Section 2.4 (2.4/2.5): Recursion and Recurrence Relations

September 17, 2007

Abstract

In this section we examine multiple applications of recursive definition, and encounter many examples. Recurrence relations are defined recursively, and solutions can sometimes be given in "closed-form" (that is, without recourse to the recursive definition). We will solve one type of linear recurrence relation to give a general closed-form solution, the solution being verified by induction.

1 Recursion

A **recursive definition** is one in which

1. A basis case (or cases) is given, and

2. an inductive or recursive step describes how to generate additional cases from known ones.

Example: the Factorial function sequence:

1. $F(0) = 1$, and

2. $F(n) = nF(n-1)$.

Note: This method of defining the Factorial function obviates the need to "explain" the fact that $F(0) = 0! = 1$. For that reason, it's better than defining the Factorial function as "the product of the first n positive integers," which it is from $n = 1$ on....

In this section we encounter examples of several different objects which are defined recursively (See Table 2.5, p. 131/139):

- **sequences** – an enumerated list of objects (e.g. Fibonacci numbers - Practice 12, p. 122/130 - history, #32/34, p. 142/143)

  I'm very fond of lisp:

  ```
  (defun fib(n)
    (case n
          (0 1)
          (1 1)
          (t (+ (fib (- n 1)) (fib (- n 2))))
          )
    )
  > (fib 4)
  5
  > (mapcar #'fib (iseq 0 8))
  (1 1 2 3 5 8 13 21 34)
  ```

  Note: The differences in examples #31 and #32 illustrate why you want to stop and think before you attempt a proof!

- **sets** (e.g. finite length and palindromic strings - Example 34 and Practice 16 and 17, pp. 124-125/133)

13
-8
5
-3
2
-1
1
0
1
1
2
3
5
8
13
21
34

- **operations** (e.g. string concatenation - Practice 18, p. 126/134)

- **algorithms** (e.g. BinarySearch - Practice 20, p. 131/139; check out Example #41, p. 130/139, for the definition of "middle".)

  Or my favorites, such as unix shell scripts. Here's one one might call "recurse", for applying an operations to all "ordinary" files:

```
#!/bin/sh
command=$1
files=`ls`
for i in $files
do
        if test -d $i
        then
            cd $i
            directory=`pwd`
            echo "changing directory to $directory..."
            recurse "$command"
            cd ..
        elif test -h $i
        then
            echo $i is a symbolic link: unchanged
else
            $command $i
        fi
done
```

2 Solving Recurrence Relations

Vocabulary:

- **linear recurrence relation:** $S(n)$ depends linearly on previous $S(r)$, $r < n$:

$$S(n) = f_1(n)S(n-1) + \cdots + f_k(n)S(n-k) + g(n)$$

The relation is called **homogeneous** if $g(n) = 0$. (Both Fibonacci and factorial are examples of homogeneous linear recurrence relations.)

- **first-order:** $S(n)$ depends only on $S(n-1)$, and not previous terms. (Factorial is first-order, while Fibonacci is second-order, depending on the two previous terms.)

- **constant coefficient:** In the linear recurrence relation, when the coefficients of previous terms are constants. (Fibonacci is constant coefficient; factorial is not.)

- **closed-form solution**: $S(n)$ is given by a formula which is simply a function of n, rather than a recursive definition of itself. (Both Fibonacci and factorial have closed-form solutions.)

The author suggests an "expand, guess, verify" method for solving recurrence relations.

Example: The story of T

1. Practice 11, p. 121/130

$T(1) = 1$

$T(n) = T(n-1) + 3 \quad n \geq 2$

$\{1, 4, 7, 10, 13, \ldots\}$

2. Practice 19, p. 128/137: Here is the recurrence relation for Example 11, p. 121/130, in lisp:

```
(defun Tee(n)
  (if (integerp n)
      (cond
       ((>= n 2)
        (+ (Tee (- n 1)) 3)
        )
       ((= n 1)
        1
        )
       (t (print "Tilt! Only positive ints allowed..."))
       )
    (print "Tilt! Only positive ints allowed...")
    )
  )
> (tee 2)
4
> (mapcar #'tee (iseq 1 10))
(1 4 7 10 13 16 19 22 25 28)
```

3. Practice 21, p. 133/148

$T(n) = 3n - 2$

$T(1) = 1$ ✓

$\quad 2 = 4$

Prove the closed form soln. by induction

Anchor ✓

$\Rightarrow$: $P(k) \to P(k+1)$

$P(k)$: $T(k) = 3k - 2$

$P(k+1)$: $T(k+1) = 3(k+1) - 2$

$T(k+1) = T(k) + 3 = 3k - 2 + 3 = 3(k+1) - 2$ ✓

Example: general linear first-order recurrence relations with constant coefficients.

$$\boxed{\begin{aligned} &S(1) = a \\ &S(n) = cS(n-1) + g(n) \end{aligned}}$$

"Expand, guess, verify" (then prove by induction!):

$$S(n) = c^{n-1}S(1) + \sum_{i=2}^{n} c^{n-i}g(i)$$

$S(1) = a$

$S(2) = cS(1) + g(2) = ca + g(2)$

$S(3) = cS(2) + g(3)$

$= c[ca + g(2)] + g(3)$

$S(4) = cS(3) + g(4)$

$= c[c[ca + g(2)] + g(3)] + g(4)$

$= c^3 a + c^2 g(2) + cg(3) + g(4)$

$\vdots$

$S(n) = c^{n-1}a + c^{n-2}g(2) + c^{n-3}g(3) + \dots + g(n)$

$$\boxed{S(n) = c^{n-1}a + \sum_{i=2}^{n} c^{n-i} g(i)}$$

Demonstration by induction:

Base: $S(1) = c^0 a$

$S(2) = ca + g(2)$ ✓

$\Rightarrow$ Assume $P(k)$

$S(k) =$

Show $P(k+1)$

$S(k+1) = c^{(k+1)-1}a + \sum_{i=2}^{k+1} c^{(k+1)-i} g(i)$

$$S(k+1) = c\,S(k) + g(k+1)$$

$$= c\left[c^{k-1}a + \sum_{i=2}^{k} c^{k-i} g(i)\right] + g(k+1)$$

$$= c^{(k+1)-1}a + \sum_{i=2}^{k} c^{(k+1)-i} g(i) + g(k+1)$$

$$= c^{(k+1)-1}a + \sum_{i=2}^{k+1} c^{(k+1)-i} \underline{\underline{g(i)}}$$

$$T(n) = 1 \cdot T(n-1) + 3$$

$$T(1) = 1$$

$$c = 1, \quad g(n) = 3, \quad a = 1$$

$$T(n) = 1 + \sum_{i=2}^{n} 1^{n-i}\, 3$$

$$= 1 + \sum_{i=2}^{n} 3$$

$$= 1 + (n-1)\,3$$

$$= 3n + 1 - 3 = 3n - 2$$