These slides accompany Lecture 26 and are integral to understanding the lecture notes. There is just too much code to write longhand at the board.

– keith
A city is a symbol.

;; The information for a city can be represented as a structure
;; (make-city-info name dests)
;; where c is a city (symbol) and dests is a list of symbols
(define-struct city (name dests))

;; A route-map is a list of city-info
(define routes
  (list (make-city-info 'Houston   (list 'Dallas 'NewOrleans))
        (make-city-info 'Dallas   (list 'LittleRock 'Memphis))
        (make-city-info 'NewOrleans (list 'Memphis))
        (make-city-info 'Memphis  (list 'Nashville)) )
)

;; find-flights: city city route-map → (list of city) or false
;; Purpose: create a path of flights from start to finish or return false
(define (find-flights start finish rm) …)

Examples:
(find-flights 'Houston 'Houston routes)
= (list 'Houston)

(find-flights 'Houston 'Dallas)
= (list 'Houston 'Dallas)

(find-flights 'Dallas 'Nashville)
= (list 'Dallas 'LittleRock 'Memphis 'Nashville)
Original Version

;; find-flights: city city route-map → (list of city) or false
;; Purpose: create a path of flights from start to finish or return false
(define (find-flights start finish rm)
  (cond
    [(symbol=? start finish) (list start)]
    [(else
      (local [[define possible-route
                (find-flights-for-list (direct-cities start rm) finish rm)]
                (cond
                  [(boolean? possible-route) false]
                  [else (cons start possible-route)]))]
  ))

;; direct-cities: city route-map → list-of-city
;; Purpose: return a list of all cities in the route map with direct flights
;;   from the city given as an argument
(define (direct-cities from-city rm)
  (local [[define from-city-info
            (filter (lambda (c)(symbol=? (city-info-name c) from-city)) rm)]
            (cond
              [(empty? from-city-info) empty]
              [else (city-info-dests (first (from-city-info)))]))

;; find-flights-for-list: list-of-city city route-map → list-of-city or false
;; Purpose: finds a flight route from some city in the input list to the
;;   destination, or returns false if no such route can be found.
(define (find-flights-for-list aloc finish rm)
  (cond
    [(empty? aloc) false]
    [else
      (local [[define possible-route
               (find-flights (first aloc) finish rm)]
               (cond
                 [(boolean? possible-route)
                  (find-flights-for-list (rest aloc) finish rm)]
                 [else possible-route]))])))
With Institutional Memory

;; find-flights: city city route-map (list of city) \(\rightarrow\) (list of city) or false
;; Purpose: create a path of flights from start to finish or return false
(define (find-flights start finish rm visited)
  (cond
    [(symbol=? start finish) (list start)]
    [(memq start visited)   false] ;; cut off this search path
    [(else
      (local [(define possible-route
                (find-flights-for-list (direct-cities start rm) finish
                                      rm  (cons start visited)))
                (cond
                  [(boolean? possible-route) false]
                  [else  (cons start possible-route)]) ] ]))

;; direct-cities: city route-map \(\rightarrow\) list-of-city
;; Purpose: return a list of all cities in the route map with direct flights
;; from the city given as an argument
(define (direct-cities  from-city  rm)
  (local [(define  from-city-info
                (filter (lambda (c)(symbol=? (city-info-name c) from-city)) rm))
            (cond
              [(empty? from-city-info)  empty]
              [else (city-info-dests (first (from-city-info)))]))))

;; find-flights-for-list: list-of-city city route-map (list of city)
;; \(\rightarrow\) list-of-city or false
;; Purpose: finds a flight route from some city in the input list to the
;; destination, or returns false if no such route can be found.
(define (find-flights-for-list aloc finish rm visited)
  (cond
    [(empty? aloc)  false]
    [else
      (local [(define possible-route
                  (find-flights (first aloc) finish rm visited))
                (cond
                  [(boolean? possible-route) false]
                  [else (find-flights-for-list (rest aloc) finish rm visited)]
                  [else  possible-route)])])))