33 Evolution of Language Interpreters

Ralf Lämmel
Vrije Universiteit, De Boelelaan 1081a, NL-1081 HV Amsterdam
CWI, Kruislaan 413, NL-1098 SJ Amsterdam
ralf@cwi.nl

Keywords Operational Semantics, Evolution, Re-Engineering, Program Transformation, Language Interpreters, Language-Based Tools

Language-based tools involve some (ad-hoc) elements of a language's intended (say, formal) semantics. These elements relate to the specific service provided by a tool. For instance, a program analyser might implement a so-called abstract interpretation; a transformation tool might employ algebraic laws as well as typing and scoping rules; a translator implements the syntax-to-syntax edge in the commuting diagram with nodes for the two syntaxes and the two semantics of the involved languages.

Type checkers and language interpreters can be seen as prototypical examples of language-based tools. Such tools implement the language semantics rather directly. The extensibility of such semantics-oriented programs (and the underlying formal semantics descriptions) has received ample attention in the programming-language community. There exist highly advanced approaches to the reuse of language descriptions or components thereof, e.g., monadic-style semantics, action semantics, abstract state machines, strategic programming, modular SOS, and modular attribute grammars. This suggests that the domain of language descriptions is suitable for studying evolution of language-based tools in general.

We aim at a pragmatic, re-engineering-like approach to the evolution of language-based tools. That is, we provide a simple, transformation-based approach to the evolution of language-based tools that are encoded in simple rule-based notations. Our focus is on interpreters. We assume that language interpreters (or other language-based tools) are programmed in rule-based notations such as structural operational semantics, definite clause grammars, attribute grammars, conditional rewrite systems, or constructive algebraic specifications. The evolution of an interpreter is then represented by meta-programs on such rule-based object programs, while relying on a suitably designed operator suite for evolutionary transformations.

We present a series of examples for interpreter evolution. Starting from a simple expression language, we go through a development that touches upon pure and impure extensions, higher-order functional, object-oriented languages, and aspect-oriented languages, as well as semantics of different styles, i.e., small-step vs. big-step SOS.

Pointers

This approach is presented at some length in [1]. (An early instance of evolutionary transformations for rule-based programs is developed in the author's PhD thesis.) One specific proof-of-concept implementation of the approach is available as the Rule Evolution Kit [2].

Bibliography

[1] R. Lämmel. Evolution of Rule-Based Programs. *Journal of Logic and Algebraic Programming*, 73, 2004. 52 pages; Special Issue on Structural Operational Semantics; To appear.

[2] The Rule Evolution Kit, version 0.77, 11 Feb. 2004. http://www.cs.vu.nl/rek/.