Analysis of Busy Beaver Machines via Induction Proofs

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 problem is to find the maximum number of 1's that can be printed by an n-state Turing machine of a particular type. A critical step in the evaluation of this value is to determine whether or not a given n-state Turing machine halts. Whilst this is undecidable in general, it is known to be decidable for $n \leq 3$, and undecidable for $n \geq 19$. In particular, the decidability question is still open for $n = 4$ and $n = 5$. In this paper we discuss our evaluation techniques for busy beaver machines based on induction methods to show the non-termination of particular classes of machines. These are centred around the generation of inductive conjectures about the execution of the machine and the evaluation of these conjectures on a particular evaluation engine. Unlike previous approaches, our aim is not limited to reducing the search space to a size that can be checked by hand; we wish to eliminate hand analysis entirely, if possible, and to minimise it where we cannot. We describe our experiments for the $n = 4$ and $n = 5$ cases appropriate inductive conjectures.

1 Introduction

The busy beaver is a well-known example of a non-computable function. It was introduced by Rado[18] as a simple example of such functions, and is defined in terms of a particularly simple class of Turing machines [21]. This class of machines has a single tape, infinite in both directions, which is blank on input. There are only two tape symbols, 0 and 1, and the machine is required to be deterministic, i.e. that for any state and input symbol there is exactly one transition. Each machine also contain a special state known as the *halt* state, from which there are no transitions. A machine is said to have n states when it has one halt state and n other states. The busy beaver function for n is then defined as the largest number of 1's that can be printed by an n-state machine which halts. 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 grow faster than any computable function. Hence it is not only non-computable, it grows incredibly quickly. Accordingly, despite over 40 year's worth of exponential increases in hardware capabilities in line with Moore's famous law, its value has only been established with certainty for $n \leq 4$. The values for $n = 1, 2, 3$ were established by Lin and Rado [11] in the 1960s and the value for $n = 4$ by Brady in the 1970s [3]. Larger values have proved more troublesome [14, 13, 10, 20], and the lower bounds for $n = 6$ are already spectacularly large [13]. There are some interesting analyses of the current champion machines for the $n = 5$ and $n = 6$ cases [17, 16], but due to the sheer size of the numbers involved, $bb(n)$ for $n \geq 7$ may never be known.

The current state of knowledge is given in the table below. We denote by $ff(n)$ (for *frantic frog*) the maximum number of state transitions performed by a terminating Turing machine with n states.

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

There is some strong evidence that $bb(5) = 4,098$, and Kellett [10] has shown this for the quadruple version of 5-state Turing machines (i.e. those in which each transition can either change what is on the tape, or move the tape pointer, but not both). This is tantalisingly close, but in the absence of a proof of equivalence between this variant and the quintuple one (or more particularly a proof that for every 5-state quintuple machine there is a 5-state quadruple equivalent), we cannot state definitively that $bb(5) = 4,098$.

There are a number of other points of interest in this area, such as the *placid platypus* problem discussed in [8], and investigating busy beaver functions for machines with 3, 4, 5 and 6 symbols.

In this paper, we concentrate on providing a simple and extensible technique for establishing non-termination. In particular, we want to find a method which can be extended, as larger classes of machines generally require more sophisticated techniques than smaller ones, and one that maximises the possibilities for automated analysis. In other words, we want to have a decision procedure for all decidable cases of this problem.

Our technique is based on an analysis of execution history, which is used to produce inductive conjectures, which are then passed to a particular engine for evaluation. The extensibility of this techniques can thus be found in the ability to produce more inductive conjectures, with a corresponding increase in the complexity of the computations that can be performed by the engine.

This paper is organised as follows. In Section 2 we discuss various cases of non-terminating machines and how they may be detected, and in Section 3 we discuss our conjecture method in some detail. In Section 4 we discuss the hypothetical engine and in Section 5 we discuss our experiments with it. Finally in Section 6 we present our conclusion and some areas of further work.

