Haskell Computer Algebra System (HCAS) Proposal

Rob Tougher (rt2301@columbia.edu)

September 19, 2007

For the class project I would like to implement a simple computer algebra system, HCAS. HCAS will be a purely functional programming language that provides a set of basic operations for constructing and manipulating algebraic expressions. Simply put, you can think of HCAS as **a subset of Haskell, plus support for computer algebra.**

HCAS will have three main features:

- **Purely Functional Language.** HCAS will be a purely functional subset of Haskell. There will be functions, recursion (and tail recursion if needed), lists, strings, pattern matches for function arguments, etc. No variables, sequencing of operations, or other items from imperative programming languages.
- **Construction of Mathematical Expressions.** HCAS will allow you to construct mathematical expressions using an intuitive syntax. You will be able to define mathematical expressions inline. That is, if you write the expression $x + y - z$, this will automatically be constructed as a math expression.
- **Navigation of Mathematical Expressions.** HCAS will use the concept of pattern matching in function arguments to allow you to navigate mathematical expressions. Consider the following simple function:

```
printType left*right =
        "Multiplication"

printType left+right =
        "Addition"
```

  This function has two definitions, one for addition and one for multiplication. The version that gets executed at runtime is chosen based on the expression argument that you pass in. So if you call the function as “(printType x*y+z)”, the version for addition will be called, because addition binds the loosest. In the body of the function “left” will refer to “x*y”, and “right” will refer to “z”.

When I'm finished with my interpreter I'd like to be able to run the following programs:

```
-- -------------------------------------------------------------
-- Should print "Addition"
-- -------------------------------------------------------------
main =
        (printType x*y+z)

printType left*right =
        "Multiplication"

printType left+right =
        "Addition"



-- -------------------------------------------------------------
-- Should result in (m*a + m*b - m*c)
-- -------------------------------------------------------------
main =
        (distribute m a+b-c)

distribute m left+right =
        (distribute m left) + (distribute m right)

distribute m left-right =
        (distribute m left) - (distribute m right)

distribute m p =
        m*p



-- -------------------------------------------------------------
-- Should result in "desreveR"
-- -------------------------------------------------------------
main =
        (reverse "Reversed")

reverse x:xs = (reverse xs) ++ [x]
reverse [] = []
```