Attacking the Busy Beaver 5
New Non-halt behaviors from the BB(5) Search Space

Owen Kellett

April 29, 2004

Abstract

While provably Turing-unsolvable, the Busy Beaver problem continues to be an intriguing exercise in computer science theory. Despite its unsolvable nature, attempting to determine results for small values of n still offers significant insight on the nature of Turing Machine behavior in general. Previous research done by Kellett and Ross[1] on the quadruple formulation of the problem combined tree normalization search techniques with specific non-halt detection routines to explicitly confirm the value of the Busy Beaver problem up through $n = 4$. For $n = 5$, however, these efforts left 98 machines out of the 1.67×10^{13} in the search space which could not be eliminated with tree-normalization, could not be categorized as one of the specificly tested non-halting behaviors, and did not halt after a pre-determined step limit. The present paper focuses on establishing new non-halt behaviors for these machines in an effort to complete the proof of the value of the Busy Beaver function for $n = 5$.

1 Introduction

For a more thorough background of the origin of these 98 machines, we refer the reader to the aforementioned work done by Kellett and Ross in establishing their results thus far. Nevertheless, we briefly summarize the problem, their approach, and their results before attacking the machines in question.

1.1 The Busy Beaver Problem

Defined by Tibor Rado in 1962, the Busy Beaver problem uses the notion of the "most productive" Turing machine out of a particular set of Turing machines. More specifically, we concentrate on "productivity" as being defined as the number of 1's printed on the tape after a machine halts when being started on an infinitely blank tape. If a machine never halts when given this input, it is given a productivity value of zero. Rado therefore defines BB(n) as the productivity of the most productive Turing machine that has n states.

Given this definition, it is easy to define different variations of this problem in terms of different formulations of Turing machines and certain output configurations on the tape. For our purposes, we shall only consider Turing machines of the quadruple formulation variety where transitions can perform a write action or a move action but not both.

1.2 Attack Strategy

A number of different strategies have been employed in attempting to find candidate machines for the Busy Beaver problem. These include brute force searches, genetic algorithms, and behavior heuristics. However, some of these strategies can never hope to explicitly prove the value of BB(n) (any type of heuristic strategy), while still others not only suffer from the same pitfall, but also are not practical given the enormity of the search space[2] for even small n (brute force).

Given this knowledge, Kellett and Ross chose to develop a tree-normalization search strategy that maintained completeness of the search, while vastly reducing the search space. In addition, they developed specific non-halt detection routines that could prove whether or not a particular machine exhibits a particular known non-halting behavior. Through the combination of these techniques, they were able to classify every single machine in the search tree as either 1) pruned from the tree and not considered because it is behaviorally equivalent to some other machine in the tree (tree normalization optimizations), 2) a non-halter and therefore given a productivity value of zero (non-halt detection routines), or 3) a halter in which

[1] http://www.cs.rpi.edu/ kelleo/busybeaver

[2] Without going into the details, the search space for the Busy Beaver problem grows exponentially as n increases. For $n = 5$, for example, the search space is roughly 1.67×10^{13} Turing machines

case its productivity can be easily determined. They were able to do this for up to $n = 4$ and therefore have proven the values of the quadruple formulation of the Busy Beaver problem up through this value.

For $n = 5$ and beyond, however, the scope of their non-halt detection algorithms does not fully cover the search space. For BB(5) and BB(6), 98 and 42166 machines are respectively left unaccounted for. The 98 machines in the BB(5) problem are the focus of the present paper.

1.3 Established Non-halt behaviors

We very briefly outline the non-halting detection behaviors that were used by Kellett and Ross to establish their results. Intuitively, many of the behaviors exhibited by the 98 holdouts are minor variations of those described below. For a more detailed overview of these behaviors, please refer to the aforementioned research.

