The Busy Beaver, the Placid Platypus and other Crazy Creatures

James Harland

School of Computer Science and Information Technology
RMIT University
GPO Box 2476V
Melbourne, 3001
Australia
jah@cs.rmit.edu.au

Abstract

The busy beaver is an example of a function which is not computable. It is based on a particular class of Turing machines, and is defined as the largest number of 1's that can be printed by a terminating machine with n states. Whilst there have been various quests to determine the precise value of this function (which is known precisely only for $n \leq 4$), our aim is not to determine this value per se, but to investigate the properties of this class of machines. On the one hand, these are remarkably simple (and, intuitively, form perhaps the simplest class of computationally complete machines); on the other hand, as some of the machines for $n = 6$ show, they are capable of representing phenomenally large numbers. We describe our quest to better understand these machines, including the *placid platypus problem*, ie. to determine the minimum number of states needed by a machine of this type to print a given number of 1's.

1 Introduction

The busy beaver function (Rado 1963) was introduced by Rado in the 1960's as an example of a non-computable function, and is defined in terms of a particular class of Turing machines(Sudkamp 2005). This class is one in which there is a single tape which is infinite in both directions, the machine is deterministic, the tape is initially blank and the tape alphabet consists of only two symbols (traditionally blank (B) and one (1)). For each machine there is a distinguished halt state, from which there are no transitions. An n-state machine of this form is one that contains one halt state and n further non-halt states. The *busy beaver* function is the maximum number of 1's that is printed by a terminating n-state Turing machine. This function is often denoted as $\Sigma(n)$; in this paper we will use the more intuitive notation of $bb(n)$. The number of 1's printed by the machine is known as its *productivity*.

This function can be shown to be non-computable by proving that it grows faster than any computable function. Hence this function holds a special interest as one with an intuitively simple definition but which is capable of some phenomenally fast growth.

Because of this rate of growth, it has been historically quite difficult to evaluate precise values for this function. Its value for $n = 1, 2, 3$ were determined by Lin and Rado in the 1960's(Lin &Rado 1964), and the case for $n = 4$ by Brady in the 1970's(Brady 1983). However, establishing $bb(5)$ and $bb(6)$ has been more problematic, particularly due to some spectacularly large lower bounds for these values(Marxen and Buntrock 1990, Marxen 2005). The only claim, to the author's knowledge, for a precise evaluation for $n = 5$ is Master's thesis published in August, 2005 (Kellett 2005), which uses a mixture of machine evaluation and human analysis (although there are some others performing similar searches, such as Georgi Georgiev(Skelet 2005)). There are some interesting analyses of the current 5-state and 6-state champions (Munafo 2005, Michel 2005), but some speculate that due to the sheer size of the numbers involved, $bb(7)$ may never be known, despite some promising techniques for evaluating machines with extremely large productivities (Holkner 2004).

A recent variation on this problem is to consider machines with more than two tape symbols, and there have been various spectacular examples of large computations with such machines (such as a 3-state machine with 3 tape symbols that prints 544,884,219 symbols before halting (Brady 2005)). The relationship between the busy beaver function and the 3n+1 sequence has been investigated by Pascal Michel (Michel 2005), which has resulted in an elegant analysis of the most complex machines currently known. Some analysis has also been done on Turing machines in which one can either move the tape head or write a new symbol on the tape, but not both in one atomic action(Ross 2003). There has also been an investigation of busy beaver functions on a one-way tape (Walsh 1982) rather than a two-way one.

The main use of this function has been as a simple example of non-computability, especially as there are also larger functions which can be defined similarly. One such function, often denoted $\mathcal{S}(n)$ in the literature, the maximum number of state transitions made by a terminating Turing machine of the above form. We denote this function as $ff(n)$.[1] A notable use of this function is given in Boolos and Jeffrey(Boolos &Jeffrey 1980), in which the busy beaver function is the subject of the first undecidability result established, rather than the more usual choice of the halting problem for Turing machines.

The relationship between $bb(n)$ and $ff(n)$ has been investigated (Julstrom 1992), and it is known that $ff(n) < bb(3n + c)$ for a constant c(Yang,Ding &Xu 1997). However, this is still rather loose, and does not give us much insight into the relationship between $bb(n)$ and $ff(n)$. In a similar manner, lower bounds on $bb(n)$ have been known for some time (Green 1964); however, those given for $n \leq 6$ have been far surpassed already.

In this paper, we introduce some new perspectives on the busy beaver function. In essence, our main interest is not so much the ability to define non-computable functions, but the properties of the un-

[1] We call this function the *frantic frog*.

derlying class of Turing machines.

