Scala to SIL

Master's Thesis Project Description

Bernhard F. Brodowsky

December 18, 2012

Introduction

Chalice [3] is a programming language which has been created for research on automatic program verification for concurrency. In order to verify a Chalice program, it is translated into the intermediate languages Boogie [2] or SIL and then a verifier generates verification conditions from that intermediate code which are then passed to an SMT solver such as Z3 [1]. Building on the experience we gained with Chalice, we now want to translate Scala [6] to SIL, a modern blend of object oriented programming and functional programming with a powerful type system and support for the actor concurrency model. There exists a previous project by Rokas Matulis [4] which adds code contracts to Scala as a library and injects code during compile time to check them at runtime. The API of the contracts of that project should be reused for static checking.

Goal

The goal of this master's thesis is to develop a program that translates a subset of Scala code that has been enriched with the aforementioned code contracts into SIL code, which can then be passed on to a verifier. The translator will be implemented as a Scala compiler plugin. The Scala code should be translated to the intermediate language SIL and from there on, the verification is equivalent to the verification of a Chalice program, so ideally, the existing verifier does not have to be changed. Supporting all Scala language features is beyond the scope of a master's thesis, but at least the Chalice permission model [3], classes, methods (except higher order ones), an axiomatisation of the core parts of the type system (excluding existential types), constructors, loop invariants, predicates, variable and field assignments and termination checking for pure functions without loops should be supported.

Extensions

Possible extensions include actors, fork-join concurrency, threads, inheritance, more general termination checking, and checking that methods that are annotated as pure are actually pure. Another possible extension would be to support the higher order functions map, filter, foreach, forall, exists and find. Such functions are in general very hard to specify and verify [5], but are also very frequently used in Scala programs.

References

[1] Leonardo De Moura and Nikolaj Bjørner. Z3: an efficient smt solver. In *Proceedings of the Theory and practice of software, 14th international conference on Tools and algorithms for the construction and analysis of systems*, TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.

[2] K. Rustan M. Leino. This is Boogie 2. Manuscript KRML 178, 2008. Available at http://research.microsoft.com/ leino/papers.html.

[3] K. Rustan M. Leino and Peter Müller. A basis for verifying multi-threaded programs. In Giuseppe Castagna, editor, *Programming Languages and Systems, 18th European Symposium on Programming, ESOP 2009*, volume 5502 of *Lecture Notes in Computer Science*, pages 378–393, Berlin, March 2009. Springer-Verlag.

[4] Rokas Matulis. Extensible code contracts for scala. Master's thesis, ETH Zurich, September 2012.

[5] A. Nanevski, G. Morrisett, and L. Birkedal. Hoare Type Theory, polymorphism and separation. *Journal of Functional Programming*, 18(5-6):865–911, 2008.

[6] Martin Odersky, Philippe Altherr, Vincent Cremet, Burak Emir, Stéphane Micheloud, Nikolay Mihaylov, Michel Schinz, Erik Stenman, and Matthias Zenger. An introduction to Scala. August 2005.