- **Backtracking** : The backtracking algorithm proves that a machine is a non-halter by demonstrating that it can never reach a set of conditions in which it does, or could potentially halt. It "backtracks" from all possible halting states and undefined transitions. If it is not possible to reach any of these situations, then it is a non-halter.
- **Subset loops** : Consider a machine with a certain subset of states where all possible transitions from these states are defined as a transition to some other state in this subset. If none of these states is the halt state, and the machine enters this subset of states at some point during execution, than this machine is a non-halter.
- **Simple loops** : A simple loop is one that moves in a generally rightward or leftward direction in some infinite repeatable pattern.
- **Christmas trees** : Christmas tree Turing machines sweep back and forth across the tape in a provably repeatable fashion. Hence their execution sequence over time takes on the appearance of a Christmas tree.
- **Multi-sweep Christmas trees** : Multi-sweep Christmas trees are much like Christmas trees, except they take more than one sweep across the tape before exhibiting their repeatable nature.
- **Leaning Christmas trees** : Leaning Christmas trees are again very similar to Christmas trees. The main difference is that they do not continue to push out their left and right extremum of each sweep in a symmetric manner. Instead, the repeatable portion of the tape is transposed to the left or to the right on each sweep. Thus their execution sequence over time resembles a Christmas tree that is "leaning" in one direction.
- **Counters** : Counters mimic binary counters by transforming the tape in a way such that at certain milestones, the tape configuration is representative of the next sequential binary number. Thus they count to infinity and never halt.

2 The Holdouts

Most of the behaviors described above are described more specifically in the aforementioned research as sets of concrete transitions that transform predefined localized portions of the tape into certain configurations. If it can be demonstrated that the set of transitions will repeat infinitely, and that a particular machine transforms the tape in a manner identical to each of the transitions in the set, than the machine can be deemed a non-halter without question. However, the immediately clear difficulty with this approach lies in identifying the components that are defined in the set of transitions. We therefore begin our analysis of the 98 holdouts with a subset that can be identified as exhibiting one of the behaviors defined above: Leaning Christmas trees. These machines escaped the current detection routine because of certain difficulties with the problem just described.

2.1 Leaning Christmas Trees

Let us first formally define the notion of a leaning Christmas Tree. A Turing Machine M is a *Leaning Christmas tree* if either M or its mirror M^c satisfy the following conditions for some state s:

1. There are nonempty words C, N, U, V, and X such that the tape configuration at some time is $0^*[C][U][V_s]0^*$, and at some later time is $0^*[C][N][U][X][V_s]0^*$.
2. The following conversions hold, where X, X', Y, Y', Z, V, V', V'', U, and U' are nonempty words and q and r are states (the symbol $\Rightarrow$ means that M transforms the left-hand side into the right-hand side after some number of steps):
    - $[X][V_s]0^* \Rightarrow_q [X'][V']0^*$
    - $[X_q][X'] \Rightarrow_q [X'][Y]$
    - $[N][U_q][X'] \Rightarrow [N][N][U'][Y']_r$

- $[Y'][_rY] \Rightarrow [Z][Y']_r$
- $[Y'][_rV'] \Rightarrow [Z][V''_s]$

3. $[N]^{i+1}[U'][Z]^i[V''] = [N]^{i+1}[U][X]^{i+1}[V]$ for all $i \geq 1$.

This definition, while complex, is also relatively straight-forward. With an accurate identification of the required words, it is easy to demonstrate whether or not a machine transforms the tape exactly according to the transitions defined above. However, given just an arbitrary Turing machine, how does one go about identifying these components? The general strategy used in the current implementation is as follows:

1. Run the machine for an arbitrary number of steps to establish a sweeping motion and account for any startup effects that may be out of line with the transitions defined.

2. Determine the step counters of the points in execution where the machine reaches its right extremum, or in other words when the configuration would be $0^*[C][N]^i[U][X]^i[V_s]0^*$ for some i. Find these step counters for the next five extrema following the initial startup of the machine.

