Attacking the Busy Beaver 5

Heiner Marxen, Specs GmbH, Berlin
Jürgen Buntrock, Technische Universität Berlin

ABSTRACT

Since T. Rado in 1962 defined the busy beaver game several approaches have used computer technology to search busy beavers or even compute special values of Σ. Σ(5) is not yet known. A new approach to the computation of Σ(5) is presented, together with preliminary results, especially Σ(5)≥4098. This approach includes techniques to reduce the number of inspected Turing machines, to accelerate simulation of Turing machines and to decide nontermination of Turing machines.

1. Introduction

The busy beaver game as defined by T. Rado in 1962 [Rad62] asks to construct simple Turing machines which produce a maximum number of ones on their tape before halting. These Turing machines have N states, one of which is the initial state, a tape alphabet with two symbols, named 0 and 1, an initial tape with all 0 symbols, an anonymous halt state, and with each step print a new symbol and move their head either left or right. For each N there is a finite number of such Turing machines. Σ(N) is a function defined to be the maximum number of ones such a Turing machine with N states produces on its tape when halting. An N state Turing machine that does produce Σ(N) ones is called a busy beaver.

Σ has been proved to be not general recursive [Rad62], and hence noncomputable. Σ(1)=1 is trivial, Σ(2)=4 is easy to show, Σ(3)=6 has been shown by [LiR65], Σ(4)=13 has been shown by e.g. [Bra75] [WCF73]. Σ(5) is not yet known. In [LSW83] Σ(5)≥501 is given, and 1984 Uhing found Σ(5)≥1915 [Dew85]. In August 1989 the authors found Σ(5)≥4098. Green has given a non-trivial (not primitive recursive) lower bound for Σ [Gre64].

While a lower bound of Σ(N) for a certain N can be shown by a single Turing machine, computation of Σ(N) itself requires to prove that all other N-state Turing machines do either not halt or produce not more ones. Obviously, the halting problem for Turing machines (which is noncomputable) is involved.

2. Overall Algorithm

Like others [Bra66] [WCF73] [LSW83] we directly and constructively follow the definition of Σ(N), ‘‘the maximum number of ones produced by a halting Turing machine (of appropriate type) with N states’’. Thus the overall algorithm is:

(1) Enumerate all N-state Turing machines M, and for each of them do (2).

(2) Simulate the behaviour of M, starting with an empty tape, until one of (3), (4), (5a) or (5b) happens.

(3) M halts. Count the ones on the tape, compare to the current record and eventually update it.

(4) M is proved to never halt. Forget about M.

(5a) The size of the tape exceeds a certain predefined limit. Note M to be undecided. With an appropriate representation of the tape this happens rarely.

(5b) The number of simulation steps exceeds a certain predefined limit. Note M to be undecided.

Some earlier approaches have left out (4). Some have tried restricted implementations, e.g. [WCF73]. As opposed to that we stress efficient methods to decide non-halting. Busy beavers can be found without (4), computation of Σ needs it.

In order to use the above algorithm to compute $\Sigma(5)$ within practical time limits (say a month), three subgoals have to be achieved:

- Not all combinatorically possible Turing machines must be considered. [Bra66] gives an upper bound of $(6N)^{2N}$, which is apx. $6*10^{14}$ for N=5, which clearly is far too much. [LiR65] give $(4N+4)^{2N}$, which for the purpose of computation of Σ can be improved to $(2N-1)*(4N)^{2N-2}$. This is $2.3*10^{11}$ for N=5, and still too large. The autors' programs currently enumerate less than $9*10^7$ Turing machines.
- Acceleration of the simulation of the behaviour of Turing machines. The current 5-state busy beaver does more than $4.7*10^7$ steps before it halts. This is already large enough to attract special attention, and in fact is much larger than the limits that have been used by [Lyn72] [LSW83] and Uhing.
- Recognition of never halting Turing machines should be efficient, happen early in the process of simulation and should be as complete as possible and necessary for N=5.

How to achieve these goals is described in the next chapters.

3. Enumeration of Turing Machines

Even if instantaneous decision of each Turing machine were done, enumeration of all combinatorically possible ones were not practical. The numbers are already prohibitively large for N=5. Fortunately it is not very hard to reduce these numbers substantially.

If there is a set S of Turing machines such that either none of them halts, or all of them halt and produce the same number of ones, then – in order to compute Σ – it is sufficient to decide/simulate just one machine out of S. If this set S can be ordered then there is a unique smallest element representing the set S. Such equivalence sets are the basic mechanism to reduce the amount of enumeration and computation.

