Lazy Functional Programming for a survey

Norman Ramsey
Tufts

November 2012

Book: Programming languages for practitioners

Why?

- For people who will write code
- Gives future practitioners something to do

I want your help: What can they do with laziness?

Modest goals for readers

A few ideas of lasting value

- Embodied in “little languages”

Build, prove and compare

- A starting point, not mastery
- *Motivated* readers might tackle ICFP, POPL

(Intended for 3rd- or 4th-year students, young professionals)

What ideas have lasting value? Proven stuff!

Can they build something using

- Functions?
- Types?
- Objects?

A little operational semantics; a little type theory

Bonus: garbage collection, logic programming

μHaskell chapter is being written in a context

One common syntax (Lisp-like)

Untyped functional programming:

- Lists, symbols, integers
- Mutation discouraged
- First-class functions
- Standard higher-order funs (`map`, `filter`, `foldr`, `curry`, `...`)

Typed functional programming:

- The agony of System F
- μML, a nearly pure language with type inference
- Some algebraic data types

μHaskell: I want eyes to pop and heads to explode

Other outcomes:

- They can apply a couple of “standard” lazy tricks
- They want to try laziness for themselves
- They think maybe it’s not just a toy

Assuming they give it a week and 6–10 exercises

To motivate and introduce, I use search

Example where eager evaluation is not so good:

- Find square multiple of 5 greater than 137

```
(find-in-list
  (lambda (n) (> n 137))
  (filter (lambda (n) (= (mod n 5) 0))
          (map square (integer-range 0 13))))
```

How high do you look? Is `[0..13]` high enough?

Mechanism: infinite lazy data structures

```
-> (val-rec ones (cons 1 ones))
... : int list
```

```
-> (val-rec nats (cons 0 (map ((curry +) 1) nats)))
... : int list
-> (full (take 10 nats))
(0 1 2 3 4 5 6 7 8 9) : int list
```

More mechanism: Incompletely evaluated list

```
-> (val squares (map square nats))
... : int list
-> (full (take 13 squares))
(0 1 4 9 16 25 36 49 64 81 100 121 144) : int list
-> (car (drop 19 squares))
361 : int
-> squares
(0 1 4 9 16 25 36 49 64 81 100 121
   144 ... ... ... ... ... ... 361 ...) : int list
```

Many small examples that do something

Putting infinite sequences to work (thanks to John Hughes)

- Searching for numbers
- Square roots and cube roots (scaled integers)

```
-> (full (take 8 (approximate-roots 9)))
(9000 5000 3400 3023 3000 3000 3000 3000) : int list
-> (full (take 8 (approximate-roots 2)))
(2000 1500 1416 1414 1414 1414 1414 1414) : int list
```

- Multiple examples of convergence
- Prime numbers by trial division
- Backing up CDs to DVDs (bin packing)

Speedup via memoization

- Naive Fibonacci

Medium-sized example: Boolean satisfiability

They've done it in an eager language by passing continuations

Two new approaches that rely on laziness

- `(filter ((curry satisfied-by?) formula)`
  `(all-assignments formula))`
- Backtracking search *á la* "Replace failure by a list of successes"

(Neither approach dominates!)

Integrative example: Paragraphs into lines

```
But, in a larger sense, we can not dedicate, we can not
consecrate, we can not hallow this ground. The brave men,
living and dead, who struggled here, have consecrated it,
far above our poor power to add or detract. The world will
little note, nor long remember what we say here, but it can
never forget what they did here. It is for us the living,
rather, to be dedicated here to the unfinished work which
they who fought here have thus far so nobly advanced. It is
rather for us to be here dedicated to the great task
remaining before us --- that from these honored dead we take
increased devotion to that cause for which they gave the
last full measure of devotion --- that we here highly
resolve that these dead shall not have died in vain --- that
this nation, under God, shall have a new birth of freedom
--- and that government of the people, by the people, for
the people, shall not perish from the earth.
```

