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.

```
@Test
public void testBaseTypes() {
    // Accept-reject tests
    accepts("cannon { int a; }");
    accepts("CaNnon { InT B; }");
    accepts("cannon { a = b45; }");
    accepts("cannon { int a3V4 = 15; }");
    accepts("cannon { bool a; }");
    accepts("cannon { bool a43 = true; }");
    accepts("cannon { bool be = false; }");
    accepts("cannon { bool f1 = f2; }");
    rejects("cannon { int 3aV4 = 15; }"); // identifier names should start with a letter
    rejects("cannon { int a5* = 15; }"); // identifier names contain only letters and digits
    rejects("{ int a; }"); // program main body starts with 'cannon' keyword
    rejects("cannon int a;"); // program main body should be enclosed by braces
    rejects("cannon { int a }"); // each statement (that does not open a new scope) should end with ';'
    rejects("cannon { bool bool = false; }"); // bool is reserver keyword and cannot be used as identifier
    rejects("cannon { integer a1 = 2; }"); // integer is non-existing type keyword (must use 'int')
    rejects("cannon { boolean a1 = true; }"); // boolean is non-existing type keyword (must use 'bool')
    // ParseTree tests
    ParseTree tree = accepts("cannon { int a5=3;}");
    Assert.assertEquals("{inta5=3;}", tree.getChild(1).getText()); // check if body is the same (whitespaces are
                                                                   // removed)
    Assert.assertEquals("int", tree.getChild(1).getChild(1).getChild(0).getText());
    Assert.assertEquals("a5", tree.getChild(1).getChild(1).getChild(1).getText());
    Assert.assertEquals("3", tree.getChild(1).getChild(1).getChild(3).getText());
}
```

Figure 1. Basic types syntax tests

```
Caught expected errors (Input: 'cannon { int 3aV4 = 15; }'):
Line 1:13 - no viable alternative at input 'int3'

Caught expected errors (Input: 'cannon { int a5* = 15; }'):
Line 1:15 - no viable alternative at input 'inta5*'

Caught expected errors (Input: '{ int a; }'):
Line 1:0 - mismatched input '{' expecting {PICKLE, CANNON}

Caught expected errors (Input: 'cannon int a;'):
Line 1:7 - missing '{' at 'int'
Line 1:13 - extraneous input '<EOF>' expecting {BOOL, INT, FORK, IF, JOIN, PRINT, SYNC, WHILE, '{', '}', ID}

Caught expected errors (Input: 'cannon { int a }'):
Line 1:15 - no viable alternative at input 'inta}'

Caught expected errors (Input: 'cannon { bool bool = false; }'):
Line 1:14 - no viable alternative at input 'boolbool'

Caught expected errors (Input: 'cannon { integer a1 = 2; }'):
Line 1:17 - no viable alternative at input 'integera1'

Caught expected errors (Input: 'cannon { boolean a1 = true; }'):
Line 1:17 - no viable alternative at input 'booleana1'
```

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.

```
@Test
public void testExpressions() {
    // Accept-reject tests
    accepts("cannon {int a = 2+9;}");
    accepts("cannon {int a = 10-5;}");
    accepts("cannon {int a = 2*3;}");
    accepts("cannon {int a = 10/3;}");
    accepts("cannon {int a = 2*3+(3+3)*4;}");
    accepts("cannon {int a = -2;}");
    accepts("cannon {int a = --3---4;}");
    accepts("cannon {int a = a*b-3;}");
    accepts("cannon {bool a = true&&false;}");
    accepts("cannon {bool a = true||true;}");
    accepts("cannon {bool a = b==c;}");
    accepts("cannon {bool a = !true;}");
    rejects("cannon {int a = 2**3;}"); // duplicate multiplication symbol
    rejects("cannon {int a = +3;}"); // only '-' and '!' are allowed prefix operators
    rejects("cannon {bool a = true&true;}"); // AND operations uses two '&' symbols
    rejects("cannon {int a = (2+3;}"); // missing ')'
    rejects("cannon {int a == 1;}"); // assignment should use only one '=' operator
    // ParseTree tests
    ParseTree tree = accepts("cannon { int a=2+(3+3*2)*8--3;}");
    Assert.assertEquals("{inta=2+(3+3*2)*8--3;}", tree.getChild(1).getText()); // check if body is the same
    Assert.assertEquals("2+(3+3*2)*8--3", tree.getChild(1).getChild(1).getChild(3).getText());
    Assert.assertEquals("2+(3+3*2)*8", tree.getChild(1).getChild(1).getChild(3).getChild(0).getText());
    Assert.assertEquals("(3+3*2)*8", tree.getChild(1).getChild(1).getChild(3).getChild(0).getChild(2).getText());
    Assert.assertEquals("(3+3*2)",
            tree.getChild(1).getChild(1).getChild(3).getChild(0).getChild(2).getChild(0).getText());
    Assert.assertEquals("3*2", tree.getChild(1).getChild(1).getChild(3).getChild(0).getChild(2).getChild(0)
            .getChild(1).getChild(2).getText());
}
```

Figure 3. Expressions syntax tests

```
Caught expected errors (Input: 'cannon {int a = 2**3;}'):
Line 1:18 - extraneous input '*' expecting {FALSE, TRUE, '(', '[', '-', '!', ID, NUM}

Caught expected errors (Input: 'cannon {int a = +3;}'):
Line 1:16 - extraneous input '+' expecting {FALSE, TRUE, '(', '[', '-', '!', ID, NUM}

Caught expected errors (Input: 'cannon {bool a = true&true;}'):
Line 1:21 - token recognition error at: '&t'
Line 1:23 - extraneous input 'rue' expecting ';'

Caught expected errors (Input: 'cannon {int a = (2+3;}'):
Line 1:20 - missing ')' at ';'

Caught expected errors (Input: 'cannon {int a == 1;}'):
Line 1:14 - no viable alternative at input 'inta=='
```

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.

```
@Test
public void testIfWhileScopes() {
    // Accept-reject tests
    accepts("cannon { while (a>0) { }}");
    accepts("cannon { while (true) { int a = 3;}}");
    accepts("cannon { while (a>b[1]) { int b = 4;}}");
    accepts("cannon { while (!false) { }}");
    accepts("cannon { if (10 > 3){ }}");
    accepts("cannon { if (a>0) { }}");
    accepts("cannon { if (a>0) { } else { }}");
    accepts("cannon { if (true||false) { int a=2; } else {int a=3;} }");
    accepts("cannon { int a = 2; {int a=3;}}");
    accepts("cannon { int a = 2; {int a=3; { a=5;}}}");
    accepts("cannon { int a = 2; {int a=3;} {int a=4;}}");
    rejects("cannon { while (a>0) int a=3;}"); // while body needs braces
    rejects("cannon { while a>0 {int a=3;}}"); // while condition needs parenthesis
    rejects("cannon { if (true) int a=3;}"); // if body needs braces
    rejects("cannon { if true {int a=3;}}"); // if condition needs parenthesis
    rejects("cannon { if (true) {int a=3;} else int a=2;}"); // else body needs braces
    rejects("cannon { if (true) {int a=3;} else (false) {int a=2;}}"); // else does not have a condition
    rejects("cannon { { int a; }"); // missing '}' to close the nested scope
    // ParseTree tests
    ParseTree tree = accepts("cannon {if (true) { a = 2;} else { a=3; }}");
    Assert.assertEquals("{if(true){a=2;}else{a=3;}}", tree.getChild(1).getText()); // check if body is the same
    Assert.assertEquals("if", tree.getChild(1).getChild(1).getChild(0).getText());
    Assert.assertEquals("true", tree.getChild(1).getChild(1).getChild(2).getText());
    Assert.assertEquals("{a=2;}", tree.getChild(1).getChild(1).getChild(4).getText());
    Assert.assertEquals("else", tree.getChild(1).getChild(1).getChild(5).getText());
    Assert.assertEquals("{a=3;}", tree.getChild(1).getChild(1).getChild(6).getText());
}
```

