ACCELERATION TECHNIQUES FOR BUSY BEAVER CANDIDATES

Alex Holkner [1]

[1] School of Computer Science and Information Technology,
RMIT University

ABSTRACT: A busy beaver is a Turing machine of N states which, given a blank tape, halts with an equal or higher number of printed symbols than all other machines of N states. Candidates for this class of Turing machine cannot realistically be run on a traditional simulator due to their high demands on memory and time. This paper describes some techniques that can be used to run these machines to completion with limited memory, in a reasonable amount of time, and without human intervention. These techniques include using k-macro tape representation and arcs, and automatic inductive proof generation and utilization.

INTRODUCTION

The busy beaver problem was described by Tibo Rado in 1960 as an example of a non-computable function [Rad62]. A Turing machine of N states is said to be of size N, not counting the halting state. The Turing machines examined in this problem have one tape head and one tape, which is infinite in both directions (left and right). The function $\Sigma(N)$ is defined as the maximum number of 1's a Turing machine of size N will produce before halting given an initially blank tape. Any Turing machine of size N which produces $\Sigma(N)$ 1's is called a busy beaver. Only 5-tuple Turing machines are considered here (that is, machines which can read, write and move the tape head in one step).

Brute force is the only known method one can use to solve for $\Sigma(N)$. There are $(4N+4)^{2N}$ combinatorial N-state Turing machines. Even if only normalised forms are considered as by Machlin and Stout [MS90], the number of machines to check is formidable, and at this time of writing $\Sigma(N)$ has been calculated conclusively only for $N \leq 4$ [Bra83] [LR65].

When enumerating Turing machines in the search for busy beaver candidates, each must be determined to either:

- Not halt. If the machine is caught in some infinite loop it must be proven that it will never halt.
- Halt, with some number of symbols on the tape. Again, this can be done by either running the machine to completion and observing the output, or by proving its output analytically.

The case where the machine does not halt has been the attention of most busy beaver related research. Various attempts have been made at calculating an upper bound on $S(N)$, the number of shifts the N state busy beaver will make. At best we have Julstrom showing $S(N) < \Sigma(20N)$ [Jul92], however this upper bound is just as incomputable as $\Sigma(N)$.

Rado and Lin [LR65] were the first to write computer programs to prove suspect machines would not halt in order to eliminate them from the 3-state candidate list. Brady [Bra83] later expanded on the set of machines that could be automatically proven non-halting in his determination of $\Sigma(4)$.

The second case, determining if a machine will halt, has not been the subject of much research due to its apparent ease. For $\Sigma(3)$ and $\Sigma(4)$ a halting machine will stop after 6 and 13 steps or less, respectively. For $\Sigma(5)$ the suspected maximum is 4098, still easily achievable with commodity hardware.

The problem with the latter case only becomes apparent when one tries to evaluate $\Sigma(6)$, which is known to be greater than 1.29×10^{865} with $S(6) > 3 \times 10^{1730}$ [MB01]. Obviously simulating such a machine raises significant complications with regard to memory and time. Running such machines to completion is the focus of this paper.

PREVIOUS RESEARCH INTO HALTING CANDIDATES

In his 1990 paper, "Attacking the Busy Beaver 5," Heiner Marxen [MB90] describes a tape compression technique and macro-machine acceleration that are fundamental to the approach taken here. Since then, Marxen has developed some proof-generating algorithms used for running 6-state busy beaver candidates which are generalized further in this paper.

Robert Munafo has provided several proofs of completion for 6-state busy beaver candidates [Mun00]. These proofs give insight into the inner working of some of the more complex machines, but do not necessarily show a way to automate the proof-generation process.

TECHNIQUE

There are three major techniques one can apply to a Turing machine emulator to make possible the simulation of complex halting busy beaver candidates. These techniques are independent of programming language, and are presented in an order such that the later techniques depend on the earlier ones.

Tape Representation

The current 6-state busy beaver candidate addresses a contiguous tape region of over 10^{865} elements. Clearly, mapping this tape to an array in memory is not feasible, even with bit packing. The approach described by Marxen and used in the later techniques in this paper is a form of run-length encoding [MB90].

The tape is represented in memory as a linked list of blocks. Each block contains k sequential elements and a repetition count (denoted in this paper as an exponent). k is an attribute of the tape, not the individual blocks; that is, all blocks must have the same length (but may of course contain different repetition counts).

For example, the tape sequence:

$$110110110101101$$

could be represented in a $k = 3$ tape as:

$$110^3\ 101^2$$

meaning 3 blocks of the sequence 110 followed by 2 blocks of 101. It is assumed that the tape to the left and right of the given tape sequence consists entirely of zeros. Note that there are infinite equivalent block representations for a given tape.

