OST
Eastern Switzerland
University of Applied Sciences

Improving the Usability of the Haskell Substitution Stepper

Task Description

1. Setting

In the imperative programming paradigm, a debugging tool with an appropriate visualisation of the program counter and internal state is often used as an aid to visualise and learn about program execution. Such tools enjoy widespread use by beginners wanting to learn how the paradigm works, as well as professional programmers trying to find causes for unexpected program behaviour.

The functional programming paradigm does not have the concept of a program counter or internal state. Other techniques are therefore required to visualise the execution of functional programs. Executing a program in a functional programming language is typically viewed as evaluating an expression using repeated substitution:

```
  sum [1,2,3]
= { applying sum }
  1 + sum [2,3]
= { applying sum }
  1 + (2 + sum [3])
= { applying sum }
  1 + (2 + (3 + sum []))
= { applying sum }
  1 + (2 + (3 + 0))
= { applying + }
  6
```

```
                        reverse [1, 2, 3]
= { applying reverse } reverse [2, 3] ++ [1]
= { applying reverse } (reverse [3] ++ [2]) ++ [1]
= { applying reverse } ((reverse [] ++ [3]) ++ [2]) ++ [1]
= { applying reverse } (([] ++ [3]) ++ [2]) ++ [1]
= { applying ++ }      ([3] ++ [2]) ++ [1]
= { applying ++ }      [3, 2] ++ [1]
= { applying ++ }      [3, 2, 1]
```

```
pure (+) <*> [1,2] <*> [3,4]
= [(+)] <*> [1,2] <*> [3,4]
= [(+) 1,(+) 2] <*> [3,4]
= [(+) 1 3,(+) 1 4,(+) 2 3,(+) 2 4]
= [4,5,5,6]
```

```
do {n <- pure 10; m <- pure 2; safediv n m}
= pure 10 >>= (\n -> (pure 2 >>= (\m -> safediv n m)))     (do syntax with explicit λ parentheses)
= Just 10 >>= (\n -> (pure 2 >>= (\m -> safediv n m)))     (definition of pure)
= (\n -> (pure 2 >>= (\m -> safediv n m))) 10              (definition of >>=)
= pure 2 >>= (\m -> safediv 10 m)                    (function application)
= Just 2 >>= (\m -> safediv 10 m)       (definition of pure)
= (\m -> safediv 10 m) 2                     (definition of >>=)
= safediv 10 2                          (function application)
= Just (10 `div` 2)           (definition of safediv)
= Just 5                      (definition of div)
```

Derivations such as these are used when teaching functional programming and reasoning about functional programs. Having tool support for generating such derivations could greatly help learning programming in and debugging programs written in the functional style.

The Haskell Substitution Stepper (HaskSubStep) is a tool that proposes to interactively generate such derivations. Although the current version of the tool is capable of generating derivations of a large subset

of Haskell expressions, the current user interface requires improvements in order for the tool to be used effectively by its target audience.

2. Goals

The main aim of this project is to improve the usability of HaskSubStep. It plans to do this by proposing an appropriate concept for user interaction for HaskSubStep and implementing it as part of the tool in the language Haskell.

The following is a brief and unstructured list of initial tasks, requirements and notes:

1. The tool must be usable for a beginner with little experience in functional programming. Its user interface must be carefully designed to this end.
2. The project should start with an initial study to consider how people could use HaskSubStep and conclude with a concept on the best way(s) to interact with HaskSubStep.
3. Since derivations may be long, the user may choose to advance the derivation or fold parts of the derivation in a controlled manner (e.g., compress or step over derivation steps related to certain trivial functions)
4. Since individual expressions at each step of a derivation may be long, the tool should allow hiding parts of large expressions that remain the same, or allow highlighting subexpressions that change between steps.
5. The user interaction may be implemented as a command-line interface (CLI), or a graphical user interface (GUI), or both.
6. In the likely case where a graphical user interface is required, a plugin to the Haskell Language Server VS Code Plugin should be considered.
7. Regardless of the choice of user interface, an open API and/or command line interface (CLI) should be provided to allow the tool to be open to additional user interfaces.
8. Contributions to other parts of HaskSubStep as also possible in case this is required for satisfying the main aim of the project, or in case there are remaining resources.
9. The results may be used in the current run of the functional programming course at the OST.
10. The results will be released under the GPLv3 license.

Further refinements and modifications to this list that serve the main aim of this project are possible during its course.

3. Deliverables

- All artefacts (source code, web pages, accepted pull requests, etc.) required to achieve the goals of this project.
- Product documentation in English that is relevant to the use and further development of the results (e.g., requirements, domain model, architecture description, code documentation, user manuals, etc.) in a form that can be developed further and is amenable to version control (e.g., LaTeX or Markdown). Ideally, all product documentation should be contained withing the artefacts required to achieve the goals of this project as stated in above.
- Project documentation that is separate from the product documentation that briefly, but precisely documents information that is only relevant to the current project (e.g., project plan, time reports, meeting minutes, personal statements, etc.).

- Additional documents as required by the department (e.g., poster, abstract, presentation, etc.)
- Any other artefacts created during the execution of this project.

All deliverables may be submitted in digital form.

4. Stakeholders

Partner: Software Engineering and Programming Language Lab, IFS Institute for Software, OST
Student: Carlo Del Rossi
Supervisor: Farhad Mehta.

5. Other Project Details

Type of project: Bachelor Thesis Project (de: Bachelorarbeit)
Duration: 20.02.2023 – 16.06.2023
Workload per student: 12 ECTS (1 ECTS = approx. 30 Hours)