Ackermann's Function in the Number of 1s Generated by Green's Machines

Bryant A. Julstrom
Department of Computer Science
St. Cloud State University
julstrom@eeyore.stcloudstate.edu

Abstract

The Busy Beaver problem, defined by Radó in 1962, seeks the maximum number $\Sigma(n)$ of 1s that an n-state Turing machine may leave, if and when it halts, on an initially blank tape [8]. In 1964, Green described a class of Turing machines, the larger built from the smaller, that generate long blocks of 1s and thus set lower bounds for values of $\Sigma(n)$; he called them Class M machines [3]. Ackermann's function is a computable function that is not primitive recursive and that returns huge values for even moderate arguments. A function $f_{\mathrm{M}}(x, y)$ defined by Green's Class M machines of x states operating on a block of y 1s also returns large values. In particular, the recurrence that describes $f_{\mathrm{M}}(x, y)$ is identical in form to a common presentation of Ackermann's function. They differ only in their base cases and the addition of 1 in the recursive case of $f_{\mathrm{M}}(x, y)$.

Introduction

Consider a Turing machine with n states whose tape is infinite in both directions, whose tape alphabet is $\{0,1\}$, and that writes and shifts on each move, including the transition to the halt state. Starting on a tape whose every cell contains 0, what is the largest number of 1s the machine may leave on the tape when it halts? This is the Busy Beaver problem [8], and the maximum number of 1s, as a function of n, is the Busy Beaver function $\Sigma(n)$.

Green [3] described a collection of machines, which he called Class M machines, that establish lower bounds for $\Sigma(n)$. Beginning with a small machine that halts when started on a tape containing all 0s, he added two new states, which call the original machine like a subroutine. This process can be continued, to build machines of any number of states, and the number of 1s they leave on the tape grows very quickly as the machines get larger.

Ackermann's function is the standard example of a computable function that is not primitive recursive. Students first encounter it as an example of recursion and in the time complexity of efficient union-find operations [4, pp.70-78][10].

Investigators of the Busy Beaver problem have noticed a relationship between the numbers of 1s Green's machines leave on the tape and Ackermann's function. Marxen and Buntrock, for example, described these values as establishing a “non-trivial (not primitive recursive) lower bound” for $\Sigma(n)$ [7]. We make that relationship explicit by identifying a recurrence, nearly identical in form to one that defines Ackermann's function, that returns the number of 1s a Class M machine of x states leaves on its tape when the tape initially contains a block of y 1s.

The following sections of the paper describe the Busy Beaver problem, Ackermann’s function, and Green’s machines, then identify a recurrence that specifies the function that Green’s Class M machines implement and relate this function to Ackermann’s function.

The Busy Beaver Problem

What is the maximum number of 1s that a halting n-state Turing machine can leave on an initially blank tape? This is the Busy Beaver problem, described by Tibor Radó in 1962 [8].

More formally, let M be a Turing machine with n states, not counting the halt state. Its tape is infinite in both directions, its tape alphabet is $\{0,1\}$, and no cell is empty. M both writes and shifts its head on every transition, including the one to the halt state. When such a machine is started on a blank tape—a tape containing all 0s—what is the largest number of 1s that, if it halts, it may leave on the tape? As a function of n, this count is the Busy Beaver function $\Sigma(n)$, and a machine of n states that halts with $\Sigma(n)$ 1s on its tape is an n-state Busy Beaver.

The Busy Beaver function is non-computable [5][8]; indeed, for large enough n, $\Sigma(n)$ grows faster than any computable function. Values of $\Sigma(n)$ have been established for small values of n: $\Sigma(1) = 1$, $\Sigma(2) = 4$ [8], $\Sigma(3) = 6$ [6], and $\Sigma(4) = 13$ [1]. Marxen and Buntrock [7] described a machine that establishes that $\Sigma(5)$ is at least 4,098, and another [2] that fixes a lower bound for $\Sigma(6)$: greater than $1.2*10^{865}$. The latter machine executes more than $3.0*10^{1730}$ moves before it halts. Figure 1 shows a 3-state Busy Beaver; on a transition labeled a/b/d, the machine reads the symbol a, writes the symbol b, and shifts its head one cell in the direction d.

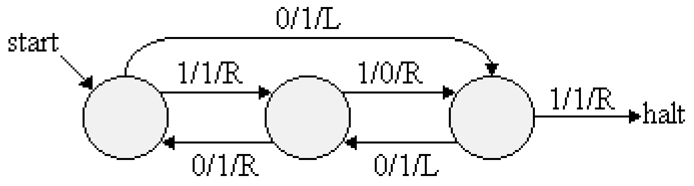

Figure 1: A three-state Busy Beaver. When started on a blank tape, this Turing machine halts and leaves six 1s.

