 2002. Springer-Verlag. Proceedings of the 25th Annual German Conference on AI, Aachen, Germany.
4. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19,20:629–679, 1994.
5. C. Pollard and I. Sag. *Head-driven Phrase Structure Grammar*. CSLI Publications, Stanford, 1994.
6. S. M. Shieber, H. Uszkoreit, F. C. Pereira, J. Robinson, and M. Tyson. The formalism and implementation of PATR-II. In J. Bresnan, editor, *Research on Interactive Acquisition and Use of Knowledge*. SRI International, Menlo Park, Calif., 1983.
7. G. Smolka and R. Treinen, editors. *The DFKI Oz documentation series*. German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, Saarbrücken, Germany, 1996.