

UNDER WORK IN PROGRESS!! paper is not finished, check back never!!
--

Complete Decision of Small Turing Machine Classes by Tree-Normal-Form Enumeration with Translated-Cycle and Backward Deciders

Cooper E.

Independent researcher · jcooperkai@gmail.com

ABSTRACT

The Busy Beaver function is uncomputable, so each value must be established by deciding every machine in its class individually. We present an independently written enumeration and decision pipeline and use it to completely decide four classical classes: every machine in $BB(2,2)$, $BB(3,2)$, $BB(2,3)$ and $BB(4,2)$ is proven either to halt or never to halt, with no residue. The maxima recovered are 6, 21, 38 and 107 steps, the last with 13 marks, matching the published values; no parameter was tuned against those answers. The pipeline combines Tree-Normal-Form enumeration with three sound deciders applied in sequence. We report the number of machines each decider removes, which is the quantity that actually determines whether a class can be closed: cyclers and translated cycles leave between 2.8 and 589 thousand undecided machines depending on the class, and backward reasoning removes all of them. We also report a soundness defect found in our own backward decider, in which exhaustion of the search window was treated as a proof of impossibility. The defect produced a confident false result -- it certified the $BB(2,4)$ champion, a machine that halts after 3,932,964 steps, as non-halting -- and is described in full because the failure mode is silent and would be easy to reproduce.

1. Introduction

For a Turing machine with n states and m symbols started on a blank tape, $BB(n,m)$ is the greatest number of steps taken by any machine of that description that eventually halts. Rado introduced the function in 1962 and showed that it grows faster than any computable function [1]. The consequence is uncomfortable and total: no algorithm can return $BB(n,m)$ in general, and no amount of computation substitutes for a proof. A value is established only when every machine in the class has been individually resolved.

That requirement is stricter than it first appears, and it is where informal attempts fail. Finding the machine that runs longest is comparatively easy; one simulates the class under a step limit and records the maximum. Establishing that this maximum is *the* maximum requires proving that every machine which did not halt within the limit never halts at all. A single unresolved machine leaves the value unproven, because that machine might run for a very long time and then stop.

Values are known for $BB(2,2) = 6$, $BB(3,2) = 21$, $BB(2,3) = 38$, $BB(4,2) = 107$ and $BB(2,4) = 3,932,964$; $BB(5,2) = 47,176,870$ was settled in 2024 by a collaborative effort with a machine-checked proof [2]. $BB(3,3)$, $BB(2,5)$, $BB(2,6)$, $BB(4,3)$, $BB(3,4)$ and $BB(6,2)$ remain open.

This paper describes an independent implementation, written from scratch without reference to existing decider code, and reports what it does and does not achieve. Section 2 covers enumeration, Section 3 the deciders, Section 4 the measured results, Section 5 a soundness defect we found in our own work, and Section 6 the limits of the approach.

2. Enumeration in Tree Normal Form

A naive enumeration of all transition tables for n states and m symbols considers $(2mn+1)^{mn}$ machines. Most are redundant. Many are relabellings of one another, and many contain transitions that are never reached from the blank tape and therefore cannot affect behaviour.

Tree Normal Form avoids both problems by refusing to write down a transition before the machine demands it. A machine begins with a single defined transition and is simulated. When the head reads a state-symbol pair whose transition is undefined, the search branches over every canonical way of defining it: each write symbol, each direction, and each next state up to and including one newly introduced state. The restriction to at most one new state per branch is what removes isomorphic duplicates, since it fixes the order in which states are named. A halting completion is added alongside. The first transition is fixed to write 1, move right, and enter state 1, which breaks the remaining symmetry at the root.

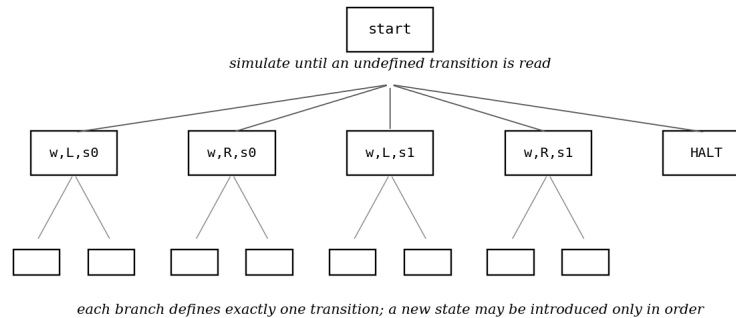


Figure 1. Tree-Normal-Form branching. A machine is simulated until it reads an undefined transition, at which point the search branches over every canonical completion of that one transition. Transitions are therefore never enumerated speculatively; they are created only at the moment the machine actually reaches them.

The saving is substantial. The full tree for $BB(4,2)$ contains 2,943,669 machines, against $17^8 = 6.98 \times 10^9$ for a naive enumeration of the same class.