Figure 5. While, if and scopes syntax tests

```
Caught expected errors (Input: 'cannon { while (a>0) int a=3;}'):
Line 1:21 - missing '{' at 'int'
Line 1:30 - extraneous input '<EOF>' expecting {BOOL, INT, FORK, IF, JOIN, PRINT, SYNC, WHILE, '{', '}', ID}

Caught expected errors (Input: 'cannon { while a>0 {int a=3;}}'):
Line 1:15 - missing '(' at 'a'
Line 1:19 - missing ')' at '{'

Caught expected errors (Input: 'cannon { if (true) int a=3;}'):
Line 1:19 - missing '{' at 'int'
Line 1:28 - extraneous input '<EOF>' expecting {BOOL, INT, FORK, IF, JOIN, PRINT, SYNC, WHILE, '{', '}', ID}

Caught expected errors (Input: 'cannon { if true {int a=3;}}'):
Line 1:12 - missing '(' at 'true'
Line 1:17 - missing ')' at '{'

Caught expected errors (Input: 'cannon { if (true) {int a=3;} else int a=2;}'):
Line 1:35 - missing '{' at 'int'
Line 1:44 - extraneous input '<EOF>' expecting {BOOL, INT, FORK, IF, JOIN, PRINT, SYNC, WHILE, '{', '}', ID}

Caught expected errors (Input: 'cannon { if (true) {int a=3;} else (false) {int a=2;}}'):
Line 1:35 - mismatched input '(' expecting '{'

Caught expected errors (Input: 'cannon { { int a; }'):
Line 1:19 - extraneous input '<EOF>' expecting {BOOL, INT, FORK, IF, JOIN, PRINT, SYNC, WHILE, '{', '}', ID}
```

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.

```java
@Test
public void testThreads() {
    // Accept-reject tests
    accepts("cannon { int a; fork { int a=3;}}");
    accepts("cannon { int a; fork { {int a=3;} }}");
    accepts("cannon { int a; fork { int a=5; fork { int a=4;}}}");
    accepts("cannon { int a; fork { int a=3;} join; if(a>0) { print(a); }}");
    accepts("cannon { int a; fork { fork {int a=3;} fork{ int a=4; } join; } }");
    accepts("cannon { int shared a; fork { sync {a=3;} } sync {a=5;} }");
    rejects("cannon { int a; fork { int a=3;};}"); // ';' is not needed after fork body
    rejects("cannon { int a; fork { int a=3;} join }"); // join statement needs ';'
    rejects("cannon { int shared a; fork { a=3;} sync a=2; }"); // sync statement needs braces
    // ParseTree tests
    ParseTree tree = accepts("cannon { int shared a; fork { sync {a=3;} } }");
    Assert.assertEquals("{intshareda;fork{sync{a=3;}}}", tree.getChild(1).getText()); // check if body is the same
    Assert.assertEquals("intshareda;", tree.getChild(1).getChild(1).getText());
    Assert.assertEquals("shared", tree.getChild(1).getChild(1).getChild(1).getText());
    Assert.assertEquals("fork{sync{a=3;}}", tree.getChild(1).getChild(2).getText());
    Assert.assertEquals("fork", tree.getChild(1).getChild(2).getChild(0).getText());
    Assert.assertEquals("{sync{a=3;}}", tree.getChild(1).getChild(2).getChild(1).getText());
    Assert.assertEquals("sync", tree.getChild(1).getChild(2).getChild(1).getChild(1).getChild(0).getText());
}
```

Figure 7. Concurrency syntax tests

```
Caught expected errors (Input: 'cannon { int a; fork { int a=3;};}'):
Line 1:32 - extraneous input ';' expecting {BOOL, INT, FORK, IF, JOIN, PRINT, SYNC, WHILE, '{', '}', ID}

Caught expected errors (Input: 'cannon { int a; fork { int a=3;} join }'):
Line 1:38 - missing ';' at '}'

Caught expected errors (Input: 'cannon { int shared a; fork { a=3;} sync a=2; }'):
Line 1:41 - missing '{' at 'a'
Line 1:47 - extraneous input '<EOF>' expecting {BOOL, INT, FORK, IF, JOIN, PRINT, SYNC, WHILE, '{', '}', ID}
```

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.

```
@Test
public void testArrays() {
    // Accept-reject tests
    accepts("cannon { int a[3];}");
    accepts("cannon { bool a[2] = [true,false];}");
    accepts("cannon { a[3] = 5*8-10;}");
    accepts("cannon { int a[3] = b;}");
    accepts("cannon { int a = b[4]*10;}");
    accepts("cannon { print(a[4]);}");
    rejects("cannon { int a[] = [3,2];}"); // array size must be specified
    rejects("cannon { int [3]a;}"); // array size is specified after the identifier
    rejects("cannon { int a[2 = [1,4];}"); // missing ']'
    rejects("cannon { int a[3][3];}"); // only one-dimension arrays are supported
    // ParseTree tests
    ParseTree tree = accepts("cannon { bool a[2] = [true,false];}");
    Assert.assertEquals("{boola[2]=[true,false];}", tree.getChild(1).getText()); // check if body is the same
    Assert.assertEquals("bool", tree.getChild(1).getChild(1).getChild(0).getText());
    Assert.assertEquals("a", tree.getChild(1).getChild(1).getChild(1).getText());
    Assert.assertEquals("2", tree.getChild(1).getChild(1).getChild(3).getText());
    Assert.assertEquals("[true,false]", tree.getChild(1).getChild(1).getChild(6).getText());
}
```

Figure 9. Arrays syntax tests

```
Caught expected errors (Input: 'cannon { int a[] = [3,2];}'):
Line 1:15 - missing NUM at ']'

Caught expected errors (Input: 'cannon { int [3]a;}'):
Line 1:13 - no viable alternative at input 'int['

Caught expected errors (Input: 'cannon { int a[2 = [1,4];}'):
Line 1:17 - missing ']' at '='

Caught expected errors (Input: 'cannon { int a[3][3];}'):
Line 1:17 - mismatched input '[' expecting {'=', ';'}
```

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.

```
@Test
public void testProcedures() {
    // Accept-reject tests
    accepts("pickle p1() { } cannon {int a=2;}");
    accepts("pickle p1(int a) { print(a); } cannon { p1(3);}");
    accepts("pickle p1() { } pickle p2(int a){ }  cannon {p1();}");
    accepts("pickle p1() { } pickle p2(int a, bool b[4]){ p1();}  cannon {p2(4, [false,true,false,false]);}");
    rejects("p1() { } cannon {int a=2;}"); // each procedure needs to start with 'pickle' keyword
    rejects("pickle 1p() { } cannon {int a=2;}"); // each procedure name needs to start with a letter
    rejects("pickle p1(int a; int b;) { } cannon {int a=2;}"); // parameters need to be separated by ','
    rejects("cannon {int a=2;} pickle p1() { } "); // procedures are declared before main body
    rejects("pickle p1 { } cannon {int a=2;}"); // each procedure needs to have parenthesis
    rejects("pickle p1(int a[]) { } cannon {int a=2;}"); // size of array need to be specified in procedure parameters
    rejects("pickle p1() { return 5;} cannon {int a=2;}"); // return functionality is not-supported
    rejects("pickle p1() { } cannon { p1;}"); // procedure calls need parenthesis
    // ParseTree tests
    ParseTree tree = accepts("pickle p1(int a) { print(a); } cannon { p1(3);}");
    Assert.assertEquals("picklep1(inta){print(a);}", tree.getChild(0).getText());
    Assert.assertEquals("{p1(3);}", tree.getChild(2).getText());
    Assert.assertEquals("pickle", tree.getChild(0).getChild(0).getText());
    Assert.assertEquals("p1", tree.getChild(0).getChild(1).getText());
    Assert.assertEquals("inta", tree.getChild(0).getChild(3).getText());
    Assert.assertEquals("{print(a);}", tree.getChild(0).getChild(5).getText());
    Assert.assertEquals("p1", tree.getChild(2).getChild(1).getChild(0).getText());
    Assert.assertEquals("(3)", tree.getChild(2).getChild(1).getChild(1).getText());
}
```

