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 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, given n, 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] At each time step, the machine follows a transition arrow from its current state depending on the current symbol being read on the tape, which tells it (1) which symbol to write, (2) whether to move one square left on the tape or one square right, and (3) which state to enter next (see Figure 1).

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 $s(M) := \infty$.

*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.

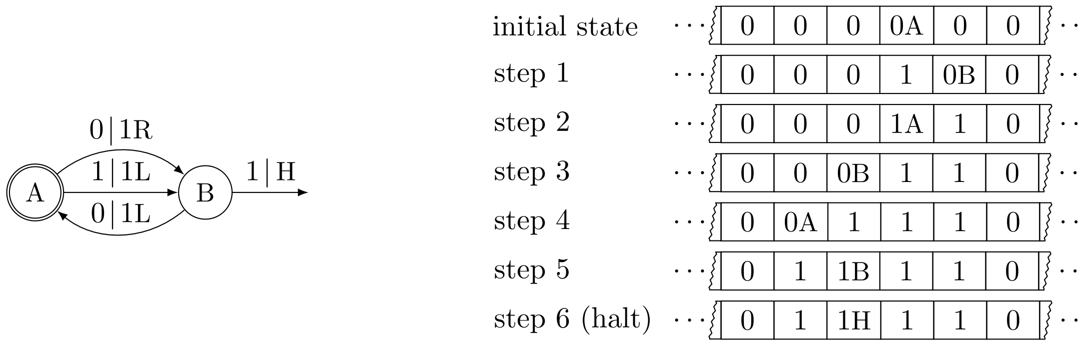

Figure 1: A 2-state Busy Beaver and its execution on an initially all-0 tape. Starting in state A, the machine finds a 0 on its tape and therefore follows the arrow labeled $0|1R$, which causes it to replace the 0 by a 1, move one square to the right, and transition into state B, and so on until the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

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.

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.

Observe that $\mathrm{BB}(n)$ is a perfectly well-defined positive integer for every n—at least, assuming one accepts that there's a definite fact about whether a given Turing machine halts or runs forever on the all-0 input. This follows from the principle that any finite set of integers (for example, the halting times of the n-state Turing machines) has a maximum.

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]). For the few other known values of the BB function see Section 4.

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.

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

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, $\mathrm{BB}(n) = \mathrm{BB}(n,2)$.) 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.[5] 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 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

[5] And, let's say, where the max is 0 if there are no valid L-programs of length at most n.

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 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].[6] While Michel's focus is different from mine—he provides vast reams of data on specific Turing machines, while I'm more concerned with general ideas and open problems—I found his survey an invaluable resource. Meanwhile, for the past decade, the so-called Googology Wiki[7] 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[8] and the Busy Beaver Wikipedia entry[9] 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 1 *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. ■

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.

[6] See also the associated website, `http://www.logique.jussieu.fr/~michel/bbc.html`
[7] `https://googology.wikia.org/wiki/Googology_Wiki`
[8] `http://turbotm.de/~heiner/BB/`
[9] `https://en.wikipedia.org/wiki/Busy_beaver`

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. Furthermore, this is essentially optimal: if we had an advice string for the n-state halting problem that was much shorter, then by hardwiring that advice into a Turing machine, we could create an n-state machine that determined $\mathrm{BB}(n)$ and then ran for longer than $\mathrm{BB}(n)$ steps, an obvious contradiction.[10] Unfortunately, to use the 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!

Although Proposition 1 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 1, fixes this defect.

Proposition 2 *Let* $f \colon \mathbb{N} \to \mathbb{N}$ *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

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

for all n, from which the proposition follows. ∎

Note, in passing, that Proposition 2 is a "different" way to prove the existence of uncomputable functions, one that never explicitly appeals to diagonalization.[11]

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 3 *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. ∎

The proof that we gave for Proposition 3 has the great advantage that it yields an *explicit upper bound*, namely $n_T - 1$, on the number of BB values that the theory T can ever determine. This fact is exploited by recent results (see Section 4.2) that show, for example, that when T is ZF set theory, we can take $n_T = 748$.

[10] This observation can also be related to the "algorithmic incompressibility" of Chaitin's halting probability Ω.