2 Classification of Machines

There are a number of ways in which a given machine may fail to terminate, or to otherwise be of no interest for the search for busy beaver machines. The machine generation process is based on the *tree normal form* method [11, 3], in which a partially complete machine is executed until it reaches a transition which is not yet defined. Some constraints are applied at this point, to ensure that only "sensible" machines are generated. These constraints include not allocating the halt transition until all other transitions are defined, ensuring that no state is isolated (i.e. unreachable from any other state) and performing some simple checks for loops. We refer to all machines pruned in this generation process as *ignoble iguanas.* These include both machines which we can determine not to terminate, as well as some others which may terminate, but which will have no bearing on the busy beaver or placid platypus problem. One example is a machine which returns the tape to all blanks at some point in the computation (we refer to such a machine as a *phlegmatic phoenix*[1]). This machine may terminate, but as far as the calculation of its productivity is concerned, the computation performed up to this point is wasted. In particular, there is another machine in the search space which will perform whatever computation takes place after this point without this initial loop. By building checks such as these into the generation process, we eliminate a number of cases in which we generate machines which are then quickly shown to be irrelevant.

Having generated and stored a particular machine, we then attempt to classify it as either a terminating machine, or as one of a variety of non-terminating machines.

[1]A phlegmatic phoenix is also an ignoble iguana. Thus, our inheritance relation is thoroughly counterintuitive from a zoological perspective.

One obvious class of non-terminating machines are those which repeat exactly the same tape configuration. We refer to these machines as *perennial pigeons.* These machines are comparatively rare in our search, as they can be largely eliminated by the generation process.

Another class of non-terminating machines are those which move infinitely in only one direction. We refer to these machines as *road runners.* Again these are largely eliminated in the generation process.

A further class of non-terminating machines are those which move in one direction, but in a cyclic process involving movement in both directions. Essentially this involves re-creating the same tape sub-section and shifting it in one direction. For example, consider Machine 1 in Figure 1.

This machine has the following execution trace. Note that the notation $1\{B\}0$ denotes that the machine in state B with the tape head pointing at the 0).

```
     {A}0
      1{B}0
      {A}10
     {A}010
      1{B}10
     {C}110
    {C}0110
   {D}00110
  {A}010110
   1{B}10110
  {C}110110
 {C}0110110
{D}00110110
....
```

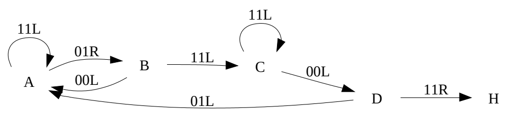
This machine repeatedly moves one step to the right and then four to the left. We can see this by noting that from the configuration $\{A\}01$ it never moves past the 1, and has a net movement to the left which re-recreates this configuration three places to the left, i.e. $\{A\}01001$.

Brady [3] refers to these machines as "travelling loops"; following the literary lead of the busy beaver, we refer to them as *dizzy ducks.*

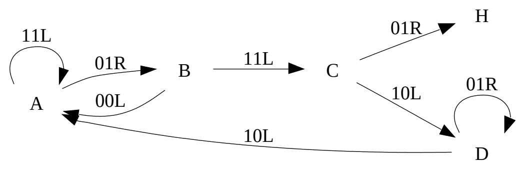
A fourth class of non-terminating machines are those for which the halt transition cannot be executed. For example, consider Machine 2 in Figure 1.

Now in order to reach the halt state, this machine must be in state C with input 0. The only way in which this configuration could be reached is to previously be in state B with input 1, which in turn can only arise from state A with input 0. Hence the only way to reach the halt state is to execute the sequence $\{A\}01 \rightarrow 1\{B\}1 \rightarrow \{C\}11$ However, this clearly cannot reach the halt state, as by the time we enter state C, the input must be 1, not 0. Hence we can show that this machine will never terminate by showing that the halt state is unreachable. We refer to this class of machines as *meandering meerkats.*

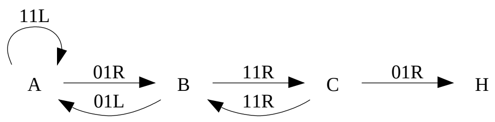
None of these machine types are particularly difficult to detect. Ignoble iguanas are pruned in the generation process (see Section 5), and some simple checks when executing a machine suffices



