Build your own 8-bit busy beaver on a breadboard!

or: Look, it's clearly decidable whether any program on your computer terminates or not.

There is a straightforward decision procedure for determining whether any deterministic algorithm running on a correctly operating physical computer either terminates or fails to terminate. This, in turn, means that every physical computer has a computable "busy beaver" quantity, the maximum number of steps taken by any terminating program before terminating.

In this paper, we provide some preliminary results in the context of an 8-bit computer design popular in electronic hobbyist circles. We procrastinated starting on this research, and are therefore only able to present lower bounds at this time. The relatively trivial future work is left as an exercise for the reader, via crowdsourcing, at http://sisyphean.glitch.me/.

ACH Reference Format:
Robert J. Simmons. 2021. Build your own 8-bit busy beaver on a breadboard! *SIGBOVIK* (April 1, 2021), 4 pages.

1 INTRODUCTION

Computer science is built upon a foundation of lies and deceit. The dirtiest secret of every second-semester computer science class is that algorithms don't really exist. Your algorithms textbook lets you prove that you can efficiently perform binary search on your array of 10^{80} `uint64_t` values, but when you try to allocate the 8-yottabyte array needed to perform that calculation, your AWS bill is gonna go through the roof real fast.

One of the foremost lies of so-called "computer science" is the existence of Turing-so-called "machines." Turing machines do **not** exist [Murphy VII 2008], as their standard definition posits the straightforwardly nonsensical existence of a tape of infinite length.

If we accept the definition of computer science presented by Newell, Simon, and Perlis [Newell et al. 1967], namely as

> …the theory and design of computers, as well as the study of all the phenomena arising from them…

then nonsense such as the study of Turing machines must be seen as belonging, *not to computer science*, but to lesser fields such as mathematics that still deign to associate with patently absurd nonsense like the infinity of positive integers.

This work is part of a project bringing crude mathematical "results" "in" "computer science" into the realm of the reality of physical computers. We seek to do so in a style accessible to a hobbyist or layperson – no soldering required.

1.1 Termination on a MacBook Pro

A bedrock result in "computer science" is that there is no general procedure for determining whether any given Turing "machine" terminates, given the Turing machine's initial configuration.

The computer on which I am authoring this paper has no access to absurdities like infinite tape. Instead, it has approximately 250 gigabytes of hard drive storage and 16 gigabytes of random access memory, plus associated internal state (registers, page tables, and flags). It's safe to say the computer has less than 4 terabits of state, a measly 2^{42} switches that can be set to an "on" or "off" state.

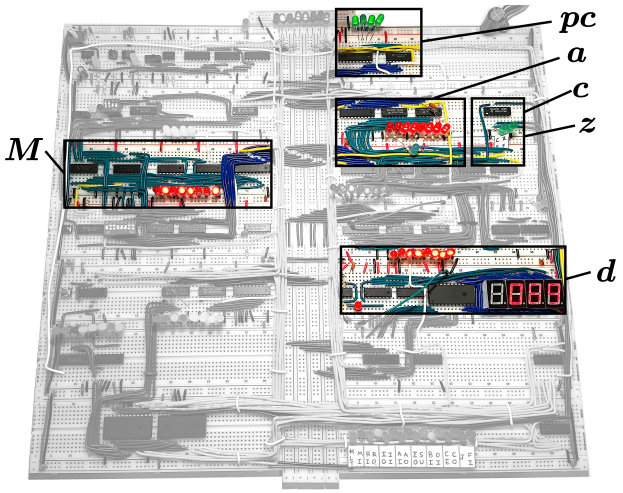

Fig. 1. A typical Ben Eater 8-Bit Breadboard Computer build, with the state relevant to the ISA highlighted. (Image from Reddit user `-wellplayed-`.)