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

However, there's also a different proof of Proposition 3, which has the advantage of never relying on Gödel's Theorem—in fact, it yields another *proof* of a version of Gödel's Theorem. Here is the other proof: suppose by contradiction that the theory T determined all the values of $\mathrm{BB}(n)$. Then by enumerating all possible proofs in T, we could eventually learn $\mathrm{BB}(n)$ for any given n. But this would mean that BB was computable, contradicting Propositions 1 and 2.

More than anything else, for me Proposition 3 captures the romance of the Busy Beaver function. Even though its values are 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.

Even if we supposed that, for every n, there was *some* reasonable extension of ZF set theory that was powerful enough to settle the value of $\mathrm{BB}(n)$—even so, there could be no systematic way to *find* those extensions. For if there were, then $\mathrm{BB}(n)$ would become computable.

I thank Oscar Cunningham for allowing me to share the following commentary on the above.[12]

> I realised an interesting thing while reading this. If we define the theory T_n as
>
> $$\mathrm{PA} + \text{“the } n^{th} \text{ Busy Beaver machine is } b\text{”}$$
>
> where b actually is the n^{th} Busy Beaver machine, then T is a sequence of effectively axiomatized consistent theories with arbitrarily high consistency strength! For any other effectively axiomatized consistent theory S there's some n_S such that $\mathrm{PA} + \mathrm{Con}(T_{n_S})$ proves $\mathrm{Con}(S)$.
>
> So the Busy Beaver numbers give us a countable ladder on which we can place the various Large Cardinal axioms for comparison to each other. Previously I'd been assuming that the structure of all possible Large Cardinal axioms was much more complicated than that, and that the order of their consistency strengths would be transfinite, with no hope of having countable cofinality.

Proposition 3 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 BB(27) steps if it halts at all. But this means that knowing the value of BB(27) would *settle Goldbach's Conjecture*—at least in the abstract sense of reducing that problem to a finite, 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.

I can't resist a further comment. For me, perhaps the central philosophical reason to care about the BB function is that it starkly renders the case for being a "Platonist" about arithmetical questions, like Goldbach's Conjecture or the Riemann Hypothesis. My own case would run as follows: do we agree that it's an iron fact that $\mathrm{BB}(2) = 6$ and $\mathrm{BB}(3) = 21$—a fact independent of all axioms, interpretations, and models? Simply because BB(2) and BB(3) have been calculated (see Section 4), as surely as $5 + 5$ has been? But if so, then why shouldn't there likewise be a fact

[12] See `https://www.scottaaronson.com/blog/?p=4916#comment-1849770`

about BB(1000)? At exactly which n does the value of BB(n) become vague or indeterminate? As we'll see in Section 4.2, if there *is* a model-independent fact about BB(1000), then there are also model-independent facts about the consistency of ZF set theory, the Goldbach Conjecture, the Riemann Hypothesis, and much else, just as an "arithmetical Platonist" would claim.

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 an oracle for the *original* BB function—or, nearly equivalently (by Proposition 1), with an oracle for HALT. (The precise definition of BB_1 will depend on various details of the oracle access mechanism, but those are unimportant for what follows.) Then 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 the number of steps over halting n-state Turing machines with oracles for BB_1. (Observe that, in our new notation, the original BB function becomes BB_0.) Next one can define BB_3 in terms of Turing machines with BB_2 oracles, then $\mathrm{BB}_4(n)$, and so on, and can proceed in this way through not only all the natural numbers, but all the computable ordinals[13]: $\mathrm{BB}_\omega(n)$, $\mathrm{BB}_{\omega^\omega}(n)$, $\mathrm{BB}_{\omega^{\omega^{\cdot^{\cdot^{\cdot}}}}}(n)$, and so on. (Technically, if α is a computable ordinal, then the definition of BB_α will depend not only on α itself, but on the *notation* by which a Turing machine specifies ordinals up to α.) 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.[14] Or—and I thank Niels Lohmann for this observation[15]—one could even diagonalize across *all* computable ordinals. So for example, let $\omega(n)$ be the supremum of all the computable ordinals whose order relations are computable by n-state Turing machines. Then set $F(n) := \mathrm{BB}_{\omega(n)}(n)$. One could of course go even further than this, but perhaps that's enough for now.

