Determination of the Fifth Busy Beaver Value

Justin Blanchard*
USA

Daniel Briggs
USA

Konrad Deka
Jagiellonian University
Poland

Nathan Fenner
USA

Yannick Forster
INRIA Paris
France

Georgi Georgiev (Skelet)
Sofia University, Faculty of Mathematics and Informatics
Bulgaria

Matthew L. House
University of Georgia
USA

Maja Kądziołka
University of Warsaw
Poland

Pavel Kropitz
Slovakia

Shawn Ligocki
USA

mxdys
China

Mateusz Naściszewski
Poland

Tristan Stérin†
PRGM DEV
France
tristan@prgm.dev

Chris Xu
UC San Diego
USA

Jason Yuen
University of Waterloo
Canada

Théo Zimmermann
LTCI, Télécom Paris, Institut Polytechnique de Paris
France

Abstract

The Busy Beaver value $S(n)$ is the maximum number of steps that an n-state 2-symbol Turing machine can perform from the all-zero tape before halting. S was historically introduced by Tibor Radó in 1962 as one of the simplest examples of an uncomputable function.

We prove that $S(5) = 47,176,870$ using the Coq proof assistant. The proof enumerates 181,385,789 Turing machines with 5 states and, for each machine, decides whether it halts or not. Our result marks the first determination of a new Busy Beaver value in over 40 years and the first Busy Beaver value ever to be formally verified, attesting to the effectiveness of massively collaborative online research (bbchallenge.org).

CCS Concepts

• **Theory of computation** → **Computability**; **Turing machines**; *Regular languages*; • **Software and its engineering** → **Software verification**.

*Alphabetical ordering.
†Corresponding author.

STOC '26, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-2536-4/2026/06
https://doi.org/10.1145/3798129.3806376

Keywords

Busy Beaver, Turing machines, computability, formal verification, Coq, Rocq

ACM Reference Format:
Justin Blanchard, Daniel Briggs, Konrad Deka, Nathan Fenner, Yannick Forster, Georgi Georgiev (Skelet), Matthew L. House, Maja Kądziołka, Pavel Kropitz, Shawn Ligocki, mxdys, Mateusz Naściszewski, Tristan Stérin, Chris Xu, Jason Yuen, and Théo Zimmermann. 2026. Determination of the Fifth Busy Beaver Value. In *Proceedings of the 58th Annual ACM Symposium on Theory of Computing (STOC '26), June 22–26, 2026, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3798129.3806376

1 Introduction

This paper is a condensed version of the arXiv paper [2], which contains additional results, the complete proofs, figures, appendices, and full author list; we invite the reader to consult it.

The bbchallenge Collaboration: $S(5)$ credits. The following contributions resulted in the determination of the fifth Busy Beaver value and in the better understanding of the landscape of small Busy Beaver values (Table 1): mxdys (Coq-BB5, Loops, RepWL); Nathan Fenner, Georgi Georgiev a.k.a Skelet, savask, mxdys (NGramCPS); Justin Blanchard, Mateusz Naściszewski, Konrad Deka (FAR); Iijil (WFAR); Maja Kądziołka (busycoq); Shawn Ligocki, Jason Yuen, Maja Kądziołka (Sporadic Machines "Shift Overflow Counters"); Shawn Ligocki, Pavel Kropitz, Maja Kądziołka (Sporadic Machine "Skelet #1"); savask, Chris Xu, mxdys (Sporadic Machine "Skelet

#17"); Shawn Ligocki, Dan Briggs, Maja Kądziołka (Sporadic Machine "Skelet #10"); Justin Blanchard, Maja Kądziołka (Sporadic Machines "Finned Machines"); Shawn Ligocki, Daniel Yuan, mxdys, Matthew L. House, Rachel Hunter, Jason Yuen ("Cryptids"); Yannick Forster, Théo Zimmermann (Coq review); Yannick Forster (Coq optimisation); Tristan Stérin (bbchallenge.org); Tristan Stérin, Justin Blanchard (FAR writeup); Tristan Stérin (paper writing).

Acknowledgement. We gratefully acknowledge Rachel Hunter, Iijil, and savask, anonymous contributors who contributed to this work as much as any co-author of this paper but could not be listed as such in this version under the ACM Authorship Policy. The main source of funding for this work was the company `prgm.dev` through the French state (*Crédit Impôt Recherche* and *Jeune Entreprise Innovante* programs). The bbchallenge Collaboration is not limited to the above contributors but regroups all those who participated in the bbchallenge's discussions through all our channels (Discord chat, forum, wiki, GitHub, emails) and in particular we thank: Nick Drozd, Andrew Ducharme, Frans Faase, Tony Guilfoyle, Johannes Hostert, Nick Howell, Jeffrey Huang, Alexandre Jouandin, Carl Kadie, Frank S. Lin, Dawid Loranc, Terry J. Ligocki, Heiner Marxen, Pascal Michel, Milo Mighdoll (milomg), Seraphina Nix, Sébastien Ohleyer, Peacemaker II, Andrés Sancho, tomtom2357, Valentin, Daniel Yuan, Polygon. We'd like to thank `bbchallenge.org` sponsor `prgm.dev`. We'd also like to thank those who helped disseminate the bbchallenge project, including Scott Aaronson, Timothy Gowers, Ben Brubaker, Damien Woods, Dave Doty, and Léo Ramaën. We thank the reviewers and the Automated Pre-Submission Feedback for helpful comments.

1.1 Main Result

Introduced by Tibor Radó in 1962 [71, 72], the Busy Beaver value $S(n)$ is the maximum number of steps that an n-state 2-symbol Turing machine can perform from the all-zero tape before halting. S is uncomputable: if an n-state machine runs for more than $S(n)+1$ steps, we know it will never halt, giving an algorithm to decide Turing's halting problem.[1]

While there is no algorithm to compute S for *all* n, we can certainly try to compute S for *some* n. Prior to this work, only the first four values of S had been proved: $S(1) = 1$, $S(2) = 6$ [71], $S(3) = 21$ [65], and $S(4) = 107$ [12]. In 1989, Marxen and Buntrock found a new champion[2] achieving 47,176,870 steps [66], showing $S(5) \geq 47{,}176{,}870$ [67, 69]. However, it remained unknown if no other machine could beat it. In 2020, based on the lack of a new 5-state champion in 30 years, Aaronson conjectured that $S(5) = 47{,}176{,}870$ [1].

Our main result is to prove this conjecture, using the Coq proof assistant [86]. The Coq proof is called Coq-BB5 and is available at `github.com/ccz181078/Coq-BB5` [74]. We also prove $\Sigma(5) = 4{,}098$ and $\text{space}(5) = 12{,}289$ [4], which are closely related uncomputable functions, see [2].

Theorem 1.1 (Coq-BB5: `Lemma BB5_value`). $S(5) = 47{,}176{,}870$.

The function S can naturally be extended to Turing machines using more than two alphabet symbols [13]; for instance, $S(2,3) = 38$ is the value of S for 2-state, 3-symbol machines [13, 50, 68]. We prove, using Coq, that $S(2,4) = 3{,}932{,}964$:

Theorem 1.2 (Coq-BB5: `Lemma BB2x4_value`). $S(2,4) = 3{,}932{,}964$.

Coq-BB5 provides formal proofs for $S(5)$ and $S(2,4)$ — as well as for previously known $S(2)$, $S(3)$, $S(4)$ and $S(2,3)$. The lists of all the Turing machines enumerated by these proofs, together with their Coq-verified halting status, are available at https://docs.bbchallenge.org/CoqBB5_release_v1.0.0/.

As a result of our work, we now have a clearer view of the landscape of small Busy Beaver values; see Table 1.

Table 1: Landscape of small Busy Beaver values. Values in bold are original to this work. Our Coq proofs cover all values listed without a > sign. Lower bounds come from exhibiting machines achieving at least the given step-counts. With 6 states, we have $S(6) > 2 \uparrow\uparrow\uparrow 5$ (Knuth's up-arrow notation is defined in [47]). See [2] for the full landscape.

