CS3:
Introduction to Symbolic Programming

Lecture 10:
Tic-tac-toe, tree recursion

Spring 2006

Nate Titterton
nate@berkeley.edu

Schedule

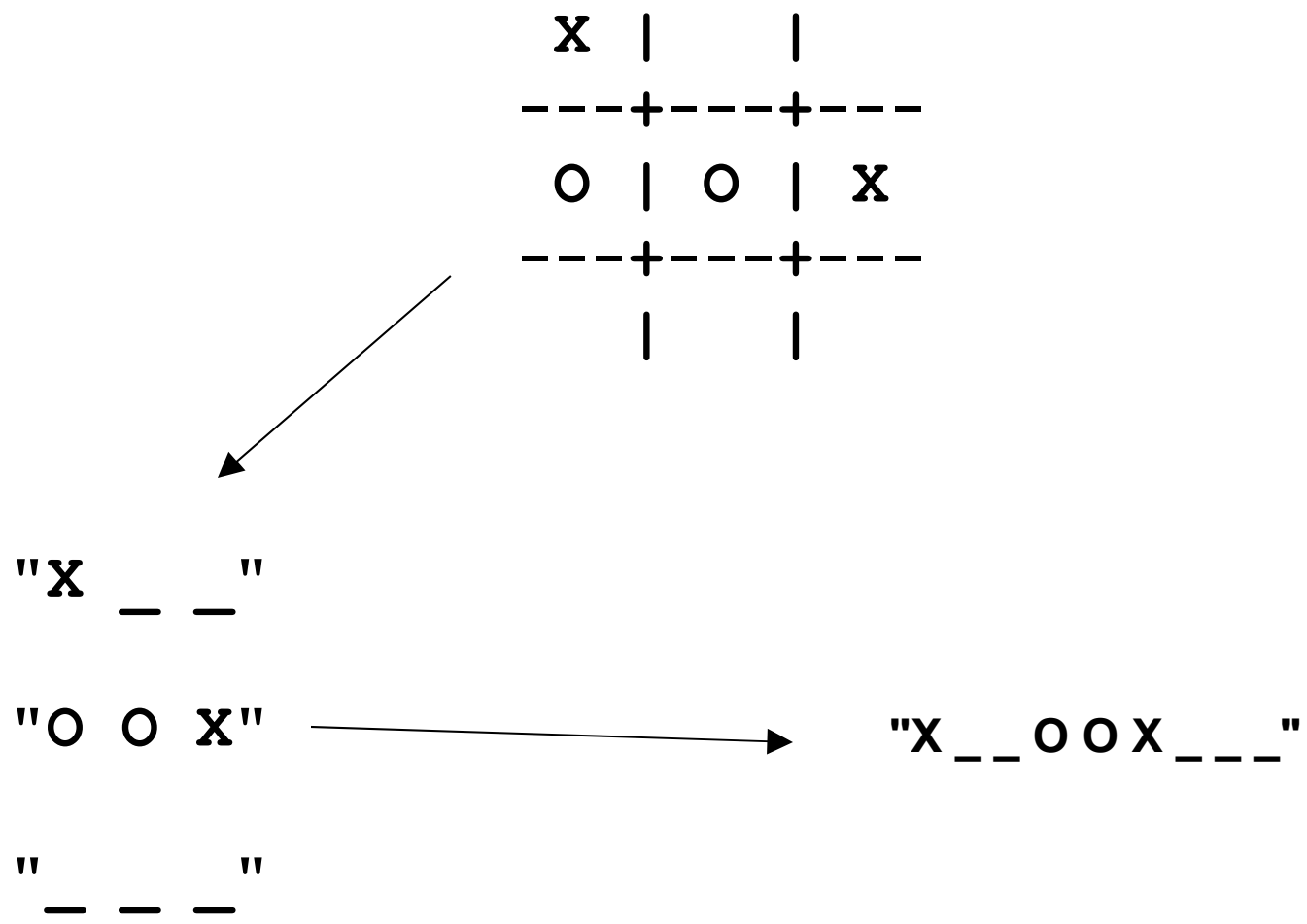
9	Mar 13-17	Introduction to Higher Order Procedures Reading: SS 7-9; "DbD" part III
10	Mar 20-24	More HOF, Tic-Tac-Toe, Tree Recursion Reading: SS 10, 15; "Change Making" case study
11	Mar 27-31	*(Spring Break)*
12	Apr 3-7	Lecture: Review Lab: Miniproject #3
13	Apr 10-14	Lecture: MIDTERM #2 Lab: Start on "Lists"

Announcements

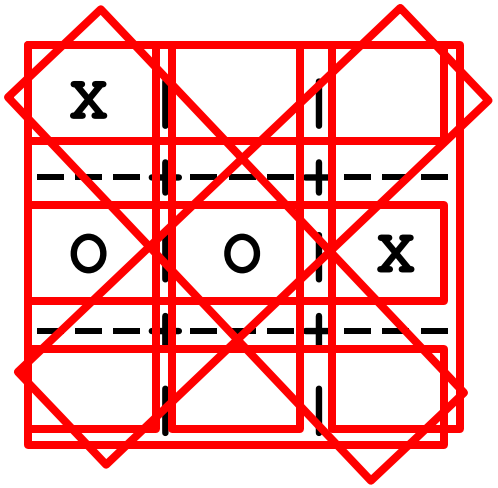
- **Mid-semester survey FOR REAL this week**
  - **You need to do this**
- **Reading this week:**
  - **Simply Scheme Chapters 10 (Tue),15 (Thur)**
  - **Change Making case study (Thur)**