The author has observed that all currently known 6-state halting machines (for example, those discovered by Marxen [MB01]) use fewer than 10 such blocks for a suitable value of k. The selection of k must be made for each machine manually, however a heuristic could easily be developed to handle this task — simply test each value of k from 1 and reject any that require more than 10 blocks after running the machine for some time.

Macro Arcs

With the tape in the format described above, one can begin constructing macro-machines atop the machine being simulated. These macro-machines are 6-tuple, and consist of:

- The current block under the tape head
- The block to write in its place
- The current state
- The state to move to
- The current direction of the tape head (left or right)

- The direction the tape head should move (left, right, or halted)

The addition of an input direction to the tuple allows entire blocks to be processed in a single shift. Implicit is the fact that if the tape head direction is left, the tape head must be at the right edge of the tape block, and conversely if the tape direction is right, the tape head must be at the left edge.

The macro-machine uses the same states as the underlying "naïve" machine. Arcs (transitions) for the macro-machine are generated, as required, from the underlying machine using the following procedure:

1. Initialize the underlying machine with the input state; use the input block as the machine's entire tape; set the tape head to begin at either the beginning or the end of the (defined) tape depending on the value of the tape input direction.

2. Run the underlying machine until its tape head moves beyond the edge of the tape or it halts.

3. Construct a new arc for the macro-machine with the given input state, block and tape head direction. The block output of this arc will be the entire tape of the underlying machine upon its termination. The tape head direction will be left, right or halted according to the final movement of the underlying tape head.

Since the initial configuration of the underlying machine is equal to the configuration of the macro-machine at that instant, and the bounds of where the underlying tape head can move are strictly limited to within one block, we can ensure that any subsequent arcs followed on the macro-machine have an equivalent sequence of valid shifts on the underlying machine.

A macro-machine of N states and block size k has a maximum number of arcs given by:

$$(2N)^k$$

However, most macro-machines use only a subset of the combinatorial possible blocks, and not all of these are visited by every state.

The number of shifts required to halt the macro-machine with an underlying machine M halting in $s(M)$ shifts will be less than or equal to

$$\frac{s(M)}{k}$$

and is typically far less.

It was mentioned earlier that tapes usually contain less than 10 blocks, even when there is a large number of symbols on the tape. Macro-machines can take advantage of this by exploiting a common pattern in candidate machines:

$$\delta(q, x_0, d) = [q, x_1, d]$$

that is, where the input state is equal to the output state and the input tape direction is equal to the output tape direction. When this arc is followed, it may be applied to the block as a whole, rather than just the first instance:

$$w{x_0}^n y \Rightarrow w{x_1}^n y$$

where $w, y \in [0, 1]^*$. In this way the tape head can quickly skip from one end of the tape to the other applying changes only at the ends, giving a linear increase in speed relative to the length of the tape.

The tuple for each arc can be further extended by adding tail-context, encoding information about the block "behind" the tape head as input and effectively creating a 7-tuple machine. The difference between this and merely increasing k is subtle. It allows a macro-machine to apply a complex pattern involving intermediate blocks in a single shift. This pattern has only been observed in some of the more sophisticated 6-state busy beaver candidates.

Proof Generation

Macro-machines improve the speed of a busy beaver by effectively applying changes to the entire tape in a single shift. They cannot, however, increase the speed of the growth of the tape. Consider a busy beaver candidate which outputs n one's. Even under ideal circumstances, a macro-machine must perform at least n shifts before halting.

Proof-machines are built upon macro-machines. Instead of defining arcs that modify a single block at one time, they define arcs, or configuration transitions, that affect the entire tape.

Configuration transitions are not generated for each shift. They are created when a recognised pattern appears during the operation of a macro-machine. The pattern for a 2-variable configuration transition is of the form:

$$s : uw^j xy^k z \Rightarrow^* s : uw^{j+\Delta j} xy^{k+\Delta k} z$$

where $u, w, x, y, z \in [0, 1]^*$, $j, k > 0$ and s is a machine state.

After each shift of the macro-machine, the current configuration (including its state, tape and tape head position) is compared with the entire history of configurations. If a previous configuration exists that differs only in the block exponents on the tape, an attempt is made to prove by induction that the first configuration, with variables substituted in place of the changing exponents, will always become the second after some number of shifts. This substituted configuration becomes the input to a macro-machine for the purpose of proving the configuration transition is valid.

For example, if the matching configurations are:

$$s : 101\ 110^5\ 101^1$$
$$s : 101\ 110^1\ 101^6$$

the input tape for the macro-machine will be:

$$101\ 110^{k+5}\ 101^{j+1}$$

The macro-machine will run until it reaches the configuration

$$101\ 110^{k+1}\ 101^{j+6}$$

or either of the terms $(k + 5)$ or $(j + 1)$ are reduced to k or j, respectively. In the former case, a configuration transition can be created. In the latter case the proof has failed, and operation of the original macro-machine continues as before.

