Correctness of the decider for cyclers
bbchallenge

Tristan Stérin

Abstract

We give the pseudocode of the decider for the "Cyclers" family of bbchallenge and we prove its correctness. For more context please refer to `https://bbchallenge.org/` and `https://discuss.bbchallenge.org/t/decider-cyclers/33` for this decider in particular. The implementation of this decider is available at: `https://github.com/bbchallenge/bbchallenge-deciders/`.

1 Pseudocode

The goal of this decider is to recognise Turing machines that cycle through the same configurations for ever where a configuration is defined by the 3-tuple: (i) state (ii) position of the head (iii) content of the memory tape. 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].

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

Algorithm 1 DECIDER-CYLERS

```
struct Configuration {
    int state
    int headPosition
    int → int tape
}
procedure DECIDER-CYLERS(machine,timeLimit)
    Configuration currConfiguration = {.state = 0, .headPosition = 0, .tape = {0:0}}
    Set<Configuration> configurationsSeen = {}
    int currTime = 0
    while currTime < timeLimit do
        if currConfiguration in configurationsSeen then
            return true
        configurationsSeen.insert(currConfiguration)
        currConfiguration = TuringMachineStep(machine,currConfiguration)
        currTime += 1
        if currConfiguration == nil then
            return false //machine has halted
    return false
```

2 Correctness

The set $\mathbb{N}$ denotes $\{0, 1, 2 \ldots\}$. The Turing machines that are studied in the context of bbchallenge use a binary alphabet and a single bi-infinite tape. A *configuration* 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

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

(ii) head's position is 0 (iii) the initial tape is all-0 – i.e. each memory cell is containing 0. In one step, machine $\mathcal{M}$ transitions from configuration c to c' and we write $c \to_{\mathcal{M}} c'$. If the machine halts during that step we write $c \to_{\mathcal{M}} \perp$. By convention, $\perp \to_{\mathcal{M}} \perp$ is a valid transition for any machine $\mathcal{M}$. The operator $\to^n_{\mathcal{M}}$ is $\to_{\mathcal{M}}$ applied $n \in \mathbb{N}$ times. We write $c \to c'$ when the machine is clear from context.

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. □