Results of all tests

In this file the results of all tests for *Pickle Cannon* language compiler are presented.

# Syntax tests results

Syntax tests check the correctness of scanning and parsing. These tests do not generate any Sprockell code as they are meant to only test the correctness of scanning and parsing. Tests are divided by the feature and most of them are simple strings of short program code. At the end there are few tests that check the correctness of the programs written in files.

## Basic types tests

In the Figure 1 the basic type tests can be seen. *accepts* method passes the test if the scanning and parsing was successful, otherwise fails. *rejects* method passes the test if the scanning or parsing was unsuccessful, otherwise fails. Lastly, there are tests that check the correctness of parse tree. These check if concrete children nodes of the parse tree contain the expected information. The errors thrown for the basic types tests can be observed in the Figure 2.

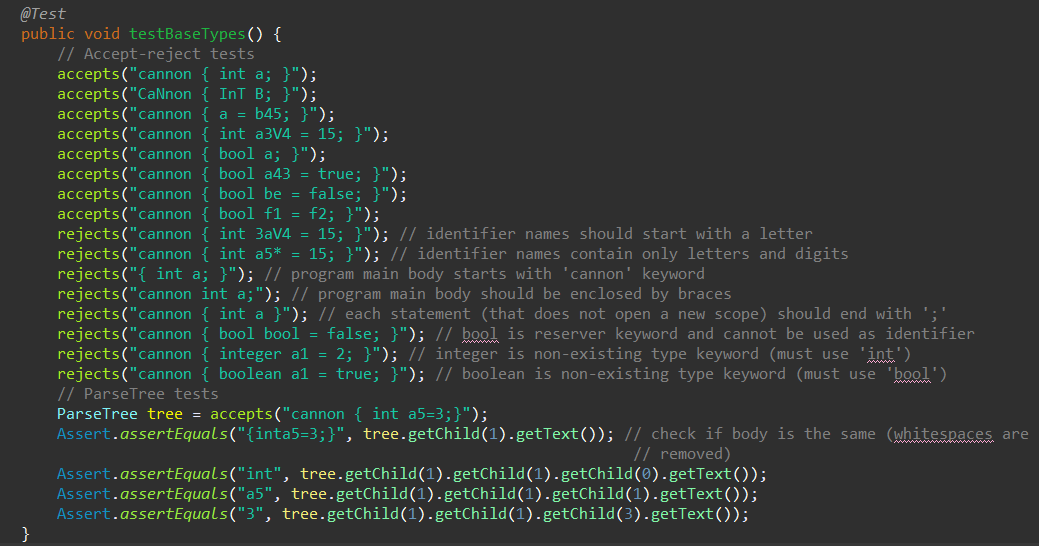


Figure 1. Basic types syntax tests

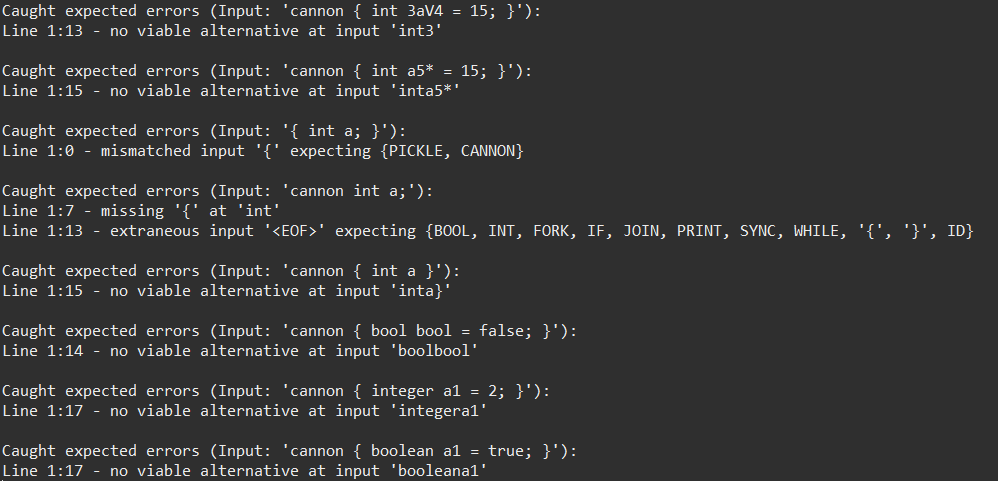


Figure 2. Basic types syntax test caught errors

## Expressions tests

In the Figure 3 the expressions tests can be seen. These test works the same as previous. *accepts* method should accept the program, *rejects* reject the program and parse tree tests should contain the expected information. The errors thrown by the *rejects* method can be seen in the Figure 4.

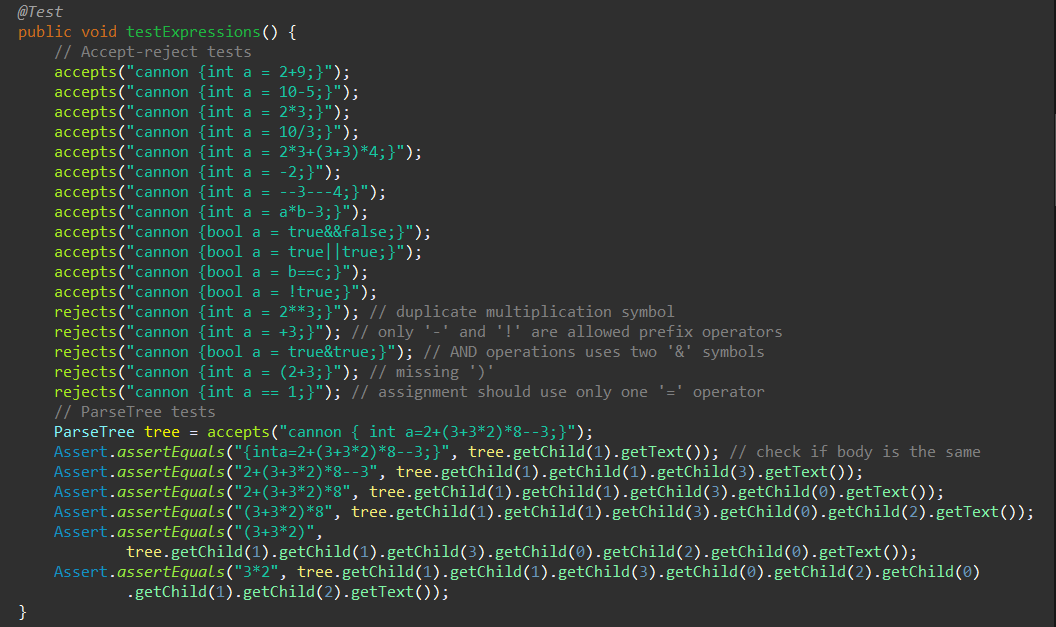


Figure 3. Expressions syntax tests

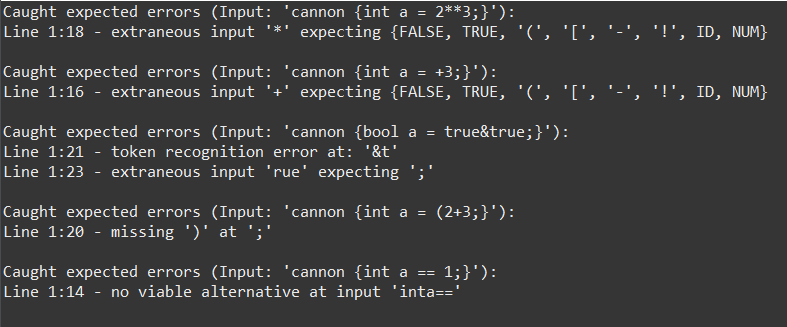


Figure 4. Expressions tests caught expected errors

## While, if and scopes tests

In the Figure 5 while, if and scopes syntax tests can be seen. These test works the same as previous. *accepts* method should accept the program, *rejects* reject the program and parse tree tests should contain the expected information. The errors thrown by the *rejects* method can be seen in the Figure 6.

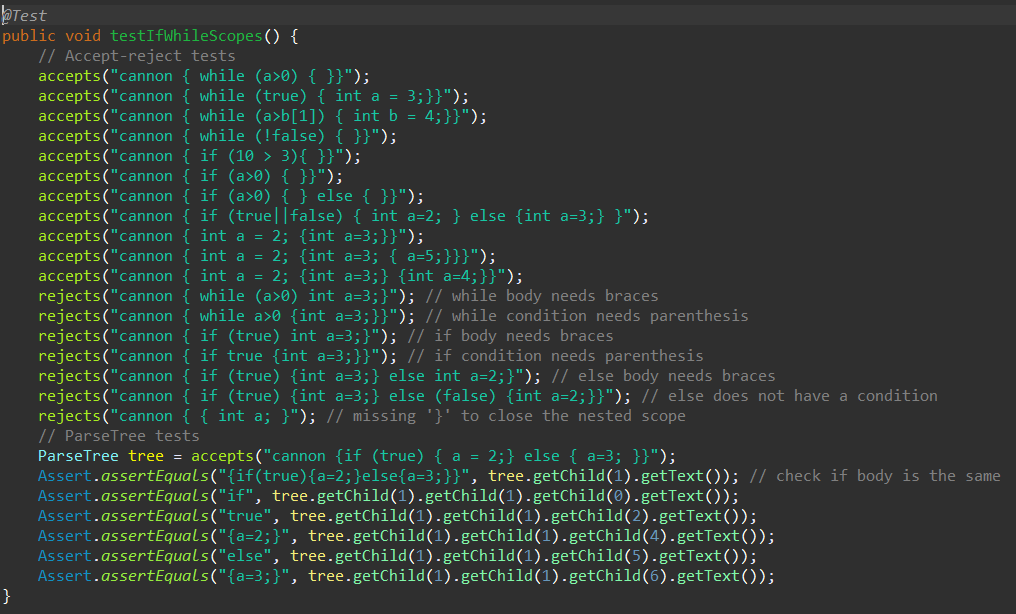


Figure 5. While, if and scopes syntax tests

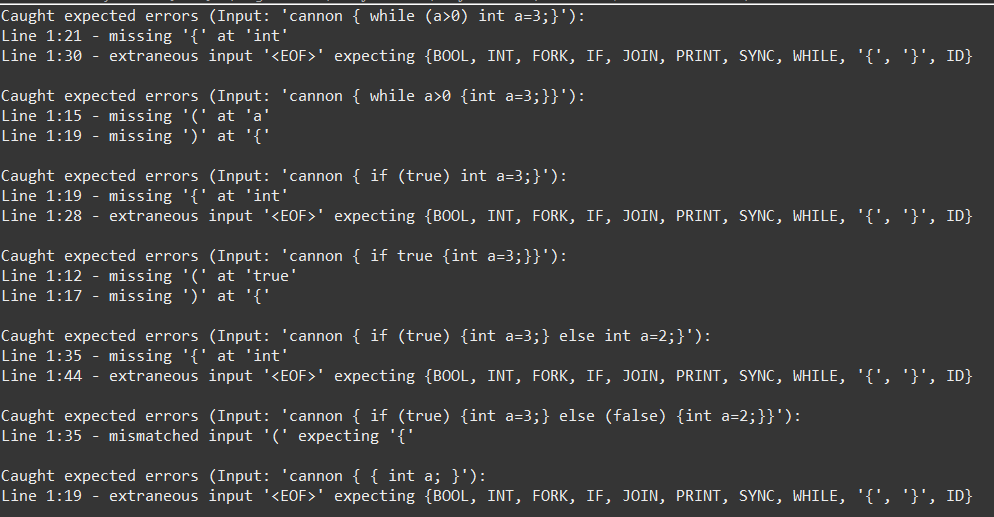


Figure 6. While, if and scopes tests caught expected errors

## Concurrency tests

In the Figure 7 concurrency tests can be seen. These test works the same as previous. *accepts* method should accept the program, *rejects* reject the program and parse tree tests should contain the expected information. The errors thrown by the *rejects* method can be seen in the Figure 8.



Figure 7. Concurrency syntax tests

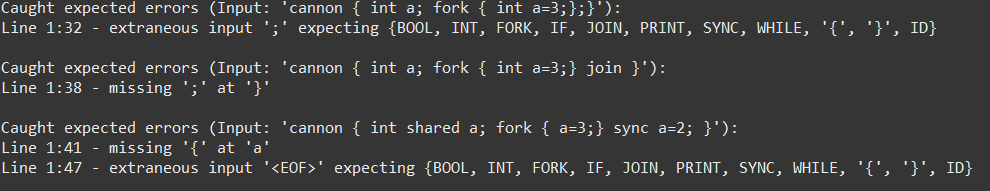


Figure 8. Concurrency tests caught expected errors

## Arrays tests

In the Figure 9 arrays tests can be seen. These test works the same as previous. *accepts* method should accept the program, *rejects* reject the program and parse tree tests should contain the expected information. The errors thrown by the *rejects* method can be seen in the Figure 10.



Figure 9. Arrays syntax tests

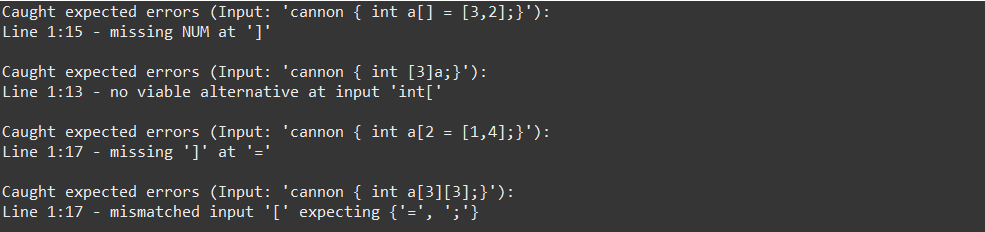


Figure 10. Arrays tests caught expected errors

## Procedures tests

In the Figure 11 procedures syntax tests can be seen. These test works the same as previous. *accepts* method should accept the program, *rejects* reject the program and parse tree tests should contain the expected information. The errors thrown by the *rejects* method can be seen in the Figure 12.