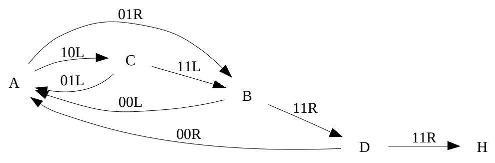
Machine 1: dizzy duck



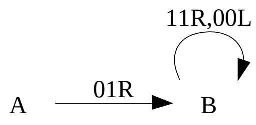
Machine 2: meandering meerkat



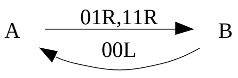
Machine 3: Christmas tree



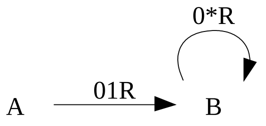
Machine 4: phlegmatic phoenix



Machine 5: ignoble iguana



Machine 6: perennial pigeon



Machine 7: road runner

Figure 1: Some basic machine types

to detect perennial pigeons, phlegmatic phoenices and road runners (see examples of these in Figure 1). Dizzy ducks require a little more care, as at least some part of the history of execution of the machine is required. The way we have implemented the test for these is to execute the machine for a given length of time (say 100 steps), extract the history of execution and examine it for patterns of this form. As well shall see, this fits in well with our detection methods for the remaining machines.

3 Inductive Proofs of Non-termination

One of the classic cases for non-termination is that of *Christmas trees*, as coined by Lin and Rado [11]. These machines basically continually add a number of 1's (or some other string) to each end of the tape. For example, consider Machine 3 of Figure 1.

The execution of this machine proceeds as follows.

```
  {A}0
  1{B}0
  {A}11
 {A}011
 1{B}11
 11{C}1
 111{B}0
 11{A}11
 1{A}111
 {A}1111
{A}01111
1{B}1111
11{C}111
111{B}11
```

```
  1111{C}1
  11111{B}0
  1111{A}11
  111{A}111
  11{A}1111
  1{A}11111
  {A}111111
 {A}0111111
 1{B}111111
 11{C}11111
 111{B}1111
 1111{C}111
 11111{B}11
 111111{C}1
 1111111{B}0
 111111{A}11
 11111{A}111
 1111{A}1111
 111{A}11111
 11{A}111111
 1{A}1111111
 {A}11111111
{A}011111111
1{B}11111111
11{C}1111111
111{B}111111
1111{C}11111
11111{B}1111
111111{C}111
1111111{B}11
11111111{C}1
111111111{B}0
11111111{A}11
...
```

It is clear by looking at the trace that this machine endlessly adds 1 to either end of the tape. Whilst Lin and Rado [11], Brady [3] and Kellett [10] have all given methods for recognising Christmas trees and similar machines, they are all based on recognising particular structures in the machine, such as a particular configuration of transitions. Clearly this an appropriate method, but the difficulty lies in knowing what cases remain uncovered by such analyses. Moreover there is something intuitively appealing about looking directly at the behaviour of the machine itself and using the trace as the basis for analysis. Our approach is to examine the execution history, in much the same manner as a human would when looking at the above trace, to formulate some appropriate conjectures about the pattern of behaviour observed, and to then evaluate these conjectures on a particular execution engine. This differs from a standard Turing machine emulator in that we commence execution with a particular tape pattern and attempt to execute the machine until this pattern recurs. If so, we have established that the machine does not terminate.

This processes appears to aggregate several heuristics together, as well as provide a common basis for dizzy duck detection and the pattern-based non-termination proofs. In addition, as the analyses of Michel [16] and the techniques of Holkner[9] have shown, similar methods will be required to establish termination for at least some 5-state and 6-state machines.

For example, consider the machine above. From looking at the trace, we can determine that the following sequence occurs:

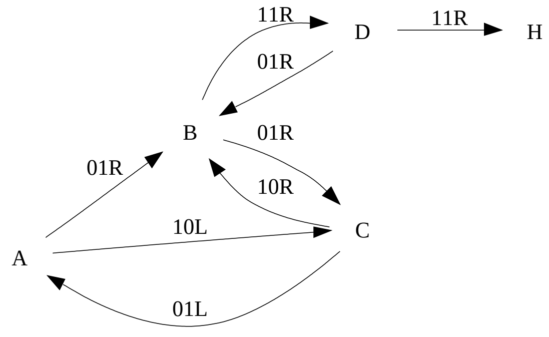
$11\{C\}1 \rightarrow 11\{C\}111 \rightarrow 11\{C\}11111$

We then form the conjecture that the pattern $11\{C\}1(11)^m$ occurs infinitely often in the trace, i.e. once for every non-negative integer m. We know that as $11\{C\}1$ is in the trace that this pattern occurs at least once. In order to complete the proof, we need to show that if the machine commences in a state of the form $11\{C\}1(11)^m$ for any m, then the computation will eventually reach a state of form $11\{C\}1(11)^{m+1}$, which we can write as $11\{C\}111(11)^m$. Performing this computation is the job of what we term the *hypothetical engine*. This engine is discussed in more detail in Section 4.