Sym.	2-State	3-State	4-State	5-State
2	$S(2) = 6$	$S(3) = 21$	$S(4) = 107$	$\mathbf{S(5) = 47{,}176{,}870}$
3	$S(2,3) = 38$	$S(3,3) > 10^{17}$		
4	$\mathbf{S(2,4) = 3{,}932{,}964}$			

1.2 Challenges

Proving $S(5) = 47{,}176{,}870$ required analysing the behaviour of 181,385,789 Turing machines – evidently requiring computer assistance. The challenge for analysing a halting machine occurs when it halts after a number of steps that is too enormous to be simulated step-by-step; this challenge was not encountered for 5-state machines since they halt in at most 47,176,870 steps – this could not have been known for certain in advance but was believed – which is easy to simulate on modern computers. The challenge for analysing a nonhalting machine is that proving that it does not halt can be hard. How hard?

Dauntingly, any Π_1^0 mathematical statement[3] can be encoded as the halting problem of a Turing machine (from all-zero tape). Such statements are common in mathematics and include famous open problems such as Goldbach's conjecture and the Riemann hypothesis [21]. Goldbach's conjecture, formulated in 1742, is one of the oldest open problems in mathematics and states that "every even positive integer greater than 2 is the sum of two prime numbers". We can build a Turing machine that halts iff the conjecture is false: by enumerating all even positive integers, and for each, testing all pairwise sums of smaller primes and halting iff we cannot express it as such a sum. A machine performing this procedure has been built using only 25 states, and the construction was formally verified using the Lean theorem prover [16, 22, 54].

This means that proving the value of $S(25)$ is at least as hard as solving Goldbach's conjecture for two reasons: (i) assuming $S(25)$ is known, we could, impractically, simulate the Goldbach machine for $S(25)$ steps to see if it has halted to settle the conjecture – this

[1] In the variant where machines are given no input and start from the all-zero tape.
[2] Turing machine 1RB1LC_1RC1RB_1RD0LE_1LA1LD_---0LA.
[3] A Π_1^0 statement is a statement of first-order logic of the form "$\forall x, \phi(x)$" where ϕ is a sentence using only bounded quantifiers, implying that for a given x, $\phi(x)$ can always be verified by a computer in finite time.

is unrealistic because $S(25) > f^2_{\omega^2}(4 \uparrow\uparrow 341)$, where f refers to the *Fast Growing Hierarchy* [3, 90] and (ii) intuitively, determining $S(25)$ requires arguments justifying the halting status of each 25-state machine, including this particular one. Similarly, the Riemann hypothesis has been encoded in a 744-state machine [95, 97]. As few as 15 states are enough to encode a hard conjecture in number theory by Erdős [80]. Worse, the consistency of common axiomatic systems such as Peano Arithmetic (PA) or Zermelo–Fraenkel set theory (ZF) is also Π^0_1 since one can enumerate proofs in these systems until the proof of a contradiction is found. This has been done in practice for ZF, using 748 states [1, 76, 95]. This result has known further improvements to 636 states [75], and even 432 states [89], pending verification. By Gödel's second incompleteness theorem, this means that proving the value of $S(748)$ cannot be done using ZF. Aaronson conjectures that as low as $S(20)$ cannot be proved in ZF and $S(10)$ cannot be proved in PA [1].

Hence, while 5-state halting machines were not feared, it remained unknown how hard proving 5-state machines nonhalting could get. This article settles the question: the smallest open problem in mathematics (on the Busy Beaver scale) does not arise from 5-state machines – but we have good contenders among 6-state machines; see §1.5.

1.3 Related Work

In 1983, Brady published the proof of $S(4) = 107$ [12]. A caveat of the proof resides in the following quote from the paper: "All of the remaining holdouts were examined by means of voluminous printouts of their histories along with some program extracted features. It was determined to the author's satisfaction that none of these machines will ever stop." A *holdout* is a machine still needing a proof of halting/nonhalting. Using Coq, we bring additional confirmation that $S(4) = 107$; see [2].

We know of two attempts at solving $S(5)$: in 2003, Georgiev (Skelet) published the program `bbfind` [30] which enumerates and decides the halting behaviour of 5-state Turing machines, claiming to leave unsolved 43 holdouts. However, attesting to the validity of these results is difficult as Skelet's program consisted of about 6,000 lines of undocumented Pascal code. That said, it turned out to be instrumental to solving $S(5)$, as `bbfind`'s "Closed Position Set" technique (see §3) was used, simplified, and improved in order to decide slightly more than 99.87% of the 5-state Turing machines excluding loops. Also, all of our 13 *Sporadic Machines*, i.e. machines for which we needed individual proofs of nonhalting, were either among Skelet's 43 holdouts or claimed to have been manually solved by him – §4 is dedicated to these longstanding holdouts. Some of Skelet's 43 holdouts were analysed by hand by Briggs starting in 2010 [14]. The second known attempt at solving $S(5)$ was in 2008 with Hertel's "Symbolic induction prover", which claimed to leave only 1,000 holdouts, 900 of which were manually proved not to halt, allegedly leaving only 100 proper holdouts [40]. In contrast to Skelet's work, the method is documented but, to the best of our knowledge, neither the code nor the 900 claimed manual proofs are made available, making verification of the result difficult apart from attempting to reproduce it from scratch.

The Busy Beaver problem was also studied in models of computation other than Radó's, including (i) the *quadruple* variation of Turing machines where each transition may move or write a new symbol, but not both [45, 77]; (ii) *turmites*, which are Turing machines that operate in 2D [13]; and (iii) lambda calculus [88]. Some authors use the notation BB(n) for either Σ [39, 71] or S [1, 73]. We avoid this naming due to this ambiguity. For additional historical perspective on the Busy Beaver problem, we refer the reader to Pascal Michel's survey and website [67, 69].

1.4 Structure of the Proof

The structure of the proof of our main result, Theorem 1.1, is as follows: machines are enumerated arborescently in *Tree Normal Form* (TNF) [11] – which drastically reduces the search space's size: from 16,679,880,978,201 5-state machines to "only" 181,385,789; see §2. Each enumerated machine is fed through a *pipeline* of proof techniques, mostly consisting of *deciders*, which are algorithms trying to decide whether the machine halts or not. Because of the uncomputability of the halting problem, there is no *universal* decider and all the craft resides in creating deciders able to decide large families of machines in reasonable time. Almost all of our deciders are instances of an abstract interpretation framework that we call *Closed Tape Language* (CTL), which consists in approximating the set of configurations visited by a Turing machine with a more convenient superset, one that contains no halting configurations and is closed under Turing machine transitions (see §3). The $S(5)$ pipeline is given in Table 3. All the deciders in this work were crafted by The bbchallenge Collaboration; see §3.

In the case of 5-state machines, 13 *Sporadic Machines* were not solved by deciders and required individual proofs of nonhalting, see §4. These machines include surprising behaviours, such as eventually reaching an infinite loop after more than 5.41×10^{51} steps of chaos (machine "Skelet #1"), base-Fibonacci double counter (machine "Skelet #10"), or obfuscated Gray code (machine "Skelet #17" [94]), and they are beautiful examples of *algorithms in the wild*: non-human-engineered algorithms that, like deep sea life, were only found by means of exploration. Algorithms, correctness proofs, and detailed results for all deciders and sporadic machines are given in [2].

Collaboratively Solving the Problem: bbchallenge.org. In 2022, Stérin created "The Busy Beaver Challenge", `bbchallenge.org`, an online platform dedicated to collaboratively solving "$S(5) = 47{,}176{,}870$" [73]. Collaboration was motivated by the great amount of Turing machines to study in order to solve the problem. With the surprise release of Coq-BB5 in the spring of 2024, almost all the collaborative work performed on `bbchallenge.org` during these 2 years was embedded in the Coq proof – Coq-BB5 also contains many original innovations. The bbchallenge Collaboration roughly comprises all who participated in the discussions across all our channels (Discord chat, forum, wiki, GitHub, emails), who total hundreds of people. Some of them are anonymous. Most contributors have no academic affiliation. Our research happened in public with no withholding of information, enabling full reproducibility of the results. Formal verification efforts had started as early as 2022 when Fenner verified some deciders using Dafny [26, 53]. We invite the reader to consult [2] for a detailed account of the collaboration [6, 15, 46, 52, 70].

Proof Assistants and Coq-BB5. Coq-BB5 is written in Coq[4] [86], which is a proof assistant and programming language based on the Calculus of Inductive Constructions [19] whose development started in 1989. With hundreds of millions of Turing machines to study, the use of a proof assistant immensely facilitates scientific consensus on the correctness of the proof: for instance, we are assured that no Turing machine was forgotten in the study and that our proofs of nonhalting are correct. Also, the 13 individual proofs for Sporadic Machines contain technical and error-prone arguments that would be harder to verify and trust if not formalised – e.g. a standalone article was dedicated to machine "Skelet #17" [94] and its Coq proof is almost 7,000 lines long.

