Correctness of bbchallenge's deciders

Tristan Stérin

Abstract

The Busy Beaver Challenge (or bbchallenge) aims at collaboratively solving the following conjecture: "BB(5) = 47,176,870" [Aaronson, 2020]. This goal amounts to decide whether or not 88,664,064 Turing machines with 5-state halt or not – starting from all-0 tape. In order to decide the behavior of these machines we write *deciders*. A decider is a program that takes as input a Turing machine and outputs `true` if it is able to tell whether the machine halts or not. Each decider is specialised in recognising a particular type of behavior that can be decided.

In this document we are concerned with proving the correctness of these deciders programs. More context and information about this methodology are available at `https://bbchallenge.org`.

1 Conventions

	0	1
A	1RB	1LC
B	1RC	1RB
C	1RD	0LE
D	1LA	1LD
E	- - -	0LE

Table 1: Transition table of the current 5-state busy beaver champion: it halts after 47,176,870 steps. `https://bbchallenge.org/1RB1LC1RC1RB1RD0LE1LA1LD---0LA&status=halt`

The set $\mathbb{N}$ denotes $\{0, 1, 2 \dots\}$.

Turing machines. The Turing machines that are studied in the context of bbchallenge use a binary alphabet and a single bi-infinite tape. Machine transitions are either undefined (in which case the machine halts) or given by (a) a symbol to write (b) a direction to move (left or right) and (c) a state to go to. Table 1 gives the transition table of the current 5-state busy beaver champion. The machine halts after 47,176,870 steps (starting from all-0 tape) when it reads a 0 in state E, which is undefined.

A *configuration* of a Turing machine is defined by the 3-tuple: (i) state (ii) position of the head (iii) content of the memory tape. In the context of bbchallenge, *the initial configuration* of a machine is always (i) state is 0, i.e. the first state to appear in the machine's description (ii) head's position is 0 (iii) the initial tape is all-0 – i.e. each memory cell is containing 0. We write $c_1 \vdash_{\mathcal{M}} c_2$ if a configuration c_2 is obtained from c_1 in one computation step of machine $\mathcal{M}$. We omit $\mathcal{M}$ if it is clear from context. We let $c_1 \vdash^s c_2$ denote a sequence of s computation steps, and let $c_1 \vdash^* c_2$ denote zero or more computation steps. We write $c_1 \vdash \perp$ if the machine halts after executing one computation step from configuration c_1. In the context of bbchallenge, halting happens when an undefined machine transition is met i.e. no instruction is given for when the machine is in the state, tape position and tape corresponding to configuration c_1.

2 Decider for "Cyclers"

The goal of this decider is to recognise Turing machines that cycle through the same configurations forever. Such machines never halt. The method is simple: remember every configuration seen by a machine and return `true` if one is visited twice. A time limit (maximum number of steps) is also given for running the test in practice: the algorithm recognises any machine whose cycle fits within this limit[1].

[1] In practice, for machines with 5 states the decider was run with 1000 steps time limit.

We assume that we are given a procedure **TuringMachineStep**(machine,configuration) which computes the next configuration of a Turing machine from the given configuration or **nil** if the machine halts at that step.

2.1 Pseudocode

Algorithm 1 DECIDER-CYLERS

```
1: struct Configuration {
2:     int state
3:     int headPosition
4:     int → int tape
5: }
6: procedure DECIDER-CYLERS(machine,timeLimit)
7:     Configuration currConfiguration = {.state = 0, .headPosition = 0, .tape = {0:0}}
8:     Set<Configuration> configurationsSeen = {}
9:     int currTime = 0
10:    while currTime < timeLimit do
11:        if currConfiguration in configurationsSeen then
12:            return true
13:        configurationsSeen.insert(currConfiguration)
14:        currConfiguration = TuringMachineStep(machine,currConfiguration)
15:        currTime += 1
16:        if currConfiguration == nil then
17:            return false //machine has halted
18:    return false
```

2.2 Correctness

Theorem 1. Let $\mathcal{M}$ be a Turing machine and $t \in \mathbb{N}$ a time limit. Let c_0 be the initial configuration of the machine. There exists $i \in \mathbb{N}$ and $j \in \mathbb{N}$ such that $c_0 \to^i c_i \to^j c_i$ with $i + j \leq t$ if and only if DECIDER-CYCLERS($\mathcal{M}$,t) returns true.

Proof. This follows directly from the behavior of DECIDER-CYCLERS($\mathcal{M}$,t): all intermediate configurations below time t are recorded and the algorithm returns true if and only if one is visited twice. This mathematically translates to there exists $i \in \mathbb{N}$ and $j \in \mathbb{N}$ such that $c_0 \to^i c_i \to^j c_i$ with $i + j \leq t$, which is what we want. □

Corollary 2. Let $\mathcal{M}$ be a Turing machine and $t \in \mathbb{N}$ a time limit. If DECIDER-CYCLERS($\mathcal{M}$,t) returns true then the behavior of $\mathcal{M}$ from all-0 tape has been decided: $\mathcal{M}$ does not halt.

Proof. By Theorem 1, there exists $i \in \mathbb{N}$ and $j \in \mathbb{N}$ such that $c_0 \to^i c_i \to^j c_i$ with $i + j \leq t$. It follows that for all $k \in \mathbb{N}$, $c_0 \to^{i+kj} c_i$. The machine never halts as it will visit c_i infinitely often. □

References

[1] S. Aaronson. The busy beaver frontier. *SIGACT News*, 51(3):32–54, Sept. 2020. `https://www.scottaaronson.com/papers/bb.pdf`.