If the proof is successful, the newly created configuration transition will be defined:

$$s : 101\ 110^k\ 101^j \Rightarrow s : 101\ 110^{k-4}\ 101^{j+5}$$

for $k > 4$ and $j > 0$.

Before each shift in the macro-machine, the set of known configuration transitions is examined for a transition that can be applied to the current tape. For a transition with a given $\Delta k < 0$ and $\Delta j > 0$ and a tape of the form:

$$uw^j xy^k z$$

the configuration will be applied $\lfloor k/ - \Delta k \rfloor$ times. The tape after the configuration transition will then be:

$$uw^{j+\Delta j(\lfloor k/\Delta k \rfloor)} xy^{k-\Delta k(\lfloor k/\Delta k \rfloor)} z$$

For example, with the above transition defined, the following configuration:

$$s : 101\ 110^{52}\ 101^1$$

will be applied $52/4 = 13$ times, giving:

$$s : 101\ 110^1\ 101^{65}$$

In this way the tape can easily grow at an exponential rate in very few proof-shifts.

The position of the tape head must of course be considered during these transitions. Marxen solves the problem by allowing variable exponents only the two blocks immediately under, or adjacent to, the tape head. The approach taken by the author is to store the tape head position with the configuration, allowing any number of variable exponents in any position within the tape.

Note that although the author's implementation of the proof-machine allows any number of variables to be defined in the transitions, only 2-variable transitions have been observed in practice.

RESULTS

The application of these techniques has an obvious impact on performance. The author's implementation was written in C and run on a 900 MHz G3 processor. All times are reported in seconds. Where no result is given the machine was run for at least 24 hours before being interrupted.

The productivity is the number of symbols printed by the candidate machine before halting in s steps. The candidate machines are from Marxen's work [MB90] [MB01].

Candidate	Productivity	s	Turing machine	Macro-machine	Proof-machine
BB-5	4098	47176870	26	0.25	0.03
BB-6-1	136612	13122572797	-	1.7	0.03
BB-6-2	95524079	8690333381690951	-	2110	0.05

The candidate machines tested are:

State	0	1
1	1L2	1R3
2	1L3	1L2
3	1L4	0R5
4	1R1	1R4
5	1L0	0R1

Table 1: BB-5, $k = 3$

State	0	1
1	1L2	1L1
2	1R3	1R2
3	0R6	1R4
4	1L1	0R5
5	0L1	1R3
6	1L5	1L0

Table 2: BB-6-1, $k = 3$

State	0	1
1	1R2	1R1
2	1L3	1L2
3	0R6	1L4
4	1R1	0L5
5	1L0	1L6
6	0L1	1L3

Table 3: BB-6-2, $k = 2$

FUTURE WORK AND CONCLUSION

The generalized proof mechanism described in this paper provides a solid foundation for building general proofs for non-halting machines. For example, if a proof is found in which both Δj and Δk are greater than zero, the machine has been shown to be non-halting. With an efficient tree traversal it may be possible to solve for $\Sigma(5)$.

Further work is required before the implementation is able to run current 6-state candidates to completion. The author's current implementation is unable to handle all cases of macro-arcs when variable exponents are involved, so many proofs fail prematurely.

The techniques presented are equally applicable to 4-tuple Turing machines (those which cannot move the tape head and write to the tape in the same shift). For example, a macro-machine with $k = 1$ provides a 5-tuple representation of a 4-tuple machine (since the input tape head direction can be ignored).

This work indicates we have a generalized mechanism for running busy beaver candidates to completion, allowing further exploration into the fields of non-computability and cellular automata.

REFERENCES

[Bra83] A. Brady. The determination of the value of Rado's noncomputable function $\Sigma(k)$ for four-state Turing machines. *Mathematics of Computation*, 40:647–665, 1983.

[Jul92] B. Julstrom. A bound on the shift function in terms of the busy beaver function. *ACM SIGACT*, 23:100–106, 1992.

[LR65] S. Lin and T. Rado. Computer studies of Turing machine problems. *Journal of the Association for Computing Machinery*, 12:192–212, 1965.

[MB90] H. Marxen and J Buntrock. Attacking the busy beaver 5. *Bulletin of EATCS*, 40:247–251, 1990.

[MB01] H. Marxen and J Buntrock. List of 6-state record machines, 2001. `http://www.drb.insel.de/~heiner/BB/bb-6list`.

[MS90] R. Machlin and Q. Stout. The complex behaviour of simple machines. *Physica D: Nonlinear Phenomena*, 42:85–98, 1990.

[Mun00] R. Munafo. Large numbers – notes at MROB, 2000. `http://home.earthlink.net/~mrob/pub/math/ln-notes1.html`.

[Rad62] T. Rado. On non-computable functions. *Bell System Technical Journal*, 41:877–884, 1962.