Coq-BB5 was released in the spring of 2024 by contributor mxdys who embedded two years of work done by the bbchallenge collaboration as well as improving and introducing new deciders to finish the proof: Coq-BB5 is not an *a posteriori* formalisation of existing work and contains many original innovations without which even a nonformalised $S(5)$ proof would have been a lot more complex. Coq-BB5 totals 27,274 lines of Coq and 638 lemmas; plus an additional 10,553 lines of Coq and 319 lemmas including imported `busycoq` proofs [49]. Coq-BB5 is one of the most compute-intensive formal proofs to date as it implements both the TNF enumeration of the 181,385,789 Turing machines and the deciders directly in Coq. Proofs such as Coq-BB5, which rely on computing algorithmic outputs [7, 36], are known as "proofs by reflection" [10], and the Coq proof of the four-color theorem also fits in that category [31, 32, 34]. Milestones in the use of proof assistants also include the Feit-Thompson theorem [33], the Kepler Conjecture [38], and the liquid tensor experiment [17, 18]. Coq-BB5 initially compiled in about 13 hours on a standard laptop, but the use of Coq's faster computing engine `native_compute` [7] and parallelisation of the proof (see §2) brought compile time down to about 45 minutes on 13 cores. The proof only relies on Coq's standard library axiom `functional_extensionality_dep`, which claims that two functions are equal if they are equal at all points. This axiom is widely accepted, consistent in Coq, and true in common set-theoretic foundations of mathematics.

1.5 Discussion

Cryptids. The Busy Beaver game is a concrete attempt at identifying the frontier between the knowable and the unknowable: what is the highest n for which we can prove the value of $S(n)$? Knowing that for $n > 5$, the proof may involve solving arbitrarily difficult problems or worse, simply be outside of PA or ZF. The Busy Beaver game is great at generating open problems in mathematics!

For instance, in all the unsolved Turing machine classes, we have found what we call "Cryptids", which are loosely defined as Turing machines whose halting problem from the all-zero tape is believed to be mathematically hard. With 6 states, Antihydra (1RB1RA_0LC1LE_1LD1LC_1LA0LB_1LF1RE_---0RA) is a machine that does not halt from the all-zero tape if and only if the following Collatz-reminiscent conjecture holds:

Conjecture 1.1 (Antihydra does not halt)**.** Consider the Collatz-like map $H : \mathbb{N} \to \mathbb{N}$ defined by $H(x) = 3\frac{x}{2}$ if x is even and $H(x) = \frac{3x-1}{2}$ if x is odd. Iterating H from $x = 8$, there are never (strictly) more than twice as many odd numbers as even numbers.

Using Conway's terminology, the conjecture *probviously* (portmanteau of the words probabilistic and obvious) holds because a probabilistic analysis of the Turing machine suggests that its probability of ever halting is minuscule, namely, $\left(\frac{\sqrt{5}-1}{2}\right)^{1073720885} \approx 4.84 \times 10^{-224394395}$. However, properly proving that Antihydra does not halt is believed mathematically hard, because of resemblance to the notoriously hard Collatz conjecture [51]: Antihydra is a Cryptid. Also, the map H was studied before Antihydra was discovered and is known to not generate Sturmian words [24]. Antihydra is the *smallest* open problem in mathematics, on the Busy Beaver scale. Or, to be exact, one of the smallest, since a dozen other 6-state Cryptids have been identified to date, such as the following, jokingly called, Beaver Math Olympiad (BMO) problem:

Problem 1.1 (BMO Problem 1)**.** Let $(a_n)_{n\geq 1}$ and $(b_n)_{n\geq 1}$ be two sequences such that $(a_1, b_1) = (1, 2)$ and

$$(a_{n+1}, b_{n+1}) = \begin{cases} (a_n - b_n, 4b_n + 2) & \text{if } a_n \geq b_n \\ (2a_n + 1, b_n - a_n) & \text{if } a_n < b_n \end{cases}$$

for all positive integers n. Does there exist a positive integer i such that $a_i = b_i$?

This problem is a reformulation of "does 1RB1RE_1LC0RA_0RD1LB_---1RC_1LF1RE_0LB0LE halt?" In hindsight, it is surprising (and lucky!) that there are no 5-state Cryptids. Cryptids illustrate the ease with which the Busy Beaver game generates small, non-trivial, open mathematical problems, challenging our intelligence and the limits of mathematical knowledge. We invite the reader to consult [2] for a detailed account of Cryptids.

Trends in Collaborative Research. Massively collaborative online research in mathematics and theoretical computer science is a relatively new phenomenon, pioneered by the Polymath Project, which collaboratively solved several problems in mathematics [35]. In many ways, `bbchallenge`'s effort is closer to experimental open-source software project development than to traditional research in mathematics or theoretical computer science: we mainly develop algorithms (deciders), and as a consequence, Coq-BB5 itself is essentially a collection of algorithms, proved correct.

The current increasing popularity of proof assistants such as Coq or Lean within the mathematical and computer science communities naturally accelerates this convergence between collaborative research and open source software development: researchers can now leverage the same tools and processes as developers (version control, *pull requests*, *issues*, etc.) to massively collaborate on proofs in a scalable way – i.e. not requiring humans to check or trust proofs. As a concrete example, shortly after we announced our $S(5)$ result, Terence Tao launched a collaborative pilot project in universal algebra, called the "Equational Theories Project" (ETP), leveraging GitHub and Lean as their means of collaboration [8, 9, 83]. The project was extremely successful, attracted more than 50 contributors, and was completed in only a few months.

Future Work. Solving $S(5)$ does not solve all questions about 5-state Turing machines, for instance we are interested in the following problems: (i) characterising all 5-state counters (see [2]); (ii)

[4] Now renamed the Rocq Prover: https://rocq-prover.org/about#Name.

finding the biggest loop among 5-state machines with no halting transitions – we already know of some that are way bigger than Sporadic Machine "Skelet #1" [56]; (iii) is there a universal Turing machine with 5 states?

Progress is ongoing in all unsolved Turing machine classes (Table 1): new deciders are being developed to tackle $S(3,3)$ [57], $S(3,4)$ [63], $S(2,5)$ [62] and $S(6)$ [64], including a generalisation of all the regular CTL deciders presented in this paper. Most of these new deciders have been formalised using Coq. There currently remain only 4 holdouts for $S(3,3)$, including a probviously halting suspected new champion, and only 60 holdouts for $S(2,5)$.

Concerning $S(6)$, the TNF enumeration of 6-state machines, which contains about 33 billion machines, has also been partially implemented in Coq, and, together with the deciders and about 2,000 individual proofs of nonhalting, "only" about 1,214 holdouts remain to date. Importantly, among these holdouts we have:

- Antihydra and other Cryptids, which are, most likely, extremely hard problems to solve.
- The possible existence of a halting machine that exceeds the current $2 \uparrow\uparrow\uparrow 5$ champion, which could require significant analysis or accelerated simulator improvements to be detected.

The open problems generated by our work have been included in a dataset of Lean-formalised conjectures, designed to serve as future challenges for AI reasoning tools [23]. More generally, given the sheer amount of problems, with broad variety of difficulty (from very easy to near-impossible), that the Busy Beaver game can generate, the ability to prove known Busy Beaver values or to make progress on unknown ones would be an ambitious benchmark for machine intelligence. Coq-BB5 has already started being used with that goal in mind: AI systems have been tested on proving its first 100 lemmas of the $S(4)$ proof, with 58% success so far [84]; see also [87, 93].

Hence, in perpetuating a longstanding tradition of hope about Busy Beaver values [2], we predict that $S(6)$ will never be proved.

2 Enumerating Turing Machines in Tree Normal Form (TNF)

Syntactically, there are $(2ns+1)^{ns}$ Turing machines with n states and s symbols. This gives $21^{10} \simeq 16.7$ trillion possible 5-state 2-symbol Turing machines. However, naively counting this way does not account for two phenomena: (i) **unreachable transitions** and (ii) **state/symbol permutations** (permuting non-initial states and non-`0` symbols creates identical machines up to renaming).

