2013W1-lecture17

October 20, 2013

Contents

1 Question of the Day

Say, how do you compute the infinite list of primes in Haskell? And, how does that display the value of chaining lazy processes?

Gee. I'm glad you asked that question.

```
 SOLUTION
-- A little syntax primer:
-- + Lines starting with -- are comments
-- + The colon operator : is an infix version of cons in Racket.
-- + We can partially evaluate an infix operator by putting it in parens
--   and giving it zero or one of its arguments.  For example, (+) is
--   the addition function (as a normal, non-infix function).  (3 -)
--   is a function of one argument that substracts its argument from 3.
-- + We can convert two-argument functions into infix operators by
--   enclosing them in backquotes like ‘mod‘.
-- + The . operator is function composition.  So, (f . g)(x) is the same
--   as f(g(x)).

ones = 1 : ones

naturals = 1 : (zipWith (+) ones naturals)

-- Look Ma, no base case!
sieve (prime:nums) = prime : sieve (filter (not . (\x -> x ‘mod‘ prime == 0)) nums)

-- In other words, the result of sieving (with, roughly, the Sieve of
-- Eratosthenes) out the numbers (prime:nums) is first prime (which is
-- assumed to be a prime) and then the result of sieving out what we get
-- back from filtering out multiples of prime from the remaining numbers.

-- So, once we’ve processed 2, we’ve introduced a filter "in the
-- pipeline" that will eliminate all multiples of 2.  Once we’ve
-- processed 3, we’ve introduced a filter for all multiples of 3.  Etc.

-- And here are the primes themselves.  We start with 2, since it’s our first prime.
-- Any time we begin a loop or recursive process by providing a first valid value,
-- we refer to that as "priming the pump", but it somehow seems ever-so-appropriate
-- here, doesn’t it?
primes = sieve (tail naturals)


-- Now, let’s run this in hugs (love that name!) and try:
--  take 10 primes
--
-- Warning: don’t just say:
--  primes
--
-- What would happen if you did that?
```

2 Logistics

2.1 Some Midterm Evaluation Notes

2.1.1 Overall Comments

Comments were quite "spread out", but a few comments stood out for lecture (and the course in gen'l):

- Understanding how semantics differ across languages was the most commonly cited value of the course
  - Good to hear! We will continue to try to bring in examples from real languages, and collected quite a few promising thoughts and questions from the fourth part of the eval
- Using a different language than Racket (or plai-typed) was the most common recommendation for improvement
  - We (sort of) plan for that to happen on the fifth programming assignment. I've personally been somewhat unimpressed by plai-typed as a language, and I think some of the pain we're suffering is due to using that rather than Racket (or at least plai).
  - That said, Scheme-like languages are very traditional for studying programming languages because they have relatively simple and clean syntax and semantics. (Compare to the level of detail we'd need to get to learning the semantics of C++ if we used that as our underlying language! You've seen me bring up the specification a few times. It's **huge** and terribly complex.)
- Requests for more practice exercises (and/or clearer practice exercises than the midterm prep exercises) was next
  - I'll do my best. Can you help by pinging me on Piazza with requests for what you'd like practice on?
  - Here's some questions to get started:
    - Implement a new syntactic form `count-bindings`. It takes a single argument—a symbol—and evaluates to the number of bindings for that symbol presently in the environment (so, more than 1 if there's both a binding and at least one shadowed binding).

        * Once we have one-argument functions, create multiple-argument functions and applications by desugaring them to one-argument functions and applications.

For tutorial, most common comments:

- Tutorial content has been very helpful
- However, it's sometimes hard to tell whether everything has been covered. (Clearer notes/learning goals?)

For assignments, most common comments:

- Generally useful
- Instructions could be clearer
  - (Can you post concrete examples to Piazza? Anon posting is fine, of course!)
- PA2 (ParselTest) did not contribute much
  - In retrospect, I agree. I'll see if I can improve that one for the future!

2.2 Programming Assignment #4

We're releasing it today. The milestone will probably be due in one week. It will likely be due two weeks from today, but we need to determine when the demos will occur. (The midterm gets in the way a bit.)

2.3 Neat PL research talk upcoming: 24 Oct, 3:30-4:30PM, X836

One bonus point for attending and posting a brief summary of the talk from the CPSC 311 perspective.

Two bonus points for attending and posting a thoughtful, thorough discussion of the talk.

Three bonus points for being the speaker. :)

TypeScript is a programming language whose goal is to support development of large JavaScript programs. TypeScript is a superset of the current JavaScript standard (ECMAScript 5) that adds an optional static type system to JavaScript. TypeScript exists only to support high-level thinking about JavaScript programs; it has no impact on runtime behavior. Because of this, TypeScript is an example of "types for tooling" vs. the more traditional idea of "types for runtime safety." TypeScript has a novel design for type inference; the goal of the design is to provide maximum convenience (few annotations required) and transparency (chains of inference are clear and local). The TypeScript compiler, incremental static analysis tools, and specification are open source (see typescriptlang.org). Several million lines of TypeScript are part of shipping Microsoft products. Since the community preview release in October, 2012, several 100K+ line TypeScript projects have grown up outside of Microsoft and the TypeScript community has created a site, at github.com, that holds over 100 community-maintained TypeScript descriptions of popular JavaScript frameworks such as jQuery.

Steve Lucco is a Technical Fellow at Microsoft, where he is responsible for Microsoft's web development tools and runtimes. He led the development of Microsoft's Chakra JavaScript engine, which powers Internet Explorer. Currently, Chakra is 30% faster than Chrome V8 on SunSpider, the most widely cited JavaScript benchmark. He started the TypeScript team and contributes to the design and implementation of TypeScript.

3 Lazy Evaluation with Environments

Back to lecture 16!

4 What have we learned today?

- From QotD: A (little) bit of the practical value of lazy evaluation.

```
 ORGMODECONFIG
#+DRAWERS: SOLUTION ORGMODECONFIG
#+COMMENT TODO: change to d:nil (or delete) to not export SOLUTION drawers
#+OPTIONS:   d:t
```