3. Deciders

Simulation alone can prove that a machine halts. It can never prove that a machine does not, since exceeding a step limit is not evidence of anything. Non-halting must be established by argument, and each argument covers a different family of behaviour. We apply three in sequence, cheapest first.

3.1 Cyclers

If a machine returns to a configuration it has already occupied -- same state, same tape, same head position -- it will repeat that configuration forever. This is the simplest family and the cheapest to detect.

3.2 Translated cycles

Far more machines repeat a configuration *shifted in space* while marching along the tape. Such machines never revisit a configuration exactly, so cycler detection misses them entirely. The following criterion decides them.

Let $t_1 < t_2$ be *record steps*: steps at which the head first visits a cell it has never visited, so that all tape beyond the head is blank. Suppose the machine is in the same state at both, with head positions $p_1 < p_2$, and write $d = p_2 - p_1$. Let L be the leftmost cell touched during the interval. If

$$\text{tape}@_{t_2} [L .. p_2] = \text{tape}@_{t_1} [L-d .. p_1]$$

then the machine replays the interval forever, translated by d , and never touches anything left of L . It therefore does not halt.

Soundness follows because the compared window covers every cell the machine can reach before the next record, and everything beyond the head is blank in both configurations by the definition of a record step. The mirror criterion handles leftward motion, and $d = 0$ recovers ordinary cyclers.

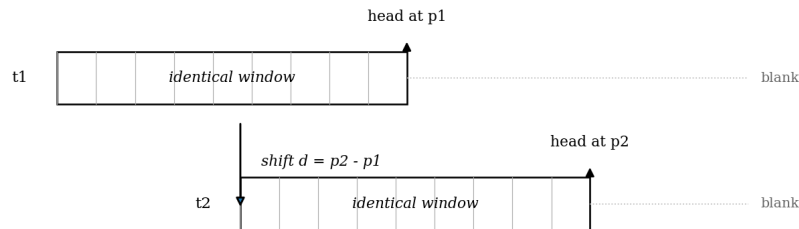


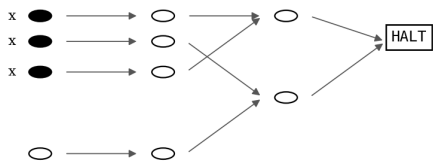
Figure 2. The translated-cycle criterion. At two record steps the machine occupies the same state with an identical tape window, offset by d . Everything beyond the head is blank in both cases, so the behaviour between the records repeats indefinitely, displaced by d each time.

3.3 Backward reasoning

The machines that survive both cycle deciders are those whose non-halting has no periodic witness. For these we reason backwards from the halt condition rather than forwards from the start.

To halt, a machine must execute a halting transition, which means reaching some state s reading some symbol r with $t[s][r] = \text{HALT}$. We take that condition as a goal and search for configurations that could immediately precede it. A predecessor of a configuration in state s with the head at p is any transition $t[s'][r'] = (w, d, s)$ whose written symbol w is consistent with what the successor configuration requires at that cell. Where it is inconsistent, no such predecessor exists and the branch dies. Tape cells are tracked as known or unknown over a bounded window.

If every branch dies within a bounded depth, no configuration reachable from any start leads to a halting transition, and the machine does not halt.



predecessors searched backwards; a contradiction kills a branch (x). All branches dead within depth $D \Rightarrow$ the halt is unreachable.

Figure 3. Backward reasoning. Predecessors of the halt condition are expanded in reverse. A branch requiring a cell to hold two different symbols is contradictory and dies. When the entire frontier dies within the depth bound, the halting transition is unreachable.

4. Results

All numbers below are produced by a single-file C++ implementation compiled with -O3, run on one core of an Apple Silicon laptop. No parameter was tuned against a known value; the classical results were compared only after each run finished.

Class	Halting	Non-halting (proved)	Undecided	Maximum	Published
BB(2,2)	15	106	0	6	6
BB(3,2)	1,379	15,170	0	21	21
BB(2,3)	866	9,527	0	38	38
BB(4,2)	183,983	2,759,686	0	107 (13 marks)	107 (13)

Table 1. Complete decisions. Every machine in each class is proved to halt or proved not to halt; the undecided column is zero throughout, which is the condition for the value to be established rather than merely observed. BB(4,2) reproduces both the step and the mark record.

The distribution of work across deciders is the more informative measurement, because it shows which argument is actually carrying the class.

Class	After cyclers	After translated cycles	After backward reasoning
BB(2,2)	92	20	0
BB(3,2)	13,187	3,065	0
BB(2,3)	8,492	2,779	0
BB(4,2)	--	589,107	0

Table 2. Undecided machines remaining after each decider. Translated-cycle detection removes roughly four fifths of what cyclers leave. Backward reasoning then removes the entire remainder in every class, including 589,107 machines in BB(4,2).