Tree Normal Form (TNF) enumeration, introduced by Brady in 1963 in his PhD thesis [11], solves both of these problems: Turing machines are recursively *discovered* starting from the machine with no transitions defined (TNF root). Each enumerated machine is processed by a pipeline of deciders (§3) which will output either `HALT`, `NONHALT` or `UNKNOWN` for each machine:

- `HALT`. If the machine halts, it means that it has met an undefined transition and children of the machine correspond to all the possible ways of defining that undefined transition. Avoiding redundant state/symbol permutations is dealt with at this point by imposing an order on the yet-to-be-seen states/symbols.
- `NONHALT`. If the machine does not halt, all its remaining undefined transitions are unreachable and the machine is a leaf of the TNF tree.
- `UNKNOWN`. If the halting status of a machine remains unknown, it is put in the basket of *holdouts*. Having solved $S(5)$ means that there are no more 5-state holdouts.

One further optimisation: at the first level of the TNF tree, machines that make a first move to the left are avoided, as they can be symmetrised to go to the right instead. Leaves of the TNF tree that have all their transitions defined are not enumerated because they are obviously nonhalting.

Table 2: TNF metrics for $S(2), \ldots, S(5)$: number of nonhalting and halting machines in the TNF tree, total number of TNF-enumerated machines and ratio between $(4n+1)^{2n}$ (syntactic count) and TNF count.

$S(n)$	Nonhalt	Halt	Total	Ratio
$S(2)$	42	19	61	107
$S(3)$	3,645	1,772	5,417	891
$S(4)$	609,216	249,693	858,909	8,121
$S(5)$	133,005,895	48,379,894	181,385,789	91,958

TNF is unreasonably effective (Table 2): for 5-state 2-symbol machines, it reduces the search-space from 16 trillion to just 181,385,789 machines.

Coq-BB5 TNF Implementation. TNF enumeration is implemented in Coq-BB5 for the proofs of $S(2), \ldots, S(5)$; see file `TNF.v`. A `SearchQueue` abstraction with DFS capabilities is implemented; the search queue is initialised with the TNF root, and deciders (§3) are run on the enumerated Turing machines. Halting machines' children are added to the queue: the goal of the proof is to empty the queue. Importantly, `Lemma SearchQueue_upd_spec` ensures that S can be computed considering only TNF-enumerated machines.

Taking advantage of the tree structure, compilation of the $S(5)$ proof was parallelised by isolating the 12 children of the second level of the TNF tree in separate, independent files; see folder `BB5_TNF_Enumeration_Roots/`. Parallelising the compilation made the proof compile in about 45 minutes (on 13 cores) instead of 13 hours. Coq-BB5's proof of $S(2,4)$ implements TNF almost exactly as described but does not impose an order on non-`0` symbols ("quasi-TNF").

TNF Normalisation. To compute the TNF-equivalent of an arbitrary Turing machine, states are reordered by first-visit order, and all R moves are swapped with L (and vice versa) if the initial move is L. This process is not computable in general, as we cannot determine in advance which states will be visited. However, for n-state machines, knowing $S(n-2)$ makes TNF normalisation computable.

3 Deciders

In this work, we call a *decider* a program that takes as input a Turing machine $\mathcal{M}$ and that returns in finite time either `HALT`, `NONHALT`, or `UNKNOWN` depending on whether it was able to detect the machine's halting status or not. A *pipeline* consists of applying different proof techniques in sequence: a machine is tested by each decider successively until one of them outputs `HALT` or `NONHALT`.

We call *Sporadic Machines* the 13 machines that were not solved by any decider but using individual proofs of nonhalting instead, see §4. Table 3 gives an approximation of the pipeline implemented in Coq-BB5 in order to prove $S(5) = 47{,}176{,}870$.

Table 3: Approximation of the $S(5)$ pipeline as implemented in Coq-BB5. All the 181,385,789 enumerated 5-state machines are decided by this pipeline. The exact pipeline with deciders parameters as well as the $S(2,4)$ and $S(4)$ pipelines are given in [2].

$S(5)$ pipeline	Nonhalt	Halt	Total decided
1. Loops	126,994,099	48,379,711	175,373,810
2. n-gram Closed Position Set (NGramCPS)	6,005,142	0	6,005,142
3. Repeated Word List (RepWL)	6,577	0	6,577
4. Finite Automata Reduction (FAR)	23	0	23
5. Weighted FAR (WFAR)	17	0	17
6. Long halters (simulation up to 47,176,870 steps)	0	183	183
7. Sporadic machines, individual proofs	13	0	13
8. 1RB-reduction	24	0	24
Total	133,005,895	48,379,894	181,385,789

Five deciders are used in Coq-BB5 to solve $S(5)$, see Table 3: Loops, n-gram Closed Position Set (NGramCPS), Repeated Word List (RepWL), Finite Automata Reduction (FAR) and Weighted Finite Automata Reduction (WFAR). To the best of our knowledge, all these deciders are original. Solving $S(2,4)$ and previously known $S(4)$ [12] only used a subset of these deciders and required a lot less effort – e.g. no individual nonhalting proofs.

All these deciders can be expressed in the same general framework, known as Closed Tape Language (CTL) which is an abstract interpretation idea attributed to Marxen and Buntrock and was first documented by Ligocki [55]:

General Framework: Closed Tape Language (CTL). For a given Turing machine, assume there is a set C of configurations such that:

(1) C contains the initial all-`0` configuration.
(2) C is *closed* under transitions: for any $c \in C$, the configuration one step later belongs to C.
(3) C does not contain any halting configuration.

Then, the machine does not halt from any configuration of C and, in particular, from the initial all-`0` configuration.

Regularity. All our deciders but WFAR are instances of *regular* CTL, meaning that set C is a regular language – i.e. C is described using a Finite State Automaton or, equivalently, a regular expression. We say that a machine is *regular* if it can be proven nonhalting using regular CTL, otherwise we say it is *irregular*. Other regular CTL deciders were developed by The bbchallenge Collaboration, but were not used to solve $S(5)$ [25, 27, 42, 85]. Interestingly, only regular deciders are needed to solve $S(4)$ and $S(2,4)$, which, assuming $S(5)$ irregularity arguments are correct, draws a conceptual line separating $S(4)$ from $S(5)$.

Deciders vs. Verifiers. While we put all the methods under the decider umbrella it is worthwhile to mention that, in Coq-BB5, only the *verifier* part of FAR and WFAR are implemented. This means that instead of searching for a CTL set C, it is given and verified correct: Coq-BB5 hardcodes a total of 40 FAR/WFAR certificates.

`1RB`-*reduction.* Any TNF-enumerated machine whose initial transition writes a `0` and which has at least one transition writing a `1` (TNF guarantees the transition is reachable), can be transformed into a machine that starts by writing a `1` and visits the same configurations (up to state-renaming), see Coq-BB5's function `TM_to_NF`. Hence, any machine whose reduction to `1RB` already has a proof of nonhalting can be decided using the same argument. In the $S(5)$ pipeline this argument is used to decide 24 machines.

We now briefly describe each decider. Algorithms, correctness proofs, and detailed results are given in [2].

Loops. Arguably, one of the most elementary arguments to prove that a Turing machine does not halt on a given input is to show that it enters a *loop* by eventually repeating the same configuration, i.e. same tape content, head position, and state. Deciding such loops reduces to the well-known mathematical problem of detecting the cycles of a function and standard detection algorithms exist [91]. Slightly less obvious, but extremely common, is the case where a configuration is repeated but translated in space (analogously to *gliders* in cellular automata), also leading to nonhalting; these have first been described and decided in Shen Lin's 1963 PhD thesis [65]. We respectively call these types of machines: (i) *Cyclers* and (ii) *Translated Cyclers*, which we regroup under the umbrella term of loops.

Here, we develop a completely different algorithm for deciding both Cyclers and Translated Cyclers. The particularity of this algorithm is that it detects loops only by analysing the history of state, read-symbol and head-positions visited by the machine, instead of considering entire configurations (i.e. with full tape content information). Hence, in theory, this algorithm can be implemented to use less memory than previously-known algorithms. This algorithm was introduced with Coq-BB5.

Let us call the *transcript* of a machine the list of successive state-symbol pairs visited by the machine from the all-`0` tape. Surprisingly, it turns out that in order to detect loops, we only have to track when a transcript repeats the same sequence twice back-to-back. When such a repetition occurs, we use the extra information of head-position to conclude:

(1) If when entering the second repetition the head is at the same position it was at the beginning of the first repetition, then we have detected a Cycler.
(2) If, at the beginning of both repetitions, the head is at the same extremity of the tape (i.e. both positions are either both a local maximum or both a local minimum) then we have detected a Translated Cycler.

