List Recursion Examples & Recursive Procedures

One-Slide Summary

- Recursive functions that operate on lists have a similar structure. **list-cruncher** is a higher-order function that can be used to implement many others.
- Decisions in a function can be abstracted out by adding a function argument. For example, **find-closest-number** is just **find-closest** plus a function defining what a **close-number** is.
- The Fibonacci numbers are a **recursively-defined sequence**.
- Almost all music uses a **stack** structure: starts on the tonic, repeats similar patterns in a structured way, ends on the tonic.

Outline

- Your Comments
- list-cruncher
- find-closest-number
  - Reminder: procedure definition strategy!
- find-closest
- Fibonacci numbers
- Recursive Transition Networks
  - vs. Backus-Naur Form Grammars
- Musical Harmony

Anonymous Course Feedback

- Too Fast v. Too Slow?
  - No CS experience? Jargon in “base lecture”?
- “I really do appreciate that he tries to read people's facial expressions and ensure that we understand before we move on.” vs. “The hand-raising is too frequent.”
- “I wish the TAs would get around to more people in lab.” vs. “I asked Paul a question about why one of my procedures wasn't working, and thoroughly explained why, and after he was done asked if his explanation made sense to make sure that I understood everything.”
- “Wes does try to involve everyone, but it seems like students are punished for wanting to participate more than once.” vs. “I think there are too many questions directed towards the class.”

Similarities and Differences

```
(define (map f p)                (define (sumlist p)
  (if (null? p)                    (if (null? p)
      null                             0
      (cons (f (car p))                (+ (car p)
            (map f (cdr p)))))            (sumlist (cdr p)))))

   (define (list-cruncher lst)
     (if (null? lst)
         base result
         (combiner (car lst)
                   (recursive-call ... (cdr lst)))))
```

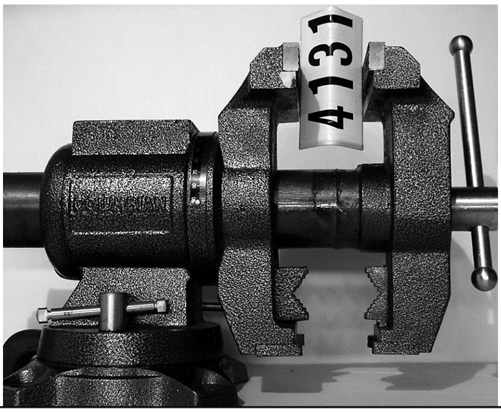
Similarities and Differences

```
(define (map f p)                (define (sumlist p)
  (if (null? p)                    (if (null? p)
      null                             0
      (cons (f (car p))                (+ (car p)
            (map f (cdr p)))))            (sumlist (cdr p)))))

   (define (list-cruncher ? ... ? lst)
     (if (null? lst)
         base result
         (combiner (car lst)
                   (recursive-call ... (cdr lst)))))
```

How could this work?

- I want to crunch all lists. How would I get started?



One Ring To Rule Them All?

```
(define (list-cruncher base proc combiner lst)
 (if (null? lst)
    base
    (combiner (proc (car lst))
              (list-cruncher base proc combiner 
                       (cdr lst)))))

(define (sumlist p)
 (list-cruncher 0 (lambda (x) x) + p))

(define (map f p)
 (list-cruncher null f cons p))
```

Crunchy Center

```
(define (list-cruncher base proc combiner lst)
 (if (null? lst)
    base
    (combiner (proc (car lst))
              (list-cruncher base proc combiner 
                       (cdr lst)))))
```

- How would you define **length** using list-cruncher?

```
(define (length lst)
  (if (null? lst) 0
     (+ 1 (length (cdr lst)))))
```

list-cruncher crunches length

```
(define (list-cruncher base proc combiner lst)
 (if (null? lst)
    base
    (combiner (proc (car lst))
              (list-cruncher base proc combiner
                       (cdr lst)))))
(define (length p)
  (if (null? p) 0
     (+ 1 (length (cdr p)))))

(define (length p)
  (list-cruncher 0 (lambda (x) 1) + p))
```

