CS3:
Introduction to Symbolic Programming

Lecture 11:
Tree-recursion
Midterm #2 review

Fall 2006

Nate Titterton
nate@berkeley.edu

Schedule

11	Nov 6-10	Tree-recursion, Review, Exam problems Lab: Tree recursions, Miniproject #3 Reading: "Change Making" case study Simply Scheme, Ch. 15
12	Nov 13-17	Lecture: *Midterm #2* Lab: Start on "Lists"
13	Nov 20-24	Lecture: Lists, and introduce the big project Lab: Lists; start on the project
14	Nov 27–Dec 1	Lecture: Lists, and ? Lab: Work on the project

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

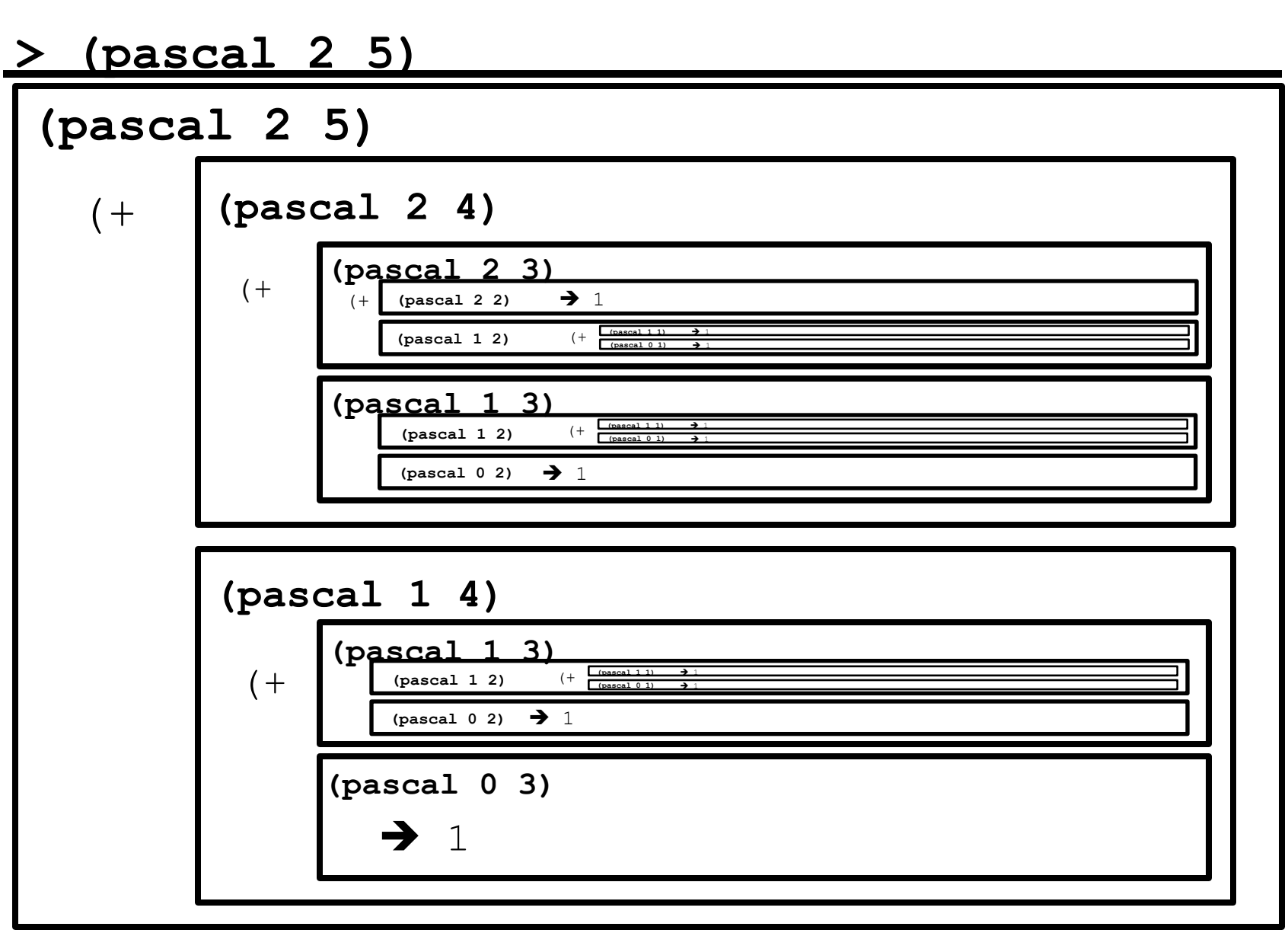
Advanced recursion

	columns (C)							
rows (R)		0	1	2	3	4	5	...
	0	1						...
	1	1	1					...
	2	1	2	1				...
	3	1	3	3	1			...
	4	1	4	6	4	1		...
	5	1	5	10	10	5	1	...
	...	...	...	...	...	...	...	...