Tic Tac Toe

The board



Triples (another representation of a board)



"X__OOX___"

(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
```

the `lambda` form

- **"`lambda`" is a special form that returns a function:**

```
(lambda (arg1 arg2 …)
   statements
      )
```

```
(lambda          (x)          (*     x     x))
```

⇨ ⇨ ⇨ ⇨ ⇨

a procedure | that takes one argument | and multiplies | it | by itself

Using `lambda` with `define`

- **These are the same:**

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

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

Can a `lambda`-defined function be recursive?

```
(lambda (sent)
  (if (empty? sent)
      '()
      (se (square (first sent))
          (???? (bf sent)))))
```

When do you NEED lambda?

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

```
(add-suffix '-is-great '(nate sam mary))
    ➔ (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

```
(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))
```

Tree recursion

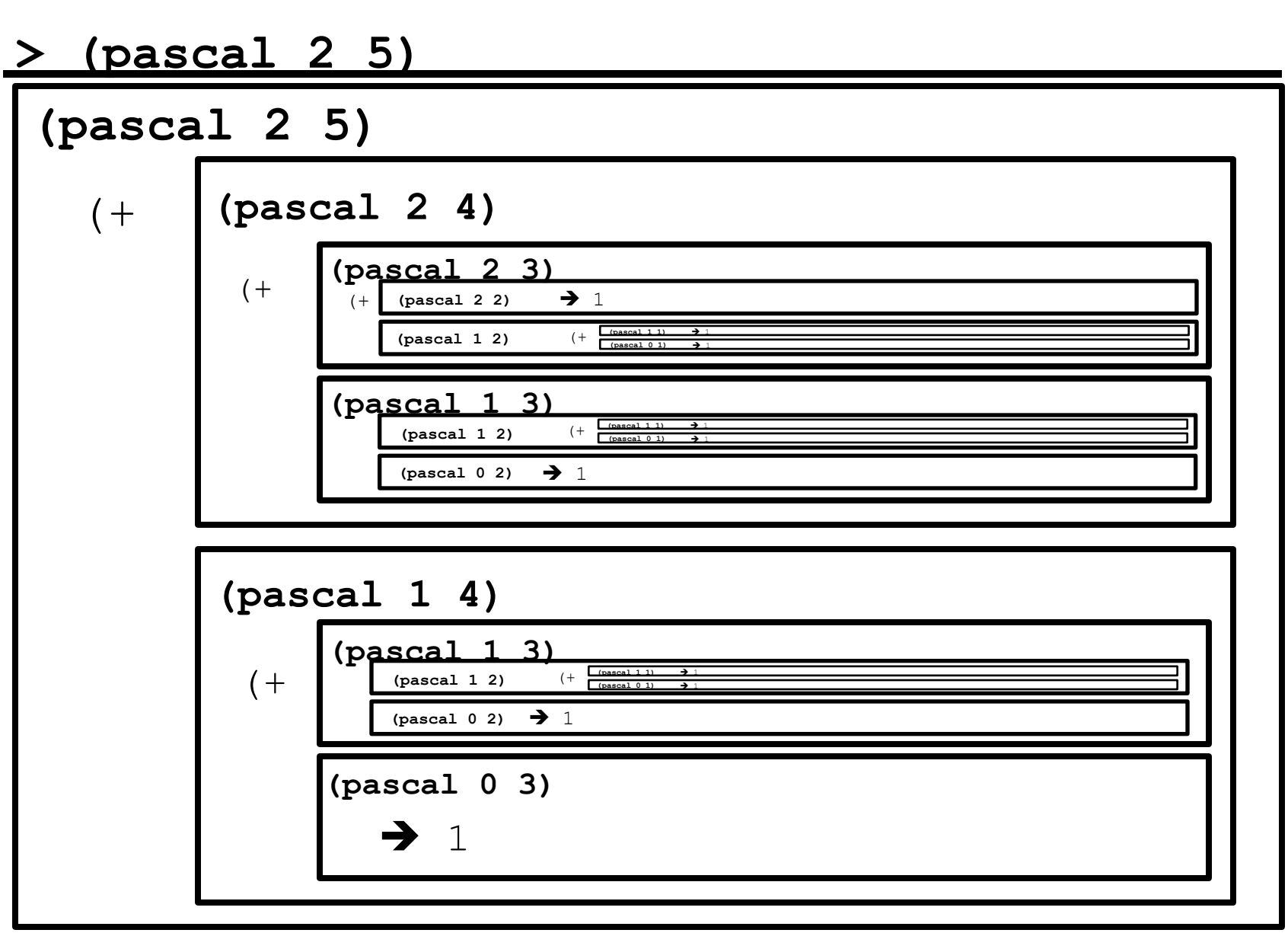
Advanced recursion

	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