Crunchy Center 2

```
(define (list-cruncher base proc combiner lst)
 (if (null? lst)
    base
    (combiner (proc (car lst))
              (list-cruncher base proc combiner 
                       (cdr lst)))))
```

- How would you define **filter** using list-cruncher?

```
(define (filter predicate lst)
  (if (null? lst) null
     (append
       (if (predicate (car lst)) (list (car lst)) null)
       (filter predicate (cdr lst)))))
```

list-cruncher crunches filters

```
(define (list-cruncher base proc combiner lst)
 (if (null? lst)
    base
    (combiner (proc (car lst))
              (list-cruncher base proc combiner 
                       (cdr lst)))))
 (define (filter predicate lst)
      (if (null? lst) null
         (append
           (if (predicate (car lst)) (list (car lst)) null)
           (filter predicate (cdr lst)))))
(define (filter pred lst)
 (list-cruncher null
  (lambda (carlst) (if (pred carlst) (list carlst) null))
  append lst))
```

Liberal Arts Trivia: Drama

- In this 1948 play by Samuel Beckett has been called “the most significant English-language play of the 20th century”. The minimal setting calls to mind “the idea of the ‘lieu vague’, a location which should not be particularised”, and the play features two characters who never meet the title character.

Liberal Arts Trivia: History

- At the height of its power, in the 16th and 17th century, this political organization spanned three continents. It controlled much of Southeastern Europe, the Middle East and North Africa, and contained 29 provinces and multiple vassal states. Noted cultural achievements include architecture (vast inner spaces confined by seemingly weightless yet massive domes, harmony between inner and outer spaces, articulated light and shadow, etc.), classical music, and cuisine.

find-closest-number

- The function **find-closest-number** takes two arguments. The first is a single number called the goal. The second is a non-empty list of numbers. It returns the number in the input list that is closest to the goal number.

```
> (find-closest-number 150 (list 101 110 120 157 340 588))
157
> (find-closest-number 12 (list 4 11 23))
11
> (find-closest-number 12 (list 95))
95
```

Recall The Strategy!

Be optimistic!

Assume you can define:

(find-closest-number goal numbers)

that finds the closest number to goal from the list of numbers.

What if there is *one more number*?

Can you write a function that finds the closest number to match from new-number and numbers?

find-closest-number hint

One Approach for the Recursive Case:

You have two possible answers: the current car of the list and the result of the recursive call. Compare them both against the goal number, and return the one that is closer.

Optimistic Function

```
(define (find-closest goal numbers)
  ;; base case missing for now
  (if (< (abs (- goal (car numbers)))
         (abs (- goal
                  (find-closest-number
                   goal (cdr numbers)))))
      (car numbers)
      (find-closest-number goal (cdr numbers))))
```

Defining Recursive Procedures

2. Think of the simplest version of the problem (almost always null), something you can already solve. (**base case**)

Is null the base case for find-closest-number?

find-closest-number defined

```
(define (find-closest-number goal numbers)
  (if (= 1 (length numbers))        ;; base case
     (car numbers)
     (if (< (abs (- goal (first numbers)))
            (abs (- goal
                    (find-closest-number
                     goal (cdr numbers)))))
           (car numbers)
           (find-closest-number (cdr numbers))))
```

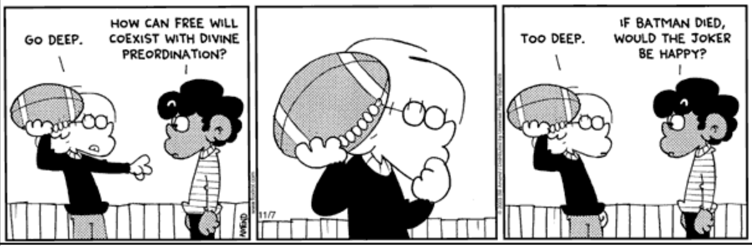
```
(define (find-closest-number goal numbers)
  (if (= 1 (length numbers))
      (car numbers)
      (if (< (abs (- goal (car numbers)))
             (abs (- goal
                     (find-closest-number goal (cdr numbers)))))
          (car numbers)
          (find-closest-number goal (cdr numbers)))))
> (find-closest-number 150
    (list 101 110 120 157 340 588))
157
> (find-closest-number 0 (list 1))
1
> (find-closest-number 0 (list ))
first: expects argument of type <non-empty list>; given ()
```

ReportManager

Error Retrieving values in VB App.
Either you didn't enter the data properly, or the developer for this screwed up royally. I'm leaning towards 'A'.