Figure 11. Procedures syntax tests

```
Caught expected errors (Input: 'p1() { } cannon {int a=2;}'):
Line 1:0 - mismatched input 'p1' expecting {PICKLE, CANNON}

Caught expected errors (Input: 'pickle 1p() { } cannon {int a=2;}'):
Line 1:7 - extraneous input '1' expecting ID

Caught expected errors (Input: 'pickle p1(int a; int b;) { } cannon {int a=2;}'):
Line 1:15 - mismatched input ';' expecting {',', ')'}

Caught expected errors (Input: 'cannon {int a=2;} pickle p1() { } '):
Line 1:18 - mismatched input 'pickle' expecting <EOF>

Caught expected errors (Input: 'pickle p1 { } cannon {int a=2;}'):
Line 1:10 - mismatched input '{' expecting '('

Caught expected errors (Input: 'pickle p1(int a[]) { } cannon {int a=2;}'):
Line 1:16 - missing NUM at ']'

Caught expected errors (Input: 'pickle p1() { return 5;} cannon {int a=2;}'):
Line 1:21 - no viable alternative at input 'return5'

Caught expected errors (Input: 'pickle p1() { } cannon { p1;}'):
Line 1:27 - no viable alternative at input 'p1;'
```

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.

```
@Test
public void testFiles() {
    accepts("src/tests/syntax/testSources/isPrimeExample.pickle", true);
    accepts("src/tests/syntax/testSources/fibonacciExample.pickle", true);
    rejects("src/tests/syntax/testSources/syntaxError.pickle", true);
}
```

Figure 13. Syntax tests for programs written in files

```
pickle isPrime(int number){
    if(number<=1){
        print(false);
    }
    else{
        int div = 2;
        int sum = 0;
        while(div < number){
            int div1 = number/div;
            if(div1*div==number){
                sum=sum+1;
            }
            div=div+1;
        }
        if(sum==0){
            print(true);
        }
        else{
            print(false);
        }
    }
}
cannon{
    isPrime(2);
    isPrime(13);
    isPrime(33);
    isPrime(100);
}
```

Figure 14. isPrimeExample.pickle source code

```
pickle fib(int n){
    if(n==0||n==1){
        print(1);
    }
    else{
        int sum = 2;
        int prev1 = 1;
        int prev2 = 1;
        while(n>2){
            prev1=prev2;
            prev2=sum;
            sum=prev2+prev1;
            n=n-1;
        }
        print(sum);
    }
}
cannon{
    fib(1);
    fib(2);
    fib(3);
    fib(4);
    fib(40);

}
```

Figure 15. fibonacciExample.pickle source code

```
pickle p1() {
    int 1a;
    bool = 3;
}

cannon
{
    while(true)
        int a=3;
}
```

Figure 16. syntaxError.pickle source code

```
Caught expected errors (Input: 'src/tests/syntax/testSources/syntaxError.pickle'):
Line 2:5 - no viable alternative at input 'int1'
Line 9:2 - missing '{' at 'int'
Line 10:1 - extraneous input '<EOF>' expecting {BOOL, INT, FORK, IF, JOIN, PRINT, SYNC, WHILE, '{', '}', ID}
```

Figure 17. Caught expected error for program written in file

JUnit results

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

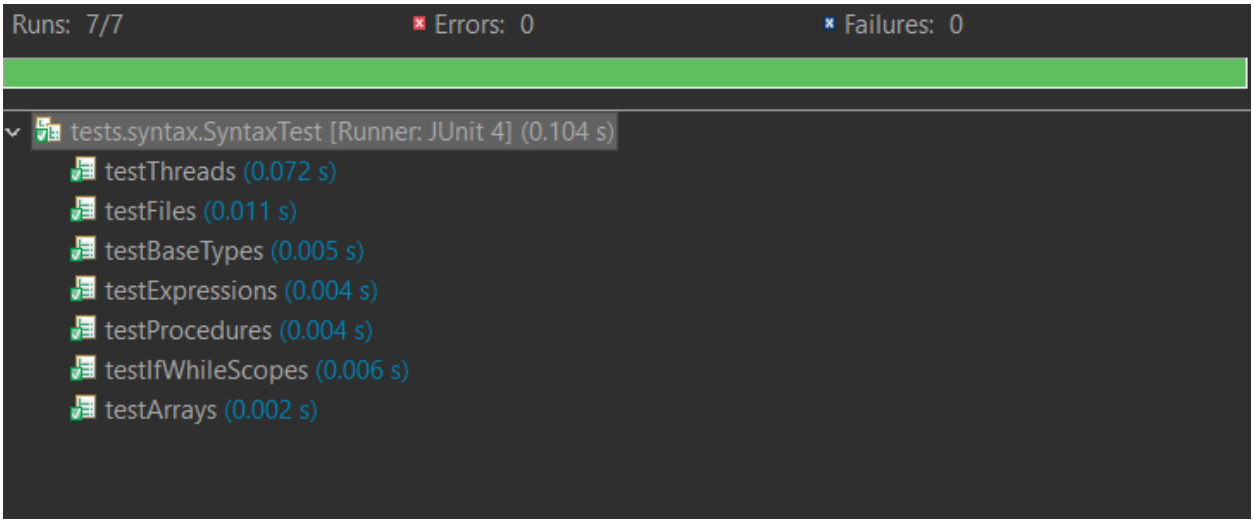


Figure 18. Syntax tests Junit results

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.

```
@Test
public void testBasicTypes() {
    accepts("cannon { int a;}");
    accepts("cannon { int shared a=2;}");
    accepts("cannon { int a=-12;}");
    accepts("cannon { bool shared b;}");
    accepts("cannon { bool a = true;}");
    rejects("cannon { int a = true; }"); // int can only be a number
    rejects("cannon { bool a=1;}"); // bool can be only true or false
}
```

Figure 19. Basic types context tests

```
Caught expected errors (Input 'cannon { int a = true; }'):
Line 1:9 - Expected type 'bool' but found 'int'

Caught expected errors (Input 'cannon { bool a=1;}'):
Line 1:9 - Expected type 'int' but found 'bool'
```

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.

```
@Test
public void testDeclarations() {
    accepts("cannon { int a; { int a;}}");
    accepts("cannon { int a=2; if(a>2) { bool a = false;} }");
    accepts("cannon { int a=2; if(a>2) { bool a = false;} else { int a[3];} }");
    rejects("cannon { int a; b=3;}"); // Variable b has not been declared
    rejects("cannon { int a; int a=3;}"); // Variable a has already been declared
    rejects("cannon { int a; {int shared b=3;}}"); // Shared variables can be declared only in the outer global
                                                   // scope
    rejects("cannon { a=3; int a=2;}"); // Variable a has not been declared
    rejects("cannon { int a=3; {a=2; bool a = true;}}"); // Variable a has already been declared (because of use)
    rejects("cannon { int a=3; {int b=2;} b=3;}"); // Variable b has not been declared in the scope
}
```

Figure 21. Declarations context tests

```
Caught expected errors (Input 'cannon { int a; b=3;}'):
Line 1:16 - Variable 'b' not declared in this scope

Caught expected errors (Input 'cannon { int a; int a=3;}'):
Line 1:16 - Variable 'a' is already declared in this scope

Caught expected errors (Input 'cannon { int a; {int shared b=3;}}'):
Line 1:17 - Variable 'b' can be declared shared only in cannon outer scope

Caught expected errors (Input 'cannon { a=3; int a=2;}'):
Line 1:9 - Variable 'a' not declared in this scope

Caught expected errors (Input 'cannon { int a=3; {a=2; bool a = true;}}'):
Line 1:24 - Variable 'a' is already declared in this scope

Caught expected errors (Input 'cannon { int a=3; {int b=2;} b=3;}'):
Line 1:29 - Variable 'b' not declared in this scope
```