To motivate this point of view, consider the table below (drawn from (Lin &Rado 1964), (Brady 1983),(Marxen and Buntrock 1990), (Marxen 2005)).

n	$bb(n)$	$ff(n)$
1	1	1
2	4	6
3	6	21
4	13	107
5	≥ 4098	$\geq 47,176,870$
6	$\geq 1.29 * 10^{865}$	$\geq 3 * 10^{1730}$

The vast increase from $n = 5$ to $n = 6$ seems to suggest that 6 states is the minimum for some particular functionality, such as multiplication or exponentiation, or a universal Turing machine. Moreover, it is counterintuitive (to say the least!) to be able to print out a sequence with as many as 10^{865} elements in it via a Turing machine with only 6 states.

This makes it clear that the range of productivities for 6-state machines is astronomical, and leads us to a problem which is in some sense dual to the busy beaver problem. The busy beaver problem may be considered as finding the maximum number of 1's that can be printed by a Turing machine with a given number of states. The dual problem is to find the minimum number of states needed in order to print a given number of 1's. Hence an n-state machine of productivity m may be considered as evidence that $bb(n) \geq m$ as well as evidence that it requires at most n states to print m 1's. A natural question is whether there is a 5-state machine of productivity m for every $bb(4) < m \leq bb(5)$. Because of the duality with the busy beaver, we denote this as the *placid platypus problem.*[2]

In order to answer questions such as these, it seems appropriate to prepare the groundwork by searching amongst the 5-state machines, but in such a way that more than just the champion machine (i.e. the one of highest productivity) is retained. It should be stressed that this investigation is very much a means to determine better analysis techniques, which will hopefully make such searching partially (or perhaps totally) redundant. We thus commence with an empirical investigation, which will hopefully lead us into a more analytic one. For example, whilst it is well known that the lower bound for $bb(5)$ is 4,098, there are actually 6 machines with productivities equal to or very close to this value, including two machines which produce 4,098 ones. However, as one of them takes 47,176,870 transitions and the other 11,798,826, one can argue that the second is actually a "better"; machine (at least in terms of the effort required to generate the 1's that are output). In addition, the first machine at one point has more than 12,000 1's on the tape, but just before terminating, it deletes two-thirds of these 1's. The first though, is usually considered the 5-state champion, as the larger value provides a faster-growing function than then small one. In addition, it is known that there are two 5-state machines with productivity 4097, and a further two with productivity 4096. However, the next highest known productivity is 1471, which suggests that these six machines are rather unusual. Analysis of this phenomenon has concentrated on determining the productivity of a particular machine; however, it would seem that it is at least as interesting to determine why only this small number of machines has this behaviour.

[2]What could be the opposite to a busy beaver? A platypus is an Australian monotreme, and hence is native to the southern hemisphere (as the beaver is to the north). Platypuses are shy and retiring by nature, making this name a natural choice.

In this paper we explore some of these issues, with a particular focus on the placid platypus. Accordingly, the main contribution of this paper is to identify and discuss some new aspects of this well-known problem, rather than to report on a particular technical development. In Section 2 we disucss some basic concepts, and in Section 3 we report on the state of our (as yet incomplete) search of the 5-state machines. Section 4 presents some details which emerge from the search and Section 5 introduces the placid platypus in some detail. Section 6 discusses some further ideas of interest.

2 Definitions

We use the following definition of a Turing machine.

Definition 1 *A Turing machine is a quadruple* $(Q \cup \{h\}, \Gamma, \delta, q_0)$ *where*

- h *is a distinguished state called a* halting state
- Γ *is the tape alphabet*
- δ *is a partial function from* $Q \times \Gamma$ *to* $Q \times \Gamma \times \{l, r\}$ *called the* transition function
- $q_0 \in Q$ *is a distinguished state called the* start state

Note that there are no transitions for state h, and that as δ is a partial function, there is at most one transition for a given pair of a state and a character in the tape alphabet.

Note that this is the so-called quintuple transition variation of Turing machines, in that a transition must specify for a given input state and input character, a new state, an output character and a direction for the tape in which to move. Hence a transition can be specified by a quintuple of the form

(State, Input, Output, Direction, NewState)

Some varieties of Turing machines allow only one of the latter two possibilities, i.e. either to write a new character on the tape or to move , and not both; for such machines, clearly only a tuple of 4 elements is required. Note that our machines are deterministic (due to δ being a function rather than a relation).

Our class of machines will be a particular type of this variety of Turing machines.

