In honour of Fibonacci.

Studying an artificial intelligence approach to programming the other day -- I read the most weird documents! -- I was reminded of the Fibonacci sequence, given by

\[ F_1 = 0, \ F_2 = 1, \]
\[ F_n = F_{n-1} + F_{n-2} \quad (-\infty < n < +\infty). \]

For \( N \geq 2 \) the relation

\[ R: \quad x = F_N \]

is trivially established by the program

\[ y, x, i := 0, 1, 2 \ \{ y = F_{i-1} \text{ and } x = F_i \text{ and } 2 \leq i \leq N \}; \]
\[ \text{do } i \neq N \rightarrow y, x, i := x, x + y, i + 1 \text{ od \{R\}} \]

a program with a time-complexity proportional to \( N \); I remembered -- although I did not know the formulae -- that \( R \) can also be established in a number of operations proportional to \( \log(N) \) and wondered -- as a matter of fact: I still wonder -- how proponents of "program transformations" propose to transform the linear algorithm (1) into the logarithmic one.

Yesterday evening I was wondering whether I could reconstruct the logarithmic scheme for the Fibonacci sequence, and whether similar schemes existed for higher order recurrence relations (for \( a, k \geq 2 \)):

\[ F_1 = F_2 = \ldots = F_{k-1} = 0, \ F_k = 1 \]
\[ F_n = F_{n-1} + \ldots + F_{n-k} \quad (-\infty < n < +\infty) \]

Eventually I found a way of deriving these schemes. For \( k = 2 \), the normal Fibonacci numbers, the method leads to the well-known formulae

\[ F_{2j} = F_j^2 + F_{j+1}^2 \]
\[ F_{2j+1} = (2F_j + F_{j+1}) \cdot F_{j+1} \quad \text{or} \quad F_{2j+1} = (2F_{j+1} - F_j) \cdot F_j. \]

This note is written, because I liked my general derivation. I shall describe it for \( k = 3 \).
Because for \( k = 3 \) we have \( F_1 = F_2 = 0 \) and \( F_3 = 1 \) we may write
\[
F_n = F_3 * F_n + (F_2 + F_1) * F_{n-1} + F_2 * F_{n-2}
\] (3).

From (3) we deduce the truth of
\[
F_n = F_{i+3} * F_{n-i} + (F_{i+2} + F_{i+1}) * F_{n-i-1} + F_{i+2} * F_{n-i-2}
\] (4)
for \( i = 0 \). The truth of (4) for all positive values of \( i \) is derived by mathematical induction; the induction step consists of

1) substituting \( F_{n-i-1} + F_{n-i-2} + F_{n-i-3} \) for \( F_{n-i} \)

2) combining after rearrangement \( F_{i+3} + F_{i+2} + F_{i+1} \) into \( F_{i+4} \).

(The proof for negative values of \( i \) is done by performing the induction step the other way round.)

Substituting in (4) \( n = 2j \) and \( i = j-1 \) we get
\[
F_{2j} = F_{j+2} * F_{j+1} + (F_{j+1} + F_j) * F_j + F_{j+1} * F_{j-1}
\]
and, by substituting \( F_{j+2} = F_{j+1} - F_j \) for \( F_{j-1} \), and subsequent rearranging
\[
F_{2j} = F_j^2 + (2F_{j+2} - F_{j+1}) * F_{j+1}
\] (5)

Substituting in (4) \( n = 2j+1 \) and \( i = j-1 \) we get
\[
F_{2j+1} = F_{j+2}^2 + (F_{j+1} + F_j) * F_{j+1} + F_{j+1} * F_j
= (2F_j + F_{j+1}) * F_{j+1} + F_{j+2}
\] (6)

Formulas (5) and (6) were the ones I was after.

Note. For \( k = 4 \) the analogue to (4) is
\[
F_n = F_{i+4} * F_{n-i} + (F_{i+3} + F_{i+2} + F_{i+1}) * F_{n-i-1} +
(F_{i+3} + F_{i+2}) * F_{n-i-2} + F_{i+3} * F_{n-i-3}
\]
(End of note.)

Plataanstraat 5
5671 AL Nuenen
The Netherlands

prof.dr.Edsger W.Dijkstra
Burroughs Research Fellow