Computation Answers 6: Turing Machines and Computable Functions

1. (a) M performs the computation:

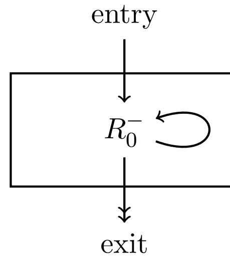
$$
\begin{aligned}
(start, \epsilon, 011\sqcup111\sqcup110) &\rightarrow_M (skip1s, 0, 11\sqcup111\sqcup110)\\
&\rightarrow_M (skip1s, 01, 1\sqcup111\sqcup110)\\
&\rightarrow_M (skip1s, 011, \sqcup111\sqcup110)\\
&\rightarrow_M (skip{*}s, 011{*}, 111\sqcup110)\\
&\rightarrow_M (shift, 011, {*}{*}11\sqcup110)\\
&\rightarrow_M (shift, 01, 1{*}{*}11\sqcup110)\\
&\rightarrow_M (append, 011, {*}{*}11\sqcup110)\\
&\rightarrow_M (skip{*}s, 0111, {*}11\sqcup110)\\
&\rightarrow_M (skip{*}s, 0111{*}, 11\sqcup110)\\
&\rightarrow_M (shift, 0111, {*}{*}1\sqcup110)\\
&\rightarrow_M (shift, 011, 1{*}{*}1\sqcup110)\\
&\rightarrow_M (append, 0111, {*}{*}1\sqcup110)\\
&\rightarrow_M (skip{*}s, 01111, {*}1\sqcup110)\\
&\rightarrow_M (skip{*}s, 01111{*}, 1\sqcup110)\\
&\rightarrow_M (shift, 01111, {*}{*}\sqcup110)\\
&\rightarrow_M (shift, 0111, 1{*}{*}\sqcup110)\\
&\rightarrow_M (append, 01111, {*}{*}\sqcup110)\\
&\rightarrow_M (skip{*}s, 011111, {*}\sqcup110)\\
&\rightarrow_M (skip{*}s, 011111{*}, \sqcup110)\\
&\rightarrow_M (skip{*}s, 011111{*}{*}, 110)\\
&\rightarrow_M (shift, 011111{*}, {*}{*}10)\\
&\rightarrow_M (shift, 011111, {*}{*}{*}10)\\
&\rightarrow_M (shift, 01111, 1{*}{*}{*}10)\\
&\rightarrow_M (append, 011111, {*}{*}{*}10)\\
&\rightarrow_M (skip{*}s, 0111111, {*}{*}10)\\
&\rightarrow_M (skip{*}s, 0111111{*}, {*}10)\\
&\rightarrow_M (skip{*}s, 0111111{*}{*}, 10)\\
&\rightarrow_M (shift, 0111111{*}, {*}{*}0)\\
&\rightarrow_M (shift, 0111111, {*}{*}{*}0)\\
&\rightarrow_M (shift, 011111, 1{*}{*}{*}0)\\
&\rightarrow_M (append, 0111111, {*}{*}{*}0)\\
&\rightarrow_M (skip{*}s, 01111111, {*}{*}0)\\
&\rightarrow_M (skip{*}s, 01111111{*}, {*}0)\\
&\rightarrow_M (skip{*}s, 01111111{*}{*}, 0)\\
&\rightarrow_M (tidyup, 01111111{*}, {*}\sqcup)\\
&\rightarrow_M (tidyup, 01111111, {*}\sqcup\sqcup)\\
&\rightarrow_M (tidyup, 0111111, 1\sqcup\sqcup\sqcup)\\
&\rightarrow_M (finalise, 01111111, \sqcup\sqcup\sqcup)\\
&\rightarrow_M (finalise, 0111111, 10\sqcup\sqcup)
\end{aligned}
$$

(b) The machine M takes a non-empty list of n natural numbers $[x_1 \dots x_n]$, and produces a singleton list containing only the sum of the numbers in the input list $[\sum_{i=1}^{n} x_i]$.

