Prof. Dr. O. Nierstrasz, Dr. Mohammad Ghafari
Manuel Leuenberger, Rathesan Iyadurai

Compiler Construction 2019

Semester project

3 Java byte code generation and optimization

- Implement optimization techniques on your intermediate representation of Mini Java. The more, the better.
- Implement a code generation unit that transforms your intermediate representation to Java bytecode. For the generation of Java byte code use the bcel library (`http://commons.apache.org/proper/commons-bcel/`).

  Hint: Use the `javap` tool to look at how the Java compiler compiles sources to bytecode. See `javap --help` for all available options. Example showing disassembled bytecode for Fibonacci:

  ```
  cd test-src/ch/unibe/scg/minijava/programs
  cp Fibonacci.minijava Fibonacci.java
  javac Fibonacci.java
  javap -c Fib.class
  ```

- The provided test suite specifies the code generation rules. Use it to ensure that your implementation works as expected and to get a feel for the number of features you have implemented. We will use these test cases to evaluate your code generation unit. However, we may also use additional tests that check additional scenarios, so you may want to implement your own additional test cases. You are not allowed to modify the existing test cases.
- Import the provided test suite into your eclipse project.
- You have time until 24 May 2019, 10:00 to complete this part of the project.
- Push your solution (the Eclipse project, including the parts provided by us) into the your Bitbucket repository.
- You are not allowed to use other peoples work. All parts of the project solution must be your own work.

3.1 Best Bytecode Generator Contest

Your optimization techniques will be evaluated from the bytecode size point of view. The best team (teams?) will be awarded with a small reward.

NB: Make sure you properly use the method `addMethod(ClassGen cg, MethodGen mg)`*, which counts the number of bytecodes.*