Improving the Usability of the Haskell Substitution Stepper

Graduate

Carlo Del Rossi

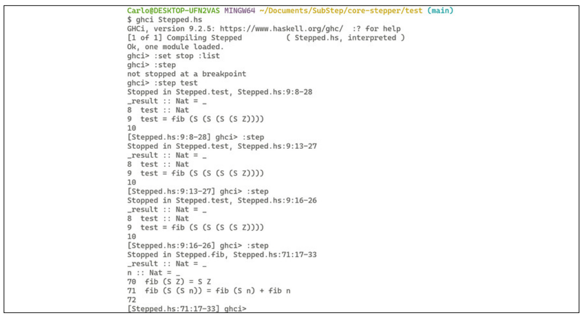
Initial Situation: With functional programming languages becoming more widespread and being taught at Universities, many programmers will eventually get in contact with them. Since functional programs can be very different from imperative ones, especially due to the heavy reliance on recursion, this could be very interesting for all the people that are not yet familiar with Haskell's concepts.

Problem: While functional languages like Haskell have debuggers, they are not as user-friendly and don't offer as much insight as debuggers for imperative languages. The internal state of the program is usually not displayed very comprehensibly, which leads to the not being very useful for learning processes.

Result: The goal of the Haskell Substitution Stepper is to provide the user with the capability to step through a Haskell program and to be able to see what happens in the background when a function is executed. This is supposed to solve the usability problems that the regular Haskell debugger has.

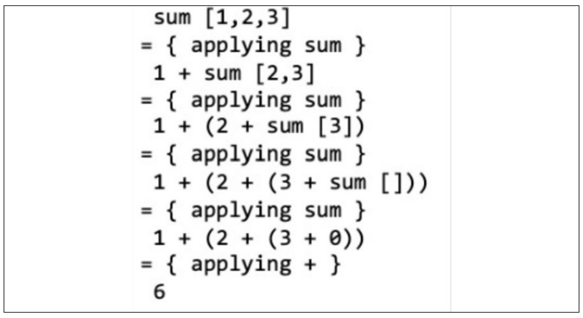
For this, we came up with an application where the user can specify a function that he would like to step through
and the application would load this function and then use Haskell's rules to step through. The user can see the whole internal state of the term that is being stepped through and the highlighting of the changes makes it easy to see what happens in each step. The user can choose between different modes of derivation and can control the flow of the derivation. The addition of helpful commands also allows the user to skip certain parts of the derivation that might not be interesting to them.

Stepping via the GHC compiler's interactive debugging mode
Own presentment



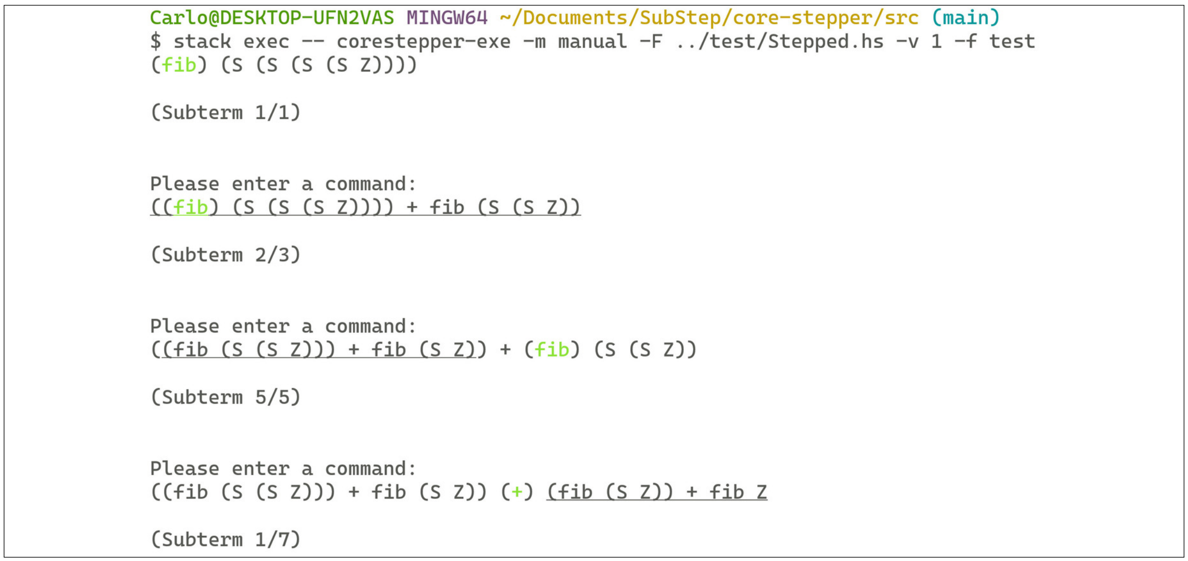
```
Carlo@DESKTOP-UFN2VAS MINGW64 ~/Documents/SubStep/core-stepper/test (main)
$ ghci Stepped.hs
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
[1 of 1] Compiling Stepped          ( Stepped.hs, interpreted )
Ok, one module loaded.
ghci> :set stop :list
ghci> :step
not stopped at a breakpoint
ghci> :step test
Stopped in Stepped.test, Stepped.hs:9:8-28
_result :: Nat = _
8  test :: Nat
9  test = fib (S (S (S (S Z))))
10
[Stepped.hs:9:8-28] ghci> :step
Stopped in Stepped.test, Stepped.hs:9:13-27
_result :: Nat = _
8  test :: Nat
9  test = fib (S (S (S (S Z))))
10
[Stepped.hs:9:13-27] ghci> :step
Stopped in Stepped.test, Stepped.hs:9:16-26
_result :: Nat = _
8  test :: Nat
9  test = fib (S (S (S (S Z))))
10
[Stepped.hs:9:16-26] ghci> :step
Stopped in Stepped.fib, Stepped.hs:71:17-33
_result :: Nat = _
n :: Nat = _
70  fib (S Z) = S Z
71  fib (S (S n)) = fib (S n) + fib n
72
[Stepped.hs:71:17-33] ghci>
```

The sketch of a possible styling for the solution (from the task description)
Own presentment



```
 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
```

Stepping via the Haskell Substitution Stepper (The selected subterms are shown in green while the diff is underlined)
Own presentment



```
Carlo@DESKTOP-UFN2VAS MINGW64 ~/Documents/SubStep/core-stepper/src (main)
$ stack exec -- corestepper-exe -m manual -F ../test/Stepped.hs -v 1 -f test
(fib) (S (S (S (S Z))))

(Subterm 1/1)


Please enter a command:
((fib) (S (S (S Z)))) + fib (S (S Z))

(Subterm 2/3)


Please enter a command:
((fib (S (S Z))) + fib (S Z)) + (fib) (S (S Z))

(Subterm 5/5)


Please enter a command:
((fib (S (S Z))) + fib (S Z)) (+) (fib (S Z)) + fib Z

(Subterm 1/7)
```

Advisor
Prof. Dr. Farhad D. Mehta

Co-Examiner
Dr. Joachim Breitner

Subject Area
Software, Application Design