The decider for loops handles 95.48% of nonhalting 5-state machines. In this sample of nonhalting loops we find approximately 86% Translated Cyclers and 14% Cyclers which suggests that, in general, Translated Cyclers are much more common than Cyclers.

n-Gram Closed Position Set (NGramCPS). The n-gram Closed Position Set (NGramCPS) decider which we introduce here is a simplification of an earlier technique, Closed Position Set (CPS), itself introduced in `bbfind` [30]. Surprisingly, NGramCPS is a relatively simple technique which makes a potent decider as it decides 99.89% of all nonhalting enumerated 5-state machines excluding loops, see Table 3.

The method is especially potent when augmenting the binary alphabet of Turing machines to record extra information on the tape, such as a fixed-length history of previously seen (state,symbol) pairs. NGramCPS was first developed without augmentations [28] which were later introduced with Coq-BB5.

The decider considers finite, *local configurations* of a Turing machine consisting of: (i) the *n-grams* (i.e. sequences of $n > 0$ symbols from the tape alphabet) respectively to the left and to the right of the head; (ii) the state the machine is in; (iii) the symbol currently read by the head, referred to as *middle* symbol. By *n-gram*, we mean a sequence of $n > 0$ symbols from the tape alphabet (for instance, the binary alphabet $\mathcal{A} = \{0, 1\}$).

The algorithm builds a set of local configurations *potentially* reachable by the machine until either an undefined transition is met or no new configurations are added to the set, i.e. the set is closed under Turing machine operations. In the first case, the decider cannot conclude and the machine is left undecided. In the second case, the decider concludes that the machine does not halt as no undefined transition (i.e. where the machine could be asked to halt) can be reached; this is a CTL argument, see above.

Tape Alphabet Augmentations. The NGramCPS decider becomes particularly powerful for deciding 5-state 2-symbol Turing machines when augmenting the 2-symbol alphabet to store more information on the tape. Two augmentations are used in Coq-BB5:

(1) **Fixed-length history.** In this variant, tape symbols encode the current binary symbol on a cell as well as a fixed-length list of (state, binary symbol) pairs previously seen on the cell.
(2) **Least Recent Usage history (LRU).** In this variant, tape symbols encode the set of state-symbol pairs seen at that cell, in order of when it was seen last, the most recent first. One fundamental difference with the fixed-length history augmentation is that here, the history is not of fixed length but is bounded by number of states times number of symbols, i.e. 10 in the case of $S(5)$.

Furthermore, it is easy to verify that if the decider returns `NONHALT` for an augmented machine, then the non-augmented machine does not halt.

Repeated Word List (RepWL). The Repeated Word List (RepWL) technique, introduced in Coq-BB5, is based on the following simple idea: if a word (or *block*) of length $l > 0$ appears consecutively on the tape more than $T > 0$ times (with $l, T \in \mathbb{N}$ fixed) then, we assume it may repeat an unbounded number of times in the future. In practice, it means we represent configurations as regular expressions and call them *regex configurations*. For instance, consider the following configuration:

$$0^\infty \texttt{ 11100 A> } 111101010111111111 \ 0^\infty$$

Using block length $l = 2$, by grouping symbols from the head outwards, we get:

$$0^\infty \texttt{ (01) (11) (00) A> } (11)^2 \ (01)^3 \ (11)^4 \ 0^\infty$$

And, using *repeat threshold* $T = 3$ we get the following regex configuration:

$$0^\infty \texttt{ (01) (11) (00) A> } (11)^2 \ (01)^{3+} \ (11)^{3+} \ 0^\infty$$

Any repetition of more than T times the same word w is replaced by the regular expression $(w)^{T+}$ meaning that word w is repeated at least T times. Using the rules of *block simulation* (when the head faces a constant block, simulate the machine until the head leaves the block) and *regex branching* (when the head faces a block with $^+$, branch into two configurations), RepWL creates a graph of regex configurations to explore. If this graph is eventually closed and contains no halting configuration, the machine does not halt; this is a CTL argument. In the $S(5)$ pipeline, RepWL decides 6,577 machines with per-machine parameters (l, T) found by grid search ($1 \le l \le 38, 2 \le T \le 4$). RepWL graph sizes range from 42 to 143,181 nodes. RepWL also has a Haskell and a Python implementation [79, 82].

Finite Automata Reduction (FAR). Finite Automata Reduction (FAR) is a *co-CTL* technique, i.e. it is dual to the Closed Tape Language (CTL) framework given above: for a given Turing machine, we are looking for a regular language that contains the set of the machine's eventually-halting configurations and, provided that the all-`0` configuration is not in the regular language, we know that the machine does not halt.

The specificity of FAR is to restrict regular languages to a class of Nondeterministic Finite Automata (NFA) – described algebraically using the Boolean semiring $\{0, 1\}$ [20] (see [2]) – for which it is computationally simple to verify that they have the co-CTL properties: (i) reject the all-`0` initial configuration, (ii) closed under Turing machine transitions, (iii) accept all eventually-halting configurations. Configurations are represented as finite words by concatenating the tape content (from left to right, making sure to include all the `1`s) and adding the state just before the position of the head. For instance, two word-representations of 0^∞ `A> 0011` 0^∞, are $\hat{c} =$ `A0011` and $\hat{c}' =$ `000A00110000`. The initial all-`0` configuration can be encoded as `A0` or even just `A`. FAR is a *universal* regular co-CTL method: any regular co-CTL proof can be expressed in FAR's framework [85]. In Coq-BB5, 23 FAR certificates are hardcoded. FAR was originally developed as a fully-fledged decider – i.e. the verifier together with search algorithms for NFAs [5, 85]. FAR has several other implementations [37, 81].

Weighted FAR (WFAR). Weighted automata are an extension of usual finite state automata where each transition is given a weight in $\mathbb{Z}$: when a word is processed, total weight $W \in \mathbb{Z}$ is computed by summing the weights of all the encountered transitions. Weighted Finite Automata Reduction (WFAR) is an extension of FAR using deterministic weighted finite automata [44]. A WFAR automaton consists of (i) a *left deterministic weighted automaton* (ii) a *right deterministic weighted automaton* and (iii) a set of accepted *weighted configurations*. WFAR is a CTL technique (instead of co-CTL for FAR). The method was initially developed as a decider [44] and integrated to Coq-BB5 as a verifier: 17 WFAR certificates are directly hardcoded in the Coq proof.

Not all machines solved by WFAR are necessarily irregular but we strongly suspect that the 17 machines it solves in the $S(5)$ pipeline (Table 3) are irregular because intensive search did not allow finding regular CTL solutions for them. Informal irregularity arguments have been given for sporadic machines "Finned #3" and "Skelet #17" [43, 96], see §4.

4 5-State Sporadic Machines

Sporadic Machines are 13 nonhalting 5-state Turing machines that were not captured by deciders (§3) and required individual Coq proofs of nonhalting. Twelve of these machines, i.e. all but "Skelet #17" (see below), were proved nonhalting in `busycoq` [49], and then integrated into Coq-BB5. Machine "Skelet #17" was the last 5-state machine to be formally proven nonhalting in Coq, as part of Coq-BB5, achieving the proof of $S(5) = 47{,}176{,}870$ – a different proof also had been released as a standalone paper [94].

Interestingly all Sporadic Machines had been identified by Georgiev ("Skelet", see §1.3) in 2003: either as part of his 43 unsolved machines, or, in the case of "Finned Machines", marked by him as "easily provable by hand" [29]. Sporadic Machines can be arranged in three buckets:

- **Finned Machines.** These are five similar machines (#1, #2, #3, #4, #5) that hold 3 unary numbers on the tape (and can merge the middle one into its neighbour), vary them while maintaining a linear relation, and in the process ensure any deviation from this linear relation would be detected and cause a halt. These machines were solved by handcrafting nonhalting certificates similar in flavour to WFAR certificates (§3). The certificates were crafted by Blanchard, translated in Coq by Kądziołka. An argument of irregularity (§3) was given for machine "Finned #3" [43]. A later-developed irregular extension of RepWL has been reported to solve these machines.
- **Shift Overflow Counters.** This family concerns Skelet's machines #15, #26, #33, #34 and #35. These machines are similar: they implement two independent binary counters, one to the left of the tape and the other to the right. Their behaviour can be described by two distinct phases: an orderly "Counter Phase" where each counter is simply incremented and a more complex "Reset Phase" triggered by one of the counters overflowing. If the counter were to overflow again during a "Reset Phase", the machines would halt. Therefore, the proof of nonhalting depends upon demonstrating that the machine maintains a "Reset Invariant" throughout the "Reset Phase" which does not allow another overflow. These machines were first analysed and described by Ligocki, who provided an informal proof of "Skelet #34" as well as conjectures about the others [58]. They were then proved in Coq by Yuen and Kądziołka as part of `busycoq` [49].
- **Skelet #1, Skelet #10, and Skelet #17.** These three machines each have unique behaviours which we detail below.

