Week 5: More with Macros

CSC324 Principles of Programming Languages

David Liu, Department of Computer Science

Announcements
Assignment 1

Due Sunday

Extra office hours

Not just autotested! (Rough marking scheme posted later today)

Back to my-class: methods
Code demo!

Literal keywords

```scheme
(define-syntax <syntax-name>
  (syntax-rules (<keyword> ...) 
    [<pattern> <template>] ...))
```
Problem: method that calls another method?

The problem of self
An instance method should be able to refer to the calling object.
(“I should be able to refer to myself.”)

**Approach 1**: `this/self` as a built-in keyword
(e.g., Java, C++, Ruby, JavaScript)
**Approach 2:** this/self as a method parameter
(e.g., Python, Rust)

Implementation first try: self everywhere!

```
(my-class (Point (x y))
  (method (size)
    (sqrt (+ (* x x)
             (* y y)))))
```

```
(my-class (Point (x y))
  (method (size self)
    (sqrt (+ (* (self 'x) (self 'x))
             (* (self 'y) (self 'y))))))
```
Implementation second try: auto-binding self

WWPD (What Would Python Do?)
Key ideas

1. Store instance methods in a separate class `__dict__`.
2. When looking up an instance method, automatically bind the first parameter (`self`).

Self-reference with `letrec`

```
(letrec ([me <expr>])
  me)
```
In JavaScript, `this` is bound when an instance method is called, not when it is accessed.

```javascript
var p = new Point(2, 3);
p.size(); // this == p
var f = p.size; // this unbound
```
Inheritance through chained lookups

```
[(hash-has-key? self__dict__ msg)
 (hash-ref self__dict__ msg)]

[(hash-has-key? class__dict__ msg)
 (fix-first me (hash-ref class__dict__ msg))]

[else ((attribute-error ...))]
```
Inheritance as an additional check (idea)

```scheme
([(hash-has-key? self__dict__ msg) 
 (hash-ref self__dict__ msg)] 

[(hash-has-key? class__dict__ msg) 
 (fix-first me (hash-ref class__dict__ msg))]
[else (check-superclass ...)]
```

prototypal inheritance: an object inherits from another object, called its prototype

```scheme
([(hash-has-key? self__dict__ msg) 
 (hash-ref self__dict__ msg)]

[else (check-prototype ...)]
```
Macros to manipulate control flow

Reminder of simple examples of the lab: and, or, cond as macros (implemented in terms of if).
Streams and lazy lists

Suppose we have a large collection of data.

Eager evaluation is not an option.
stream
an abstract model of a sequence of values over time

Examples
Processing large files
Delayed/slow input
Python range
**lazy list**: a stream implementation based on lists.

A **lazy list** is either:

- empty
- a value “cons” another lazy list, but where the “cons” is lazy

```scheme
(define empty-stream 'empty)
(define (stream-empty? stream) (equal? stream empty-stream))

(define-syntax stream-cons
  (syntax-rules ()
    [(stream-cons <first-expr> <rest-expr>)
      (cons (thunk <first-expr>) (thunk <rest-expr>))]))

(define (stream-first stream) ((car stream)))
(define (stream-rest stream) ((cdr stream)))
```
Note: see built-in racket/stream library.

BWAH

But What About Haskell?

In Haskell, cons (:) is already lazy!
Decoupling the production and consumption of data

(warmup)

```python
i = 0
while i < 1000:
    f(i)
    i += 1
```

vs.

```python
for i in range(0, 1000):
    f(i)
```
range :: Integer -> Integer -> [Integer]
range start end = if start >>= end
  then
    []
  else
    start : (range (start + 1) end)

range' :: Integer -> [Integer]
range' start = start : (range' (start + 1))
The bisection method

```python
def bisect(f, tol, a, b):
    # Precondition: f(a) and f(b) have different signs.
    c = (a + b) / 2
    while abs(f(c)) >= tol:
        if sign(f(a)) == sign(f(c)):
            a = c
        else:
            b = c
        c = (a + b) / 2
    return c
```
\[
\text{bisect } f \text{ tol a b } = \\
\quad \text{let } c = (a + b) / 2 \\
\quad \text{in} \\
\qquad \text{if abs (f c) < tol} \\
\qquad \quad \text{then } c \\
\qquad \text{else if signum (f a) == signum (f c)} \\
\qquad \quad \text{then} \\
\qquad \qquad \text{bisect } f \text{ tol c b} \\
\qquad \text{else} \\
\qquad \qquad \text{bisect } f \text{ tol a c} \\
\]

A bisections stream

\[
\text{bisections } f \text{ a b } = \\
\quad \text{let } c = (a + b) / 2 \\
\qquad y = f c \\
\quad \text{in} \\
\qquad \text{if signum (f a) == signum y} \\
\qquad \quad \text{then} \\
\qquad \qquad c : \text{bisections } f \text{ c b} \\
\qquad \text{else} \\
\qquad \qquad c : \text{bisections } f \text{ a c} \\
\]

Code demo!
David’s fun picture