The Busy Beaver Frontier

Scott Aaronson[*]

Abstract

The Busy Beaver function, with its incomprehensibly rapid growth, has captivated generations of computer scientists, mathematicians, and hobbyists. In this survey, I offer a personal view of the BB function 58 years after its introduction, emphasizing lesser-known insights, recent progress, and especially favorite open problems. Examples of such problems include: when does the BB function first exceed the Ackermann function? Is the value of $\mathrm{BB}(20)$ independent of set theory? Can we prove that $\mathrm{BB}(n+1) > 2^{\mathrm{BB}(n)}$ for large enough n? Given $\mathrm{BB}(n)$, how many advice bits are needed to compute $\mathrm{BB}(n+1)$? Do all Busy Beavers halt on all inputs, not just the 0 input? Is it decidable whether $\mathrm{BB}(n)$ is even or odd?

1 Introduction

The Busy Beaver[1] function, defined by Tibor Radó [13] in 1962, is an *extremely* rapidly-growing function, defined by maximizing over the running times of all n-state Turing machines that eventually halt. In my opinion, the BB function makes the concepts of computability and uncomputability more vivid than anything else ever invented. When I recently taught my 7-year-old daughter Lily about computability, I did so almost entirely through the lens of the ancient quest to name the biggest numbers one can. Crucially, that childlike goal, if pursued doggedly enough, *inevitably* leads to something like the BB function, and hence to abstract reasoning about the space of possible computer programs, the halting problem, and so on.[2]

Let's give a more careful definition. Following Radó's conventions, we consider Turing machines over the alphabet $\{0,1\}$, with a 2-way infinite tape, states labeled by $1,\ldots,n$ (state 1 is the initial state), and transition arrows labeled either by elements of $\{0,1\}\times\{L,R\}$ (in which case the arrows point to other states), or else by "Halt" (in which case the arrows point nowhere).[3]

Given a machine M, let $s(M)$ be the number of steps that M takes before halting, including the final "Halt" step, when M is run on an all-0 initial tape. If M never halts then we set $s(M) := \infty$.

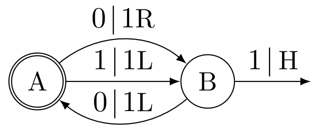
Also, let $T(n)$ be the set of Turing machines with n states. For later reference, note that $|T(n)| = (4n+1)^{2n}$. A calculation reveals that, if we identify machines that are equivalent under permuting the states, then each n-state machine can be specified using $n\log_2 n + O(n)$ bits.

[*]University of Texas at Austin. Email: aaronson@utexas.edu. Supported by a Vannevar Bush Fellowship from the US Department of Defense, a Simons Investigator Award, and the Simons "It from Qubit" collaboration.

[1]Radó named the function "Busy Beaver" after the image of a beaver moving back and forth across the Turing machine tape, writing symbols. This survey will have no beaver-related puns.

[2]I developed this perspective for broader audiences in a 1999 essay (`https://www.scottaaronson.com/writings/bignumbers.html`) as well as a 2017 public lecture (`https://www.scottaaronson.com/blog/?p=3445`).

[3]Radó also allowed the machines to write a symbol and move the tape head on the final "Halt" step. We omit this, since for all the Busy Beaver functions we'll consider, either the choice of what to do on the last step is irrelevant, or else we can assume without loss of generality that the machine writes '1' and moves one square (say) to the left.



initial state	⋯	0	0	0	0A	0	0	⋯
step 1	⋯	0	0	0	1	0B	0	⋯
step 2	⋯	0	0	0	1A	1	0	⋯
step 3	⋯	0	0	0B	1	1	0	⋯
step 4	⋯	0	0A	1	1	1	0	⋯
step 5	⋯	0	1	1B	1	1	0	⋯
step 6 (halt)	⋯	0	1	1H	1	1	0	⋯

Figure 1: A 2-state Busy Beaver and its execution on an initially all-0 tape. Starting in state A, the machine finds a 0 on its tape and therefore follows the arrow labeled $0|1R$, which causes it to replace the 0 by a 1, move one square to the right, and transition into state B, and so on until the machine reaches the Halt arrow on its 6^{th} step.

We now define the Busy Beaver function as follows:

$$\mathrm{BB}(n) := \max_{M \in T(n)\,:\,s(M)<\infty} s(M).$$

In words: among all the finitely many n-state Turing machines, some run forever when started on an all-0 input tape and some halt. The n^{th} Busy Beaver number, $\mathrm{BB}(n)$, is obtained by throwing away all the n-state machines that run forever, and then maximizing the number of steps over all the machines that halt. A machine M that achieves the maximum is also called an "n-state Busy Beaver." What Radó called the "Busy Beaver Game" is the game of *finding* these Busy Beavers, and the corresponding $\mathrm{BB}(n)$ values, for n as large as possible.

To illustrate, Figure 1 shows a 2-state Turing machine that runs for 6 steps on an initially all-0 tape, thus demonstrating that $\mathrm{BB}(2) \geq 6$. In fact this machine turns out to be a 2-state Busy Beaver, so that $\mathrm{BB}(2) = 6$ exactly (see [9]).

Let's make an observation that we might call the "understatement of the century":

Proposition 1 $\mathrm{BB}(n) < \mathrm{BB}(n+1)$ *for all* n.

Proof. Given an n-state Busy Beaver M, we can add an $(n+1)^{st}$ state—the new initial state—which stalls for a single time step, writes a 0 on the tape, and then transitions to running M. ∎

Technically, what we've called $\mathrm{BB}(n)$ is what Radó called the "shift function," or $S(n)$. Radó also defined a "ones function," $\Sigma(n) \leq S(n)$, which counts the maximum number of 1's that an n-state Turing machine can have on its tape at the time of halting, assuming an all-0 initial tape. Other variants, such as the maximum number of visited tape squares, can also be studied; these variants have interesting relationships to each other but all have similarly explosive growth.[4] Personally, I find the shift function to be by far the most natural choice, so I'll focus on it in this survey, mentioning the Σ variant only occasionally.

In the literature on Busy Beaver, people also often study the function $\mathrm{BB}(n,k)$, which is the generalization of $\mathrm{BB}(n)$ to Turing machines with a k-symbol alphabet. (Thus, $\mathrm{BB}(n) = \mathrm{BB}(n,2)$.)

[4]Empirically, for example, one seems to have $\mathrm{BB}(n) \approx \Sigma(n)^2$, as we'll discuss in Section 5.4.

People have also studied variants with a 2-dimensional tape, or where the tape head is allowed to stay still in addition to moving left or right, etc. More broadly, given *any* programming language L, whose programs consist of bit-strings, one can define a Busy Beaver function for L-programs:

$$\mathrm{BB}_L(n) := \max_{P \in L \cap \{0,1\}^{\leq n} \,:\, s(P) < \infty} s(P),$$

where $s(P)$ is the number of steps taken by P on a blank input. Alternatively, some people define a "Busy Beaver function for L-programs" using *Kolmogorov complexity.* That is, they let $\mathrm{BB}'_L(n)$ be the largest integer m such that $\mathrm{K}_L(m) \leq n$, where $\mathrm{K}_L(m)$ is the length of the shortest L-program whose output is m on a blank input.

