CSE 413: Programming Languages and their Implementation

Scheme - Lists
Ruth Anderson
Autumn 2007

Today’s Outline
• Administrative Info – Office Hours
• More Scheme
  » cons, car, cdr
  » Processing Lists

Office Hours
• Ruth Anderson
  • M 3:30-4:30, Thurs 1-2pm
• Jeremy Brudvik
  • Wed 3:30-4:30, Thurs 12-1pm
• Paramjit Sandhu
  • Tues 2:30-3:30, Thurs 2:30-3:30

(cons a b)
• Takes a and b as args, returns a compound data object that contains a and b as its parts
• We can extract the two parts with accessor functions car and cdr ("could-er")

(car (cons 3 4))
(car (cons 3 4))

(car (cons a (cons 3 4)))
(car (cons a (cons 3 4)))

(car (cons (car b) (cons a 3)))
(car (cons (car b) (cons a 3)))

(car (cdr b))
(cdr (car b))
(cdr b)

(car (cons 3 4))
(car (cons 3 4))

(define a (cons 1 2))
(define a (cons 1 2))

(a 1 2)
(b 1 2)

(car a)
(car (cons (car b) (cons a 3)))

(cdr a)
(cdr (car b))

(cdr (cons a (cons 3 4)))
(cdr b)

(cdr (cons 1 2))

3
\( (\text{car} \ (\text{cdr} \ c)) \)

\[
(\text{define} \ c \ (\text{cons} \ (\text{cons} \ 1 \ 2) \ (\text{cons} \ 3 \ 4)))
\]

\[
\text{c} \\
\text{(car (cdr c))} \\
\text{(cdr (car c))} \\
\text{(car (cdr c))} \\
\text{(cdr (cdr c))}
\]

\( (\text{cadr} \ c) \)

• We can abbreviate the repeated use of car and cdr

\[
(\text{define} \ c \ (\text{cons} \ (\text{cons} \ 1 \ 2) \ (\text{cons} \ 3 \ 4)))
\]

\[
\text{c} \\
\text{(cadr c)} \\
\text{(car (cdr c))} \\
\text{(cdr (car c))} \\
\text{(cdr (cdr c))}
\]

\( \text{pair? predicate} \)

• \((\text{pair?} \ z)\) is true if \(z\) is a pair

\[
(\text{define} \ c \ (\text{cons} \ (\text{cons} \ 1 \ 2) \ (\text{cons} \ 3 \ 4)))
\]

\[
\text{c} \\
\text{(pair? c)} \\
\text{(pair? (car c))} \\
\text{(pair? (cdr c))} \\
\text{(pair? (caar c))} \\
\text{(pair? (cdar c))}
\]

\( \text{nil} \)

• if there is no element present for the car or cdr branch of a pair, we indicate that with the value \(\text{nil}\)

• \((\text{null?} \ z)\) is true if \(z\) is \(\text{nil}\)

\[
(\text{define} \ d \ (\text{cons} \ 1 \ '()))
\]

\[
\text{d} \\
\text{(car d)} \\
\text{(cdr d)} \\
\text{(null? (car d))} \\
\text{(null? (cdr d))}
\]

\( \text{Lists} \)

• By convention, a list is a sequence of linked pairs
  » \(\text{car}\) of each pair is the data element
  » \(\text{cdr}\) of each pair points to list tail or the empty list

\[
(\text{define} \ e \ (\text{cons} \ 1 \ (\text{cons} \ 2 \ (\text{cons} \ 3 \ '()))))
\]

\[
e \\
1 \\
\text{(define} \ e \ (\text{list} \ 1 \ 2 \ 3))
\]
procedure list

(list a b c ...)

- `list` returns a newly allocated list of its arguments
  - the arguments can be atomic items like numbers or quoted symbols
  - the arguments can be other lists
- The backbone structure of a list is always the same
  - a sequence of linked pairs, ending with a pointer to null (the empty list)
  - the `car` element of each pair is the list item
  - the list items can be other lists

List structure

Examples of list building

- (cons 1 (cons 2 '(())))
- (cons 1 (list 2))
- (list 1 2)

How to process lists?

- A list is zero or more connected pairs
- Each node is a pair
- Thus the parts of a list (this pair, following pairs) are lists
- A natural way to express list operations?

cdr down

(define (length m)
  (if (null? m)
      0
      (+ 1 (length (cdr m)))))

sum the items in a list

(define (add-items list)
  (+ 1 (add-items (cdr list))))

(define a (list 4 5 6))
(define b (list 7 a 8))

(define (length m)
  (if (null? m)
      0
      (+ 1 (length (cdr m)))))

(define (add-items list)
  (+ 1 (add-items (cdr list))))

(define a (list 4 5 6))
(define b (list 7 a 8))