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.

```
//@Test
public void testExpressions() {
    accepts("cannon { int a=2+3-9;}");
    accepts("cannon { int a=2+2*3+1/-2;}");
    accepts("cannon { int a=2+(8+3)*(1+3)/2;}");
    accepts("cannon { bool a=true&&false;}");
    accepts("cannon { bool a = 2==2;}");
    accepts("cannon { bool a=true||false;}");
    accepts("cannon { bool a=true&&(false||true);}");
    rejects("cannon { int a=2+true;}"); // addition must use int type
    rejects("cannon { bool a = 1&&1;}"); // AND must use bool type
    rejects("cannon { bool a = 2==true;}"); // mismatched types
    rejects("cannon { bool a = 3>=true;}"); // mismatched types
}
```

Figure 23. Expressions context tests

```
Caught expected errors (Input 'cannon { int a=2+true;}'):
Line 1:17 - Expected type 'int' but found 'bool'

Caught expected errors (Input 'cannon { bool a = 1&&1;}'):
Line 1:18 - Expected type 'bool' but found 'int'
Line 1:21 - Expected type 'bool' but found 'int'

Caught expected errors (Input 'cannon { bool a = 2==true;}'):
Line 1:21 - Expected type 'int' but found 'bool'

Caught expected errors (Input 'cannon { bool a = 3>=true;}'):
Line 1:21 - Expected type 'int' but found 'bool'
```

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.

```
@Test
public void testIfWhile() {
    accepts("cannon { if(2>3) {print (3);}}");
    accepts("cannon { int a; if(a>0) { print (a); } else { print(0);}}");
    accepts("cannon { if(true) {print(false);} else {print(true);}}");
    accepts("cannon { int a=5; while(a>0) {a=a-1;} print(a);}");
    accepts("cannon { while(false) {int a=3;} print(2);}");
    rejects("cannon { if (2) {print (3);}}"); // if condition must be a bool
    rejects("cannon { if (true) {int a=3;} print(a);}"); // variable 'a' has not been declared in the outer scope
    rejects("cannon { if (2+3*9/3) {print (3);}}"); // if condition must be a bool
    rejects("cannon { while(0) {print (1);}}"); // while condition must be a bool
    rejects("cannon { while(false||true) {int a=3;} print(a); }"); // 'a' has not been declared in the outer scope
}
```

Figure 25. While, if context tests

```
Caught expected errors (Input 'cannon { if (2) {print (3);}}'):
Line 1:13 - Expected type 'bool' but found 'int'

Caught expected errors (Input 'cannon { if (true) {int a=3;} print(a);}'):
Line 1:36 - Variable 'a' not declared in this scope

Caught expected errors (Input 'cannon { if (2+3*9/3) {print (3);}}'):
Line 1:13 - Expected type 'bool' but found 'int'

Caught expected errors (Input 'cannon { while(0) {print (1);}}'):
Line 1:15 - Expected type 'bool' but found 'int'

Caught expected errors (Input 'cannon { while(false||true) {int a=3;} print(a); }'):
Line 1:45 - Variable 'a' not declared in this scope
```

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.

```
@Test
public void testThreads() {
    accepts("cannon { fork {print (2);} print(1);}");
    accepts("cannon { fork { fork { print(3);} print(2); } print(1);}");
    accepts("cannon { int shared a=2; fork { sync { a=3;}} a=1;}");
    accepts("cannon { int shared a=2; fork { sync { a=a+2;}} join; a=a+1;}");
    accepts("cannon { int shared a=2; fork { fork { sync {a=a+3;} } join; } print(a);}");
    rejects("cannon { int a=3; fork {a=2;} }"); // 'a' must be declared shared
    rejects("cannon { while(true) { fork {print (3);} } }"); // cannot fork inside a while loop
    rejects("cannon { fork { int shared a=3; }}"); // shared variables can only be declared in outer global scope
    rejects("cannon { fork { int a=3; } print(a); }"); // 'a' has not been declared in outer scope
}
```

Figure 27. Concurrency context tests

```
Caught expected errors (Input 'cannon { int a=3; fork {a=2;} }'):
Line 1:24 - Variable 'a' not declared in this scope or is not shared

Caught expected errors (Input 'cannon { while(true) { fork {print (3);} } }'):
Line 1:23 - Cannot fork a thread inside a function, a while loop or if-else statement

Caught expected errors (Input 'cannon { fork { int shared a=3; }}'):
Line 1:16 - Variable 'a' can be declared shared only in cannon outer scope

Caught expected errors (Input 'cannon { fork { int a=3; } print(a); }'):
Line 1:33 - Variable 'a' not declared in this scope
```

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.

```java
@Test
public void testArrays() {
    accepts("cannon { int a[3];}");
    accepts("cannon { bool a[2]=[true,false];}");
    accepts("cannon { int a[3]; a[0+1]=3*8;}");
    accepts("cannon { int a[3]=[2,3,1]; bool c = a==[3,2];}");
    accepts("cannon { int a[3]=[1,2,3]; int b[3]=a;}");
    accepts("cannon { int shared a[3];}");
    rejects("cannon { int a[0]; }"); // array size must be greater than 0
    rejects("cannon { int a[3] = [1,2,false];}"); // all array elements must be of the declared type
    rejects("cannon { int a[3]=[3,2,1]; int b[2]=a;}"); //during assignment array sizes must match
    rejects("cannon { int a[3]; bool b[3]; bool c = a==b;}"); //mismatched array types
    rejects("cannon { int a[2+1];}"); //array size must be declared as a number, not expression
    // NOTE: there is no detection for trying to access elements out of array bounds
}
```

Figure 29. Arrays context tests

```
Caught expected errors (Input 'cannon { int a[0]; }'):
Line 1:9 - Array 'a' size must be greater than 0

Caught expected errors (Input 'cannon { int a[3] = [1,2,false];}'):
Line 1:25 - Expected type 'int' but found 'bool'

Caught expected errors (Input 'cannon { int a[3]=[3,2,1]; int b[2]=a;}'):
Line 1:36 - Expected type 'int[2]' but found 'int[3]'

Caught expected errors (Input 'cannon { int a[3]; bool b[3]; bool c = a==b;}'):
Line 1:42 - Expected type 'int[]' but found 'bool[]'

Caught expected errors (Input 'cannon { int a[2+1];}'):
Line 1:16 - mismatched input '+' expecting ']'
```

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.

```java
@Test
public void testProcedures() {
    accepts("pickle p1() { p1(); } cannon { p1(); }");
    accepts("pickle p1(int a) { print(a);} cannon { p1(3); }");
    accepts("pickle p1() { p2();} pickle p2() { print(0); } cannon { p1(); }");
    accepts("pickle p1(int a[4]) { print(a);} cannon { p1([1,2,3,4]); }");
    accepts("pickle p1(int a) { print(a);} cannon { fork { p1(3); } }");
    rejects("pickle p1(int a) {} cannon { p1(); }"); // missing argument in procedure call
    rejects("pickle p1(int a) {} cannon { p1(false); }"); // mismatched argument type in procedure call
    rejects("pickle p1(int a[3]) {} cannon { p1([2,3]); }"); // mismatched argument type in procedure call
    rejects("pickle p1(int a) {} pickle p1() {} cannon { p1(false); }"); // duplicate procedure names
    rejects("pickle p1(bool a) { int a=3;} cannon { p1(false); }"); // 'a' already declared in the scope
    rejects("pickle p1(bool a) { int shared b=3;} cannon { p1(false); }"); // shared variables cannot be declared in
                                                                          // procedure
    rejects("pickle p1(bool a) { fork { print(3); } } cannon { p1(false); }"); // cannot fork in the procedure
}
```

