61A Lecture 28

Friday, November 7

---

Announcements

- Homework 7 due Wednesday 4/8 @ 11:59pm
- Homework party Tuesday 4/7 5pm-6:30pm in 2050 VLSB
- Quiz 2 due Thursday 4/9 @ 11:59pm
- Homework 8 due Wednesday 4/15 @ 11:59pm
- Project 1, 2, 3 composition revisions due Monday 4/13 @ 11:59pm
- Project 4 due Thursday 4/23 @ 11:59pm (Big!)

---

Scheme Recursive Art Contest: Start Early!

Fall 2012 Featherweight Winner
176 Scheme Tokens

Fall 2013 Heavyweight Winner
1857 Scheme Tokens

Extra lecture on this image:
Thursday 4/9 5pm in 2050 VLSB

Dynamic Scope

The way in which names are looked up in Scheme and Python is called lexical scope (or static scope). You can see what names are in scope by inspecting the definition.

Lexical scope: The parent of a frame is the environment in which a procedure was defined.

Dynamic scope: The parent of a frame is the environment in which a procedure was called.

Special form to create dynamically scoped procedures (this special form only exists in Project 4 Scheme)

(define (mu (lambda (x) (+ x y)))
(define g (lambda (x y) (f (+ x x))))
(g 3 7)

Error: unknown identifier: y

Dynamic scope:
The parent for f’s frame is the global frame:
F
x
y
1
3
6

Tail Recursion

In Python, recursive calls always create new active frames.

Recursion and Iteration in Python

In Python, recursive calls always create new active frames.

Factorial in Python:

```
def factorial(n, k):
    if n == 0:
        return k
    else:
        return factorial(n-1, n*k)
```

Time

Space

---

Functional Programming

All functions are pure functions.

No re-assignment and no mutable data types.

Name-value bindings are permanent.

Advantages of functional programming:

- The value of an expression is independent of the order in which sub-expressions are evaluated.
- Sub-expressions can safely be evaluated in parallel or on demand (lazily).
- Referential transparency: The value of an expression does not change when we substitute one of its subexpression with the value of that subexpression.

But... no for/while statements! Can we make basic iteration efficient? Yes!

---
Tail Recursion

From the Revised Report on the Algorithmic Language Scheme:

"Implementations of Scheme are required to be properly tail-recursive. This allows the execution of an iterative computation in constant space, even if the iterative computation is described by a syntactically recursive procedure."

(define factorial n k)
  (if (zero? n)
      k
      (factorial (- n 1) (k n))))

Tail Calls

A procedure call that has not yet returned is active. Some procedure calls are tail calls. A Scheme interpreter should support an unbounded number of active tail calls using only a constant amount of space.

A tail call is a call expression in a tail context:

- The last body sub-expression in a lambda expression
- Sub-expressions 2 & 3 in a tail context if expression
- All non-predicate sub-expressions in a tail context cond
- The last sub-expression in a tail context begin

Eval with Tail Call Optimization

The return value of the tail call is the return value of the current procedure call. Therefore, tail calls shouldn't increase the environment size.

Example: Length of a List

(define length s)
  (if (null? s)
      0
      (+ 1 (length (cdr s)))))

Map and Reduce

Which Procedures are Tail Recursive?

Which of the following procedures run in constant space? Θ(1)

; Compute the length of s.
(define length s)
  (if (null? s)
      0
      (+ 1 (length (cdr s)))))

; Return whether s contains v.
(define contains s v)
  (if (null? s)
      false
      (if (= v (car s))
          true
          (contains (cdr s) v))))

; Return the n-th Fibonacci number.
(define fib n)
  (if (= n 0)
      0
      (+ (fib (- n 1)) (fib (- n 2)))))

; Return whether s has any repeated elements.
(define has-repeat s)
  (if (null? s)
      false
      (if (contains? (cdr s) (car s))
          true
          (has-repeat (cdr s)))))

; Return the nth Fibonacci number.
(define factorial n k)
  (if (zero? n)
      k
      (factorial (- n 1) (k n))))
Example: Reduce

\[
\text{(define (reduce procedure s start)}
\]
\[
\text{if (null? s) start)
\]
\[
\text{(reduce procedure (cdr s)) (procedure start (car s)))
\]

Recursive call is a tail call.
Space depends on what procedure requires

\[
\text{(reduce * '(3 4 5) 2)} \quad 120
\]

\[
\text{(reduce (lambda (x y) (cons y x)) '(3 4 5) '(2)) \quad (5 4 3 2)} \quad 17
\]

Example: Map with Only a Constant Number of Frames

\[
\text{(define (map procedure s)}
\]
\[
\text{if (null? s) m)
\]
\[
\text{(map-reverse (cdr s)) (cons (procedure (car s)) m))
\]
\[
\text{(reverse (map-reverse s nil)))
\]

\[
\text{(define (reverse s)}
\]
\[
\text{define (reverse-iter s r)}
\]
\[
\text{if (null? s) r)
\]
\[
\text{(reverse-iter (cdr s)) (cons (car s) r))}
\]
\[
\text{(reverse-iter s nil))}
\]

An Analogy: Programs Define Machines

Programs specify the logic of a computational device

Interpreters are General Computing Machine

An interpreter can be parameterized to simulate any machine

\[
\text{(define (factorial n)}
\]
\[
\text{if (zero? n) 1 (* n (factorial (- n 1))))
\]

Our Scheme interpreter is a universal machine

A bridge between the data objects that are manipulated by our programming language and the programming language itself

Internally, it is just a set of evaluation rules