Ackermann's Function

Ackermann's function is probably the most frequently cited example of a computable function that is not primitive recursive (*e.g.*, [9, pp.5-9]). A common presentation of the function is this recurrence:

$$A(x, y) = \begin{cases} y + 1, \text{ if } x = 0 \\ A(x-1, 1), \text{ if } y = 0 \text{ and } x > 0 \\ A(x-1, A(x, y-1)), \text{ otherwise} \end{cases}$$

For small arguments x and y, $A(x,y)$ is itself moderate, as Table 1 illustrates. The table lists the values of Ackermann's function for $x = 0, 1, 2, 3$ and for $y = 0, 1, 2, \ldots, 10$. In particular, we see that $A(0, y) = y + 1$; $A(1, y) = y + 2$; $A(2, y) = 2y + 3$; and $A(3, y) = 8\,(2^y - 1) + 5$.

Table 1: Values $A(x,y)$ of Ackermann's function for small values of x and y.

x	y=0	1	2	3	4	5	6	7	8	9	10
0	1	2	3	4	5	6	7	8	9	10	11
1	2	3	4	5	6	7	8	9	10	11	12
2	3	5	7	9	11	13	15	17	19	21	23
3	5	13	29	61	125	253	509	1021	2045	4093	8189

In general, x determines how fast Ackermann's function grows as a function of y. For $x \geq 4$, that growth is extreme, and conventional notation cannot represent $A(x,y)$ in a closed form.

Similarly, Ackermann's function can be easily implemented in any high-level programming language by translating the recurrence above into a recursive procedure, but executing that procedure for all but the smallest arguments generally fails through overflow of number representations or the run-time stack.

Green's Machines

In 1964, Green described a mechanism for constructing Turing machines that leave large numbers of 1s on initially blank tapes, thus establishing lower bounds for the Busy Beaver function $\Sigma(n)$ for larger values of n [3]. He called these Class M machines; to be in Class M, a machine must satisfy the following conditions:

1. Its input is either a blank tape or a block of consecutive 1s. In the latter case, its head starts over the rightmost 1.
2. The machine computes a function $M(m)$; when it halts, it leaves a contiguous block of 1s.
3. The rightmost 1 of a machine's output $M(m)$ falls on the cell originally occupied by the rightmost 1 of its input.
4. When it moves to the halt state, it reads the 0 to the right of a block of 1s and shifts left.
5. The head never shifts more than one position to the right of the initial rightmost 1.

Figure 2 shows a two-state Class M machine that computes the function $M(m) = m + 1$ by appending a 1 immediately to the left of the 1s initially on the tape.

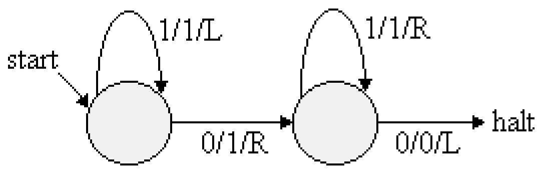

Figure 2: A two-state Class M machine that computes the function $M(m) = m + 1$ [3].

Let M_n be an n-state Class M machine. Figure 3 shows how to augment M_n with two new states to obtain a larger Class M machine M_{n+2}. M_{n+2}'s new states call M_n like a subroutine. When M_{n+2} is started on a tape holding m 1s, it shifts left until it encounters a 0; there it computes $M_n(0)$. It moves a 1 from the input's left end to $M_n(0)$'s right end, and invokes M_n again. It continues until the original input is consumed; the number of 1s left on the tape is then

$$M_{n+2}(m) \;=\; 1 + M_n(1 + M_n(1 + \ldots + M_n(1 + M_n(0)) \ldots)) \;=\; 1 + M_n(M_{n+2}(m-1)).$$

Because M_{n+2} is also a class M machine, we can build M_{n+4} from it, and so on. The next section considers in more detail the functions that Green's Class M machines implement.

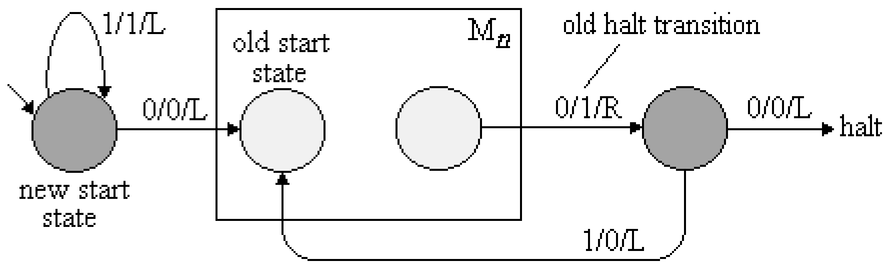

Figure 3: Building a larger Class M machine M_{n+2} by adding two states (green) to a class M machine M_n (yellow) [3].