Greedy

```
But, in a larger sense, we can not dedicate, we can not
consecrate, we can not hallow this ground. The brave men,
living and dead, who struggled here, have consecrated it,
far above our poor power to add or detract. The world will
little note, nor long remember what we say here, but it can
never forget what they did here. It is for us the living,
rather, to be dedicated here to the unfinished work which
they who fought here have thus far so nobly advanced. It
is rather for us to be here dedicated to the great task
remaining before us --- that from these honored dead we
take increased devotion to that cause for which they gave
the last full measure of devotion --- that we here highly
resolve that these dead shall not have died in vain --- that
this nation, under God, shall have a new birth of freedom
--- and that government of the people, by the people, for
the people, shall not perish from the earth.
```

Knuth-Plass

(Both sides monospaced)

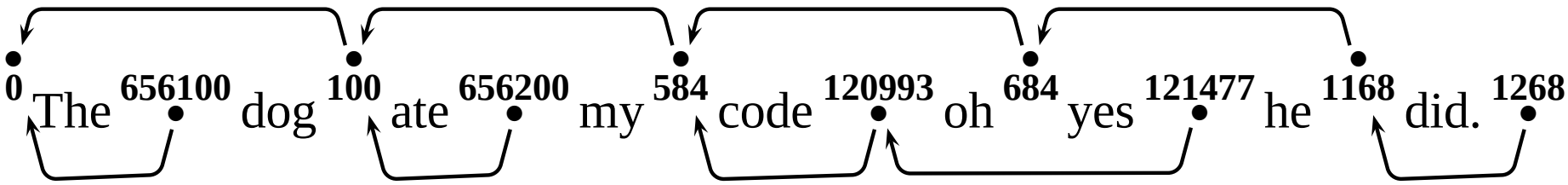
Knuth-Plass on a simple paragraph

Input: “The dog ate my code oh yes he did.”

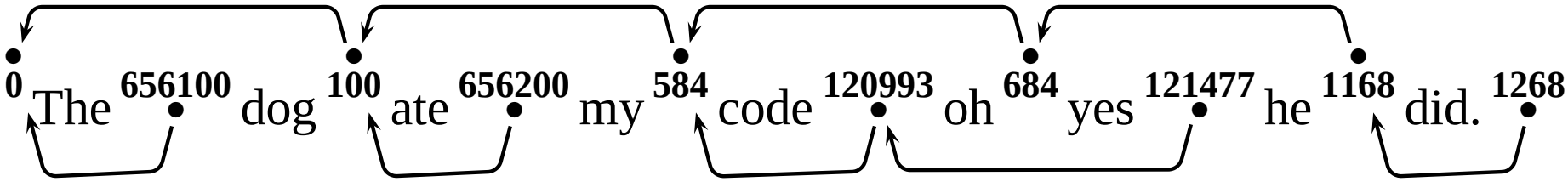
Output for 7 columns, shrink 1, stretch 2:

```
The dog
ate my
code oh
yes he
did.
```

“Best” breaks and their costs:

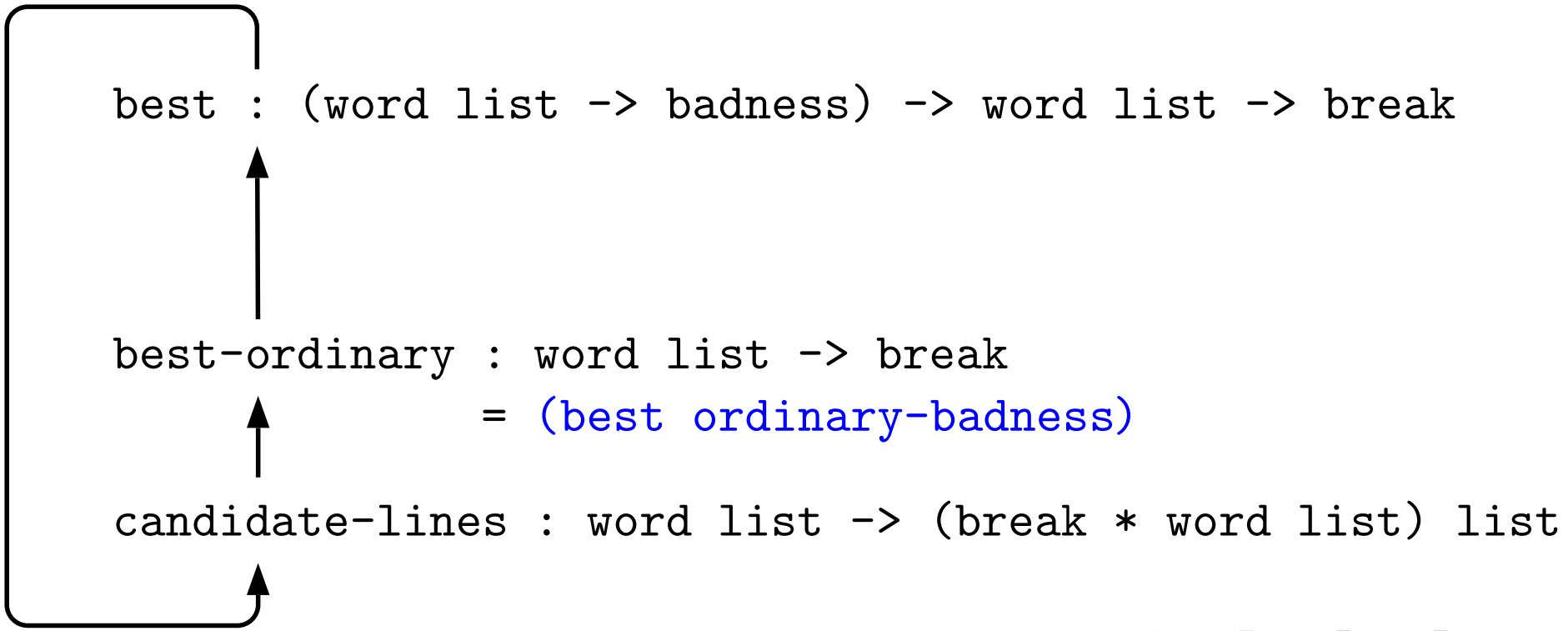


Core of solution uses a mutually recursive nest

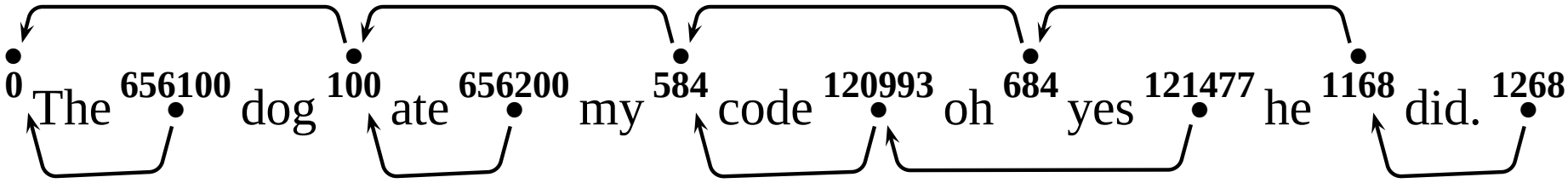


Best breakpoint computed using

```
ordinary-badness, last-badness:  word list -> badness
```



Memoization makes it practical



Insert memo list into recursion:

```
best : (word list -> badness) -> word list -> break
```

```
memo : break list
     = (map (best badness) (tails words))
```

One new definition

```
best-ordinary : word list -> break
              = (lambda (ws) (... memo ... ws ...))
```

One changed definition

```
candidate-lines : word list -> (break * word list) list
```

Laziness + Memoization = Gettysburg Address

Four score and seven years ago our fathers brought forth on this continent a new nation, conceived in liberty, and dedicated to the proposition that all men are created equal.