Perhaps the ultimate open problem in BusyBeaverology—albeit, not an especially well-defined problem!—is *just how much further* we can go, while restricting ourselves to notations that are "sufficiently clear and definite."[16] Regardless, it seems likely that a "who can name the bigger number" contest, carried out between two experts, would quickly degenerate into an argument over which notations, for ordinals and so forth, are allowed when defining a generalized BB function.

I can't resist including a beautiful fact about the generalized BB functions, which was brought to

[13] 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.

[14] 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`

[15] See comments 62 and 96 at `https://www.scottaaronson.com/blog/?p=4916`

[16] 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.

my attention by Bjørn Kjos-Hanssen.[17] Computability theorists call a language L *hyperarithmetic* if it's computable given an oracle for BB_α, for *some* computable ordinal α. Then:

Proposition 4 *L is hyperarithmetic, if and only if it's computable given an oracle for any sufficiently rapidly-growing function.*

Proof. For the forward direction, if L is computable given BB_α, then L is also computable given any upper bound f on BB_α. For we can compute BB_α given such an f, by just simulating all n-state oracle Turing machines (and their oracles, and their oracles' oracles, and so on) for the appropriate number of steps.

For the reverse direction, Solovay [14] showed that L is hyperarithmetic if and only if for all infinite sets $S \subseteq \mathbb{N}$, there exists a subset $A \subseteq S$ such that L is computable given an oracle for A. Fix an $f : \mathbb{N} \to \mathbb{N}$ such that L is computable given an oracle for any upper bound on f. Then for all infinite $S \subseteq \mathbb{N}$, the subset $A \subseteq S$ whose n^{th} element is an S-element exceeding $f(n)$ satisfies Solovay's condition, so L is hyperarithmetic. ■

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 $g\colon \mathbb{N} \to \mathbb{N}$ such that*

(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

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

for some nondecreasing function $w\colon \mathbb{N} \to \mathbb{N}$ (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 $w(1) \coloneqq 1$. 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 $x \in \mathrm{HALT}$, then set $w(n+1) \coloneqq w(n) + 1$. Otherwise, set $w(n+1) \coloneqq w(n)$.

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^*,

[17] See `https://www.scottaaronson.com/blog/?p=4916#comment-1850680`

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-like." 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.[18]

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 $\mathrm{BB}(1), \ldots, \mathrm{BB}(7)$, as of July 2020. Because it will be relevant later, the values of Radó's ones function Σ are also listed.

n	$\mathrm{BB}(n)$	$\Sigma(n)$	**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	$> 10^{2 \times 10^{10^{10^{18{,}705{,}352}}}}$	$> 10^{10^{10^{10^{18{,}705{,}352}}}}$	"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."

[18] 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.

$n = \mathbf{2}$	**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

(The $n = 7$ machine has an "N/A" entry because it never returns to state A.)

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, via the following orbit (which happens to be extremely long, compared to a random orbit of g):

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

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 except for the very last iteration, it spends a number of steps mapping x to $g(x)$ 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 $4^{30,340}$, 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[19] that $\mathrm{BB}(18)$ already exceeds *Graham's number*[20] G: a contender for the largest 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 3, 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

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

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

Turing machine is not quite the same as "naturalness," but it has the great advantage of being definite.[21]

Seeking to get the ball rolling with this subject, in 2016, my then-student Adam Yedidia and I [15] proved the following.

Theorem 6 ([15]) *There's an explicit*[22] 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.[23] 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:

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:[24]

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:[25]

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

[21] 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.

[22] 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.

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

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

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

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 BB(5). Let me stick my neck out:

Conjecture 10 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 several people, including Georgi Georgiev and Daniel Briggs, has reduced the BB(5) problem to only 25 machines whose halting status is not yet known.[26] These machines are all conjectured to run forever. At my request, Daniel Briggs simulated these 25 "holdout" machines for 100 billion steps each and confirmed that none of them halt by then,[27] which implies in particular that *either* BB(5) = 47,176,870 (i.e., Conjecture 10 holds) or else $\mathrm{BB}(5) > 10^{11}$. It would be good to know whether the non-halting of some or all of these holdouts 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 BB(7) or 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 $5 \leq n \leq 18$, where $n \leq 18$ comes (for example) from the result of "Wythagoras," mentioned in Section 4.1, that BB(18) exceeds Graham's number.