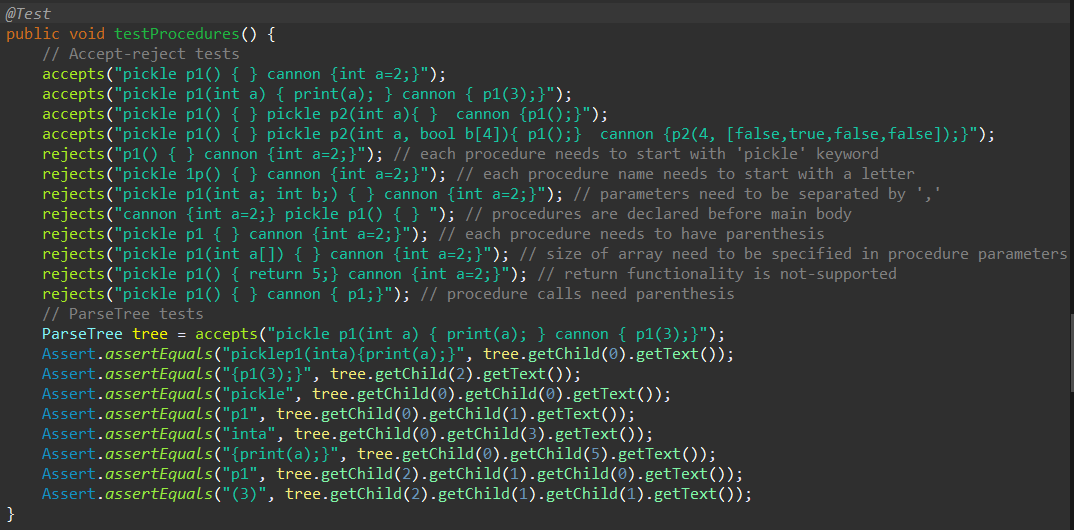


Figure 11. Procedures syntax tests

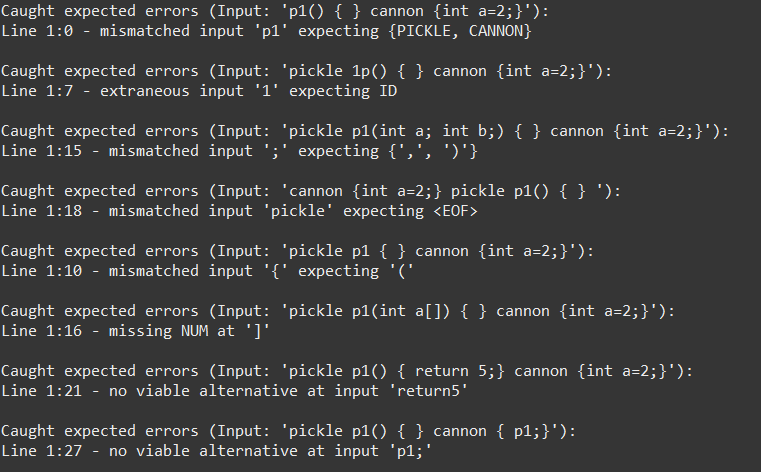


Figure 12. Procedures tests caught expected errors

## File tests

In the Figure 13 tests for programs written in files can be seen. Program source codes can be seen in the Figure 14, Figure 15 and Figure 16. These test works the same as previous. *accepts* method should accept the program, *rejects* reject the program and parse tree tests should contain the expected information. The errors thrown by the *rejects* method can be seen in the Figure 17.

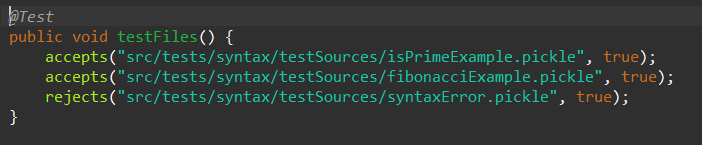


Figure 13. Syntax tests for programs written in files

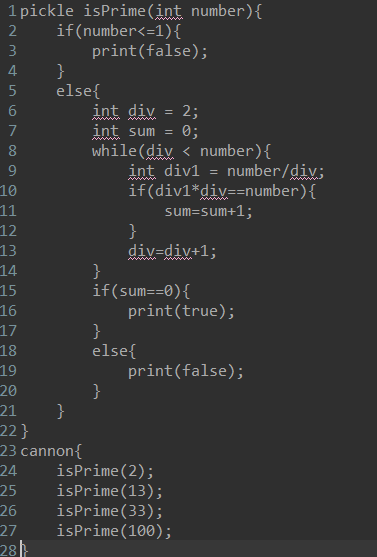


Figure 14. isPrimeExample.pickle source code

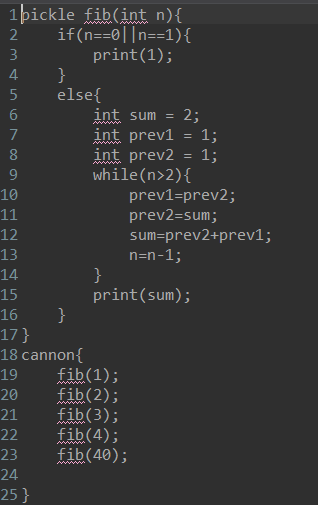


Figure 15. fibonacciExample.pickle source code

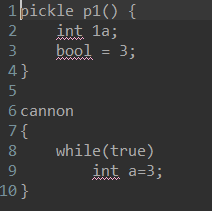


Figure 16. syntaxError.pickle source code

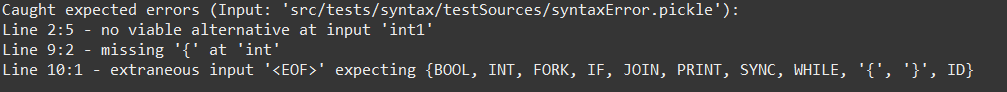


Figure 17. Caught expected error for program written in file

## JUnit results

As can be seen in the Figure 18, all tests pass.

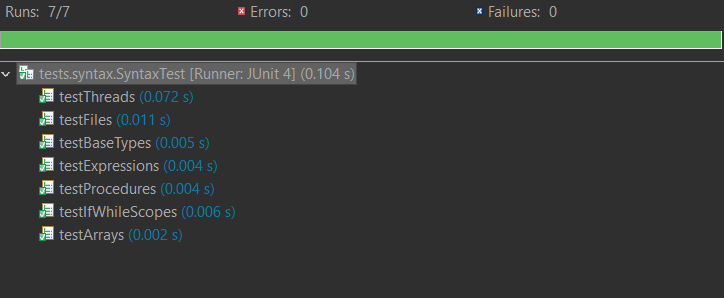


Figure 18. Syntax tests Junit results

# Context tests results

Context tests check the correctness of compiler elaboration. These tests do not generate any Sprockell code as they are meant to only test the correctness of elaboration. Tests are divided by the feature and most of them are simple strings of short program code. At the end there are few tests that check the correctness of the programs written in files.

## Basic types tests

In the Figure 19 basic types context tests can be seen. These test works the same as previous. *accepts* method should accept the program, *rejects* reject the program. The errors thrown by the *rejects* method can be seen in the Figure 20.

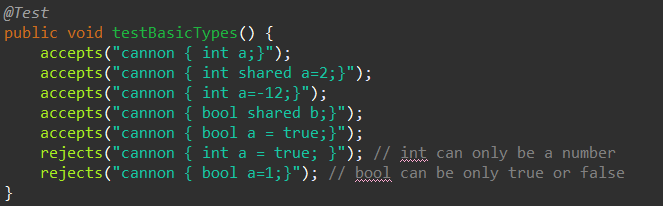


Figure 19. Basic types context tests

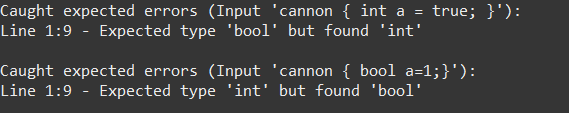


Figure 20. Basic types caught expected errors

## Declarations tests

In the Figure 21 declarations context tests can be seen. These test works the same as previous. *accepts* method should accept the program, *rejects* reject the program. The errors thrown by the *rejects* method can be seen in the Figure 22.

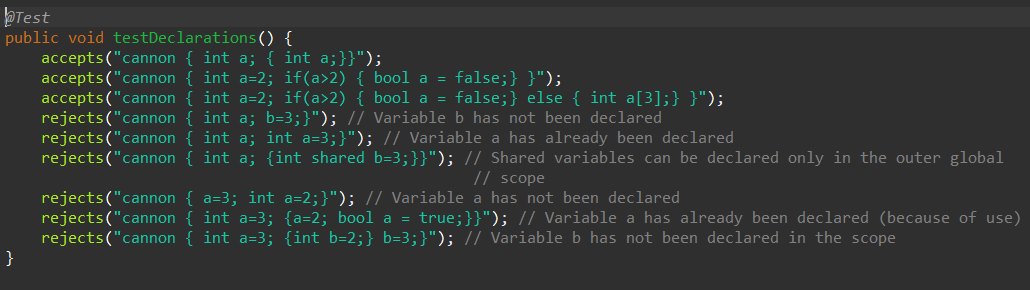


Figure 21. Declarations context tests

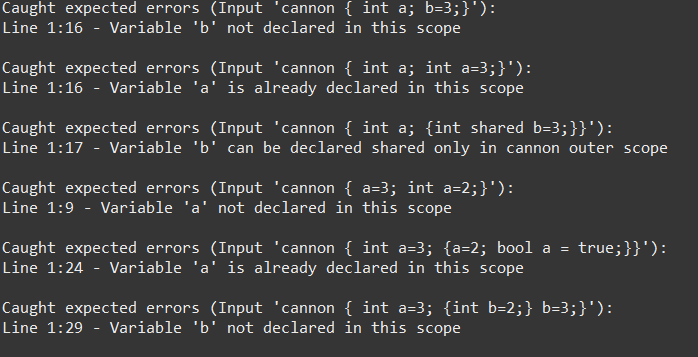


Figure 22. Declarations tests caught expected errors

## Expressions tests

In the Figure 23 expressions context tests can be seen. These test works the same as previous. *accepts* method should accept the program, *rejects* reject the program. The errors thrown by the *rejects* method can be seen in the Figure 24.

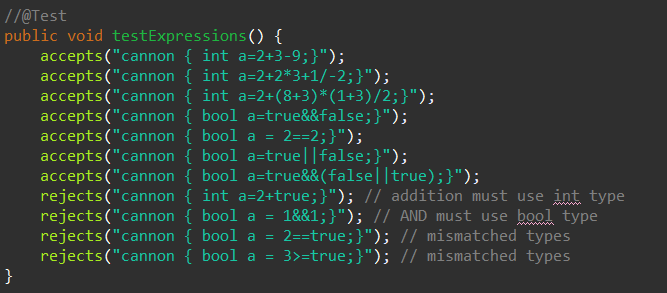


Figure 23. Expressions context tests

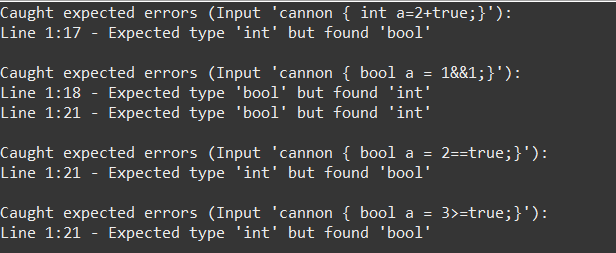


Figure 24. Expressions tests caught expected errors

## While, if tests

In the Figure 25 while, if context tests can be seen. These test works the same as previous. *accepts* method should accept the program, *rejects* reject the program. The errors thrown by the *rejects* method can be seen in the Figure 26.

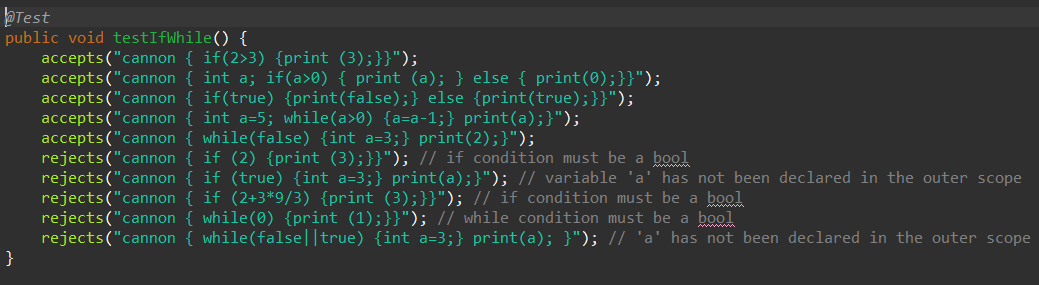


Figure 25. While, if context tests

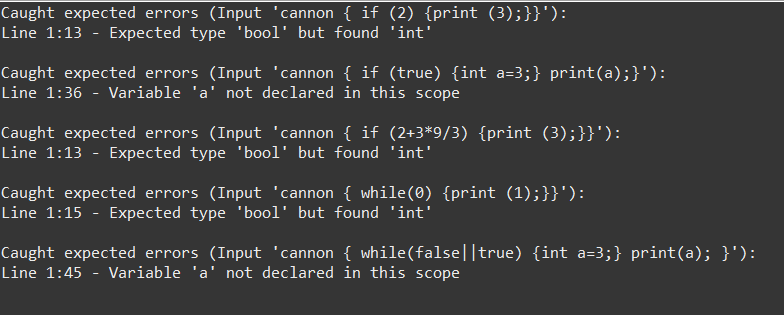


Figure 26. While, if tests caught expected errors

## Concurrency tests

In the Figure 27 concurrency context tests can be seen. These test works the same as previous. *accepts* method should accept the program, *rejects* reject the program. The errors thrown by the *rejects* method can be seen in the Figure 28.