Naturally there can be more than one such pattern to be found in the trace, but clearly one such proof suffices to establish non-termination. All proofs of non-termination can be thought of as particular methods for showing that the halt transition is never executed. This particular technique is based on showing that once a particular state and pattern on the tape is reached, it recurs infinitely often.

Hence, once we have the execution trace, we search it firstly for a combination of state and input symbol which is repeated in the trace (such as state C and input 1, or state A and input 0 in the above example). We then examine the configurations in the trace which contain this combination, and look for what we call a *progression*. This can be thought of as a pair of configurations which conform to the same pattern, but for which the earlier configuration is a simpler form of the later one. For example, in the above trace we note that the first two occurrences of state C with input 1 are $11\{C\}1$ and $11\{C\}111$. As the first configuration is a sub-pattern of the second, we find that there is a progression. In more complicated cases, we look at occurrence counts for particular strings. For example, given the two configurations $11\{A\}0011$ and $111\{A\}0010011$, by representing these as $1^2\{A\}(001)^1 1^1$ and $1^3\{A\}(001)^2 1^1$ respectively, we find that there is a progression, as the basic patterns of $1^*\{A\}(001)^* 1^*$ are the same, and the number of occurrences in the second configuration are all at least as large as those in the first. Finally, before accepting this as a hypothesis, we insist that there be at least three instances of the pattern occurring in the trace. In the Christmas tree example above, we note that the configurations $11\{C\}1$, $11\{C\}111$, $11\{C\}11111$ and $11\{C\}1111111$ all occur in the trace, and so this requirement is met. In the latter case, we would require that $1111\{A\}0010010011$ also be present in the trace before accepting this pattern as a hypothesis.

The reason for this requirement is that there are some machines for which a progression is not merely the repeated addition of a particular string. In particular, there are some for which the progression is *multiplicative* rather than *additive*. For example, consider the machine below.



Here we find the configurations $1^6\{D\}0$ and $1^{18}\{D\}0$. Treating these as additive progressions would result in the hypothesis $1^6(1^{12})^m\{D\}0$, predicting the third occurrence as $1^{30}\{D\}0$. However, the third instance in the trace is in fact $1^{42}\{D\}0$, thus showing that this is not an additive progression. When treated as a multiplicative progression, we find that the next occurrence after $1^n\{D\}0$ is predicted to be $1^{2n+6}\{D\}0$. This is confirmed by inspection of the trace, in which the fourth occurrence is found to be $1^{90}\{D\}0$.

Hence there are some 4-state machines which cannot be handled by additive progressions alone. Whilst making multiplicative hypotheses does not seem overly difficult, the current implementation of the hypothetical engine does not support multiplicative hypotheses. We are currently working on ways to extend the engine to include such hypotheses. We refer to machines that require this form of induction as *killer kangaroos*.

It should also be noted that it seems sensible to use the heuristic that the first state tried in this process is the one which contains the halt transition. We refer to this state as the *pivotal* state. For example, in the machine above, state D with input 1 will execute the halt transition, and hence D is the pivotal state. Hence we first try to find a progression for state D with input 0. If such a progression cannot be found, possibly because the machine never enters the pivotal state or does so only once, we then proceed to consider other states.

It is worth keeping in mind that for a fixed finite tape length, a non-terminating machine will either remain within this tape length and repeat a configuration, or move outside one or both of the tape length's boundaries. One can think of the dizzy duck detection technique as capturing those machines which cross only one of these boundaries; the methods based on progressions are used to analyse those which cross both. Hence a typical trace for these methods will involve moving from one end of the tape to the other, and possibly several times.

Hence our main points of difference from other approaches can be summarised as:

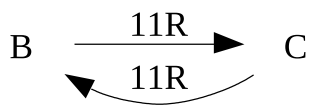
- The use of execution history as a basis for non-termination conjectures
- The use of an inductive execution engine to establish the truth of the non-termination conjectures (where possible)
- The aim to automate as much of the analysis as possible

The third point is worthy of some further discussion.