Definition 2 *A* unary Turing machine *is a Turing machine containing a single two-way infinite tape, and whose tape alphabet includes only blanks and 1.*

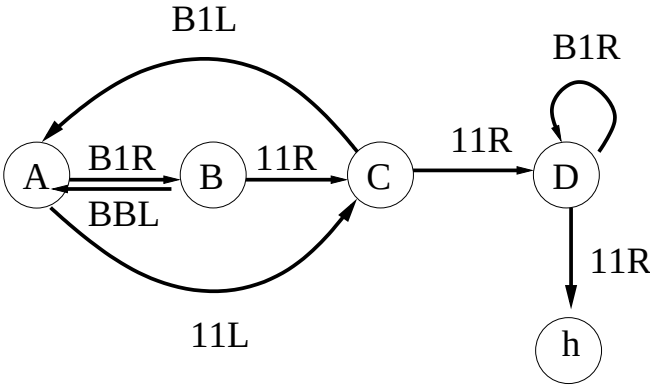
A beaver machine *is a unary Turing machine whose tape is entirely blank on input.*

An animal machine *is a unary Turing machine whose tape contains a finite number of 1's on input.*

We denote by an n-state Turing machine one in which $|Q| = n$. In other words, an n-state Turing machine has n "real" states and a halt state.

Clearly a crucial issue in the evaluation of the busy beaver function is to determine whether or not a given machine terminates. Whilst in general this is undecidable, it is still an open problem, to the author's knowledge, whether the $n = 5$ case is decidable. In practice, this will not be a great problem if the number of machines whose status cannot by determined (often referred to as *holdouts*) is small enough for hand analysis (say under 100).

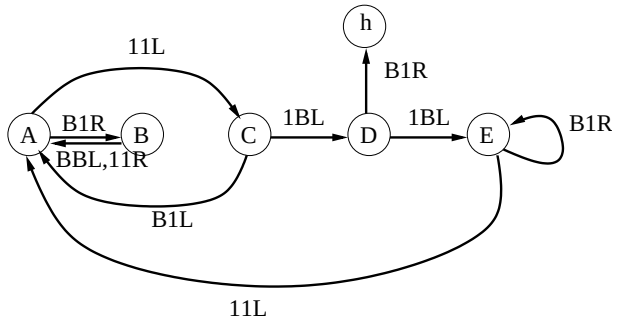
There are various ways in which a beaver machine may fail to terminate. One such way is to move endlessly in one direction, as the machine below does.



This machine prints three 1's before endlessly moving towards the right.

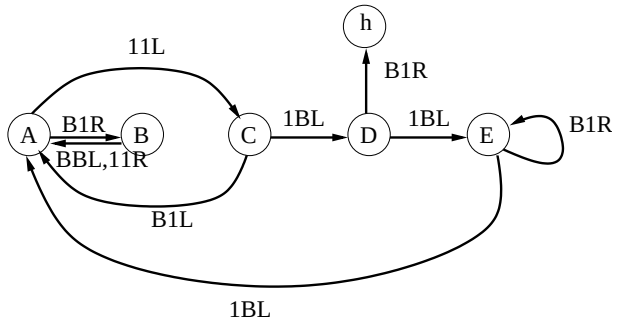
We call such machines *escapees*.

Another way is to return to exactly the same configuration as an earlier one. For example, consider the machine below.



This continuously returns to a configuration in state E with a single 1 on the tape. We call such machines *cyclers*.

A more subtle way is to reproduce the same context further along the tape. For example consider the machine below.



This machine does not repeat the overall state of the tape, but only because it is repeatedly shifting the repeated pattern along to the left. We call such machines *dizzy ducks*. Note also that this machine only differs from the previous one by changing the output of the transition for state E with a one from a one to a blank.

Each of these three cases is relatively easy to detect, and relatively common in the search conducted thus far. A further useful observation is that if the tape becomes all blank during computation, then we may immediately discard the machine from consideration. This is not because the machine will necessary loop (although it may, of course), but because there will be some other machine that produces the same eventual outcome without the tape going blank. To see this, let us assume that the tape becomes blank in state 4, and the computation continues from there. Then there is another machine which has the same computation behaviour as the original machine does from state 4, but commencing from the second machine's initial state. Hence we can immediately disregard any machine which makes the tape return to being all blank. We call such machines *blankers*.

3 Searching for busy beavers

Searching for machines of a particular type (such as the busy beaver) is a matter of enumerating all possible transitions, and then performing various tests. As the number of n-state machines is exponential, it is impractical to do so for every machine. For an n-state machine, there are $2n$ transitions, each with 2 possible outputs, 2 possible directions and $n+1$ possible new states, leading to a total of $(4(n+1)^{2n}$ machines. However, as the halt state will only be used once in each machine, this naive maximum can be reduced to $2n \times (4n)^{2n-1}$. In addition, by fixing the first transition (Lin &Rado 1964), we can reduce this further to $(2n-1) \times (4n)^{2n-2}$.