The most important equivalence class has already been used by [Bra66], where its result is termed *tree normal form* (TNF). The normalization sceme used in [LiR65] is similar. Others [WCF73] [LSW83] have used TNF also. It identifies the set of machines that differ only in the naming of the states (isomorphism) and in transitions which never got used. All these machines obviously exhibit identical behaviour. Prior to simulation it is normally not known which transition will get used. Hence we arrive at the following way to enumerate all Turing machines by enumeration of single transitions during simulation:

(1) Mark all 2N transitions of the machine M undefined (i.e. arbitrary). Do step (2) with M.

(2) Simulate M (representing a class of machines) until one of (3), (4), (5) or (6) happens.

(3) M halts. Count ones etc.

(4) M is decided (proved) to never halt. Forget about M.

(5) The tape size or step number limit is reached. Note M to be undecided.

(6) A transition marked undefined is to be used. Enumerate all meaningful values (see below) for this transition, and recursively do step (2) with M set to each of these more completely defined machines.

The set of legal transition values is formed by an arbitrary combination of (N+1) target states, 2 symbols to write, and 2 directions to move. The set of meaningful transitions enumerated in step (6) is a subset of all legal ones by the following reduction rules:

- From the set of states M has not yet taken, only the smallest one (according to some arbitrary but fixed order on the states) is used as target state (isomorphism).
- When defining the very first transition force a state change (else never halts), write a one (isomorphism), and move left (symetry). This completely fixes the very first transition.
- When defining the very last transition, consider only the halt state as target state.
- If the target state is the halt state, go left (independant) and write a one (never worse). This completely fixes the last transition.
- If there is a state that M has not yet taken, do not consider the halt state as target state ($\Sigma(N+1) > \Sigma(N)$).

This approximately enumerates the TNF. There are further equivalence classes of medium importance, e.g:

- If there are two states which are (syntactically) equivalent, these two can be identified ($\Sigma(N+1) > \Sigma(N)$). Example: $(x,0)\rightarrow(x,0,R)$, $(x,1)\rightarrow(z,0,L)$, $(y,0)\rightarrow(y,0,R)$ and $(y,1)\rightarrow(z,0,L)$ imply that states x and y are equivalent.
- If a sequence of three transitions is guaranteed to have the same effect as a single transition, only one of both constructions need be inspected. Example: Let x,y,z and s be states with $x \neq y$, a, b, and c arbitrary symbols, and $D \in \{L,R\}$, then $(x,0)\rightarrow(y,0,L)$, $(x,1)\rightarrow(y,1,L)$, and $(y,b)\rightarrow(z,c,D)$ implies that $(s,a)\rightarrow(x,b,R)$ and $(s,a)\rightarrow(z,c,D)$ have the same effect.

4. Acceleration of Simulation

The basic acceleration technique is to use a *macro machine*. A K-macro-machine operates on blocks of K tape symbols (here bits) as one macro symbol. It has a duplicated set of states to code whether the left- or rightmost bit of the macro symbol is under the head. A K-macro-machine for an N-state Turing machine has $2*N*2^K$ transitions, each of which is defined by running the original machine on a limited tape of length K.

The speed-up factor ranges between $1+\varepsilon$ and $N*K*2^K$ for some small ε (which depends on N and K). Thus, not much speed-up is guaranteed, but it can be significant. For small values of K (e.g. 6) the transition table of the macro machine can be implemented directly. Sparse table and caching techniques allow even $K \geq 30$ to be implemented. Note, that only those transitions need to be computed, which actually get used for simulation.

The next and most important acceleration technique is to use explicit tape symbol repetition counts, here termed *exponents*. E.g. ...abbaaab... is coded $...a^1b^2a^3b^1...$. The most obvious acceleration occurs when a transition like $(x,a)\rightarrow(x,b,R)$ happens to meet the left side of a heap of ‘a’ symbols, which in one big step are converted to the same amount of ‘b’ symbols. (If the target state is $y \neq x$ then one ‘a’ is cut off the heap to get converted into one ‘b’.) Up to now, all 5-state busy beaver champions did perform such simple repetitions (for some small K) most of their time. As a rule of thumb, if a Turing machine executes many steps before halting, it does perform many simple iterations of this type. A very nice effect of this tape representation is that in nearly all ‘‘interesting’’ cases the tape is not longer than 7 cells. Longer tapes indicate nearly always non-termination or inappropriate K. Practical experience indicates that K should be chosen seperately and carefully for each Turing machine.

Worth noting is a class of Turing machines, ‘‘counting machines’’, which do not gain speed from exponents, as their basic behaviour on the tape is to enumerate (e.g. the binary representations of) successive natural numbers. Other techniques to accellerate especially counting machines are currently studied by the authors. Because both of the above techniques involve tape representation, they also influence other aspects, especially some non-termination decisions. Explication of iteration can be generalized, but it is not yet clear which generalization does pay off. Further research is necessary here.

