Scheme

Albert Ye

October 31, 2022

Scheme is a dialect of lisp from the '70s but somehow Berkeley decides it's still relevant to teach.

1 Scheme expressions

Scheme has two types of expressions, which can either be **primitive** or **combinations**.

From what I gather **primitive expressions** are just one variable/function (such as "2" or "quotient") whereas combinations are adding variables and functions to evaluate things. Call expressions include an operator and 0 or more operands in parentheses.

For example: `(quotient 10 2)` is a valid expression that returns `5`.

2 Special forms

A **special form** is a combination that isn't a call expression.

- if expressions:
  `(if <predicate> <consequent> <alternative>)`
- and/or:
  `(and <e1> ... <en>)`
  `(or <e1> ... <en>)`
- binding symbols:
  `(define <symbol> <expression>)`
- new procedures:
  `(define (<symbol> <formal parameters>) <body>)`

2.1 define form

Takes an expression and binds the value of the expression to a name (which must be a valid Scheme symbol)

For example, `(define x 2)` sets `x` to the value 2.

Using the `let` command allows you to set the value temporarily without binding it to the name. For example: `(define c (let ((a 3) (b (+ 2 2))) (sqrt (+ (* a a) (* b b)))))`

2.2 define procedure

Constructs a new procedure with specified parameters and uses the `body` expressions as its body which it binds to `name`.

For example `(define (double x) (* 2 x))` defines a function

A **lambda expression** is just an anonymous procedure.

2.3 list procedure

"you use a lot of recursion
and lists are linked lists" -chez

The **list** procedure takes in an arbitrary number of arguments and constructs a list with the values of these arguments.

`(list 1 2 3)` makes a list `1, 2, 3`

`(list 1 (list 2 3) 4)` makes a list `1, (2, 3), 4`

`(list (cons 1 (cons 2 nil)) 3 4)` makes `(1, 2), 3, 4`

2.4 car and cdr and cons

Not related to quotation but it first showed up here; **car** is the first element of a list, **cdr** is the list of remaining elements after the first, and **cons** is short for construct which appends to a list.

car and **cdr** pretty much make all lists follow the linked list paradigm.

If you only want to construct one element with another element, then you can just use `(cons 1 nil)`.

3 Quotation

Symbols typically refer to values, and quotation is used to refer to the symbols directly.

`(list 'a 'b)` makes the list `a, b` while `(list a b)` makes the list `1, 2` if $a = 1, b = 2$.

`'(neverseen)` returns "neverseen" because it's still a symbol, the only caveat is that nothing can be done with "neverseen".

4 List Processing

- `append a b`, adds the value `b` to the list `a`.
- `filter cond list`, returns a list of all $x \in list$ that satisfy `cond`
- `apply func list` applies `func` over all values in `list`.
- `map lambda(x) list` returns the elements of a list after applying a lambda over it.
- `eval exp` basically allows you to run a program `exp` you just put into a list