Testing

//functions

//scoping function tests

**Introduction**

We decided to do extensive testing to ensure we had a working and valid compiler. We came up with several different tests which we will discuss briefly. The combination of these tests, tests whether all parts are working separately, and completer tests to test whether these parts all work and fit together. For some parts we did unit testing (Junit 4), but for some we did system tests with predefined programs, as it was quite hard to get Haskell output in Java.

**SYNTAX TESTING**

**LexerTester**

This unit tests verifies whether the keywords of Lava are tokenized properly. All keywords are at least tested once. The test checks a String, which is tokenized, and matches it with the keywords given in the test. We also tests for extra whitespaces and inserted comment lines. Some test instances contain values which should not parse ( example : #). This test can be executed by running LavaVocabLexerTester.java.

**ParserTester**

Our parser tester parses Strings, and checks whether it parses, and prints the errors if it fails. First we created some stub “Chambers” in our simpleProgramTests, and some stub “Chambers” which should fail. We tested separate grammar rules in our parser tester, for example we created 2 tests with several instances for the statement grammar rule. One test with incorrect code and one with correct code. Every rule in the Statement grammar rule is at least tested once. We also did this for expressions. These rules combined are tested with all our test programs, which all obviously should pass. This test can be executed by running LavaParserTester.java

**CONTEXTUAL ERRORS**

**CheckerTester**

This unit tests checks for types, scopes etc. Every test contains 5 test instances. We first test for return types. Does a function return the type that is given in the function definition? Also, some instances contain expressions and calculations within the function, and still check whether the return type is the one expected. A second test checks if variables can be given values that are the type given in the variable definition. This also is checked with several wrong test instances, to verify that our type checker works. Fork and joins are also tested, as fork only accepts functions which return void. The scope is tested with various assignments and it checks whether these variables are accessible within the current scope. This also is done with several wrong instances where the variable is not defined in current scope. Also, as we did in the ParserTester we check all the test programs for any errors.

**SEMANTIC ERRORS**

For the semantic error testing we decided not to create unit tests, as it is hard to create a unit tests where Haskell, Java and Sprockell are combined. We created some simple tests, which takes an input, and results an output. We verify whether the output is the value we expected. In these tests all kinds of expressions, concurrent calculations etc. Also, other standard programs like fib, max, gcd etc. are tested for correct output. We included compiled versions (.exe/.sh) so you can easily test these programs without compiling them yourself. The output should always be on localMem index 1 (0= reserved). Except when the program is concurrent, then it should be on mem

|  |  |
| --- | --- |
| Test name | Expected output |
| simpletest1 | 63 |
| simpletest2 | 500 |
| Simpletest3 | 500 |
| Simpletest4 | -47 |
|  |  |
|  |  |
|  |  |

**OTHER TESTS**

**Generator Test**

We have a small test where small program