PRIVE

Evaluation of numbers written in the Fibonacci system

Maarten M. Fokkinga

1979-07-19

Abstract

We give a very simple algorithm which evaluates a number written in the Fibonacci system from left to right according to Horner's scheme. Both an a priori mathematical problem analysis and a direct algorithmic construction are given. Finally we investigate the generalization for arbitrary recurrence relations.

Acknowledgement

The problem has been suggested by Theo van der Genugten.

1. The problem statement

The Fibonacci sequence is defined by

(0) $F_0 = \ldots,\ F_1 = \ldots,$ and for $j \geq 0$ $F_{j+2} = F_j + F_{j+1}$.

Let a possibly empty sequence $a_0\ a_1\ a_2 \ldots a_{n-1}$ $(n \geq 0)$ be given.

It is requested to determine the value

R: $w = a_0*F_{n-1} + \ldots + a_{n-1}*F_0$

with the constraint that the sequence a may only be scanned, from left to right, once. Hence the value of n might be determined implicitly and need not be known before a_{n-1} has been scanned.

2. An a priori mathematical analysis

We realize that half-way the computational proces we will have computed the value

(1) $w_j = a_0*F_{j-1} + .. + a_{j-1}*F_0$

for some j: 0..n . Indeed, when $j = n$ the value w_j equals the requested value w ; further, no use is made of the value of n and the sequence a has been scanned from left to right. We try to set up a recurrence relation for w_j :

w_{j+1} = (from(1):) $a_0*F_j + .. + a_j*F_0$
$= (a_0*F_j + .. + a_{j-1}*F_1) + a_j*F_0$
$= v_j + a_j*F_0$

provided we define the entity v_j , involving the sequence scanned so far, as follows:

(2) $v_j = a_0*F_j + .. + a_{j-1}*F_1$.

Now we need to express v_{j+1} recurrently too:

v_{j+1} = (from(2):) $a_0*F_{j+1} + .. + a_j*F_1$
$= (a_0*F_{j+1} + .. + a_{j-1}*F_2) + a_j*F_1$
= (from (0):) $(a_0*(F_{j-1}+F_j) + .. + a_{j-1}*(F_0+F_1)) + a_j*F_1$
$= (a_0*F_{j-1} + .. + a_{j-1}*F_0) + (a_0*F_j + .. + a_{j-1}*F_1) + a_j*F_1$
= (from (2,3):) $w_j + v_j + a_j*F_1$.

Fortunately, we are through!

The program now is a simple repetition. The invariant relation reads

P: $0 \leq j \leq n$ **and** $w = w_j$ **and** $v = v_j$.

The program reads

```
j, w, v := 0, 0, 0;
do j≠n → j, w, v := j+1, v+a_j*F_0, w+v+a_j*F_1 od .
```

3. A direct algorithmic construction

We try to establish relation R by means of a repetition. To this end we derive the invariant relation from R by "replacing a constant by a variable" (the standard approach!). We choose to replace (all!) occurrences of n by a variable j :

P0: $w = a_0*F_{j-1} + .. + a_{j-1}*F_0$ and $0 \le j \le n$.

(The second term has been introduced to restrict the range of j). The program should then read

j, w := 0, 0; {P0}

do $j \ne n \rightarrow$ j, w := j+1, "new w" od {R} .

In order to know how to refine "new w" we compute

wp("j,w := j+1, "new w"", P0) =

= "new w" = $a_0*F_{j+1-1} + .. + a_{j+1-1}*F_0$ and $0 \le j+1 \le n$

= "new w" = $(a_0*F_j + .. + a_{j-1}*F_1) + a_j*F_0$ and $0 \le j+1 \le n$.

This is true, on account of P0 , provided we refine

"new w" : $v + a_j*F_0$

and we establish before the assignment the relation

P1: $v = a_0*F_j + .. + a_{j-1}*F_1$.

However, instead of establishing P1 inside the repetition from scratch, we may as well take relation P1 outside the repetition:

j, w, v := 0, 0, 0; {P0 and P1}

do $j \ne n \rightarrow$ j, w := j+1, $v+a_j*F_0$; "reestablish P1" od .