Lin and Rado, Brady, and Kellett all use computational methods to reduce the search space, leaving some class of holdouts, i.e. some number of machines which need to be analysed by hand (although Brady does state that his automated analysis could correctly analyse all the 40 holdouts found by Lin and Rado). In terms of determining the busy beaver function, this is entirely appropriate. In our case, we wish to not only explore this question, but several others, such as the placid platypus function. In addition, the decidability of the termination problem for this class of machines for particular values of n is not yet complete. It is known that termination is decidable for $n \leq 3$ [12], and that it is undecidable for $n \geq 19$ [12], as evidenced by a 19-state universal Turing machine. To the best of the author's knowledge, the decidability of the cases for $n = 4$ and $n = 5$ is still open, although Michel [15] has argued that the $n = 5$ case is likely to be undecidable, due to the similarities between his analysis of the 5-state machines of high productivity and the Collatz sequence.

4 The Hypothetical Engine

The main question for the hypothetical execution engine is how to deal with tape strings like $1(11)^n$ where n is a variable. When n is a constant, the techniques of Marxen and Buntrock[14], Holkner[9] and others are directly applicable. Roughly speaking, when n is a variable, we rely on finding cycles in the machine. For example, consider the machine below and the configuration $\{B\}(11)^m010$.



Given that for any input X we have that $\{B\}11X \to 1\{C\}1X \to 11\{B\}X$, we can then deduce that from the configuration $\{B\}(11)^m010$ we can get to $(11)^m\{B\}010$. We can then compute "as normal" until we encounter further variables.

Hence we need to determine what transition to make when faced with a configuration of the form $L\ \{S\}I^N\ R$.[2] There are three cases:

1. Computation commencing with the configuration $\{S\}I$ remains within the tape segment containing I until it exits from the right hand end of the tape segment in state S. Hence $\{S\}IX \to NewI\{S\}X$ for any input symbol X. Accordingly, any tape configuration of the form $L\ \{S\}I^N\ R$ will result in the configuration $L\ (NewI)^N\{S\}\ R$.

   Because this is independent of the contents of L and R, this transformation can be thought of as *context-free.* Continuing our linguistic theme, we refer to this case as the *wild wombat.*

2. Computation commencing with the configuration $\{S\}I$ exits from the left hand end of the tape segment containing I. In this case we continue the computation, until either we exit the right hand side in state S, or we exceed some bound limit. In the latter case, this proof attempt then fails. In the former case, we then have a context-sensitive transformation of the following form:

   $L\{S\}IX \to NewL\{S\}X$

   In order to repeatedly apply this transformation for the configuration $L\{S\}I^NR$, we require that $L = L_2.L_1$ and $NewL = L_2.L_3.L_1$ so that we have

   $L_2.L_1\{S\}IX \to L_2.L_3.L_1\{S\}X$

   In other words, we divide L into two parts: one that is unchanged in the computation (L_2) and one that is changed (L_1). Either of these may be empty. Then in order to repeatedly apply the same transformation, we require that the rightmost characters of $NewL$ be the same as L_1; otherwise, this proof attempt fails. If these conditions are satisfied, we then have the following transformation:

   $L_2.L_1\{S\}I^NR \to L_2.(L_3)^N.L_1\{S\}R$

[2] Similar remarks apply, of course, when the tape pointer is at the right hand end of I^N. Without loss of generality we discuss only the left hand case here.

Because this is dependent of the contents of L, this transformation can be thought of as *context-sensitive.* We refer to this case as the *slithery snake.*

3. Computation commencing with the configuration $\S\}I$ exits from the right hand end of the tape segment containing I, but not in state S. In this case, we also proceed with the computation until it returns to this precise position in state S, or we exceed some bound limit. In the latter case, this proof attempt fails. Otherwise, we have the problem of what symbols the computation should proceed with. There are two possibilities:

   (a) "Switching" a string from elsewhere

   (b) Splitting into cases

   For the first case, consider the configuration $\{S\}(11)^N110011$. Now as $(11)^N11$ and $11(11)^N$ represent the same strings, we can switch from this configuration to the equivalent one $\{S\}11(11)^N0011$, from which we can then clearly proceed. In the second case, we can do something similar, but by splitting this case up into one where $N = 0$ and one where $N > 0$. In our example, this means we have to handle the cases for $\{S\}110011$ and $\{S\}11(11)^N0011$. This latter configuration we can compute with, as above. In the former case, it seems natural to proceed from this point to the appropriate instance of the target case. However, in some experiments we found that this point in the computation had in some cases already passed the zero instance of the target case. Hence we find the zero instance of the initial configuration and then proceed from there to (hopefully) the zero instance of the target configuration.