3. Use the tape configurations at each extremum and compare them to extract the components. For example comparing the first two extremum reveals the C, N, X, U, and V components by examining the differences between the two configurations.

4. Without going into too many of the specifics, the rest of the tape components can be derived by attempting to perform the above transitions at the corresponding points where they should appear, and extracting the additional components from these procedures. If at any point this fails, then the entire proof fails and the machine cannot be classified as a leaning Christmas tree.

The main problem in this approach lies in the determination of the right extrema in step 2. Because left extremum of each sweep is not continually pushed outward like in a standard Christmas tree, and instead indeterminately lies to the *right* of the previous left extremum, it is extremely difficult to determine when these right extremum occur. This is especially true in machines that exhibit both leftward and rightward minor motions during one major sweep in one direction across the tape. As a result, the current implementation does not always properly identify these step counters in some machines that are in fact leaning Christmas trees.

While the automated process is clearly difficult, identifying the components by hand is time consuming but possible. After a thorough examination of the 98 holdouts, it was determined that 10 of them $(0, 1, 3, 9, 12, 13, 14, 32, 88, 95)$ are leaning Christmas trees. A description of each of them along with the correctly identified words can be found in the "annotatedRuns/" directory of the included package.

2.2 Counter Variations

As previously noted, a more thorough description of the already established non-halting behaviors can be found in the work already done by Kellett and Ross. We now particularly encourage the reader to become familiar with the counter non-halting behavior. We examine some particular variations of the counter behavior that can be found among the remaining holdouts.

2.2.1 Ordinary Counters

For easier reference, we include the definition of ordinary counters based on that as described by Kellett and Ross in the aforementioned research:

1. For machine M or its mirror M^c, there are nonempty words E, A, B, T, and Z. In the context of a binary counter, E represents the end cell that is used as a checkpoint. A and B are used to respectively represent the values 0 and 1 of a binary number. T is a transitory word that occurs during the incrementing of the counter in between checkpoints. Finally, Z is a blank word which consists of all 0's and is of the same length as A, B, and T.

2. At some point during execution, the machine reaches the following configuration where c is some state: $0^*[E][_cZ][Z]^*$.

3. The following transitions hold where r is also some state:

   - $[_cA] \Rightarrow_r [B]$
   - $[_cB] \Rightarrow [T]_c$
   - $[_cZ] \Rightarrow_r [B]$
   - $[T_r] \Rightarrow_r [A]$
   - $[E_r] \Rightarrow [E]_c$

2.2.2 Base 3 Counters

Intuitively, base 3 counters are extremely similar to the already established counter behavior except for the one obvious difference. Instead of counting in representative binary notation, they count in base 3. For completeness, we outline the full set of requirements for a machine to be considered a base 3 counter. Unsurprisingly, it is extremely similar to the original counter specification shown above.

1. For machine M or its mirror M^c, there are nonempty words E, A, B, C, T, and Z. In the context of a base 3 counter, E represents the end cell that is used as the checkpoint. A, B, and C are used to represent the values 0, 1, and 2 respectively in terms of a base 3 number. T is a transitory word that occurs during the incrementing of the representative number in between checkpoints. Finally, Z is a blank word which consists of all 0's and is of the same length as the A, B, C, and T words.
2. At some point during execution, the machine reaches the following configuration where c is some state: $0^*[E][_cZ][Z]^*$.
3. The following transitions hold where r is also some state:
   - $[_cA] \Rightarrow_r [B]$
   - $[_cB] \Rightarrow_r [C]$
   - $[_cC] \Rightarrow [T]_c$
   - $[_cZ] \Rightarrow_r [B]$
   - $[T_r] \Rightarrow_r [A]$
   - $[E_r] \Rightarrow [E]_c$

This definition can clearly be extended for base 4, base 5, etc. counters by adding additional components and transitions similar to those involved in the transformation of the base 2 specification to base 3.