Absent external outputs[1] this computer is a deterministic machine whose behavior is entirely determined by those 2^{42} bits of *state*. If, during an uninterrupted course of operation, the same pattern of 2^{42} bits is encountered and then, after n execution steps, encountered *again*, it is a certainty that after another n steps of uninterrupted execution that pattern will, *once again*, be encountered. *A repeated state gives an immediate proof of non-termination.*

There are a mere $2^{2^{42}}$ possible states that this machine might be in, leading to a trivial algorithm for determining whether any program halts or not: let the machine run through $2^{2^{42}}$ steps of execution. If the program is still running, then by the pigeonhole principle, one of those states was repeated, and so the program does not terminate.

1.2 Plan of this work

The workaday computer scientist generally cannot wait for $2^{2^{42}}$ execution steps, even with the enhanced efficiency of the new M1 chip, the first chip designed specifically for Mac [Apple 2021]. Therefore, we will explore the clear decidability of the halting problem on a popular hobbyist computer with a more approachable state space.

Having done this, we will turn to several variations of the *Busy Beaver problem*, the search for the longest-running halting program on a computational device. We also introduce a *Sisyphean Beaver problem* as a contribution to the hobbyist DIY computer community.

We conclude by crowdsourcing future results.

[1] This work concerns only deterministic programs and algorithms. In this context, "deterministic" is meant to exclude programs relying on input from the outside world after starting execution, whether in the form of interactive input or physics-based random number generators.

2 BEN EATER'S 8-BIT BREADBOARD COMPUTER

Ben Eater is a former Khan Academy instructor [Eater 2018], educational YouTuber [Eater 2021], and designer of the world's worst VGA card [Eater 2019]. Through a series of explanatory YouTube videos and commercially available kits, he has popularized a simple computer design, inspired by Malvino's SAP-1 [Malvino 1977], that can be built on a dozen or so breadboards with relatively primitive integrated circuits [Eater 2017].

The 8-bit breadboard computer designed by Eater uses several clock cycles to compute a single instruction, and so has a few bytes of internal state that carry out the multiple parts of a single instruction. We can safely ignore them for this presentation, and describe the machine model for the "Eater ISA" as having a total of 86 bits of internal state:

- Two one-byte registers, an accumulator a and a display register d that displays its contents in decimal.
- One four-bit memory address register pc.
- 16 bytes of addressable memory M.
- Two one-bit flag registers c and z that are set whenever an ADD or SUB operation is performed. The c register is set to the carry-out bit of the adder, and the z bit is set to 1 if the result of the operation is a zero, and is set to 0 otherwise.

The Eater ISA is described in Figure 2. An execution step consists of two phases. In the first phase, the machine uses the program counter pc to fetch the next instruction from $M[pc]$. In the second phase, the machine updates its state according to the fetched instruction's function. The machine always starts with a, d, pc, c, and z set to zero.

2.1 Undefined behavior

Six opcodes are undefined; to fully specify the machine's behavior these must be resolved. In this paper, we'll consider two possibilities:

- The opcodes are truly invalid, and any initial state that leads to the machine attempting to execute an invalid opcode cannot be said to either terminate or run forever.
- All unused opcodes are uniformly aliased to one of the eleven other opcodes. For example, if they are uniformly aliased to NOP, then `00`, `9C`, and `C0` are all no-op instructions. If they are all uniformly aliased to LDI, then `5C`, `9C`, and `CC` all load the value 13 (i.e. `0xC`) into the accumulator.

The two most reasonable instructions for aliasing to are certainly NOP and HLT, perhaps followed by OUT. Eater's own implementation of the ISA effectively aliases undefined opcodes to NOP.

We leave to future work more esoteric and/or practical uses of this undefined behavior, such as playing happy birthday [Wikipedia contributors 2021], becoming self-aware [Adams 1979], or rotating the board [Simmons 2018b].

2.2 Even simpler

By moving just a couple of wires, an implemented 8-bit breadboard computer can have its 16-byte memory M modified by pinning one, two, three, or all four of the memory's high-order bits to a specific value. This has the effect of turning the 16-byte memory into an 8-byte, 4-byte, 2-byte, or 1-byte memory (respectively).

