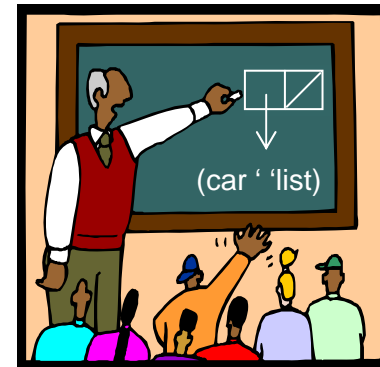


6.001 recitation

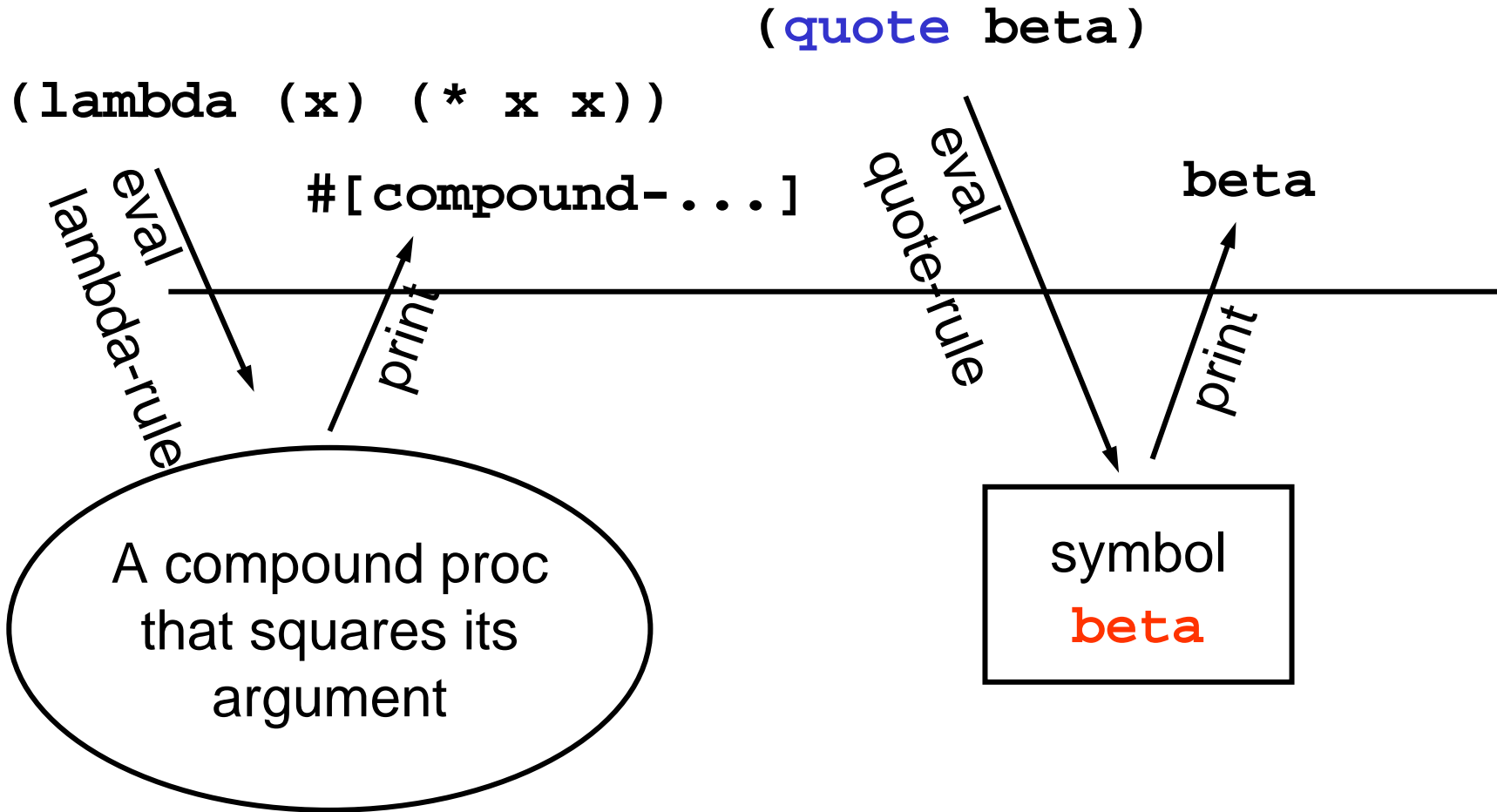
3/14/06

- Symbols
- Robots & A-lists



Dr. Kimberle Koile

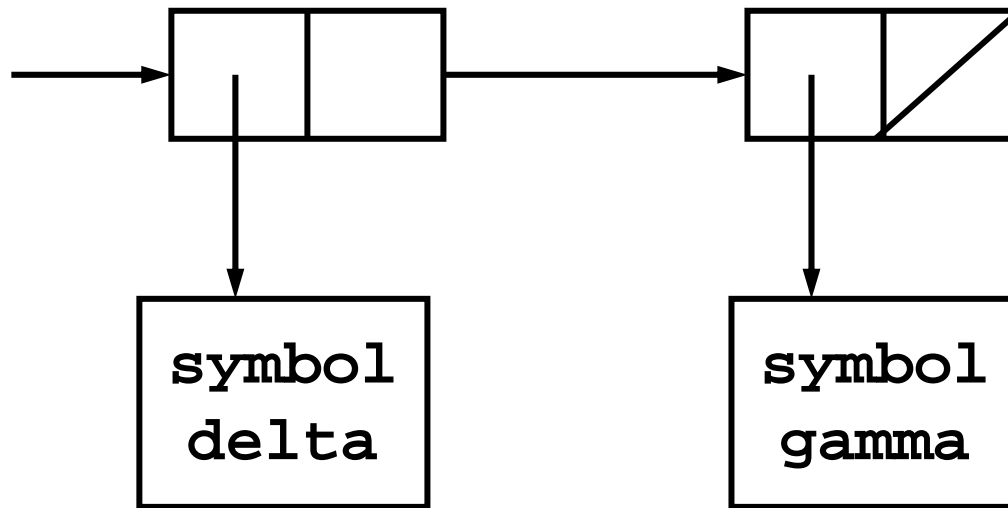
# Symbols in Scheme



## Symbols are ordinary values

`(list 1 2)`  $\Rightarrow$  `(1 2)`

`(list (quote delta) (quote gamma))`  
 $\Rightarrow$  `(delta gamma)`



practice

---

`(define a 1)`

`(define b 2)`

What does the Scheme interpreter print for each of the expressions:

`(list a b)`

`(list 'a 'b)`

`(list 'a b)`

`(car 'a)`

more practice

---

`(define a 1)`

`(define b 2)`

What does the Scheme interpreter print for each of the expressions:

`(car "a)`

`(cadr "list)`

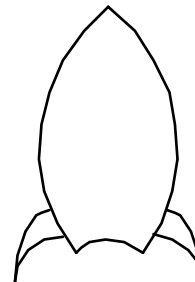
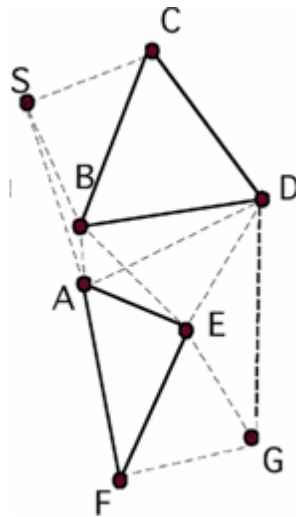
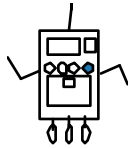
`((cadr 'list) a b)`

# Wallace and Gromit

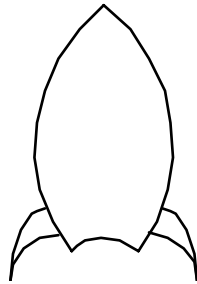
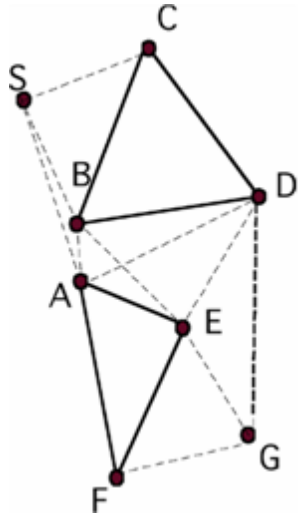
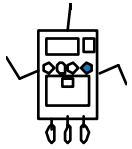


## data abstraction, symbols, and search

Wallace and Gromit have just finished their vacation on the moon and are about to head back to Earth in their rocket ship (located at position G below). The local robot desperately wants to go back with them, but must hurry to get to the rocket ship in time. (He's at S below.) He has to navigate around two obstacles (shown as triangles AEF and BCD.) He uses his nifty search engine to find the best path. In recitation 14 (October 27) we'll figure out which way he goes. Today let's figure out the representations needed for his search engine. Below is a *graph* representing possible paths from the robot's starting location (S) to the rocket ship's location (G). The graph consists of *nodes* (labeled S, and A to G) which are connected by *links* (aka arcs or edges). Nodes have such properties as id, e.g., S; links in which they are endpoints; and estimated distances to the goal node. Links have properties such as the nodes that are endpoints and length; e.g., the link between S and B has endpoints node S and node B, and a length of 5 (units not specified). *Paths* through the graph can be represented as ordered sets of nodes and/or links.



