34 Two co-transformations of grammars and related transformation rules

Wolfgang Lohmann
University of Rostock, Albert-Einstein-Str. 21, D-18051 Rostock
wlohmann@informatik.uni-rostock.de

34.1 Introduction

Despite of their basic role in software development, grammars are still often considered as static artifacts, used for language definition or once prepared for tool construction. Hence, there is insufficient support for working with grammars as software artifacts themselves. However, grammars are permanently changed, e.g. to debug them, to improve languages, to create grammars for tools. In [1] the authors demonstrate, how transformation tasks can be simplified, when underlying grammars are tailored especially to subtasks. The need for a discipline of grammar engineering has been emphasized by [2].

In the following, we will describe two examples of co-transformations. A co-transformation transforms mutually dependent software artifacts of different kinds simultaneously, while the transformation is centred around a grammar (or schema, API, or a similar structure) that is shared among the artifacts [3]. We are interested in consequences of grammar adaptations to transformations based on these grammars, and investigate, if the transformation rules can be migrated to work with the modified grammar.

34.2 Extending grammar rules

A slight change to a grammar by the insertion of a single non-terminal can lead to corrupt transformation rules, as these are based on the original grammar. While adapting the grammar information about the grammar changes can be collected. It can then be used to derive a migration of transformation rules by adapting the patterns which are based on the abstract syntax of the grammar. Since patterns are extended, it is necessary to define default values for introduced positions. Additionally, changed semantics has to be expressed with additional rules. The method is described more detailed in [4].

The approach has been used to tackle the problem of layout preservation in source-to-source transformations like refactoring. Non-terminals for layout information were introduced in the grammar at each terminal. The rules, for example, those for specifying a refactoring, had to be adapted to work with the extended abstract syntax. For the transport of layout information, a heuristics has been used.

Note, that the approach can also be used to simplify patterns for rewriting rules. Several transformation tasks need parts of the patterns only. The original grammar can be considered to be the extended subgrammar for that pattern. Thus, rules over the subgrammar can be migrated to work on abstract syntax trees according to the original grammar.

34.3 Semantics-preserving left recursion removal

Several tools for source-to-source transformation are based on top-down parsers. Top-down parsers are simple. Their structure is similar to the grammar, and it is easy to add semantics. However, they restrict the user to use grammars without left recursion. Removing left recursion of a given grammar often makes it unreadable, and prevents a rewriter to concentrate on the original grammar. Additionally, the question arises, whether the tool implements the semantics of the original language, if it is implemented based on a different grammar than in the original language definition. Moreover, existing implementations of semantics for the original grammar cannot be reused directly. A co-transformation on attribute grammars can help here.

A grammar and transformation rules on its abstract syntax can be considered as attribute grammar. It is possible to remove left recursion in the grammar and at the same time migrate the semantic rules. In the case of S-attribute grammars, each newly introduced non-terminal gets the synthesized attributes of the original non-terminal as well as inherited attributes of the same type. The computation is redirected to the inherited attribute of the non-terminal following the recursive one in the grammar rule. The ϵ-derivation causes a copy of the result computed so far to the corresponding synthesized attribute, which are then just copied upwards. Similarly, other types of attribute grammars can be adapted, like I-attributed and multi-pass attribute grammars. The approach is explained and justified in detail in [5].

Using the approach, it is possible to use small and easy top-down based tools for simple maintenance task with left recursion containing grammars recovered from YACC specifications, as long as they do not contain ϵ-productions. The approach also contributes to simplifying rewriting rules, since a programmer can continue to use semantic rules on a better readable left recursive grammar.

34.4 Final remarks

The two given examples are grammar adaptations, where much of necessary migrations of the related transformation rules can be derived automatically. The general concept is pictured in Fig. 34.1.

For a real support of grammar changes and related rules there is still much work to be done.

The given grammar adaptations can be applied to make writing transformations easier. The first helps in abstracting away unnecessary complexity when rewriting. The user can concentrate to specify the tasks while the automatic grammar adaptation and migration of rules takes care of the ‘uninteresting‘ parts.

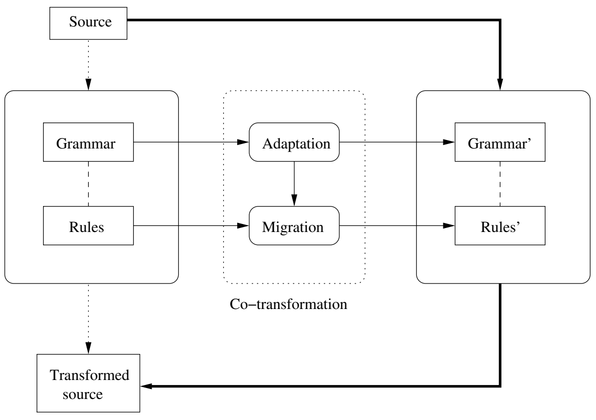

Figure 34.1: Co-transformation: grammars and rules

The second example enables the user to specify program transformations based on a grammar containing left recursion even if he uses top-down parsing technology. Hence, he is freed from using more complex grammar resulting from algorithm of left recursion removal. Moreover, it provides a method to reduce the distance of an original grammar in a specification and some 'tool'-tailored grammar necessary for technical reasons.

We are looking for more patterns, where a modification of the grammar induces a change to transformation rules over programs according to the grammar. A known example is the transformation of attribute grammars to an S-attribute grammar.

Acknowledgement

The work on automatic removal of left recursion while preserving semantics profited from collaboration with Markus Stoy during his diploma thesis.

Bibliography

[1] T.R. Dean, J.R. Cordy, A.J. Malton, and K.A. Schneider. Agile Parsing in TXL. *Journal of Automated Software Engineering*, 10(4):311–336, October 2003.

[2] P. Klint, R. Lämmel, and C. Verhoef. Towards an engineering discipline for grammarware. 32 pages, submitted for journal publication, August17 2003.

[3] Ralf Lämmel. Transformations everywhere. *Science of Computer Programming*, 2004. To appear; The guest editor's introduction to the SCP special issue on program transformation.

[4] Wolfgang Lohmann and Günter Riedewald. Towards automatical migration of transformation rules after grammar extension. In *Proc. of 7th European Conference on Software Maintenance and Reengineering (CSMR'03)*, pages 30–39. IEEE Computer Society Press, March 2003.

[5] Wolfgang Lohmann, Günter Riedewald, and Markus Stoy. Semantics-preserving migration of semantic rules after left recursion removal in attribute grammars. In *Proc. of 4th Workshop on Language Descriptions, Tools and Applications (LDTA 2004)*, 2004.