Menu

- Using Laziness
- Evidence of Laziness? (Quiz Results)
- Laziness through Truthiness (Static Type Checking)

Lazy Evaluation Recap

- Don't evaluate expressions until their value is really needed
  - We might save work this way, since sometimes we don't need the value of an expression
  - We might change the meaning of some expressions, since the order of evaluation matters
- Change the Evaluation rule for Application
- Use thunks to delay evaluations

Lazy Application

```
def evalApplication(expr, env):
  subexprvals = map (lambda sexpr: meval(sexpr, env), expr)
  return mapply(subexprvals[0], subexprvals[1:])
```

```
def evalApplication(expr, env):
  # make Thunk object for each operand expression
  ops = map (lambda sexpr: Thunk(sexpr, env), expr[1:])
  return mapply(forceeval(expr[0], env), ops)
```


Lazy Data Structures

```
(define cons
  (lambda (a b)
    (lambda (p)
      (if p a b))))

(define car
  (lambda (p) (p #t)))

(define cdr
  (lambda (p) (p #f)))
```

Note: for PS7, you are defining these as *primitives*, which would not evaluate lazily.

Using Lazy Pairs

```
(define cons
  (lambda (a b)
    (lambda (p)
      (if p a b))))

(define car
  (lambda (p) (p #t)))
(define cdr
  (lambda (p) (p #f)))
```

```
LazyCharme> (define pair (cons 3 error))
LazyCharme> pair
<Procedure ['p'] / ['if', 'p', 'a', 'b']>
LazyCharme> (car pair)
3
LazyCharme> (cdr pair)
Error: Undefined name: error
```

Infinite Lists

```
(define ints-from
   (lambda (n)
      (cons n (ints-from (+ n 1)))))
```

```
LazyCharme> (define allnaturals (ints-from 0))
LazyCharme> (car allnaturals)
0
LazyCharme> (car (cdr allnaturals))
1
LazyCharme> (car (cdr (cdr (cdr allnaturals)))))
4
```


Infinite Fibonacci Sequence

```
(define fibo-gen (lambda (a b)
    (cons a (fibo-gen b (+ a b)))))

(define fibos (fibo-gen 0 1))

(define get-nth (lambda (lst n)
    (if (= n 0) (car lst)
       (get-nth (cdr lst) (- n 1)))))

(define fibo
  (lambda (n)  (get-nth fibos n)))
```


Alternate Implementation

```
(define merge-lists
  (lambda (lst1 lst2 proc)
    (if (null? lst1) null
        (if (null? lst2) null
           (cons (proc (car lst1) (car lst2))
                 (merge-lists (cdr lst1) (cdr lst2) proc))))))

(define fiboms
  (cons 0
        (cons 1
              (merge-lists fiboms (cdr fiboms) +))))
```


Quiz Results

Quiz Answers

1. Programming languages designed by John Backus: Fortran, FP, FL
   (BNF – not a programming language)
2. What did Gödel prove?
   That any axiomatic system powerful enough to express "This statement cannot be proven in the system" must be incomplete.
3. What does SSS0 mean? 3

Quiz 4: Environment Class

```
class Environment:
  def __init__(self, parent):
    self._parent = parent
    self._frame = { }
  def addVariable(self, name, value):
    self._frame[name] = value
  def lookupVariable(self, name):
    if self._frame.has_key(name):
      return self._frame[name]
    return [redacted]
    else:
      evalError("Undefined name: %s" % (name))
```

Quiz 5: Viruses

- Is it possible to define a procedure that protects computer users from all viruses?

Here's one procedure:
o Unplug the computer from the power
o Encase it in concrete
o Throw it in the Potomac River

This is a very different question from the "Is it possible to determine if a procedure specification is a virus?" question (which we proved in class is impossible by showing how a solution to it could be used to solve the Halting Problem.

Types

Types

Numbers

Strings

programs that halt

Colors

Beatle's Songs that don't end on the Tonic

lists of lists of lists of anything

- Type is a (possibly infinite) set of values
- You can do some things with some types, but not others

Why have types?

- Detecting programming errors: (usually) better to notice error than report incorrect result
- Make programs easier to read, understand and maintain: thinking about types can help understand code
- Verification: types make it easier to prove properties about programs
- Security: can use types to constrain the behavior of programs

Types of Types

Does regular Scheme have types?

```
> (car 3)
car: expects argument of type <pair>; given 3
> (+ (cons 1 2))
+: expects argument of type <number>; given (1 . 2)
```

Yes, without types (car 3) would produce some silly result. Because of types, it produces a type error.

Type Taxonomy

- Latent vs. Manifest
  - Are types visible in the program text?
- Static vs. dynamic checking
  - Do you have to run the program to know if it has type errors?
- Weak vs. Strong checking
  - How strict are the rules for using types?
    - (e.g., does the predicate for an if need to be a Boolean?)
  - Continuum (just matter of degree)

Scheme/Python/Charme

- Latent or Manifest?
  - All have latent types (none visible in code)
- Static or Dynamic?
  - All are dynamic (checked when expression is evaluated)
- Weak or Strong?
  - Which is the strictest?

Strict Typing

```
Scheme> (+ 1 #t)
+: expects type <number> as 2nd argument, given: #t; other arguments were: 1
Python>>> 1 + True
2
Charme> (+ 1 #t)
2
```


Scheme/Python/Charme → Java/StaticCharme

- Scheme, Python, and Charme have Latent, Dynamically checked types
  - Don't see explicit types when you look at code
  - Checked when an expression is evaluated
- Java, StaticCharme have Manifest, Statically checked types
  - Type declarations must be included in code
  - Types are checked statically before running the program (Java: not all types checked statically)

Java Example

```
class Test {
  int tester (String s)
  {
    int x;
    x = s;
    return "okay";
  }
}
```

The result is an integer

The parameter must be a String

The place x holds an integer

```
> javac types.java
types.java:5: Incompatible
type for =. Can't convert
java.lang.String to int.
      x = s;
        ^
types.java:6: Incompatible
type for return. Can't convert
java.lang.String to int.
      return "okay";
      ^
2 errors
```

javac compiles (and type checks) the program. It does ***not*** execute it.

What do we need to do change our Charme interpreter to provide manifest types?

Charge

- PS7 Due Friday
- Friday: Finish StaticCharme interpreter
- Monday: Project ideas and team requests for PS9