5. Nontermination Decisions

Deciding that a Turing machine will never halt is always done by predicting the behaviour of the machine exactly enough, that it is sure that no not yet defined transition will ever be used, i.e. those that are already defined will be reused infinitely. This can be achieved by quite different arguments.

(a) Constructively give the pattern of further behaviour (forward reasoning).

(b) Show that all paths to arrive at using another transition cannot start here (backward reasoning).

To (b): When in the process of enumeration a transition into the halting state is defined, it will immediately be used, the machine gets evaluated and is discarded. Thus, during normal simulation all defined transitions do not halt. Let D and U be the set of currently defined and not yet defined transitions. In order to halt, one transition $u \in U$ has to be needed and defined. That u has to be reached exclusively through transitions from D, i.e. by one out of the set P of all pathes of length L ($L \leq S+1$) of the form $D^{L-1} \times U$, where S is the number of steps already done by the machine in question. Let Q be the set of all those $p \in P$, that are possible according to two restrictions. First, the target state of each transition in p has to match the source state of the next transition in p. Second, for each transition t in p the tape context defined by the transitions in p before t must not contradict with the source symbol of t. If there is an L such that Q is empty, then all u are currently unreachable and the machine cannot halt any more. Such can be implemented by backward simulation. [WCF73] have used this restricted to $L \le 5$.

To (a): If we find a pattern of future operation of which we can prove that it will be iterated infinitely, the machine will never halt. This is still a very general statement. Instantiations are:

- If there is a set S of states such that all transitions from elements of S are defined and their target state is also in S, (and the machine is in one of these states) it will never again leave S and thus not halt (closed state/transition cluster).
- If a machine reproduces the exact same configuration (state and tape relative to its head), it will repeat exactly so‡.
- If a machine produces a ‘‘greater configuration’’ it will continue to eventually produce even greater configurations (reusing the same transitions). This is explained below.

A general notion of ‘‘greater configuration’’ is hard to describe verbose, hence we use entry #8 of table 1 as a simple example.

#	Input Bit	Transition on State						Steps	Ones	Comment
		A	B	C	D	E	F			
1	0	B1L	C1L	D1L	A1R	H1L	—	47,176,870	4098	current BB(5), step champion
	1	C1R	B1L	E0R	D1R	A0R	—			
2	0	B1L	C1R	A1L	A1L	H1L	—	11,798,826	4098	also BB(5)
	1	A1L	B1R	D1R	E1R	C0R	—			
3	0	B1L	C1R	A1L	C0R	C1L	—	∞	—	symbol size 80
	1	B0R	E0L	D0R	A1R	H1L	—			
4	0	B1L	C1R	D0R	A1L	H1L	—	$> 2*10^9$	?	‘‘chaotic’’
	1	B1R	E0L	A0L	D0R	C0L	—			
5	0	B1L	A0R	A0R	E0L	B0R	—	∞	—	simple counter
	1	A1R	C0L	D1L	B1R	H1L	—			
6	0	B1L	A0R	C0R	E1L	B0L	—	?	?	complex ‘‘counter’’
	1	A1R	C0L	D1L	A0R	H1L	—			
7	0	B1L	C1R	F0R	A1L	A0L	E1L	13,122,572,797	136,612	example for many ones with 6 states
	1	A1L	B1R	D1R	E0R	C1R	H1L			
8	0	B1L	C0R	H1L	—	—	—	∞	—	example for greater configuration
	1	A1R	B1L	A1R	—	—	—			

TABLE 1. Transition Tables of Some Interesting Turing Machines

After 1 step this machine reaches the configuration B: $\underline{0}1$, after 5 steps B: $\underline{0}11$, etc. More generally, from B: $\underline{0}1^n$ ($n \ge 1$) in 2n+2 steps it reaches B: $\underline{0}1^{n+1}$. This process is essentially independent of n, and will be repeated infinitly. The 1-macro-machine with exponents in 5 macro steps transforms the configuration bR†: $\underline{0}^1 1^n$ ($n \ge 2$) into cL: $\underline{1}^n$, aL: $1^1 \underline{1}^{n-1}$, aL: $1^n \underline{0}^1$, bR: $\underline{1}^n 1$, and then bR: $\underline{0}^1 1^{n+1}$, which is ‘‘greater’’, i.e. the state, all symbols and exponents are equal, except for one exponent, which became larger. This happened in 5 macro steps, which all together are unable to handle original exponents ≥ 5 differently. If such a transformation has happened for some exponent ≥ 5 this (production of a greater configuration) is guaranteed to happen for all further exponents, as they are also ≥ 5. The exponent can be considered to be ‘‘not recognized’’ by the machine.

