Vrije Universiteit Brussel

Faculty of Science
Department of Computer Science

Capita Selecta of Software Engineering: Project

Second deliverable

Oliver Verhaegen - 87116
overhaeg@vub.ac.be

Teacher: Maja D'Hondt

April 27, 2015

1 Source Code Repository

The source code for this project is avalaible at https://github.com/overhaeg/CSPL-Project.

2 Project Description

The goal of this project is the implementation of an interpreter of System Fω. System Fω is an extension of System F, which is itself an extension of the Simply Typed Lambda Calculus (STLC). STLC introduces typing to lambda calculus, with a single function operator. System F extends STLC by adding support for parametric polymorphism, and System Fω extends this further with higher-order polymorphism. The interpreter will require a Parser, a Type checker and an Evaluator.
System Fω will be implemented using the Haskell language, and should therefore be runnable on every system for which an implementation of the Glasgow Haskell Compiler (GHC) exists. To help with the implementation of the parser we will use the Happy Parser Generator for Haskell. We will use HUnit as the framework for implementing unit testing. HUnit is very similar to JUnit. To check test coverage, we use the inbuilt HPC tool of GHC.
This interpreter was implemented as a three-part project for the course Capita Selecta of Programming Languages.

3 Overall Planning

The first milestone had to be postponed for one week due to time incompatibilities with other projects. The overall planning of the second milestone will stay the same.

Milestone 1
Date: March 2 - April 26
Time: 40 - 45 hours
Objective: Implement the tests related with Release 1 and 2.

Milestone 2
Date: April 27 - June 8
Time: 40 - 45 hours
Objective: Implement the tests related with Release 3.

4 Detailed Planning v1: Review

We achieved to mostly complete every task we scheduled for the first milestone, but we had to add items to the tasks to check if error are correctly triggered when they should, especially for the Typechecker.
The most effort was put in the first two tasks as we had to set up the test process, but we never came close to working up to 8 hours on a task. We could easily have merged tasks together, and it is in fact what we did during the implementation. The project still only supports basic features and we didn't encounter major problems or blocks during the implementation of the tests. We wrote 14 tests for the Parser, 28 tests for the Typechecker and 26 tests for the Evaluator. The only minor problem we encountered was a test that doesn't reflect the actual behavior of the program, that is a test that doesn't seem to catch the exception thrown by the Typechecker when using an unbounded variable in a lambda. The problem seems to reside in the way the test framework catches the exception (the exception is thrown by a subexpression of the tested expression) and we will try to find a solution to this in the next milestone. All other tests run successfully.
For this milestone we achieved a test coverage of 53% on the Parser, 88% on the Typechecker and 65% on the Evaluator. The low coverage of the Parser is hard to analyze since it is generated by Happy. It is also possible that it is related to the fact that most parsable expressions will not be accepted by the Typechecker or the Evaluator and are therefore not tested, since it is not the job of the Parser to reject these expressions. The Typechcker has a much higher coverage, the missing coverage is caused by not testing trivial error handling, and not testing every operator in the body of lambda's. The Evaluator however has only a 65%, mainly because we didn't test every case where substitution (on application) applies. We plan on adding some tests on substitution to increase the test coverage of the Evaluator.

5 Detailed Planning v2

The next and last milestone is Milestone 2, for which we will mainly work on developping tests for Release 3.

Correction of Release 2

- Correct the faulty test problem.
- Write more tests to cover more substitution cases.

Parser: Abstract Types and Type Expressions

- Make a unit test for each new symbol added to the Parser.
- Make unit tests for nested expressions that should be accepted by the Parser.

Type Checker: Kinds, Abstract Types

- Make unit tests to check the valid typing of valid expressions to Abstract types
- Make unit tests to control if the Typechecker correctly supports kinds.
- Make unit tests to check if badly typed parsable expression are correctly rejected by the Typechecker, with Abstract types and kinds in mind.
- We expect a test coverage of more than 90% for the whole TypeChecker.

Evaluator: System Fω

- Make units tests to check that the use of kinds and abstract types is correctly evaluated.
- Make unit tests to control that the Evaluator fully supports valid System Fω expressions.
- The test coverage should be much higher than for the previous milestone, we expect more than 90% test coverage.

We plan to divide the milestone in 3 tasks. The first task will consist of correcting the previous release and handling the Parser part. The second task will focus on handling tests for the Typechecker and the last task will be used to develop tests for the Evaluator.