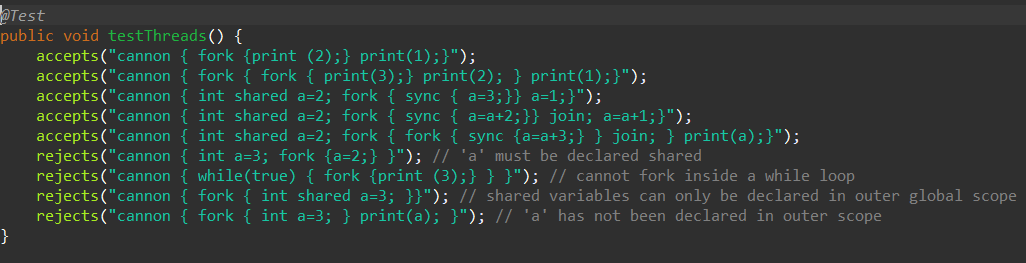


Figure 27. Concurrency context tests

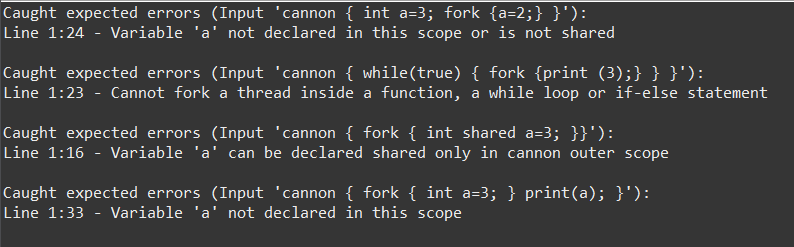


Figure 28. Concurrency tests caught expected errors

## Arrays tests

In the Figure 29 arrays context tests can be seen. These test works the same as previous. *accepts* method should accept the program, *rejects* reject the program. The errors thrown by the *rejects* method can be seen in the Figure 30.

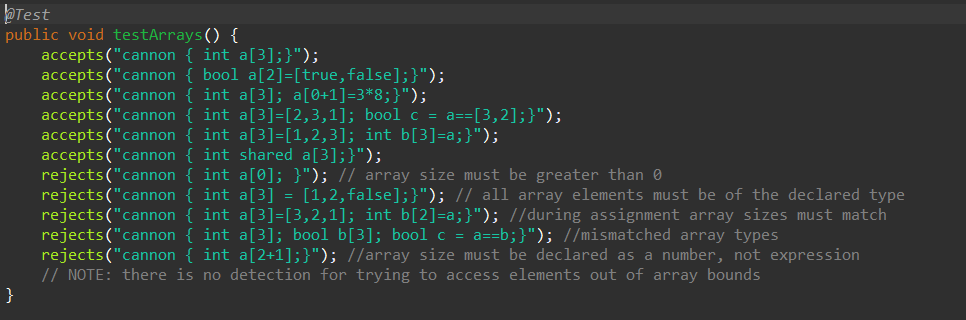


Figure 29. Arrays context tests

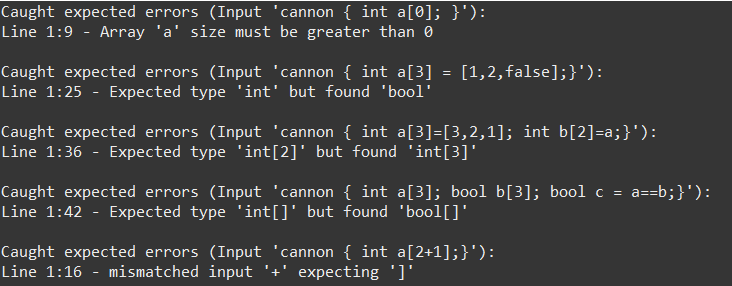


Figure 30. Arrays tests caught expected errors

## Procedures tests

In the Figure 31 procedures context tests can be seen. These test works the same as previous. *accepts* method should accept the program, *rejects* reject the program. The errors thrown by the *rejects* method can be seen in the Figure 32.

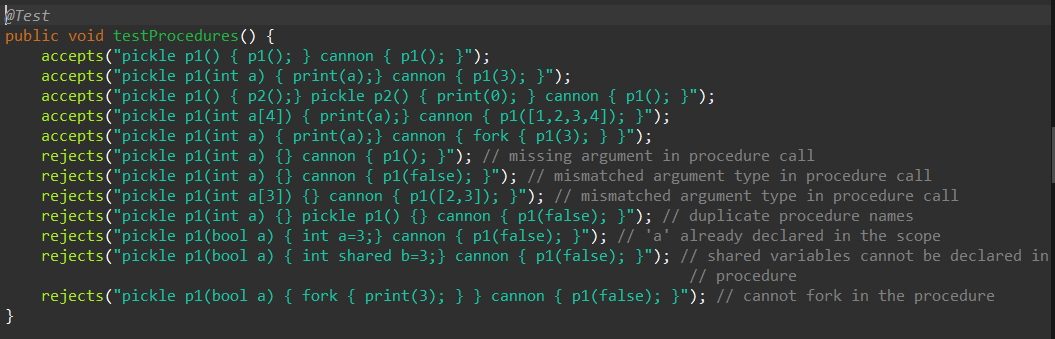


Figure 31. Procedures context tests

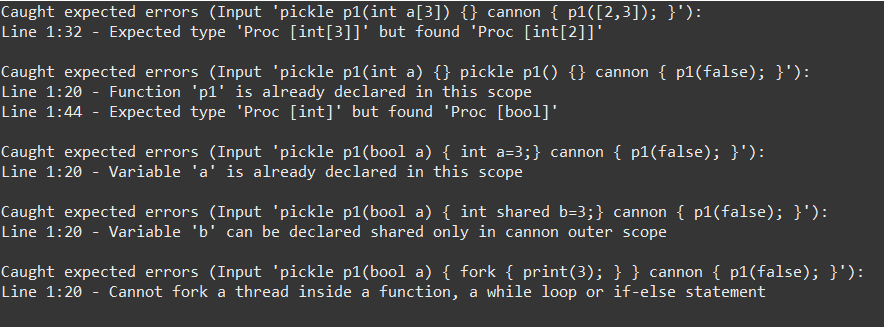


Figure 32. Procedures tests caught expected errors

## File tests

In the Figure 33 tests for programs written in files can be seen. Program source codes can be seen in the Figure 34, Figure 35, Figure 36, Figure 37, Figure 38, Figure 39, Figure 40, Figure 41, Figure 42, Figure 43 and Figure 44. These test works the same as previous. *accepts* method should accept the program, *rejects* reject the program and parse tree tests should contain the expected information. The errors thrown by the *rejects* method can be seen in the Figure 45 and Figure 46.

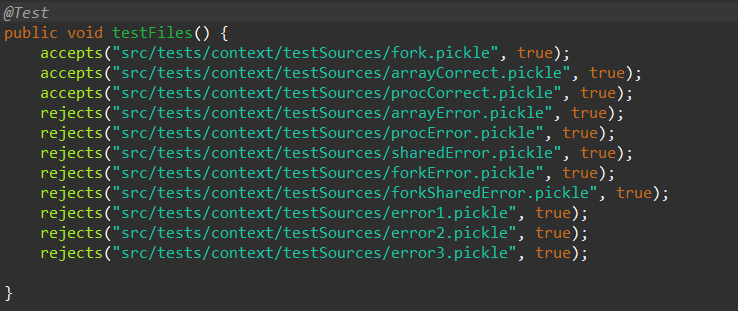


Figure 33. Context tests for programs written in files

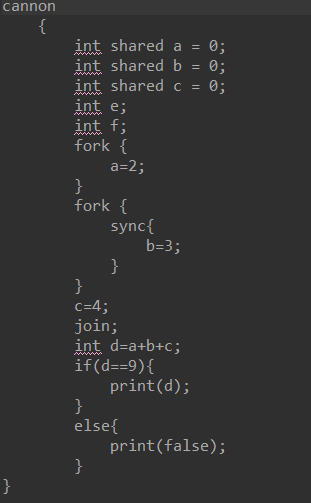


Figure 34. fork.pickle source code

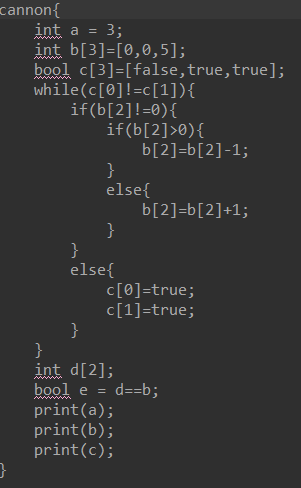


Figure 35. arrayCorrect.pickle source code

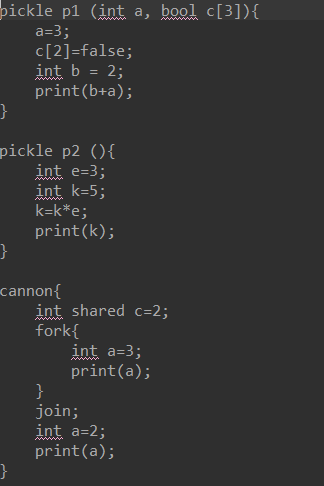


Figure 36. procCorrect.pickle source code

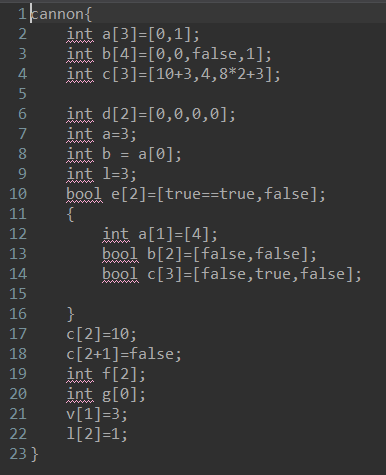


Figure 37. arrayError.pickle source code

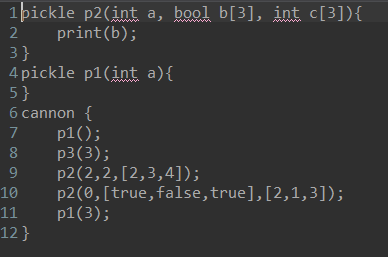


Figure 38. procError.pickle source code

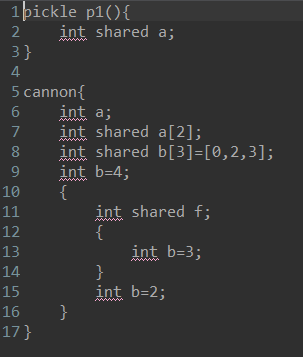


Figure 39. sharedError.pickle source code

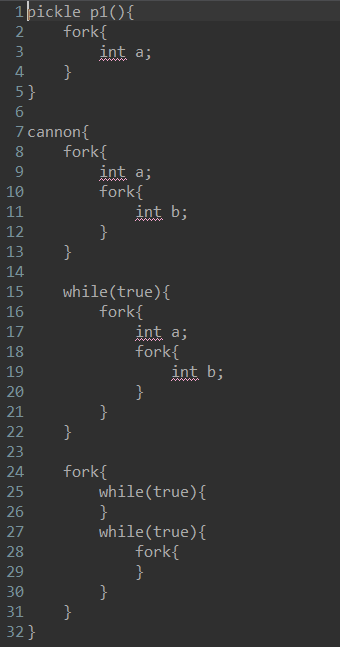


Figure 40. forkError.pickle source code

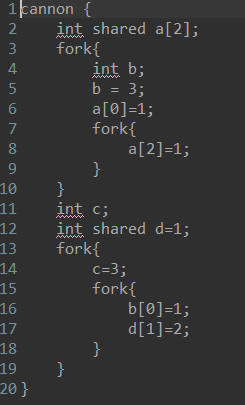


Figure 41. forkSharedError.pickle source code

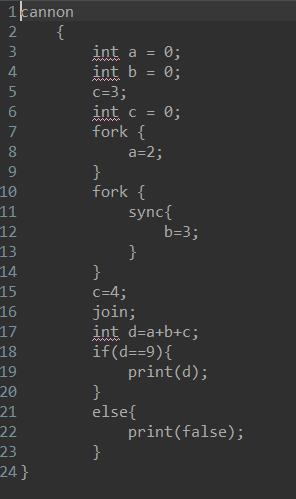


Figure 42. error1.pickle source code

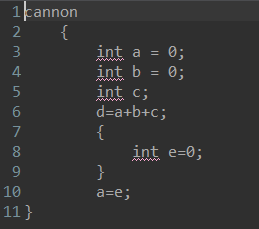


Figure 43. error2.pickle source code

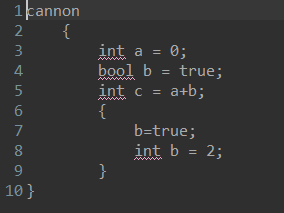


Figure 44. error3.pickle source code

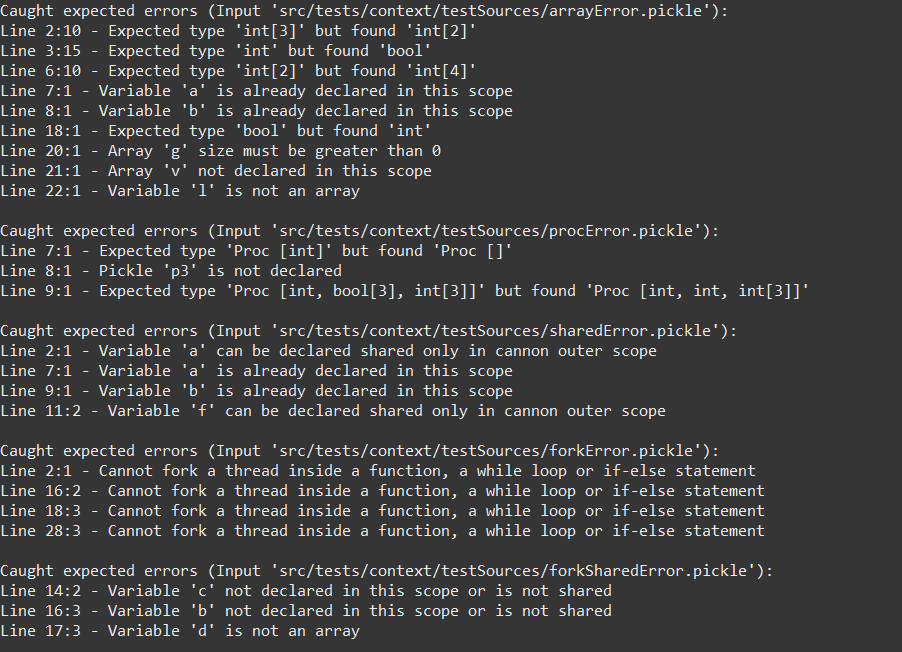


Figure 45. Caught expected errors for programs written in files (1)

