Why Functional Programming Matters

FRANK VASQUEZ

Overview

- Who is this guy?
- Misconceptions
- What is this strange language?
- Gluing functions together
- Gluing programs together
- Why now?

John Hughes

Trinity College, Cambridge

Oxford PhD

Chalmers University

Haskell

QuickCheck

QuviQ

Erlang

CHALMERS

Less is not more

Omitting features can't make a language more powerful.

Lack of side-effects cannot account for order of magnitude reduction in LOC.

goto structured programming analogy

Modular programs are easier to code, extend and test.

Good glue

"A chair can be made quite easily by making the parts - seat, legs, back etc. and sticking them together in the right way. But this depends on an ability to make joints and wood glue. Lacking that ability, the only way to make a chair is to carve it in one piece out of a solid block of wood, a much harder task."

2 new kinds of glue: high-order functions & lazy evaluation

Miranda

listof X ::= nil | cons X (listof X)

[]	means	nil
[1]	means	cons 1 nil
[1,2,3]	means	cons 1 (cons 2 (cons 3 nil))

sum nil = 0

sum (cons num list) = num + sum list

Gluing functions together

reduce f a

Replaces all occurrences of cons in a list by f, and all occurrences of nil by a.

copy = reduce cons nil

sum = reduce add 0

sum [1,2,3]

cons 1 (cons 2 (cons 3 nil))

add 1 (add 2 (add 3 0))

6

Append

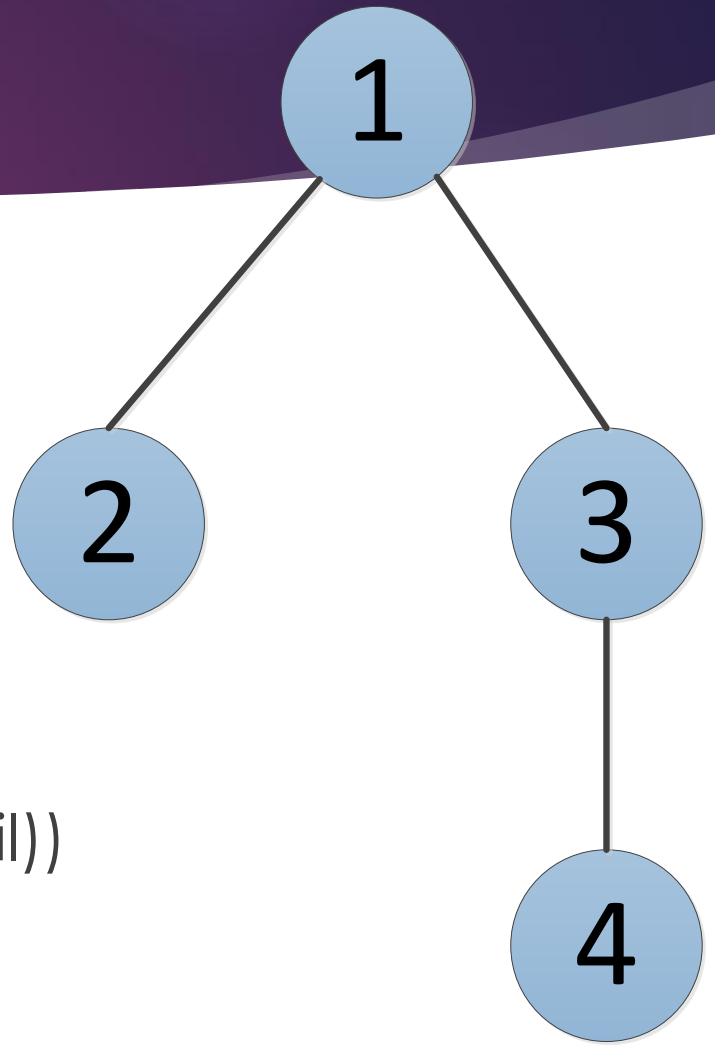
append a b = reduce cons b a

append [1,2] [3,4]
reduce cons [3,4] [1,2]
(reduce cons [3,4]) (cons 1 (cons 2 nil))
cons 1 (cons 2 [3,4]))
[1,2,3,4]

