Design of a Simple Functional Programming Language and Environment for CS2

Brian T. Howard
Department of Computer Science
DePauw University
Overview

Motivation
- Introduce recursion, abstraction, data types
- Exposure to other languages

Context
- CS1/2 in Java and C++
- BlueJ, Eclipse, and Visual Studio
- History of using Scheme and Haskell
Development of HasCI and FUNNIE

- Modeled on Haskell
  - Pattern-matching
  - Static typing with type inference
  - Lazy evaluation
- Algebraic Reasoning
- Familiar Syntax
- Attractive and Easy Environment
HasCl examples:

\[ \text{fact} :: (\text{Num}) \rightarrow \text{Num} \]
\[ \text{fact}(0) = 1; \]
\[ \text{fact}(n) = n \times \text{fact}(n - 1); \]
\[ \begin{align*}
\text{fact}(3) &= 3 \times \text{fact}(2) \\
&= 3 \times 2 \times \text{fact}(1) \\
&= 3 \times 2 \times 1 \times \text{fact}(0) \\
&= 3 \times 2 \times 1 \times 1 = 6
\end{align*} \]

Haskell equivalent:

\[
\begin{align*}
\text{fact 0} &= 1 \\
\text{fact n} &= n \times \text{fact} (n - 1)
\end{align*}
\]
\[
\text{Sum} :: ([\text{Num}]) \rightarrow \text{Num}
\]

\[
\text{sum}([\ ] ) = 0;
\]

\[
\text{sum}([x : xs]) = x + \text{sum}(xs);
\]

\[
\text{sum}([1, 2, 3]) = 1 + \text{sum}([2, 3])
\]

\[
= 1 + 2 + 3 + 0
\]

\[
= 6
\]

\text{sum} [ ] = 0

\text{sum} (x : xs) = x + \text{sum} x s
\[ \text{find} :: (a, [a]) \rightarrow \text{Bool} \]
\[
\text{find}(x, [ ]) = \text{false};
\]
\[
\text{find}(x, [y : ys]) = \begin{cases} 
\text{true} & \text{if } (x == y) \\
\text{false} & \text{else find}(x, ys);
\end{cases}
\]
\[
\text{find } x [ ] = \text{False} \quad \text{(ctor)}
\]
\[
\text{find } x (y : ys) = \begin{cases} 
\text{true} & \text{if } x = y \\
\text{false} & \text{else find } x \ ys
\end{cases}
\]
\[
\text{hasZero} = \text{find} \ 0
\]
\[
\text{hasZero} \ [7, 3, 4, 0, 6, 8] \Rightarrow \text{true}
\]
```haskell
quicksort :: ([a]) -> [a];
pivot :: (a, [a]) -> ([a], [a]);
quicksort([ ]) = [ ];
quicksort([x : xs]) =
    let (left, right) = pivot(x, xs)
    in quicksort(left)
        ++ [x]
        ++ quicksort(right);

pivot(x, [ ]) = ([ ], [ ]);  
pivot(x, [y : ys]) =
    let (left, right) = pivot(x, ys)
    in if (y < x) then ([y : left], right)
        else (left, [y : right]);
```

power of list manipulations and patterns
fibs :: [Num],

fibs = [1, 1 : zipWith((+), fibs, tail(fibs))];
\[
\text{Sieve} :: ([\text{Num}]) \rightarrow ([\text{Num}]) ;
\text{primes} :: ([\text{Num}]) ;
\]
\[
sieve([\ ] ) = [ \ ] ;
\]
\[
sieve([p : xs]) =
\[
[p : sieve([x | let x \leftarrow xs, 
\text{if} \ (x \& p \neq 0)])] ;
\]
\[
\text{primes} = sieve([2 \ldots]) ;
\]
\[
\text{list comprehension}
\]
\[
\text{qsort}([\ ] ) = [ \ ] ;
\]
\[
\text{qsort}([x : xs]) = \text{qsort}([y \mid let y \leftarrow xs, \text{if} \ y < x]) + [x] + \text{qsort}([y \mid let y \leftarrow xs, \text{if} \ y \geq x]) ;
\]
data Tree = Empty
    | Node(Tree, Num, Tree);  

insert(x, Empty) = Node(Empty, x, Empty);  
insert(x, Node(left, y, right)) =
    if (x < y)  
        then Node(insert(x, left), y, right)  
    else Node(left, y, insert(x, right));  

tfind(x, Empty) = false;  
tfind(x, Node(left, y, right)) =
    if (x == y) then true  
    else if (x < y) then tfind(x, left)  
        else tfind(x, right);  

http://funnie.sourceforge.net/

Thanks to:
NSF REU at DePauw, 2003-
    Chris Colvard, Sara Pagliaro, Matt Ellis,
    Brian McCarty, Miles Huffman,
    David Cheeseman, Andy Bartholomew

DePauw Science Research Fellows, 2003-
    Ryan Smith, Sandy Mtandwa, Elise Hudson,
    Alex Iliev, Laura Stevens, Sean Teska

IBM Eclipse Innovation Grant, 2004
    Atanas Vlahov

DePauw CS2 classes, Fall 2003-