Combining Tools and Languages for Source-Based Static Analysis and Optimization of High-Level Abstractions

Markus Schordan
Institute of Computer Languages
Vienna University of Technology, Austria
markus@complang.tuwien.ac.at

Currently several dozens of static-analysis tools are readily available for software development. Static-analysis tools exist for most common programming languages, though the majority of tools support some subset of C, C++, or Java. A comparison of these tools shows that the number of accepted languages is small, though many differences exist in the range of language constructs that are fully supported. The analysis results are difficult to compare because the they are reported in some proprietary format. For that reason analysis results from different tools usually cannot be combined with reasonable effort either.

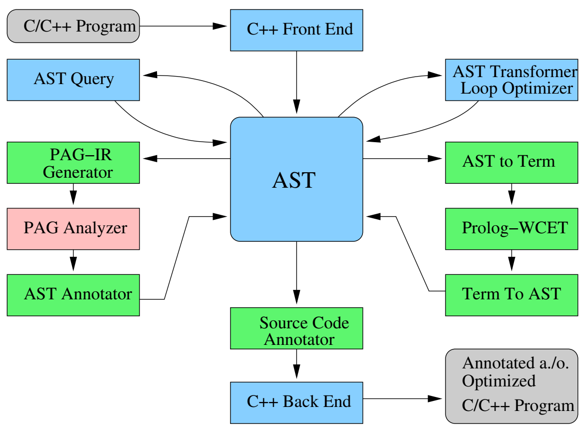

Figure 1. Infrastructure: Composed of LLNL-ROSE (blue), PAG (red), SATIrE (green)

We have crafted an infrastructure by combining existing tools and languages for providing a solid basis for a compact specification of different program analyses for industrial applications (see Fig. 1). The present state is that we have achieved the first stage of supporting one popular language family, C/C++, and that compact program analysis specifications are possible by using the Program Analyzer Generator (PAG) [1] from AbsInt. Program Transformations can be performed as source-to-source transformation by using the LLNL-ROSE infrastructure [2], developed at Lawrence Livermore National Laboratory. The integration of PAG into LLNL-ROSE is automated by the Static Analysis Integration Engine (SATIrE), developed at TU Vienna. Furthermore we also provide a connection to Prolog which allows the specification of program analyses and transformations as logic programs, currently focusing on worst case execution time (WCET) analysis and annotation.

The talk presents the challenges and solutions of integrating the framework LLNL-ROSE, the analyzer generator PAG, and different program representations of C/C++, for allowing compact analysis specification, source-to-source optimization, generating external formats, and automating source code annotation. We also describe the interfaces that we can offer for integration with other frameworks. Eventually results are presented for some C/C++ analyses that we have performed with our infrastructure.

References

[1] F. Martin. PAG – an efficient program analyzer generator. *International Journal on Software Tools for Technology Transfer*, 2(1):46–67, 1998.

[2] M. Schordan and D. Quinlan. Specifying transformation sequences as computation on program fragments with an abstract attribute grammar. In *Proceedings of the Fifth IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'05)*, pages 97–106. IEEE Computer Society Press, 2005.