This number remains formidable. For $n = 5$, this still leaves around 2.3×10^{11} machines to evaluate. In practice, though, this figure can be reduced significantly.

The standard way in which to search for busy beaver machines is to use the *tree normal form* method of Lin & Rado(Lin &Rado 1964). In essence, this involves emulating the machine as transitions are generated. By running an incomplete machine until an unspecified transition is required, we can then ensure that the new transition obeys certain constraints, such as not generating a blank tape, and not creating a cycler or an escapee. In addition, this method ensures that the halt transition is added to the machine only when all other transitions have been allocated. In other words, we ensure that the halt transition will be the last one used.

We execute the machine until the halt transition has been allocated, or until some specified maximum number of transitions has been reached. As we are dealing with 5-state machines, we used a maximum of 110, i.e. three more than $ff(4)$, thus ensuring a "genuine" 5-state machine.

At this point, we could store the machine and proceed to analysis. This reduces the number of machines that must be stored to a little under 10^8. This is still non-trivial, but much more in line with what can be achieved by commodity hardware (such as a typical desktop PC). However, our analysis of such machines showed the presence of a surprisingly large number of cyclers and blankers, whose behaviour would have been noticed earlier if we had run all generated machines for 110 transitions, rather than halting once the final transition was decided. Hence we executed all machines thus generated for 110 transitions, removing those which were found to be cyclers or blankers in this time. This drastically reduced the search space even further (at a significant increase in the time required to perform the search), leaving "only" 15,595,622 machines to be stored and further analysed.

Note also that when searching for busy beavers, it is sensible to allocate the output of the halt transition to be always 1. This clearly cannot decrease the productivity, and may in fact increase it. This also helps, in a minor way, to reduce the search space. We will denote such machines as *halting up* machines; we will return to this point later.

We have a prototype implementation of this approach, but both the development of this program and the search itself are still ongoing. The current implementation consists of about 1,800 lines of Ciao

Prolog (Ciao Manual 2005) (around half of which is redundant). A feature of this implementation, when compared to some others, is that we are interested in keeping the evaluation results for all generated machines, and not just those which halt. Whilst this may seem overly pedantic, and wasteful of storage space, it seems to be a useful resource for the desired further analysis of this class of machines. The storage requirements themselves are not outrageous by modern standards; even using a naive method of storage takes no more than 3 Gigabytes without any compression, which is well within the storage capacity of an average PC. An intended side-effect of our investigation is a publicly available database of these machines; clearly it is important for this database to be as irredundant as possible, and hence we hope to eventually provide a more analytically compact representation of this information than a mere collection of all instances of the generated machines.[3]

For 4 states, there are in principle 117,440,512 machines to be evaluated. The above techniques reduced this to "only" 444,481, of which 208,011 did not terminate, and a further 115,750 made the tape blank during computation. This leaves 120,720 which terminated (or around 0.1% of the original number). Of these, only 1,939 had productivity of 7 or more (i.e. an improvement on the 3-state busy beaver), and only 22,283 had productivity of 5 or more. A more detailed analysis is given below.

≤ 4	5	6	7	8	9	10	11+
98,437	15,089	5,255	1,487	357	74	11	10

Hence the maximal machines are very rare in the general population, even when non-terminating machines are removed. This strongly suggests that an analytical criterion is possible.

Our statistics are less complete for the $n = 5$ case. However a preliminary analysis of the search results to date is given below.

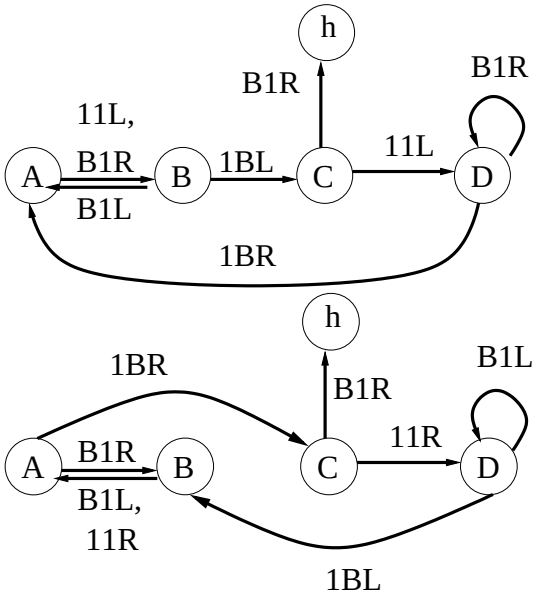
Type	Percentage
halts	69.2
cyclers + escapees	0.001
repeaters	30.1
blankers	0.0005
unclassified	0.6

Hence classifying 99.4% of the cases leaves "only" 106,379 machines still to be classified. Such machines will require more sophisticated analyses, such as inductive methods of showing that a machine does not terminate (as suggested in (Brady 1983)). In addition, there are various termination analysis techniques that are used in analysing constraint logic programs and other declarative programming paradigms which may be directly applicable here. Incorporating such techniques into the implementation is work in progress.