Figure 31. Procedures context tests

```
Caught expected errors (Input 'pickle p1(int a[3]) {} cannon { p1([2,3]); }'):
Line 1:32 - Expected type 'Proc [int[3]]' but found 'Proc [int[2]]'

Caught expected errors (Input 'pickle p1(int a) {} pickle p1() {} cannon { p1(false); }'):
Line 1:20 - Function 'p1' is already declared in this scope
Line 1:44 - Expected type 'Proc [int]' but found 'Proc [bool]'

Caught expected errors (Input 'pickle p1(bool a) { int a=3;} cannon { p1(false); }'):
Line 1:20 - Variable 'a' is already declared in this scope

Caught expected errors (Input 'pickle p1(bool a) { int shared b=3;} cannon { p1(false); }'):
Line 1:20 - Variable 'b' can be declared shared only in cannon outer scope

Caught expected errors (Input 'pickle p1(bool a) { fork { print(3); } } cannon { p1(false); }'):
Line 1:20 - Cannot fork a thread inside a function, a while loop or if-else statement
```

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.

```java
@Test
public void testFiles() {
    accepts("src/tests/context/testSources/fork.pickle", true);
    accepts("src/tests/context/testSources/arrayCorrect.pickle", true);
    accepts("src/tests/context/testSources/procCorrect.pickle", true);
    rejects("src/tests/context/testSources/arrayError.pickle", true);
    rejects("src/tests/context/testSources/procError.pickle", true);
    rejects("src/tests/context/testSources/sharedError.pickle", true);
    rejects("src/tests/context/testSources/forkError.pickle", true);
    rejects("src/tests/context/testSources/forkSharedError.pickle", true);
    rejects("src/tests/context/testSources/error1.pickle", true);
    rejects("src/tests/context/testSources/error2.pickle", true);
    rejects("src/tests/context/testSources/error3.pickle", true);

}
```

Figure 33. Context tests for programs written in files

```
cannon
    {
        int shared a = 0;
        int shared b = 0;
        int shared c = 0;
        int e;
        int f;
        fork {
            a=2;
        }
        fork {
            sync{
                b=3;
            }
        }
        c=4;
        join;
        int d=a+b+c;
        if(d==9){
            print(d);
        }
        else{
            print(false);
        }
}
```

Figure 34. fork.pickle source code

```
cannon{
    int a = 3;
    int b[3]=[0,0,5];
    bool c[3]=[false,true,true];
    while(c[0]!=c[1]){
        if(b[2]!=0){
            if(b[2]>0){
                b[2]=b[2]-1;
            }
            else{
                b[2]=b[2]+1;
            }
        }
        else{
            c[0]=true;
            c[1]=true;
        }
    }
    int d[2];
    bool e = d==b;
    print(a);
    print(b);
    print(c);
}
```

Figure 35. arrayCorrect.pickle source code

```
pickle p1 (int a, bool c[3]){
    a=3;
    c[2]=false;
    int b = 2;
    print(b+a);
}

pickle p2 (){
    int e=3;
    int k=5;
    k=k*e;
    print(k);
}

cannon{
    int shared c=2;
    fork{
        int a=3;
        print(a);
    }
    join;
    int a=2;
    print(a);
}
```

Figure 36. procCorrect.pickle source code

```
cannon{
    int a[3]=[0,1];
    int b[4]=[0,0,false,1];
    int c[3]=[10+3,4,8*2+3];

    int d[2]=[0,0,0,0];
    int a=3;
    int b = a[0];
    int l=3;
    bool e[2]=[true==true,false];
    {
        int a[1]=[4];
        bool b[2]=[false,false];
        bool c[3]=[false,true,false];

    }
    c[2]=10;
    c[2+1]=false;
    int f[2];
    int g[0];
    v[1]=3;
    l[2]=1;
}
```

Figure 37. arrayError.pickle source code

```
pickle p2(int a, bool b[3], int c[3]){
    print(b);
}
pickle p1(int a){
}
cannon {
    p1();
    p3(3);
    p2(2,2,[2,3,4]);
    p2(0,[true,false,true],[2,1,3]);
    p1(3);
}
```

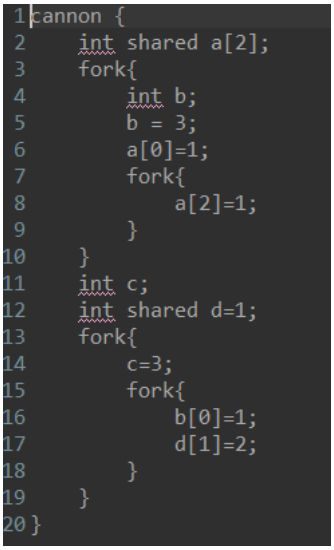
Figure 38. procError.pickle source code

```
pickle p1(){
    int shared a;
}

cannon{
    int a;
    int shared a[2];
    int shared b[3]=[0,2,3];
    int b=4;
    {
        int shared f;
        {
            int b=3;
        }
        int b=2;
    }
}
```

Figure 39. sharedError.pickle source code

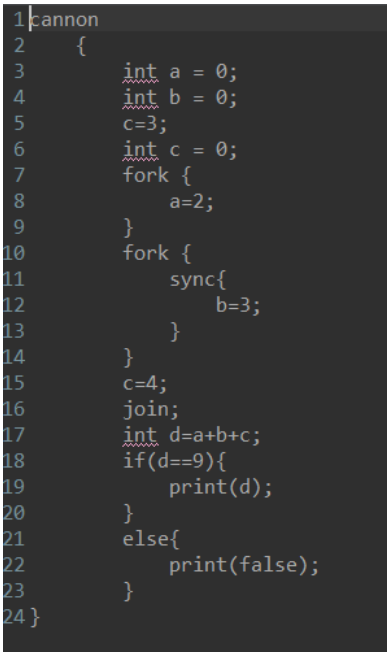
```
pickle p1(){
    fork{
        int a;
    }
}

cannon{
    fork{
        int a;
        fork{
            int b;
        }
    }

    while(true){
        fork{
            int a;
            fork{
                int b;
            }
        }
    }

    fork{
        while(true){
        }
        while(true){
            fork{
            }
        }
    }
}
```

Figure 40. forkError.pickle source code



```
cannon {
    int shared a[2];
    fork{
        int b;
        b = 3;
        a[0]=1;
        fork{
            a[2]=1;
        }
    }
    int c;
    int shared d=1;
    fork{
        c=3;
        fork{
            b[0]=1;
            d[1]=2;
        }
    }
}
```

Figure 41. forkSharedError.pickle source code



```
cannon
    {
        int a = 0;
        int b = 0;
        c=3;
        int c = 0;
        fork {
            a=2;
        }
        fork {
            sync{
                b=3;
            }
        }
        c=4;
        join;
        int d=a+b+c;
        if(d==9){
            print(d);
        }
        else{
            print(false);
        }
}
```

Figure 42. error1.pickle source code

```
cannon
    {
        int a = 0;
        int b = 0;
        int c;
        d=a+b+c;
        {
            int e=0;
        }
        a=e;
}
```

Figure 43. error2.pickle source code

```
cannon
    {
        int a = 0;
        bool b = true;
        int c = a+b;
        {
            b=true;
            int b = 2;
        }
}
```

Figure 44. error3.pickle source code

```
Caught expected errors (Input 'src/tests/context/testSources/arrayError.pickle'):
Line 2:10 - Expected type 'int[3]' but found 'int[2]'
Line 3:15 - Expected type 'int' but found 'bool'
Line 6:10 - Expected type 'int[2]' but found 'int[4]'
Line 7:1 - Variable 'a' is already declared in this scope
Line 8:1 - Variable 'b' is already declared in this scope
Line 18:1 - Expected type 'bool' but found 'int'
Line 20:1 - Array 'g' size must be greater than 0
Line 21:1 - Array 'v' not declared in this scope
Line 22:1 - Variable 'l' is not an array

Caught expected errors (Input 'src/tests/context/testSources/procError.pickle'):
Line 7:1 - Expected type 'Proc [int]' but found 'Proc []'
Line 8:1 - Pickle 'p3' is not declared
Line 9:1 - Expected type 'Proc [int, bool[3], int[3]]' but found 'Proc [int, int, int[3]]'

Caught expected errors (Input 'src/tests/context/testSources/sharedError.pickle'):
Line 2:1 - Variable 'a' can be declared shared only in cannon outer scope
Line 7:1 - Variable 'a' is already declared in this scope
Line 9:1 - Variable 'b' is already declared in this scope
Line 11:2 - Variable 'f' can be declared shared only in cannon outer scope

Caught expected errors (Input 'src/tests/context/testSources/forkError.pickle'):
Line 2:1 - Cannot fork a thread inside a function, a while loop or if-else statement
Line 16:2 - Cannot fork a thread inside a function, a while loop or if-else statement
Line 18:3 - Cannot fork a thread inside a function, a while loop or if-else statement
Line 28:3 - Cannot fork a thread inside a function, a while loop or if-else statement

Caught expected errors (Input 'src/tests/context/testSources/forkSharedError.pickle'):
Line 14:2 - Variable 'c' not declared in this scope or is not shared
Line 16:3 - Variable 'b' not declared in this scope or is not shared
Line 17:3 - Variable 'd' is not an array
```

