CS 61A Week 6

Topic: Generic Operators

Reading: Abelson & Sussman, Sections 2.4 through 2.5.2 (pages 169–200)

Homework:

Abelson & Sussman, exercises 2.74, 2.75, 2.76, 2.77, 2.79, 2.80, 2.81, 2.83

Note: Some of these are thought-exercises; you needn't actually run any Scheme programs for them! (Some don't ask you to write procedures at all; others ask for modifications to a program that isn't online.)

• If you haven't already finished this week's lab exercises that involve the `scheme-1` interpreter, do it now. Then write a `map` primitive for `scheme-1` (call it `map-1` so you and Scheme don't get confused about which is which) that works correctly for all mapped procedures.

• Modify the `scheme-1` interpreter to add the `let` special form. Hint: Like a procedure call, `let` will have to use `substitute` to replace certain variables with their values. Don't forget to evaluate the expressions that provide those values!

Extra for experts:

Another approach to the problem of type-handling is *type inference*. If, for instance, a procedure includes the expression `(+ n k)`, one can infer that `n` and `k` have numeric values. Similarly, the expression `(f a b)` indicates that the value of `f` is a procedure.

Write a procedure called `inferred-types` that, given a definition of a Scheme procedure as argument, returns a list of information about the parameters of the procedure. The information list should contain one element per parameter; each element should be a two-element list whose first element is the parameter name and whose second element is a word indicating the type inferred for the parameter. Possible types are listed on the next page.

Continued on next page.

Week 6 continued...

`?`	(the type can't be inferred)
`procedure`	(the parameter appeared as the first word in an unquoted expression or as the first argument of `map` or `every`)
`number`	(the parameter appeared as an argument of `+`, `-`, `max`, or `min`)
`list`	(the parameter appeared as an argument of `append` or as the second argument of `map` or `member`)
`sentence-or-word`	(the parameter appeared as an argument of `first, butfirst, sentence,` or `member?`, or as the second argument of `every`)
`x`	(conflicting types were inferred)

You should assume for this problem that the body of the procedure to be examined does not contain any occurrences of `if` or `cond`, although it may contain arbitrarily nested and quoted expressions. (A more ambitious inference procedure both would examine a more comprehensive set of procedures and could infer conditions like "nonempty list".)

Here's an example of what your inference procedure should return.

```
(inferred-types
  '(define (foo a b c d e f)
    (f (append (a b) c '(b c)) (+ 5 d) (sentence (first e) f)) ) )
```

should return

```
((a procedure) (b ?) (c list) (d number)
 (e sentence-or-word) (f x))
```

If you're *really* ambitious, you could maintain a database of inferred argument types and use it when a procedure you've seen is invoked by another procedure you're examining!

Unix feature of the week: `du`, `df`, `quota`

Emacs feature of the week: `M-q` (format paragraphs), `C-M-q` (format Scheme code)