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

5.2 The Threshold of Unknowability

What's the smallest n such that the value of $\mathrm{BB}(n)$ 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 BB(748). My own guess is that the actual precipice of unknowability is much closer:

Conjecture 11 ZF *does not prove the value of* BB(20).

While I'd be thrilled to be disproved,[28] 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

[26] 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.

[27] See `https://github.com/danbriggs/Turing/blob/master/doc/analysis7-24-20.txt`

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

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 PA *does not prove the value of* BB(10).

Short of Conjecture 12, 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.

Finally, we might wonder: are Con(ZF) or Con(PA) *equivalent* to some statement of the form $\mathrm{BB}(n) = k$? I conjecture that the answer is no, so that (for example) there's a significant gap between the smallest n_1 such that $\mathrm{ZF} + \mathrm{Con}(\mathrm{ZF})$ fails to prove the value of $\mathrm{BB}(n_1)$, and the smallest n_2 such that from the value of $\mathrm{BB}(n_2)$ one can prove Con(ZF).

5.3 Uniformity of Growth

Consider the following conjecture, whose non-obviousness was first brought to my attention by a blog comment of Joshua Zelinsky:[29]

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 seems true with a *massive* margin to spare—perhaps even with $n_0 = 6$. Indeed, the conjecture *still* seems self-evident, even if we replace the exponentiation by Ackermann or 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 n.

It's not hard to see that $\mathrm{BB}(n+1) > \mathrm{BB}(n)$ for all n, since given an n-state Busy Beaver we can add an $(n+1)^{st}$ state that stalls for one extra step. Recently, spurred by an earlier version of this survey, Bruce Smith obtained the following improvement (which he's kindly allowed me to share) to that trivial bound:[30]

Proposition 14 $\mathrm{BB}(n+1) \geq \mathrm{BB}(n) + 3$.

Proof. Let M be an n-state Busy Beaver. Let S be M's state immediately prior to halting when it's run on an all-0 tape, and when M enters state S for the last time, let its tape contain $b \in \{0, 1\}$ in the current square and $c \in \{0, 1\}$ in the square immediately to the left. We form a new $(n+1)$-state machine M' that's identical to M, except that

[29] See `https://www.scottaaronson.com/blog/?p=1385#comment-73298`
[30] See `https://www.scottaaronson.com/blog/?p=4916#comment-1850628`

(1) when M' is in state S and reads b, rather than halting, it writes b, moves left, and transitions to a new state Z,

(2) when M' is in state Z and reads c, it writes $1-c$, moves right, and transitions to S, and

(3) when M' is in state Z and reads $1-c$, it halts.

One can check that, on the step where M would've halted, M' instead moves left, then right, then left before halting. Hence M' runs for $\mathrm{BB}(n)+3$ steps. ■

Smith, along with an earlier pseudonymous commenter, also observed the following:[31]

Proposition 15 $\mathrm{BB}(n+2) \geq \left(1+\frac{2}{n}\right) \mathrm{BB}(n)$.

Proof. Let M be an n-state Busy Beaver, and let S be a state that's visited at least $\mathrm{BB}(n)/n$ times when M is run on an all-0 tape. We form an $(n+2)$-state machine M' that's identical to M, except that it has two new states Y and Z, and:

(1) when M' is in state S, it leaves the tape unchanged, moves right, and transitions to Y,

(2) when M' is in state Y, it leaves the tape unchanged, moves left, and transitions to Z, and

(3) when M' is in state Z, it does exactly as M would have done in S.

Then M' has the same behavior as M, except that it runs for 2 extra steps whenever it visits S. ■

In contrast to the above, suppose we want to modify a Turing machine M to run for *way* more than $s(M)$ steps: for example, $2^{s(M)}$. Then the trouble is that we quickly find ourselves adding *many* new states to M. 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 way to increase the running time massively by adding more states is due to Ben-Amram and Petersen [1] from 2002:

Theorem 16 ([1]) *Let f be any computable function. Then there exists a constant c_f such that, for all n,*

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

The main idea in the proof of Theorem 16 is what Adam Yedidia and I [15] 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

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

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:

[31] See `https://www.scottaaronson.com/blog/?p=1385#comment-74461` and `https://www.scottaaronson.com/blog/?p=4916#comment-1850926`

Lemma 17 (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 17: say, to $O(\sqrt{n})$ or even $O(\log n)$. On the other hand, to establish large, uniform gaps between (say) $\text{BB}(n)$ and $\text{BB}(n+1)$, 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 Zelinsky's problem. More concretely, let L be a programming language whose programs consist of bit-strings. Recall that $\text{BB}_L(n)$ 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 $\text{BB}_L(n)$, 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,

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

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,

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

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

5.4 Shifts Versus Ones

Using Lemma 17 (i.e., introspective encoding), Ben-Amram and Petersen [1] established other interesting inequalities: for example, that there exists a constant c such that

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

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 18 $\text{BB}(n) < \Sigma(n+1)$ *for all* $n \geq 4$.

