The Theory of Computation

Bernard M.E. Moret

Addison-Wesley-Longman, 1998

Detailed Solution
Chapter 5, Problem 17

If we consider the original busy beaver problem—how long can a program of length n run and still stop, that is clearly undecidable, since otherwise we could easily solve the halting problem: given a program of length n and input y, feed $n + |y|$ to the busy beaver function, get a maximum number of steps and now run the program for at most these many steps on input y: if the program halts, we are done; if it does not, we can legitimately assert that it will not ever halt on y, since we gave it the largest number of steps it could ever require.

The version of the busy beaver problem formulated in terms of output value is equally undecidable: if we could decide it, we could solve a slight variant of the problem of finding the shortest program that prints n and halts (a problem that we showed in the text to be undecidable. The variant simply asks for the length of the shortest program that prints *at least* n and halts—it is clear that our proof in the text applies unchanged to this variant. Our reduction works like this: Given n, the (lower bound on the) number to print, we run a loop on some index i from 1 until success and ask the busy beaver function whether a program of length i can print a number at least as large as n and halt. The search must stop, since each longer program can print a larger number; thus this loop uses a putative solution to the busy beaver problem to provide a solution to the shortest program problem; since finding the shortest program to print (at least) n is undecidable, finding the largest value printed by a program of length n is also undecidable.

We could also use the same strategy we used in proving the shortest program undecidable: a direct contradiction obtained by constructing a new program from the existing one. If we had a routine f that printed the value $f(n)$, the largest number that a program of length n with no input can print and still halt, we could construct the new program g for each fixed m by running the following loop

```
i := 1;
while f(i) < m do i := i+1;
print f(i)
```

This new program prints the smallest value $f(i)$ such that no program of length less than m can print it. Yet our new program g has length $O(\log m)$ and prints $f(i)$, a contradiction, since m grows asymptotically faster than $\log m$. Thus the subroutine f does not exist.