# data representation: symbols and alists



Link lengths:

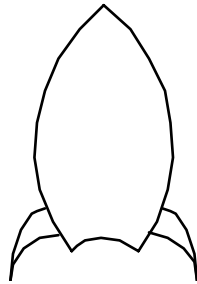
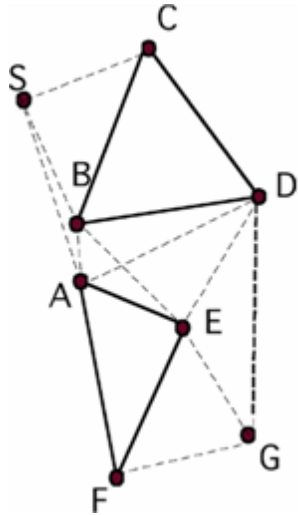
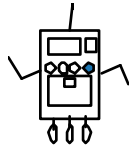
S-A	6	B-D	6
S-B	5	B-E	5
S-C	4	C-D	6
A-B	1	D-E	5
A-D	7	D-G	8
A-E	3	E-F	6
A-F	7	E-G	4
B-C	6	F-G	4

Estimates of distance to G from:

A	7
B	9
C	13
D	7
E	4
F	4
G	0
S	1



# data abstraction: symbols and alists



```
(define *node-data* '((A 7) (B 9) (C 13) (D 7) (E 4) (F 4) (G 0) (S 1)))
```

```
(define *link-data* '(((S A) 6) ((S B) 5) ((S C) 4) ((A B) 1) ((A D) 7) ((A E) 3)  
  ((A F) 7) ((B C) 6) ((B D) 6) ((B E) 5) ((C D) 6) ((D E) 5)  
  ((D G) 8) ((E F) 6) ((E G) 4) ((F G) 4)))
```

## alist practice: assoc

---

```
(define *node-data* '((A 7) (B 9) (C 13) (D 7) (E 4) (F 4) (G 0) (S 1)))
```

**What does the Scheme interpreter print for each of the expressions:**

(assoc 'A \*node-data\*) =>

(assoc A \*node-data\*) =>

(assoc 9 \*node-data\*) =>

(assoc '7 \*node-data\*) =>

(assoc '(C 13) \*node-data\*) =>

## nodes and links

```
(define *node-data* '((A 7) (B 9) (C 13) (D 7) (E 4) (F 4) (G 0) (S 1)))
```

```
(define *link-data* '(((S A) 6) ((S B) 5) ((S C) 4) ((A B) 1) ((A D) 7) ((A E) 3)  
  ((A F) 7) ((B C) 6) ((B D) 6) ((B E) 5) ((C D) 6) ((D E) 5)  
  ((D G) 8) ((E F) 6) ((E G) 4) ((F G) 4)))
```

To get the estimated distance to the goal for a node, we could use the *\*node-data\** list and the procedure *assoc* or *find-assoc*:

```
(define (get-node-estimate node-id)
```

```
)
```

