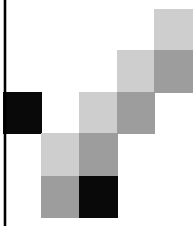


Picking and Choosing…

…Amb Above the Line

Intro to Nondeterminism

- So before we programmed to compute an answer to a problem.
- What if we set up a system where we give a program certain constraints?
- We then give that program a “solution space” and it does all the work for us. In a sense it finds all the solutions to the problem!

Intro to Nondeterminism

- This is Nondeterministic Programming
- There can be more than one solution to a problem
- So how can we do this by altering the MCE?
- We introduce the new procedures amb, require, & try-again

How does this work?

- Structure:
  - **Specify solution space:**
    Ex. (amb 1 2 3)
  - **Specify contraints:**
    Ex. (require (< n 10))
  - **Do something with the solution that satisfy the constraints.**
    Ex. (define (even? x) (= (remainder x 2) 0))
    (let ((a (amb 1 2 3 4 5)))
    (require (even? a))
    a)
  - **Find another answer:** try-again

What is AMB?

- Amb is a special form.
  - Why? Because it doesn't evaluate all of its arguments (doesn't follow the rules of evaluation)
  - It returns a single value or **fails** if there are no more values

What happens…

- What happens when a solution doesn't satisfy the requirements or there are no more values left?

  FAILURE!!!
- A failure is not the same as an error! It just means we need to go back and try another solution

Back to Amb

- Amb sequentially chooses its values from left to right.
- So you type:
  (amb 1 2 3) → 1
  try-again → 2
  try-again → 3
  try-again → no more values
- (amb) ← always fails

What do these print?

After multiple try-agains what would happen?

- (amb 1 2 3)
- (amb (list 1 2 3))
- (amb 1 (amb 2 (amb 3)))
- (amb (amb 1) (amb 2) (amb 3))
- (amb (amb 2 3) 1 (amb 4))

Try-Again

- try-again finds another solution to the "current problem"
- What happens if we start a new problem before the current one isn't finished?
  - try-again is now going to work with the new problem
- Try-again should be only used at the command prompt

Require

- Require lets you put constraints on your solution.
- Implementation:

```
(define (require p)
    (if (not p) (amb)))
```

- If we don't satisfy a requirement, we fail by calling (amb)

Practice!

What does this return after multiple try-agains?

```
(let ((a (amb 1 2 3))
      (b (amb -1 4 3)))
  (require (< a b))
  (list a b))
```

More Practice…

```
(define (foo x)
    (cond ((not (pair? x)) (amb))
          ((word? (cdr x)) (cdr x))
          (else (amb (foo (car x))
                     (foo (cdr x))))))
```

- (foo '(a (b c) (d e . f) (g (h . i) j) k))

  _____

- try-again

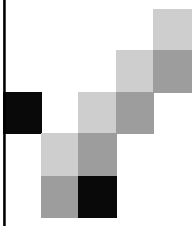
  _____

- try-again

  _____

More Practice…

- (define (foo x)
  (cond ((not (pair? x)) (amb))
  ((word? (cdr x)) (cdr x))
  (else (amb (foo (car x))
  (foo (cdr x))))))
- (foo '(a (b c) (d e . f) (g (h . i) j) k))
  f
- try-again
  i
- try-again
  no more values



Amb Below the Line

…the fun just never ends…

Intro to Amb Below…

- So we know that amb chooses its values from left to right, and if it doesn't have any choices left, it magically goes back and returns you no more values.
- So what the heck is actually happening?

Continuations: Success and Failure

- Continuations are a successful computation and a promise to compute in that environment until a failure is reached.
- When a failure is reached, a "roll-back" mechanism happens where it goes back to the preceding environment and continues from there.
- So a failure basically tries to find a new value.
- Yes it's confusing, but slowly understand…

Let's take a look…

```
(define (ambeval exp env succeed fail)
   (cond ((self-evaluating? exp)
            (succeed exp fail))
           ((variable? exp)
            (succeed (lookup-variable-value exp env)
   fail))
          ((quoted? exp)
           (succeed (text-of-quotation exp) fail))
   …))
```

What is succeed?

- Succeed is a procedure that takes two arguments, a value and a failure procedure.
  (succeed exp fail)
- In the driver-loop you can see that the success procedure passed will take a value and print it and call the failure if try-again is typed in as input.

What is fail?

- Well fail, as I said before, means to go and find a another value until it COMPLETELY fails which will go back to the driver-loop and you need to start a new problem.
- So you can think of amb as not having to always go back to the calling procedure. The driver-loop won't be called again until the problem you're working has failed.

Simple example…

- So what happens when I type:
  > 3
  What is the success proc? How about fail?

  What does this return?

  And what happens when I type try-again?

Example…

- So ambeval will get passed 3 as its expression
- The success procedure that does a print and a call to internal-loop with next-alternative.
- The failure is a procedure with no arguments that basically calls the driver-loop again.

Example…

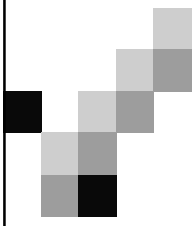
- (ambeval '3
  (lambda (val next)
  (print val)
  (internal-loop next))
  (lambda ()
  (display 'no-more-values)
  (newline)
  (driver-loop)))
- So it'll go into the self-evaluating clause and do (succeed exp fail)
- What will this do?

More example…

- So what happens when you do amb?

```
(define (eval-amb exp env succeed fail)
    (define (try-next choices)
        (if (null? choices)
          (fail)
          (ambeval (car choices)
                   env
                   succeed
                  (lambda ()
                    (try-next (cdr choices))))))
    (try-next (amb-choices exp))) ;; strips off 'amb' tag
```

- So let's trace this: (amb 3 4)



Next Time: Query

…pattern matching…