Now we are engaged in a great civil war, testing whether that nation, or any nation, so conceived and so dedicated, can long endure. We are met on a great battle-field of that war. We have come to dedicate a portion of that field, as a final resting place for those who here gave their lives that that nation might live. It is altogether fitting and proper that we should do this.

But, in a larger sense, we can not dedicate, we can not consecrate, we can not hallow this ground. The brave men, living and dead, who struggled here, have consecrated it, far above our poor power to add or detract. The world will little note, nor long remember what we say here, but it can never forget what they did here. It is for us the living, rather, to be dedicated here to the unfinished work which they who fought here have thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us --- that from these honored dead we take increased devotion to that cause for which they gave the last full measure of devotion --- that we here highly resolve that these dead shall not have died in vain --- that this nation, under God, shall have a new birth of freedom --- and that government of the people, by the people, for the people, shall not perish from the earth.

Greedy

Four score and seven years ago our fathers brought forth on this continent a new nation, conceived in liberty, and dedicated to the proposition that all men are created equal.

Now we are engaged in a great civil war, testing whether that nation, or any nation, so conceived and so dedicated, can long endure. We are met on a great battle-field of that war. We have come to dedicate a portion of that field, as a final resting place for those who here gave their lives that that nation might live. It is altogether fitting and proper that we should do this.

But, in a larger sense, we can not dedicate, we can not consecrate, we can not hallow this ground. The brave men, living and dead, who struggled here, have consecrated it, far above our poor power to add or detract. The world will little note, nor long remember what we say here, but it can never forget what they did here. It is for us the living, rather, to be dedicated here to the unfinished work which they who fought here have thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us --- that from these honored dead we take increased devotion to that cause for which they gave the last full measure of devotion --- that we here highly resolve that these dead shall not have died in vain --- that this nation, under God, shall have a new birth of freedom --- and that government of the people, by the people, for the people, shall not perish from the earth.

Knuth-Plass

I like some of this

Three things people can do

- Have fun with infinite lists
- Write search using modular generate and test
- Memoize

Two things people can gawk at

- Clean dynamic programming
- Line breaking

Some things concern me

- Who cares about sequences of numbers?
- Line breaking may be too big a leap
- Not enough good exercises!

Two questions you can answer

If you've learned a little lazy functional programming,

- What do you know?
- What can you do?

Randomized improvement of greedy algorithms

"Bubble search" concept from Lesh and Mitzenmacher

```
bubbleSearch
  :: (result -> result -> Ordering) -- quality
  -> ([a] -> result)     -- greedy search algorithm
  -> Double              -- biased coin
  -> [a]                 -- items to be searched
  -> Random [result]     -- monadic, monotone results
```

(E.g., type `a` is a CD and `result` is a packing of DVDs.)

Some ideas for exercises

Smallest prime at least 1000 and contains a 7

Memoization combinators?

Bubble search (higher-order improvement for greedy algorithms)

Stupid line-breaking tricks

Add lazy streams to an eager language

μHaskell concrete syntax is small

Definition forms:

- `val`, `val-rec`, `define`, datatype definition
- top-level expression

Expression forms:

- literal, variable, application, λ-abstraction
- let-binding (`let`, `let*`, `letrec`)
- `if`
- non-nested `case` (datatype elimination, forcing)

Numerical example: square root (thanks RJMH)

$\sqrt{n}$ approximations: $x_{i+1} = \frac{1}{2}(x_i + n/x_i)$

```
(define approximate-roots (n)  ;;; on SCALED integers
  (letrec ((n# (scale n))
           (roots-from (lambda (x_i)
              (let ((x_{i+1} (/ (+s x_i (/s n# x_i)) 2)))
                 (cons x_i (roots-from x_{i+1}))))))
    (roots-from n#)))
```

Transcript:

```
-> (val three (approximate-roots 9))
... : int list
-> (full (take 8 three))
(9000 5000 3400 3023 3000 3000 3000 3000) : int list
-> (full (take 8 (approximate-roots 2)))
(2000 1500 1416 1414 1414 1414 1414 1414) : int list
```