CS3: Introduction to Symbolic Programming

Lecture 11: Tree-recursion
Midterm #2 review

Fall 2006  Nate Titterton
nate@berkeley.edu
<table>
<thead>
<tr>
<th>Week</th>
<th>Dates</th>
<th>Lecture:</th>
<th>Lab:</th>
<th>Reading:</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Nov 6-10</td>
<td>Tree-recursion, Review, Exam problems</td>
<td>Tree recursions, Miniproject #3</td>
<td>&quot;Change Making&quot; case study</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Simply Scheme, Ch. 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Nov 13-17</td>
<td>Lecture: <em>Midterm #2</em></td>
<td>Lab: Start on &quot;Lists&quot;</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Nov 20-24</td>
<td>Lecture: Lists, and introduce the big project</td>
<td>Lab: Lists; start on the project</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Nov 27–Dec 1</td>
<td>Lecture: Lists, and ?</td>
<td>Lab: Work on the project</td>
<td></td>
</tr>
</tbody>
</table>
Announcements

• Midterm 2 is coming…
  - Next week, 80 minutes (4:10-5:30).
  - Room 4 Leconte
  - Open book, open notes, etc.
  - Check for practice exams and solution on the course portal and in the reader.

• Midterm 2 review session
  - This Sunday, Nov 12, 2-4
  - 430 Soda (as last time)
What does midterm #2 cover?

- Advanced recursion (accumulating, multiple arguments, etc.). Including tree-recursion
- All of higher order functions
- Those "big" homeworks (bowling, compress, and occurs-in)
- Elections miniproject (!)
- Reading and programs:
  - Change making,
  - Difference between dates #3 (HOF),
  - tic-tac-toe
- SS chapters 14, 15, 7, 8, 9, 10
- Everything before the first Midterm (although, this won't be the focus of a question)
Programming Style and Grading

• During grading, we are going to start becoming “more strict” on style issues
  - Starting with miniproject #3
  - For the big project, style is important

• Why?
  - Program maintenance: 6 months later, will you know what your code does?
  - Code “literacy”: sharing code
What issues of style matter?

• Keep procedures small!
• Good names for procedures and parameters
• Adequate comments
  - Above and *within* procedures
• Put tests cases in a comment block
• Indent to aid program comprehension
• Proper use of global variables
• Avoid nesting conditional statements
• Data abstraction
Tree 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></th>
<th>columns (C)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<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)))))
)
"I have some bags of chips and some drinks. How many different ways can I finish all of these snacks if I eat one at a time?

(snack 1 2) $\rightarrow$ 3
- This includes (chip, drink, drink), (drink, chip, drink), and (drink, drink, chip).

(snack 2 2) $\rightarrow$ 6
- (c c d d), (c d c d), (c d d c)
  (d c c d), (d c d c), (d d c c)
Midterm like Problems...
make-bookends (a small problem)

• Write make-bookends, which is used this way:

```lisp
((make-bookends 'o) 'hi)  \rightarrow  ohio

((make-bookends 'to) 'ron)  \rightarrow  toronto

(define tom-proc (make-bookends 'tom))
(tom-proc "")  \rightarrow  tomtom
```
Write successive-concatenation

(sc '(a b c d e))
⇒ (a ab abc abcd abcd e)

(sc '(the big red barn))
⇒ (the thebig thebigred thebigredbarn)

(define (sc sent)
  (accumulate
   (lambda ??
     )
   sent))
make-decreasing

• **make-decreasing**
  - Takes a sentence of numbers
  - Returns a sentence of numbers, having removed elements of the input that were not larger than all numbers to the right of them.

\[(\text{make-decreasing '}(9\ 6\ 7\ 4\ 6\ 2\ 3\ 1))\]\n  \[\Rightarrow\ (9\ 7\ 6\ 3\ 1)\]

\[(\text{make-decreasing '}(3))\]\n  \[\Rightarrow\ (3)\]

Write first as a recursion, then as a HOF
Lecture 11:
Tree-recursion
Midterm #2 review
### Schedule

<table>
<thead>
<tr>
<th>Week</th>
<th>Dates</th>
<th>Lecture Topic</th>
<th>Lab Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Nov 6-10</td>
<td>Tree-recursion, Review, Exam problems</td>
<td>Lab: Tree recursions, Miniproject #3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reading: &quot;Change Making&quot; case study</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Simply Scheme, Ch. 15</td>
</tr>
<tr>
<td>12</td>
<td>Nov 13-17</td>
<td>Lecture: Midterm #2</td>
<td>Lab: Start on &quot;Lists&quot;</td>
</tr>
<tr>
<td>13</td>
<td>Nov 20-24</td>
<td>Lecture: Lists, and introduce the big project</td>
<td>Lab: Lists; start on the project</td>
</tr>
<tr>
<td>14</td>
<td>Nov 27–Dec 1</td>
<td>Lecture: Lists, and ?</td>
<td>Lab: Work on the project</td>
</tr>
</tbody>
</table>
Announcements

• Midterm 2 is coming…
  - Next week, 80 minutes (4:10-5:30).
  - Room 4 Leconte
  - Open book, open notes, etc.
  - Check for practice exams and solution on the course portal and in the reader.

• Midterm 2 review session
  - This Sunday, Nov 12, 2-4
  - 430 Soda (as last time)
What does midterm #2 cover?