2.2.3 Alternating Counters

Alternating counters deviate from the behavior of ordinary counters by the behavior of the transition that occurs when the carry signal c hits the blank word Z. In alternating counters, the size of the Z word is smaller than the size of the words for the representative one, two, and transitory words. Therefore, when the transformation occurs, the resulting tape configuration is in an inconsistent state. Therefore, each time this transition occurs, the representative structure of the tape is modified in a similar fashion as the last step that occurs in the Christmas tree behavior. Let us look at the specification of this behavior more closely to clarify:

1. For machine M or its mirror M^c, there are nonempty words E, E', A, B, B', T, and Z. The A, B, and T words again respectively represent the values of 0, 1, and a transitory word. The additional words E', and B' are included to account for the abnormally sized blank Z.
2. At some point during execution, the machine reaches the following configuration where c is some state: $0^*[E][T][_cZ][Z]^*$.
3. The following transitions hold for some states c, r, and r':
   - $[_cA] \Rightarrow_r [B]$
   - $[_cB] \Rightarrow [T]_c$
   - $[_cZ] \Rightarrow [B'_{r'}]$
   - $[B_{r'}] \Rightarrow_r [B]$
   - $[T_r] \Rightarrow_r [A]$
   - $[E_r] \Rightarrow [E]_c$
   - $[E'_r] \Rightarrow [E']_c$
4. $[E][T]^i[B'] = [E'][T]^{i+1}[B]$ for all $i \geq 1$.

The last requirement allows one to redefine the makeup of the tape in order for the correct transitions to be applied. After the carry signal c reaches a blank Z, the tape will be redefined from having an E component to having an E'. Then the next time this occurs, it will be redefined again to an E. It will "alternate" like this forever.

2.2.4 Resetting Counters

Intuitively, resetting counters periodically "reset" themselves back to zero and then start counting up again. In fact, resetting counters differ from ordinary counters in only one minor modification to one transition. In plain counters, when the carry signal c reaches the blank word Z, the Z is transformed into a B word which is representative of the number 1. When resetting counters encounter this scenario, the Z is instead transformed into an A which is the 0 representative. Thus a resetting counter will count up to 2^0, reset, 2^1, reset, 2^2, reset, and so on. The formal specification for a resetting counter is trivially derived from the original counter definition in sect. 2.2.1. We therefore do not include it here.

2.2.5 Complex Counters

Complex counters follow a modified specification to that outlined in sect. 2.2.1 that increases the scope of the behavior while still maintaining the guaranteed non-haltingness of the machines that follow it. First of all, the original description specifies two single states c and r that must remain consistent throughout the transisitions in order for the proof to hold. A simple extension can redefine c and r as sets of states instead of one single state. If this is done, than all transitions which specify c as a state must be split into n different transitions where n is the size of the set c and each transition uses a different element of the set c. The same would obviously apply to set r. Similarly, an extension can be made for each of the words A, B, T, etc. included in the specification.

Clearly, automated detection of such an extension can become a significant burden. Even relatively small sets used in the specification would give rise to a significantly greater number of defined transitions. Additionally, larger sets of words would give rise to an even greater overall set of words that need to be identified, which is one of the main obstacles of the automated routines in the first place. Regardless, with a relatively small number of holdouts to examine, machines of this nature can be identified by manual analysis.

2.2.6 Combination Counters

All of the above counter behaviors have a very clear, provably infinite specification which a machine must follow in order to be considered in that particular class of non-halt behavior. However, many of the 98 holdout machines do not fall into one explicit category of counters. Instead, some of them exhibit characteristics from some, or even all of the behaviors shown above. The specifications for any of these combination classes of machines can be derived by meshing the necessary specifications together. Going through each of the possible combinations here would be largely redundant, so we do not include them for brevity.

After manually examining the holdouts, 30 of them have been identified as some variation of the counter behaviors described above. While some of them have been explicitly proven as their identified behaviors by manually identifying each of the components that make up the specification, others remain as very strong inferences based on a close examination of their visual behavior and recognition of features common to each particular category.

