61A LECTURE 25 – DECLARATIVE PROGRAMMING

Steven Tang and Eric Tzeng
August 6, 2013

Midterm grades up
- Class did well as a whole!
- glookup -s test2
- Regrades: Talk to your TA in person.

Laziness
Recall our previous sequence interface:
- A sequence has a finite, known length
- A sequence allows element selection for any element

In the cases we’ve seen so far, satisfying the sequence interface requires storing the entire sequence in a computer’s memory

Problems?
- Infinite sequences - primes, positive integers
- Really large sequences - all Twitter tweets, votes in a presidential election

Announcements
- Proj4 has been out
- Due in 7 days – Start if you haven’t!
- Recursive art contest deadline one day before project is due
- Future “homework” assignment will be to vote on your favorite submissions
- Final exam next Thursday
  - 7-10pm in 1 Pimentel
  - Any conflicts – notify us immediately
- Final exam review session this weekend
  - See Piazza Poll to vote on your time
  - Potential extra credit – more information later in the week

Laziness
Recall our previous sequence interface:
- A sequence has a finite, known length
- A sequence allows element selection for any element

In the cases we’ve seen so far, satisfying the sequence interface requires storing the entire sequence in a computer’s memory

Problems?
- Infinite sequences - primes, positive integers
- Really large sequences - all Twitter tweets, votes in a presidential election

Announcements
- Proj4 has been out
- Due in 7 days – Start if you haven’t!
- Recursive art contest deadline one day before project is due
- Future “homework” assignment will be to vote on your favorite submissions
- Final exam next Thursday
  - 7-10pm in 1 Pimentel
  - Any conflicts – notify us immediately
- Final exam review session this weekend
  - See Piazza Poll to vote on your time
  - Potential extra credit – more information later in the week

Laziness
Recall our previous sequence interface:
- A sequence has a finite, known length
- A sequence allows element selection for any element

In the cases we’ve seen so far, satisfying the sequence interface requires storing the entire sequence in a computer’s memory

Problems?
- Infinite sequences - primes, positive integers
- Really large sequences - all Twitter tweets, votes in a presidential election

Announcements
- Proj4 has been out
- Due in 7 days – Start if you haven’t!
- Recursive art contest deadline one day before project is due
- Future “homework” assignment will be to vote on your favorite submissions
- Final exam next Thursday
  - 7-10pm in 1 Pimentel
  - Any conflicts – notify us immediately
- Final exam review session this weekend
  - See Piazza Poll to vote on your time
  - Potential extra credit – more information later in the week

Laziness
Recall our previous sequence interface:
- A sequence has a finite, known length
- A sequence allows element selection for any element

In the cases we’ve seen so far, satisfying the sequence interface requires storing the entire sequence in a computer’s memory

Problems?
- Infinite sequences - primes, positive integers
- Really large sequences - all Twitter tweets, votes in a presidential election

Announcements
- Proj4 has been out
- Due in 7 days – Start if you haven’t!
- Recursive art contest deadline one day before project is due
- Future “homework” assignment will be to vote on your favorite submissions
- Final exam next Thursday
  - 7-10pm in 1 Pimentel
  - Any conflicts – notify us immediately
- Final exam review session this weekend
  - See Piazza Poll to vote on your time
  - Potential extra credit – more information later in the week

Laziness
Recall our previous sequence interface:
- A sequence has a finite, known length
- A sequence allows element selection for any element

In the cases we’ve seen so far, satisfying the sequence interface requires storing the entire sequence in a computer’s memory

Problems?
- Infinite sequences - primes, positive integers
- Really large sequences - all Twitter tweets, votes in a presidential election

Announcements
- Proj4 has been out
- Due in 7 days – Start if you haven’t!
- Recursive art contest deadline one day before project is due
- Future “homework” assignment will be to vote on your favorite submissions
- Final exam next Thursday
  - 7-10pm in 1 Pimentel
  - Any conflicts – notify us immediately
- Final exam review session this weekend
  - See Piazza Poll to vote on your time
  - Potential extra credit – more information later in the week
Streams
A stream is a recursive list with an explicit first element and a lazily computed rest of the list

```python
class Stream(Rlist):
    """A lazily computed recursive list."""
    def __init__(self, first, compute_rest=lambda: Stream.empty):
        assert callable(compute_rest)
        self.first = first
        self._compute_rest = compute_rest
        self._rest = None

    @property
    def rest(self):
        """Return the rest of the stream, computing it if necessary."""
        if self._compute_rest is not None:
            self._rest = self._compute_rest()
            self._compute_rest = None
        return self._rest
```

Integer Streams
An integer stream is a stream of consecutive integers

An integer stream starting at \( k \) consists of \( k \) and a function that returns the integer stream starting at \( k+1 \)

```python
def integer_stream(first=1):
    """Return a stream of consecutive integers, starting with first."
    def compute_rest():
        return integer_stream(first+1)
    return Stream(first, compute_rest)
```

A Stream of Primes
The stream of integers not divisible by any \( k \leq n \) is:
- The stream of integers not divisible by any \( k < n \),
- Filtered to remove any element divisible by \( n \)
- This recurrence is called the Sieve of Eratosthenes

\[ 2, 3, 5, 7, 11, 13 \]

```python
def primes(istream):
    """Return a stream of primes, given a stream of consecutive integers."""
    def compute_rest():
        not_divisible = lambda x: x % istream.first != 0
        return primes(filter_stream(not_divisible, istream.rest))
    return Stream(istream.first, compute_rest)
```

Try it
- Write a function add_streams that takes two streams and returns a new stream formed by summing corresponding elements in the argument streams. Stop when either of the streams ends.
- Bonus: see if you can use add_streams to define the Fibonacci stream!

