CS 603: Programming Language Organization

Lecture 10

Spring 2005

Department of Computer Science

University of Alabama

Joel Jones

Outline

- Questions
  - Yours
  - What most differentiates Scheme from C or C++?
- μ-Scheme
- Reading for next time

μ-Scheme

- What distiniguishes scheme from C++?

Pair Up:

What does distinguish?

  - Recursion,
  - anonymous functions
  - functions as values => higher order functions

Values in μ-Scheme

- Compare values of Impcore to Scheme's

> Pair Up:
>
> What values does μ-Scheme have?

- Compare aggregations of C to Schemes—i.e. (struct, union, arrays) vs. lists

> Pair Up:
>
> How can you "fake" structs, unions, and arrays using lists?

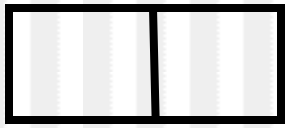
Impcore vs. μ-Scheme

Pair Up:

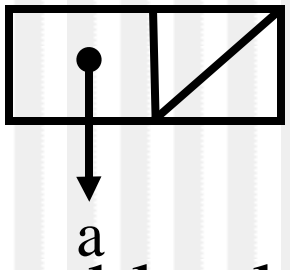
What does `(+ 2 3)` mean in impcore? In μ-Scheme? What does `'(+ 2 3)` mean in μ-Scheme?

Box/cons cell model

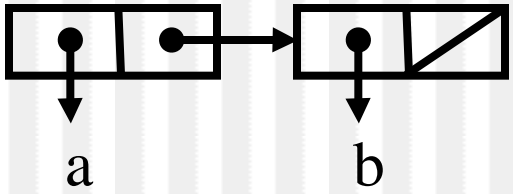
- Cons cell–primitive data structure used to construct lists, basically a 2-tuple, drawn as:



- A single element list, `'(a)` is drawn as:



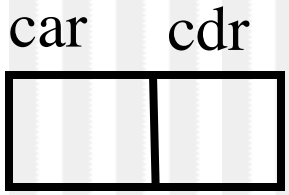
- A double element list, `'(a b)` is drawn as:



Lists (cont.)

- To answer previous question, `'(+ 2 3)` is a list

> Pair Up:
>
> What is the box diagram for `'(+ 2 3)`? Draw it on the board.



- car–what first element points to
- (car '(a b)) = a
- cdr–what second element points to
- (cdr '(a b)) = '(b)

Lists (cont.)

- Cons–constructs/makes a new cons cell
  - `'(+ 2 3)` is shorthand for:
  - `(cons '+ (cons 2 (cons 3 '())))`
- Pronunciation:
  - `car` = kawr, `cdr` = cudder, `cons` = kawns
- More complicated formed by concatenation:
  - `(cadr L)` = `(car (cdr L))`, katter
  - `(cddr L)` = `(cdr (cdr L))`, kadidder
  - `(cdar L)` = `(cdr (car L))`, kadar

Lists—Examples

Pair Up:

Write a `cons` expression to get `'((a) b)`.

Write the expression to get the "a" from `'((a) b)`.

- S-expression predicates, append, +1 +2 => map

S-expressions

- s-expression – a *symbol*, a *number*, or a *list* $(S_1, \ldots, S_n)$ of zero or more S-expressions
- nil list – list of zero elements ‘ ( )

Operations on S-Expressions:

- Review
  - `car`: first element of list or `'()`
  - `cdr`: everything in list except first element
  - `cons`: if $S = (S_1,\ldots,S_n)$, then `(cons S' S)` is $(S' S_1,\ldots,S_n)$
- Predicates
  - `=`: returns `#t` if equal, `#f` otherwise, for non-lists and empty lists
  - `number?`, `symbol?`, `list?`, `null?`: `#t` if argument of type, `#f` otherwise
  - `<`, `>`: `#t` if both numbers and condition holds, `#f` otherwise
- Math
  - `+`, `-`, `*`, `/`: like impcore

Syntax
(changes from impcore)

- *value* ::= integer | *quoted-const* | `#t` | `#f`
- *value-op* ::= `+` | `-` | `*` | `/` | `=` | `<` | `>` | `cons` | `car` | `cdr` | `number?` | `symbol?` | `list?` | `null?` | `print`
- *quoted-const* ::= ‘ *S-expression*
- *S-expression* ::= integer | *symbol* | (*S-expression**)
- *symbol* ::= name

Example of a list using function

```
(define length (l)
   (if (null? l)
      0
      (+ 1 (length (cdr l)))))
```

Reading & Questions for Next Class

- Chapter 3.8–3.10
- You might also want to look at the online version of the book "Structure and Interpretation of Computer Programs" at: http://mitpress.mit.edu/sicp/