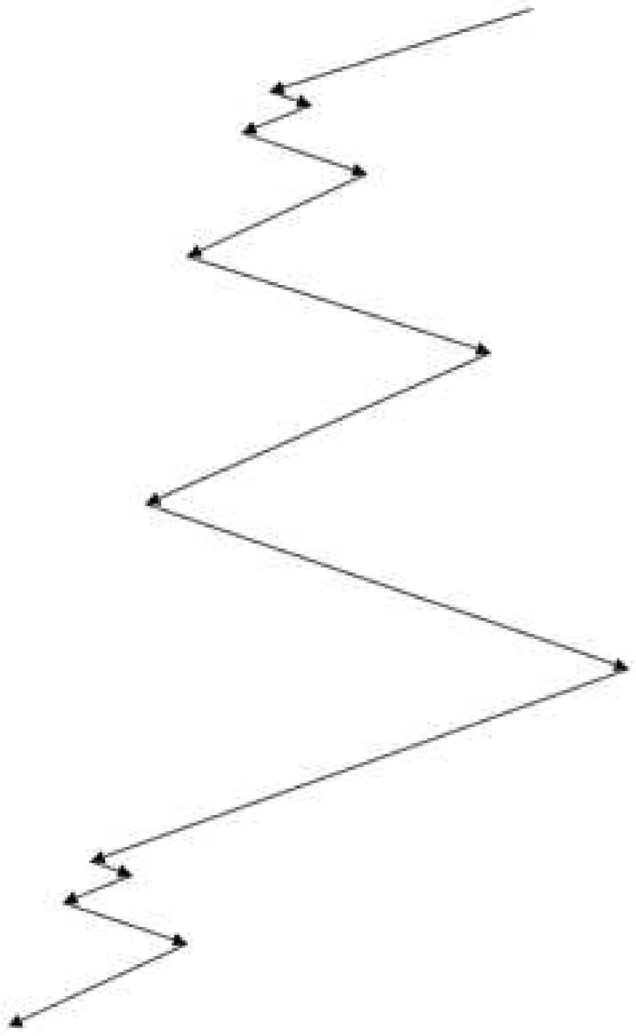

Figure 1: Nested Christmas tree conceptual behavior

2.3 Nested Christmas Trees

Once again we invite the reader to refer to the aforementioned work by Kellett and Ross in order to become more familiar with the original Christmas tree specification outlined in the work. As is described there, Christmas tree non-halters are chracterized by an infinite, repeatable, sweeping motion, where the extreme right and left boundaries of the read head are pushed farther out on each sweep. We have already seen several variations of these Christmas trees, including the already analyzed leaning Christmas trees, as well as multi-sweep Christmas trees, which require an arbitrarily large number of sweeps across the tape to occur before the identifiable cyclic pattern repeats. Let us now examine an even more bizarre variation dubbed nested Christmas trees.

Nested Christmas trees exhibit the same repeatable sweeping motion across the tape as do regular Christmas trees. However, the one very distinct difference, is that during each return sweep of the tree, the machine undergoes a miniature, nested set of sweeps resembling a nested Christmas tree until it reaches the previous extremum point of the read head. Consider fig. 2 for a more intuitive understanding of how nested Christmas trees work. Imagine that the top of the figure represents some particular point of execution of the machine. Specifically, the read head is located at the right most extremum point of the end of some sweep. The end of the arrow thus represents the location of the read head on the tape (which is not shown for simplicity). Now as one progresses down the figure, imagine that the tape at successively later points in time are shown, and that the read head is located at the intersection of the arrows and the invisible tape.

Thus we have a conceptual visual representation of how the machine behaves over time. As we can see by the figure, when the tape reaches the end of the sweep moving to the left, it undergoes a nested set of sweeps until it reaches the previous right extremum point of the read head. It then sweeps back to the left again, and starts up another nested tree. The pattern continues forever and thus the machine is a non-halter.