Skelet #1. (1RB1RD_1LC0RC_1RA1LD_0RE0LB_---1RC). This machine is a Translated Cycler, but with enormous parameters: its pre-period (number of steps to wait before the pattern first appears) is about 5.42×10^{51} and its period is 8,468,569,863. This was discovered by means of accelerated simulation by Kropitz and Ligocki [48] and thorough analysis by Ligocki [59, 60]. The 10^{51} pre-period was computed later by Huang [41]. The result was confirmed correct after Kądziołka formalised it in Coq as part of `busycoq` [49]. No reasonable step-limit would allow any simulation-based decider to solve this machine.

Skelet #10 (Double Fibonacci Counter). (1RB0RA_0LC1RA_1RE1LD_1LC0LD_---0RB). This machine implements two independent *base Fibonacci* counters, one to the left of the tape and the other to the right. Counting in base Fibonacci means exploiting Zeckendorf's theorem [92]: any natural number can be expressed as a sum of Fibonacci numbers in exactly one way, excluding using numbers immediately adjacent in the Fibonacci sequence, where the Fibonacci sequence is $F = 1, 2, 3, 5, 8, 13, 21, 34 \ldots$ For instance, $17 = 1 + 3 + 13$ and this decomposition would be represented as `100101` in big-endian binary: the i^{th} bit from the right is 1 if we use F_i in the sum. Each of the two counters of Skelet #10 enumerates natural numbers in base Fibonacci, using slightly different encodings and the machine halts iff the counters ever get out of sync – which, does not happen. The machine was analysed independently by Briggs and Ligocki [14, 61] and Ligocki's proof was formalised in Coq by Kądziołka as part of `busycoq` [49]. Skelet #10 is the only known 5-state *double* Fibonacci counter, but there are several known *single* Fibonacci counters, such as 1RB0RA_0LC1RA_1LD0LC_1RE1LC_---0RB, solved by Coq-BB5's NGramCPS, see §3.

Skelet #17. (1RB---_0LC1RE_0LD1LC_1RA1LB_0RB0RA). *The final boss.* This machine manages a list of integers $n_1, \ldots, n_k \in \mathbb{N}$ represented in unary on the tape. The list can only increase and undergoes a set of complex transformations related to Gray code, and, the machine halts iff $n_1 = n_2 = 0$ and $n_3, \ldots, n_k$ are all even, which, never happens. This description was first drafted by savask [78], proven in a standalone paper by Xu [94] and, finally, formalised (using a different proof) in Coq by mxdys as part of Coq-BB5. Skelet #17 was the last 5-state machine to be solved. An argument of irregularity was given in [96]. Space-time diagrams for each Sporadic Machine are given in the arXiv version of this paper [2], which also includes a *zoology* of 5-state Turing machines.

References

[1] Scott Aaronson. 2020. The Busy Beaver Frontier. *SIGACT News* 51, 3 (Sept. 2020), 32–54. doi:10.1145/3427361.3427369 https://www.scottaaronson.com/papers/bb.pdf.

[2] The bbchallenge Collaboration, Justin Blanchard, Daniel Briggs, Konrad Deka, Nathan Fenner, Yannick Forster, Georgi Georgiev (Skelet), Matthew L. House, Rachel Hunter, Iijil, Maja Kądziołka, Pavel Kropitz, Shawn Ligocki, mxdys, Mateusz Naściszewski, savask, Tristan Stérin, Chris Xu, Jason Yuen, and Théo Zimmermann. 2026. Determination of the fifth Busy Beaver value. arXiv:2509.12337 [cs.LO] https://arxiv.org/abs/2509.12337 v2.

[3] bbchallenge wiki. 2025. Champions. Wiki: https://wiki.bbchallenge.org/wiki/Champions. Accessed: 2025-08-10.

[4] A. M. Ben-Amram, B. A. Julstrom, and U. Zwick. 1996. A note on busy beavers and other creatures. *Mathematical systems theory* 29, 4 (01 July-August 1996), 375–386. doi:10.1007/BF01192693

[5] Justin Blanchard. 2022. Finite Automata Reduction (FAR). https://github.com/bbchallenge/bbchallenge-deciders/tree/main/decider-finite-automata-reduction.

[6] bmc7505. 2023. The Busy Beaver Challenge. Hacker News item. https://news.ycombinator.com/item?id=34689081 Online; posted on Feb 7, 2023; https://news.ycombinator.com/item?id=34689081.

[7] Mathieu Boespflug, Maxime Dénès, and Benjamin Grégoire. 2011. Full Reduction at Full Throttle. In *Certified Programs and Proofs*, Jean-Pierre Jouannaud and Zhong Shao (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 362–377.

[8] Matthew Bolan, Joachim Breitner, Jose Brox, Nicholas Carlini, Mario Carneiro, Floris van Doorn, Martin Dvorak, Andrés Goens, Aaron Hill, Harald Husum, Hernán Ibarra Mejia, Zoltan A. Kocsis, Bruno Le Floch, Amir Livne Bar-on, Lorenzo Luccioli, Douglas McNeil, Alex Meiburg, Pietro Monticone, Pace P. Nielsen, Emmanuel Osalotioman Osazuwa, Giovanni Paolini, Marco Petracci, Bernhard Reinke, David Renshaw, Marcus Rossel,

Cody Roux, Jérémy Scanvic, Shreyas Srinivas, Anand Rao Tadipatri, Terence Tao, Vlad Tsyrklevich, Fernando Vaquerizo-Villar, Daniel Weber, and Fan Zheng. Dec . The Equational Theories Project. https://github.com/teorth/equational_theories https://github.com/teorth/equational_theories v0.2.0 swh:1:dir:426b52ba40033a37c18474ef44068e23c55df4bc.

[9] Matthew Bolan, Joachim Breitner, Jose Brox, Nicholas Carlini, Mario Carneiro, Floris van Doorn, Martin Dvorak, Andrés Goens, Aaron Hill, Harald Husum, Hernán Ibarra Mejia, Zoltan A. Kocsis, Bruno Le Floch, Amir Livne Bar-on, Lorenzo Luccioli, Douglas McNeil, Alex Meiburg, Pietro Monticone, Pace P. Nielsen, Emmanuel Osalotioman Osazuwa, Giovanni Paolini, Marco Petracci, Bernhard Reinke, David Renshaw, Marcus Rossel, Cody Roux, Jérémy Scanvic, Shreyas Srinivas, Anand Rao Tadipatri, Terence Tao, Vlad Tsyrklevich, Fernando Vaquerizo-Villar, Daniel Weber, and Fan Zheng. 2025. The Equational Theories Project: Advancing Collaborative Mathematical Research at Scale. arXiv:2512.07087 [math.RA] https://arxiv.org/abs/2512.07087 https://arxiv.org/abs/2512.07087.

[10] Samuel Boutin. 1997. Using reflection to build efficient and certified decision procedures. In *International Symposium on Theoretical Aspects of Computer Software*. Springer, 515–529.

[11] Allen H. Brady. 1964. *Solutions of restricted cases of the halting problem applied to the determination of particular values of a non-computable function.* Ph. D. Dissertation. Oregon State University.

[12] Allen H. Brady. 1983. The Determination of the Value of Rado's Noncomputable Function $\Sigma(k)$ for Four-State Turing Machines. *Math. Comp.* 40, 162 (1983), 647–665. http://www.jstor.org/stable/2007539

[13] Allen H Brady. 1990. The Busy Beaver Game and the Meaning of Life. In *The Universal Turing Machine: A Half-Century Survey*. Oxford University Press. arXiv:https://academic.oup.com/book/0/chapter/422572862/chapter-pdf/52547412/isbn-9780198537748-book-part-9.pdf doi:10.1093/oso/9780198537748.003.0009

[14] Dan Briggs. 2010. Turing. https://github.com/danbriggs/Turing.