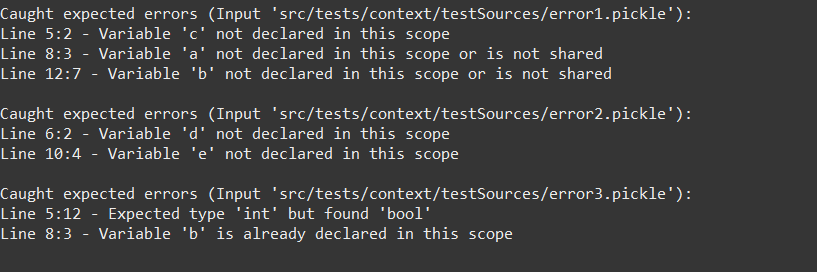


Figure 46. Caught expected errors for programs written in files (2)

## JUnit results

As can be seen in the Figure 47, all tests pass.

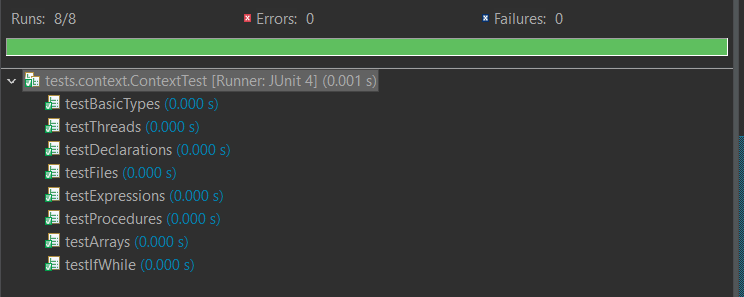


Figure 47. Context tests JUnit results

# Semantic tests results

Semantic tests check the correctness of program run-time behaviour. These tests do generate Sprockell code and automatically run it to test the correctness of the compiler. There few simple tests to check the correctness of features (they are written in strings) and there more complicated programs that are written in files.

## Types and assignments tests

In the Figure 48 types and assignments semantic tests can be seen. Generated Sprockell instruction files can be found in Table 1, Table 2, Table 3, Table 4 and Table 5. *check* method runs the generated file with *runhaskell* command through the terminal and collects the output. If output matches the expected output test passes, otherwise fails.

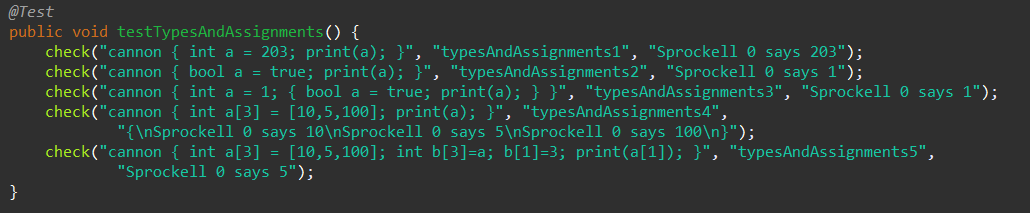


Figure 48. Types and assignments semantic tests

Table 1. Generated typesAndAssignments1.hs file

|  |
| --- |
| import Sprockell  prog :: [Instruction]  prog = [  Jump (Abs (1))  , Push regSP  , Pop regA  , Compute Decr regA reg0 regA  , Load (ImmValue (1)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (203)) regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , WriteInstr regB numberIO  , EndProg  ]  main = run [prog] |

Table 2. Generated typesAndAssignments2.hs file

|  |
| --- |
| import Sprockell  prog :: [Instruction]  prog = [  Jump (Abs (1))  , Push regSP  , Pop regA  , Compute Decr regA reg0 regA  , Load (ImmValue (1)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (1)) regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , WriteInstr regB numberIO  , EndProg  ]  main = run [prog] |

Table 3. Generated typesAndAssignments3.hs file

|  |
| --- |
| import Sprockell  prog :: [Instruction]  prog = [  Jump (Abs (1))  , Push regSP  , Pop regA  , Compute Decr regA reg0 regA  , Load (ImmValue (2)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (1)) regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (1)) regB  , Load (ImmValue (1)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (1)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , WriteInstr regB numberIO  , EndProg  ]  main = run [prog] |

Table 4. Generated typesAndAssignments4.hs file

|  |
| --- |
| import Sprockell  prog :: [Instruction]  prog = [  Jump (Abs (1))  , Push regSP  , Pop regA  , Compute Decr regA reg0 regA  , Load (ImmValue (3)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (10)) regB  , Push regB  , Load (ImmValue (5)) regB  , Push regB  , Load (ImmValue (100)) regB  , Push regB  , Load (ImmValue (0)) regD  , Compute Sub regA regD regD  , Load (ImmValue (2)) regB  , Compute Sub regD regB regD  , Compute Lt regB reg0 regC  , Branch regC (Rel (6))  , Pop regC  , Store regC (IndAddr regD)  , Compute Incr regD reg0 regD  , Compute Decr regB reg0 regB  , Jump (Rel (-6))  , Load (ImmValue (0)) regB  , Load (ImmValue (3)) regC  , Compute GtE regB regC regC  , Branch regC (Rel (8))  , Load (ImmValue (0)) regC  , Compute Add regC regB regC  , Compute Sub regA regC regC  , Load (IndAddr regC) regC  , Push regC  , Compute Incr regB reg0 regB  , Jump (Rel (-9))  , Load (ImmValue (123)) regC  , WriteInstr regC charIO  , Load (ImmValue (10)) regC  , WriteInstr regC charIO  , Load (ImmValue (2)) regB  , Compute Lt regB reg0 regC  , Branch regC (Rel (6))  , Compute Add regSP regB regC  , Load (IndAddr regC) regC  , WriteInstr regC numberIO  , Compute Decr regB reg0 regB  , Jump (Rel (-6))  , Load (ImmValue (125)) regC  , WriteInstr regC charIO  , Load (ImmValue (10)) regC  , WriteInstr regC charIO  , Load (ImmValue (3)) regB  , Compute Add regSP regB regSP  , EndProg  ]  main = run [prog] |

Table 5. Generated typesAndAssignments5.hs file

|  |
| --- |
| import Sprockell  prog :: [Instruction]  prog = [  Jump (Abs (1))  , Push regSP  , Pop regA  , Compute Decr regA reg0 regA  , Load (ImmValue (6)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (10)) regB  , Push regB  , Load (ImmValue (5)) regB  , Push regB  , Load (ImmValue (100)) regB  , Push regB  , Load (ImmValue (0)) regD  , Compute Sub regA regD regD  , Load (ImmValue (2)) regB  , Compute Sub regD regB regD  , Compute Lt regB reg0 regC  , Branch regC (Rel (6))  , Pop regC  , Store regC (IndAddr regD)  , Compute Incr regD reg0 regD  , Compute Decr regB reg0 regB  , Jump (Rel (-6))  , Load (ImmValue (0)) regB  , Load (ImmValue (3)) regC  , Compute GtE regB regC regC  , Branch regC (Rel (8))  , Load (ImmValue (0)) regC  , Compute Add regC regB regC  , Compute Sub regA regC regC  , Load (IndAddr regC) regC  , Push regC  , Compute Incr regB reg0 regB  , Jump (Rel (-9))  , Load (ImmValue (3)) regD  , Compute Sub regA regD regD  , Load (ImmValue (2)) regB  , Compute Sub regD regB regD  , Compute Lt regB reg0 regC  , Branch regC (Rel (6))  , Pop regC  , Store regC (IndAddr regD)  , Compute Incr regD reg0 regD  , Compute Decr regB reg0 regB  , Jump (Rel (-6))  , Load (ImmValue (3)) regB  , Load (ImmValue (1)) regC  , Load (ImmValue (3)) regD  , Compute Add regD regC regD  , Compute Sub regA regD regD  , Store regB (IndAddr regD)  , Load (ImmValue (1)) regB  , Load (ImmValue (0)) regC  , Compute Add regC regB regC  , Compute Sub regA regC regC  , Load (IndAddr regC) regC  , WriteInstr regC numberIO  , EndProg  ]  main = run [prog] |

## Expressions tests

In the Figure 49 expressions semantic tests can be seen. Generated Sprockell instruction files can be found in Table 6, Table 7, Table 8, Table 9 and Table 10. *check* method runs the generated file with *runhaskell* command through the terminal and collects the output. If output matches the expected output test passes, otherwise fails.

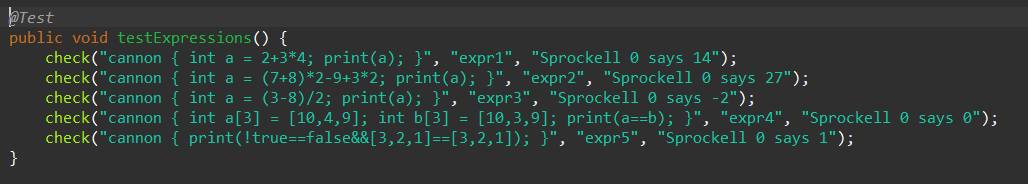


Figure 49. Expressions semantic tests

Table 6. Generated expr1.hs file

|  |
| --- |
| import Sprockell  prog :: [Instruction]  prog = [  Jump (Abs (1))  , Push regSP  , Pop regA  , Compute Decr regA reg0 regA  , Load (ImmValue (1)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (2)) regB  , Load (ImmValue (3)) regC  , Load (ImmValue (4)) regD  , Compute Mul regC regD regC  , Compute Add regB regC regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , WriteInstr regB numberIO  , EndProg  ]  main = run [prog] |

Table 7. Generated expr2.hs file

|  |
| --- |
| import Sprockell  prog :: [Instruction]  prog = [  Jump (Abs (1))  , Push regSP  , Pop regA  , Compute Decr regA reg0 regA  , Load (ImmValue (1)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (7)) regB  , Load (ImmValue (8)) regC  , Compute Add regB regC regB  , Load (ImmValue (2)) regC  , Compute Mul regB regC regB  , Load (ImmValue (9)) regC  , Compute Sub regB regC regB  , Load (ImmValue (3)) regC  , Load (ImmValue (2)) regD  , Compute Mul regC regD regC  , Compute Add regB regC regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , WriteInstr regB numberIO  , EndProg  ]  main = run [prog] |

Table 8. Generated expr3.hs file

|  |
| --- |
| import Sprockell  prog :: [Instruction]  prog = [  Jump (Abs (1))  , Push regSP  , Pop regA  , Compute Decr regA reg0 regA  , Load (ImmValue (1)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (3)) regB  , Load (ImmValue (8)) regC  , Compute Sub regB regC regB  , Load (ImmValue (2)) regC  , Compute GtE regB reg0 regD  , Branch regD (Rel (3))  , Load (ImmValue (-1)) regD  , Compute Mul regB regD regB  , Compute GtE regC reg0 regE  , Branch regE (Rel (3))  , Load (ImmValue (-1)) regE  , Compute Mul regC regE regC  , Compute Mul regD regE regD  , Push regD  , Load (ImmValue (-1)) regD  , Compute Incr regD reg0 regD  , Compute GtE regB regC regE  , Compute Sub regB regC regB  , Branch regE (Rel (-3))  , Pop regE  , Compute Mul regD regE regD  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Store regD (IndAddr regB)  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , WriteInstr regB numberIO  , EndProg  ]  main = run [prog] |