With a generalized understanding of how nested Christmas trees work, let us formalize the behavior:

1. For machine M or its mirror M^c, there are nonempty words U, V, X, Y, Z, U^n, V', and V''. U and V are end components that cap the left and the right sides of the overall tree. X, Y, and Z represent the repeating interior components during different points of the execution of the machine. V' and V'' are intermediate components that the right end component is in while the machine is in between sweeps. The U^n component is used as the right end component of the nested sweeps.

2. At some point during execution, the machine reaches the following configuration where s is some state: $0^*[U][X][V_s]0^*$.

3. The following transitions hold for some states q, q^n, and r^n:
   - $[V_s] \Rightarrow_q [V']$
   - $[X_q] \Rightarrow_q [Y]$
   - $0^*[U_q] \Rightarrow 0^*[U^n]_{r^n}$
   - $[_{r^n}Y] \Rightarrow_{q^n} [Z]$
   - One of the following two holds:
     - $0^*[U^n_{q^n}] \Rightarrow 0^*[U^n]_{r_n}$
     - $0^*[U^n_{q^n}] \Rightarrow 0^*[U^n][Z]_{r_n}$
   - $[_{r^n}Z] \Rightarrow [Z]_{r^n}$
   - $[_{r^n}V'] \Rightarrow [V''_s]$ if $V' \neq Y$
   - $[_{r^n}0^*] \Rightarrow [V''_s]0^*$ if $V' = Y$
   - $[Z_{q^n}] \Rightarrow_{q^n} [Z]$

4. $\forall i(0^*[U^n][Z]^i[V'']0^* \Rightarrow \exists j(0^*[U][X]^j[V]0^*))$

This specification is slightly complicated to verbally explain, but it is essentially adapted from a simplified version of the original Christmas tree specification with the inclusion of additional transitions to encapsulate the nested Christmas tree behavior of the return sweep. Again, after a manual analysis of the holdout machines, it was inferred that 27 of the remaining machines can be classified as nested Christmas trees. Similar to the case with the counters, not all of these machines have been explicitly confirmed as being nested Christmas trees. In fact, it is likely that some of the 27 machines do not actually follow the above specification but require some modification of it in order to be proven a non-halter. The extremely time consuming nature of the process of manually identifying the necessary components, confirming the relevant transitions, and documenting this work for every machine is the obstacle that has prevented this work from being already completed.

Nevertheless, a visual inspection of the machines mentioned leaves little doubt that they are at least some variation of a nested Christmas tree. Explicit proofs regarding the non-haltingness of each of these machines will come at a later date.

2.4 Uneven multi-sweep Christmas trees

Another 18 of the holdouts in question exhibit a behavior that is extremely similar to that of multi-sweep Christmas trees. In fact, for each of these machines in question, the specification that can prove its non-haltingness is identical to the particular multi-sweep class that it is a part of (2 sweeps, 3 sweeps, etc.). Now of course the obvious question becomes why are they not flagged by the automated detection routine as such.

The details concerning how these machines escape the automated routine are very specific to the implementation and therefore need not be fully discussed here. However, refer to fig. 2 for a general idea as to why these machines are not automatically detected. As one can see, on certain sweeps, the read head does not reach the extremum of the previous sweep. Therefore, complications arise when attempting to automatically extract the necessary components outlined in the specification.

Attempts to explicitly prove the non-haltingness of these machines but manually identifying the components and verifying the transitions of the specifications have not been made. Not only is this process incredibly time consuming like those for the nested Christmas trees and the counter variations, but the hope is that the necessary changes can be incorporated into the automated detection routine so that the proofs can be done automatically.