For convenience -- as appeared in earlier drafts of this paper -- we will reestablish P1 simultaneously with the assignment to j and w :

j, w, v := 0, 0, 0;

do $j \ne n \rightarrow$ j, w, v := j+1, $v+a_j*F_0$, "new v" od .

In order to know how to refine "new v"" , P1) =

wp("j,w,v := j+1, $v+a_j*F_0$, "new v", P1) =

= "new v" = $a_0*F_{j+1} + .. + a_{j+1-1}*F_1$

= "new v" = $(a_0*(F_{j-1}+F_j) + .. + a_{j-1}*(F_0+F_1)) + a_j*F_1$

= "new v" = $(a_0*F_{j-1} + .. + a_{j-1}*F_0) + (a_0*F_j + .. + a_{j-1}*F_1) + a_j*F_1$

= "new v" = $w + v + a_j*F_1$.

The transition to the last line is valid on account of P0 and P1 immediately before the assignment. Hence we may refine

"new v" : $w + v + a_j*F_1$.

Herewith the program has been finished:

j, w, v := 0, 0, 0;

do $j \ne n \rightarrow$ j, w, v := j+1, $v+a_j*F_0$, $w+v+a_j*F_1$ od .

Remark. The direct algorithmic construction shows exactly the same reasoning as the mathematical analysis. (End of remark.)

3. Generalization

Let $k \geq 1$ and let f be a function such that

(0) for some constants $c_0,..,c_{k-1}$

$$f(x_0,..,x_{k-1}) = c_0*x_0 + .. + c_{k-1}*x_{k-1} \quad .$$

Let S be a recurrent sequence, defined by means of f :

(1) $S_0,..,S_{k-1}$ are given,

$$S_{j+k} = f(S_j,..,S_{j+k-1}) \text{ for } j \geq 0 \quad .$$

We give a simple and efficient algorithm to determine

R: $w = a_0*S_{n-1} + .. + a_{n-1}*S_0$

for arbitrary sequence $a_0 .. a_{n-1}$ $(n \geq 0)$, which scans the sequence from left to right (and doesn't use the value of n before a_{n-1} has been scanned). We also show that (0) is a complete characterization for all functions f for which the algorithm is correct.

Notation. "Si: m..n. ti" means: the sum of all terms ti in which i ranges over m..n .

Define k sequences $w_0,..,w_{k-1}$ as follows.

(2) For each j , $0 \leq j \leq n$,

$$w_{0,j} = \underline{S}i\text{: } 0..j-1.\ a_i*S_{j-1-i} \quad ,$$
$$w_{1,j} = \underline{S}i\text{: } 0..j-1.\ a_i*S_{j-1-i+1} \quad ,$$
$$\vdots$$
$$w_{k-1,j} = \underline{S}i\text{: } 0..j-1.\ a_i*S_{j-1-i+k-2} \quad ,$$
$$w_{k-1,j} = \underline{S}i\text{: } 0..j-1.\ a_i*S_{j-1-i+k-1} \quad .$$

The required value w equals $w_{0,n}$. It appears that $w_{0,j+1},..,w_{k-1,j+1}$ can be expressed recurrently in their predecessors $w_{0,j},..,w_{k-1,j}$ as follows.

(3)
$$w_{0,j+1} = a_j*S_0 + w_{1,j} \quad ,$$
$$w_{1,j+1} = a_j*S_1 + w_{2,j} \quad ,$$
$$\vdots$$
$$w_{k-2,j+1} = a_j*S_{k-2} + w_{k-1,j} \quad ,$$
$$w_{k-1,j+1} = a_j*S_{k-1} + f(w_{0,j},..,w_{k-1,j}) \quad .$$

The first $k-1$ equalities follow directly from (2); the last equality exploits (0) and (1) as well.

Thanks to the recurrence relation a single repetition suffices. The invariant relation reads

$0 \le j \le n$ and $w_0 = w_{0,j}$ and .. and $w_{k-1} = w_{k-1,j}$.