Table 9. Generated expr4.hs file

|  |
| --- |
| import Sprockell  prog :: [Instruction]  prog = [  Jump (Abs (1))  , Push regSP  , Pop regA  , Compute Decr regA reg0 regA  , Load (ImmValue (6)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (10)) regB  , Push regB  , Load (ImmValue (4)) regB  , Push regB  , Load (ImmValue (9)) regB  , Push regB  , Load (ImmValue (0)) regD  , Compute Sub regA regD regD  , Load (ImmValue (2)) regB  , Compute Sub regD regB regD  , Compute Lt regB reg0 regC  , Branch regC (Rel (6))  , Pop regC  , Store regC (IndAddr regD)  , Compute Incr regD reg0 regD  , Compute Decr regB reg0 regB  , Jump (Rel (-6))  , Load (ImmValue (10)) regB  , Push regB  , Load (ImmValue (3)) regB  , Push regB  , Load (ImmValue (9)) regB  , Push regB  , Load (ImmValue (3)) regD  , Compute Sub regA regD regD  , Load (ImmValue (2)) regB  , Compute Sub regD regB regD  , Compute Lt regB reg0 regC  , Branch regC (Rel (6))  , Pop regC  , Store regC (IndAddr regD)  , Compute Incr regD reg0 regD  , Compute Decr regB reg0 regB  , Jump (Rel (-6))  , Load (ImmValue (0)) regB  , Load (ImmValue (3)) regC  , Compute GtE regB regC regC  , Branch regC (Rel (8))  , Load (ImmValue (0)) regC  , Compute Add regC regB regC  , Compute Sub regA regC regC  , Load (IndAddr regC) regC  , Push regC  , Compute Incr regB reg0 regB  , Jump (Rel (-9))  , Load (ImmValue (0)) regB  , Load (ImmValue (3)) regC  , Compute GtE regB regC regC  , Branch regC (Rel (8))  , Load (ImmValue (3)) regC  , Compute Add regC regB regC  , Compute Sub regA regC regC  , Load (IndAddr regC) regC  , Push regC  , Compute Incr regB reg0 regB  , Jump (Rel (-9))  , Load (ImmValue (1)) regB  , Load (ImmValue (3)) regC  , Compute LtE regC reg0 regD  , Branch regD (Rel (9))  , Pop regD  , Load (ImmValue (2)) regE  , Compute Add regSP regE regE  , Load (IndAddr regE) regE  , Compute Equal regD regE regD  , Compute And regB regD regB  , Compute Decr regC reg0 regC  , Jump (Rel (-9))  , Load (ImmValue (3)) regC  , Compute Add regSP regC regSP  , WriteInstr regB numberIO  , EndProg  ]  main = run [prog] |

Table 10. Generated expr5.hs file

|  |
| --- |
| import Sprockell  prog :: [Instruction]  prog = [  Jump (Abs (1))  , Push regSP  , Pop regA  , Compute Decr regA reg0 regA  , Load (ImmValue (0)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (1)) regB  , Compute Equal regB reg0 regB  , Compute Equal regB reg0 regB  , Load (ImmValue (3)) regC  , Push regC  , Load (ImmValue (2)) regC  , Push regC  , Load (ImmValue (1)) regC  , Push regC  , Load (ImmValue (3)) regC  , Push regC  , Load (ImmValue (2)) regC  , Push regC  , Load (ImmValue (1)) regC  , Push regC  , Load (ImmValue (1)) regC  , Load (ImmValue (3)) regD  , Compute LtE regD reg0 regE  , Branch regE (Rel (9))  , Pop regE  , Load (ImmValue (2)) regF  , Compute Add regSP regF regF  , Load (IndAddr regF) regF  , Compute Equal regE regF regE  , Compute And regC regE regC  , Compute Decr regD reg0 regD  , Jump (Rel (-9))  , Load (ImmValue (3)) regD  , Compute Add regSP regD regSP  , Compute And regB regC regB  , WriteInstr regB numberIO  , EndProg  ]  main = run [prog] |

## While, if tests

In the Figure 50 while, if semantic tests can be seen. Generated Sprockell instruction files can be found in Table 11, Table 12 and Table 13. *check* method runs the generated file with *runhaskell* command through the terminal and collects the output. If output matches the expected output test passes, otherwise fails.

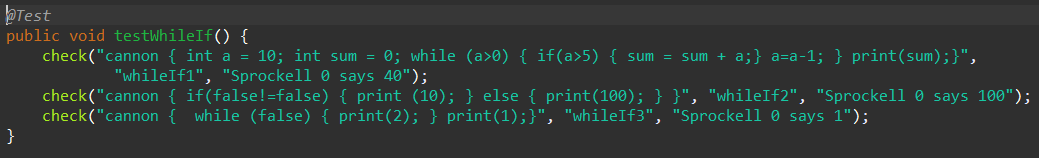


Figure 50. While, if semantic tests

Table 11. Generated whileIf1.hs file

|  |
| --- |
| import Sprockell  prog :: [Instruction]  prog = [  Jump (Abs (1))  , Push regSP  , Pop regA  , Compute Decr regA reg0 regA  , Load (ImmValue (2)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (10)) regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (0)) regB  , Load (ImmValue (1)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (0)) regC  , Compute Gt regB regC regB  , Compute Equal regB reg0 regC  , Branch regC (Abs (49))  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (5)) regC  , Compute Gt regB regC regB  , Compute Equal regB reg0 regC  , Branch regC (Abs (39))  , Load (ImmValue (1)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Load (IndAddr regC) regC  , Compute Add regB regC regB  , Load (ImmValue (1)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Jump (Abs (39))  , Nop  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (1)) regC  , Compute Sub regB regC regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Jump (Abs (14))  , Nop  , Load (ImmValue (1)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , WriteInstr regB numberIO  , EndProg  ]  main = run [prog] |

Table 12. Generated whileIf2.hs file

|  |
| --- |
| import Sprockell  prog :: [Instruction]  prog = [  Jump (Abs (1))  , Push regSP  , Pop regA  , Compute Decr regA reg0 regA  , Load (ImmValue (0)) regB  , Compute Sub regSP regB regSP  , Compute NEq reg0 reg0 reg0  , Compute Equal reg0 reg0 regB  , Branch regB (Abs (12))  , Load (ImmValue (10)) regB  , WriteInstr regB numberIO  , Jump (Abs (14))  , Load (ImmValue (100)) regB  , WriteInstr regB numberIO  , Nop  , EndProg  ]  main = run [prog] |

Table 13. Generated whileIf3.hs file

|  |
| --- |
| import Sprockell  prog :: [Instruction]  prog = [  Jump (Abs (1))  , Push regSP  , Pop regA  , Compute Decr regA reg0 regA  , Load (ImmValue (0)) regB  , Compute Sub regSP regB regSP  , Compute Equal reg0 reg0 regB  , Branch regB (Abs (11))  , Load (ImmValue (2)) regB  , WriteInstr regB numberIO  , Jump (Abs (6))  , Nop  , Load (ImmValue (1)) regB  , WriteInstr regB numberIO  , EndProg  ]  main = run [prog] |

## Concurrency tests

In the Figure 51 concurrency semantic tests can be seen. Program source files can be found in Figure 52, Figure 53 and Figure 54. Generated Sprockell instruction files can be found in Table 14, Table 15 and Table 16. *check* method runs the generated file with *runhaskell* command through the terminal and collects the output. If output matches the expected output test passes, otherwise fails.

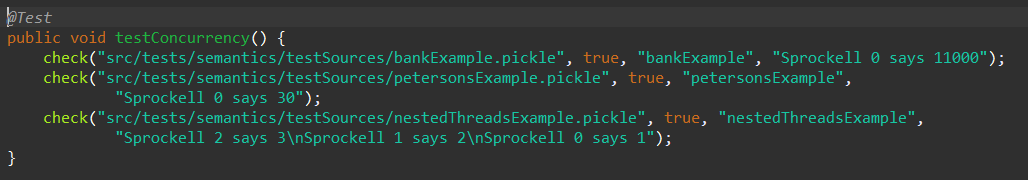


Figure 51. Concurrency sematic tests

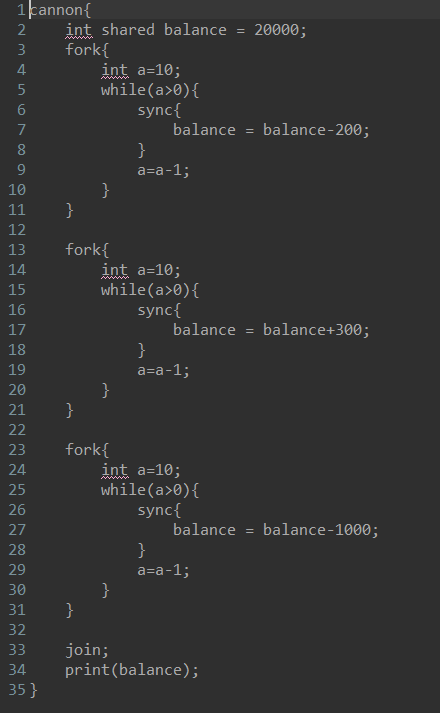


Figure 52. bankExample.pickle source code

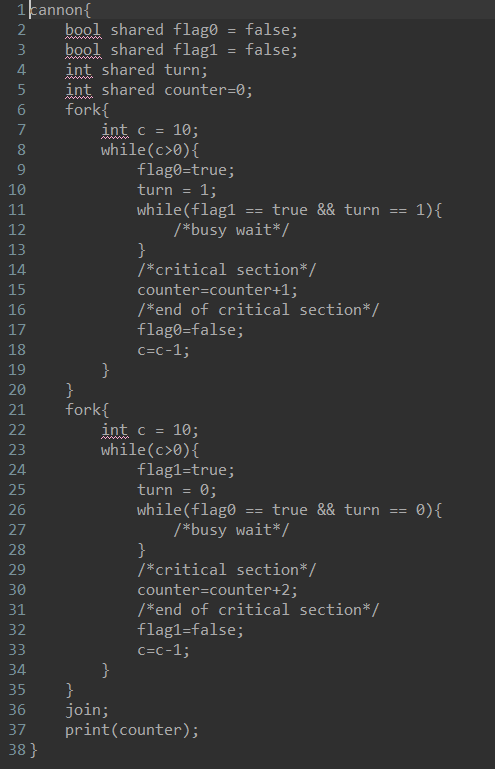


Figure 53. petersonsExample.pickle source code

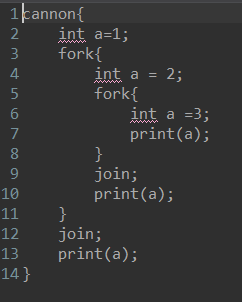


Figure 54. nestedThreadsExample.pickle source code

Table 14. Generated bankExample.hs file

|  |
| --- |
| import Sprockell  prog :: [Instruction]  prog = [  Jump (Abs (1))  , Push regSP  , Pop regA  , Compute Decr regA reg0 regA  , Branch regSprID (Rel (2))  , Jump (Rel (6))  , ReadInstr (IndAddr regSprID)  , Receive regB  , Compute Equal regB reg0 regC  , Branch regC (Rel (-3))  , Jump (Ind regB)  , Load (ImmValue (0)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (20000)) regB  , WriteInstr regB (DirAddr (4))  , Load (ImmValue (18)) regB  , WriteInstr regB (DirAddr (1))  , Jump (Abs (54))  , Load (ImmValue (1)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (10)) regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (0)) regC  , Compute Gt regB regC regB  , Compute Equal regB reg0 regC  , Branch regC (Abs (51))  , TestAndSet (DirAddr (0))  , Receive regB  , Branch regB (Rel (2))  , Jump (Rel (-3))  , ReadInstr (DirAddr (4))  , Receive regB  , Load (ImmValue (200)) regC  , Compute Sub regB regC regB  , Load (ImmValue (4)) regC  , WriteInstr regB (IndAddr regC)  , WriteInstr reg0 (DirAddr (0))  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (1)) regC  , Compute Sub regB regC regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Jump (Abs (24))  , Nop  , WriteInstr reg0 (IndAddr regSprID)  , EndProg  , Load (ImmValue (57)) regB  , WriteInstr regB (DirAddr (2))  , Jump (Abs (93))  , Load (ImmValue (1)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (10)) regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (0)) regC  , Compute Gt regB regC regB  , Compute Equal regB reg0 regC  , Branch regC (Abs (90))  , TestAndSet (DirAddr (0))  , Receive regB  , Branch regB (Rel (2))  , Jump (Rel (-3))  , ReadInstr (DirAddr (4))  , Receive regB  , Load (ImmValue (300)) regC  , Compute Add regB regC regB  , Load (ImmValue (4)) regC  , WriteInstr regB (IndAddr regC)  , WriteInstr reg0 (DirAddr (0))  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (1)) regC  , Compute Sub regB regC regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Jump (Abs (63))  , Nop  , WriteInstr reg0 (IndAddr regSprID)  , EndProg  , Load (ImmValue (96)) regB  , WriteInstr regB (DirAddr (3))  , Jump (Abs (132))  , Load (ImmValue (1)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (10)) regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (0)) regC  , Compute Gt regB regC regB  , Compute Equal regB reg0 regC  , Branch regC (Abs (129))  , TestAndSet (DirAddr (0))  , Receive regB  , Branch regB (Rel (2))  , Jump (Rel (-3))  , ReadInstr (DirAddr (4))  , Receive regB  , Load (ImmValue (1000)) regC  , Compute Sub regB regC regB  , Load (ImmValue (4)) regC  , WriteInstr regB (IndAddr regC)  , WriteInstr reg0 (DirAddr (0))  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (1)) regC  , Compute Sub regB regC regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Jump (Abs (102))  , Nop  , WriteInstr reg0 (IndAddr regSprID)  , EndProg  , ReadInstr (DirAddr (1))  , Receive regB  , ReadInstr (DirAddr (2))  , Receive regC  , Compute Or regB regC regB  , ReadInstr (DirAddr (3))  , Receive regC  , Compute Or regB regC regB  , Branch regB (Rel (-8))  , ReadInstr (DirAddr (4))  , Receive regB  , WriteInstr regB numberIO  , EndProg  ]  main = run [prog,prog,prog,prog] |