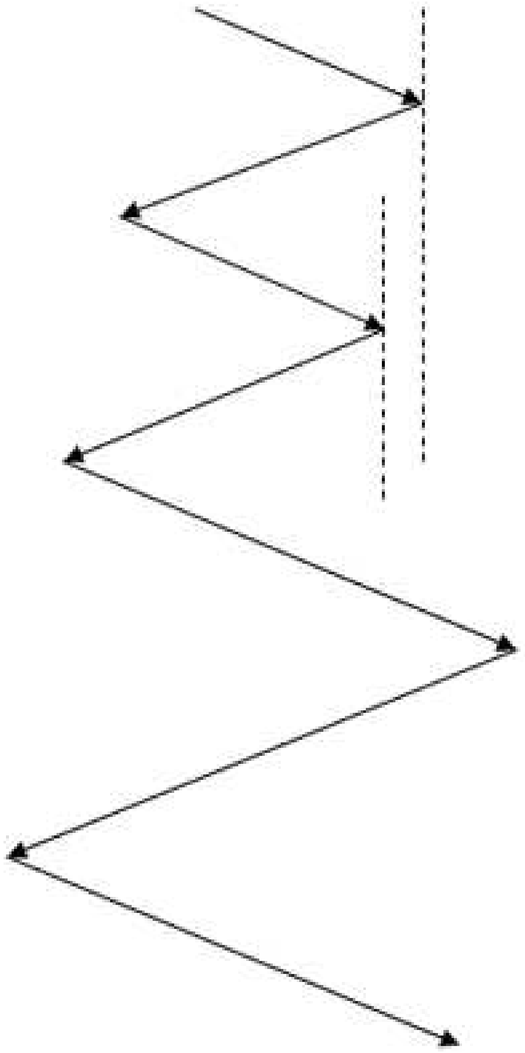
Figure 2: Uneven multi-sweep conceptual behavior

2.5 Other behaviors

After classifying the holdouts into the already mentioned categories, 13 machines remain that do not follow any of the aforementioned behaviors. Formal behaviors for these machines have not yet been defined; however, careful visual analysis of their behaviors have led to the following very strong inferences:

- 1 of these machines is most definitely what we call a "startup effects Christmas tree." The current Christmas tree detection routine runs the machine for a specified number of steps before beginning to attempt to identify components and transitions of the Christmas tree behavior. This accounts for any "startup effects" that the machine may undergo before exhibiting the infinite repeatable behavior. This 1 machine in question has an unusually long period where it exhibits startup behavior and therefore overruns the arbitrarily chosen "startup effects" threshold. A simple increase of this threshold should pickup this machine.

- 5 machines are somewhat similar to nested Christmas trees. However, in these cases, instead of performing a miniature nested Christmas tree behavior on its return sweep, the machines appear to incorporate a nested *counter* into their return sweeps. We call these machines nested Counter Christmas trees.

- 1 machine appears to be the corresponding equivalent of multi-sweep Christmas trees for leaning Christmas trees. The behavior seems to be a double sweep leaning Christmas tree which would be an intuitive extension of leaning Christmas trees.

- 3 machines are very bizarre and can be best described as 1.5 sweep Christmas trees. On every other sweep, the read head reaches only halfway to the previous extremum point before returning.

- The final 3 machines are most easily named asymmetric Christmas trees. They exhibit a similar back and forth sweeping motion of regular Christmas trees except their interior components are not identical all the way from end to end. Instead, one component repeats until halfway to the other extremum, and then another, different component repeats until the end. Thus it is asymmetric.

2.6 Conclusion

Again, while many of the above classified machines have not yet been explicitly proven, after careful visual analysis of all of them, it is extremely difficult to conceive of the possibility that any of them halt at some point. As a result, the 98 holdouts of the BB(5) problem left by the work of Kellett and Ross almost certainly exhibit one of the non-halting behaviors described. Therefore, with the work outlined in this paper in conjunction with the work already established by Kellett and Ross, it can be said with almost 100% certainty that the established records for the BB(5) problem are in fact the true champions, and that the result of the BB(5) is now explicitly confirmed.