Software Engineering

WS 2022/23, Assignment 03

Prof. Dr. Sven Apel
Annabelle Bergum
Sebastian Böhm
Christian Hechtl

Handout: 21.12.2022
Handin: 11.01.2023 23:59 CET

Organizational Section:

- The assignment must be accomplished by yourself. You are not allowed to collaborate with anyone. Plagiarism leads to failing the assignment.
- The deadline for the submission is fixed. A late submission leads to a desk reject of the assignment.
- We provide a project skeleton that must be used for the assignment. The skeleton can be downloaded from the CMS.
- The submission must consist of a *ZIP* archive containing only the project skeleton folder (i.e., the folder included in the provided skeleton)
- Any violation of the submission format rules leads to a desk reject of the assignment.
- Questions regarding the assignment can be asked in the forum or during tutorial sessions. Please do not share any parts that are specific to your solution, as we will have to count that as attempted plagiarism.
- If you encounter any technical issues, inform us immediately.

Task 1 [25 Points]

Your task is to implement type checkers for the simple programming language presented in the lecture and its type-system, both for the version without variability and the version with variability. The implementation has to be done in Scala[1] using the project skeleton provided on the CMS. The assignment is divided into the following two subtasks:

a) Implement a type checker for the simple programming language *without variability* presented in the lecture and its type-system (slides 70/71) in the file `SimpleTypeChecker.scala`. [10 Points]

b) Implement a type checker for the simple programming language *with variability* presented in the lecture and its type-system (slides 72-74) in the file `VariabilityTypeChecker.scala`. [15 Points]

Grading Your submission will be graded based on the following criteria:

- Your submission must be in the correct format, i.e., a *ZIP* archive with the same layout as the project skeleton (may result in 0 points if violated).
- Your submission must compile, i.e., the command `sbt compile` must succeed (may result in 0 points if violated).
- We run unit tests (the ones provided + additional tests) against your submission. Points will be deducted for each failing test.

[1] https://www.scala-lang.org/

Project Skeleton You must implement your solution based on the provided project skeleton. The skeleton has the following structure:

```
│ build.sbt
├─ project
│  ├─ build.properties
│  └─ plugins.sbt
└─ src
   ├─ main
   │  └─ scala
   │     ├─ Expression.scala
   │     ├─ TypeChecking.scala
   │     ├─ SimpleTypeChecker.scala
   │     └─ VariabilityTypeChecker.scala
   └─ tests
      └─ scala
         └─ *Test.scala
```

The file `Expression.scala` contains implementations of the programming language constructs and types. The file `TypeChecking.scala` contains the general interface for the type checkers. In addition, we provide an implementation of the type context for task a). This class is also reused for task b) for which we also provide an implementation of the variability context. The project skeleton also contains documentation for all provided classes and functions. You are not allowed to modify these provided classes and functions. The locations where you should add your implementation are marked with a `TODO`.

We also provide some basic tests which can be found in the folder `tests`. The file `build.sbt` and the files in the folder `project` contain a build script that we use to build your submission and run the tests.[2] You must not edit these files.

The build script should also enable you to import the project skeleton into any IDE with Scala support (e.g., IntelliJ with the Scala plugin). Scala can be quite sensitive regarding the used language version and things might break if you use the wrong one. For the assignment, we use *Scala 3.2.1* which is also specified in the build script. You can also run the tests included with the project skeleton using the build script. To run all tests execute the command `sbt test`.

Type Checker Result The type checker result is represented by the interface `TypeCheckResult` and its implementations `Success` and `Failure`. In the case of a successful type check an instance of `Success` has to be returned with the inferred type as its parameter. If the type checker detects a type error, it has to return an instance of `Failure` with the (sub-) expression that is currently checked, the context object, and an optional message describing the type error as parameters. This information is used by our tests to determine whether your implementation detected the error correctly. The message is not checked in the tests but can help you during debugging.

Note for advanced users: The result type supports monad operations meaning it can be used in for-comprehensions.[3]

SAT Solver For some type rules of task b), you will need a SAT solver to check certain preconditions. For this assignment, we use CafeSAT[4] which is a simple SAT solver implemented in Scala.

The project skeleton already imports all the relevant packages. Checking whether a formula is satisfiable is as simple as this (assuming `a, b, c` are formulas):

```scala
Solver.solveForSatisfiability((!a || b) && c) match {
  case None => print("Not satisfiable");
  case Some(_) => print("Satisfiable")
}
```

Note how you can use `!`, `||`, and `&&` for *not*, *or*, and *and*. The API does not support implication so you have to encode implication using the other operators. The primitives *true* and *false* are available as `Formulas.True` and `Formulas.False`. Mind the prepended `Formulas` as, otherwise, you might confuse them with the `True` and `False` literals from our programming language!

[2]https://www.scala-sbt.org/1.x/docs/
[3]https://docs.scala-lang.org/tour/for-comprehensions.html
[4]https://github.com/regb/cafesat