A Function Defined by Green's Machines

Let M_2 be the Class M machine in Figure 2 that implements the function $M_2(m) = m + 1$. In general, let M_{2k} be the Class M machine with an even number of states built from M_2 by applying Green's construction $(k - 1)$ times. M_{2k} computes a function $M_{2k}(m)$; when started on a tape containing a block of m 1s, M_{2k} halts and leaves $M_{2k}(m)$ 1s on the tape. What can we say about $M_{2k}(m)$?

A Turing machine must have at least one state, so $k \geq 1$. If $k = 1$, M_{2k} is M_2, and $M_{2k}(m) = M_2(m) = m + 1$. If $m = 0$—that is, if M_{2k} is started on a blank tape—the machine immediately executes the transition to the start state of $M_{2(k-1)}$. This machine computes $M_{2(k-1)}(0)$, which includes writing a single 1 on the old halt transition. After this transition, M_{2k} halts, since there are no 1s to the right of the output of a Class M machine. Thus

$$M_{2k}(0) = 1 + M_{2(k-1)}(0) = 1 + 1 + M_{2(k-2)}(0) = \ldots = 1 + 1 + \ldots + M_2(0) = k.$$

Finally, for larger k and m, we have seen in the previous section that

$$M_{2k}(m) = 1 + M_{2(k-1)}(M_{2k}(m-1)).$$

Now let $f_M(x, y) = M_{2x}(y)$, so that

$$f_M(x, y) = \begin{cases} y + 1, \text{ if } x = 1 \\ x, \text{ if } y = 0 \\ 1 + f_M(x-1, f_M(x, y-1)), \text{ otherwise.} \end{cases}$$

Though the base cases are different from that of Ackermann's function, the recurrence is the same, plus one. Table 2 lists the values of $f_M(x, y)$ for small values of x and y.

We see that $f_M(1, y) = y + 1$; $f_M(2, y) = 2y + 2$; and $f_M(3, y) = 3(2^{y+1} - 1)$. As with Ackermann's function, conventional notation cannot represent $f_M(x, y)$ for $x \geq 4$. Most significantly, the recurrence of Ackermann's function re-appears in this function implemented by Turing machines built according to Green's construction.

Table 2: Values of the function $f_M(x,y)$ derived from Green's machines for small values of x and y.

x	y=0	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10	11
2	2	4	6	8	10	12	14	16	18	20	22
3	3	9	21	45	93	189	381	765	1533	3069	6141

Conclusion

Green described the construction of a class of Turing machines designed to leave large blocks of 1s on an initially blank tape. These machines define a function of two variables: the number of 1s left by a Class M machine with x states started on a tape containing a block of y 1s. The recurrence that describes this function is similar in form to a common presentation of Ackermann's function. The two recurrences differ only in their base cases and in one term of the Class M recurrence's recursive case. The latter recurrence demonstrates how Ackermann-like functions can arise naturally from the construction of Turing machines.

References

1. Brady, Allen H. (1983). The determination of the value of Radó's noncomputable function $\Sigma(k)$ for four-state Turing machines. *IEEE Transactions on Electronic Computers*, V.EC-15, pp.802-803.

2. Buntrock, Jürgen and Heinar Marxen (2001). www.drb.insel.de/~heinar/BB/bb-6list .

3. Green, Milton W. (1964). A lower bound on Radó's sigma function for binary Turing machines. In *Switching Circuit Theory and Logical Design: Proceedings of the Fifth Annual Symposium*, pp.91-94.

4. Horowitz, Ellis and Sartaj Sahni (1978). *Fundamentals of Computer Algorithms*. Potomac, MD: Computer Science Press.

5. Julstrom, Bryant A. (1993). Noncomputability and the Busy Beaver problem. *The UMAP Journal*, V.14, no.1, pp.39-74.

6. Lin, Shen and Tibor Radó (1965). Computer studies of turing machine problems. *Journal of the ACM*, V.12, pp.196-212.

7. Marxen, Heiner and Jürgen Buntrock (1990). Attacking the Busy Beaver problem 5. *Bulletin of the European Association for Theoretical Computer Science*, V.40, pp.247-251.

8. Radó, Tibor (1962). On non-computable functions. *Bell System Technical Journal*, V.41, pp.877-884.

9. Rogers, Hartley, Jr. (1987). *Theory of Recursive Functions and Effective Computability*. Cambridge, MA: The MIT Press.

10. Tarjan, Robert (1975). On the efficiency of a good but not linear set merging algorithm. *Journal of the ACM*, V.22, no.2, pp.215-225.