CSE 341: Programming Languages

Spring 2006
Lecture 18 — Delayed Evaluation, Memoization & Streams
Call-by: Best of both worlds?

call-by-value: eval every argument before call

call-by-name: eval arguments at every actual use (via thunk)

call-by-need ("lazy" evaluation): Evaluate every argument the first time it’s used. Save answer for subsequent uses.

• Asymptotically it’s the best

• But behind-the-scenes bookkeeping can be costly

• And it’s hard to reason about with effects
  – Typically used in (sub)languages without effects

• Nonetheless, a key idiom with syntactic support in Scheme
  – And related to memoization
Memoization

A “cache” of previous results is equivalent if results cannot change.

• Could be slower: cache too big or computation too cheap
• Could be faster: just a lookup
  – On homework: An example where it’s a lot faster by preventing an exponential explosion.

An association list is not the fastest data structure for large memo tables, but works fine for 341.

Question: Why does assoc return the pair?
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: Deep connection to sequential feedback circuits

Note: Connection to UNIX pipes