Datatypes

treeof X ::= node X (listof (treeof X))

node 1
(cons (node 2 nil)
(cons (node 3
(cons (node 4 nil) nil))
nil))



Reduce Tree

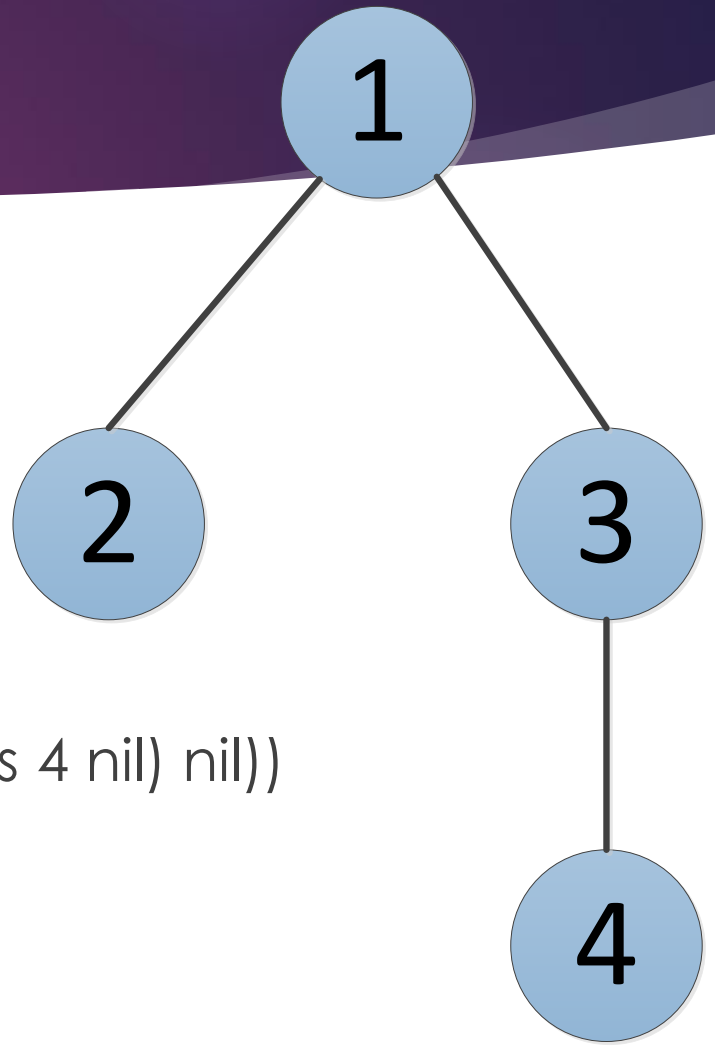
```
redtree f g a (node label subtrees) =
    f label (redtree' f g a subtrees)

redtree' f g a (cons subtree rest) =
    g (redtree f g a subtree) (redtree' f g a rest)

redtree' f g a nil = a
```