Table 15. Generated petersonsExample.hs file

|  |
| --- |
| import Sprockell  prog :: [Instruction]  prog = [  Jump (Abs (1))  , Push regSP  , Pop regA  , Compute Decr regA reg0 regA  , Branch regSprID (Rel (2))  , Jump (Rel (6))  , ReadInstr (IndAddr regSprID)  , Receive regB  , Compute Equal regB reg0 regC  , Branch regC (Rel (-3))  , Jump (Ind regB)  , Load (ImmValue (0)) regB  , Compute Sub regSP regB regSP  , WriteInstr reg0 (DirAddr (3))  , WriteInstr reg0 (DirAddr (4))  , WriteInstr reg0 (DirAddr (5))  , Load (ImmValue (0)) regB  , WriteInstr regB (DirAddr (6))  , Load (ImmValue (21)) regB  , WriteInstr regB (DirAddr (1))  , Jump (Abs (73))  , Load (ImmValue (1)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (10)) regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (0)) regC  , Compute Gt regB regC regB  , Compute Equal regB reg0 regC  , Branch regC (Abs (70))  , Load (ImmValue (1)) regB  , Load (ImmValue (3)) regC  , WriteInstr regB (IndAddr regC)  , Load (ImmValue (1)) regB  , Load (ImmValue (5)) regC  , WriteInstr regB (IndAddr regC)  , ReadInstr (DirAddr (4))  , Receive regB  , Load (ImmValue (1)) regC  , Compute Equal regB regC regB  , ReadInstr (DirAddr (5))  , Receive regC  , Load (ImmValue (1)) regD  , Compute Equal regC regD regC  , Compute And regB regC regB  , Compute Equal regB reg0 regC  , Branch regC (Abs (52))  , Jump (Abs (40))  , Nop  , ReadInstr (DirAddr (6))  , Receive regB  , Load (ImmValue (1)) regC  , Compute Add regB regC regB  , Load (ImmValue (6)) regC  , WriteInstr regB (IndAddr regC)  , Load (ImmValue (3)) regB  , WriteInstr reg0 (IndAddr regB)  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (1)) regC  , Compute Sub regB regC regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Jump (Abs (27))  , Nop  , WriteInstr reg0 (IndAddr regSprID)  , EndProg  , Load (ImmValue (76)) regB  , WriteInstr regB (DirAddr (2))  , Jump (Abs (128))  , Load (ImmValue (1)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (10)) regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (0)) regC  , Compute Gt regB regC regB  , Compute Equal regB reg0 regC  , Branch regC (Abs (125))  , Load (ImmValue (1)) regB  , Load (ImmValue (4)) regC  , WriteInstr regB (IndAddr regC)  , Load (ImmValue (0)) regB  , Load (ImmValue (5)) regC  , WriteInstr regB (IndAddr regC)  , ReadInstr (DirAddr (3))  , Receive regB  , Load (ImmValue (1)) regC  , Compute Equal regB regC regB  , ReadInstr (DirAddr (5))  , Receive regC  , Load (ImmValue (0)) regD  , Compute Equal regC regD regC  , Compute And regB regC regB  , Compute Equal regB reg0 regC  , Branch regC (Abs (107))  , Jump (Abs (95))  , Nop  , ReadInstr (DirAddr (6))  , Receive regB  , Load (ImmValue (2)) regC  , Compute Add regB regC regB  , Load (ImmValue (6)) regC  , WriteInstr regB (IndAddr regC)  , Load (ImmValue (4)) regB  , WriteInstr reg0 (IndAddr regB)  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (1)) regC  , Compute Sub regB regC regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Jump (Abs (82))  , Nop  , WriteInstr reg0 (IndAddr regSprID)  , EndProg  , ReadInstr (DirAddr (1))  , Receive regB  , ReadInstr (DirAddr (2))  , Receive regC  , Compute Or regB regC regB  , Branch regB (Rel (-5))  , ReadInstr (DirAddr (6))  , Receive regB  , WriteInstr regB numberIO  , EndProg  ]  main = run [prog,prog,prog] |

Table 16. Generated nestedThreadsExample.hs file

|  |
| --- |
| import Sprockell  prog :: [Instruction]  prog = [  Jump (Abs (1))  , Push regSP  , Pop regA  , Compute Decr regA reg0 regA  , Branch regSprID (Rel (2))  , Jump (Rel (6))  , ReadInstr (IndAddr regSprID)  , Receive regB  , Compute Equal regB reg0 regC  , Branch regC (Rel (-3))  , Jump (Ind regB)  , Load (ImmValue (1)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (1)) regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (20)) regB  , WriteInstr regB (DirAddr (1))  , Jump (Abs (50))  , Load (ImmValue (1)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (2)) regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (29)) regB  , WriteInstr regB (DirAddr (2))  , Jump (Abs (41))  , Load (ImmValue (1)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (3)) regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , WriteInstr regB numberIO  , WriteInstr reg0 (IndAddr regSprID)  , EndProg  , ReadInstr (DirAddr (2))  , Receive regB  , Branch regB (Rel (-2))  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , WriteInstr regB numberIO  , WriteInstr reg0 (IndAddr regSprID)  , EndProg  , ReadInstr (DirAddr (1))  , Receive regB  , Branch regB (Rel (-2))  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , WriteInstr regB numberIO  , EndProg  ]  main = run [prog,prog,prog] |

## General algorithm tests

In the Figure 55 general algorithms semantic tests can be seen. Program source files can be found in Figure 56, Figure 57, Figure 58, Figure 59, Figure 60 and Figure 61. Generated Sprockell instruction files can be found in Table 17, Table 18, Table 19, Table 20, Table 21 and Table 22. *check* method runs the generated file with *runhaskell* command through the terminal and collects the output. If output matches the expected output test passes, otherwise fails.

