How functions finish

Direct: return answer;

True CPS: throw k answer;

uScheme: (k answer)
Design Problem: Missing Value

Provide a witness to existence:

\[(\text{witness } p? \; xs) = x, \; \text{where} \; (\text{member } x \; xs),\]
\[\text{provided} \; (\text{exists}? \; p? \; xs)\]

Problem: What if there exists no such \(x\)?
Solution: A New Interface

Success and failure continuations!

Laws:

\[(\text{witness-cps } p? \ x s \ \text{succ} \ \text{fail}) = (\text{succ} \ x)\]
; where \(x\) is in \(xs\) and \((p? \ x)\)

\[(\text{witness-cps } p? \ x s \ \text{succ} \ \text{fail}) = (\text{fail})\]
; where \((\text{not} \ (\text{exists? } p? \ x s))\)
Your turn: Refine the laws

\[(\text{witness-cps } p?\ x\ s\ u\ c\ c\ f) = (\text{succ } x)\]
; where \(x\) is in \(xs\) and \((p?\ x)\)

\[(\text{witness-cps } p?\ x\ s\ u\ c\ c\ f) = (\text{fail})\]
; where \((\text{not } (\text{exists? } p?\ xs))\)

\[(\text{witness-cps } p?\ '()\ s\ u\ c\ c\ f) = ?\]

\[(\text{witness-cps } p?\ (\text{cons } z\ zs)\ s\ u\ c\ c\ f) = ?\]
; when \((p?\ z)\)

\[(\text{witness-cps } p?\ (\text{cons } z\ zs)\ s\ u\ c\ c\ f) = ?\]
; when \((\text{not } (p?\ z))\)
Refine the laws

\[(\text{witness-cps } p\? \ x s \ \text{succ} \ \text{fail}) = (\text{succ} \ x)\]
\[
\quad; \text{where } x \text{ is in } x s \text{ and } (p\? \ x)
\]

\[(\text{witness-cps } p\? \ x s \ \text{succ} \ \text{fail}) = (\text{fail})\]
\[
\quad; \text{where } (\text{not} \ (\text{exists}? \ p\? \ x s))
\]

\[(\text{witness-cps } p\? \ '() \ \text{succ} \ \text{fail}) = (\text{fail})\]

\[(\text{witness-cps } p\? \ (\text{cons} \ z \ z s) \ \text{succ} \ \text{fail}) = (\text{succ} \ z)\]
\[
\quad; \text{when } (p\? \ z)
\]

\[(\text{witness-cps } p\? \ (\text{cons} \ z \ z s) \ \text{succ} \ \text{fail}) =
\quad(\text{witness-cps } p\? \ z s \ \text{succ} \ \text{fail})\]
\[
\quad; \text{when } (\text{not} \ (p\? \ z))\]
Coding \texttt{witness} with continuations

\begin{verbatim}
(define witness-cps (p? xs succ fail)
  (if (null? xs)
      (fail)
      (let ((x (car xs))
        (if (p? x)
            (if (p? x)
                (succ x)
                (witness-cps p? (cdr xs) succ fail))))))
\end{verbatim}
“Continuation-Passing Style”

All tail positions are continuations or recursive calls

(define witness-cps (p? xs succ fail)
  (if (null? xs)
    (fail)
    (let ((x (car xs)))
      (if (p? x)
        (if (p? x)
          (succ x)
          (witness-cps p? (cdr xs) succ fail))))))

Compiles to tight code
Example Use: Instructor Lookup

-> (val 2017f '((Fisher 105)(Cowen 170)(Chow 116)))
-> (instructor-info 'Fisher 2017f)
 (Fisher teaches 105)
-> (instructor-info 'Chow 2017f)
 (Chow teaches 116)
-> (instructor-info 'Souvaine 2017f)
 (Souvaine is-not-on-the-list)
Instructor Lookup: The Code

; info has form: '(Fisher 105)
; classes has form: '(info_1, ..., info_n)
(define instructor-info (instructor classes)
  (let ((
    (s ; success continuation
      (lambda (info)
        (list3 instructor 'teaches (cadr info))))
    (f ; failure continuation
      (lambda ()
        (list2 instructor 'is-not-on-the-list))
  )))
  (witness-cps pred
    classes s f)))
Instructor Lookup: The Code

; info has form: ' (Fisher 105)
; classes has form: ' (info_1, ..., info_n)
(define instructor-info (instructor classes)
  (let (
    (s ; success continuation
      (lambda (info)
        (list3 instructor 'teaches (cadr info))))
    (f ; failure continuation
      (lambda ()
        (list2 instructor 'is-not-on-the-list))))
  (witness-cps (o ((curry =) instructor) car)
    classes s f))
Instructor Lookup: The Code

; info has form: '(Fisher 105)
; classes has form: '(info_1, ..., info_n)
(define instructor-info (instructor classes)
  (let (
    (s (lambda (info) ; success continuation
        (list3 instructor 'teaches (cadr info)))))
    (f ; failure continuation
      ))
  (witness-cps (o ((curry =) instructor) car)
    classes s f))
Instructor Lookup: The Code

; info has form: ' (Fisher 105)
; classes has form: ' (info_1, ..., info_n)
(define instructor-info (instructor classes)
  (let (
    (s (lambda (info) ; success continuation
        (list3 instructor 'teaches (cadr info))))
    (f (lambda () ; failure continuation
        (list2 instructor 'is-not-on-the-list))))
    (witness-cps (o ((curry =) instructor) car)
                  classes s f)))
Exercise: Find a satisfying assignment if one exists

(val f1 ' (and x y z w p q (not x)))

(val f2 ' (not (or x y)))

(val f3 ' (not (and x y z)))

(val f4 ' (and (or x y z)
    (or (not x) (not y) (not z))))
Satisfying assignments

(val f1 '(and x y z w p q (not x))) ; NONE

(val f2 '(not (or x y)))
    ; { x |-> #f, y |-> #f } 

(val f3 '(not (and x y z)))
    ; { x |-> #f, ... } 

(val f4 '(and (or x y z)
             (or (not x) (not y) (not z))))
    ; { x |-> #f, y |-> #t, ... }
Continuations for Search

start +----------+ succeed
-------->| |------------>
| solver |
<--------|<----------
fail +----------+ resume

start  Gets partial solution, fail, succeed
(On homework, “solution” is assignment)
fail   Partial solution won’t work (no params)
succeed Gets improved solution + resume
resume If improved solution won’t work, try another (no params)