4 "'Maximal' machines

Consider the two machines below.

[3]This collection will be made available at `http://www.cs.rmit.edu.au/~jah/busybeaver`.



The first one is the 4-state busy beaver, with productivity 13 and taking 107 steps to terminate, thus showing that $bb(4) \geq 13$ and $ff(4) \geq 107$. However, the second hand machine also has productivity 13, but only takes 96 steps to terminate. Hence it is tempting to consider the right hand machine as a more optimal one than the left hand one, in that it has the same productivity for fewer steps of execution.

This situation is reflected in the 5-state case. Consider the six machines in Figure 4.

The machine m_1 is the 5-state busy beaver candidate. However, as in the case for $n = 4$, there is a machine (m_2) of the same productivity which takes fewer steps (denoted hops in the table above), and it is arguably a better machine for this reason. This pairwise behaviour recurs for productivities of 4,097 and 4,096 as well. In addition, m_4 and m_6 are remarkably similar; the differ only on two transitions (state B with input 1 and state D with input 1). A more in-depth understanding of the relationships between these machines is clearly crucial to an understanding of the busy beaver function.

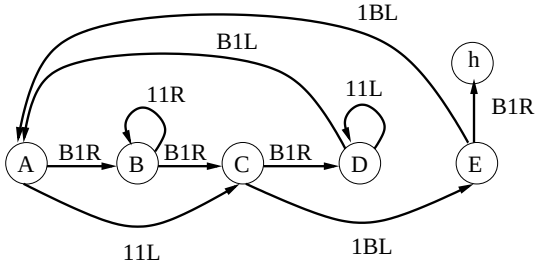
5 The Placid Playtpus

It is trivial to show that $bb(n) \geq n$, as it is trivial to find an n-state machine that prints n 1's and then halts. Boolos and Jeffrey show how it is possible to take a given beaver machine, and construct another which produces doubles the length of the string of 1's output by the original machine. In our framework, this takes an extra states, thus showing that $bb(n + 7) \geq 2n$ Hence given an n-state machine which prints m 1's, we can repeatedly apply these additional states to get a $n+7k$-state machine which prints $m \times 2^k$ 1's.

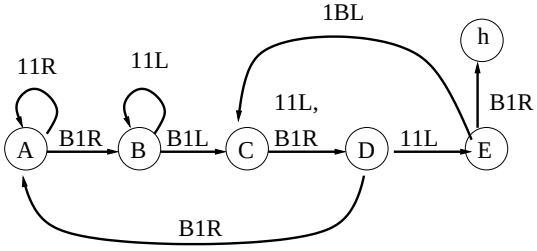
A result claimed in (Dewdney 1993), but left as an exercise, is that $bb(n) \geq 2^n$. In other words, to print m 1's, a machine that has no more than $log_2 m$ states is required.

This leads the following question: given a string of n 1's, what is the minimum size of of a beaver machine which prints it? We will refer to this function as the *placid platypus* function, and denote it by $pp(n)$. This is in some sense a dual to the busy beaver function, in that if $bb(n) \geq m$, we have that $pp(m) \leq n$. In other words, an n-state beaver machine which prints m 1's shows both that $bb(n) \geq m$ and $pp(m) \leq n$.

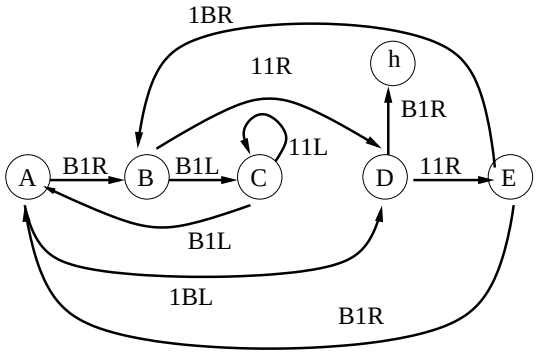
Dually to the frantic frog function, we also define the *weary wombat* function, which is the minimum number of state transitions for an n-state machine necessary to print $pp(n)$ 1's. We denote this function