Opcode	Mnemonic	Function
0	NOP	$pc \leftarrow pc + 1$
1	LDA	$pc \leftarrow pc + 1 \quad a \leftarrow M[n]$
2	ADD	$pc \leftarrow pc + 1 \quad a \leftarrow a + M[n]$
3	SUB	$pc \leftarrow pc + 1 \quad a \leftarrow a - M[n]$
4	STA	$pc \leftarrow pc + 1 \quad M[n] \leftarrow a$
5	LDI	$pc \leftarrow pc + 1 \quad a \leftarrow n$
6	JMP	$pc \leftarrow n$
7	JC	$pc \leftarrow n$ if the c flag is set $pc \leftarrow pc + 1$ otherwise
8	JZ	$pc \leftarrow n$ if the z flag is set $pc \leftarrow pc + 1$ otherwise
14	OUT	$pc \leftarrow pc + 1 \quad d \leftarrow a$
15	HLT	*halt the machine*

Fig. 2. ISA specification for Ben Eater's 8-bit breadboard computer. Each eight-bit instruction has a four-byte opcode in the high-order bits followed by a four-byte operand n in the low-order bits. Opcodes 9 to 13 are unspecified, and ADD and SUB additionally (re)set the c and z flags.

3 THE BUSY BEAVER

The busy beaver function, $BB(n)$, is a classic example of a fast-growing non-computable function. It is defined in terms of Turing "machines" that read and write binary digits to a tape. $BB(n)$ is the maximum number of steps taken by an n-state, two-symbol Turing machine that halts [Adam Yedidia and Scott Aaronson 2016].

It's straightforward to *enumerate* the n-state Turing machines: in each of the n states, the Turing machine has to specify what it will do if it reads a 0 and if it reads a 1. There are only five possibilities: halt or write (a 0 or a 1) and move (left or right). Thus, there are 10^n initial Turing machine configurations with n states.

The hard part is figuring out whether each of the 10^n machines halt. If you can show a machine ever returns to a prior state, then it definitely will run forever. If you can show a machine halts, then, very well. But the tape, existing as it does as a piece of blatant mathematical nonsense, is *infinite*: you can't play the trick you played with my MacBook and just wait patiently for it to perform $2^{2^{42}}$ steps of computation.

Indeed, a fundamental characteristic of mathematical fictions like Turing "machines" or lambda calculus evaluation is that they may fail to terminate by repeating old states, *and* they can fail to terminate in other ways too.

For example, the lambda calculus term $(\lambda x.xx)(\lambda x.xx)$ evaluates to itself via a call-by-value evaluation strategy, and reaching a single repeated state suffices to show that it will evaluate forever [Simmons 2018a]. That's an instance of a lambda term failing to terminate by repeating an old state. On the other hand, $(\lambda x.(xx)x)(\lambda x.(xx)x)$ will never repeat a previous state in its endless evaluation (Figure 3).

4 THE BREADBOARD BUSY BEAVER

The Ben Eater Eight-Bit Breadboard Busy Beaver, $BEEBBBB(s)$, is a computable function, defined as maximum number of execution steps that an 8-bit breadboard computer running the Eater ISA with s bits of addressable memory can take before halting.

$$
\begin{aligned}
&(\lambda x.(xx)x)(\lambda x.(xx)x)\\
\rightarrow\ &((\lambda x.(xx)x)(\lambda x.(xx)x))(\lambda x.(xx)x)\\
\rightarrow\ &(((\lambda x.(xx)x)(\lambda x.(xx)x))(\lambda x.(xx)x))(\lambda x.(xx)x)\\
\rightarrow\ &((((\lambda x.(xx)x)(\lambda x.(xx)x))(\lambda x.(xx)x))(\lambda x.(xx)x))(\lambda x.(xx)x)\\
\rightarrow\ &(((((\lambda x.(xx)x)(\lambda x.(xx)x))(\lambda x.(xx)x))(\lambda x.(xx)x))(\lambda x.(xx)x))(\lambda x.(xx)x)\\
\rightarrow\ &((((((\lambda x.(xx)x)(\lambda x.(xx)x))(\lambda x.(xx)x))(\lambda x.(xx)x))(\lambda x.(xx)x))(\lambda x.(xx)x))(\lambda x.(xx)x)\\
\rightarrow\ &(((((((\lambda x.(xx)x)(\lambda x.(xx)x))(\lambda x.(xx)x))(\lambda x.(xx)x))(\lambda x.(xx)x))(\lambda x.(xx)x))(\lambda x.(xx)x))(\lambda x.(xx)x)
\end{aligned}
$$

