---

16 August 1976 EWD 578

More about the function *fusc*. (A sequel to EWD570)

In EWD570 I introduced the function *fusc*, given by

$fusc(1) = 1, \quad fusc(2n) = fusc(n), \quad fusc(2n+1) = fusc(n) + fusc(n+1)$.

Compatible with the second part of that definition we derive from the third part $fusc(0) = 0$. I showed there the following iterative program for the computation of $fusc(N)$ —with *peven* and *podd* standing for "positive and even" and "positive and odd" respectively—

$n, a, b := N, 1, 0;$
do $peven(n) \rightarrow a, n := a + b, n / 2$
□ $podd(n) \rightarrow b, n := b + a, (n\text{-}1) / 2$
od $\{fusc(N) = b\}$

On my last trip through the USA, while lecturing to a Burroughs audience, my audience derived this program after it had decided —after only a few very modest hints!— that a good candidate for an invariant relation would be

P: $fusc(N) = a*fusc(n) + b*fusc(n+1)$

The audience arrived at this suggestion after a few simple considerations. The first observation was that

$fusc(N) = fusc(n)$

would be simple to initialize by means of $n := N$. They quickly saw that this was too simple, and considered

$fusc(N) = a*fusc(n)$

equally trivially initialized by $n, a := N, 1$; it was then remarked that initialization would not be complicated by an additive term

$fusc(N) = a*fusc(n) + b$

as that is initialized by $n, a, b := N, 1, 0$. The observation that for $n = 0$ the first term would disappear but that $fusc(n+1) = 1$ would then hold suggested, together with the third part of the definition for *fusc* the fully blown-up P as given above. Separating the cases

$$\begin{aligned} n = 2k: \quad fusc(N) &= a*fusc(n) + b*fusc(n+1) \\ &= a*fusc(2k) + b*fusc(2k+1) \\ &= (a+b)*fusc(k) + b*fusc(k+1) \end{aligned}$$

$$\begin{aligned} n = 2k+1: \quad fusc(N) &= a*fusc(n) + b*fusc(n+1) \\ &= a*fusc(2k+1) + b*fusc(2k+2) \\ &= a*fusc(k) + (a+b)*fusc(k+1) \end{aligned}$$

my audience quickly derived —to its pleasant surprise!— the iterative program given above.

* *
*

From the above program, two properties of the function *fusc* follow. The first one is that the value of the function *fusc* does not change if we invert in the binary representation of the argument all "internal" digits, i.e. all the binary digits between the most- and the least-significant ones. For instance $fusc(19) = fusc(29)$ because, in binary 19 and 29 are 10011 and 11101 respectively. This follows from the comparison of the a,b-pairs during those two computations. After the processing of the least significant digit of the arguments, both have $a,b = 1,1$. As a result of the inverted internal digits, the one computation has the role of a and b interchanged with respect to the other computation. Because the sum of the two values is a symmetric function of its arguments and, as a result of the last —i.e. most-significant— 1 in the argument that sum of a and b is delivered (in b) as the final value, both computations deliver the same result.

The next property is more surprising. (At least, I think so.) Let us try to represent the pair a,b by the single value m , according to the convention

$$a = fusc(m+1) \qquad b = fusc(m)$$

In the case of $peven(n)$ the operation on a,b has the form $a, b := a+b, b$ or:

$$\begin{aligned} fusc(m+1), fusc(m) &:= fusc(m+1)+fusc(m), fusc(m) \\ &:= fusc(2m+1), fusc(2m) \end{aligned}$$

an operation that translates into $m := 2m$. Similarly $a, b := a, a+b$ translates into $m := 2m+1$. Substituting all this in our iterative program we get

$n, m := N, 0;$
do $peven(n) \rightarrow m, n := 2*m, n/2$
 □ $podd(n) \rightarrow m, n := 2*m+1, (n-1)/2$
od $\{fusc(N) = fusc(m)\}$

i.e. the *fusc*-value does not change if we write the binary digits of the argument in the reverse order. For example $fusc(19) = fusc(25)$ because 19 and 25 are in binary 10011 and 11001 respectively. I think this second property more surprising!

* *
*

In a way which does not admit generalization I discovered the equivalence

$$2|fusc(n) <=> 3|n$$

i.e. $fusc(n)$ is even iff n is a multiple of 3. Inspired by a recent exercise of Don Knuth I tried to characterize the arguments n such that $3|fusc(n)$.

With braces used to denote zero or more instances of the enclosed, the vertical bar as the BNF "or", and the question mark "?" to denote either a 0 or a 1 , the syntactical representation for such an argument (in binary) is

{0}1{?0{1}0|?1{0}1}?1{0}

I derived this by considering —as a direct derivation of my program— the finite state automaton that computes $fusc(N)$|**mod** 3 . It was the first time of my life that I did what others have done many times before, i.e. relating a finite state automaton to a grammar. The exercise is up till now only of modest interest; it taught me that division by a fixed factor and (simple!) syntactic analysis are processes that are very closely related to each other, and that insight I think somehow illuminating.

* *
*

Since the distribution of EW570 it has been discovered that more mathematicians have occupied themselves with the function *fusc* —they only gave it a different name!— a fact that is not surprising in view of its properties. J.J.Seidel and F.L.Bauer have independently pointed out to me that it is no. 56 in Sloane's *Dictionary of Integer Sequences*, that refers to an article by G.de Rham, *Elemente der Mathematik*, Vol.2 (1947) pg.95 . It was fun!

Plataanstraat 5
NL-4565 NUENEN
The Netherlands

prof.dr. Edsger W. Dijkstra
Burroughs Research Fellow

---

transcribed by Corrado Cantelmi
revised 16-Sep-2011