In this survey, however, I'll keep things simple by focusing on Radó's original shift function, $\mathrm{BB}(n) = S(n)$, except when there's some conceptual reason to consider a variant.

1.1 In Defense of Busy Beaver

The above definitional quibbles raise a broader objection: isn't the Busy Beaver function *arbitrary*? If it depends so heavily on a particular computational model (Turing machines) and complexity measure (number of steps), then isn't its detailed behavior really a topic for recreational programming rather than theoretical computer science?

My central reason for writing this survey is to meet that objection: to show you the insights about computation that I think emerged from the study of the BB function, and especially to invite you to work on the many open problems that remain.

The charge of arbitrariness can be answered in a few ways. One could say: *of course* we don't care about the specific BB function, except insofar as it illustrates the general *class* of functions with BB-like uncomputable growth. And indeed, much of what I'll discuss in this survey carries over to the entire class of functions.

Even so, it would be strange for a chess master to say defensively: "no, of course I don't care about chess, except insofar as it illustrates the class of *all* two-player games of perfect information." Sometimes the only way to make progress in a given field is first to agree on semi-arbitrary rules—like those of chess, baseball, English grammar, or the Busy Beaver game—and then to pursue the consequences of those rules intensively, on the lookout for unexpected emergent behavior.

And for me, *unexpected emergent behavior* is really the point here. The space of all possible computer programs is wild and vast, and human programmers tend to explore only tiny corners of it—indeed, good programming practice is often about making those corners even tinier. But if we want to learn more about program-space, then cataloguing random programs isn't terribly useful either, since a program can only be judged against some *goal.* The Busy Beaver game solves this dilemma by relentlessly optimizing for a goal completely orthogonal to any normal programmer's goal, and seeing what kinds of programs result.

But why *Turing machines*? For all their historic importance, haven't Turing machines been completely superseded by better alternatives—whether stylized assembly languages or various code-golf languages or Lisp? As we'll see, there *is* a reason why Turing machines were a slightly unfortunate choice for the Busy Beaver game: namely, the loss incurred when we encode a state transition table by a string of bits or vice versa. But Turing machines also turn out to have a massive advantage that compensates for this. Namely, because Turing machines have no "syntax" to speak of, but only graph structure, we *immediately* start seeing interesting behavior even with machines of only 3, 4, or 5 states, which are feasible to enumerate. And there's a second advantage.

Precisely *because* the Turing machine model is so ancient and fixed, whatever emergent behavior we find in the Busy Beaver game, there can be no suspicion that we "cheated" by changing the model until we got the results we wanted.

In short, the Busy Beaver game seems like about as good a yardstick as any for gauging humanity's progress against the uncomputable.

1.2 A Note on the Literature

Most 20^{th}-century research on the Busy Beaver function was recorded in journal articles—many of them still well worth reading, even when their results have been superseded. For better or worse, though, many of the newer bounds that I'll mention in this survey have so far been documented only in code repositories and pseudonymous online forum posts.

The clearest, most comprehensive source that I've found, for the history and current status of attempts to pin down the values of $\mathrm{BB}(n)$ and its variants, is a 73-page survey article by Pascal Michel [12].[5] Meanwhile, for the past decade, the so-called Googology Wiki[6] has been the central clearinghouse for discussion of huge numbers: for example, it's where the best current lower bound on $\mathrm{BB}(7)$ was announced in 2014. Heiner Marxen's Busy Beaver page[7] and the Busy Beaver Wikipedia entry[8] are excellent resources as well.

2 Basic Properties

I'll now review the central "conceptual properties" of the BB function—by which I mean, those properties that would still hold if we'd defined BB using RAM machines, Lisp programs, or any other universal model of computation instead of Turing machines.

First, $\mathrm{BB}(n)$ grows *so* rapidly that the ability to compute any upper bound on it would imply the ability to solve the halting problem:

Proposition 2 *One can solve the halting problem, given oracle access to any function* $f : \mathbb{N} \to \mathbb{N}$ *such that* $f(n) \geq \mathrm{BB}(n)$ *for all* n. *Hence, no such* f *can be computable.*

Proof. To decide whether an n-state Turing machine M halts on the all-0 input, simply run M for up to $f(n)$ steps. If M hasn't halted yet, then by the definition of BB, it never will. ■

Indeed, notice that an n-state Busy Beaver, if we had it, would serve as an $O(n \log n)$-bit advice string, "unlocking" the answers to the halting problem for *all* $n^{O(n)}$ Turing machines with n states or fewer. Unfortunately, to use this advice, we'd still need to do a computation that lasted $\mathrm{BB}(n)$ steps—but at least we'd know in advance that the computation would halt!

Conversely, it's clear that one can compute BB given an oracle for the language HALT, which consists of descriptions of all Turing machines that halt on the all-0 input. Thus, BB is Turing-equivalent to the halting problem.

Although Proposition 2 goes some way toward explaining the Busy Beaver function's explosive growth, it's not sharp. For example, it doesn't imply that $\mathrm{BB}(n)$ dominates every computable function f, but only that $\mathrm{BB}(n) \geq f(n)$ for *infinitely many* n. The following proposition, which is incomparable with Proposition 2, fixes this defect.

[5] See also the associated website, `http://www.logique.jussieu.fr/~michel/bbc.html`

[6] `https://googology.wikia.org/wiki/Googology_Wiki`

[7] `http://turbotm.de/~heiner/BB/`

[8] `https://en.wikipedia.org/wiki/Busy_beaver`

Proposition 3 *Let $f : \mathbb{N} \to \mathbb{N}$ be any computable function. Then there exists an n_f such that* $\mathrm{BB}(n) > f(n)$ *for all $n \geq n_f$.*

Proof. Let M_f be a Turing machine that computes $f(n)$, for any n encoded on M_f's input tape. Suppose M_f has c states. Then for all n, there exists a Turing machine $M_{f,n}$ with $c + O(\log n)$ states that, given an all-0 input tape, first writes n onto the input tape, then simulates M_f in order to compute $f(n)$, and finally executes an empty loop for (say) $f(n)^2$ steps. Hence

$$\mathrm{BB}(c + O(\log n)) > f(n)$$

for all n, from which the proposition follows. ∎

Note, in passing, that Proposition 3 is a "different" way to prove the existence of uncomputable functions, one that never explicitly appeals to diagonalization.[9]

Finally, given that the Busy Beaver function is uncomputable, one could ask how many of its values are "humanly knowable." Once we fix an axiomatic basis for mathematics, the answer turns out to be "at most finitely many of them," and that by a simple application of Gödel.

Proposition 4 *Let T be a computable and arithmetically sound axiomatic theory. Then there exists a constant n_T such that for all $n \geq n_T$, no statement of the form "*$\mathrm{BB}(n) = k$*" can be proved in T.*

Proof. Let M be a Turing machine that, on the all-0 input tape, enumerates all possible proofs in T, halting only if it finds a proof of $0 = 1$. Then since T is sound, M never halts. But T can't *prove* that M never halts, since otherwise T would prove its own consistency, violating the second incompleteness theorem.