   In either case, we have that

   L $\{S\}II^NR \rightarrow NewL\{S\}I^NNewR.$

   As in the previous case, to be able to repeatedly apply this transformation, we require $L = L_2.L_1$ and $NewL = L_2.L_3.L_1$ and $R = NewR$. Hence we have that

   $L_2.L_1\{S\}II^NR \rightarrow L_2.L_3.L_1\{S\}I^NR$

   from which we deduce that

   $L_2.L_1\{S\}II^NR \rightarrow L_2.(L_3)^N.L_1\{S\}IR$

   We refer to this case as the *maniacal monkey.*

These three techniques have been implemented in the hypothetical engine. As we have seen, there is a need to extend this to the evaluation of multiplicative inductive conjectures as well.

5 The Blue Bilby, Ebony Elephant and White Whale

We have implemented the above conjecture generator and hypothetical engine. The current implementation is around 5,000 lines of Ciao Prolog [5]. This code is available from the author's website at `www.cs.rmit.edu.au/~jah/busybeaver`. This implementation follows the *tree normal form* method of generating machines, i.e. evaluating a partially complete machine until an unallocated transition is found, allocating it a value and then performing various checks on the resulting machine. If the test fails, an alternative allocation is sought. Otherwise, computation then proceeds with the extended machine. This process continues until only the halt transition remains to be allocated, at which point the machine generation is complete.

We then have to classify the machine as either terminating or non-terminating. As noted above, the checks in the generation process catch most or all of the phlegmatic phoenices, perennial pigeons, and road runners. Hence we test for ignoble iguanas (basically a variety of simple loop tests), and for meandering meerkats, which is done by running the machine backwards from the halt state for up to a bounded number of transitions. If we end up in a dead end, the halt transition is unreachable and the machine is a meandering meerkat. Otherwise, the test fails and we proceed to the next case.

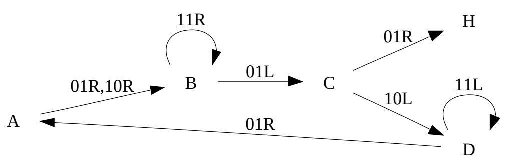
We then emulate the machine for a fixed number of steps. This provides not only a means of proving termination by emulating the machine and showing it reaches the halt state, but also a record of the execution history, which can then be mined for dizzy ducks and used by the above inductive methods.

The results of our experiments to date can be found in Table 5. For cultural consistency with other alliterative animals, we refer to machines which terminate as *terminating termites.* In the case for $n = 3$, we found that there were 2,112 machines to be analysed after the tree normal form generation.[3] Of these, only 26 required the inductive methods; the rest either terminated or were classified as dizzy ducks, meandering meerkats, road runners, phlegmatic phoenices, or perennial pigeons. Of the 26 requiring induction, 24 required only the wild wombat case, 2 required the slithery snake, and none required the maniacal monkey.

In the case for $n = 4$, we found that there were 350,440 machines to be classified. Of these, 5,112 were found to require induction, with 61 machines remaining unclassified at the time of writing. Of the remaining 61, 41 were killer kangaroos, i.e. requiring multiplicative induction hypotheses, and a corresponding addition to the hypothetical engine. We are actively pursuing both of these extensions. The remaining 21 cases do not require multiplicative hypotheses, but do require a more sophisticated means of evaluation. For example, consider the machine below.

[3] We denote the search for beaver machines with 3,4, and5 states as the quest for the *blue bilby*, *ebony elephant*, and the *white whale* respectively. We refer to the search for 6 state machines as the quest for the *demon duck of doom*, a term used to describe an Australian Thunderbird from the megafauna era thousands of years ago (`http://www.abc.net.au/science/features/megafauna/default.htm`).

States	Machines	Termites	Loops	Ducks	Wombats	Snakes	Monkeys	Kangaroos	Unclassified
2	11	7	2	2	0	0	0	0	0
3	2,112	903	436	747	24	2	0	0	0
4	350,440	134,048	84,482	126,735	3,292	1,803	18	41	21
5									



Execution of this machine produces the following trace of the pivotal state:

```
     {c}11
    11{c}11
   10{c}111
   1111{c}11
  1011{c}111
  110{c}1111
  111111{c}11
 101111{c}111
 11011{c}1111
 1110{c}11111
 11111111{c}11
```

From the trace it is straightforward to see that a reasonable (additive) hypothesis is $(11)^m c11$, which has four instances in the trace. The difficulty is that between occurrences of this pattern there are transitions of the form $1^n 01^k c 1^m$ to $1^{n+1} 01^{k-2} c 1^{m+1}$. Hence such machines require a "secondary" induction in addition to that already present in the engine. We are also actively this extension to the engine. We refer to these machines as *addictive adders*.

