

CSE 341: Programming Languages

Spring 2006

Lecture 17 — Local Binding, Delayed Evaluation, Thunks

Today

- Local Bindings
- Delaying evaluation: Function bodies evaluated only at application
- Key idioms of delaying evaluation
 - Conditionals
 - Streams
 - Laziness
 - Memoization
- In general, evaluation rules defined by language semantics
 - Some languages have “lazy” function application!

Local bindings

There are 3 forms of local bindings with different semantics:

- `let`
- `let*`
- `letrec`

Also, in function bodies, a sequence of definitions is equivalent to `letrec`.

But at top-level redefinition is assignment!

This makes it ghastly hard to encapsulate code, but in practice:

- people assume non-malicious clients
- implementations provide access to “real primitives”

For your homework, assume top-level definitions are immutable.

Delayed Evaluation

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

- function arguments are “eager” (*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. (*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

Thunks

One (among several) meanings of “think” is just a function taking no arguments, which works great for delaying evaluation.

- Instead of passing a value directly, pass a thunk (function) which yields the value when it is called

If thunks are lightweight enough syntactically, why not make if eager?
(Smalltalk does this!)