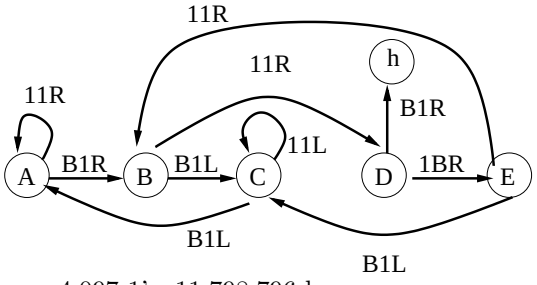
m_1: 4,098 1's, 47,176,870 hops



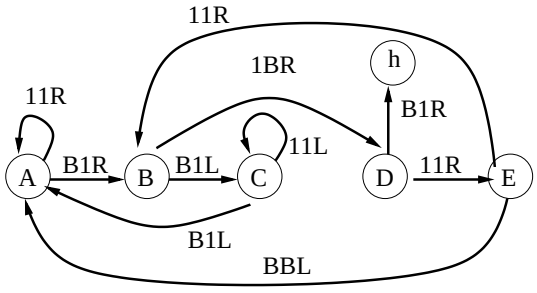
m_2: 4,098 1's, 11,798,826 hops



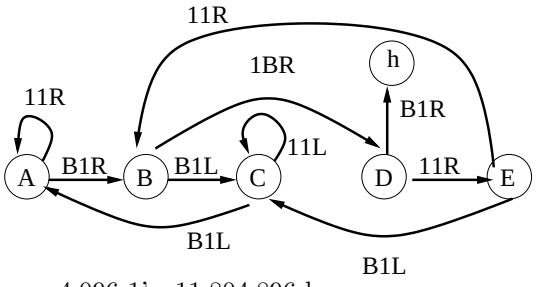
m_3: 4,097 1's, 23,554,764 hops



m_4: 4,097 1's, 11,798,796 hops



m_5: 4,096 1's, 11,804,910 hops



m_6: 4,096 1's, 11,804,896 hops

Figure 1: Monster 5-state machines

by $ww(n)$.

It is not hard to some early values of $pp(n)$ and $ww(n)$, as shown in Figure 5.

Above the productivity of 20, our search is much less complete (possibly due to out implementation capping the maximum number of hops initially at 110). As can be seen in Figure 3 below, it is as yet unknown whether there is a 5-state machine of productivity 29, 30, 31, 34 or 37.

In fact, it is possible to obtain a machine of productivity 31 from the one of productivity 32, by changing the halt transition. In other words, as the machine for productivity 32 is a halting up machine, this shows that there is a halting down machine of productivity 31.

This suggests that an interesting question is whether for every $bb(4) < m \leq bb(5)$ there is a machine of corresponding productivity. Certainly this is true for the range $bb(3) < m \leq bb(4)$, and must be false for the range $bb(5) < m \leq bb(6)$. The former case we have shown by construction, and in the latter one, there are simply too few machines to cover a range of more than 10^{800} numbers. Another way to put the question is to determine the minimum value m such that there is no 5-state machine of productivity m. Certainly $bb(5) + 1$ is an upper bound on this value; the question is whether it is a lower bound as well. A related question is that of the largest continuous range of representable numbers: in other words, what is the largest value of m such that for every $bb(4) < k < m$ there is a 5-state machine of productivity k? In general, the distribution of platypus machines for 5-state and 6-state machines remains an intriguing question.

A further aspect of this question is the ability to produce an analytical answer. Certainly we can settle the question of the distribution of 5-state machines by enumerating all of them and examining the results. However, we see this as a necessary first step towards a systematic method by which one can construct an m-state machine of productivity n, where m satisfies a constraint such as $m \leq \log_2 n$. For example, if there is no 5-state machine with productivity 29, can we construct a 6-state or 7-state machine by some algorithmic process?

An extension of this line of thought is to consider equivalences across varying numbers of states. For example, consider the four machines in Figure6state, all of which have productivity 6.

In a sense, these machines are all equivalent. In addition, the similarity between the 4-state and 5-state machines suggests that it may be possible to reduce the 5-state machine to a 4-state one (and possibly further still to the 3-state one) by a process of state elimination. Such techniques are known for finite-state automata, and are not generally applicable to Turing machines. However, for this restricted class and in this particular context, it seems reasonable to investigate an approach along these lines.

6 Discussion

We have seen how examining the machines which lead to the busy beaver function have lead to some further interesting questions. A similar one that can be asked is what are the maximum productivities for some sub-classes of machines. In particular, what is the maximum productivity of a 5-state machine in which there is no transition with input 1 and output blank? (We call these machines *monotonic*, in that no 1 is ever erased from the tape). What is the maximum productivity for *eager* machines? (i.e. those with transitions whose output is always 1)? What is the maximum productivity for *contiguous* machines? (i.e. those which always maintain the 1's in a single string)? This latter problem is called the little busy beaver problem in (Yang,Ding &Xu 1997), but no values are given for it. All of these are seemingly natural classes for humans attempting to construct busy beaver machines by hand, but none of these properties holds for the six 5-state machines with the largest known productivities.