Now suppose M has n_T states. Then for all $n \geq n_T$, the value of $\mathrm{BB}(n)$ must be unprovable in T. For otherwise T could prove that M never halted, by simulating M for $\mathrm{BB}(n) \geq \mathrm{BB}(n_T)$ steps and verifying that M hadn't halted by then. ∎

More than anything else, for me Proposition 4 captures the romance of the Busy Beaver function. Even though its values are mathematically well-defined, there can never be any systematic way to make progress in determining them. Each additional value (if it can be known at all) is a fresh challenge, requiring fresh insights and ultimately even fresh axioms.[10]

Indeed, Proposition 4 is just a special case of a more general phenomenon. As we'll see in Section 5.2, there happens to be a Turing machine with 27 states, which given a blank input, runs through all even numbers 4 and greater, halting if it finds one that isn't a sum of two primes. This machine halts if and only Goldbach's Conjecture is false, and by the definition of BB, it halts in at most $\mathrm{BB}(27)$ steps if it halts at all. But this means that knowing the value of $\mathrm{BB}(27)$ would *settle Goldbach's Conjecture*—at least in the abstract sense of reducing that problem to a finite, $\mathrm{BB}(27)$-step computation. Analogous remarks apply to the Riemann Hypothesis and every other unproved mathematical conjecture that's expressible as a "Π_1 sentence" (that is, as a statement that some computer program runs forever). In that sense, the values of the Busy Beaver function—and not only that, but its relatively *early* values—encode a large portion of all interesting mathematical truth, and they do so purely in virtue of how large they are.

[9] Implicitly, one might call the proof "diagonalization-adjacent."

[10] In the statement of Proposition 4, one might wonder about the dependence of the constant n_T on the theory T. For example: is it possible that for every n, there's *some* large-cardinal extension of ZF set theory that's powerful enough to settle the value of $\mathrm{BB}(n)$? A short answer is that, even if so, there can be no systematic way to *find* those large-cardinal extensions. For if there were, then $\mathrm{BB}(n)$ would become computable. (Incidentally, this gives an alternative way to prove Proposition 4 itself.)

3 Above and Below Busy Beaver

It's natural to ask if there are functions that grow even *faster* than Busy Beaver—let's say "much" faster, in the sense that they still couldn't be computably upper-bounded even given an oracle for BB or HALT. The answer is easily shown to be yes.

Define the "super Busy Beaver function," $\mathrm{BB}_1(n)$, exactly the same way as $\mathrm{BB}(n)$, except that the Turing machines being maximized over are now equipped with a HALT oracle in some suitable way. (The "original" BB function would then be $\mathrm{BB}_0(n)$.) Since the arguments in Section 2 relativize, we find that $\mathrm{BB}_1(n)$ dominates not only $\mathrm{BB}(n)$ itself, but *any* function computable using a BB oracle.

Continuing, one can define the function $\mathrm{BB}_2(n)$ by maximizing over halting n-state Turing machines with oracles for "SUPERHALT": that is, the halting problem for Turing machines with HALT oracles. Next one can define $\mathrm{BB}_3(n)$, $\mathrm{BB}_4(n)$, and so on, and can proceed in this way through all the computable ordinals[11]: $\mathrm{BB}_\omega(n)$, $\mathrm{BB}_{\omega^\omega}(n)$, $\mathrm{BB}_{\omega^{\omega^{\omega^{\cdot^{\cdot^{\cdot}}}}}}(n)$, and so on. Each such function will grow faster than any function computable using an oracle for *all* the previous functions in this well-ordered list.

One can even define, for example, $\mathrm{BB}_{\omega_{\mathrm{ZF}}}(n)$, where ω_{ZF} is the computable ordinal that's the supremum of all the computable ordinals that can be proven to exist in ZF set theory.[12] Or $\mathrm{BB}_{\omega_{\mathrm{LC}}}(n)$, where LC is some large-cardinal theory extending ZF, and ω_{LC} is the computable ordinal that similarly encodes LC's power.

These latter are the fastest-growing sorts of integer sequences that I know how to define, using any ideas that I'd personally accept as mathematically definite. Perhaps the *ultimate* open problem in BusyBeaverology—albeit, not an especially well-defined problem!—is whether there's any way to go further than this.[13] If there isn't, then it seems that a "who can name the bigger number" contest, carried out between two experts, would quickly degenerate into an argument over which large-cardinal axioms are allowed when defining a generalized BB function.

3.1 Semi-Busy Beavers

As a teenager, soon after I learned about the Busy Beaver function, I wondered the following: is there any function whose growth rate is *intermediate* between the computable functions and the functions like Busy Beaver?

It turns out that, using standard techniques from computability theory, it's not hard to construct such a function:

Theorem 5 *There exists a function $g : \mathbb{N} \to \mathbb{N}$ such that*

[11]A *computable ordinal* is simply the order type of some well-ordering of the positive integers, whose order relation can be decided by a Turing machine. The supremum of the computable ordinals is the so-called *Church-Kleene ordinal*, ω_{CK}, which is not computable.

[12]For a self-contained proof that the ordinal ω_{ZF} is computable, see for example `https://mathoverflow.net/questions/165338/why-isnt-this-a-computable-description-of-the-ordinal-of-zf`

[13]Rayo (see `https://googology.wikia.org/wiki/Rayo%27s_number`) claimed to define even faster-growing sequences, by using second-order logic. However, I'm personally unwilling to regard an integer sequence as "well-defined," if (as in Rayo's case) the values of the integers might depend on the truth or falsehood of the Axiom of Choice, the Continuum Hypothesis, or other statements of transfinite set theory. Assuming the *consistency* of certain transfinite set theories, in order to construct monstrous computable ordinals (and from there, hyper-rapidly growing integer sequences), is as far as I'm willing to go right now.

(i) for all computable functions f, there exists an n_f such that $g(n) > f(n)$ for all $n \geq n_f$, but

(ii) HALT *(or equivalently,* BB*) is still uncomputable given an oracle for g.*

Proof. Let $f_1, f_2, \ldots$ be an enumeration of all computable functions from $\mathbb{N}$ to $\mathbb{N}$. We'll set

$$g(n) := \max_{i \leq w(n)} f_i(n),$$

for some nondecreasing function $w : \mathbb{N} \rightarrow \mathbb{N}$ (to be chosen later) that increases without bound. This is already enough to ensure property (i)—i.e., that g eventually dominates every computable function f_i. So all that remains is to choose w so that g satisfies property (ii).

Let $R_1, R_2, \ldots$ be an enumeration of Turing machines that are candidate reductions from HALT to g. Then we construct w via the following iterative process: set $w(1) := 1$. Then, for each $n = 1, 2, 3, \ldots$, if there exists an input $x \in \{0,1\}^*$ such that the machine $R^g_{w(n)}(x)$ queries g only on values $n' \leq n$ (i.e., the values for which g has already been defined), and then *fails* to decide correctly whether $x \in \text{HALT}$, then set $w(n+1) := w(n) + 1$. Otherwise, set $w(n+1) := w(n)$.

