MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001 Structure and Interpretation of Computer Programs
Spring, 2007

Recitation 5, Friday February 23

Data Abstraction: cons, list Dr. Kimberle Koile

Language Elements

- Primitives
  - primitive data: numbers, strings, Booleans
  - primitive procedures
- Means of Combination
  - procedure application
  - compound data
- Means of Abstraction
  - naming
  - compound procedures
    - block structure
    - higher order procedures
  - conventional interfaces – lists
  - data abstraction
    - constructors
    - accessors
    - contract
    - operations

Data Abstractions

cons

1. Constructor

```
;cons: A, B → Pair<A,B>; A & B = anytype
(cons <x> <y>) → <p>
```

2. Accessors

```
(car <p>)        ; car: Pair<A,B> → A
(cdr <p>)        ; cdr: Pair<A,B> → B
```

3. Contract

```
(car (cons <x> <y>)) → <x>
(cdr (cons <x> <y>)) → <y>
```

4. Operations

```
; pair?: anytype → boolean
(pair? <p>)
```

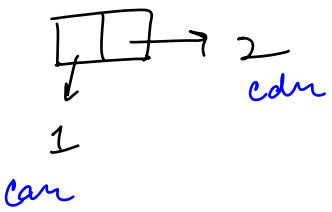
5. Abstraction Barrier

IGNORANCE NEED TO KNOW

6. Concrete Representation & Implementation

Could have alternative implementations!

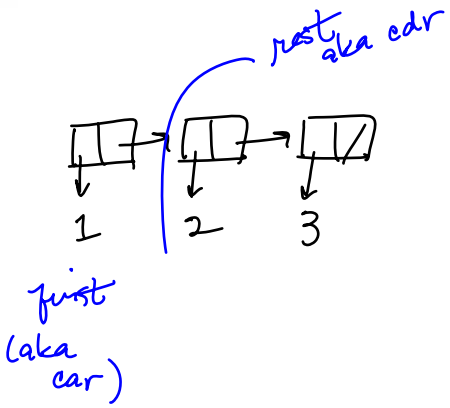
(cons 1 2)



(1 . 2) printed repr

list

1. Constructor
   (list <a> <b> …) => <l>
2. Accessors
   (first <l>)
   (rest <l>)
3. Contract
   (first (list <a> <b> <c>)) => <a>



(rest (list <a> <b> <c>)) => (<b> <c>)

list (cont'd)

4. Operations

   (list? <l>) ; returns #t if <l> is a list

   (adjoin <z> <l>) ; adds <z> to the front of the list

   …

5. Abstraction Barrier

6. Concrete Representation and Implemenation

   (cons <a> (cons <b> (cons <c> '())))

   (define first car)

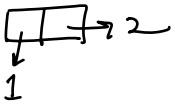
   (define rest cdr)

   (define adjoin cons)

Examples

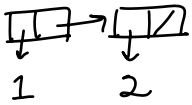
(define a 1)
(define b 2)
(define c 3)

(car (cons a b)) ==> 1



(cdr (cons a b)) ==> 2

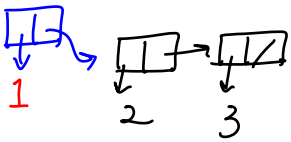
(first (list a b)) ==> 1



(rest (list a b)) ==> (2)

(pair? (list a b)) ==> #t

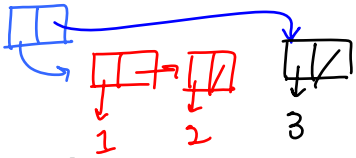
(adjoin a (list b c)) ==> (1 2 3)



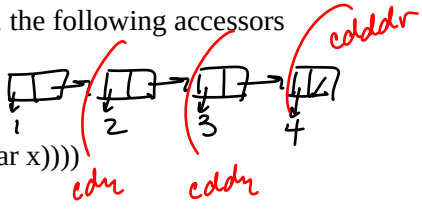
what is want (2 3 1)?
(append (list b c)
(list a))

addr of a cons cell

(adjoin (list a b) (list c)) ==>



In Scheme, we often want to access elements deep in a cons structure. Therefore, the following accessors have been defined to help us out:



(cadr x) ==> (car (cdr x))
(caddr x) ==> (car (cdr (cdr x)))
(cdaar x) ==> (cdr (car (car x)))

(cddr x) ==> (cdr (cdr x))
(cdadar x) ==> (cdr (car (cdr (car x))))

For lists, we also often want to easily access the n'th element of a list. The accessors first, second, third, ..., tenth are defined to access the corresponding values of a list. For example, (third (list 1 2 3 4)) => 3

How could you define first, second, third, and fourth using the c???r functions?

(first x) ==> (car x)

(second x) ==> (cadr x)

(third x) ==> (caddr x)

(fourth x) ==> (cadddr x), i.e. (car (cdr (cdr (cdr x))))