Verifying Scala’s Vals and Lazy Vals

Problem Description

Simon Fritsche

March 25, 2014

Introduction:

Scala’s vals are a special kind of variables: Immutable variables. Such immutable variables are nice to have because it can make code more understandable and can help to program in a functional style. Additionally one can declare a val as lazy, which will delay the evaluation of the initializer until the use of the lazy val. But only the evaluation is delayed since both vals and lazy vals have to be initialized at the point of declaration.
Reasoning about these two kinds of variables can be very difficult due to an arbitrarily complex initializer (reading uninitialized fields, side effects, reading themselves, method calls).
We want to verify Scala programs by translating them to Silver, an Intermediate Verification Language. The resulting Silver program can then be verified by a Silver verifier such as Carbon or Silicon, or it can be passed on to any other Silver backend. A prototype for Scala to Silver translation has already been developed[1].
This thesis aims partly at improving the translation of lazy vals and vals. A current limitation is, for example, the fact that only one method call may occur in the initializer of a lazy val, and additionally that this method call must be the outermost expression. This restriction is there to simplify the inference of the field invariant (the invariant that holds for a val after the initialization).
Part of this thesis is also an empirical study where some substantial Scala projects (Sbt, Akka, Lift, …) will be analyzed to determine the use cases of lazy vals in real code, and the gained results are going to be used to decide how strongly current limitations would affect the verification of real code.
Another feature of Scala, for which a translation to Silver is currently missing are singleton objects[2]. These objects are similar to Java/C#’s static classes. Since the whole singleton object gets initialized on use, this is another lazy feature of Scala and we intend to verify them by building on the same approach we have chosen for verifying lazy vals.

Main Goals:

- Make the Scala to Silver translator work with the newest Silver and Scala version

- Find out how lazy vals are used in practice in order to evaluate which restrictions can be kept without excluding too many actual use cases
- Improve the translation of lazy vals to overcome limitations
- Introduce the verification of singleton objects[2]
- Add new tests to the test bench that are inspired by use cases found during the empirical study

Possible Extensions:

Depending on the progress of the Bachelor's thesis some of the following tasks will be tackled:

- Investigate the verification of Scala's Call-By-Name[3] arguments and find out if they correspond to full-fledged closures, whose verification will probably be outside of the scope of this project, or if they can be handled in a way similar to how we handle lazy vals
- Further continue the empirical study and possibly find out more about the uses of Scala's lazy features by going more into the details or looking at additional projects
- Improve on the specification and translation of vals, such that no explicit read permissions are necessary to access a val that has already been initialized, which is a counter-intuitive limitation of the current solution

References:
[1] Bernhard Brodowsky. Scala to SIL, Master's thesis, ETH Zurich
[2] Programming in Scala, first edition, Chapter 4.3 from Martin Odersky
[3] Programming in Scala, first edition, Chapter 9.5 from Martin Odersky