CS3: Introduction to Symbolic Programming

Lecture 10: Tic-tac-toe, tree recursion

Spring 2006

Nate Titterton
nate@berkeley.edu
<table>
<thead>
<tr>
<th>Week</th>
<th>Date</th>
<th>Lecture/Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Mar 13-17</td>
<td>Introduction to Higher Order Procedures&lt;br&gt;Reading: SS 7-9; &quot;DbD&quot; part III</td>
</tr>
<tr>
<td>10</td>
<td>Mar 20-24</td>
<td>More HOF, Tic-Tac-Toe, Tree Recursion&lt;br&gt;Reading: SS 10, 15; &quot;Change Making&quot; case study</td>
</tr>
<tr>
<td>11</td>
<td>Mar 27-31</td>
<td><em>(Spring Break)</em></td>
</tr>
<tr>
<td>12</td>
<td>Apr 3-7</td>
<td>Lecture: Review&lt;br&gt;Lab: Miniproject #3</td>
</tr>
<tr>
<td>13</td>
<td>Apr 10-14</td>
<td>Lecture: MIDTERM #2&lt;br&gt;Lab: Start on &quot;Lists&quot;</td>
</tr>
</tbody>
</table>
Announcements

• Mid-semester survey FOR REAL this week
  - You need to do this

• Reading this week:
  - Simply Scheme Chapters 10 (Tue), 15 (Thur)
  - Change Making case study (Thur)
Tic Tac Toe
The board

x |   |  
-----+---+---
O | O | x  
-----+---+---
   |   |   

"X _ _"

"O O X"  "X__o o x__"  

"_ _ _"
Triples (another representation of a board)

"X_ _ O O X_ _ _"

(  x23  oox  789  xo7  2o8  3x9  xo9  3o7  )
procedures and lambda
In Scheme, procedures are *first-class* objects

- You can assign them a name
- You can pass them as arguments to procedures
- You can return them as the result of procedures
- You can include them in data structures

1. Well, you don't know how to do all of these yet.

3. What else in scheme is a *first-class* object?
The "hard" one is #3: returning procedures

;; this returns a procedure
(define (make-add-to number)
  (lambda (x) (+ number x)))

;; this also returns a procedure
(define add-to-5 (make-add-to 5))

;; hey, where is the 5 kept!?
(add-to-5 8) → 13

((make-add-to 3) 20) → 23
"lambda" is a special form that returns a function:

\[
\text{(lambda (arg1 arg2 ...)
    \text{statements})}
\]

\[
\text{(lambda (x) (* x x))}
\]

a procedure that takes one argument and multiplies it by itself
• These are the same:

```
(define (square x)
 (* x x))
```

```
(define square
 (lambda (x)
 (* x x)))
```
Can a \texttt{lambda}-defined function be recursive?

\begin{verbatim}
(lambda (sent)
  (if (empty? sent)
    '()
    (se (square (first sent))
      (???? (bf sent)))))
\end{verbatim}
When do you NEED lambda?

1. When you need the context (inside a two-parameter procedure)

   (add-suffix '-is-great '(nate sam mary))
   \[\Rightarrow\] (nate-is-great sam-is-great mary-is-great)

3. When you need to make a function on the fly
Review

Higher order procedures
Higher order function (HOFs)

• A HOF is a procedure that takes a procedure as an argument.

• There are three main ones that work with words and sentences:
  
  - *every* – do something to each element
  - *keep* – return only certain elements
  - *accumulate* – combine the elements
A definition of every

```
(define (my-every proc ws)
  (if (empty? ws)
      '()
      (se (proc (first ws))
           (my-every (bf ws))
                 )))
```

Every does a lot of work for you:

• Checking the conditional
• Returning the proper base case
• Combing the various recursive steps
• Invoking itself recursively on a smaller problem
Which HOFs would you use to write these?

1)  capitalize-proper-names
    (c-p-n '(mr. smith goes to washington)) ➞ (mr. Smith goes to Washington)

3)  count-if
    (count-if odd? '(1 2 3 4 5)) ➞ 3

5)  longest-word
    (longest-word '(I had fun on spring break)) ➞ spring

7)  count-vowels-in-each
    (c-e-l '(I have forgotten everything)) ➞ (1 2 3 3)

9)  squares-greater-than-100
    (s-g-t-100 '(2 9 13 16 9 45)) ➞ (169 256 2025)

11) root of the sum-of-squares
    (sos '(1 2 3 4 5 6 7)) ➞ 30

13) successive-concatenation
    (sc '(a b c d e)) ➞ (a ab abc abcd abcde)
Write successive-concatenation

\[(\text{sc } '(a \ b \ c \ d \ e)) \Rightarrow (a \ ab \ abc \ abcd \ abcde)\]

\[(\text{sc } '(\text{the big red barn})) \Rightarrow (\text{the thebig thebigred thebigredbarn})\]

(define (sc sent)
  (accumulate
    (lambda ??
      )
    sent))
Tree recursion
### Advanced recursion

#### Pascal’s Triangle

- How many ways can you choose $C$ things from $R$ choices?
- Coefficients of the $(x+y)^R$: look in row $R$
- etc.

<table>
<thead>
<tr>
<th>Rows (R)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>1</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>5</td>
<td>10</td>
<td>10</td>
<td>5</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
(define (pascal C R)
  (cond
    ((= C 0) 1) ;base case
    ((= C R) 1) ;base case
    (else ;tree recurse
      (+ (pascal C (- R 1))
        (pascal (- C 1) (- R 1)))))
)