Adding Concrete Syntax to a Prolog-Based Program Synthesis System (Extended Abstract)

Bernd Fischer[1] and Eelco Visser[2]

[1] RIACS / NASA Ames Research Center, Moffett Field, CA 94035, USA
fisch@email.arc.nasa.gov
[2] Institute of Information and Computing Sciences, Universiteit Utrecht
3508 TB Utrecht, The Netherlands
visser@acm.org

1 Introduction

Program generation and transformation systems work on two language levels, the object-level (i.e., the language of the manipulated programs), and the meta-level (i.e., the implementation language of the system itself). The meta-level representations of object-level program fragments are usually built in an essentially syntax-free fashion using the operations provided by the meta-language. However, syntax matters and a large conceptual distance between the two languages makes it difficult to maintain and extend such systems. Here we describe how an existing Prolog-based system can gradually be retrofitted with concrete object-level syntax using the approach outlined in [5], thus shrinking this distance.

2 AutoBayes

AutoBayes [1] is a fully automatic program synthesis system for statistical data analysis problems like the analysis of planetary nebulae images taken by the Hubble space telescope [2]. Its implementation currently comprises about 80,000 lines of SWI-Prolog code. AutoBayes derives code from statistical models of the data using a schema-based approach. Schemas consist of parameterized code fragments (i.e., templates) and constraints. The code fragments are written in a sanitized variant of C (e.g., no pointers) that also contains a number of domain-specific extensions (e.g., matrix expressions). The fragments represent the solution methods of the domain, for example statistical algorithms like k-means clustering or numeric optimization algorithms like the Nelder-Mead simplex method. The constraints determine whether a schema is applicable and how the parameters – and thus the code – can be instantiated, either directly by the schema or by AutoBayes calling itself recursively with a modified model.

M. Bruynooghe (Ed.): LOPSTR 2004, LNCS 3018, pp. 56–58, 2004.

3 Program Generation in Prolog

In the current AutoBayes-version, schemas are implemented as simple Prolog-clauses that return a term representing the appropriately instantiated algorithms. The schemas assemble the fragments from many small code pieces that are bound to Prolog-variables acting as meta-variables; otherwise, the abstract syntax would become unreadable. The schemas are also sprinkled with many calls to small meta-programming predicates that for example generate fresh object-level variables or build other constructs of the object language. A particular nuisance is the repeated use of Prolog's built-in `=..`-operator to construct compound object-level terms with variables as head symbols, which is necessary for second-order term formation. This general style makes it hard to follow and understand the overall structure of the algorithm and thus difficult for a domain expert to modify and write schemas.

4 Migration to Concrete Syntax

Schema migration involves three easy steps. First, terms representing program fragments in abstract syntax are replaced by the equivalent fragments in concrete syntax; these fragments are marked by a quotation operator. Then, calls to meta-programming predicates are replaced by the appropriate object-level constructs. For example, the explicit generation of fresh object variables is expressed *in the object code* by tagging the corresponding meta-variable with a special anti-quotation operator. Finally, the schema is refactored by inlining the program fragments.

The necessary extension of Prolog with concrete syntax relies on the combination of three techniques. (*i*) The syntax definition formalism SDF2 [4] is used to specify the syntax of both languages as well as the embedding (i.e., quotation mechanism and meta-variables). (*ii*) The transformation language Stratego [6] is used to map syntax trees over this combined language back into pure Prolog programs. This eliminates the concrete syntax. (*iii*) Stratego is also used to adapt the resulting schemas into the exact form required to interface them with the remaining AutoBayes-system. For example, second-order term formations and fresh variable anti-quotations are hoisted out of the resulting abstract syntax terms and turned back into the original constructors and fresh variable generators provided by the meta-programming kernel.

5 Conclusions

We applied the generalized approach to meta-programming with concrete object-syntax to some of the schemas in AutoBayes. In these cases, the introduction of concrete syntax reduces the schema size by about 30% and improves the readability and locality of the schemas. In particular, abstracting out second-order term formation and fresh-variable generation allows the formulation of larger continuous fragments. Moreover, meta-programming with concrete syntax is cheap:

using Stratego and SDF2, the overall effort to develop all supporting tools was less than three weeks, Once the tools were in place, the migration of a schema was a matter of a few hours. Finally, the experiment has also demonstrated that it is possible to introduce concrete syntax support gradually, without forcing a disruptive migration of the entire system to the extended meta-language. The seamless integration with the "legacy" meta-programming kernel is achieved with a few additional transformations, which can be implemented quickly in Stratego.

Acknowledgments

A preliminary version of this paper was presented at LOPSTR 2003; a full version appears as [3]. We thank all reviewers for their comments.

References

1. B. Fischer and J. Schumann. AutoBayes: A system for generating data analysis programs from statistical models. *JFP*, 13(3):483–508, 2003.
2. B. Fischer and J. Schumann. Applying AutoBayes to the analysis of planetary nebulae images. In *ASE-18*, pp. 337–342. IEEE Comp. Soc. Press, 2003.
3. B. Fischer and E. Visser. Retrofitting the AutoBayes Program Synthesis System with Concrete Syntax. Tech. Report UU-CS-2004-012, Utrecht Univ. In [7].
4. E. Visser. *Syntax Definition for Language Prototyping*. PhD thesis, Univ. Amsterdam, 1997.
5. E. Visser. Meta-Programming with Concrete Object Syntax. In *Generative Programming and Component Engineering*, *LNCS 2487*, pp. 299–315. Springer, 2002.
6. E. Visser. Program transformation with Stratego/XT: rules, strategies, tools, and systems in StrategoXT-0.9. Tech. Report UU-CS-2004-011, Utrecht Univ. In [7].
7. *Domain-Specific Program Generation*. Springer, 2004. To appear.