CSE 341: Programming Languages

Spring 2007
Lecture 19 — Delayed Evaluation & Streams
Delayed Evaluation

For each language construct, there are rules governing when subexpressions get evaluated. In ML, Scheme, and Java:

- function arguments are “eager” \((\text{call-by-value})\)
- conditional branches are not

We could define a language in which function arguments were not evaluated before call, but instead at each use of argument in body. \((\text{call-by-name})\)

- Sometimes faster: \((\text{lambda} \ (x) \ 3)\)
- Sometimes slower: \((\text{lambda} \ (x) \ (+ \ x \ x))\)
- Equivalent if function argument has no effects/non-termination
Streams

- A stream is an “infinite” list — you can ask for the rest of it as many times as you like and you’ll never get null.

- The universe is finite, so a stream must really be an object that acts like an infinite list.

- The idea: use a function to describe what comes next.

Note: Connection to UNIX pipes
An Example

The Riemann zeta function:

\[ \zeta(s) = \prod_{i \geq 1} \frac{1}{1 - p_i^{-s}} \]

where \( p_i \) is the \( i \)th prime.

Curiously,

\[ \zeta(2) = \frac{\pi^2}{6} \]