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

```
Caught expected errors (Input 'src/tests/context/testSources/error1.pickle'):
Line 5:2 - Variable 'c' not declared in this scope
Line 8:3 - Variable 'a' not declared in this scope or is not shared
Line 12:7 - Variable 'b' not declared in this scope or is not shared

Caught expected errors (Input 'src/tests/context/testSources/error2.pickle'):
Line 6:2 - Variable 'd' not declared in this scope
Line 10:4 - Variable 'e' not declared in this scope

Caught expected errors (Input 'src/tests/context/testSources/error3.pickle'):
Line 5:12 - Expected type 'int' but found 'bool'
Line 8:3 - Variable 'b' is already declared in this scope
```

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

JUnit results

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

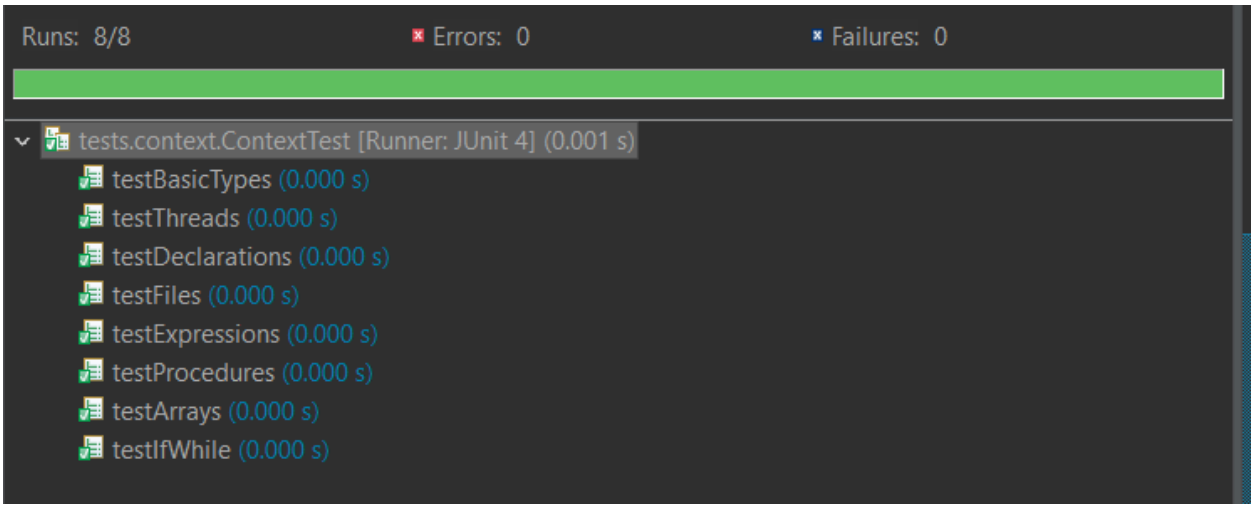


Figure 47. Context tests JUnit results

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.

```
@Test
public void testTypesAndAssignments() {
    check("cannon { int a = 203; print(a); }", "typesAndAssignments1", "Sprockell 0 says 203");
    check("cannon { bool a = true; print(a); }", "typesAndAssignments2", "Sprockell 0 says 1");
    check("cannon { int a = 1; { bool a = true; print(a); } }", "typesAndAssignments3", "Sprockell 0 says 1");
    check("cannon { int a[3] = [10,5,100]; print(a); }", "typesAndAssignments4",
            "{\nSprockell 0 says 10\nSprockell 0 says 5\nSprockell 0 says 100\n}");
    check("cannon { int a[3] = [10,5,100]; int b[3]=a; b[1]=3; print(a[1]); }", "typesAndAssignments5",
            "Sprockell 0 says 5");
}
```

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.

```
@Test
public void testExpressions() {
    check("cannon { int a = 2+3*4; print(a); }", "expr1", "Sprockell 0 says 14");
    check("cannon { int a = (7+8)*2-9+3*2; print(a); }", "expr2", "Sprockell 0 says 27");
    check("cannon { int a = (3-8)/2; print(a); }", "expr3", "Sprockell 0 says -2");
    check("cannon { int a[3] = [10,4,9]; int b[3] = [10,3,9]; print(a==b); }", "expr4", "Sprockell 0 says 0");
    check("cannon { print(!true==false&&[3,2,1]==[3,2,1]); }", "expr5", "Sprockell 0 says 1");
}
```

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.

```
@Test
public void testWhileIf() {
    check("cannon { int a = 10; int sum = 0; while (a>0) { if(a>5) { sum = sum + a;} a=a-1; } print(sum);}",
            "whileIf1", "Sprockell 0 says 40");
    check("cannon { if(false!=false) { print (10); } else { print(100); } }", "whileIf2", "Sprockell 0 says 100");
    check("cannon {  while (false) { print(2); } print(1);}", "whileIf3", "Sprockell 0 says 1");
}
```

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.

```
@Test
public void testConcurrency() {
    check("src/tests/semantics/testSources/bankExample.pickle", true, "bankExample", "Sprockell 0 says 11000");
    check("src/tests/semantics/testSources/petersonsExample.pickle", true, "petersonsExample",
            "Sprockell 0 says 30");
    check("src/tests/semantics/testSources/nestedThreadsExample.pickle", true, "nestedThreadsExample",
            "Sprockell 2 says 3\nSprockell 1 says 2\nSprockell 0 says 1");
}
```

Figure 51. Concurrency sematic tests

```
1  cannon{
2      int shared balance = 20000;
3      fork{
4          int a=10;
5          while(a>0){
6              sync{
7                  balance = balance-200;
8              }
9              a=a-1;
10         }
11     }
12
13     fork{
14         int a=10;
15         while(a>0){
16             sync{
17                 balance = balance+300;
18             }
19             a=a-1;
20         }
21     }
22
23     fork{
24         int a=10;
25         while(a>0){
26             sync{
27                 balance = balance-1000;
28             }
29             a=a-1;
30         }
31     }
32
33     join;
34     print(balance);
35 }
```

Figure 52. bankExample.pickle source code

```
cannon{
    bool shared flag0 = false;
    bool shared flag1 = false;
    int shared turn;
    int shared counter=0;
    fork{
        int c = 10;
        while(c>0){
            flag0=true;
            turn = 1;
            while(flag1 == true && turn == 1){
                /*busy wait*/
            }
            /*critical section*/
            counter=counter+1;
            /*end of critical section*/
            flag0=false;
            c=c-1;
        }
    }
    fork{
        int c = 10;
        while(c>0){
            flag1=true;
            turn = 0;
            while(flag0 == true && turn == 0){
                /*busy wait*/
            }
            /*critical section*/
            counter=counter+2;
            /*end of critical section*/
            flag1=false;
            c=c-1;
        }
    }
    join;
    print(counter);
}
```

Figure 53. petersonsExample.pickle source code

```
cannon{
    int a=1;
    fork{
        int a = 2;
        fork{
            int a =3;
            print(a);
        }
        join;
        print(a);
    }
    join;
    print(a);
}
```

