Types of Types

One-Slide Summary

- In **lazy evaluation**, expressions are not evaluated until their values are needed. We can use lazy evaluation to program with **infinite data structures**, such as a list of all natural numbers.
- **A type** is a (possibly infinite) set of values.
- Each type supports a set of **valid operations**.
- Types can be latent or manifest, static or dynamic, strong or weak.
- We can change the Charme interpreter to support manifest (program visible) types.

Outline

- Administration
- Lazy Evaluation Recap
- Quiz Results
- Types
  - Type Taxonomy
  - Static Charme
    - Charme with Manifest Types

Administrivia

- Start PS7 Now
- **Kinga’s Web Fault Research Survey**
  - Worth 2 points of Extra Credit on Exam 2
  - Plus possibly $$$ …
- **2009 Computing and Communication Scholarship for Undergraduate Women**
  - $1000 merit scholarship, due June 30th

The Textbook

- I get the sense that some of the students are attempting to read the book on-line. I would encourage everyone to read it on paper. It is pretty well established that people read faster and understand better on paper than on the screen.
  - David Evans, Course Book Author

Problem Set 8

- Understand and modify a dynamic web application
- Already posted
- Due Monday April 13th

Problem Set 9

- Team requests and ideas due Friday April 10th (email me before midnight)
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

```python
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

```scheme
(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

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

Infinite Lists

```scheme
(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 allnaturals))))
4

Infinite Fibonaccci Sequence

```scheme
(define fibo-gen
    (lambda (a b)
       (cons a (fibo-gen b (+ a b)))))
(define fibos
    (fibo-gen 0 1))
```

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

LazyCharme> (define get-nth (lambda (lst n)
    (if (= n 0) (car lst)
        (get-nth (cdr lst) (- n 1)))))
LazyCharme> (define fibo (lambda (n) (get-nth fibos n)))
LazyCharme> (get-nth fibos 4)
34

LazyCharme> (define get-nth (lambda (lst n)
    (if (= n 0) (car lst)
        (get-nth (cdr lst) (- n 1)))))
LazyCharme> (define fibo (lambda (n) (get-nth fibos n)))
LazyCharme> (get-nth fibos 4)
34
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 ;; merge-list variant
   (cons 0
         (cons 1
               (merge-lists fiboms (cdr fiboms) +))))

Come back and understand this slide to study for the exams.

Quiz Results

Quiz Answers

• 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]
      elif self._parent:
         return self._parent.lookupVariable(name)
      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).

Liberal Arts Trivia: Cognitive Science

• This philosophy of mind dominated for the first half of the 20th century. It developed as a reaction to the inadequacies of introspectionism. In it, all things which organisms do - including acting, thinking and feeling - should be regarded as actions or reactions, usually to the environment. It holds that there are no philosophical differences between publicly observable processes (actions) and privately observable processes (thinking and feeling).

• Bonus: B.F. Who?
Liberal Arts Trivia: Civil Rights

• The landmark 1967 Supreme Court case *Loving v. Virginia* declared Virginia’s anti-miscegenation statue, the “Racial Integrity Act of 1924”, unconstitutional. This effectively ended laws preventing what?

Types

- Numbers
- Strings
- programs that halt
- Colors
- Beatle’s Songs that don’t end on the Tonic
- lists of lists of lists of anything

- A **Type** is a (possibly infinite) set of values
- You can do some things with some types, but not others
  - Each Type has associated valid operations

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?
  - You tell me!

**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**

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

The result is an integer
The place x holds an integer
The parameter must be a String

> javac types.java

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

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

Liberal Arts Trivia: Media Studies
- This technique in film editing combines a series of short shots into a sequence of condensed narrative. It is usually used to advance the story as a whole and often to suggest the passage of time.

Liberal Arts Trivia: United Kingdom History
- This United Kingdom Tory prime minister was most famous for his military work during the Peninsular Campaign and the Napoleonic Wars. He was nicknamed the “Iron Duke” because of the iron shutters he had fixed to his windows to stop pro-reform mobs from breaking them - as an MP he was opposed to reform. It is unclear whether the well-known beef tenderloin, pate and puff pastry dish is named after him.

Programming Language Design Space
Expressiveness

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Python</th>
<th>Charme</th>
<th>LazyCharme</th>
<th>StaticCharme</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>&quot;Truthiness&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Manifest Types
- Need to change the grammar rules to include types in definitions and parameter lists

Definition ::= (define Name : Type Expression)
Parameters ::= ε | Parameter Parameters
Parameter ::= Name : Type
Type ::= ??

Types of Types
- Latent → Manifest: change grammar, represent types
- Dynamically Checked → Statically Checked: typecheck expressions before eval
Types in Charme

CType ::= CPrimitiveType
CType ::= CProcedureType
CType ::= CProductType

CPrimitiveType ::= Number | Boolean
CProcedureType ::= (CProductType -> Type)
CProductType ::= (CTypeList)
CTypeList ::= CType CTypeList

Representing Types

CType ::= CPrimitiveType | CProcedureType | CProductType
CPrimitiveType ::= Number | Boolean
CProcedureType ::= (CProductType -> Type)
CProductType ::= (CTypeList)
CTypeList ::= CType CTypeList

CPrimitiveType

class CPrimitiveType(CType):
    def __init__(self, s):
        self._name = s
    def __str__(self):
        return self._name
    def isPrimitiveType(self):
        return True
    def matches(self, other):
        return other.isPrimitiveType() 
                  and self._name == other._name

Get out paper!
CProcedureType

class CProcedureType(CType):
    def __init__(self, args, rettype):
        self._args = args
        self._rettype = rettype
    def __str__(self):
        return "(" + str(self._args) + " -> " + str(self._rettype) + ")"
def isProcedureType(self):
    return True
def getReturnType(self):
    return self._rettype
def getParameters(self):
    return self._args

def matches(self, other):
    return other.isProcedureType() \
           and self.getParameters().matches(other.getParameters()) \
           and self.getReturnType().matches(other.getReturnType())

CProductType

class CProductType(CType):
    def __init__(self, types):
        self._types = types
    def __str__(self):
        ???
def isProductType(self):
    return True
def matches(self, other):
    if other.isProductType():
        st = self._types
        ot = other._types
        if len(st) == len(ot):
            for i in range(0, len(st)):
                if not st[i].matches(ot[i]):
                    return False
            # reached end of loop => all matched
            return True
    return False

Homework

• Show up to Gary McGraw guest lecture on Monday
• Problem Set 7 due Monday
• Problem Set 9 Team Requests Friday Apr 10th