Panel 1

Design of a Simple Functional Programming Language and Environment for CS2

Brian T. Howard
Department of Computer Science
DePauw University

1

Panel 2

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

2

Panel 3

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

3

Panel 4

HasCl examples:

fact :: (Num) -> Num

```
fact(0) = 1;
fact(n) = n * fact(n - 1);
```

fact(3) = 3 * fact(2)
= 3 * 2 * fact(1)
= 3 * 2 * 1 * fact(0)
= 3 * 2 * 1 * 1 = 6

Haskell equivalent:

```
fact 0 = 1
fact n = n * fact (n - 1)
```

parens for args. vs. solely for precedence

4

Panel 5

Sum :: ([Num]) -> Num

```
sum([ ]) = 0;
sum([x : xs]) = x + sum(xs);
```

Sum([1,2,3]) = 1 + Sum([2,3])

= 1 + 2 + 3 + 0

= 6

```
sum [ ] = 0
sum (x : xs) = x + sum xs
```

5

Panel 6

find :: (a, [a]) -> Bool;

```
find(x, [ ]) = false;
find(x, [y : ys]) = if (x == y) then true
                    else find(x, ys);
```

```
find x [ ] = False
find x (y : ys) = if x = y then True
                  else find x ys
```

currying; meaningful u.c. (ctor)

hasZero = find 0

hasZero [7, 3, 4, 0, 6, 8] ⇒ true

6

Panel 7

```
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]);
```

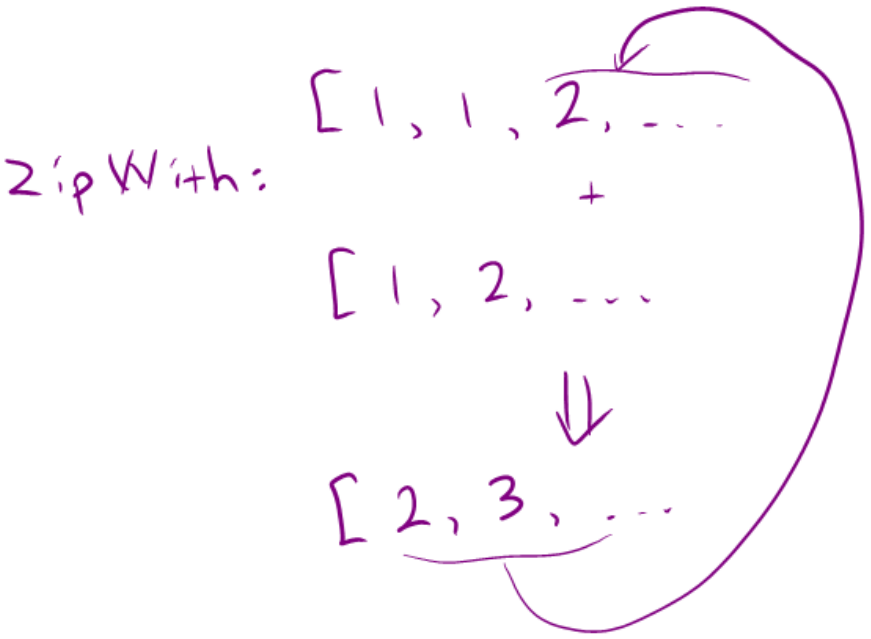
power of list manipulations and patterns

7

Panel 8

fibs :: [Num];

```
fibs = [1, 1 : zipWith((+), fibs, tail(fibs))];
```



lazy lists

(mention Iterator in Java & C++ - streams

8

Panel 9

Sieve :: ([Num]) -> [Num]; primes :: [Num];

```
sieve([ ]) = [ ];
sieve([p : xs]) =
   [p : sieve([x | let x <- xs,
                   if (x % p != 0)])];

primes = sieve([2 ..]);
```

list comprehension

```
qsort([ ]) = [ ];
qsort([x : xs]) = qsort([y | let y <- xs, if y<x])
                  ++ [x]
                  ++ qsort([y | let y <- xs, if y>=x]);
```

9

Panel 10

```
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);
```

User-defined datatypes

10

Panel 11

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-

11