Provided w increases without bound, clearly this construction satisfies property (ii), since it eventually "kills off" every possible reduction from HALT to g. So it remains only to verify that w increases without bound. To see this, note that if $w(n)$ "stalled" forever at some fixed value w^*, then g would be computable, and $R^g_{w^*}$ would decide HALT on all inputs. Therefore HALT would be computable, contradiction. ■

More generally, one can create a dense collection of growth rates that interpolate between computable and Busy Beaver. And with a little more work, one can ensure that these growth rates are all computable given a BB oracle. However, the intermediate growth rates so constructed will all be quite "unnatural"; finding "natural" intermediates between the computable functions and HALT is a longstanding open problem.[14]

4 Concrete Values

Having covered the theory of the Busy Beaver function and various generalizations, I'll now switch gears, and discuss what's known about the values of the actual, concrete BB function, defined using 1-tape, 2-symbol Turing machines as in Section 1.

One warning: the recent bounds that I'll cite typically come not from peer-reviewed papers, but from posts to online forums or code repositories, with varying levels of explanation and documentation. It's possible that bugs remain—particularly given the infeasibility of testing programs that are supposed to run for $> 10^{10^{10^{10}}}$ steps! In this survey, I'll give publicly available constructions the benefit of the doubt, but it would be great to institute better vetting and ideally formal verification.

The following table lists the accepted values and lower bounds for $\text{BB}(1), \ldots, \text{BB}(7)$, as of July 2020. Because it will be relevant later, the values of Radó's ones function Σ are also listed.

[14]The following computational task can be shown to have intermediate difficulty between computable and HALT: given as input a Turing machine M, if M accepts on the all-0 input then accept, if M rejects then reject, and if M runs forever, then either accept *or* reject. However, I don't know how to use this task to define an intermediate growth rate.

n	BB (n)	$\Sigma(n)$	**Reference**
1	1	1	Trivial
2	6	4	Lin 1963 (see [9])
3	21	6	Lin 1963 (see [9])
4	107	13	Brady 1983 [2]
5	$\geq 47,176,870$	$\geq 4,098$	Marxen and Buntrock 1990 [10]
6	$> 7.4 \times 10^{36,534}$	$> 3.5 \times 10^{18,267}$	Kropitz 2010 [8]
7	$> 10^{2\times 10^{10^{10^{18,705,353}}}}$	$> 10^{10^{10^{10^{18,705,353}}}}$	"Wythagoras" 2014 (see [12, Section 4.6])

For completeness, here are the Busy Beavers and (for $n \geq 5$) current champions themselves. The states are labeled A, B, C, ... (A is the initial state), H means Halt, and (e.g.) 1RB in the A0 entry means "if in state A and reading 0, then write 1, move right, and transition to state B."

$n = \mathbf{2}$	**A**	**B**
0	1RB	1LA
1	1LB	H

$n = \mathbf{3}$	**A**	**B**	**C**
0	1RB	1LB	1LC
1	H	0RC	1LA

$n = \mathbf{4}$	**A**	**B**	**C**	**D**
0	1RB	1LA	H	1RD
1	1LB	0LC	1LD	0RA

$n = \mathbf{5}$	**A**	**B**	**C**	**D**	**E**
0	1RB	1RC	1RD	1LA	H
1	1LC	1RB	0LE	1LD	0LA

$n = \mathbf{6}$	**A**	**B**	**C**	**D**	**E**	**F**
0	1RB	1RC	1LD	1RE	1LA	H
1	1LE	1RF	0RB	0LC	0RD	1RC

$n = \mathbf{7}$	**A**	**B**	**C**	**D**	**E**	**F**	**G**
0	1RB	1RC	1LD	1LF	H	1RG	1LB
1	N/A	0LG	1RB	1LE	1LF	0LD	0RF

It's not hard to see that BB $(1) = 1$: on an all-0 input tape, a 1-state Turing machine either halts on the very first step, or else it moves infinitely to either the left or the right. For larger values, the hardest part is of course to show that all machines that run for longer than the claimed bound actually run forever. For this, Lin and Radó [9] and Brady [2] used a combination of automated proof techniques and hand analysis of a few "holdout" cases. Meanwhile, to find the current $n = 5$ champion, Marxen and Buntrock [10] had to do a computer search that was sped up by numerous tricks, such as pruning the search tree of possible machines, and speeding up the simulation of a given machine by grouping together many adjacent tape squares into blocks. Kropitz [8] then built on those techniques to find the current $n = 6$ champion, and "Wythagoras" adapted Kropitz's machine to produce the current $n = 7$ champion.

But what do the record-holding machines actually *do*, semantically? Let's consider just one example: the 5-state machine found by Marxen and Buntrock [10], which established that BB $(5) \geq 47,176,870$. According to Michel [12, Section 5.2.1], the Marxen-Buntrock machine effectively applies a certain iterative map, over and over, to a positive integer encoded on the tape. The map is strongly reminiscent of the infamous *Collatz map*, which (recall) is the function $f : \mathbb{N} \to \mathbb{N}$ defined by

$$f(x) := \begin{cases} x/2 & \text{if } x \text{ is even} \\ 3x+1 & \text{if } x \text{ is odd} \end{cases}$$

The *Collatz Conjecture* says that, for every positive integer x, repeated application of the above map ($f(x)$, $f(f(x))$, etc.) leads eventually to 1. This has been verified for all $x \leq 2^{68}$ but remains open.

For the Marxen-Buntrock machine, the relevant map is instead

$$g(x) := \begin{cases} \frac{5x+18}{3} & \text{if } x \equiv 0 \pmod 3 \\ \frac{5x+22}{3} & \text{if } x \equiv 1 \pmod 3 \\ \perp & \text{if } x \equiv 2 \pmod 3 \end{cases}$$

The question is whether, if we start from $x = 0$ and then repeatedly set $x := g(x)$, we'll ever reach the $\perp$ state. The answer turns out to be yes, as follows:

$$\begin{gathered} 0 \to 6 \to 16 \to 34 \to 64 \to 114 \to 196 \to 334 \to 564 \\ \to 946 \to 1584 \to 2646 \to 4416 \to 7366 \to 12284 \to \perp. \end{gathered}$$

By iterating g and halting when $x = \perp$ is reached, the Marxen-Buntrock machine verifies the above fact about g. The machine runs for tens of millions of steps because of one further detail of its construction: namely, that it spends a number of steps in each iteration (except the final iteration) that scales roughly like $\frac{5}{9}x^2$.

As it turns out, the current 6-state champion, due to Kropitz, *also* applies a Collatz-like map, albeit one that sends an integer x to an exponentially larger integer (see [12, Section 5.3.1]). With the Kropitz machine, there are only five iterations until halting—but because of the repeated exponentiation, that's enough to produce an astronomical x, of order 4^{30340}, and hence an astronomical runtime. Most other strong contenders over the past 30 years have also applied Collatz-like maps to integers or pairs of integers, although not quite all of them. This of course raises the possibility—whether enticing or terrifying!—that progress on determining the small values of $\mathrm{BB}(n)$ might go hand in hand with progress on the Collatz Conjecture and its variants. (Indeed, given this connection, it might come as no surprise that, as shown by John Conway [4], a generalized version of the Collatz Conjecture is Turing-undecidable.)