For that matter, more sophisticated measurements of the tape usage may be used, such as the largest string of contiguous 1's, the amount of tape used, or the distance moved from the point of origin. Holkner (Holkner 2004) has observed that several of the 6-state monster machines use no more than 10 distinct "blocks', i.e. repetitions of short strings. Hence whilst the overall length is very large, it has a very regular structure. Finding an appropriate metric for this structure is an intriguing question.

Whilst there have been some investigation of the busy beaver function for one-way tapes, other constraints on the use of the tape may be interesting. One possibility is to constrain the amount of tape that may be used, such as by using a circular tape, or by bounding the number of tape cells that can be written to, analogous to the difference between linear-bounded automata and Turing machines (Sudkamp 2005).

As noted above, the are some "maximising" strategies built into the search methods for busy beaver machines, such as searching only for halting up machines. A stronger restricion is the one that ensures that the halt transition can only be used once all other transitions have been used. Whilst this is appropriate for the busy beaver function, in that it seems reasonable to require that all possible transitions be used before terminating, it is possible that the the lack of necessity to use every transition may lead to some simpler placid platypus machines. The cost is, of course, a huge increase in the search space, making an analytical criterion for the detection of platypus machines more critical.

A final point to note is that the machines which do not terminate may also be of interest. In Conway's game of life, for example, some of the most interesting configurations are those which lead to self-reproducing behaviour, which, in our context, correspond to particular kinds of non-terminating machines. Hence dizzy ducks, which reproduce the same context but further along the tape, may be of interest in this sense. Potentially interesting also are *phoenix* machines, i.e. those which continually print some number of 1's, and then return the tape to all blanks in the initial state. Such a machine will eternally convert the blank tape into a fixed number of 1's and back again.

7 Conclusions and Further Work

We have seen how a quest for determining the value of $bb(5)$ can lead to a number of new questions about this class of Turing machines. In this sense, the main contribution of this paper is not some particular piece of technical progress, but to consider various new questions, which, we hope, are of more general interest. It is difficult to conceive of a practical application of busy beaver machines[4], but the fundamental simplicity of the machines together with the astounding sizes of some of the numbers involved suggest that there is something of significant interest under the surface. Once we have achieved a better analytical understanding of these 2-symbol machines, we may be able to better understand the even more astoud-

[4] Except, perhaps as a particularly bizarre form of a screen saver!

n	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
pp(n)	2	3	3	4	4	4	4	4	4	4	5	5	5	5	5	5	5
ww(n)	4	7	11	12	14	19	30	40	53	96	≤ 41	≤ 45	≤ 50	≤ 57	≤ 63	≤ 43	≤ 62

Figure 2: Lower Placid Platypus values

n	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	
pp(n)	5	5	5	5	5	5	5	5			5	5	5		5	5	(* these
ww(n)*	72	98	69	223	155	298	343	102			512	427	559		691	808	

are upper bounds only)