Answers
```python
def add_streams(s1, s2):
    if s1 is Stream.empty or s2 is Stream.empty:
        return Stream.empty
    return Stream(s1.first + s2.first,
                  lambda: add_streams(s1.rest, s2.rest))

fibs = Stream(0,
              lambda: Stream(1,
                             lambda: add_streams(fibs, fibs.rest)))
```
Last “super-big” topic in course

- Lot to cover this lecture...
- We will continue this topic tomorrow as well
- Need to finish questions 1-4 on your Scheme project for Logic programming to work
- Bring your version of scheme to lab tomorrow!

Databases

A database is a collection of records (tuples) and an interface for adding, editing, and retrieving records.

The Structured Query Language (SQL) is perhaps the most widely used programming language on Earth.

```
SELECT * FROM toy_info WHERE color='yellow';
```

SQL is an example of a declarative programming language.
It separates what to compute from how it is computed
The language interpreter is free to compute the result in any way it deems appropriate

Declarative Programming

The main characteristics of declarative languages:
- A “program” is a description of the desired solution
- The interpreter figures out how to generate such a solution

By contrast, in procedural languages such as Python & Scheme:
- A “program” is a description of procedures
- The interpreter carries out execution/evaluation rules

Building a universal problem solver is a difficult task
Declerative programming languages compromise by solving only a subset of all problems
They typically trade off data scale for problem complexity

Simple Facts

A simple fact expression in the Logic language declares a relation to be true

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent eisenhower fillmore))
```

The Logic Language

The Logic language is invented for this course
- Based on the Scheme project & ideas from Prolog
- Expressions are facts or queries, which contain relations
- Expressions and relations are both Scheme lists
- For example, (likes Albert dogs) is a relation
- Implementation fits on a single sheet of paper

Today’s theme:

```
http://awhimsicalbohemian.typepad.com/.a/6a0e5538b84f33883301538dea8f19970b-800wi
```

Relations are Not Procedure Calls

In Logic, a relation is not a call expression
- In Scheme, we write (abs -3) to call abs on -3
- In Logic, (abs -3 3) asserts that the abs of -3 is 3

For example, if we wanted to assert that 1 + 2 = 3:

```
(add 1 2 3)
```

Why declare knowledge in this way? It will allow us to solve problems in two directions:

```
(add 1 2 _)
(add _ 2 3)
(add 1 _ 3)
(__ 1 2 3)
```
Queries
A query contains one or more relations. The Logic interpreter returns whether (and how) they are all simultaneously satisfied.

Queries may contain variables: symbols starting with ?

logics (fact (parent delano herbert))
logics (fact (parent abraham barack))
logics (fact (parent eisenhower fillmore))
logics (query (parent abraham ?child))
Success!
child: barack
child: clinton

Recursive Facts
A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

logics (query (ancestor ?a herbert))
Success!
a: delano
b: fillmore
c: eisenhower

Hierarchical Facts
Relations can contain relations in addition to atoms:

logics (fact (dog (name abraham) (color white)))
Variables can refer to atoms or relations:

logics (query (dog (name clinton) ?color))
Success:
color: white

Searching to Satisfy Queries
The Logic interpreter performs a search in the space of relations for each query to find a satisfying assignment:

logics (query (ancestor ?a herbert))
Success!
a: delano
b: fillmore
c: eisenhower

Compound Facts
A fact can include multiple relations and variables as well:

(fact <conclusion> <hypothesis> <hypothesis> ... <hypothesis>)
Means <conclusion> is true if all <hypothesis> are true.

logics (query (child herbert delano))
Success!

logics (query (child abraham fillmore))
Failure.

logics (query (child ?child fillmore))
Success!
child: abraham
child: delano
child: grover

Variables can refer to atoms or relations:

logics (query (dog (name clinton) ?color))
Success:
color: white
Example: Combining Multiple Data Sources
Which dogs have an ancestor of the same color?

```logic
(query (dog (name ?name) (color ?color))
    (ancestor ?ancestor ?name)
    (dog (name ?ancestor) (color ?color)))
```

Success!

name: barack  color: tan  ancestor: eisenhower
name: clinton  color: white  ancestor: abraham
name: grover  color: tan  ancestor: eisenhower
name: herbert  color: brown  ancestor: fillmore

Example: Appending Lists
Two lists append to form a third list if:
- The first list is empty and the second and third are the same
  
  ```(a b c) (a b c)```
- Both of the following hold.
  - List 1 and 3 have the same first element
  - The rest of list 1 and all of list 2 append to form the rest of list 3

```logic
(fact (append-to-form () ?x ?x))
(fact (append-to-form (?a . ?r) ?y (?a . ?s))
    (append-to-form ?r ?y ?s))
```

Logic Example: Anagrams
A permutation (i.e., anagram) of a list is:
- The empty list for an empty list
- The first element of the list inserted into an anagram of the rest of the list

```logic
(fact (anagram () ()))
(fact (anagram (?a . ?r) ?b)
    (insert ?a ?b ?r)
    (anagram ?r ?b))
```

You try it out!
- Write facts to make double-elements work

```logic
(query (double-elements (3 4) ?result))
Success!
result: (3 3 4 4)
```

```logic
(query (double-elements ?start (4 4 5)))
Success!
start: (4 5)
```