Currently, the inequality $\text{BB}(n) < \Sigma(n+1)$ is known to fail only at $\text{BB}(3) = 21$ 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 $\text{BB}(n) \approx \Sigma(n)^2$ for all n. Or more precisely:

Conjecture 19

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

A heuristic argument for Conjecture 19 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 19 doesn't even exist.

5.5 Evolving Beavers

Suppose you already knew $\mathrm{BB}(n)$. Could a trusted wizard[32] 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 20 (implicit in Chaitin [3]) *Let L be any standard prefix-free universal programming language, such as Lisp. Then* $\mathrm{K}\left(\mathrm{BB}_L(n+1) \mid \mathrm{BB}_L(n)\right) = O(\log n)$.[33]

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

[32] 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.

[33] If one is interested in *unconditional* Kolmogorov complexity, the analysis of Chaitin's Ω used in the proof of this theorem also establishes that $\mathrm{K}\left(\mathrm{BB}_L(n)\right) = n \pm O(\log n)$.

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 20 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 20 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 20, but it does beat the trivial $n \log_2 n \pm O(n)$ bits that would be needed to describe an $(n+1)$-state Busy Beaver "from scratch."

Theorem 21 $\mathrm{K}(\mathrm{BB}(n+1) \mid \mathrm{BB}(n)) = O(n)$.

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 20. 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 20, combined with Lemma 17 (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 20, 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 21, to show (for example) that

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

Bruce Smith observes[34] that, for Turing machines, $\Omega(\log n)$ is the best that we could hope for here, since $\mathrm{K}\left(\mathrm{BB}(n)\right) = n \log_2 n \pm O(n)$. On the other hand, for bit-based programs, could Theorem 20 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.[35] 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 22 *The map $g\colon \mathbb{N} \to \mathbb{N} \cup \{\perp\}$, from Section 4, leads to $\perp$ when iterated starting from any natural number x.*

Conjecture 22 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 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?[36] Does any Busy Beaver act as a universal Turing machine?

5.7 Uniqueness of Busy Beavers

I thank Joshua Zelinsky and Nick Drozd[37] for the following question. Call Turing machines M and M' "essentially different" if they're not equivalent under permuting the states and interchanging left with right. Then there are two essentially different 1-state Turing machines that demonstrate $\mathrm{BB}(1) = 1$: one that halts immediately on reading either 0 or 1, and one that halts on 0 but continues on 1. Likewise, there turn out to be several essentially different 2-state machines that demonstrate $\mathrm{BB}(2) = 6$. By contrast, the 3- and 4-state Busy Beavers are essentially unique, as is the current 5-state candidate. What happens for larger n?

Conjecture 23 *For all $n \geq 3$, there is an essentially unique n-state Busy Beaver.*

[34] See `https://www.scottaaronson.com/blog/?p=4916#comment-1851116`

[35] 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.

[36] Since this survey was posted online, Pascal Michel informed me of his finding that a previous record-holding 6-state machine, found by Terry and Shawn Ligocki and showing that $\mathrm{BB}(6) > 2.5 \times 10^{2879}$, runs forever on certain finite inputs, using more and more tape squares to do so.

