CSE 413: Programming Languages and their Implementation

Scheme - Lists

Hal Perkins

Autumn 2008
Today’s Outline

• Administrative Info – Office Hours
• More Scheme
  » cons, car, cdr
  » Processing Lists
Office Hours &c.

- **Office Hours**
  - Hal Perkins – CSE 548
    - Mon. after class to 5:30, Tue 2:00 – 3:00
  - Laura Marshall – CSE 218
    - Thur. 1:30 – 3:00

- **Assignment 1 is out – due next Thur., 11 pm**
  - Most everything needed by today; loose ends Monday

- **Email**
  - Please put “413” somewhere in the subject
  - We’re outnumbered & can’t be a 24/7 quick-response help desk. Take advantage of discussion list, references, online resources.
(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 \texttt{car} and \texttt{cdr}

```
(define a (cons 'x 'y))
```
car and cdr

(define a (cons 'x 'y))
(car a)
(cdr a)

• We can build arbitrary pairs with cons, but the workhorse data structures in Scheme are proper lists
Lists

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

```
    e → 1 → 2 → 3
```

CSE 413 Au 08 - Scheme - Lists 6
nil

• if there is no element present for the car or cdr branch of a pair, we indicate that with the value nil
  » nil (or null) represents the empty list '()

• (null? z) is true if z is nil

```
(define d (cons 'x '()))
(car d)
(cdr d)
(null? (car d))
(null? (cdr d))
```
(define e (cons 1 (cons 2 (cons 3 '()))))

\[
\begin{array}{c}
\text{e} \\
1 \\
2 \\
3 \\
\end{array}
\]

(define e (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

(define a (list 4 5 6))

(define b (list 7 a 8))
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?
(define (length m)
    (if (null? m)
        0
        (+ 1 (length (cdr m))))
)
sum the items in a list

(\texttt{add-items \ (list 2 5 4)})