Generalizing find-closest-number

- How would we implement find-closest-number-without-going-over?
- What about find-closest-word?
- ...


GO DEEP.
HOW CAN FREE WILL COEXIST WITH DIVINE PREORDINATION?
TOO DEEP.
IF BATMAN DIED, WOULD THE JOKER BE HAPPY?
11/7

Generalizing find-closest-number

- How would we implement find-closest-number-without-going-over?
- What about find-closest-word?
- ...

The "closeness" metric should be a procedure parameter!

find-closest

```
(define (find-closest goal lst closeness)
  (if (= 1 (length lst))
     (car lst)
     (if (< (closeness goal (car lst))
            (closeness goal
              (find-closest goal (cdr lst) closeness)))
        (car lst)
        (find-closest goal (cdr lst) closeness)))
```

How can we implement find-closest-number using find-closest?

Using find-closest

```
(define (find-closest-number goal numbers)
  (find-closest goal numbers
                (lambda (a b) (abs (- a b)))))


(define (find-closest-below goal numbers)
  (find-closest goal numbers
                (lambda (a b)
                    (if (>= a b) (- a b) 99999))))
```

find-closest

```
(define (find-closest goal lst closeness)
  (if (= 1 (length lst))
    (car lst)
    (if (< (closeness goal (car lst))
           (closeness goal
             (find-closest goal (cdr lst) closeness)))
      (car lst)
      (find-closest goal (cdr lst) closeness))))
```

How can we avoid evaluating **find-closest** twice?

find-closest

```
(define (find-closest goal lst closeness)
  (if (= 1 (length lst))
    (car lst)
    (pick-closest closeness goal (car lst)
                  (find-closest goal (cdr lst) closeness))))

(define (pick-closest closeness goal num1 num2)
  (if (< (closeness goal num1)
         (closeness goal num2))
     num1
     num2))
```

Seen Anything Like This?

```
(define (find-best-match sample tiles color-comparator)
  (if (= (length tiles) 1)
    (car tiles)
    (pick-better-match
      sample
      (car tiles)
      (find-best-match
        sample
        (cdr tiles)
        color-comparator)
      color-comparator))))
```

```
(define (pick-better-match
           sample tile1 tile2
           color-comparator)
  (if (color-comparator sample
        (tile-color tile1) (tile-color tile2))
      tile1
      tile2))
```

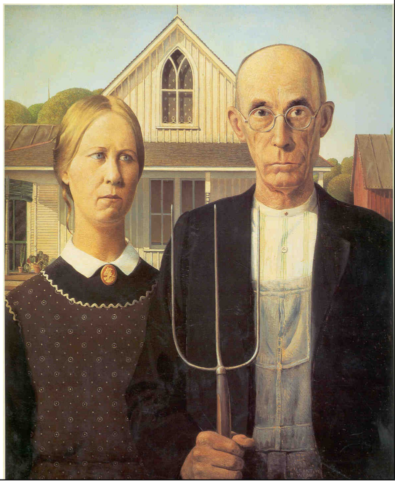
find-best-match from PA1 (Photomosaics) is just **find-closest** !
pick-better-match is just **pick-closest** ! You could write all of PA1.

Liberal Arts Trivia: Philosophy

- This branch of philosophy, which Aristotle called “First Philosophy”, investigates principles of reality transcending those of any particular science. It is concerned with explaining the ultimate nature of being and the world (e.g., determinism and free will, mind and matter, space and time). Its modern name comes from the fact that Aristotle's chapters about it were placed “beyond” his chapters on matter and force.

Liberal Arts Trivia: Painting

- Name this 1930 oil-on-beaverboard painting by Grant Wood. It is one of the most familiar images of 20th century American art and has achieved an iconic status.



GEB Chapter V

You could spend the rest of your life just studying things in this chapter (25 pages)!

