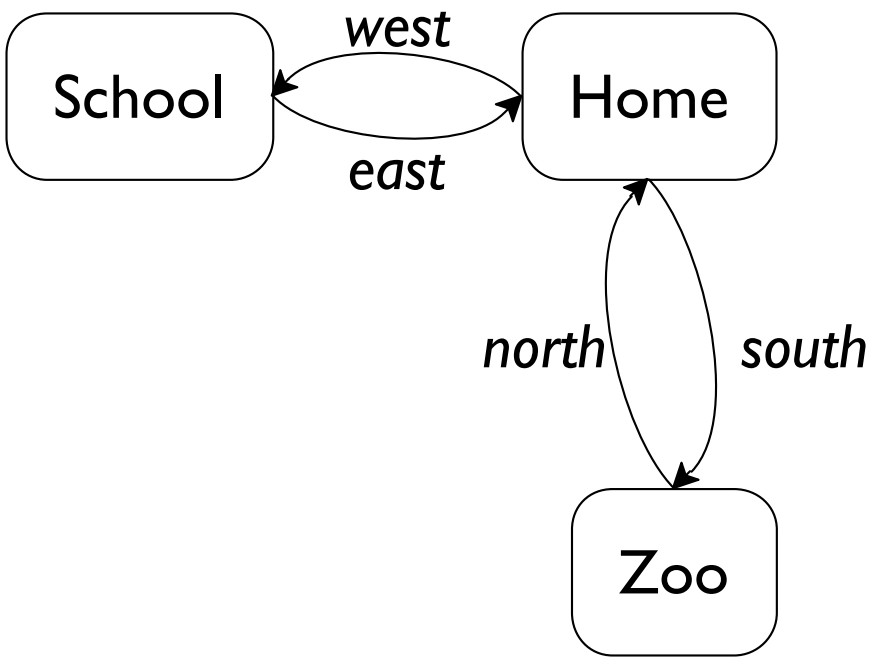
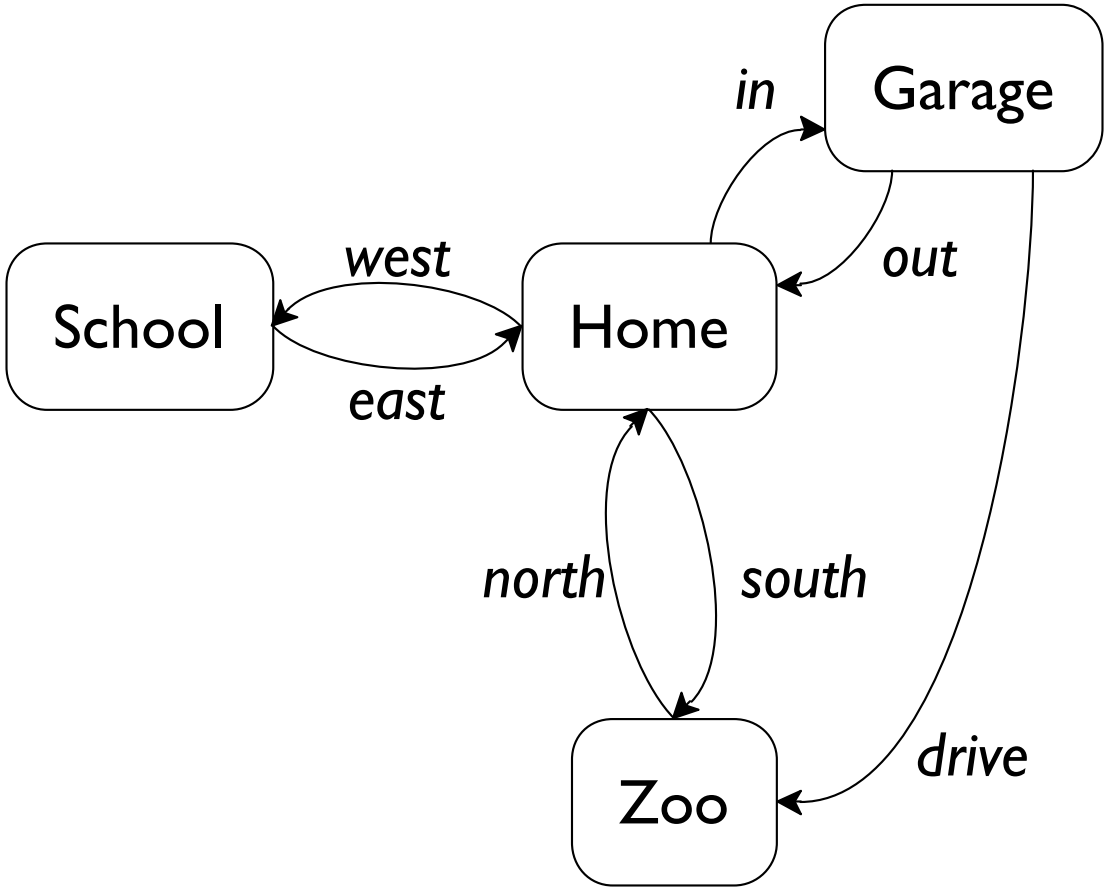
Text Adventure Game

```
"You are at Home"
> (go! "east")
"You are at School"
> (go! "east")
"You can't go that direction"
> (go! "west")
"You are at Home"
> (go! "south")
"You are at Zoo"
```