Fig. 3. The non-repeating evaluation of $(\lambda x.(xx)x)(\lambda x.(xx)x)$

BEEBBBB(s) is only well-defined for $s = 0, 1, 2, 3$, and 4, given that the 8-bit ISA does not have an obvious extension to allow addressing beyond 4 bits.[2]

Unlike the Busy Beaver function, *BEEBBBB*(s) is trivial to bound above. An 8-bit breadboard computer with s bits of addressable memory has $s + 22$ bytes of state, and because the d register cannot influence execution, we can safely pretend that the machine has only $8s + 14$ bytes of state. Thus, the machine can exist in

$$2^{8\times 2^s+14} = 16384 \times 256^{2^s}$$

distinct configurations, and so we define this function as the Upper Bound for the Ben Eater Eight-Bit Breadboard Busy Beaver, *UBBEEBBBB*(s), shown in Figure 4. The pigeonhole principle necessitates that if the machine runs for *UBBEEBBB*$(s) + 1$ steps, it has repeated at least one state and will therefore repeat that state an infinite number of additional times.

The 14 non-memory state bits have set initial values, so bounding *BEEBBBB*(s) from below can be done through random or exhaustive state-space exploration of the 256^{2^s} possible initial states [Sturtevant and Ota 2018].

We have found the exact value of *BEEBBBB*(s) for $s \leq 2$, and by random state space exploration have investigated the Below Bound for the Ben Eater Eight-Bit Breadboard Busy Beaver, *BBBEEBBBB*(s), for $s = 3$ and $s = 4$, as shown in Figure 5. The precise value of *BEEBBBB*(s), and therefore *BBBEEBBBB*(s), depends on the interpretation of undefined opcodes. In Figure 5, we present results for all eleven variants described in Section 2.1.

4.1 The Four-Byte Busy Beaver

By investigating all 4 billion possible initial configurations, under each of the eleven variants conditions described in Section 2.1. we determined the value of *BEEBBBB*(2), the Ben Eater Eight-Bit Breadboard Busy Beaver for a computer with $2^2 = 4$ bytes of addressable memory, for all interpretations of the undefined opcodes. If executions that reach undefined opcodes are excluded, or if they are treated as HLT, LDA, SUB, LDI, JMP, *or* JC, then *BEEBBBB*$(2) = 773$.[3]

There are several distinct programs that achieve this maximal execution, but they all follow the same pattern, and the pattern is kind of cute. Here's a representative four-byte breadboard busy beaver:

[2]Many hobbyists have extended Ben Eater's design to allow 8-bit or even 16-bit addresses, but this requires changing the Eater ISA, and we leave the investigation here to future work.

[3]If undefined opcodes are interpreted as a NOP or JZ, then *BEEBBBB*$(2) = 835$, if they are interpreted as ADD, then *BEEBBBB*$(2) = 838$, and if they are interpreted as STA, then *BEEBBBB*$(2) = 1446$.

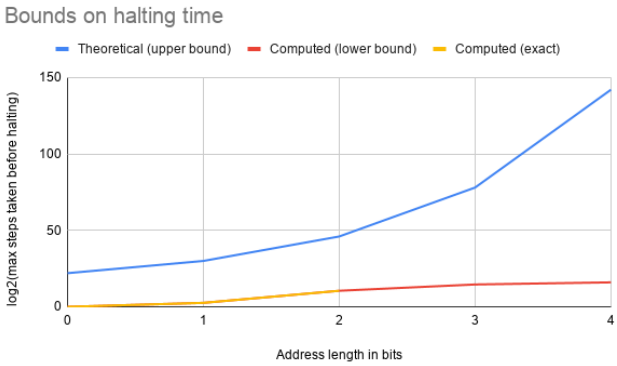

Fig. 4. The theoretical upper bound of halting time for *any* machine with $8 \times 2^s + 14$ bits of state, compared to the computed exact and lower bounds of halting time for the 8-bit breadboard computer and Eater ISA.

```
M[0] = 31 (Subtract M[1] from accumulator)
M[1] = 01 (No-op, also the literal 1)
M[2] = 70 (If carry flag set, jump to beginning)
M[3] = 43 (Store the accumulator's value in M[3])
```