[37] See comments 66 and 77 at `https://www.scottaaronson.com/blog/?p=4916`

5.8 Structure of Busy Beavers

I thank (once again) Joshua Zelinsky[38] for the following question. If we draw the Busy Beavers and current Busy Beaver candidates for $n \leq 6$ as directed graphs, with the n states as vertices and the $2n$ transitions as edges, we find that all are *strongly connected*: that is, every state (including the initial state) is reachable from every other state via some sequence of symbols. By contrast, the $n = 7$ candidate found by "Wythagoras" does *not* have this property, as the initial state is never revisited. But, partly for that reason, the $n = 7$ candidate seems clearly suboptimal. This leads to the following conjecture:

Conjecture 24 *Every Busy Beaver Turing machine, viewed as a directed graph, is strongly connected.*

Or at least, one might conjecture that every Busy Beaver is strongly connected on all but $O(1)$ states. A related intuition, though harder to formalize, is that Busy Beavers shouldn't be "cleanly factorizable" into main routines and subroutines—but rather, that the way to maximize runtime should be via "spaghetti code," or a single n-state amorphous mass.

5.9 The Spectrum of Runtimes

Recall from Section 1 that $T(n)$ is the set of n-state Turing machines, and $s(M)$ is the running time of machine M on an all-0 input tape. Let $R(n)$ be the *spectrum of possible runtimes* of n-state machines on the all-0 input:

$$R(n) \coloneqq \{k \in \mathbb{N} : s(M) = k \text{ for some } M \in T(n)\}.$$

There are many interesting questions that we could ask about $R(n)$, beyond just its maximum element (namely $\mathrm{BB}(n)$). As one example, how many distinct runtimes are there for n-state machines? As a second example, how does the gap between $\mathrm{BB}(n)$ and the *second*-longest runtime behave, as a function of n? For $n \in \{2, 3\}$, this gap is only 1, but for $n = 4$ the gap appears to be 10, and for $n = 5$, assuming Conjecture 10, it appears to be $23,622,106$, almost exactly half the runtime of the current 5-state champion (see Michel [12]).

Here is a third example: following a suggestion by a commenter on my blog,[39] define the n^{th} *Lazy Beaver Number*, or $\mathrm{LB}(n)$, to be the least $k \in \mathbb{N}$ such that $k \notin R(n)$: that is, such that no n-state machine runs for exactly k steps. Unlike the BB function, the LB function is clearly computable. Moreover, by a simple counting argument, we have

$$\mathrm{LB}(n) \leq |T(n)| + 1 \leq (4n + 1)^{2n} + 1.$$

What else can we say about the growth rate of LB? Certainly $\mathrm{LB}(n + 1) > \mathrm{LB}(n)$ for all n, since we can always use an extra state either to maintain the same runtime or to increase it by 1. Also, via explicit constructions of Turing machines, and using Lemma 17 (the Introspective Encoding Lemma of Ben-Amram and Petersen [1]), it is not hard to show that there exists a constant c such that $\mathrm{LB}(n) \geq D(n)/c^n$ for all n, where $D(n)$ is the number of essentially different n-state Turing machines. I conjecture that a stronger lower bound holds:

[38] See `https://www.scottaaronson.com/blog/?p=4916#comment-1850560`

[39] See `https://www.scottaaronson.com/blog/?p=4916#comment-1850265`

Conjecture 25

$$\mathrm{LB}(n) \geq \frac{D(n)}{n^{O(1)}}.$$

Regarding concrete values, my modest searches revealed that $\mathrm{LB}(n) = \mathrm{BB}(n) + 1$ for $n \in \{1, 2, 3\}$, while Niels Lohmann (personal communication) calculated that $\mathrm{LB}(4) = 72$ and $\mathrm{LB}(5) = 427$. It also looks feasible to calculate $\mathrm{LB}(6)$.

5.10 Beeping Busy Beavers

The following question emerged from an email conversation between me and Harvey Friedman. Recall, from Section 3, that one can define a function $\mathrm{BB}_1(n)$, involving Turing machines with oracles for the original BB function, which grows uncomputably quickly even given an oracle for BB. Could we compute the first values of BB_1? Alas, this is liable to be uninteresting, just because the details of how a Turing machine queries a BB oracle (by writing n onto an oracle tape, etc.) will involve many kludgy and non-canonical choices, and one might need many states before one saw the effect of those choices.