4.1 Further Out

There are more general computable lower bounds on $\mathrm{BB}(n)$, although they're of interest only for small values of n (as $n \to \infty$ it becomes easy to beat them). For example, in 1964 Green [6] showed that $\mathrm{BB}(2n) \geq \mathrm{A}(n-2)$ for all $n \geq 2$. Here A is the *Ackermann function*, which is *also* famous for its explosive growth even though it pales when compared to Busy Beaver: $\mathrm{A}(n) := \mathrm{A}(n,n)$, where

$$\begin{aligned} \mathrm{A}(0,n) &:= n+1, \\ \mathrm{A}(m+1,0) &:= \mathrm{A}(m,1), \\ \mathrm{A}(m+1,n+1) &:= \mathrm{A}(m, \mathrm{A}(m+1,n)). \end{aligned}$$

Actually Green proved even that $\Sigma(2n) \geq \mathrm{A}(n-2)$, where Σ is Radó's ones function.

Building on Green's work, as well as earlier work by online forum users, in 2016 "Wythagoras" was able to show[15] that $\mathrm{BB}(18)$ already exceeds *Graham's number*[16] G: a contender for the largest

[15] See `https://googology.wikia.org/wiki/User_blog:Wythagoras/The_nineteenth_Busy_Beaver_number_is_greater_than_Graham%27s_Number!`

[16] See `https://en.wikipedia.org/wiki/Graham%27s_number`

number ever to appear in mathematical research, typically expressed via recursive use of Knuth's up-arrow notation.

4.2 Concrete Bounds on Knowability

Recall Proposition 4, one of the most striking facts about the Busy Beaver function: that for any axiomatic theory (for example, ZF set theory), there exists a constant c such that the theory determines at most c values of $\mathrm{BB}(n)$. But what is c? Is it more like 10^7 or like 10? Surprisingly, until very recently, there seem to have been no attempts to address that question.

The question is interesting because it speaks to a very old debate: namely, whether Gödelian independence from strong formal systems is a property only of "absurdly complicated" statements in arithmetic, such as those that talk directly about the formal systems themselves, or whether independence rears its head even for "natural" statements. Of course, expressibility by a small Turing machine is not quite the same as "naturalness," but it has the great advantage of being definite.[17]

Seeking to get the ball rolling with this subject, in 2016, my then-student Adam Yedidia and I [14] proved the following.

Theorem 6 ([14]) *There's an explicit*[18] 7910*-state Turing machine that halts iff the set theory* $\mathrm{ZF}+\mathrm{SRP}$ *(where* SRP *means Stationary Ramsey Property) is inconsistent. Thus, assuming that theory is consistent,* ZF *cannot prove the value of* $\mathrm{BB}(7910)$.

Our proof of Theorem 6 involved three ideas. First, we relied heavily on work by Harvey Friedman, who gave combinatorial statements equivalent to the consistency of $\mathrm{ZF}+\mathrm{SRP}$. Second, since hand-designing a 7910-state Turing machine was out of the question, we built a custom programming language called "Laconic," along with a sequence of compilers (to multitape and then single-tape Turing machines) that aggressively minimized state count at the expense of running time and everything else. This was enough to produce a machine with roughly $400,000$ states. Third, to reduce the state count, we repeatedly used the idea of "introspective encoding": that is, a Turing machine that first writes a program onto its tape, and then uses a built-in interpreter to execute that program. This idea eliminates many redundancies that blow up the number of states.

Soon afterward, Stefan O'Rear improved our result to get a 1919-state machine, and then following continued improvements, a 748-state machine.[19] While it reuses a variant of our Laconic language, as well as the introspective encoding technique, O'Rear's construction just directly searches for an inconsistency in ZF set theory, thereby removing the reliance on Friedman's work as well as on the SRP axiom. Assuming O'Rear's construction is sound, we therefore have:

[17] There are many examples of "simple" problems in arithmetic—Diophantine equations, tiling, matrix mortality, etc.—that encode *universal Turing computation*, and that therefore, *given a suitable input instance*, would encode Gödel undecidability as well. However, it's important to realize that none of these count as "simple" examples of Gödel undecidability in the sense we mean, unless the input that produces the Gödel undecidability is *also* simple. The trouble is that typically, such an input will need to encode a program that enumerates all the theorems of ZF, or something of that kind—a relatively complicated object.

[18] As an early reader pointed out, the word "explicit" is needed here because otherwise such theorems are vacuously true! Consider, for example, a 1-state Turing machine M "defined" as follows: if $\mathrm{ZF}+\mathrm{SRP}$ is inconsistent, then choose M to halt in the first step; otherwise choose M to run forever.

[19] See `https://github.com/sorear/metamath-turing-machines/blob/master/zf2.nql`

Theorem 7 (O'Rear) *There's an explicit* 748*-state Turing machine that halts iff* ZF *is inconsistent. Thus, assuming* ZF *is consistent,* ZF *cannot prove the value of* $\mathrm{BB}(748)$.

Meanwhile, a GitHub user named "Code Golf Addict" apparently showed:[20]

Theorem 8 *There's an explicit* 27*-state Turing machine that halts iff Goldbach's Conjecture is false.*

And Matiyasevich, O'Rear, and I showed:[21]

Theorem 9 *There's an explicit* 744*-state Turing machine that halts iff the Riemann Hypothesis is false.*

5 The Frontier

My main purpose in writing this survey is to make people aware that there are enticing *open problems* about the Busy Beaver function—many of them conceptual in nature, many of them potentially solvable with modest effort.

5.1 Better Beaver Bounds

Perhaps the most obvious problem is to pin down the value of $\mathrm{BB}(5)$. Let me stick my neck out:

Conjecture 10 $\mathrm{BB}(5) = 47,176,870$.

Recall that in 1990, Marxen and Buntrock [10] found a 5-state machine that halts after $47,176,870$ steps. The problem of whether any 5-state machine halts after even more steps has now stood for thirty years. In my view, the resolution of this problem would be a minor milestone in humanity's understanding of computation.

Apparently, work by various people has reduced the $\mathrm{BB}(5)$ problem to only 25 machines whose halting status is not yet known.[22] It would be good to know whether the non-halting of some or all of these machines can be reduced to Collatz-like statements, as with the *halting* of the Marxen-Buntrock machine (see Section 4).

It would also be great to prove better lower bounds on $\mathrm{BB}(n)$ for $n > 5$. Could $\mathrm{BB}(7)$ or $\mathrm{BB}(8)$ already exceed Graham's number G? As I was writing this survey, my 7-year-old daughter Lily (mentioned in Section 1) raised the following question: what's the first n such that $\mathrm{BB}(n) > \mathrm{A}(n)$, where $\mathrm{A}(n)$ is the Ackermann function defined in Section 4? Right now, I know only that $5 \le n \le 18$, where $n \le 18$ comes (for example) from the result of "Wythagoras," mentioned in Section 4.1, that $\mathrm{BB}(18)$ exceeds Graham's number.

I find it difficult to guess whether the values of $\mathrm{BB}(6)$ and $\mathrm{BB}(7)$ will *ever* be known.

[20]See `https://gist.github.com/anonymous/a64213f391339236c2fe31f8749a0df6` or `https://gist.github.com/jms137/cbb66fb58dde067b0bece12873fadc76` for an earlier 47-state version with better documentation.