- Music Harmony
- Stacks and Recursion
- Theology
- Language Structure
- Number Sequences
- Chaos
- Fractals (PS3 out today. Start early. Why?)
- Quantum Electrodynamics (later lecture)
- DNA (later lecture)
- Sameness-in-differentness
- Game-playing algorithms (later lecture)

Fibonacci's Problem

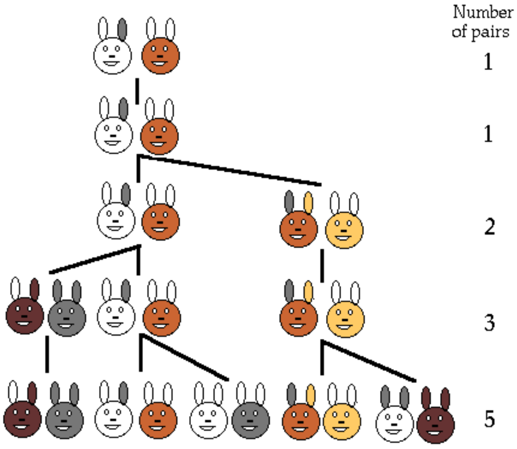
Filius Bonacci, 1202 in Pisa:

Suppose a newly-born pair of rabbits, one male, one female, are put in a field. Rabbits mate at the age of one month so that at the end of its second month a female can produce another pair of rabbits.

Suppose that our rabbits **never die** and that the female **always** produces one new pair (one male, one female) **every month** from the second month on.

How many pairs will there be in one year?

Rabbits



From http://www.mcs.surrey.ac.uk/Personal/R.Knott/Fibonacci/fibnat.html

Fibonacci Numbers

GEB p. 136:

These numbers are best defined **recursively** by the pair of formulas

FIBO (*n*) = FIBO (n – 1) + FIBO (n – 2)

for n > 2

FIBO (1) = FIBO (2) = 1

for n <= 2

Can we turn this into a Scheme procedure?

Defining FIBO

1. Be optimistic - assume you can solve it, if you could, how would you solve a bigger problem.
2. Think of the simplest version of the problem, something you can already solve.
3. Combine them to solve the problem.

These numbers are best defined recursively by the pair of formulas
FIBO (*n*) =
FIBO (n – 1)
+ FIBO (n – 2)
for n > 2
FIBO (1) = FIBO (2) = 1

Defining fibo

```
;;; (fibo n) evaluates to the nth Fibonacci
;;; number
(define (fibo n)
  (if (or (= n 1) (= n 2))
      1 ;;; base case
      (+ (fibo (- n 1))
         (fibo (- n 2)))))
```

FIBO (1) = FIBO (2) = 1

FIBO (*n*) =
FIBO (n – 1)
+ FIBO (n – 2)
for n > 2

Fibo Results

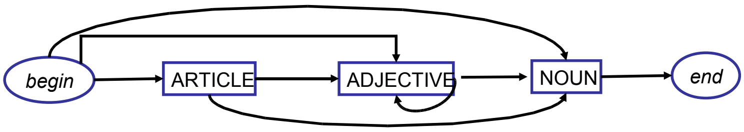
> (fibo 2)
1
> (fibo 3)
2
> (fibo 4)
3
> (fibo 10)
55
> (fibo 60)
Still working after 4 hours…

Why can't our 4Mx Apollo Guidance Computer figure out how many rabbits there will be in 5 years?

To be continued...

Recursive Transition Networks

ORNATE NOUN



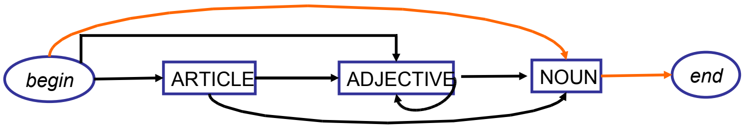
Can we describe this using Backus Naur Form?

Turn Off Yahoo! Toolbar Smart Tools?