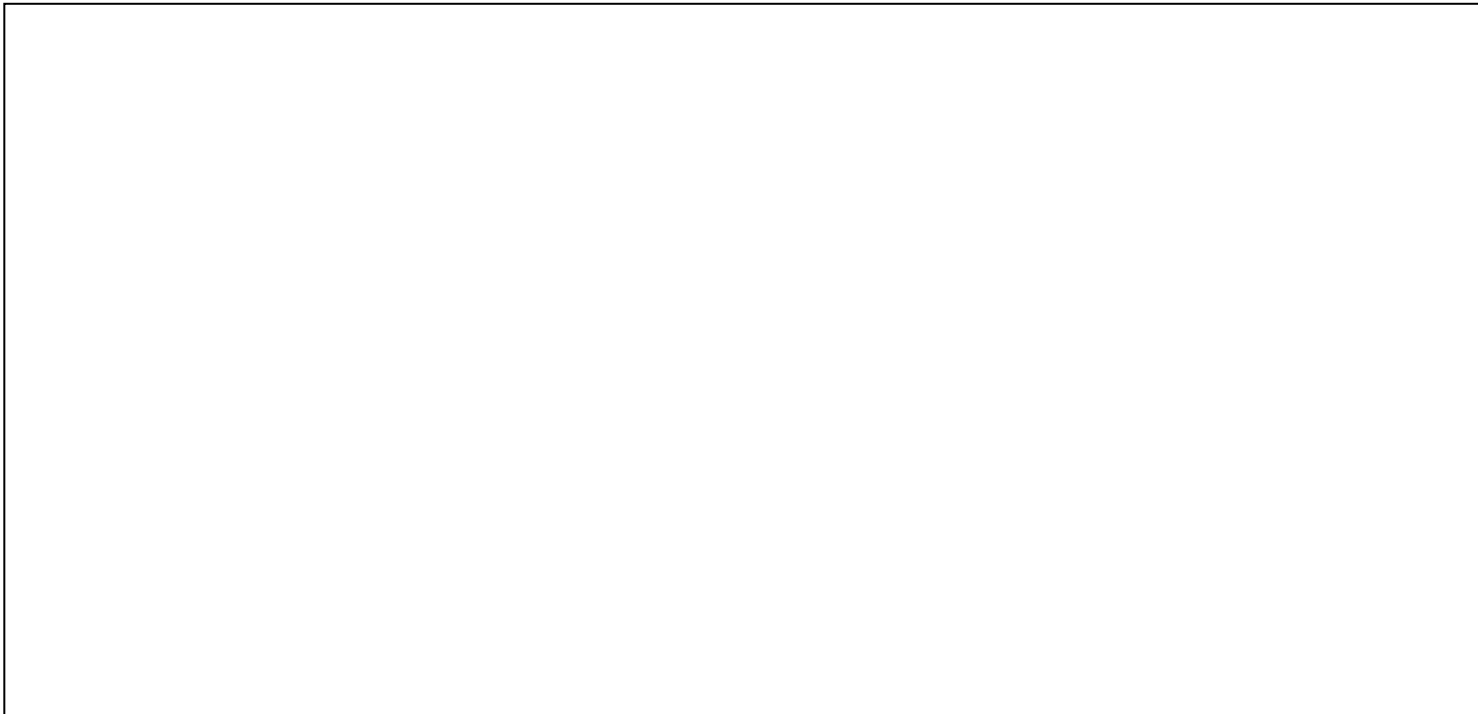
## nodes and links

```
(define *node-data* '((A 7) (B 9) (C 13) (D 7) (E 4) (F 4) (G 0) (S 1)))
```

```
(define *link-data* '(((S A) 6) ((S B) 5) ((S C) 4) ((A B) 1) ((A D) 7) ((A E) 3)  
  ((A F) 7) ((B C) 6) ((B D) 6) ((B E) 5) ((C D) 6) ((D E) 5)  
  ((D G) 8) ((E F) 6) ((E G) 4) ((F G) 4)))
```

To get the length of a link, we could use `*link-data*` and `assoc` or `find-assoc`:

```
(define (get-link-length node-id1 node-id2)
```



```
)
```

## creating nodes using alists

Consider these representations for nodes and links:

```
(define (make-node id estimate-to-goal)
  (cons id estimate-to-goal))
(define (node-id node)
  (car node))
(define (node-estimate-to-goal node)
  (cdr node))
(define (make-link node1 node2 length)
  (list (list node1 node2) length)))
```

```
(define *node-data* '((A 7) (B 9) (C 13) (D 7) (E 4) (F 4) (G 0) (S 1)))
```

**1. Use map to create nodes using \*node-data\*.**

```
(define *nodes*
```



```
)
```

```
)
```

## finding a node

**2. Find a node in *\*nodes\** given a symbol representing a node id.**

Use this function to test for node-id equality:

```
(define equal-node-id? (id1 id2)
```

```
)
```

```
(define find-node (id)
```

```
)
```