[21]See `https://github.com/sorear/metamath-turing-machines/blob/master/riemann-matiyasevich-aaronson.nql`

[22]See `https://skelet.ludost.net/bb/nreg.html` for a list of 43 machines whose halting status was open around 2010. Daniel Briggs reports that 18 of these machines have since been proved to run forever, leaving only 25; see `https://github.com/danbriggs/Turing` for more information.

5.2 The Threshold of Unknowability

What's the smallest k such that the value of $\mathrm{BB}(k)$ is provably independent of ZF set theory? As we saw in Section 4.2, there's now a claimed construction showing that ZF doesn't prove the value of $\mathrm{BB}(748)$. My own guess is that the actual precipice of unknowability is much closer:

Conjecture 11 ZF *does not prove the value of* $\mathrm{BB}(20)$.

While I'd be thrilled to be disproved,[23] I venture this conjecture for two reasons. First, the effort to optimize the sizes of undecidable machines only started in 2015—but since then, sporadic work by two or three people has steadily reduced the number of states from more than a million to fewer than a thousand. Why should we imagine that we're anywhere near the ground truth yet? Second, as we saw in Section 4, the known lower bounds show that 5-state and 6-state machines can already engage number theory problems closely related to the Collatz Conjecture. How far off could Gödel-undecidability possibly be?

We could also ask when the BB function first eludes Peano arithmetic. Given that multiple "elementary" arithmetical statements are known to be independent of PA—for example, Goodstein's Theorem [5] and the Kirby-Paris hydra theorem [7]—here I'll conjecture an even earlier precipice than for ZF:

Conjecture 12 *Peano Arithmetic does not prove the value of* $\mathrm{BB}(10)$.

Short of resolving Conjecture 12, or anything close to it, an excellent project would be to derive *any* upper bound on the number of BB values provable in PA, better than what's known for ZF.

5.3 Uniformity of Growth

Consider the following conjecture:

Conjecture 13 *There exists an* n_0 *such that for all* $n \geq n_0$,

$$\mathrm{BB}(n+1) > 2^{\mathrm{BB}(n)}.$$

Given what we know about the BB function's hyper-rapid growth, Conjecture 13 (perhaps even with $n_0 = 6$) seems *obviously* true. Indeed, it still seems obviously true, even if we replaced the exponentiation by any other computable function. But try proving it!

From the results in Section 2, all that follows is that $\mathrm{BB}(n+1) > 2^{\mathrm{BB}(n)}$ for *infinitely many* n, not for almost all. Embarrassingly, as far as I know, the best separation between $\mathrm{BB}(n+1)$ and $\mathrm{BB}(n)$ that's been *proven* is still the trivial Proposition 1, that $\mathrm{BB}(n+1) > \mathrm{BB}(n)$.

The trouble is that, given a Turing machine M, if we try to modify M to make it run for (say) $s(M)^2$ or $2^{s(M)}$ steps rather than $s(M)$, we quickly find ourselves adding *many* more states. This is closely related to the fact that Turing machines have no built-in notion of subroutines, modularity, or local and global workspaces.

For Turing machines, the best known result in the direction of Conjecture 13 is due to Ben-Amram and Petersen [1] from 2002.

[23] Even if Conjecture 11 is false, showing that ZF *does* settle the value of $\mathrm{BB}(20)$ (presumably, by settling that value ourselves) strikes me as an astronomically harder undertaking than settling P vs. NP.

Theorem 14 ([1]) *Let f be any computable function. Then there exists a constant c_f such that, for all n,*

$$\mathrm{BB}\left(n + 8\left\lceil n/\log_2 n\right\rceil + c_f\right) > f\left(\mathrm{BB}\left(n\right)\right).$$

The main idea in the proof of Theorem 14 is what Adam Yedidia and I [14] later termed *introspective encoding.* Given an n-state Turing machine M, suppose we could design another Turing machine M', with only slightly more states than M (say, $n+c$ states), which wrote a coded description of M onto its tape. Then by adding $O(1)$ additional states, we could do whatever we liked with the coded description—including simulating M, or (say) counting the number of steps $s(M)$ until M halts, and then looping for $2^{s(M)}$ or $2^{2^{s(M)}}$ steps. Thus, for any computable function f, such an encoding would imply the bound

$$\mathrm{BB}\left(n + c + O_f(1)\right) > f\left(\mathrm{BB}\left(n\right)\right).$$

Unfortunately, given an n-state Turing machine M, the obvious machine that writes a description of M (up to isomorphism) to the tape has $n\log_2 n + O(n) \gg n$ states: one for each bit of M's description. However, by using a more careful encoding, it turns out that we can get this down to $n + O(n/\log n)$ states: off by only an additive error term from the information-theoretic lower bound of n states. More generally:

Lemma 15 (Introspective Encoding Lemma, implicit in [1]) *Let $x \in \{0,1\}^{\lceil n\log_2 n\rceil}$. Then there exists a Turing machine M_x, with $n + O\left(\frac{n}{\log n}\right)$ states, that outputs x (and only x) when run on an all-0 tape.*

Note that, by a counting argument, such an M_x must have at least $n - O\left(\frac{n}{\log n}\right)$ states in general.

Thus, one natural way to make progress toward Conjecture 13 would be to improve the error term in Lemma 15: say, to $O(\sqrt{n})$ or even $O(\log n)$. On the other hand, to establish large, uniform gaps between (say) $\mathrm{BB}(n)$ and $\mathrm{BB}(n+1)$, one might need to move beyond the introspection technique.

Let me remark that the situation is better for most programming languages; Turing machines are almost uniquely bad for this problem. More concretely, let L be a programming language whose programs consist of bit-strings. Recall that $\mathrm{BB}_L(n)$ is the largest finite number of steps taken on a blank input by any L-program at most n bits long. Then to study the fine-grained growth of $\mathrm{BB}_L(n)$, the relevant question is this: given an n-bit L-program Q, how long must an L-program Q' be that stores Q as a string and then interprets the string?

If Q' can be only $\log_2 n + O(1)$ bits longer than Q itself—say, because Q' just needs to contain n and Q, in addition to a constant-sized interpreter—then we get that for all computable functions f, there exists a constant c_f such that for all n,

$$\mathrm{BB}_L\left(n + \log_2 n + c_f\right) > f\left(\mathrm{BB}_L\left(n\right)\right).$$

In the special case of Lisp, matters are better still, because of Lisp's "quote" mechanism and its built-in interpreter. There we get that for all computable functions f, there exists a constant c_f such that for all n,

$$\mathrm{BB}_{\mathrm{Lisp}}\left(n + c_f\right) > f\left(\mathrm{BB}_{\mathrm{Lisp}}\left(n\right)\right).$$

Moreover, for simple functions f the constant c_f should be quite small.

5.4 Shifts Versus Ones

Using Lemma 15 (i.e., introspective encoding), Ben-Amram and Petersen [1] established other interesting inequalities: for example, that there exists a constant c such that

$$\mathrm{BB}(n) < \Sigma\left(n + 8\lceil n/\log_2 n\rceil + c\right)$$

for all n, where $\Sigma(n)$ is Radó's ones function. This is the best current result upper-bounding BB in terms of Σ. If, however, we could move beyond introspection, then perhaps we could get a much tighter relationship, like the following:

Conjecture 16 $\mathrm{BB}(n) < \Sigma(n+1)$ *for all* $n \geq 4$.