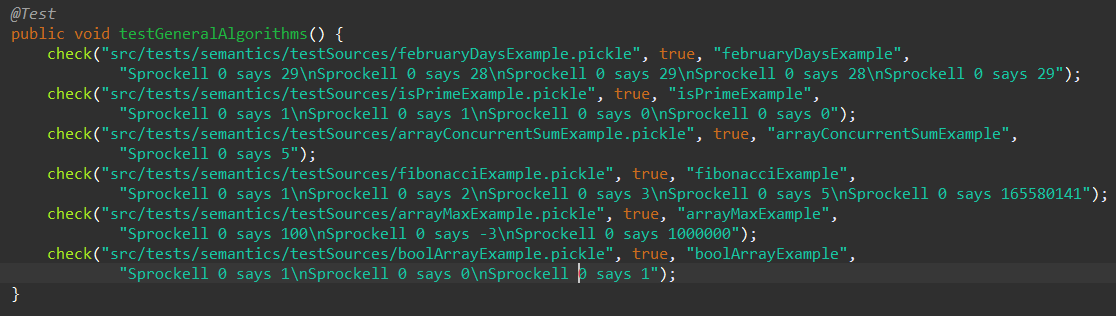


Figure 55. General algorithms semantic tests

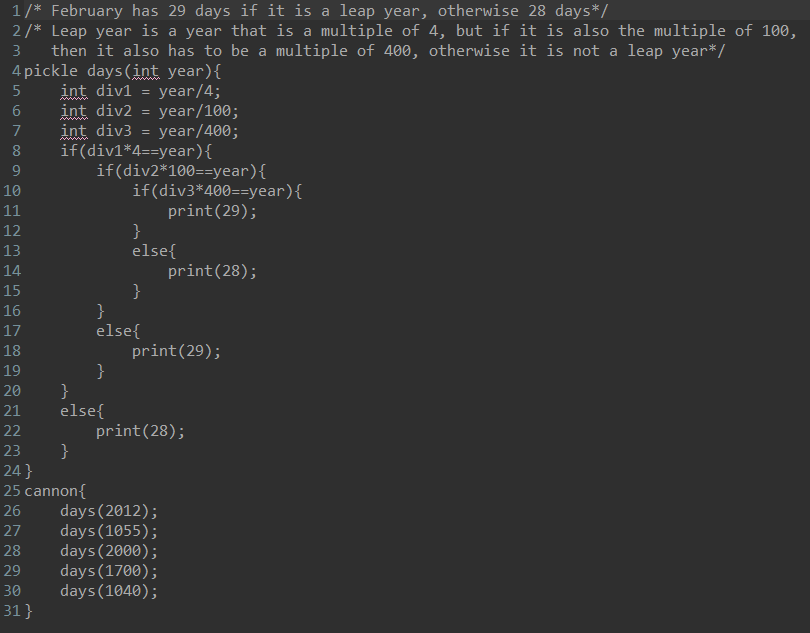


Figure 56. februaryDaysExample.pickle source code

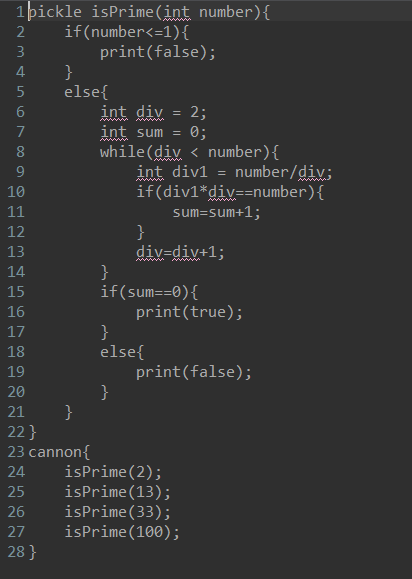


Figure 57. isPrimeExample.pickle source code

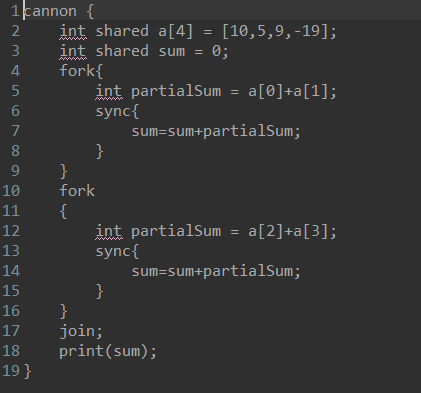


Figure 58. arrayConcurrentSumExample.pickle source code

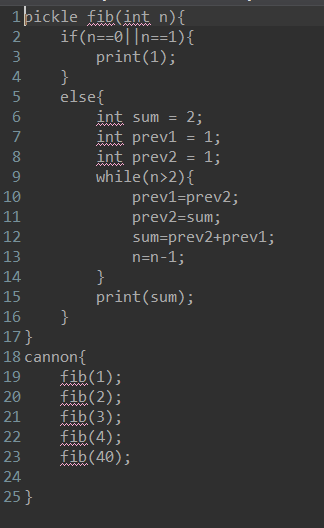


Figure 59. fibonacciExample.pickle source code

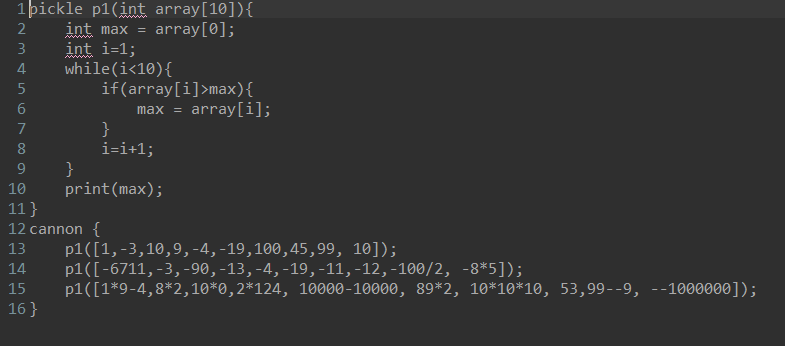


Figure 60. arrayMaxExample.pickle source code

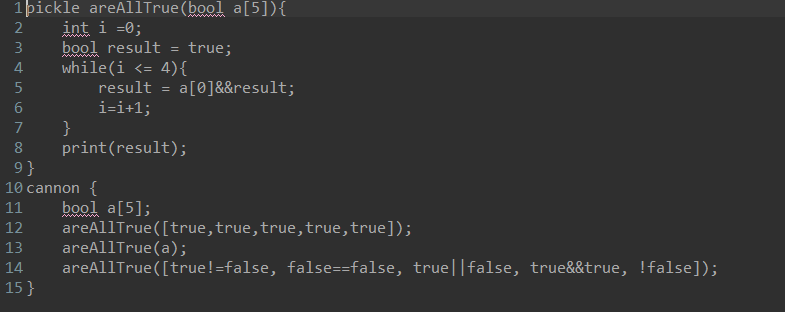


Figure 61. boolArrayExample.pickle source code

Table 17. Generated februaryDaysExample.hs file

|  |
| --- |
| import Sprockell  prog :: [Instruction]  prog = [  Jump (Abs (129))  , Load (ImmValue (3)) regB  , Compute Sub regA regB regSP  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (4)) regC  , Compute GtE regB reg0 regD  , Branch regD (Rel (3))  , Load (ImmValue (-1)) regD  , Compute Mul regB regD regB  , Compute GtE regC reg0 regE  , Branch regE (Rel (3))  , Load (ImmValue (-1)) regE  , Compute Mul regC regE regC  , Compute Mul regD regE regD  , Push regD  , Load (ImmValue (-1)) regD  , Compute Incr regD reg0 regD  , Compute GtE regB regC regE  , Compute Sub regB regC regB  , Branch regE (Rel (-3))  , Pop regE  , Compute Mul regD regE regD  , Load (ImmValue (1)) regB  , Compute Sub regA regB regB  , Store regD (IndAddr regB)  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (100)) regC  , Compute GtE regB reg0 regD  , Branch regD (Rel (3))  , Load (ImmValue (-1)) regD  , Compute Mul regB regD regB  , Compute GtE regC reg0 regE  , Branch regE (Rel (3))  , Load (ImmValue (-1)) regE  , Compute Mul regC regE regC  , Compute Mul regD regE regD  , Push regD  , Load (ImmValue (-1)) regD  , Compute Incr regD reg0 regD  , Compute GtE regB regC regE  , Compute Sub regB regC regB  , Branch regE (Rel (-3))  , Pop regE  , Compute Mul regD regE regD  , Load (ImmValue (2)) regB  , Compute Sub regA regB regB  , Store regD (IndAddr regB)  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (400)) regC  , Compute GtE regB reg0 regD  , Branch regD (Rel (3))  , Load (ImmValue (-1)) regD  , Compute Mul regB regD regB  , Compute GtE regC reg0 regE  , Branch regE (Rel (3))  , Load (ImmValue (-1)) regE  , Compute Mul regC regE regC  , Compute Mul regD regE regD  , Push regD  , Load (ImmValue (-1)) regD  , Compute Incr regD reg0 regD  , Compute GtE regB regC regE  , Compute Sub regB regC regB  , Branch regE (Rel (-3))  , Pop regE  , Compute Mul regD regE regD  , Load (ImmValue (3)) regB  , Compute Sub regA regB regB  , Store regD (IndAddr regB)  , Load (ImmValue (1)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (4)) regC  , Compute Mul regB regC regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Load (IndAddr regC) regC  , Compute Equal regB regC regB  , Compute Equal regB reg0 regC  , Branch regC (Abs (119))  , Load (ImmValue (2)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (100)) regC  , Compute Mul regB regC regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Load (IndAddr regC) regC  , Compute Equal regB regC regB  , Compute Equal regB reg0 regC  , Branch regC (Abs (115))  , Load (ImmValue (3)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (400)) regC  , Compute Mul regB regC regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Load (IndAddr regC) regC  , Compute Equal regB regC regB  , Compute Equal regB reg0 regC  , Branch regC (Abs (111))  , Load (ImmValue (29)) regB  , WriteInstr regB numberIO  , Jump (Abs (113))  , Load (ImmValue (28)) regB  , WriteInstr regB numberIO  , Nop  , Jump (Abs (117))  , Load (ImmValue (29)) regB  , WriteInstr regB numberIO  , Nop  , Jump (Abs (121))  , Load (ImmValue (28)) regB  , WriteInstr regB numberIO  , Nop  , Compute Incr regA reg0 regB  , Compute Incr regB reg0 regC  , Compute Add regC reg0 regSP  , Compute Incr regSP reg0 regSP  , Load (IndAddr regB) regA  , Load (IndAddr regC) regC  , Jump (Ind regC)  , Push regSP  , Pop regA  , Compute Decr regA reg0 regA  , Load (ImmValue (0)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (142)) regB  , Push regB  , Push regA  , Compute Decr regSP reg0 regB  , Load (ImmValue (2012)) regC  , Push regC  , Compute Add regB reg0 regA  , Jump (Abs (1))  , Load (ImmValue (150)) regB  , Push regB  , Push regA  , Compute Decr regSP reg0 regB  , Load (ImmValue (1055)) regC  , Push regC  , Compute Add regB reg0 regA  , Jump (Abs (1))  , Load (ImmValue (158)) regB  , Push regB  , Push regA  , Compute Decr regSP reg0 regB  , Load (ImmValue (2000)) regC  , Push regC  , Compute Add regB reg0 regA  , Jump (Abs (1))  , Load (ImmValue (166)) regB  , Push regB  , Push regA  , Compute Decr regSP reg0 regB  , Load (ImmValue (1700)) regC  , Push regC  , Compute Add regB reg0 regA  , Jump (Abs (1))  , Load (ImmValue (174)) regB  , Push regB  , Push regA  , Compute Decr regSP reg0 regB  , Load (ImmValue (1040)) regC  , Push regC  , Compute Add regB reg0 regA  , Jump (Abs (1))  , EndProg  ]  main = run [prog] |

Table 18. Generated isPrimeExample.hs file

|  |
| --- |
| import Sprockell  prog :: [Instruction]  prog = [  Jump (Abs (108))  , Load (ImmValue (3)) regB  , Compute Sub regA regB regSP  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (1)) regC  , Compute LtE regB regC regB  , Compute Equal regB reg0 regC  , Branch regC (Abs (12))  , WriteInstr reg0 numberIO  , Jump (Abs (100))  , Load (ImmValue (2)) regB  , Load (ImmValue (1)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (0)) regB  , Load (ImmValue (2)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (1)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Load (IndAddr regC) regC  , Compute Lt regB regC regB  , Compute Equal regB reg0 regC  , Branch regC (Abs (87))  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (1)) regC  , Compute Sub regA regC regC  , Load (IndAddr regC) regC  , Compute GtE regB reg0 regD  , Branch regD (Rel (3))  , Load (ImmValue (-1)) regD  , Compute Mul regB regD regB  , Compute GtE regC reg0 regE  , Branch regE (Rel (3))  , Load (ImmValue (-1)) regE  , Compute Mul regC regE regC  , Compute Mul regD regE regD  , Push regD  , Load (ImmValue (-1)) regD  , Compute Incr regD reg0 regD  , Compute GtE regB regC regE  , Compute Sub regB regC regB  , Branch regE (Rel (-3))  , Pop regE  , Compute Mul regD regE regD  , Load (ImmValue (3)) regB  , Compute Sub regA regB regB  , Store regD (IndAddr regB)  , Load (ImmValue (3)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (1)) regC  , Compute Sub regA regC regC  , Load (IndAddr regC) regC  , Compute Mul regB regC regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Load (IndAddr regC) regC  , Compute Equal regB regC regB  , Compute Equal regB reg0 regC  , Branch regC (Abs (77))  , Load (ImmValue (2)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (1)) regC  , Compute Add regB regC regB  , Load (ImmValue (2)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Jump (Abs (77))  , Nop  , Load (ImmValue (1)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (1)) regC  , Compute Add regB regC regB  , Load (ImmValue (1)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Jump (Abs (20))  , Nop  , Load (ImmValue (2)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (0)) regC  , Compute Equal regB regC regB  , Compute Equal regB reg0 regC  , Branch regC (Abs (98))  , Load (ImmValue (1)) regB  , WriteInstr regB numberIO  , Jump (Abs (99))  , WriteInstr reg0 numberIO  , Nop  , Nop  , Compute Incr regA reg0 regB  , Compute Incr regB reg0 regC  , Compute Add regC reg0 regSP  , Compute Incr regSP reg0 regSP  , Load (IndAddr regB) regA  , Load (IndAddr regC) regC  , Jump (Ind regC)  , Push regSP  , Pop regA  , Compute Decr regA reg0 regA  , Load (ImmValue (0)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (121)) regB  , Push regB  , Push regA  , Compute Decr regSP reg0 regB  , Load (ImmValue (2)) regC  , Push regC  , Compute Add regB reg0 regA  , Jump (Abs (1))  , Load (ImmValue (129)) regB  , Push regB  , Push regA  , Compute Decr regSP reg0 regB  , Load (ImmValue (13)) regC  , Push regC  , Compute Add regB reg0 regA  , Jump (Abs (1))  , Load (ImmValue (137)) regB  , Push regB  , Push regA  , Compute Decr regSP reg0 regB  , Load (ImmValue (33)) regC  , Push regC  , Compute Add regB reg0 regA  , Jump (Abs (1))  , Load (ImmValue (145)) regB  , Push regB  , Push regA  , Compute Decr regSP reg0 regB  , Load (ImmValue (100)) regC  , Push regC  , Compute Add regB reg0 regA  , Jump (Abs (1))  , EndProg  ]  main = run [prog] |

Table 19. Generated arrayConcurrentSumExample.hs file

|  |
| --- |
| import Sprockell  prog :: [Instruction]  prog = [  Jump (Abs (1))  , Push regSP  , Pop regA  , Compute Decr regA reg0 regA  , Branch regSprID (Rel (2))  , Jump (Rel (6))  , ReadInstr (IndAddr regSprID)  , Receive regB  , Compute Equal regB reg0 regC  , Branch regC (Rel (-3))  , Jump (Ind regB)  , Load (ImmValue (0)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (10)) regB  , Push regB  , Load (ImmValue (5)) regB  , Push regB  , Load (ImmValue (9)) regB  , Push regB  , Load (ImmValue (19)) regB  , Load (ImmValue (-1)) regC  , Compute Mul regC regB regC  , Push regC  , Load (ImmValue (3)) regD  , Load (ImmValue (3)) regB  , Compute Add regD regB regD  , Compute Lt regB reg0 regC  , Branch regC (Rel (6))  , Pop regC  , WriteInstr regC (IndAddr regD)  , Compute Decr regD reg0 regD  , Compute Decr regB reg0 regB  , Jump (Rel (-6))  , Load (ImmValue (0)) regB  , WriteInstr regB (DirAddr (7))  , Load (ImmValue (38)) regB  , WriteInstr regB (DirAddr (1))  , Jump (Abs (69))  , Load (ImmValue (1)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (0)) regB  , Load (ImmValue (3)) regC  , Compute Add regC regB regC  , ReadInstr (IndAddr regC)  , Receive regC  , Load (ImmValue (1)) regB  , Load (ImmValue (3)) regD  , Compute Add regD regB regD  , ReadInstr (IndAddr regD)  , Receive regD  , Compute Add regC regD regC  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Store regC (IndAddr regB)  , TestAndSet (DirAddr (0))  , Receive regB  , Branch regB (Rel (2))  , Jump (Rel (-3))  , ReadInstr (DirAddr (7))  , Receive regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Load (IndAddr regC) regC  , Compute Add regB regC regB  , Load (ImmValue (7)) regC  , WriteInstr regB (IndAddr regC)  , WriteInstr reg0 (DirAddr (0))  , WriteInstr reg0 (IndAddr regSprID)  , EndProg  , Load (ImmValue (72)) regB  , WriteInstr regB (DirAddr (2))  , Jump (Abs (103))  , Load (ImmValue (1)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (2)) regB  , Load (ImmValue (3)) regC  , Compute Add regC regB regC  , ReadInstr (IndAddr regC)  , Receive regC  , Load (ImmValue (3)) regB  , Load (ImmValue (3)) regD  , Compute Add regD regB regD  , ReadInstr (IndAddr regD)  , Receive regD  , Compute Add regC regD regC  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Store regC (IndAddr regB)  , TestAndSet (DirAddr (0))  , Receive regB  , Branch regB (Rel (2))  , Jump (Rel (-3))  , ReadInstr (DirAddr (7))  , Receive regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Load (IndAddr regC) regC  , Compute Add regB regC regB  , Load (ImmValue (7)) regC  , WriteInstr regB (IndAddr regC)  , WriteInstr reg0 (DirAddr (0))  , WriteInstr reg0 (IndAddr regSprID)  , EndProg  , ReadInstr (DirAddr (1))  , Receive regB  , ReadInstr (DirAddr (2))  , Receive regC  , Compute Or regB regC regB  , Branch regB (Rel (-5))  , ReadInstr (DirAddr (7))  , Receive regB  , WriteInstr regB numberIO  , EndProg  ]  main = run [prog,prog,prog] |

Table 20. Generated fibonacciExample.hs file

|  |
| --- |
| import Sprockell  prog :: [Instruction]  prog = [  Jump (Abs (82))  , Load (ImmValue (3)) regB  , Compute Sub regA regB regSP  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (0)) regC  , Compute Equal regB regC regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Load (IndAddr regC) regC  , Load (ImmValue (1)) regD  , Compute Equal regC regD regC  , Compute Or regB regC regB  , Compute Equal regB reg0 regC  , Branch regC (Abs (19))  , Load (ImmValue (1)) regB  , WriteInstr regB numberIO  , Jump (Abs (74))  , Load (ImmValue (2)) regB  , Load (ImmValue (1)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (1)) regB  , Load (ImmValue (2)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (1)) regB  , Load (ImmValue (3)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (2)) regC  , Compute Gt regB regC regB  , Compute Equal regB reg0 regC  , Branch regC (Abs (69))  , Load (ImmValue (3)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (2)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (1)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (3)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (3)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (2)) regC  , Compute Sub regA regC regC  , Load (IndAddr regC) regC  , Compute Add regB regC regB  , Load (ImmValue (1)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (0)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (1)) regC  , Compute Sub regB regC regB  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Jump (Abs (31))  , Nop  , Load (ImmValue (1)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , WriteInstr regB numberIO  , Nop  , Compute Incr regA reg0 regB  , Compute Incr regB reg0 regC  , Compute Add regC reg0 regSP  , Compute Incr regSP reg0 regSP  , Load (IndAddr regB) regA  , Load (IndAddr regC) regC  , Jump (Ind regC)  , Push regSP  , Pop regA  , Compute Decr regA reg0 regA  , Load (ImmValue (0)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (95)) regB  , Push regB  , Push regA  , Compute Decr regSP reg0 regB  , Load (ImmValue (1)) regC  , Push regC  , Compute Add regB reg0 regA  , Jump (Abs (1))  , Load (ImmValue (103)) regB  , Push regB  , Push regA  , Compute Decr regSP reg0 regB  , Load (ImmValue (2)) regC  , Push regC  , Compute Add regB reg0 regA  , Jump (Abs (1))  , Load (ImmValue (111)) regB  , Push regB  , Push regA  , Compute Decr regSP reg0 regB  , Load (ImmValue (3)) regC  , Push regC  , Compute Add regB reg0 regA  , Jump (Abs (1))  , Load (ImmValue (119)) regB  , Push regB  , Push regA  , Compute Decr regSP reg0 regB  , Load (ImmValue (4)) regC  , Push regC  , Compute Add regB reg0 regA  , Jump (Abs (1))  , Load (ImmValue (127)) regB  , Push regB  , Push regA  , Compute Decr regSP reg0 regB  , Load (ImmValue (40)) regC  , Push regC  , Compute Add regB reg0 regA  , Jump (Abs (1))  , EndProg  ]  main = run [prog] |

