add more handouts and update index
mhe authored 1 month ago

7845144f

Runxy.md 2.18 KB

Runxy

This command line interface allows to run e.g.

```
$ runhaskell Runxy.hs factorial.xy 5
120
```

for the file factorial.xy. The usage is

```
  runhaskell Runxy.hs <filename> <Int>
```

Alternatively, for faster execution time, you can compile `Runxy.hs` with

```
$ ghc --make Runxy.hs
```

and then run it with

```
$ ./Runxy <filename> <Int>
```

You can also use optimization options, such as

```
$ ghc --make -O2 Runxy.hs
```

We name the module `Main` even though the file is called `Runxy.hs`, so that we can compile to produce an executable as explained above:

```
module Main where

import System.Environment
```

We import the module `System.Environment` so that we can read the command line arguments with `getArgs`. We also need to import our own modules:

```
import AbstractSyntax
import Parser
import Interpreter
```

Because our little language doesn't have IO facilities, we use the variable `x` to hold the input and the variable `y` to hold the output. So this function creates a storage with the value `x` for the variable `"x"`, and with all other variables uninitialized, giving an error if we try to use them:

```
initialStorage :: Integer -> Storage
initialStorage x = update "x" x emptyStorage
```

Now, given a program in abstract syntax and a value for `x`, we run it with the above interpreter, and extract the value of the variable `"y"`.

```
runxy :: Program -> Integer -> Integer
runxy p x = m' "y"
  where
    m  = initialStorage x
    m' = run p m
```

Finally, the `main` function reads the command line arguments with `getArgs`, then uses the module `Parser` to parse the file, and the module `Interpreter` to run the syntax tree produced by the parser on the given integer:

```
main :: IO()
main =
```

```
do
  args <- getArgs
  if length args == 2
     then
       do
         concreteProgram <- readFile (args !! 0)
         let abstractProgram = parseProgram concreteProgram
         let x = read(args !! 1)
         let y = runxy abstractProgram x
         putStrLn (show y)
     else
         putStrLn "Usage: runhaskell Runxy.hs <filename> <Integer>"
```

Back to table of contents