Currently, the inequality $\mathrm{BB}(n) < \Sigma(n+1)$ is known to fail only at $\mathrm{BB}(3) = 21$ and $\Sigma(4) = 13$.

Based on the limited data we have, I can't resist venturing an outrageously strong conjecture: namely, that Radó's shift function and ones function are quadratically related, with $\mathrm{BB}(n) \approx \Sigma(n)^2$ for all n. Or more precisely:

Conjecture 17
$$\lim_{n\to\infty} \frac{\log \mathrm{BB}(n)}{\log \Sigma(n)} = 2.$$

A heuristic argument for Conjecture 17 is that the known Busy Beavers, and current champions, all seem to move back and forth across the tape order $\Sigma(n)$ times, visiting a few more squares each time, until order $\Sigma(n)$ squares have been visited. But of course, another possibility is that the limit in Conjecture 17 doesn't even exist.

5.5 Evolving Beavers

Suppose you already knew $\mathrm{BB}(n)$. Could a trusted wizard[24] send you a short message that would let you calculate $\mathrm{BB}(n+1)$ as well? Equivalently, in the hunt for an $(n+1)$-state Busy Beaver, how useful of a clue is an n-state Busy Beaver?

Chaitin [3] raised the above question, motivated by a hypothetical model of "Turing machine biology," in which larger Busy Beavers need to evolve from smaller ones. Perhaps surprisingly, he observed that $\mathrm{BB}(n)$ *can* provide a powerful clue about $\mathrm{BB}(n+1)$.

To state Chaitin's result formally, recall that a language $L \subset \{0,1\}^*$ is called *prefix-free* if no string in L is a proper prefix of any other. Also, given a programming language L, recall that BB_L is the variant of the Busy Beaver function for L. Finally, given strings x and y, recall that the *conditional Kolmogorov complexity* $\mathrm{K}(y|x)$ is the bit-length of the shortest program P, in some universal programming language, such that $P(x) = y$.

I'll prove the following result for completeness, since the proof might be hard to extract from [3]. I thank my former student, Luke Schaeffer, for explaining the proof to me.

Theorem 18 (implicit in Chaitin [3]) *Let L be any standard prefix-free universal programming language, such as Lisp. Then* $\mathrm{K}(\mathrm{BB}_L(n+1) \mid \mathrm{BB}_L(n)) = O(\log n)$.

[24] The wizard *has* to be trusted, since otherwise we could compute $\mathrm{BB}(n+1)$ (and then $\mathrm{BB}(n+2)$, etc.) ourselves, by iterating over all possible messages until we found one that worked. So for complexity theorists: we're asking here for Karp-Lipton advice, not for a witness from Merlin.

Proof. We can assume without loss of generality that n itself is provided to the program for $\mathrm{BB}_L(n+1)$ along with $\mathrm{BB}_L(n)$, since that adds only an $O(\log n)$ overhead.

The key idea is to relate BB_L to Chaitin's famous halting probability $\Omega = \Omega_L$, which is defined as

$$\Omega := \sum_{P \in L \,:\, P(\varepsilon) \text{ halts}} 2^{-|P|},$$

where $|P|$ is the bit-length of P and ε is the empty input. Since L is prefix-free, $\Omega \in (0,1)$.

Let $\Omega_n < \Omega$ be the approximation to Ω obtained by truncating its binary expansion to the first n bits.

Then our first claim is that, if we know $\mathrm{BB}_L(n)$, then we can compute Ω_{n-c}, for some $c = O(\log n)$. To see this, recall that knowledge of $\mathrm{BB}_L(n)$ lets us solve the halting problem for all L-programs of length at most n. Now let A_β be a program that hardwires a constant $\beta \in (0,1)$, with $n-c$ bits of precision, and that dovetails over all L-programs, maintaining a running bound q. Initially $q = 0$. Whenever a program P is found to halt, A_β sets $q := q + 2^{-|P|}$. If q ever exceeds β then A_β halts. Clearly, then, A_β eventually halts if $\Omega > \beta$, and it runs forever otherwise. Furthermore, such an A_β can be specified with at most n bits: $n-c$ bits for β, and c bits for everything else in the program, including the $O(\log n)$ overhead from the prefix-free encoding. But this means that, by repeatedly varying β and then using $\mathrm{BB}_L(n)$ to decide whether A_β halts, we can determine the first $n-c$ bits of Ω.

Our second claim is that, if we know Ω_n, then we can compute $\mathrm{BB}_L(n)$. To do so, we simply dovetail over all L-programs, again maintaining a running lower bound q on Ω. Initially $q = 0$. Whenever a program P is found to halt, we set $q := q + 2^{-|P|}$. We continue until $q \geq \Omega_n$—which must happen eventually, since Ω_n is a strict lower bound on Ω. By this point, we claim that every halting program P of length at most n must have halted. For suppose not. Then we'd have

$$q + 2^{-|P|} \geq \Omega_n + 2^{-n} > \Omega,$$

an absurdity. But this means that we can now compute $\mathrm{BB}_L(n)$, by simply maximizing the running time $s(P)$ over all the programs $P \in L \cap \{0,1\}^{\leq n}$ that have halted.

Putting the claims together, if we know $\mathrm{BB}_L(n)$ then we can compute Ω_{n-c}, for some $c = O(\log n)$. But if we know Ω_{n-c}, then we need to be told only $c+1$ additional bits in order to know Ω_{n+1}. Finally, if we know Ω_{n+1} then we can compute $\mathrm{BB}_L(n+1)$. ■

Can one prove a version of Theorem 18 for the "classic" Busy Beaver function $\mathrm{BB}(n)$, defined using Turing machines rather than prefix-free programming languages? I can indeed do so, by combining the ideas of Theorem 18 with the introspective encoding from Section 5.3. As far as I know, this little result is original to this survey. The bound I'll get—that $\mathrm{BB}(n+1)$ is computable given $\mathrm{BB}(n)$ together with $O(n)$ advice bits—is quantitatively terrible compared to the $O(\log n)$ advice bits from Theorem 18, but it does beat the trivial $O(n \log n)$ bits that would be needed to describe an $(n+1)$-state Busy Beaver "from scratch."

Theorem 19 *$\mathrm{K}(\mathrm{BB}(n+1) \mid \mathrm{BB}(n)) = O(n)$.*

Proof. Let enc be a function that takes as input a description $\langle M \rangle$ of a Turing machine M, and that outputs a binary encoding, in some prefix-free language, of $\langle M \rangle$'s equivalence class under permuting the states. It's possible to ensure that, if M has n states, then $\mathrm{enc}(\langle M \rangle) \in \{0,1\}^{n \log_2 n + O(n)}$. Using enc, we can define Chaitin's halting probability for Turing machines as follows:

$$\Omega := \sum_{M(\varepsilon) \text{ halts}} 2^{-|\mathrm{enc}(\langle M \rangle)|}.$$

We now simply follow the proof of Theorem 18. Let $\Omega_m < \Omega$ be the approximation to Ω obtained by truncating its binary expansion to the first m bits.

