Software Engineering

WS 2023/24, Assignment 03

Prof. Dr. Sven Apel
Lukas Abelt
Sebastian Böhm

Handout: 09.12.2024
Handin: 06.01.2024 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 use of generative AI tool for solving the assignment is not allowed and will be considered as plagiarism.
- The deadline for the submission is fixed. A late submission leads to a desk reject of the assignment.
- You must use the provided project skeleton for the assignment. The skeleton can be downloaded from the CMS.
- The submission must consist of a *ZIP* archive following the layout of the project 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 via email. Please do not share any parts that are specific to your solution in the forum, 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 a type checker 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 Kotlin[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 (chapter 5 slides 109/110) in the file `SimpleTypeChecker.kt`. [10 Points]

b) Implement a type checker for the simple programming language *with variability* presented in the lecture and its type-system (chapter 5 slides 111/112) in the file `VariabilityTypeChecker.kt`. [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 `./gradlew assemble` (`./gradlew.bat assemble` on Windows) must succeed (may result in 0 points if violated).
- We run unit tests (the ones provided + additional tests) against your submission. Points will be awarded based on passed tests.
- You must not modify provided code in any way since this might lead to the project not building with the tests during evaluation. However, you are allowed (and encouraged) to add additional helper functions and classes.

[1] `https://kotlinlang.org/`

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

```
gradle/
lib/
src
  main
    kotlin
      expression
        Expression.kt
      vexpression
        VExpression.kt .4 solver/
      SimpleTypeChecker.kt
      TypeChecking.kt
      VariabilityTypeChecker.kt
  tests
    kotlin
      SimpleTypeCheckerTest.kt
      VariabilityTypeCheckerTest.kt
build.gradle.kts
gradlew
gradlew.bat
settings.gradle.kts
```

The files `Expression.kt` and `VExpression.kt` contain implementations of the programming language constructs and types. Be careful when importing from these files because they contain classes with the same names. Always double-check if you are using the correct version for each subtask! The file `TypeChecking.kt` 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. 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.gradle.kts` contains a build script that we use to build your submission and run the tests.[2] The `lib` folder contains the SAT-solver library used for task b). You must not edit these files.

The build script should also enable you to import the project skeleton into any IDE with Kotlin support. You can also run the tests included with the project skeleton using the build script. To run all tests execute the command `./gradlew test` (or `./gradlew.bat test` on Windows).

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. The type checker should traverse the expression depth-first from left to right and return the first failure it encounters.

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[3] which is a simple SAT solver implemented in Scala.

To simplify the use of this solver with Kotlin, we provide some wrapper classes in the `solver` package. Checking whether a formula is satisfiable is as simple as this (assuming `a, b, c` are formulas):

```kotlin
if (Solver.isSatisfiable((!a or b) and c)) {
  print("Satisfiable")
} else {
  println("Not satisfiable");
}
```

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

[2] https://gradle.org/
[3] https://github.com/regb/cafesat