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, Figure 61 and Figure 62. Generated Sprockell instruction files can be found in Table 17, Table 18, Table 19, Table 20, Table 21, Table 22 and Table 23. *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.

```
@Test
public void testGeneralAlgorithms() {
    check("src/tests/semantics/testSources/februaryDaysExample.pickle", true, "februaryDaysExample",
            "Sprockell 0 says 29\nSprockell 0 says 28\nSprockell 0 says 29\nSprockell 0 says 28\nSprockell 0 says 29");
    check("src/tests/semantics/testSources/isPrimeExample.pickle", true, "isPrimeExample",
            "Sprockell 0 says 1\nSprockell 0 says 1\nSprockell 0 says 0\nSprockell 0 says 0");
    check("src/tests/semantics/testSources/arrayConcurrentSumExample.pickle", true, "arrayConcurrentSumExample",
            "Sprockell 0 says 5");
    check("src/tests/semantics/testSources/fibonacciExample.pickle", true, "fibonacciExample",
            "Sprockell 0 says 1\nSprockell 0 says 2\nSprockell 0 says 3\nSprockell 0 says 5\nSprockell 0 says 165580141");
    check("src/tests/semantics/testSources/arrayMaxExample.pickle", true, "arrayMaxExample",
            "Sprockell 0 says 100\nSprockell 0 says -3\nSprockell 0 says 1000000");
    check("src/tests/semantics/testSources/boolArrayExample.pickle", true, "boolArrayExample",
            "Sprockell 0 says 1\nSprockell 0 says 0\nSprockell 0 says 1");
    check("src/tests/semantics/testSources/extendedProgram.pickle", true, "extendedProgram",
            "Sprockell 0 says 4\nSprockell 0 says 1\nSprockell 0 says 10\nSprockell 0 says 9\nSprockell 0 says 24");
}
```

Figure 55. General algorithms semantic tests

```
/* February has 29 days if it is a leap year, otherwise 28 days*/
/* Leap year is a year that is a multiple of 4, but if it is also the multiple of 100,
   then it also has to be a multiple of 400, otherwise it is not a leap year*/
pickle days(int year){
    int div1 = year/4;
    int div2 = year/100;
    int div3 = year/400;
    if(div1*4==year){
        if(div2*100==year){
            if(div3*400==year){
                print(29);
            }
            else{
                print(28);
            }
        }
        else{
            print(29);
        }
    }
    else{
        print(28);
    }
}
cannon{
    days(2012);
    days(1055);
    days(2000);
    days(1700);
    days(1040);
}
```

Figure 56. februaryDaysExample.pickle source code

```
pickle isPrime(int number){
    if(number<=1){
        print(false);
    }
    else{
        int div = 2;
        int sum = 0;
        while(div < number){
            int div1 = number/div;
            if(div1*div==number){
                sum=sum+1;
            }
            div=div+1;
        }
        if(sum==0){
            print(true);
        }
        else{
            print(false);
        }
    }
}
cannon{
    isPrime(2);
    isPrime(13);
    isPrime(33);
    isPrime(100);
}
```

Figure 57. isPrimeExample.pickle source code

```
cannon {
    int shared a[4] = [10,5,9,-19];
    int shared sum = 0;
    fork{
        int partialSum = a[0]+a[1];
        sync{
            sum=sum+partialSum;
        }
    }
    fork
    {
        int partialSum = a[2]+a[3];
        sync{
            sum=sum+partialSum;
        }
    }
    join;
    print(sum);
}
```

Figure 58. arrayConcurrentSumExample.pickle source code

```
pickle fib(int n){
    if(n==0||n==1){
        print(1);
    }
    else{
        int sum = 2;
        int prev1 = 1;
        int prev2 = 1;
        while(n>2){
            prev1=prev2;
            prev2=sum;
            sum=prev2+prev1;
            n=n-1;
        }
        print(sum);
    }
}
cannon{
    fib(1);
    fib(2);
    fib(3);
    fib(4);
    fib(40);

}
```

Figure 59. fibonacciExample.pickle source code

```
pickle p1(int array[10]){
    int max = array[0];
    int i=1;
    while(i<10){
        if(array[i]>max){
            max = array[i];
        }
        i=i+1;
    }
    print(max);
}
cannon {
    p1([1,-3,10,9,-4,-19,100,45,99, 10]);
    p1([-6711,-3,-90,-13,-4,-19,-11,-12,-100/2, -8*5]);
    p1([1*9-4,8*2,10*0,2*124, 10000-10000, 89*2, 10*10*10, 53,99--9, --1000000]);
}
```

Figure 60. arrayMaxExample.pickle source code

```
pickle areAllTrue(bool a[5]){
    int i =0;
    bool result = true;
    while(i <= 4){
        result = a[0]&&result;
        i=i+1;
    }
    print(result);
}
cannon {
    bool a[5];
    areAllTrue([true,true,true,true,true]);
    areAllTrue(a);
    areAllTrue([true!=false, false==false, true||false, true&&true, !false]);
}
```

Figure 61. boolArrayExample.pickle source code

```
pickle positiveArray(int a[6]){
    int i=0;
    while(i<=5){
        if(a[i]>0){
            print(a[i]);
        }
        i=i+1;
    }
}

cannon {
    int shared total = 0;
    int shared a[6] = [4,-2,1,-3,10,9];
    fork {
        int i=0;
        int sum = 0;
        while(i<3){
            if(a[i]>0){
                sum=sum+a[i];
            }
            i=i+1;
        }
        sync {
            total = total + sum;
        }
    }
    fork {
        int i=3;
        int sum =0;
        while(i<6){
            if(a[i]>0){
                sum=sum+a[i];
            }
            i=i+1;
        }
        sync {
            total = total + sum;
        }
    }
    join;
    positiveArray(a);
    print(total);
}
```

Figure 62. extendedProgram.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]
```

Table 23. Generated extendedProgram.hs file

```
import Sprockell

prog :: [Instruction]
prog = [
   Jump (Abs (52))
 , Load (ImmValue (6)) regB
 , Compute Sub regA regB regSP
 , Load (ImmValue (0)) regB
 , Load (ImmValue (6)) regC
 , Compute Sub regA regC regC
 , Store regB (IndAddr regC)
 , Load (ImmValue (6)) regB
 , Compute Sub regA regB regB
 , Load (IndAddr regB) regB
 , Load (ImmValue (5)) regC
 , Compute LtE regB regC regB
 , Compute Equal regB reg0 regC
 , Branch regC (Abs (44))
 , Load (ImmValue (6)) 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 (0)) regB
 , Compute Gt regC regB regC
 , Compute Equal regC reg0 regB
 , Branch regB (Abs (34))
 , Load (ImmValue (6)) 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
 , WriteInstr regC numberIO
 , Jump (Abs (34))
 , Nop
 , Load (ImmValue (6)) regB
 , Compute Sub regA regB regB
 , Load (IndAddr regB) regB
 , Load (ImmValue (1)) regC
 , Compute Add regB regC regB
```

```
 , Load (ImmValue (6)) regC
 , Compute Sub regA regC regC
 , Store regB (IndAddr regC)
 , Jump (Abs (7))
 , 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
 , 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 (0)) regB
 , WriteInstr regB (DirAddr (3))
 , Load (ImmValue (4)) regB
 , Push regB
 , Load (ImmValue (2)) regB
 , Load (ImmValue (-1)) regC
 , Compute Mul regC regB regC
 , Push regC
 , Load (ImmValue (1)) regB
 , Push regB
 , Load (ImmValue (3)) regB
 , Load (ImmValue (-1)) regC
 , Compute Mul regC regB regC
 , Push regC
 , Load (ImmValue (10)) regB
 , Push regB
 , Load (ImmValue (9)) regB
 , Push regB
 , Load (ImmValue (4)) regD
 , Load (ImmValue (5)) 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 (95)) regB
 , WriteInstr regB (DirAddr (1))
 , Jump (Abs (164))
 , Load (ImmValue (2)) regB
 , Compute Sub regSP regB regSP
 , Load (ImmValue (0)) 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 (3)) regC
 , Compute Lt regB regC regB
 , Compute Equal regB reg0 regC
 , Branch regC (Abs (148))
 , Load (ImmValue (0)) regB
 , Compute Sub regA regB regB
 , Load (IndAddr regB) regB
 , Load (ImmValue (4)) regC
 , Compute Add regC regB regC
 , ReadInstr (IndAddr regC)
 , Receive regC
 , Load (ImmValue (0)) regB
 , Compute Gt regC regB regC
 , Compute Equal regC reg0 regB
 , Branch regB (Abs (138))
 , 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
 , Load (ImmValue (4)) regD
 , Compute Add regD regC regD
 , ReadInstr (IndAddr regD)
```

