How functions finish

Direct: `return answer;`

True CPS: `throw k answer;`

uScheme: `(k answer)`

Design Problem: Missing Value

Provide a witness to existence:

```
(witness p? xs) == x, where (member x xs),
                            provided (exists? p? xs)
```

Problem: What if there exists no such `x`?

Solution: A New Interface

Success and failure continuations!

Laws:

```
(witness-cps p? xs succ fail) = (succ x)
     ; where x is in xs and (p? x)

(witness-cps p? xs succ fail) = (fail)
     ; where (not (exists? p? xs))
```

Your turn: Refine the laws

```
(witness-cps p? xs succ fail) = (succ x)
     ; where x is in xs and (p? x)
(witness-cps p? xs succ fail) = (fail)
     ; where (not (exists? p? xs))

(witness-cps p? '() succ fail) = ?

(witness-cps p? (cons z zs) succ fail) = ?
    ; when (p? z)

(witness-cps p? (cons z zs) succ fail) = ?
    ; when (not (p? z))
```

Refine the laws

```
(witness-cps p? xs succ fail) = (succ x)
     ; where x is in xs and (p? x)
(witness-cps p? xs succ fail) = (fail)
     ; where (not (exists? p? xs))

(witness-cps p? '() succ fail) = (fail)

(witness-cps p? (cons z zs) succ fail) = (succ z)
    ; when (p? z)

(witness-cps p? (cons z zs) succ fail) =
     (witness-cps p? zs succ fail)
    ; when (not (p? z))
```

Coding `witness` with continuations

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

“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)
             (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
                                                  )
     (f                    ; failure continuation
                                                 ))
     (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
                                                    )
     (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                  ; 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))))
```

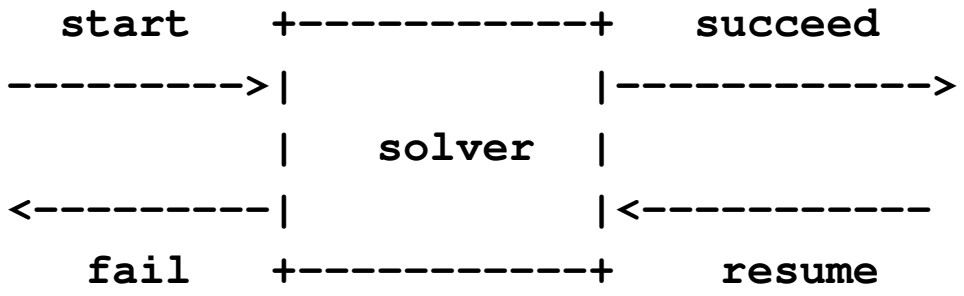
Satisfying assignments

```
(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	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)