But there's an alternative. Define a *beeping Turing machine* to be exactly the same as one of the Turing machines from Section 1, except that it emits a "beep" as it exits a certain designated state, called the "beep state." Then given a beeping Turing machine M, let $b(M)$ be the number of the last step on which M beeps (or $b(M) := \infty$ if M beeps infinitely often), when M is run on an initially all-0 input tape. Finally, define the n^{th} Beeping Busy Beaver number by

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

Clearly $\mathrm{BBB}(n) \geq \mathrm{BB}(n)$ for all n, since we could designate the state from which M halts as its beep state. One can show that, as n gets larger, BBB must grow uncomputably faster than even BB—indeed, it grows at a similar rate to BB_1, in the sense that BBB and BB_1 are both computable given an oracle for any upper bound on the other. This is because the problem of whether a given machine M beeps finitely many times is complete for Σ_2, the second level of the arithmetical hierarchy. By contrast, the question of whether M halts is complete for Σ_1.[40]

Via case analysis, I've confirmed that $\mathrm{BBB}(1) = 1$ and $\mathrm{BBB}(2) = 6$, same as for the standard BB function. I've also confirmed that

$$\mathrm{BBB}(3) \geq 55 > \mathrm{BB}(3) = 21,$$

via the following 3-state machine, with B as its beep state:

	A	**B**	**C**
0	1LB	1RA	1RC
1	0RB	0LC	1RA

[40] A different way to get BB_1-like growth, which was actually Friedman's original suggestion, would be to define a Busy Beaver function for *nondeterministic* Turing machines, with the "running time" of such a machine taken to be the number of steps in its longest halting path. However, BBB struck me as more amenable to empirical study for small values of n.

After last visiting state B at step 55, this machine spends an eternity in state C. I found many other 3-state machines that approach this performance but none that exceed it, which leads to the guess BBB(3) = 55.

Beyond proving or disproving that guess, it would be extremely interesting to know by just how much BBB(4) exceeds BB(4),[41] and to compare the best lower bounds for $n \in \{5, 6, 7\}$. Also, can we show that the value of BBB(n) is independent of ZF set theory, for n much smaller than what we know for BB?

5.11 Busy Beaver and Number Theory

I'll conclude with questions posed by my former student Andy Drucker. He asked:

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

Currently, we know only that BB(2) is even, while BB(1), BB(3), and BB(4) are odd. If Conjecture 10 holds then BB(5) is even.

We could likewise ask: is BB(n) infinitely often prime? 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 world 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 Bill Gasarch for proposing this article, Lily Aaronson for raising the question about BB(n) versus A(n), Daniel Briggs and Georgi Georgiev for answering my questions about the 5-state holdout machines, Oscar Cunningham for the comment in Section 2, Andy Drucker for raising the questions in Section 5.11, Harvey Friedman for inspiring Section 5.10, Marijn Heule for Figure 1 and for inspiring Section 5.6, Bjørn Kjos-Hanssen for Proposition 4, Niels Lohmann for the observation in Section 3 and for calculating LB(4) and LB(5), Luke Schaeffer for helping me understand the

[41] After this survey was placed online, Nick Drozd found that BBB(4) ≥ 2,819. See `https://www.scottaaronson.com/blog/?p=4916#comment-1852398`

ideas in Sections 5.3 and 5.5, Bruce Smith for Propositions 14 and 15, and Joshua Zelinsky for raising the questions in Sections 5.3, 5.7, and 5.8. I also thank Nick Drozd, Adam Klivans, Heiner Marxen, Pascal Michel, Toby Ord, John Stillwell, Ronald de Wolf, and the aforementioned people for discussions and feedback.

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. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.136.5997&rep=rep1&type=pdf`.

[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. `https://pdfs.semanticscholar.org/4d96/c256994f190617a34aac56c9b9bfb23f43d9.pdf`.

[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. `https://faculty.baruch.cuny.edu/lkirby/accessible_independence_results.pdf`.

[8] P. Kropitz. *Busy Beaver Problem.* Bachelors thesis, Charles University in Prague, 2011. In Czech. `https://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. `http://turbotm.de/~heiner/BB/mabu90.html`.

[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. `https://arxiv.org/abs/0906.3749`, 2019.

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

[14] R. M. Solovay. Hyperarithmetically encodable sets. *Transactions of the AMS*, 239:99–122, 1978. `https://www.ams.org/journals/tran/1978-239-00/S0002-9947-1978-0491103-7/S0002-9947-1978-0491103-7.pdf`.

[15] A. Yedidia and S. Aaronson. A relatively small Turing machine whose behavior is independent of set theory. *Complex Systems*, (25):4, 2016. `https://arxiv.org/abs/1605.04343`.