SICP section 1.2.2

June 28, 2007 at 05:09 Tags SICP

Section 1.2.2

This section presents *tree recursion*, with Fibonacci numbers as the example. The authors also show how to transform the recursive Fibonacci computation into an iterative one by designing the state variables needed to be passed from call to call. There's not much to discuss here, but let's see the solutions to the exercises.

Exercise 1.11

First, lets translate the definition into CL code:

```
(defun Fr (n)
  (cond ((< n 3) n)
        (t (+ (Fr (- n 1))
              (* 2 (Fr (- n 2)))
              (* 3 (Fr (- n 3)))))))
```

Now, lets apply a transformation similar to what the authors did with Fibonacci. The key idea here is to keep as much state variables as we have recursive calls, and use them as a *queue*, with the first presenting the most recent computation, and so on:

```
a <- a + 2b + 3c (this is F(n+1) - the result)
b <- a (F(n) - last result)
c <- b (F(n-1) - one before last result)
```

To comply with the definition of `F`, `a, b, c` will be initialized to `2, 1, 0` respectively. Now we can write the code:

```
(defun F-iter (n)
  (if (< n 3)
    n
    (F-iter-aux 2 1 0 n)))

(defun F-iter-aux (a b c count)
  (if (= count 2)
    a
    (F-iter-aux (+ a (* 2 b) (* 3 c))
                a
                b
                (- count 1))))
```

This technique actually makes a lot of sense if you think of how you'd solve the problem in another programming language using a loop. You would have to keep the *state variables* for `F(n-1), F(n-2), F(n-3)` explicitly and update them during the loop. Instead, in this tail-recursive Lisp solution, the state variables are passed along as function arguments, but in fact serve the same purpose.

Exercise 1.12

If we design a function that returns the number in the Pascal triangle on some row `row` and column `col`, the code is straightforward (rows and columns are numbered from 1):

```
(defun pascal (row col)
  (cond ((= row 1) 1)
        ((= row col) 1)
        ((= col 1) 1)
        (t (+ (pascal (1- row) col)
              (pascal (1- row) (1- col))))))
```

Note that this function will return incorrect results for nonexistent cells of the Pascal triangle. I think it is safe to assume that the function that calls `pascal` in order to actually draw the triangle won't pass in invalid values.

Exercise 1.13

I will post the proof using pureASCII to ensure viewability on all browsers. Also, I will define qr5 as the square root of 5, which appears frequently in the proof.

```
fi = (1 + qr5)/2
psi = (1 - qr5)/2

As the hint suggests, I will try to prove
the lemma:

(*) Fib(n) = (fi^n - psi^n)/qr5

To prove by induction, I'll first prove
the base cases:

Fib(0) = (fi^0 - psi^0)/qr5 = 0
Fib(1) = (fi^1 - psi^1)/qr5
  fi - psi = qr5 =>
  Fib(1) = qr5/qr5 = 1

Done. Now the induction step. Assuming
that the lemma (*) is true for n-1 and
n-2, I will prove that it's true for n:

Fib(n) = Fib(n-1) + Fib(n-2) =
(fi^(n-1) - psi^(n-1) + fi^(n-2) - psi^(n-2)) / qr5 =
((fi+1)*fi^(n-2) - (psi+1)*psi(n-2)) / qr5

But fi is the renown Golden Ratio, so:

fi^2 = fi+1

If you check with a calculator, you'll
see that the same can be said of psi:

(2) psi^2 = psi+1

So if we substitute fi+1 and psi+1:

(fi^2*fi^(n-2) - psi^2*psi^(n-2)) / qr5 =
(fi^n - psi^n) / qr5
Q.E.D. hint

Now, to use this hint for the final proof,
just notice that psi = -0.618, so when n
becomes large, psi becomes negligible, and
so Fib(n) can be approximated well by:

fi^n/qr5
Q.E.D.
```