The first three instructions are never modified, so the first two instructions, taken together, always subtract 1 from the accumulator. Because subtraction is done via twos-complement addition, subtracting 1 is equivalent to adding 255, so the the carry bit will always be set *except* when the accumulator was 0 prior to subtraction.

The accumulator starts out set to 0, so the first time $pc = 2$, the carry bit is not set and the branch is not taken. When $pc = 3$ subsequently, the instruction in position 3 will overwrite itself with the value in the accumulator: 255, or FF in hexadecimal. This value, critically, is interpreted as a halt instruction.

The program counter will then overflow so that $pc = 0$.

The instructions 1 through 3 will then run 256 times, with the carry bit set the first 255 of those times; the accumulator will be decremented each time, until it once again contains 0. The two-hundred-and-fifty-sixth decrement will fail to set the c flag, so the JC instruction will not modify the program counter, allowing it to advance to 3 for the second time. $M[3]$ contains FF, a halt instruction, so the computer halts.

$$4 + (3 \times 256) + 1 = 773$$

This gives us a *BEEBBBB*$(2) = 773$ if we disallow the execution of any undefined instructions.

4.2 The Sisyphean Beaver

For finite computing machines like my MacBook Pro or the 8-bit breadboard computer, non-termination requires that previous states be repeated over and over. Note that non-termination in these settings is generally *desirable*. I don't want my MacBook Pro's operating system to terminate unless I tell it to! Likewise, as the ultimate end goal of many 8-bit breadboard computers is to hang on one's wall and produce interesting blinkenlights indefinitely: a halting program is undesirable for this goal. A *Sisyphean Beaver* is therefore also of interest: an initial state whose execution enters the longest possible cycle.

We will define the Ben Eater Eight-Bit Breadboard Endless Beaver $BEEBBEB(s)$ as the length of the longest cycle in the execution of any 8-bit breadboard computer, running the Eater ISA.

5 CROWDSOURCING RESULTS

We intend, by April 1, 2021, to have http://sisyphean.glitch.me/ set up to solicit community assistance at raising the lower bound of $BEEBBB(s)$ and $BEEBEB(s)$ for $s = 3$ and $s = 4$, where at present there are only lower bounds. In this search, we will only consider the NOP interpretation of undefined instructions, in keeping with the implementation of most actual 8-bit breadboard computers.

It can be reasonably expected that many of the programs with the longest loops or longest halting times will ignore the OUT instruction that stores a value in the d register, which displays its contents in decimal. That's a shame, because these are some lights that a proud 8-bit breadboard computer owner would presumably wish to have blinken. Therefore, we will initially present at least 256 different leade-boards for each of four conditions ($s = 3$ and $s = 4$, with both halting and looping variants). One board for programs that do not set the d register within their path to halting, one for the programs that set it to 1 distinct value, one for programs that set it to 2 distinct values, and so on through programs that set the d register to all 256 possible values.

If you've created 1024 leaderboards, you probably missed one. Additional "leagues" or rankings based interestingness and/or entropy of various sequences are left for future work.

6 FUTURE WORK

The upper bounds in Figure 4 hold for *any* machine with $8 \times 2^s + 14$ bits of configurable state. In particular, they would work with a *arbitrary* ISA, and defining ISAs or alternate execution semantics that allow one to approach these bounds without being obviously pathological is an interesting challenge, especially if one restricts onesself to ISAs that can be implemented with primitive logic chips, keeping with the spirit of the 8-bit breadboard computer.

