Tool Demonstration: Silver Extensible Compiler Frameworks and Modular Language Extensions for Java and C[*]

Eric Van Wyk, Lijesh Krishnan, Derek Bodin, Eric Johnson, August Schwerdfeger, Phil Russell
Department of Computer Science and Engineering
University of Minnesota

In this tool demonstration of Silver extensible compiler frameworks for Java and C we illustrate how new languages that are adapted to specific problem domains can be easily created, by their users, by importing a set of desired domain-specific language extensions into an extensible host language compiler. Language extensions for computational geometry and database access will be shown. We also show extensions that provide general purpose language features such as algebraic types and pattern matching can be imported into an extensible language compiler.

Each Silver extensible compiler framework supports the development of language extensions that have two important facets. First, language extensions should satisfy a *completeness* requirement. That is, they should be as well-developed as host language features and fit seamlessly into the host language. In particular, the language feature designer should be able to specify new language constructs together with their domain-specific semantic analyses and techniques for their optimization. One aspect of this requirement is that language extension should report a useful error message when they are used incorrectly. Second, the extensions should be *modular* so that a programmer can extend his or her language by choosing from a collection of previously defined features knowing only the functionality they provide and with no implementation-level knowledge or a detailed analysis of their interactions. Thus we draw a distinction between the programmer importing an extension and the feature designer who implements it.

We will show extensible compilers for both C and Java. These compilers are defined by an attribute grammar written in the Silver attribute grammar language. A Silver compiler analyses attribute grammar specifications and generates an executable compiler for the defined (extended) language by translating the Silver specifications into an efficient Haskell representation. Language extensions are also specified as Silver attribute grammar fragments and the framework tools automatically compose specifications of the host language and chosen language extensions into a specification for the custom extended language.

We have built several modular language extensions that extend either Java or C in this framework and will demonstrate aspects of the implementation of these extensions performed by the domain-expert feature designer. We will also show the process of using the framework to automatically compose language features performed by a programmer.

Two domain-specific language extensions highlight the analysis (via error checking) and optimization capabilities of the frameworks. An extension that embeds SQL into Java so that queries can be written directly, as opposed to creating character strings containing the SQL queries as is done in library-based approaches. This extension checks at compiler-time that no syntax errors and no type errors have been made. A second domain-specific extension, written to extend C, introduces optimizing program transformations for unbounded numeric types and perturbation transformations for coping with data degeneracies that are specific to the domain of computational geometry. We have also created a number of modular language extensions that add new general purpose features to Java. One such extension adds algebraic data types and pattern matching similar to those found in ML and Pizza [1]. These extensions make use of an enhancement to attribute grammars called *forwarding* [2]. This provides the modularity that allows programmers, not language designers, to construct new custom extended languages from a host language and a set of modular language extensions.

Silver and the C, Java, and extension specifications are available at `http://www.melt.cs.umn.edu`.

References

[1] M. Odersky and P. Wadler. Pizza into Java: translating theory into practice. In *Proc. of ACM Symposium on Principles of Programming Languages (POPL)*, pages 146–159, 1997.

[2] E. Van Wyk, O. de Moor, K. Backhouse, and P. Kwiatkowski. Forwarding in attribute grammars for modular language design. In *Proc. 11th Intl. Conf. on Compiler Construction*, volume 2304 of *LNCS*, pages 128–142. Springer-Verlag, 2002.

[*]This work was partially funded by NSF CAREER Award #0347860, NSF CCF Award #0429640, and the McKnight Foundation.