```
 , Receive regD
 , Compute Add regB regD regB
 , Load (ImmValue (1)) regC
 , Compute Sub regA regC regC
 , Store regB (IndAddr regC)
 , Jump (Abs (138))
 , Nop
 , Load (ImmValue (0)) regB
 , Compute Sub regA regB regB
 , Load (IndAddr regB) regB
 , Load (ImmValue (1)) regC
 , Compute Add regB regC regB
 , Load (ImmValue (0)) regC
 , Compute Sub regA regC regC
 , Store regB (IndAddr regC)
 , Jump (Abs (105))
 , Nop
 , TestAndSet (DirAddr (0))
 , Receive regB
 , Branch regB (Rel (2))
 , Jump (Rel (-3))
 , ReadInstr (DirAddr (3))
 , Receive regB
 , Load (ImmValue (1)) regC
 , Compute Sub regA regC regC
 , Load (IndAddr regC) regC
 , Compute Add regB regC regB
 , Load (ImmValue (3)) regC
 , WriteInstr regB (IndAddr regC)
 , WriteInstr reg0 (DirAddr (0))
 , WriteInstr reg0 (IndAddr regSprID)
 , EndProg
 , Load (ImmValue (167)) regB
 , WriteInstr regB (DirAddr (2))
 , Jump (Abs (236))
 , Load (ImmValue (2)) 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
 , 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 (6)) regC
 , Compute Lt regB regC regB
 , Compute Equal regB reg0 regC
 , Branch regC (Abs (220))
 , Load (ImmValue (0)) regB
 , Compute Sub regA regB regB
 , Load (IndAddr regB) regB
 , Load (ImmValue (4)) regC
 , Compute Add regC regB regC
 , ReadInstr (IndAddr regC)
 , Receive regC
 , Load (ImmValue (0)) regB
 , Compute Gt regC regB regC
 , Compute Equal regC reg0 regB
 , Branch regB (Abs (210))
 , 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
 , Load (ImmValue (4)) regD
 , Compute Add regD regC regD
 , ReadInstr (IndAddr regD)
 , Receive regD
 , Compute Add regB regD regB
 , Load (ImmValue (1)) regC
 , Compute Sub regA regC regC
 , Store regB (IndAddr regC)
 , Jump (Abs (210))
 , Nop
 , Load (ImmValue (0)) regB
 , Compute Sub regA regB regB
 , Load (IndAddr regB) regB
 , Load (ImmValue (1)) regC
 , Compute Add regB regC regB
 , Load (ImmValue (0)) regC
 , Compute Sub regA regC regC
 , Store regB (IndAddr regC)
 , Jump (Abs (177))
 , Nop
 , TestAndSet (DirAddr (0))
 , Receive regB
 , Branch regB (Rel (2))
```

```
 , Jump (Rel (-3))
 , ReadInstr (DirAddr (3))
 , Receive regB
 , Load (ImmValue (1)) regC
 , Compute Sub regA regC regC
 , Load (IndAddr regC) regC
 , Compute Add regB regC regB
 , Load (ImmValue (3)) 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))
 , Load (ImmValue (259)) regB
 , Push regB
 , Push regA
 , Compute Decr regSP reg0 regB
 , Load (ImmValue (0)) regC
 , Load (ImmValue (6)) regD
 , Compute GtE regC regD regD
 , Branch regD (Rel (8))
 , Load (ImmValue (4)) regD
 , Compute Add regD regC regD
 , ReadInstr (IndAddr regD)
 , Receive regD
 , Push regD
 , Compute Incr regC reg0 regC
 , Jump (Rel (-9))
 , Compute Add regB reg0 regA
 , Jump (Abs (1))
 , ReadInstr (DirAddr (3))
 , Receive regB
 , WriteInstr regB numberIO
 , EndProg
 ]

main = run [prog,prog,prog]
```

Infinite run tests

In the Figure 63 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.

```
@Test
public void testInfiniteLoop() throws InterruptedException {
    Thread thread = new Thread() {
        @Override
        public void run() {
            check("cannon { while (10>3) { int a =3;}}", "infiniteLoop", "");
        }
    };

    thread.start();

    // Let the current thread sleep (not the created thread!)
    Thread.sleep(5000);

    Assert.assertTrue(thread.isAlive());
}

/*
 * IMPORTANT! Do not forget to kill ghc process manually as its execution is not
 * stopped due to infinite cycle
 */
@Test
public void testDivisionByZero() throws InterruptedException {
    Thread thread = new Thread() {
        @Override
        public void run() {
            check("cannon { int a = 3/0;}", "divisionByZero", "");
        }
    };

    thread.start();

    // Let the current thread sleep (not the created thread!)
    Thread.sleep(5000);

    Assert.assertTrue(thread.isAlive());
}
```

Figure 63. Infinite run semantic tests

JUnit results

As can be seen from the Figure 64 all tests pass.

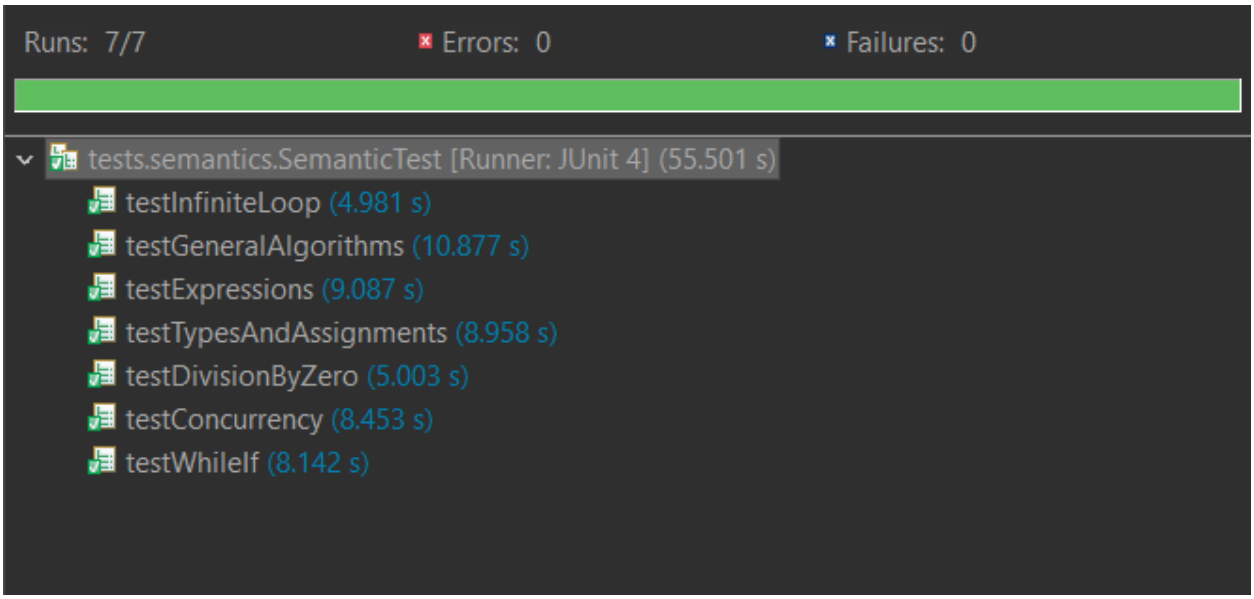


Figure 64. Semantic tests JUnit results