Then our first claim is that, if we know $\mathrm{BB}(n)$, then we can compute $\Omega_{n \log_2 n - c}$, for some $c = O(n)$. This follows from the same analysis as in Theorem 18, combined with Lemma 15 (the Introspective Encoding Lemma). The latter is what produces the loss of $O(n)$ bits.

Our second claim is that, if we know $\Omega_{n \log_2 n + c'}$, for some $c' = O(n)$, then we can compute $\mathrm{BB}(n+1)$. This follows from the same analysis as in Theorem 18, combined with the fact that $\mathrm{enc}(\langle M \rangle)$ maps each n-state machine to a string of length $n \log_2 n + O(n)$.

Putting the claims together, if we know $\mathrm{BB}(n)$ then we can compute $\Omega_{n \log_2 n - c}$. So we need to be told only $c + c' = O(n)$ additional bits in order to know $\Omega_{n \log_2 n + c'}$, from which we can compute $\mathrm{BB}(n+1)$. ■

I can now pose the open problem of this section. Can one improve Theorem 19, to show (for example) that

$$\mathrm{K}(\mathrm{BB}(n+1) \mid \mathrm{BB}(n)) = O(\log n)?$$

For that matter, could Theorems 18 *or* 19 be improved to get the number of advice bits below $O(\log n)$—possibly even down to a constant? How interrelated *are* the successive values of $\mathrm{BB}(n)$?

5.6 Behavior on Nonzero Inputs

Michel (see [11]) proved a striking property of the 2-, 3-, and 4-state Busy Beavers. Namely, all of these machines turn out to halt on *every* finite input—that is, every initial tape with only finitely many 1's—rather than only the all-0 input.[25] Michel also showed that the 5-state champion discovered by Marxen and Buntrock [10] (see Section 4) halts on every finite input, *if and only if* the following conjecture holds:

Conjecture 20 *The map $g : \mathbb{N} \to \mathbb{N} \cup \{\perp\}$, from Section 4, leads to $\perp$ when iterated starting from any natural number x.*

Conjecture 20 looks exceedingly plausible, from heuristics as well as numerical evidence—but proving it is yet another unsolved "Collatz-like" problem in number theory. The situation for the current 6- and 7-state champions is similar (see [12, Section 5.7]): they plausibly halt on all inputs, but only if some Collatz-like conjecture is true.

These observations inspire broader questions. For example, do *all* Busy Beavers halt on all finite inputs? If so, what could possibly be the explanation? If not, then does any Busy Beaver have some input that causes it to reach an infinite loop on a fixed set of tape squares, rather than using more and more squares? Does any Busy Beaver act as a universal Turing machine?

[25]By contrast, if we allow infinite inputs, then even a 1-state Busy Beaver can easily be made to run forever, by a starting it on an all-1's tape.

5.7 Busy Beaver and Number Theory

Perhaps my *favorite* open questions about the Busy Beaver function were posed by my former student Andy Drucker. He asked:

Is BB(n) *infinitely often even? Is it infinitely often odd? Is the set* $\{n : \text{BB}(n) \text{ is odd}\}$ *computable?*

Currently, we know only that BB$(2) = 6$ is even, while BB$(1) = 1$, BB$(3) = 21$, and BB$(4) = 107$ are odd.

We could likewise ask: is BB(n) infinitely often prime? Is it infinitely often composite? (Right now one prime value is known: BB$(4) = 107$.) Is BB(n) ever a perfect square or a power of 2? Etc.

Of course, just like many of the questions discussed in previous sections, the answers to these questions could be highly sensitive to the model of computation. Indeed, it's easy to define a Turing-complete model of computation wherein every valid program is constrained to run for an even number of steps (or a square number of steps, etc), so that some of these number-theoretic questions would be answered by fiat!

But what are the answers in "natural" models of computation, like Turing machines (as for the usual BB function), RAM machines, or Lisp programs?

Admittedly, these are not typical research questions for computability theory, since they're so model-dependent. But that's part of why I've grown to like the questions so much. Even to make a start on them, it seems, one would need to say something new and general about computability, *beyond* what's common to all Turing-universal models—something able to address "computational epiphenomena," like whether a machine will run for an odd or even number of steps, after we've optimized it for a property completely orthogonal to that question.

Nearly sixty years after Radó defined it, Busy Beaver—a finite-valued function with infinite aspirations—continues to beckon and frustrate those for whom the space of all possible programs is a universe to be explored. Granted that we'll never swim in it, can't we wade just *slightly* deeper into Turing's ocean of unknowability?

6 Acknowledgments

I thank Daniel Briggs, Andy Drucker, Harvey Friedman, Bill Gasarch, Georgi Georgiev, Marijn Heule, Adam Klivans, Heiner Marxen, Luke Schaeffer, and Ronald de Wolf for their feedback. I especially thank Bill Gasarch for proposing this article, Marijn Heule for Figure 1 and for calling my attention to the question about whether Busy Beavers halt on all inputs, Lily Aaronson for raising the question about BB(n) versus A(n), Andy Drucker for raising the questions in Section 5.7, and Luke Schaeffer for helping me understand the ideas in Sections 5.3 and 5.5.

References

[1] A. M. Ben-Amram and H. Petersen. Improved bounds for functions related to Busy Beavers. *Theory Comput. Syst.*, 35(1):1–11, 2002.

[2] A. H. Brady. The determination of the value of Rado's noncomputable function $\Sigma(k)$ for four-state Turing machines. *Mathematics of Computation*, 40(162):647–665, 1983.

[3] G. Chaitin. To a mathematical theory of evolution and biological creativity. Technical Report 391, Centre for Discrete Mathematics and Theoretical Computer Science, 2010. `www.cs.auckland.ac.nz/research/groups/CDMTCS/researchreports/391greg.pdf`.

[4] J. H. Conway. Unpredictable iterations. In *Proc. 1972 Number Theory Conference, University of Colorado, Boulder*, pages 49–52, 1972.

[5] R. Goodstein. On the restricted ordinal theorem. *J. Symbolic Logic*, 9:33–41, 1944.

[6] M. W. Green. A lower bound on Rado's Sigma function for binary Turing machines. In *Proc. IEEE FOCS*, pages 91–94, 1964.

[7] L. Kirby and J. Paris. Accessible independence results for Peano arithmetic. *Bulletin of the London Mathematical Society*, 14:285–293, 1982.

[8] P. Kropitz. *Busy Beaver Problem.* Bachelors thesis, Charles University in Prague, 2011. In Czech. is.cuni.cz/webapps/zzp/detail/49210.

[9] S. Lin and T. Radó. Computer studies of Turing machine problems. *J. of the ACM*, 12(2):196–212, 1965.

[10] H. Marxen and J. Buntrock. Attacking the Busy Beaver 5. *Bulletin of the EATCS*, 40:247–251, 1990.

[11] P. Michel. Busy beaver competition and Collatz-like problems. *Archive for Mathematical Logic*, 32:351–367, 1993.

[12] P. Michel. The Busy Beaver competition: a historical survey. arXiv:0906.3749, 2019.

[13] T. Radó. On non-computable functions. *Bell System Technical Journal*, 41(3):877–884, 1962.

[14] A. Yedidia and S. Aaronson. A relatively small Turing machine whose behavior is independent of set theory. *Complex Systems*, (25):4, 2016.