‡ Reproduction of the initial (all zero) tape but with another state than the start state is a special case. Here states can be renamed (isomorphism). Either the machine does not halt or it will be enumerated and evaluated in its isomorph form.

† The notation sD ($D \in \{L, R\}$) is used for that state of the macro machine which corresponds to the state s of the original machine and has the D-most bit of the macro symbol under its head.

Although partial independence of further operation from the tape (from one exponent in the example) is clearly a non-trivial property, it is very general and quite important. There are special cases which occur frequently and can be handled efficiently. A very simple special case has been used by [WCF73]. The method to prove partial independence can also be arbitrarily complex, i.e. this decision type may be extended as needed.

6. Results and Examples

The authors have written two experimental programs, one in C (ca 8000 lines) and one in ML (a functional programming language) which from all machines (ca 88 million) currently leaves approximately 0.3% undecided. This can and will be improved substantially. Both programs discover the earlier champions, several 4096 and 4097 machines, and two 4098 machines. These two and some others are specified in table 1. Entry #7 demonstrates $\Sigma(6) \geq 136{,}612$ and is the authers' current 6-state champion.

7. Future Directions

Considerable effort has been necessary to attack $\Sigma(5)$ and it is not yet totally sufficient. Also the amount of computation time is non-trivial (about ten days using a 33 MHz Clipper CPU). The various combinatoric upper bounds suggest that the computation of $\Sigma(6)$ will need around 500 times as many machines to be enumerated as the computation of $\Sigma(5)$, and of course 6-state machines are harder to decide. On the other hand, some thought and computer technology have made possible substantial progress since 1962. Also, computation of Σ can be partitioned very easily and with a network of, say 2000, computers (workstations) $\Sigma(6)$ still appears to be attackable. Admittedly, this is currently not more than just the belief of the authors and contrary to the belief of others, e.g. of Allen H. Brady as quoted in [Dew85]. A more detailled paper about the computation of $\Sigma(5)$ is to come and might explain this belief a bit better.

References

[Bra66] Brady, A.H., *The conjectured highest scoring machines for Rado's $\Sigma(k)$ for the value k=4*, IEEE Transactions on Electronic Computers, vol. EC-15, pp. 802-803, October 1966.

[Bra75] Brady, A.H., *Solution of the non-computable ''Busy Beaver'' game for k=4*, Abstracts for: ACM Computer Science Conference (Washington, DC, February 18-20, 1975), p. 27, ACM, 1975.

[Dew85] Dewdney, A.K., *Computer Recreations*, Scientific American, vol. 252, no. 4, pp. 12-16 esp. 16, April 1985. Translation: *Computer-Kurzweil*, Spektrum der Wissenschaft, pp. 8-12 (esp.12), June 1985.

[Gre64] Milton W. Green, *A lower bound on Rado's sigma function for binary Turing machines*, Switching circuit theory and logical design, Proceedings of the Fifth Annual Symposium, Princton University, Princeton, N.J., November 11-13, 1964, The Institute of Electrical and Electronics Engineers, Inc., New York 1964, pp. 91-94.

[LiR65] Lin, S. and T. Rado, *Computer Studies of Turing Machine Problems*, Journal of the Association for Computing Machinery, vol. 12, no. 2, pp. 196-212, April 1965.

[LSW83] Jochen Ludewig, Uwe Schult, Frank Wankmüller, *Chasing the Busy Beaver - Notes and Observations on a Competition to Find the 5-state Busy Beaver*, Forschungsberichte des Fachbereichs Informatik, Nr. 159, Universität Dortmund, Dortmund, 1983.

[Lyn72] Lynn, D.S., *New Results for Rado's Sigma Function for Binary Turing Machines*, IEEE Transactions on Computers, vol. C-21, no. 8, pp. 894-896, August 1972.

[Rad62] Rado, T., *On non-computable functions*, The Bell System Technical Journal, vol. 41, no. 3, pp. 877-884, May 1962.

[WCF73] Weimann, B., K. Casper and W. Fenzl, *Untersuchungen über haltende Programme für Turingmaschinen mit 2 Zeichen und bis zu 5 Befehlen*, pp. 72-81, in: GI Gesellschaft für Informatik eV: 2. Jahrestagung (Karlsruhe, October 2-4, 1972), Lecture Notes in Economics and Mathematical Systems, vol. 78, Springer Verlag, Berlin, 1973.