Table 21. Generated arrayMaxExample.hs file

|  |
| --- |
| import Sprockell  prog :: [Instruction]  prog = [  Jump (Abs (68))  , Load (ImmValue (11)) regB  , Compute Sub regA regB regSP  , Load (ImmValue (0)) regB  , Load (ImmValue (0)) regC  , Compute Add regC regB regC  , Compute Sub regA regC regC  , Load (IndAddr regC) regC  , Load (ImmValue (10)) regB  , Compute Sub regA regB regB  , Store regC (IndAddr regB)  , Load (ImmValue (1)) regB  , Load (ImmValue (11)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (11)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (10)) regC  , Compute Lt regB regC regB  , Compute Equal regB reg0 regC  , Branch regC (Abs (56))  , Load (ImmValue (11)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (0)) regC  , Compute Add regC regB regC  , Compute Sub regA regC regC  , Load (IndAddr regC) regC  , Load (ImmValue (10)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Compute Gt regC regB regC  , Compute Equal regC reg0 regB  , Branch regB (Abs (46))  , Load (ImmValue (11)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (0)) regC  , Compute Add regC regB regC  , Compute Sub regA regC regC  , Load (IndAddr regC) regC  , Load (ImmValue (10)) regB  , Compute Sub regA regB regB  , Store regC (IndAddr regB)  , Jump (Abs (46))  , Nop  , Load (ImmValue (11)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (1)) regC  , Compute Add regB regC regB  , Load (ImmValue (11)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Jump (Abs (15))  , Nop  , Load (ImmValue (10)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , WriteInstr regB numberIO  , Compute Incr regA reg0 regB  , Compute Incr regB reg0 regC  , Compute Add regC reg0 regSP  , Compute Incr regSP reg0 regSP  , Load (IndAddr regB) regA  , Load (IndAddr regC) regC  , Jump (Ind regC)  , Push regSP  , Pop regA  , Compute Decr regA reg0 regA  , Load (ImmValue (0)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (105)) regB  , Push regB  , Push regA  , Compute Decr regSP reg0 regB  , Load (ImmValue (1)) regC  , Push regC  , Load (ImmValue (3)) regC  , Load (ImmValue (-1)) regD  , Compute Mul regD regC regD  , Push regD  , Load (ImmValue (10)) regC  , Push regC  , Load (ImmValue (9)) regC  , Push regC  , Load (ImmValue (4)) regC  , Load (ImmValue (-1)) regD  , Compute Mul regD regC regD  , Push regD  , Load (ImmValue (19)) regC  , Load (ImmValue (-1)) regD  , Compute Mul regD regC regD  , Push regD  , Load (ImmValue (100)) regC  , Push regC  , Load (ImmValue (45)) regC  , Push regC  , Load (ImmValue (99)) regC  , Push regC  , Load (ImmValue (10)) regC  , Push regC  , Compute Add regB reg0 regA  , Jump (Abs (1))  , Load (ImmValue (171)) regB  , Push regB  , Push regA  , Compute Decr regSP reg0 regB  , Load (ImmValue (6711)) regC  , Load (ImmValue (-1)) regD  , Compute Mul regD regC regD  , Push regD  , Load (ImmValue (3)) regC  , Load (ImmValue (-1)) regD  , Compute Mul regD regC regD  , Push regD  , Load (ImmValue (90)) regC  , Load (ImmValue (-1)) regD  , Compute Mul regD regC regD  , Push regD  , Load (ImmValue (13)) regC  , Load (ImmValue (-1)) regD  , Compute Mul regD regC regD  , Push regD  , Load (ImmValue (4)) regC  , Load (ImmValue (-1)) regD  , Compute Mul regD regC regD  , Push regD  , Load (ImmValue (19)) regC  , Load (ImmValue (-1)) regD  , Compute Mul regD regC regD  , Push regD  , Load (ImmValue (11)) regC  , Load (ImmValue (-1)) regD  , Compute Mul regD regC regD  , Push regD  , Load (ImmValue (12)) regC  , Load (ImmValue (-1)) regD  , Compute Mul regD regC regD  , Push regD  , Load (ImmValue (100)) regC  , Load (ImmValue (-1)) regD  , Compute Mul regD regC regD  , Load (ImmValue (2)) regC  , Compute GtE regD reg0 regE  , Branch regE (Rel (3))  , Load (ImmValue (-1)) regE  , Compute Mul regD regE regD  , Compute GtE regC reg0 regF  , Branch regF (Rel (3))  , Load (ImmValue (-1)) regF  , Compute Mul regC regF regC  , Compute Mul regE regF regE  , Push regE  , Load (ImmValue (-1)) regE  , Compute Incr regE reg0 regE  , Compute GtE regD regC regF  , Compute Sub regD regC regD  , Branch regF (Rel (-3))  , Pop regF  , Compute Mul regE regF regE  , Push regE  , Load (ImmValue (8)) regC  , Load (ImmValue (-1)) regD  , Compute Mul regD regC regD  , Load (ImmValue (5)) regC  , Compute Mul regD regC regD  , Push regD  , Compute Add regB reg0 regA  , Jump (Abs (1))  , Load (ImmValue (223)) regB  , Push regB  , Push regA  , Compute Decr regSP reg0 regB  , Load (ImmValue (1)) regC  , Load (ImmValue (9)) regD  , Compute Mul regC regD regC  , Load (ImmValue (4)) regD  , Compute Sub regC regD regC  , Push regC  , Load (ImmValue (8)) regC  , Load (ImmValue (2)) regD  , Compute Mul regC regD regC  , Push regC  , Load (ImmValue (10)) regC  , Load (ImmValue (0)) regD  , Compute Mul regC regD regC  , Push regC  , Load (ImmValue (2)) regC  , Load (ImmValue (124)) regD  , Compute Mul regC regD regC  , Push regC  , Load (ImmValue (10000)) regC  , Load (ImmValue (10000)) regD  , Compute Sub regC regD regC  , Push regC  , Load (ImmValue (89)) regC  , Load (ImmValue (2)) regD  , Compute Mul regC regD regC  , Push regC  , Load (ImmValue (10)) regC  , Load (ImmValue (10)) regD  , Compute Mul regC regD regC  , Load (ImmValue (10)) regD  , Compute Mul regC regD regC  , Push regC  , Load (ImmValue (53)) regC  , Push regC  , Load (ImmValue (99)) regC  , Load (ImmValue (9)) regD  , Load (ImmValue (-1)) regE  , Compute Mul regE regD regE  , Compute Sub regC regE regC  , Push regC  , Load (ImmValue (1000000)) regC  , Load (ImmValue (-1)) regD  , Compute Mul regD regC regD  , Load (ImmValue (-1)) regC  , Compute Mul regC regD regC  , Push regC  , Compute Add regB reg0 regA  , Jump (Abs (1))  , EndProg  ]  main = run [prog] |

Table 22. Generated boolArrayExample.hs file

|  |
| --- |
| import Sprockell  prog :: [Instruction]  prog = [  Jump (Abs (51))  , Load (ImmValue (6)) regB  , Compute Sub regA regB regSP  , Load (ImmValue (0)) regB  , Load (ImmValue (5)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (1)) regB  , Load (ImmValue (6)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Load (ImmValue (5)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (4)) regC  , Compute LtE regB regC regB  , Compute Equal regB reg0 regC  , Branch regC (Abs (39))  , Load (ImmValue (0)) regB  , Load (ImmValue (0)) regC  , Compute Add regC regB regC  , Compute Sub regA regC regC  , Load (IndAddr regC) regC  , Load (ImmValue (6)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Compute And regC regB regC  , Load (ImmValue (6)) regB  , Compute Sub regA regB regB  , Store regC (IndAddr regB)  , Load (ImmValue (5)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , Load (ImmValue (1)) regC  , Compute Add regB regC regB  , Load (ImmValue (5)) regC  , Compute Sub regA regC regC  , Store regB (IndAddr regC)  , Jump (Abs (11))  , Nop  , Load (ImmValue (6)) regB  , Compute Sub regA regB regB  , Load (IndAddr regB) regB  , WriteInstr regB numberIO  , Compute Incr regA reg0 regB  , Compute Incr regB reg0 regC  , Compute Add regC reg0 regSP  , Compute Incr regSP reg0 regSP  , Load (IndAddr regB) regA  , Load (IndAddr regC) regC  , Jump (Ind regC)  , Push regSP  , Pop regA  , Compute Decr regA reg0 regA  , Load (ImmValue (5)) regB  , Compute Sub regSP regB regSP  , Load (ImmValue (0)) regC  , Compute Sub regA regC regC  , Load (ImmValue (4)) regB  , Compute Sub regC regB regC  , Compute Lt regB reg0 regD  , Branch regD (Rel (5))  , Store reg0 (IndAddr regC)  , Compute Incr regC reg0 regC  , Compute Decr regB reg0 regB  , Jump (Rel (-5))  , Load (ImmValue (82)) regB  , Push regB  , Push regA  , Compute Decr regSP reg0 regB  , Load (ImmValue (1)) regC  , Push regC  , Load (ImmValue (1)) regC  , Push regC  , Load (ImmValue (1)) regC  , Push regC  , Load (ImmValue (1)) regC  , Push regC  , Load (ImmValue (1)) regC  , Push regC  , Compute Add regB reg0 regA  , Jump (Abs (1))  , Load (ImmValue (99)) regB  , Push regB  , Push regA  , Compute Decr regSP reg0 regB  , Load (ImmValue (0)) regC  , Load (ImmValue (5)) regD  , Compute GtE regC regD regD  , Branch regD (Rel (8))  , Load (ImmValue (0)) regD  , Compute Add regD regC regD  , Compute Sub regA regD regD  , Load (IndAddr regD) regD  , Push regD  , Compute Incr regC reg0 regC  , Jump (Rel (-9))  , Compute Add regB reg0 regA  , Jump (Abs (1))  , Load (ImmValue (119)) regB  , Push regB  , Push regA  , Compute Decr regSP reg0 regB  , Load (ImmValue (1)) regC  , Compute NEq regC reg0 regC  , Push regC  , Compute Equal reg0 reg0 reg0  , Push reg0  , Load (ImmValue (1)) regC  , Compute Or regC reg0 regC  , Push regC  , Load (ImmValue (1)) regC  , Load (ImmValue (1)) regD  , Compute And regC regD regC  , Push regC  , Compute Equal reg0 reg0 reg0  , Push reg0  , Compute Add regB reg0 regA  , Jump (Abs (1))  , EndProg  ]  main = run [prog] |

## Infinite run tests

In the Figure 62 infinite run semantics tests can be seen. These tests generated Sprockell programs that enter into infinite loop. First program uses while cycle that is always true, and second program uses division by 0 which results into an infinite cycle. That is why these two programs are ran into separate threads and if they are alive after 5 seconds tests pass, otherwise they fail. However, it is important to take into account that created *ghc* processes are not killed by Java and must be killed separately through task manager on Windows, or terminal on Linux.

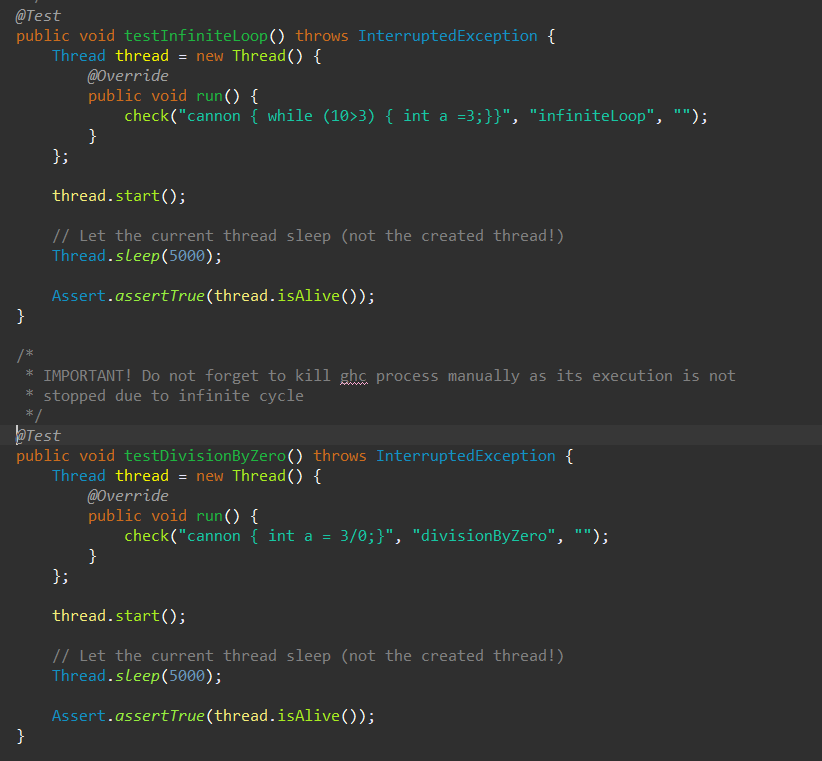


Figure 62. Infinite run semantic tests

## JUnit results

As can be seen from the Figure 63 all tests pass.

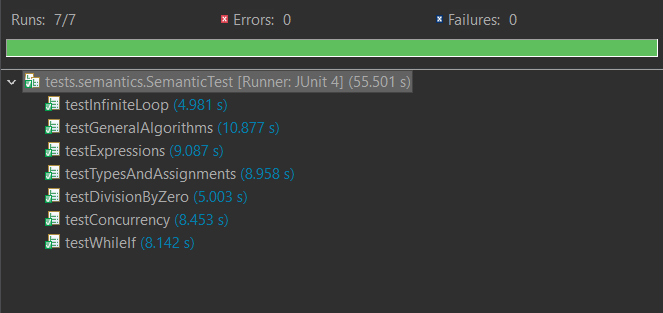


Figure 63. Semantic tests JUnit results