The search for $n = 5$ is much less complete. However, it is anticipated that widespread use of the multiplicative inductive method will be required, and presumably some stronger methods still. A summary of our results to date is in the table below. The *Machines* column is the number of machines after the generation process, which eliminates many simple kinds of loop. The *Loops* column records some easily detected loops, such as meandering meerkats, but does not include dizzy ducks or any of the machines counted in other columns in the table.

Whilst our search for the ebony elephant is still incomplete, we believe that our method has shown itself to be useful. The most direct point of comparison is with the work of Brady [3]. His automated analysis (which was undertaken in the 1970s with much less powerful equipment) left 218 cases unclassified (or "holdouts") and used a number of different heuristics. We believe our method is conceptually simple and can be readily extended to further cases (such as the multiplicative hypotheses discussed above). It is also leaves fewer cases unclassified (although Brady did not attempt to minimise the 218 unclassified cases; his aim was to reduce the search space to a number manageable by hand rather than to maximise the number of cases analyses automatically). It is interesting to observe the behaviour of our methods on the 6 classified cases and 3 unclassified cases described in [3]. Not only does our method show the non-termination of all 6 classified case, it is able to show that 2 of the 3 holdouts also do not terminate. The final machine is a killer kangaroo, and hence requires multiplicative induction.

The strength of the inductive conjectures required for each class of machines is also of interest. The additive hypotheses used here are basically a particular class of regular expressions. We think it is particularly interesting that a small number of 4-state machines require inductive hypotheses of greater strength, similar to the way in which there were 2 3-state machines requiring the slithery snake technique rather than the wild wombat case alone. A similar phenomenon is observed with the 5-state busy beaver candidates, i.e. machines with high productivities, in that there are 8 machines out of a total of 69,471,096 with productivities of more than 1,000. We refer to this as the *handful of holdouts* phenomenon.

6 Conclusions and Further Work

We have seen how an inductive technique based on the history of execution can be used as a means of proving non-termination. This technique is conceptually simple and readily extensible, and has the potential to form a basis for techniques to prove the termination of machines, as well as their non-termination. We believe that this technique will be an important tool in the quest for the demon duck of doom (but it will be by no means the only one).

We have seen how this technique can be used to reduce the number of holdouts for the $n = 4$ case. Given that the decidability of the $n = 4$ case is still open, it is entirely possible that we can reduce this number to 0. It is also intriguing to note that whilst the largest number of transitions used in a terminating computation in a 4-state machine is 107, it often takes significantly more steps than that to establish non-termination. For example, there some 4-state dizzy ducks for which the pattern is only detected after 393 steps. In our experiments we used a maximum of 750 steps. However, it is not yet clear if this bound needs to be increased or not.

An interesting correspondence was pointed out by Brady [4] between the 4-state 2-symbol case and the 2-state 4-symbol case. Given that the number of transitions in each type of machine is the same, a natural question is whether it is possible to translate machines from one type to the other (or possibly for a sub-class of machines).

We have also seen how additive induction hypotheses (which are basically a sub-class of regular expressions) can be used with the current hypothetical engine to classify all 3-state machines and almost all 4-state machines. We have also seen how multiplicative conjectures and a correspondingly more sophisticated engine will be required to reduce the 4-state holdouts further. A question of interest is then the precise level of inductive strength that is required for the 5-state and 6-state machines.

References

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

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

[3] 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.

[4] Allen Brady, private communication, August, 2006.

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

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

[7] 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.

[8] James Harland, *The Busy Beaver, the Placid Platypus and Other Crazy Creatures*, Proceedings of Computing: the Australasian Theory Symposium (CATS'06), Hobart, January, 2006. Published as Volume 51 - Theory of Computation 2006 of the ACS Conferences in Research and Practice in Information Technology (CRPIT) series.

[9] 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`.

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

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

[12] Maurice Margenstern, *Frontier between Decidability and Undecidability: A Survey*, Theoretical Computer Science 231(2):217-251, January 2000.

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

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

[15] Pascal Michel, *Busy beaver competition and Collatz-like problems*, Archive for Mathematical Logic 32 (5) 1993, 351-367.

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

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

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

[19] 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.

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

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