□ Don't ask me again

Recursive Transition Networks

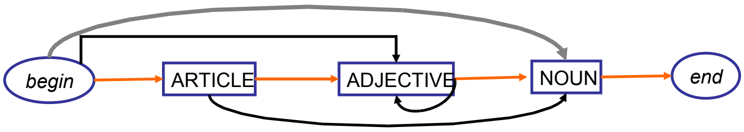
ORNATE NOUN



ORNATE NOUN ::= NOUN

Recursive Transition Networks

ORNATE NOUN

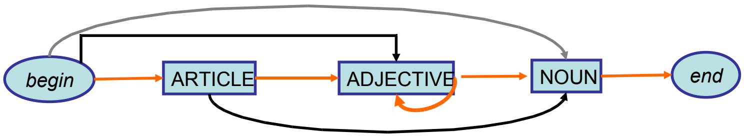


ORNATE NOUN ::= NOUN

ORNATE NOUN ::= ARTICLE ADJECTIVE NOUN

Recursive Transition Networks

ORNATE NOUN



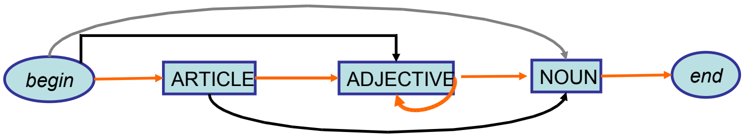
ORNATE NOUN ::= ARTICLE ADJECTIVE NOUN

ORNATE NOUN ::= ARTICLE ADJECTIVE ADJECTIVE NOUN

ORNATE NOUN ::= ARTICLE ADJECTIVE ADJECTIVE ADJECTIVE NOUN
ORNATE NOUN ::= ARTICLE ADJECTIVE ADJECTIVE ADJECTIVE ADJECTIVE NOUN
ORNATE NOUN ::= ARTICLE ADJECTIVE ADJECTIVE ADJECTIVE ADJECTIVE ADJECTIVE ADJECTIVE NOUN

Recursive Transition Networks

ORNATE NOUN



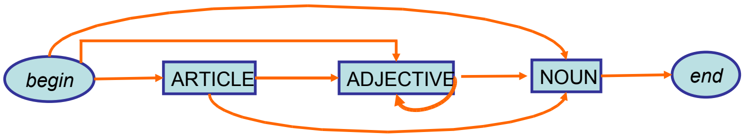
ORNATE NOUN ::= ARTICLE ADJECTIVES NOUN

ADJECTIVES ::= ADJECTIVE ADJECTIVES

ADJECTIVES ::=

Recursive Transition Networks

ORNATE NOUN


begin

ARTICLE

ADJECTIVE

NOUN

end

ORNATE NOUN ::= *OPTARTICLE ADJECTIVES NOUN*

ADJECTIVES ::= *ADJECTIVE ADJECTIVES*

ADJECTIVES ::= ε

OPTARTICLE ::= *ARTICLE*

OPTARTICLE ::= ε

Which notation is *better*?

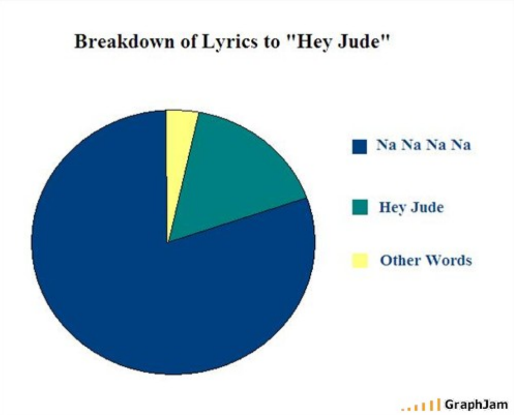
Music Harmony

Kleines Harmonisches Labyrinth

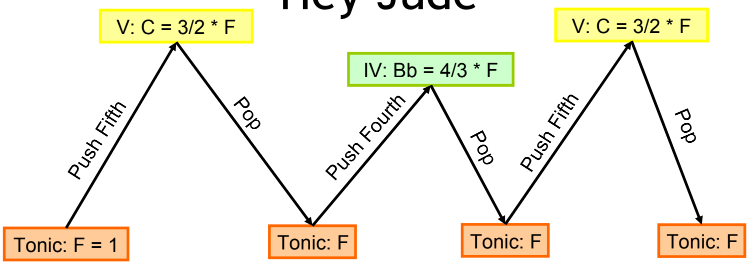
(Little Harmonic Labyrinth)