- Advanced recursion (accumulating, multiple arguments, etc.). Including tree-recursion
- All of higher order functions
- Those "big" homeworks (bowling, compress, and occurs-in)
- Elections miniproject (!)
- Reading and programs:
  - Change making,
  - Difference between dates #3 (HOF),
  - tic-tac-toe
- SS chapters 14, 15, 7, 8, 9, 10
- Everything before the first Midterm (although, this won’t be the focus of a question)
Programming Style and Grading

• During grading, we are going to start becoming “more strict” on style issues
  - Starting with miniproject #3
  - For the big project, style is important

• Why?
  - Program maintenance: 6 months later, will you know what your code does?
  - Code “literacy”: sharing code
What issues of style matter?

• Keep procedures small!
• Good names for procedures and parameters
• Adequate comments
  - Above and within procedures
• Put tests cases in a comment block
• Indent to aid program comprehension

• Proper use of global variables
• Avoid nesting conditional statements
• Data abstraction
Tree recursion

Click to add text
### Advanced recursion

<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>

**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.

Fall 2006 CS3: 8
(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)))
    )))

\[
\begin{align*}
> (\text{pascal } 2 \ 5) \\
(\text{pascal } 2 \ 5) \\
+ \\
(\text{pascal } 2 \ 4) \\
+ \\
(\text{pascal } 2 \ 3) \\
\Rightarrow 1 \\
+ \\
(\text{pascal } 1 \ 3) \\
\Rightarrow 1 \\
+ \\
(\text{pascal } 0 \ 2) \\
\Rightarrow 1
\end{align*}
\]
Chips and Drinks

"I have some bags of chips and some drinks. How many different ways can I finish all of these snacks if I eat one at a time?

(snack 1 2) \rightarrow 3
  - This includes (chip, drink, drink), (drink, chip, drink), and (drink, drink, chip).

(snack 2 2) \rightarrow 6
  - (c c d d), (c d c d), (c d d c)
    (d c c d), (d c d c), (d d c c)

;;; snack

(define (snack chips drinks)
  (cond ((and (= 0 chips) (= 0 drinks))
        ;; both are 0, no more ways...
        0)
        ((or (= 0 chips) (= 0 drinks))
        ;; one is zero, one isn't, one remaining way
        1)
        (else (+ (snack (- chips 1) drinks)
                  (snack chips (- drinks 1))))))
Midterm like Problems...

Click to add text
**make-bookends (a small problem)**

- **Write** make-bookends, which is used this way:

  \[
  (\text{make-bookends }'o') 'hi) \Rightarrow \text{ohio}
  \]

  \[
  (\text{make-bookends }'to') 'ron) \Rightarrow \text{toronto}
  \]

  \[
  (\text{define tom-proc (make-bookends }'tom))
  (\text{tom-proc }"") \Rightarrow \text{tomtom}
  \]

  \[
  (\text{define (make-bookends wd})
  \quad (\text{lambda (inner-wd) (word wd inner-wd wd)}))
  \]
Write successive-concatenation

(sc 'a b c d e))  ➞  (a ab abc abcd abcde)
(sc '(the big red barn))  ➞  (the thebig thebigred thebigredbarn)

(define (sc sent)
  (accumulate
    (lambda ??
      sent)))

That inner lambda is tricky. Remember, the accumulate needs to return a sentence, so the right argument may be a word (the first time it is called) or a sentence (every other time):

(lambda (wd so-far)
  (if (word? so-far)
      (se wd (word wd so-far))  ;; initial invocation
      (se wd                      ;; other invocations
        ;; prepend-each
        (every
          (lambda (so-far-element)
            (word wd sent-so-far-element))
          so-far)))
  )

That every inside also requires a lambda, because the function needs to have one argument, but also use the value of wd.
make-decreasing

- make-decreasing
  - Takes a sentence of numbers
  - Returns a sentence of numbers, having removed elements of the input that were not larger than all numbers to the right of them.

(make-decreasing '(9 6 7 4 6 2 3 1)) ➞ (9 7 6 3 1)
(make-decreasing '(3)) ➞ (3)

Write first as a recursion, then as a HOF

;; recursion -- right to left
(define (make-decreasing sent)
  (cond ((empty? sent) '())
        ((empty? (butfirst sent))
         (first sent))
        ((> (last (butlast sent))
            (last sent))
         (se (make-decreasing (butlast sent))
             (last sent)))
        (else
         (se (make-decreasing (butlast (butlast sent)))
             (last sent))))
)

;; recursion -- left to right
(define (make-decreasing sent)
  (cond ((or (empty? sent)
             (empty? (bf sent)))
         sent)
        ((bigger-than-all? (first sent) (bf sent))
         (se (first sent)
             (make-decreasing (bf sent))))
        (else (make-decreasing (bf sent))))
)

(define (bigger-than-all? num sent)
  (cond ((empty? sent) #t)
        ((> num (first sent))
         (bigger-than-all? num (bf sent)))
        (else #f)))

;; HOF
(define (make-decreasing sent)
  (accumulate
   (lambda (num sent-so-far)
    ;; first time thru
    (if (word? sent-so-far) ;; first time thru
        (if (< sent-so-far num)
            (se num sent-so-far)
            (se sent-so-far)))
        (if (< (first sent-so-far) num)
            (se num sent-so-far)
            sent-so-far))))