The program reads

$j, w_0, \ldots, w_{k-1} := 0, 0, .., 0;$
do $j \ne n \rightarrow$
$j, w_0, .., w_{k-1} := j+1, a_j * S_0 + w_1, \ldots, a_j * S_{k-1} + f(w_0, .., w_{k-1})$
od .

Applications.

The unary, binary and decimal system are instances of the general case, viz.

a. $S_0 = 1$ and $f(x_0) = x_0$: unary system,
b. $S_0 = 1$, and $f(x_0) = 2 * x_0$: binary system,
c. $S_0 = 1$, and $f(x_0) = 10 * x_0$: decimal system.

However the above algorithm allows the "digits" a_j to be of unbounded value. Note also that case a is the standard summation $a_0 + .. + a_{n-1}$.

Another example is the Fibonacci sequence

d. $S_0 = 0$, $S_1 = 1$ and $f(x_0, x_1) = x_0 + x_1$.

For all these cases the algorithm is the most efficient one (measured in the number of additions and multiplications): it is an implementation of Horner's scheme!

Completeness of requirement (0).

In the last line of the proof of the recurrence relation (3), it appears that a sufficient and necessary condition for f reads

(4) $\underline{S}j.\ a_j * f(x_{0,j}, .., x_{k-1,j}) =$
$f((\underline{S}_j . a_j * x_{0,j}), \ldots, (\underline{S}_j .\ a_j * x_{k-1,j}))$.

The requirement (0) is equivalent with (4), and so it is a complete characterization of all f for which the algorithm is correct. The implication (4) $\Longrightarrow$ (0) is easy; for the converse (0) $\Longrightarrow$ (4) we argue as follows,

For arbitrary $x_0, .., x_{k-1}$ define

(5) $a_j = x_j$ for $0 \le j \le k-1$,

(6) $x_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \ne j \end{cases}$ for $0 \le j \le k-1$.

Then we find

$$
\begin{aligned}
& f(x_0,..,x_{k-1}) = \\
&= \text{(from (5,6):) } f((\underline{S}j\colon 0..k-1.\ a_j*x_{0,j}),\ ..\ ,(\underline{S}j\colon 0..k-1.\ a_j*x_{k-1,j})) \\
&= \text{(from (4):) } \underline{S}j\colon 0..k-1.\ f(a_j*x_{0,j},\ ..\ ,a_j*x_{k-1,j}) \\
&= \text{(from (5,6):) } x_0*f(1,0,..,0) + .. + x_{k-1}*f(0..,0,1)
\end{aligned}
$$

so we may choose $c_0 = f(1,0..,0),..., c_{k-1} = f(0..,0,1)$ and we see that (0) holds as well.

<u>Note, added later</u> Joost Engelfriet pointed my attention to the following recurrence relation for the w_j:

$$
\begin{aligned}
w_{j+2} &= (a_0*F_{j+1} + \;--\; + a_{j-1}*F_2) + (a_j*F_1 + a_{j+1}*F_0) \\
&= (a_0*F_j + \;--\; + a_{j-1}*F_1) + (a_j*F_0 - a_j*F_0) + \\
&\quad (a_0*F_{j-1} + \;--\; + a_{j-1}*F_0) + (a_j*F_1 + a_{j+1}*F_0) \\
&= w_j + w_{j+1} + (a_j*F_1 + a_{j+1}*F_0 - a_j*F_0)
\end{aligned}
$$

with $w_0 = 0$, $w_1 = a_0*F_0$.

So another program reads

$j, a, wo, w1 := 0, a_0, 0, a_0*F_0$;

$\underline{do}$ $j \neq n \rightarrow j, a, wo, w1 := j+1, a_j, w1, wo + w1 + a*F_1 + a_j*F_0 - a*F_0$ $\underline{od}$,

with invariant relation

$wo = w_j$ $\underline{and}$ $w1 = w_{j+1}$ $\underline{and}$ $a = a_j$ $\underline{and}$ $0 \le j \le n$.