[15] Ben Brubaker. 2024. Amateur Mathematicians Find Fifth "Busy Beaver" Turing Machine. *Quanta Magazine* (2 Jul 2024). https://www.quantamagazine.org/amateur-mathematicians-find-fifth-busy-beaver-turing-machine-20240702/ Online; published July 2, 2024; https://www.quantamagazine.org/amateur-mathematicians-find-fifth-busy-beaver-turing-machine-20240702/.

[16] Code Golf Addict. 2016. list27.txt. https://gist.github.com/anonymous/a64213f391339236c2fe31f8749a0df6.

[17] Johan Commelin and the Lean community. 2022. *Completion of the Liquid Tensor Experiment.* Accessed: 2025-10-18.

[18] Johan Commelin and Adam Topaz. 2023. Abstraction boundaries and spec driven development in pure mathematics. arXiv:arXiv:2309.14870

[19] Thierry Coquand, Jean Gallier, and Le Cedex. 1999. A Proof of Strong Normalization For the Theory of Constructions Using a Kripke-Like Interpretation. In *Informal Proceedings of the Workshop on Logical Frameworks*.

[20] R.A. Cuninghame-Green. 1991. Minimax algebra and applications. *Fuzzy Sets and Systems* 41, 3 (1991), 251–267. doi:10.1016/0165-0114(91)90130-I

[21] Martin Davis, Yuri Matiyasevich, and Julia Robinson. 1976. Hilbert's Tenth Problem: Diophantine Equations – Positive Aspects of a Negative Solution. In *Mathematical Developments Arising from Hilbert Problems*, Felix E. Browder (Ed.). Proceedings of Symposia in Pure Mathematics, Vol. 28. American Mathematical Society, 323–378.

[22] Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. 2015. The Lean Theorem Prover (System Description). In *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings (Lecture Notes in Computer Science, Vol. 9195)*, Amy P. Felty and Aart Middeldorp (Eds.). Springer, 378–388. doi:10.1007/978-3-319-21401-6_26

[23] DeepMind (Google). 2025. Formal Conjectures: A collection of formalized statements of conjectures in Lean. GitHub repository. URL: https://github.com/google-deepmind/formal-conjectures.

[24] A Dubickas. 2009. ON INTEGER SEQUENCES GENERATED BY LINEAR MAPS. *Glasgow Mathematical Journal* 51, 2 (2009), 243–252. doi:10.1017/S0017089508004655

[25] Frans Faase. 2022. Symbolic TM. https://github.com/FransFaase/SymbolicTM.

[26] Nathan Fenner. 2022. bbchallenge Dafny deciders. https://github.com/Nathan-Fenner/bbchallenge-dafny-deciders.

[27] Nathan Fenner. 2022. bbchallenge-regexy-decider. https://github.com/Nathan-Fenner/bbchallenge-regexy-decider.

[28] Nathan Fenner. 2023. n-GRAM CPS Decider. https://github.com/Nathan-Fenner/bb-simple-n-gram-cps.

[29] Georgi Georgiev. 2003. Busy Beaver nonregular machines for class TM(5). https://skelet.ludost.net/bb/nreg.html. Accessed: 2025-04-03.

[30] Georgi Georgiev. 2003. Busy Beaver prover - `bbfind`. https://skelet.ludost.net/bb/. Accessed: 2024-11-25.

[31] Georges Gonthier. 2008. Formal proof–the four-color theorem. *Notices of the AMS* 55, 11 (2008), 1382–1393.

[32] Georges Gonthier. 2023. A computer-checked proof of the Four Color Theorem.

[33] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, et al. 2013. A machine-checked proof of the odd order theorem. In *International conference on interactive theorem proving*. Springer, 163–179.

[34] Georges Gonthier and Assia Mahboubi. 2010. An introduction to small scale reflection in Coq. *Journal of formalized reasoning* 3, 2 (2010), 95–152.

[35] Timothy Gowers and Michael Nielsen. 2009. Massively collaborative mathematics. *Nature* 461, 7266 (01 Oct 2009), 879–881. doi:10.1038/461879a

[36] Benjamin Grégoire and Xavier Leroy. 2002. A compiled implementation of strong reduction. *SIGPLAN Not.* 37, 9 (Sept. 2002), 235–246. doi:10.1145/583852.581501

[37] Tony Guilfoyle. 2023. FAR C++ reproduction. https://github.com/TonyGuil/bbchallenge/tree/main/FAR.

[38] Thomas Hales, Mark Adams, Gertrud Bauer, {Tat Dat} Dang, John Harrison, {Le Truong} Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, {Tat Thang} Nguyen, {Quang Truong} Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason Rute, Alexey Solovyev, {Thi Hoai An} Ta, {Nam Trung} Tran, {Thi Diep} Trieu, Josef Urban, Ky Vu, and Roland Zumkeller. 2017. A FORMAL PROOF of the Kepler CONJECTURE. *Forum of Mathematics, Pi* 5 (2017). doi:10.1017/fmp.2017.1

[39] James Harland. 2006. The Busy Beaver, the Placid Platypus and other Crazy Creatures. In *Proceedings of the 12th Computing: The Australasian Theroy Symposium - Volume 51* (Hobart, Australia) *(CATS '06)*. Australian Computer Society, Inc., AUS, 79–86.

[40] Joachim Hertel. 2009. Computing the Uncomputable Rado Sigma Function. *The Mathematica Journal* 11(2): 270-283 (2009). doi:10.3888/tmj.11.2-8

[41] hipparcos. 2025. turing_machine. https://github.com/jhuang97/turing_machine.

[42] Iijil. 2022. Bruteforce-CTL. https://github.com/Iijil1/Bruteforce-CTL.

[43] Iijil. 2023. Finned 3 is irregular. https://discuss.bbchallenge.org/t/10756090-is-irregular/137.

[44] Iijil1. 2025. *Iijil1/MITMWFAR: v1.0.0.* doi:10.5281/zenodo.14914502 https://doi.org/10.5281/zenodo.14914502.

[45] Owen Kellett. 2005. *A multi-faceted attack on the busy beaver problem.* Master's Thesis. https://homepages.hass.rpi.edu/heuveb/research/BB/downloads/OwenThesis.pdf.

[46] Reinhard Kleindl. 2025. Durchbruch beim mathematischen Problem des „fleißigen Bibers". *Der Standard* (6 Jan. 2025). https://www.derstandard.de/story/3000000249211/durchbruch-beim-mathematischen-problem-des-fleissigen-bibers https://www.derstandard.de/story/3000000249211/durchbruch-beim-mathematischen-problem-des-fleissigen-bibers.

[47] Donald E. Knuth. 1976. Mathematics and Computer Science: Coping with Finiteness. *Science* 194, 4271 (1976), 1235–1242. doi:10.1126/science.194.4271.1235

[48] Pavel Kropitz. 2023. bbc. https://github.com/univerz/bbc/tree/no1.

[49] Maja Kądziołka. 2023. busycoq. https://github.com/meithecatte/busycoq/blob/master/verify/Skelet1.v.

[50] G Lafitte and C Papazian. 2007. The fabric of small Turing machines. In *Computation and Logic in the Real World, Proceedings of the Third Conference on Computabiliy in Europe, CiE '07* (Siena, Italy). Springer-Verlag, Berlin, Heidelberg, 219–227.

[51] Jeffrey C. Lagarias. 1985. The 3x + 1 Problem and Its Generalizations. *The American Mathematical Monthly* 92, 1 (1985), 3–23. http://www.jstor.org/stable/2322189

[52] David Larousserie. 2024. Mathématiques : le défi du "castor affairé" résolu. *Le Monde* (17 July 2024). https://www.lemonde.fr/sciences/article/2024/07/17/mathematiques-le-defi-du-castor-affaire-resolu_6251337_1650684.html https://www.lemonde.fr/sciences/article/2024/07/17/mathematiques-le-defi-du-castor-affaire-resolu_6251337_1650684.html.

[53] K. Rustan M. Leino. 2010. Dafny: an automatic program verifier for functional correctness. In *Proceedings of the 16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning* (Dakar, Senegal) *(LPAR'10)*. Springer-Verlag, Berlin, Heidelberg, 348–370.

[54] Yijun Leng. 2025. *lengyijun/goldbach_tm: 25-state Turing machine.* doi:10.5281/zenodo.14635917

[55] Shawn Ligocki. 2022. *CTL Filter.* Accessed: 2023-03-20.

[56] Shawn Ligocki. 2022. *Mother of Giants.* Accessed: 2025-08-10.