Figure 3: Higher Placid Platypus values

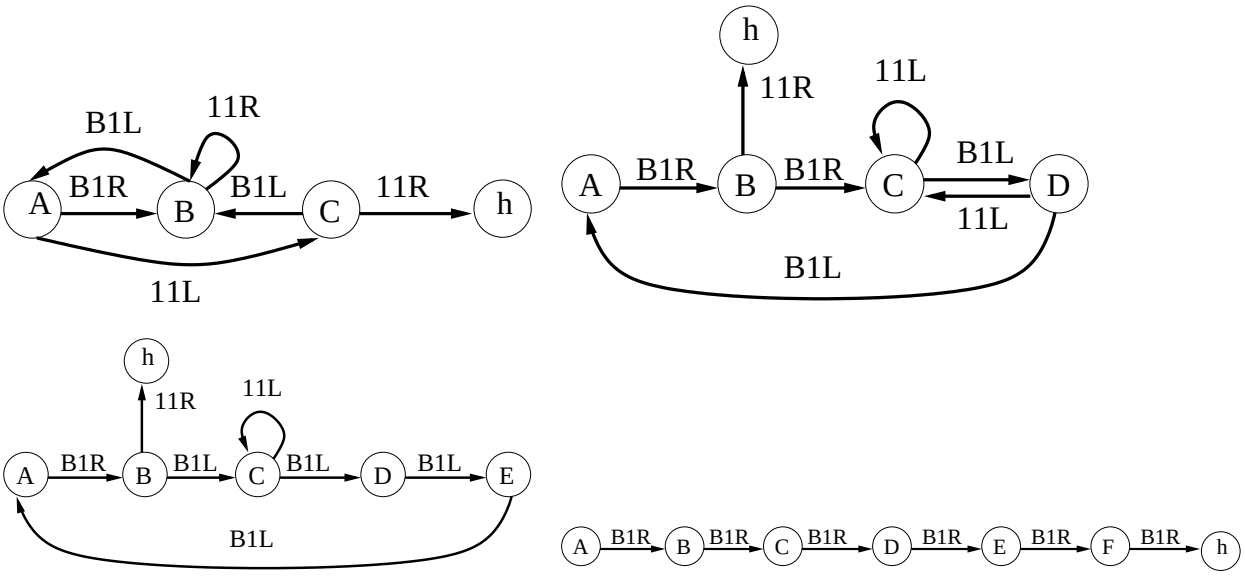


Figure 4: Machines of productivity 6

ing numbers being generated by the search for 3-, 4-, 5- and 6-symbol busy beavers.

Acknowledgements

The author is grateful to Jeanette Holkner, Michael Winikoff and Sandra Uitdenbogerd for assistance with zoological names and descriptions.

References

George Boolos and Richard Jeffrey, *Computability and Logic*, 2nd edition, Cambridge University Press, 1980.

Allen Brady, Busy Beaver Problem of Tibor Rado, `http://www.cse.unr.edu/~al/BusyBeaver.html`.

Allen Brady, *The Determination of the value of Rado's noncomputable function $\Sigma(k)$ for four-state Turing machines*, Mathematics of Computation 40(162): 647-665, 1983.

The Ciao Prolog Development System WWW Site, `http://clip.dia.fi.upm.es/Software/Ciao`.

A. Dewdney, *The (New) Turing Omnibus*, Computer Science Press, 1993.

Milton Green, A lower bound on Rado's sigma function for binary Turing machines, Proceedings of the Fifth Annual IEEE Symposium on Switching Circuit Theory and Logical Design 91-94, Princeton, November, 1964.

Heiner Marxen and Jürgen Buntrock, *Attacking the Busy Beaver 5*, Bulletin of the EATCS 40:247-251, February 1990.

Alex Holkner, *Acceleration Techniques for Busy Beaver Candidates*, in Gad Abraham and Benjamin I.P. Rubenstein (eds.), *Proceedings of the Second Australian Undergraduate Students' Computing Conference* 75-80, December, 2004. ISBN 0-975-71730-8. Available from `http://www.cs.berkeley.edu/~benr/publications/auscc04`.

B. Julstrom, *A Bound on the Shift Function in terms of the Busy Beaver Function*, ACM SIGACT News 23(3):100-106, 1992.

Owen Kellett, A Multi-Faceted Attack on the Busy Beaver Problem, Master's Thesis, Rensselaer Polytechnic Institute, August, 2005.

Wang Kewen and Shurun Xu, *New Relation Between The Shift Function and the Busy Beaver Function*, Chinese Journal of Advanced Software Research,2(2):192-197, 1995.

Shen Lin and Tibor Rado, *Computer Studies of Turing Machine Problems*, Journal of the Association for Computing Machinery 12(2):196-212, 1964.

Heiner Marxen, Busy Beaver web page, `http://www.drb.insel.de/~heiner/BB/index.html`.

Pascal Michel, Behavior of Busy Beavers, `http://www.logique.jussieu.fr/~michel/beh.html`.

R. Munafo, Large Numbers – Notes, `http://home.earthlink.net/~mrob/pub/math/ln-notes1.html`.

Tibor Rado, *On non-computable functions*, Bell System Technical Journal 41: 877-884, 1963.

Kyle Ross, *Use of Optimisation Techniques in Determining Values for the Quadruplorum Variants of Rado's Busy Beaver Function*, Masters thesis, Rensselaer Polytechnic Institute, 2003.

Georgi Georgiev, Busy Beaver Prover, `http://skelet.ludost.net/bb/index.html`.

Thomas Sudkamp, *Languages and Machines: An Introduction to the Theory of Computer Science*, (3rd ed.), Addison Wesley, 2005.

T. Walsh, *The Busy Beaver on a One-Way Infinite Tape*, ACM SIGACT News 14(1):38-43, 1982.

Ruiguang Yang, Longyun Ding and Shurun Xu, *Some Better Results Estimating the Shift Function in Terms of Busy Beaver Function*, ACM SIGACT News 28(1): 43-48, March, 1997.