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.

```
(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))
     )))
```


> (pascal 2 5)
(pascal 2 5)
(+
(pascal 2 4)
(+
(pascal 2 3)
(+
(pascal 2 2)
➔ 1
(pascal 1 2)
(+
(pascal 1 1)
➔ 1
(pascal 0 1)
➔ 1
(pascal 1 3)
(pascal 1 2)
(+
(pascal 1 1)
➔ 1
(pascal 0 1)
➔ 1
(pascal 0 2)
➔ 1
(pascal 1 4)
(+
(pascal 1 3)
(pascal 1 2)
(+
(pascal 1 1)
➔ 1
(pascal 0 1)
➔ 1
(pascal 0 2)
➔ 1
(pascal 0 3)
➔ 1

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)` → `3`

- This includes (chip, drink, drink), (drink, chip, drink), and (drink, drink, chip).

`(snack 2 2)` → `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:**

```
((make-bookends 'o) 'hi) ➔ ohio

((make-bookends 'to) 'ron) ➔ toronto

(define tom-proc (make-bookends 'tom))
(tom-proc "") ➔ tomtom
```

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))
```

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

CS3:
Introduction to Symbolic Programming

Lecture 11:
Tree-recursion
Midterm #2 review

Fall 2006

Nate Titterton
nate@berkeley.edu

Schedule

11	Nov 6-10	Tree-recursion, Review, Exam problems Lab: Tree recursions, Miniproject #3 Reading: "Change Making" case study Simply Scheme, Ch. 15
12	Nov 13-17	Lecture: *Midterm #2* Lab: Start on "Lists"
13	Nov 20-24	Lecture: Lists, and introduce the big project Lab: Lists; start on the project
14	Nov 27–Dec 1	Lecture: Lists, and ? Lab: Work on the project

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

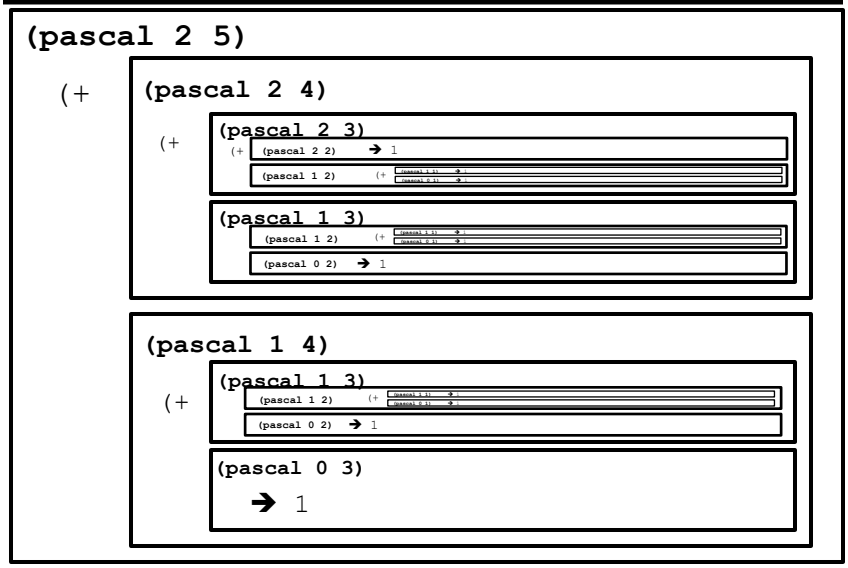
	columns (C)							
rows (R)		0	1	2	3	4	5	...
	0	1						...
	1	1	1					...
	2	1	2	1				...
	3	1	3	3	1			...
	4	1	4	6	4	1		...
	5	1	5	10	10	5	1	...
	...	...	...	...	...	...	...	...

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.

```
(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))
     )))
```

> (pascal 2 5)


(pascal 2 5)
(+
(pascal 2 4)
(+
(pascal 2 3)
(+ (pascal 2 2) → 1
(pascal 1 2)
(pascal 1 3)
(pascal 1 2)
(pascal 0 2) → 1
(pascal 1 4)
(+
(pascal 1 3)
(pascal 1 2)
(pascal 0 2) → 1
(pascal 0 3)
→ 1

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) → 3

- **This includes (chip, drink, drink), (drink, chip, drink), and (drink, drink, chip).**

(snack 2 2) → 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:**

```
((make-bookends 'o) 'hi) ➔ ohio

((make-bookends 'to) 'ron) ➔ toronto

(define tom-proc (make-bookends 'tom))
(tom-proc "") ➔ tomtom
```



```
(define (make-bookends wd)
   (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)
          (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)))
       sent))
```