Chad Reynolds CS:5850 Programming Language Foundations
Final Project Report Due: December 15, 2016

Introduction

My project goal was to implement pure lambda calculus with De Bruijn indices, in Haskell. To accomplish this I wrote a simple (and mostly functioning) parser using the Parsec library, which converts the string input to the internal datatype for lambda terms. The program attempts to reduce this term to normal form, or loops endlessly if a normal form does not exist. Once a normal form is found the program outputs the reduction sequence of terms, it also writes out LaTeX markup to a file that displays this reduction sequences as syntax trees.

Code Overview

- **term.hs** - datatype definition for De Bruijn indexed lambda terms
- **nat.hs** - simple natural number implementation
- **reduction.hs** - beta-reduction and normalization functions
- **subst.hs** - the substitution and renaming functions used in beta reduction, as defined in Hankin [1]
- **prettyPrinter.hs** - the toStr and toLaTeX functions for nice output
- **parser.hs** - the parser, takes strings and converts them into the Term datatype for processing
- **main.hs** - the main loop, where terms are input and their reduction to normal form is output

Setup & Installation

The program relies on a working Haskell installation, and also an installation of the Parsec library. It was written and tested using GHC version

8.0.1.20161117 and Parsec version 3.1.11. Make is also useful to run the commands in the Makefile I created. All commands are expected to be run from the project's base directory. Running them in subfolders will either fail or cause files to be generated improperly.

Once everything is installed, running *make* will build the lambdaCalc executable and *make interactive* will start the ghci REPL with examples.hs in scope. The examples file has a number of lamba terms, variables, comparisons of reductions, the SKI combinators, and also loads the Term datatype definition, reduction, and substitution for interactive testing.

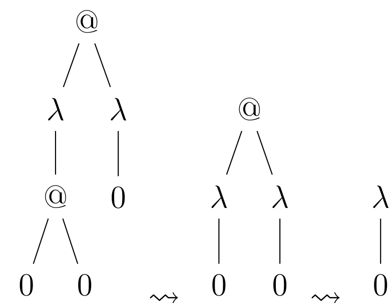
Running the Code

Running the lambdaCalc executable will bring up a prompt to enter in a lambda term to be reduced. Lambdas are represented by \ characters and terms are written using standard De Bruijn syntax, where the variables are natural numbers that reference their binder. Application is implicit, but there are cases where the parser fails to recognize valid terms or improperly parses others. To avoid this problem, fully parenthesize all of the terms you provide as input. For example, (\.00)\.0 is valid, but should be entered as (\.00) (\.0) to avoid any errors.

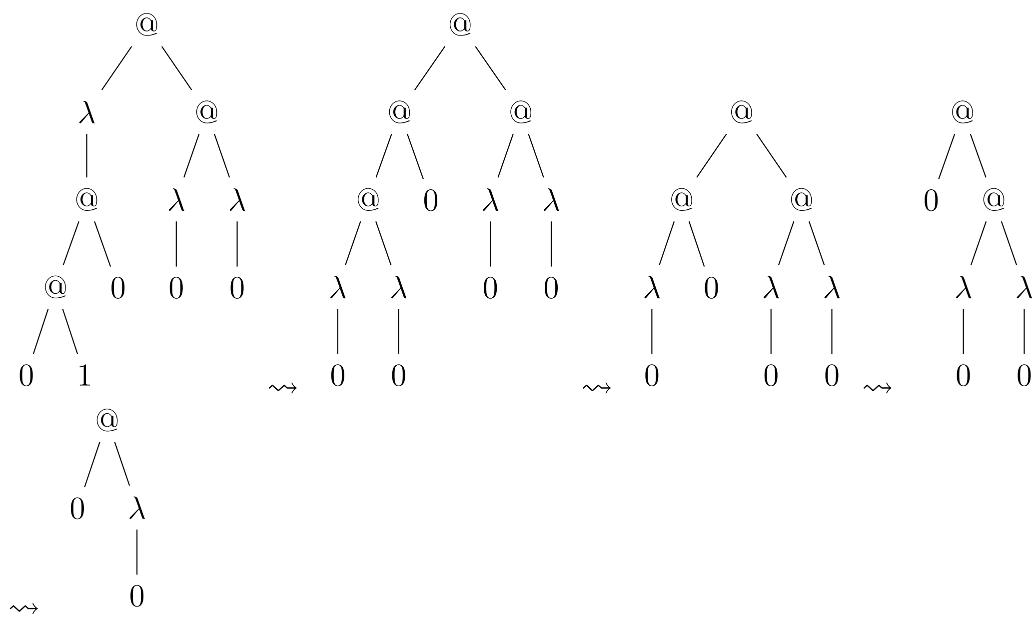
After a term is entered, if it terminates a reduction sequence to its normal form will be displayed. In the *out* directory a .tex file will be written with a name in the format **reduction<prompt_number>.tex**. All of these files can be compiled to pdf at once using the *make pdf* command.

Sample Input & Output

Please enter a lambda term:
(\.00)(\.0)
Reduction sequence:
((\.(0 0)) (\.0)) >
((\.0) (\.0)) >
(\.0)



(\.010)((\.0)(\.0))
Reduction sequence:
((\.((0 1) 0)) ((\.0) (\.0))) >
((((\.0) (\.0)) 0) ((\.0) (\.0))) >
(((\.0) 0) ((\.0) (\.0))) >
(0 ((\.0) (\.0))) >
(0 (\.0))



References

[1] Hankin, C. *An Introduction to Lambda Calculi for Computer Scientists.* Texts in computing. Kings College, 2004.