Hey Jude

John Lennon and Paul McCartney, 1968



Hey Jude


V: C = 3/2 * F

IV: Bb = 4/3 * F

V: C = 3/2 * F

Push Fifth

Pop

Push Fourth

Pop

Push Fifth

Pop

Tonic: F = 1

Tonic: F

Tonic: F

Tonic: F

Tonic: Hey Jude, don't make it

V: bad. take a sad song and make it

Tonic: better Re-

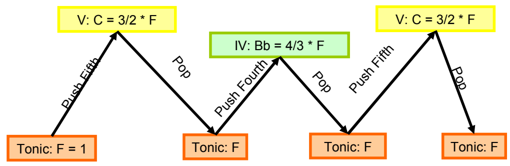
IV: member to let her into your

Tonic: heart, then you can

V: start to make it bet-

Tonic: -ter.

Verse ::=


V: C = 3/2 * F

IV: Bb = 4/3 * F

V: C = 3/2 * F

Push Fifth

Pop

Push Fourth

Pop

Push Fifth

Pop

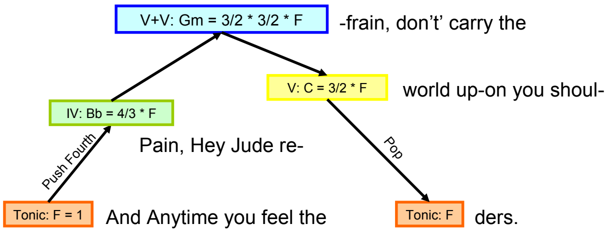
Tonic: F = 1

Tonic: F

Tonic: F

Tonic: F

Bridge ::=


V+V: Gm = 3/2 * 3/2 * F

-frain, don't' carry the

V: C = 3/2 * F

world up-on you shoul-

IV: Bb = 4/3 * F

Pain, Hey Jude re-

Push Fourth

Pop

Tonic: F = 1

And Anytime you feel the

Tonic: F

ders.

HeyJude ::= *Verse VBBN VBBN Verse Verse Better Coda*
VBBN ::= *Verse Bridge Bridge Nanana (ends on C)*
Coda ::= F Eb Bb F *Coda*

Music

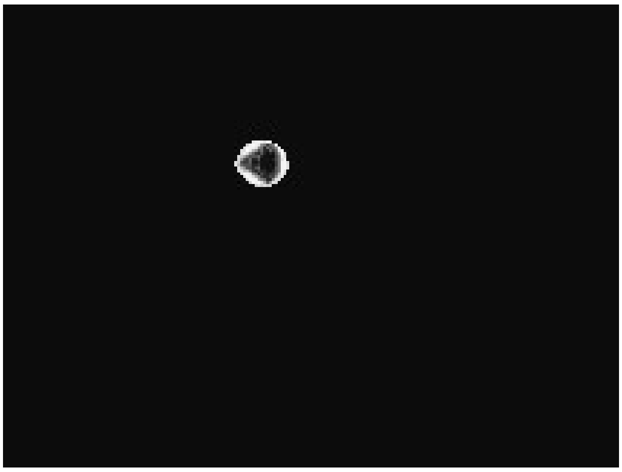
- Almost All Music Is Like This
  - Pushes and pops the listener's stack, but doesn't go too far away from it
  - Repeats similar patterns in structured way
  - Keeps coming back to Tonic, and Ends on the Tonic
- Any famous Beatles song that doesn't end on Tonic?

Charge

- **Challenge:** Try to find a "pop" song with a 3-level deep harmonic stack
- **PS3:** due in 10 days.

Be optimistic!

You know everything you need to finish it now, and it is longer than PS2, so get started now!



Beatles: "A Day in the Life" (starts on G, ends on E)

http://www.fractalwisdom.com/FractalWisdom/fractal.html

Homework

- Problem Set 2 Coding Due @ Midnight
- Start Problem Set 3 Now
  - No, really.
  - Due Wed Feb 11
- Read Course Book Chapter 6
  - By Monday Feb 9