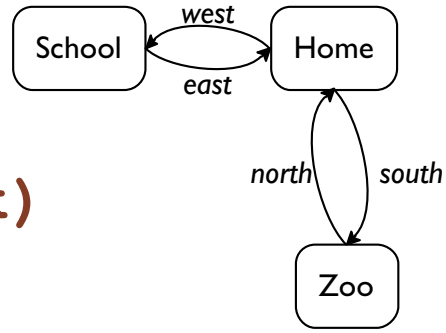
Map



Map



Model

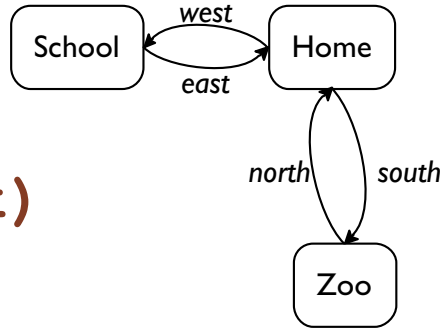



```
; A place is
;   (make-place string list-of-link)
(define-struct place (name links))

; A link is
;   (make-link string place)
(define-struct link (dir dest))

(define home
  (make-place "Home" ....))
```

Model

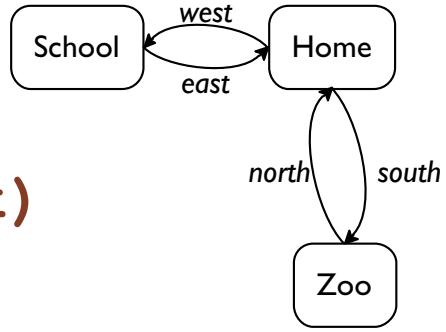



```
; A place is
;   (make-place string list-of-link)
(define-struct place (name links))

; A link is
;   (make-link string place)
(define-struct link (dir dest))

(define home
  (make-place "Home"
              (list (make-link "west" school)
                    (make-link "south" zoo))))
```

Model



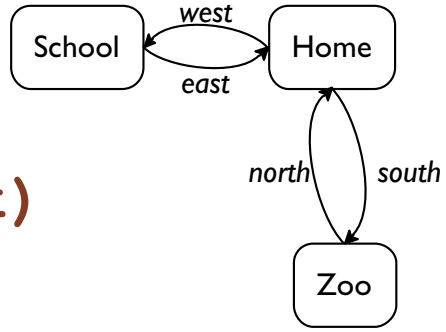

```
; A place is
;   (make-place string list-of-link)
(define-struct place (name links))

; A link is
;   (make-link string place)
(define-struct link (dir dest))

(define school
  (make-place "School" ...))

(define home
  (make-place "Home"
              (list (make-link "west" school)
                    (make-link "south" zoo))))
```

Model



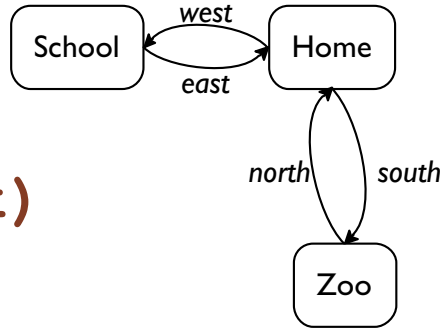

```
; A place is
;   (make-place string list-of-link)
(define-struct place (name links))

; A link is
;   (make-link string place)
(define-struct link (dir dest))

(define school
  (make-place "School"
              (list (make-link "east" home))))

(define home
  (make-place "Home"
              (list (make-link "west" school)
                    (make-link "south" zoo))))
```

Model




```
; A place is
;   (make-place string list-of-link)
(define-struct place (name links))

; A link is
;   (make-link string place)
(define-struct link (dir dest))

(define school
  (make-place "School"
              (list (make-link "east" home))))

(define home
  (make-place "Home"
              (list (make-link "west" school)
                    (make-link "south" zoo))))
```

Cannot use **home** before it is defined

Creating Cycles with Assignment

```
(define school (make-place "School" empty))

(define zoo (make-place "Zoo" empty))

(define home
  (make-place "Home"
              (list (make-link "west" school)
                    (make-link "south" zoo))))

(set-place-links! school
                  (list (make-link "east" home)))
(set-place-links! zoo
                  (list (make-link "north" home)))
```

Creating Cycles with shared

```
(define current-place
  (shared ([home
            (make-place "Home"
                        (list (make-link "west" school)
                              (make-link "south" zoo)))]
           [school
            (make-place "School"
                        (list (make-link "east" home)))]
           [zoo
            (make-place "Zoo"
                        (list (make-link "north" home)))])
    home))
```

Moving

```
; go! : string -> string
; Changes the current place by moving in the
;  given direction, if possible, and returns a
;  description of the new state
; Effect: sets current-place
(define (go! dir)
  (local [(define new-place
            (find-place dir
                        (place-links current-place)))]
    (cond
     [(place? new-place)
      (begin
        (set! current-place new-place)
        (string-append "You are at " (place-name new-place)))]
     [else
      "You can't go that direction"])))
```

Finding a Move

```
; find-place : string list-of-link -> place-or-false
(define (find-place dir links)
  (cond
   [(empty? links) false]
   [(cons? links)
    (if (in-direction? dir (first links))
        (link-dest (first links))
        (find-place dir (rest links)))]))

; in-direction? : string link -> boolean
(define (in-direction? dir link)
  (string=? (link-dir link) dir))
```

Looking for a Place

Given a string and a place, determine whether some place with the given string name is reachable from the given place.

The solution requires an accumulator to record where you've been—otherwise you might go in circles

Stuff

Adjust `place` so that each place has a list of things. Report all of the things in a location after moving.

Implement `get!`, which picks up a thing in the current room. After picking up something, it is no longer in the room, but is instead in the player's possession. After picking up somethig, report everything in the player's possession.