Functional Data Structures

Mattox Beckman

Illinois Institute of Technology
Computer Science

Who we are / October 3, 2011
Who is Mattox?

Name: Mattox Beckman

History: PhD, Fall 2003, University of Illinois at Urbana-Champaign

Area: Programming Languages

Specialty: Partial Evaluation, Functional Programming

Professional Interests: Teaching; Partial Evaluation; Interpreters; Functional Programming; Semantics and Types; Continuations

Personal Interests: Home-brewing; Theology; Investing; Plants; Tarantulas; Evolution; Travel; Korean Culture... and many many more...
Why do programming languages look the way they do?

- Imperative Languages: von Neuman model
  \[ a := a + 1 \]

- OO Languages: message passing model
  \[ a.increment() \]

- Functional Languages: lambda calculus
  \[ a + 1 \]

- Logic Language: unification / first order logic
  \[ \text{inc}(a,b) :- b \text{ is } a+1 \]
Why Functional Programming?

- **Economy of Expression**
  
  ```
  guess [] = []
  guess (x:xs) = guess [y | y <- xs, y < x] ++ [x] ++
                  guess [y | y <- xs, y >= x]
  ```

- **Lazy Evaluation**

- **No Assignment!!**
  - May use more memory
  - Easier to verify
  - Easy to parallelize
What if we run $t_2 = \text{add}(t_1, 35)$ on this?
What if we run $t3 = \text{add}(t2, 55)$ on this?
How to do this?

\[
\begin{align*}
&\text{10} + 30 = 40 \\
&11 + 31 = 42
\end{align*}
\]
And not this?
Some Solutions

- Hard code the functions. (Yuck)
- Use a “trace”. (Works, but cumbersome.)
- Build a new language construct!
Smart Data Structures

- We can’t check where sharing occurs.
- Maybe the data structure can keep track of sharing for us.

```clojure
(defn app [f x] ((first x) f))

(defn mk-const [x]
  [(fn [f] (mk-const (f x))) x])

(defn mk-pair [a b]
  [(fn [f] (mk-pair (app f a) (app f b))) [a b]])

(defn mk-spair [a]
  [(fn [f] (mk-spair (app f a))) [a a]])

(defn loud-inc [x] (println "Inc called!") (+ x 1))
```
For a proof of concept, we try this with a pair that does not keep track of sharing.

user> (def c1 (mk-const 10))
user> (def p1 (mk-pair c1 c1))
user> p1
[@28b6e768] [[@1271ba> 10] [@1271ba> 10]]
user> (app loud-inc p1)
Inc called!
Inc called!
[@239d5fe6] [[@3103074e> 11] [@3dd4ab05> 11]]
This time, the pair is smart about things.

user> (def p2 (mk-spair c1))
user> p2
[<@67ce08c7> [[<@1271ba> 10] [<@1271ba> 10]]]
user> (app loud-inc p2)
Inc called!
[<@38f0b51d> [[<@4302a01f> 11] [<@4302a01f> 11]]]
Future Work

- We’d like this to be less cumbersome.
- In fact, we’d like this to be done completely behind the scenes.