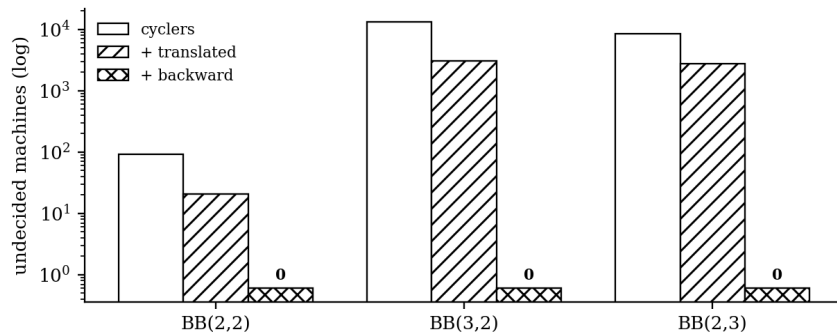


Figure 4. Undecided machines after each decider stage, logarithmic scale. The final stage reaches exactly zero in all classes, marked explicitly since zero cannot be drawn on a logarithmic axis.

The pattern is consistent and worth stating plainly: the two cycle deciders do the bulk of the volume, and backward reasoning does the part that determines whether the class closes. A pipeline with only the first two would report a correct maximum for every class in Table 1 while proving nothing, because a correct maximum and a decided class are different achievements.

5. A soundness defect, and why it is reported

An earlier version of the backward decider contained a defect that produced confident false results. It is described here because the failure mode is silent, plausible, and easy to reintroduce.

The backward search tracks tape cells over a bounded window. When expansion moved the head outside that window, the implementation discarded the branch and continued. An empty frontier was then interpreted as "every branch died", which is the condition for concluding that the halt is unreachable. Two different situations had been given one code path: a branch that died because it was *contradictory*,

and a branch that vanished because the search had *run out of room to look*. The first is a proof. The second is an absence of information.

The consequence was not a crash or a warning but a wrong certificate. Run against BB(2,4) under a step limit of 100,000, the pipeline reported zero undecided machines. The champion of that class halts after 3,932,964 steps, far beyond the limit, so it necessarily reached the backward decider -- which certified it as non-halting.

The correction is to track truncation explicitly and to conclude non-halting from an empty frontier only when nothing was discarded. After the fix, all four classes in Table 1 were re-run and produced byte-identical output, confirming that their backward proofs had never depended on truncation.

Two general points follow. First, a decider that is unsound in a rare branch will usually still produce correct maxima, because maxima come from forward simulation; only the non-halting claims are corrupted, and those are invisible without an independent check. Second, the guard that caught this was validation against a known value. Had BB(2,4) not been checked, the defect would have survived into the open classes, where nothing would have contradicted it.

6. Limitations

This pipeline decides the four classes in Table 1 and does not decide any open class. Three obstacles separate the two situations, and they are of different kinds.

Decider coverage. Three deciders suffice for the classes above. Larger classes contain machines whose non-halting has neither a periodic witness nor a bounded backward refutation; these require methods we have not implemented, including closed-tape-language arguments and halting-segment analysis.

Simulation depth. Champions in open classes run far past any limit we can apply. BB(3,3) is known to exceed 1.19×10^{17} steps. A machine of that kind cannot be resolved by forward simulation at all, so decider strength, not compute, is the binding constraint.

Machines that are provably hard. Some machines have halting behaviour equivalent to open problems in number theory, with Collatz-like iteration structure. No decider settles them without settling the underlying mathematics. Their presence in a class is a genuine barrier rather than an engineering gap, and it is why certain classes may remain open indefinitely.

We therefore claim no new Busy Beaver value. What is established is an independent reproduction, with complete decision, of four classical values, together with a measurement of how the work divides between deciders -- and a documented account of a soundness defect that a maximum-only check would not have detected.

7. Reproducibility

The implementation is a single C++ file with no dependencies.

```
clang++ -O3 -march=native -std=c++20 -o dec dec.cpp  
./dec <states> <symbols> <step-limit> <backward-depth>
```

Every figure in Tables 1 and 2 is reproducible in minutes on a laptop. Runs are deterministic; there is no randomness in enumeration or in any decider.

References

1. T. Rado. On non-computable functions. *Bell System Technical Journal*, 41(3):877-884, 1962.
2. The bbchallenge Collaboration. Determination of BB(5). Machine-checked proof, 2024.
3. S. Lin and T. Rado. Computer studies of Turing machine problems. *Journal of the ACM*, 12(2):196-212, 1965.
4. A. H. Brady. The determination of the value of Rado's noncomputable function $\Sigma(k)$ for $k = 4$. *Mathematics of Computation*, 40(162):647-665, 1983.
5. T. and S. Ligocki. Busy Beaver champions for multi-symbol machines, 2005-2008.