A less significant change that would (mostly) preserve the Eater ISA would be to move from a von Neumann architecture, where programs are just data stored in addressable and modifiable memory, to a Harvard architecture where instructions were drawn from a *separate* 16-byte read-only array $I[pc]$ that is distinct from the 2^s byte array $M[n]$. This change would make for substantially more powerful halting computations and more interesting Sisyphean beavers with no change to the design of Eater ISA and minimal changes to its semantics or to the physical computer's architecture.

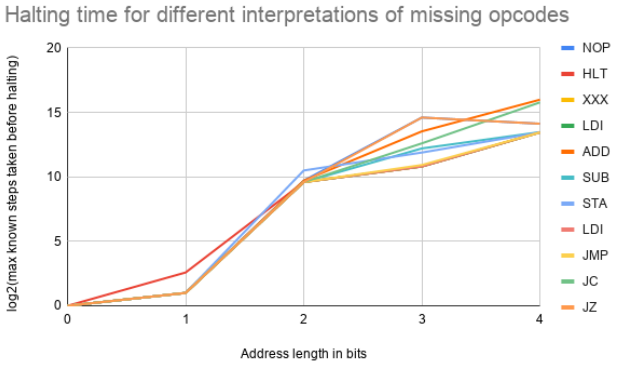

Fig. 5. Various computed lower bounds for halting time given various uniform interpretations of undefined instructions. The line XXX represents what happens when any run that reaches an undefined instruction is simply thrown out. Values are exact for 0, 1, and 2, and are lower bounds from random state space exploration for 3 and 4.

REFERENCES

Adam Yedidia and Scott Aaronson 2016. *A Relatively Small Turing Machine Whose Behavior Is Independent of Set Theory.* Retrieved March 19, 2021 from https://www.scottaaronson.com/busybeaver.pdf

Douglas Adams. 1979. *The hitchhiker's guide to the galaxy.* Pan Books.

Apple. 2021. *Small chip. Giant leap.* Cupertino in California. https://www.apple.com/mac/m1/

Ben Eater. 2017. *Build an 8-bit computer from scratch.* Retrieved March 19, 2021 from https://eater.net/8bit

Ben Eater. 2018. *Linkedin page.* Retrieved March 19, 2021 from https://www.linkedin.com/in/beneater/

Ben Eater. 2019. *Let's build a video card!* Retrieved March 19, 2021 from https://eater.net/vga

Ben Eater. 2021. *YouTube page.* Retrieved March 19, 2021 from https://www.youtube.com/channel/UCS0N5baNlQWJCUrhCEo8WlA

Albert Paul Malvino. 1977. *Digital Computer Electronics.* McGraw-Hill.

Tom Murphy VII. 2008. A non-non-destructive strategy for proving P = NP. In *A Record of The Proceedings of SIGBOVIK 2008 (SIGBOVIK, Vol. 2)*, Ciel Elf and Guy Fantastic (Eds.). The Association of Computational Heresy, Pittsburgh, PA, 13–15.

Allen Newell, Alan J. Perlis, and Herbert A. Simon. 1967. What is computer science? *Science* 157 (1967), 1373–1374.

Robert J. Simmons. 2018a. On unlexable programming languages. In *A Record of The Proceedings of SIGBOVIK 2011 (SIGBOVIK, Vol. 5).* The Association of Computational Heresy, Pittsburgh, PA, 79–82.

Robert J. Simmons. 2018b. That's Numberwangcoin!. In *A Record of The Proceedings of SIGBOVIK 2018 (SIGBOVIK, Vol. 12).* The Association of Computational Heresy, Pittsburgh, PA, 36–38.

Nathan R. Sturtevant and Matheus Jun Ota. 2018. Exhaustive and Semi-Exhaustive Procedural Content Generation. In *Proc. 14th Artif. Intell. Interactive Digit. Entertainment Conf.* 109–115.

Wikipedia contributors. 2021. *Happy Birthday to You — Wikipedia, The Free Encyclopedia.* Retrieved March 19, 2021 from https://en.wikipedia.org/wiki/Happy_Birthday_to_You