Project Outline

Daniel Briseno, Colton Gering

This document will outline a Haskell calculator program in abstract terms, using function definitions to outline behaviors that the program will implement. Haskell's mathematical nature should make it easy to transfer these abstract function definitions into functional code.

For the time being, this calculator will only accept natural numbers and the operators $+; -; *; /$. Once we have a functional calculator using these simple inputs we will hopefully add functionality for (), exponents, and variables.

1 Raw Input to Sanitized Input

The first job of the calculator program will be to turn raw user input into usable, sanitized input. We must assume that the user can give any string as input, valid or invalid. So we will begin by defining the following language (call it HL for Human Language), which will define any user input:

$$\forall\, str \in HL; \quad str := str | str + str | str - str | str * str | str/str | + | - | / | *$$

Where str can be anything of string type. Here we include the operators $+, -, *, /$ as valid strings since the user could give a string consisting of only an operator as input. Note that in this language we do not care if the string is valid or invalid. Therefore, we now need to define an intermediate language, $\mathcal{L}$:

$$\forall\, exp \in \mathcal{L}; \quad exp := exp | exp + exp | exp * exp | exp/exp | \emptyset$$

This language will be our language of sanitized inputs, and here is where we check for validity. We now need to define a function Φ. For the sake of simplicity of our definitions, let us define $op \in \{+, *, -, /\} \subset HL$

$$\begin{aligned}
\Phi : HL &\to \mathcal{L} \\
\Phi(str) &= exp \\
\Phi(str + str) &= \Phi(str) + \Phi(str) \\
\Phi(str * str) &= \Phi(str) * \Phi(str) \\
\Phi(str - str) &= \Phi(str) - \Phi(str) \\
\Phi(str/str) &= \Phi(str)/\Phi(str) \\
\Phi(op\ \ str) &= \emptyset \\
\Phi(str\ \ op) &= \emptyset \\
\Phi(str\ \ op\, op\ \ str) &= \emptyset
\end{aligned}$$

We have not yet taken care of strings which do not contain operators, but are of non-numeric type (this will remain invalid until we add variables). This will be addressed when we go from our language $\mathcal{L}$ to $\mathbb{R}$.

2 ARS

Now that we have usable string input, we can begin to define an *ARS* which can capture order of operations. So we will define our set of equations E. Let $a, b, c \in \mathcal{L}$.

$$\begin{aligned} E &\subset \mathcal{L} \times \mathcal{L} \\ (a + b, (a) + b) &\in E \\ (a - b, (a) - b) &\in E \end{aligned}$$

These equations should capture order of operations over $+, -, *, /$.