SICP section 4.2.3

December 27, 2007 at 06:44 Tags SICP

Exercise 4.32

As the authors themselves note, the lazy `car` permits us to implicitly delay both elements of a cons cell. This allows us to construct arbitrary data structures from lists that are completely lazy. One good example is trees.

Recall why lazyness is useful in the first place. One of its chief advantages is the ability to seamlessly handle infinite data structures. So having a lazy `car` as well as a lazy `cdr` will allows us to manipulate infinite trees without worrying. Infinite trees can be useful in artificial intelligence, for instance, where they represent all possible positions a game can reach – and this is used to evaluate potential moves in games like chess.

Exercise 4.33

First of all, let's recall that our evaluator doesn't know how to handle the quote symbo`'` at the moment. In a real Lisp interpreter, the `'` symbol is translated by the *reader*[1] to `(quote ...)`, which is what the evaluator sees. We haven't yet implemented a reader for our evaluator, so we'll have to stick to an explicit `(quote ...)`.

Therefore, instead of:

```
(car '(a b c))
```

We'll be dealin with:

```
(car (quote (a b c)))
```

Also, up until now we've had `cons`, `car` and `cdr` "borrowed" from the host language as primitive procedures. Now we'll implement them, as shown:

```
(interpret
  '(define (cons x y)
    (lambda (m) (m x y))))

(interpret
  '(define (car z)
    (z (lambda (p q) p))))

(interpret
  '(define (cdr z)
    (z (lambda (p q) q))))
```

And replace `text-of-quotation` with:

```
(defun text-of-quotation (exp env)
  (let ((quote-operand (cadr exp)))
    (if (consp quote-operand)
      (eval. (make-lazy-list (cadr exp)) env)
      quote-operand)))

(defun make-lazy-list (elems)
  (if (null elems)
    '()
    (list
      'cons
      (car elems)
      (make-lazy-list (cdr elems)))))
```

Note that now `env` must be passed to `text-of-quotation`. For this reason, the main dispatching `cond` in `eval.` must be slightly modified.

Exercise 4.34

Here's an outline of the solution:

1. At the moment, lazy lists are applications of `cons`. We have to mark them specially, some kind of lazy-list-application.

2. When the printer meets an expression of type lazy-list-application, it follows the list links for some pre-specified amount of elements, say 20, and prints them out. After the 20th element, it prints … to avoid trying to print infinite lists.

[1] The first stage of the read-eval-print loop (REPL).

For comments, please send me an email.

Back to top