Labels

labels = redtree cons append nil

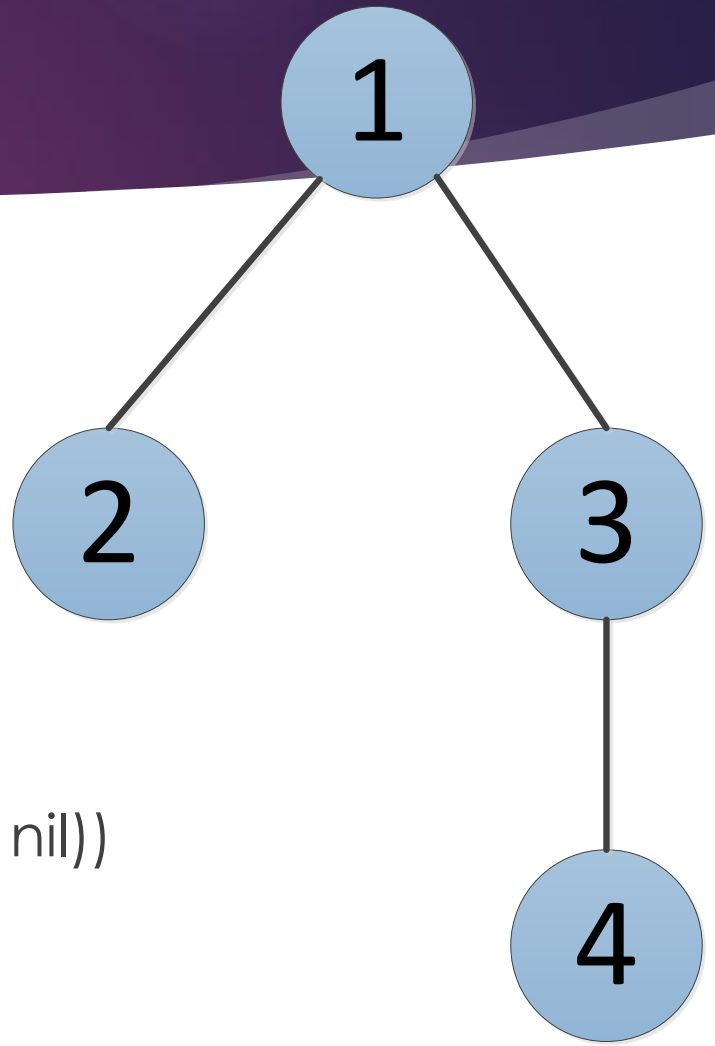
```
cons   1
       (append  (cons 2 nil)
                (append  (cons  3
                                (append (cons 4 nil) nil))
                         nil))
```

[1,2,3,4]



Map Tree

```
(f . g) x = f (g x)          (node . f) x = node (f x)
maptree f = redtree (node . f) cons nil

node  (f 1)
        (cons  (node (f 2) nil)
               (cons (node  (f 3)
                            (cons (node (f 4) nil) nil))
                     nil))
```



Simplicity & Performance

- Underscore
- React
- unidirectional dataflow
- Om
- Flux
- Mori
- Immutable

Gluing programs together

(g . f)

g (f input)

"F is only started once g tries to read some input, and only runs for long enough to deliver the output g is trying to read. Then f is suspended and g is run until it tries to read another input. As an added bonus, if g terminates without reading all of f's output then f is aborted."

Square Roots

Newton Raphson algorithm:

Start with an initial approximation a0.
Compute better and better approximations based on previous.
Stop when approximations differ by less than a tolerance eps.

```
sqrt a0 eps N = within eps (repeat (next N) a0)
```

Generator

Generate a list of approximations.

repeat f a = cons a (repeat f (f a))

[a, f a, f(f a), f(f(f a)), …]

next N x = (x + N/x) / 2

repeat (next N) a0

Selector

"[T]akes a tolerance and a list of approximations and looks down the list for two successive approximations that differ by no more than the given tolerance."

```
within eps (cons a (cons b rest)) =
    = b,                          if abs(a-b) <= eps
    = within eps (cons b rest),   otherwise
```

Streams

- Lisp thunks
- ECMAScript Harmony
- yield
- generators
- coroutines
- callback hell
- Koa

Credits

It's Raining Haskell

an interview with Simon Peyton-Jones & John Hughes

http://channel9.msdn.com/Blogs/Charles/YOW-2011-Simon-Peyton-Jones-and-John-Hughes-Its-Raining-Haskell

Erlang Factory 2015 Panel Discussion

José Valim, Robert Virding, John Hughes & Guido van Rossum

http://www.youtube.com/watch?v=oZwfi8JZ3kU

More Credits

Immutability: Putting the Dream Machine to Work

by David Nolen

https://www.youtube.com/watch?v=SiFwRtCnxv4

Immutable Data in React & Flux

by Lee Byron

http://www.youtube.com/watch?v=I7IdS-PbEgI

Final Credits

Callbacks vs Coroutines

by TJ Holowaychuk

http://medium.com/@tjholowaychuk/callbacks-vs-coroutines-174f1fe66127