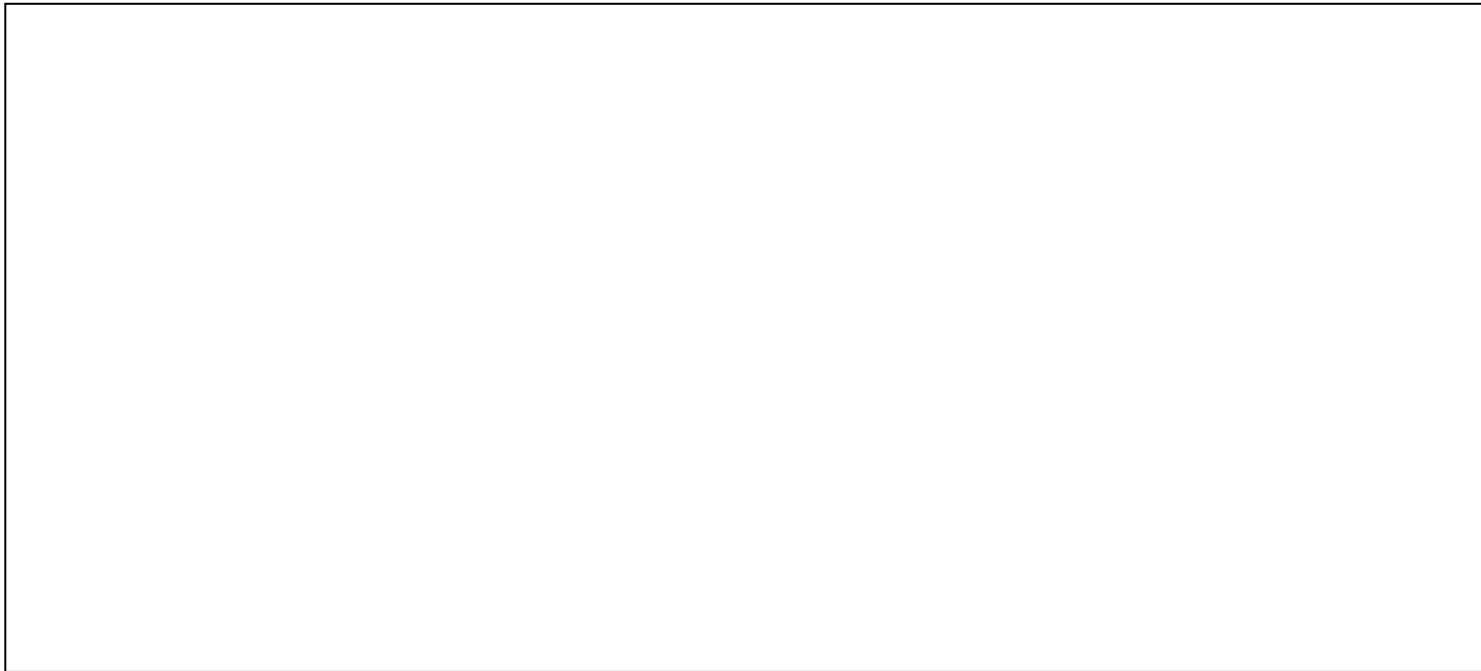
## creating a link

### 3. Use map to create links using *\*link data\**.

```
(define *link-data* '(((S A) 6) ((S B) 5) ((S C) 4) ((A B) 1) ((A D) 7) ((A E) 3)  
  ((A F) 7) ((B C) 6) ((B D) 6) ((B E) 5) ((C D) 6) ((D E) 5)  
  ((D G) 8) ((E F) 6) ((E G) 4) ((F G) 4)))
```

```
(define (make-link node1 node2 length)  
  (list (list node1 node2) length))
```

```
(define *links*
```



```
)
```

## creating a node containing links!

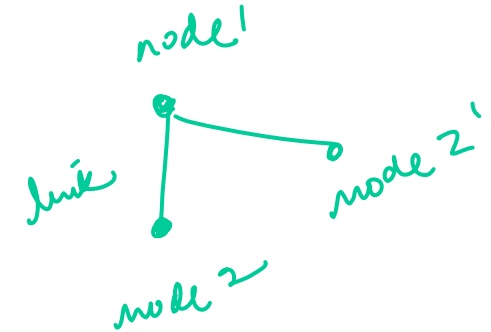
### 4. Assume our representation for nodes now includes links:

```
(define (make-node id estimate-to-goal links)
  (list id estimate-to-goal links))
```

```
(define (make-link node1 node2 length)
  (list (list node1 node2) length))
```

```
(define *node-data* '((A 7) (B 9) (C 13) (D 7) (E 4) (F 4) (G 0) (S 1)))
```

```
(define *link-data* '(((S A) 6) ((S B) 5) ((S C) 4) ((A B) 1) ((A D) 7) ((A E) 3)
  ((A F) 7) ((B C) 6) ((B D) 6) ((B E) 5) ((C D) 6) ((D E) 5)
  ((D G) 8) ((E F) 6) ((E G) 4) ((F G) 4)))
```



How do we create links when we need nodes + they haven't been created yet?!