[57] Shawn Ligocki. 2023. *BB(3, 3) is Hard (Bigfoot).* Accessed: 2025-08-10.

[58] Shawn Ligocki. 2023. *Shift Overflow Counters.* Accessed: 2025-04-04.

[59] Shawn Ligocki. 2023. *Skelet #1 is infinite ... we think.* Accessed: 2024-02-11.

[60] Shawn Ligocki. 2023. *Skelet #1: What I Know.* Accessed: 2025-04-07.

[61] Shawn Ligocki. 2023. *Skelet #10: Double Fibonacci Counter.* Accessed: 2025-04-07.

[62] Shawn Ligocki. 2024. *BB(2, 5) is Hard (Hydra).* Accessed: 2025-08-10.

[63] Shawn Ligocki. 2024. *BB(3, 4) > Ack(14).* https://www.sligocki.com/2024/05/22/bb-3-4-a14.html Accessed: 2025-08-10.

[64] Shawn Ligocki. 2024. *BB(6) is Hard (Antihydra).* Accessed: 2025-08-10.

[65] Shen Lin. 1963. *Computer studies of Turing machine problems.* Ph. D. Dissertation. Ohio State University, Graduate School.

[66] H. Marxen and Jürgen Buntrock. 1990. Attacking the Busy Beaver 5. *Bull. EATCS* 40 (1990), 247–251.

[67] Pascal Michel. [n. d.]. The Busy Beaver Competitions. Website: https://bbchallenge.org/~pascal.michel/ha.html. Accessed 04-08-2025.

[68] Pascal Michel. 2004. Small Turing machines and generalized busy beaver competition. *Theoretical Computer Science* 326, 1 (2004), 45–56. doi:10.1016/j.tcs.2004.05.008

[69] Pascal Michel. 2022. *The Busy Beaver Competition: a historical survey.* Technical Report. arXiv:0906.3749 [math.LO] https://arxiv.org/abs/0906.3749v7 https://arxiv.org/abs/0906.3749v7.

[70] Quanta Magazine. 2025. Amateurs Solve a Famous Computer Science Problem On Discord. YouTube video. https://www.youtube.com/watch?v=rmx3FBPzDuk Uploaded by Quanta Magazine on YouTube; https://www.youtube.com/watch?v=rmx3FBPzDuk.

[71] Tibor Radó. 1962. On non-computable functions. *Bell System Technical Journal* 41, 3 (1962), 877–884. https://archive.org/details/bstj41-3-877/mode/2up.

[72] Tibor Radó. 1963. On a simple source for non-computable functions. In *Proceedings of the Symposium on Mathematical Theory of Automata (New York)* (Ney York, USA). Polytechnic Press of the polytechnic Institute of Brooklyn.

[73] *Redacted*. 2022. *bbchallenge.org, initial release.* doi:10.5281/zenodo.14955828 https://doi.org/10.5281/zenodo.14955828.

[74] *Redacted*. 2025. *Coq-BB5 release v1.0.0.* doi:10.5281/zenodo.17061968 https://doi.org/10.5281/zenodo.17061968.

[75] Rohan Ridenour. 2024. "Use decz for ZFC (636 states)". GitHub commit. Commit 6fc33bef6ba8885d26aed94c83e88bdabbedb0f1; https://github.com/CatsAreFluffy/metamath-turing-machines/commit/6fc33bef6ba8885d26aed94c83e88bdabbedb0f1.

[76] Johannes Riebel. 2023. *The Undecidability of BB(748).* Bachelor's Thesis. University of Augsburg. https://www.ingo-blechschmidt.eu/assets/bachelor-thesis-undecidability-bb748.pdf.

[77] Kyle Ross, Owen Kellett, Bram van Heuveln, and Selmer Bringsjord. 2005. A New-Millenium Attack on the Busy Beaver Problem. https://homepages.hass.rpi.edu/heuveb/research/BB/downloads/superpaper.pdf.

[78] savask. 2023. Analysis of Skelet's machine 17. https://chrisxudoesmath.com/papers/skelet17_savasks_analysis.pdf. Accessed: 2025-04-07.

[79] savask. 2024. RepWL Haskell implementation. https://github.com/savask/turing/blob/main/Repeat.hs.

[80] Tristan Stérin and Damien Woods. 2024. Hardness of Busy Beaver Value BB(15). In *Reachability Problems*, Laura Kovács and Ana Sokolova (Eds.). Springer Nature Switzerland, Cham, 120–137.

[81] Tristan Stérin. 2023. FAR Python reproduction. https://github.com/bbchallenge/bbchallenge-deciders/tree/main/decider-finite-automata-reduction-reproduction.

[82] Tristan Stérin. 2024. RepWL Python implementation. https://github.com/bbchallenge/bbchallenge-deciders/tree/main/decider-repeated-word-list-reproduction.

[83] Terrence Tao. 2024. A pilot project in universal algebra to explore new ways to collaborate and use machine assistance? Blog: https://terrytao.wordpress.com/2024/09/25/. Accessed: 2025-05-11.

[84] Laetitia Teodorescu, Guillaume Baudart, Emilio Jesús Gallego Arias, and Marc Lelarge. 2024. NLIR: Natural Language Intermediate Representation for Mechanized Theorem Proving. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24.* https://openreview.net/forum?id=QzOc0tpdef

[85] The bbchallenge Collaboration, Justin Blanchard, Konrad Deka, Nathan Fenner, Tony Guilfoyle, Iijil, Maja Kądziołka, Pavel Kropitz, Shawn Ligocki, Pascal Michel, Mateusz Naściszewski, and Tristan Stérin. 2025. Turing machines deciders, part I. arXiv:2504.20563 [CS.LO] https://arxiv.org/abs/2504.20563 https://arxiv.org/abs/2504.20563.

[86] The Coq Development Team. 2024. *The Coq Proof Assistant v8.20.* doi:10.5281/zenodo.14542673 https://doi.org/10.5281/zenodo.14542673.

[87] Trieu H. Trinh, Yuhuai Wu, Quoc V. Le, He He, and Thang Luong. 2024. Solving olympiad geometry without human demonstrations. *Nature* 625, 7995 (01 Jan 2024), 476–482. doi:10.1038/s41586-023-06747-5

[88] John Tromp. 2020. Busy Beaver for lambda calculus BBλ. https://oeis.org/A333479. Entry A333479 in The On-Line Encyclopedia of Integer Sequences.

[89] Andrew Wade. 2025. "Alignment shenanigans (ZFC → 549)". GitHub commit. Commit 5d676aec074a94f598959cb3b7733a8f7781762f; https://github.com/andrew-j-wade/metamath-turing-machines/commit/5d676aec074a94f598959cb3b7733a8f7781762f.

[90] S. S. Wainer. 1970. A classification of the ordinal recursive functions. *Archiv für Mathematische Logik und Grundlagenforschung* 13, 1-2 (1970), 136–153.

[91] Wikipedia. 2025. Cycle detection — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Cycle%20detection&oldid=1265904359. Aaccessed: 2025-01-30.

[92] Wikipedia. 2025. Zeckendorf's theorem — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Zeckendorf's%20theorem&oldid=1242695626. Accessed 2025-04-07.

[93] Zijian Wu, Suozhi Huang, Zhejian Zhou, Huaiyuan Ying, Jiayu Wang, Dahua Lin, and Kai Chen. 2024. InternLM2.5-StepProver: Advancing Automated Theorem Proving via Expert Iteration on Large-Scale LEAN Problems. arXiv:2410.15700 [cs.AI] https://arxiv.org/abs/2410.15700

[94] Chris Xu. 2024. Skelet #17 and the fifth Busy Beaver number. arXiv:2407.02426 [math.CO] https://arxiv.org/abs/2407.02426 https://arxiv.org/abs/2407.02426.

[95] Adam Yedidia and Scott Aaronson. 2016. A Relatively Small Turing Machine Whose Behavior Is Independent of Set Theory. *Complex Systems* 25, 4 (Dec. 2016), 297–328. doi:10.25088/complexsystems.25.4.297

[96] Daniel Yuan and Justin Blanchard. 2024. Skelet 17 is irregular. https://cosearch.bbchallenge.org/contribution/rgr5sxom.

[97] Scott Aaronson Yuri Matiyasevich, Stefan O'Rear. 2016. metamath - Riemann Hypothesis. https://github.com/sorear/metamath-turing-machines/blob/master/riemann-matiyasevich-aaronson.nql.