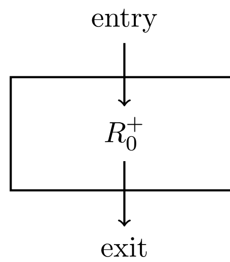
2. Define $M = (Q, \Sigma, s, \delta)$ where $\Sigma = \{\text{␣}, 0, 1\}$, $Q = \{s, t\}$, and δ is given by:

δ	␣	0	1
s	$(t, 1, R)$	$(t, 1, R)$	$(s, 0, R)$
t			

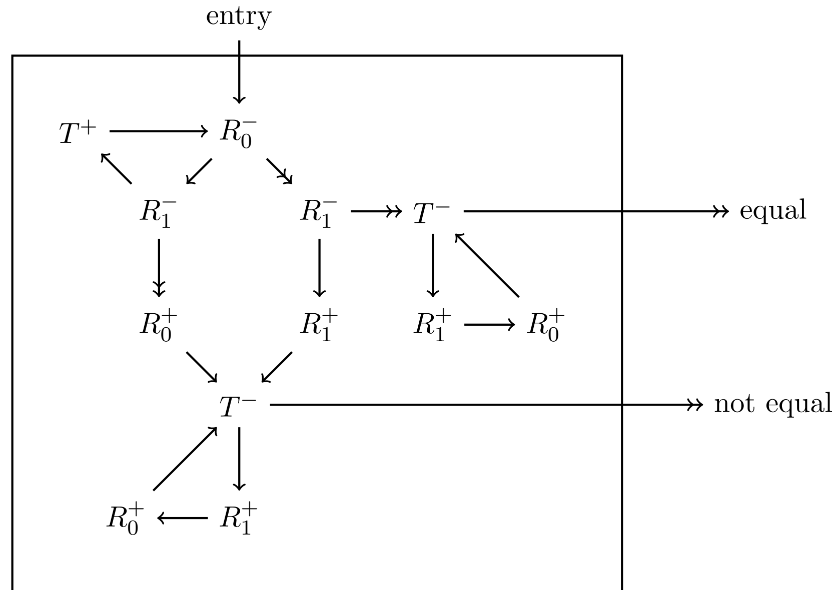
3. (a) To implement the successor machine with Minsky machines, we define gadgets to implement each of the instructions. Clear R_0:



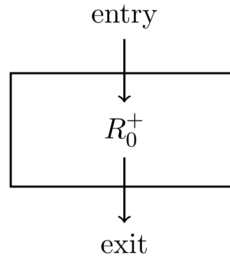
Increment R_0:



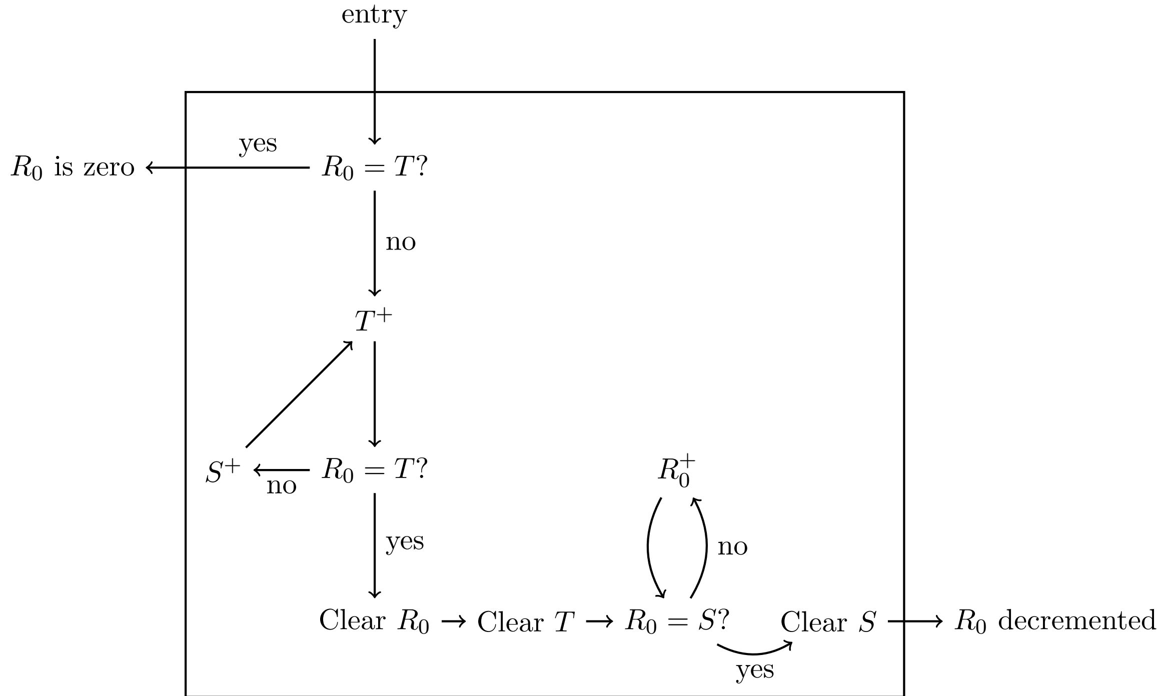
Test $R_0 = R_1$:



Conversely, we can implement the Minsky machine instructions using Successor machine gadgets. Increment R_0:



Test R_0 for zero and decrement:



(b) The Church-Turing thesis is that anything that is algorithmically computable is computable by a Turing machine. Anything that is computable by a Minksy machine is computable by a Turing machine, and hence anything that is computable by a Successor machine is also computable by a Turing machine. The Successor machine can reasonably seen as a formalisation of an algorithm, which could be implemented. If it could compute anything that is not Turing-computable, it would falsify the Church-Turing thesis. The fact that it cannot therefore supports the thesis.

(c) If the halting problem for Successor machines were decidable, so too would be the halting problem for Minsky machines, which we have seen is not decidable. Hence the halting problem is not decidable.

4. (a) Given a register machine M, we know that the partial function of one argument that it computes is also Turing computable (Slide 19). This means that we can construct a Turing machine T that, on the input 0␣0 will halt exactly when M halts with all registers initially 0. It is easy to construct a Turing machine T' that writes 0␣0 onto an empty tape and then starts executing T with the head at the left-most 0. In fact, the function that given the code of M computes the code of the corresponding T' should be computable. (Effectively, such a program is a compiler.)

Suppose that we have a decision procedure[1] for the set of numbers corresponding to the Turing machines which eventually halt when run on the empty input. Then we can use this to decide the halting problem for register machines by combining it with the register-machine-to-Turing-machine compiler above. But the halting problem is undecidable, so there can be no such decision procedure, and the set is undecidable.

(b) The set of satisfiable first-order sentences must be undecidable. If it were decidable, we would have a procedure for deciding the halting problem for Turing machines: first compute the formula corresponding to the Turing machine, then determine if it is satisfiable, and hence whether the Turing machine halts. Since the halting problem is not decidable, there can be no decision procedure for the set of satisfiable first-order sentences.

(c) The set of valid first-order sentences is semidecidable. Consider a machine that, given a sentence of first-order logic (encoded as the number s) starts at $p = 0$ and determines if (s, p) is a valid sentence-proof-pair. If it is, it halts; if not, it increments p and loops, checking the next candidate proof.

If s does have some proof, which must have some number, say p', the machine will eventually halt. If s has no proof, the machine will continue to run forever.

The set of valid first-order sentences is not decidable. A sentence F is satisfiable if and only if the sentence $\neg F$ is not valid. Therefore, if we had a decision procedure for first-order sentences, we would also have a decision procedure for satisfiable sentences (by simply feeding in the complement). From the first part, we know that this is not possible.

5. One problem with the idea of a perfect virus scanner is that the concept of a virus is poorly defined. For instance a program that overwrote your master boot record (MBR) might be considered a virus; however, there are legitimate reasons for overwriting the MBR, so when should a program that overwrites the MBR be considered a virus? However, let's simplify things and make the following assumptions:

- There is some operation such that, if a program performs it then it should be considered a virus.
- A program can run for ever, and use an unbounded amount of memory, without being considered a virus.

Given a register machine, we can write a program that simulates that machine. We make our program perform the illegal operation once the register machine halts. If the register machine never halts, the program should not be considered a virus, since it will never execute the illegal operation. If it does halt, then the program should be considered a virus. A perfect virus scanner would therefore be able to solve the halting problem for register machines, which should be impossible by the Church-Turing thesis.

For a more direct approach, assuming that the virus scanner is not itself a virus(!), we could construct a program P using the virus scanner code V to check if V identifies P as a virus. If it does, then it simply exits normally; otherwise it performs the illegal action. Thus P is a virus if and only if V does not identify it as one, and so V cannot be a perfect virus scanner.

[1] A decision procedure for a set is a register machine (or equivalently a Turing machine) that computes the characteristic function for the set.