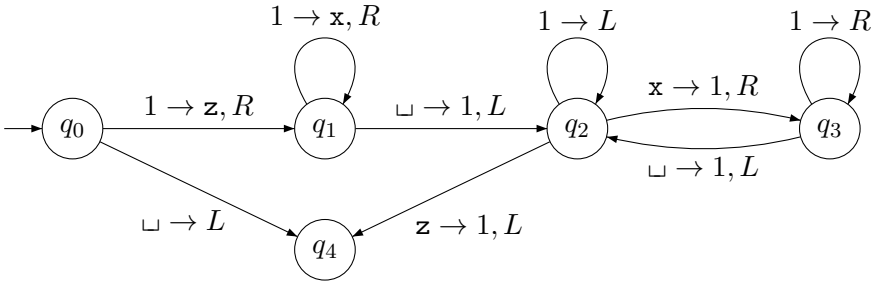
The University of Melbourne
School of Computing and Information Systems
COMP30026 Models of Computation

Selected Tutorial Solutions, Week 12

97. Here is the Turing machine D:



Note that, for the two transitions into q_4, the 'L' has no effect, since, at that point, the tape head is positioned over the leftmost tape cell. The machine doubles the number of ones that it finds on the tape.

98. Here is how the 2-PDA recogniser for B operates:

(a) Push a \$ symbol onto stack 1 and also onto stack 2.
(b) As long as we find an `a` in input, consume it and push an `a` onto stack 1.
(c) As long as we find a `b` in input, consume it and push a `b` onto stack 2.
(d) As long as we find a `c` in input, consume it and pop both stacks.
(e) If the top of each stack has a \$ symbol, pop these.
(f) If we got to this point and the input has been exhausted, accept.

If the 2-PDA got stuck at any point, that meant reject.

99. To simulate M running on input $x_1x_2\cdots x_n$, the 2-PDA P first pushes a \$ symbol onto stack 1 and also onto stack 2. It then runs through its input, pushing $x_1, x_2, \ldots x_{n-1}, x_n$ onto stack 1. It then pops each symbol from stack 1, pushing it to stack 2. That is, it pushes $x_n, x_{n-1}, \ldots x_2, x_1$ onto stack 2, in that order. Note that x_1 is on top.

P is now ready to simulate M. Note that it has consumed all of its input already, but it is not yet in a position to accept or reject.

For each state of M, P has a corresponding state. Assume P is in the state that corresponds to some M state q.

For each M-transition $\delta(q, a) = (r, b, R)$, P has a transition that pops a off stack 2 and pushes b onto stack 1. If stack 2 now has \$ on top, P pushes a blank symbol onto stack 2. Then P goes to the state corresponding to r.

For each M-transition $\delta(q, a) = (r, b, L)$, P has a transition that first pops a off stack 2, replacing it by b. It then pops the top element off stack 1 and transfers it to the top of stack 2, unless it happens to be \$. And then of course P goes to the state corresponding to r.

If this seems mysterious, try it out for a simple Turing machine and draw some diagrams along to way, with snapshots of the Turing machine's tape and tape head next to the 2-PDA's corresponding pair of stacks. The invariant is that what sits on top of the 2-PDA's stack 2 is exactly what is under the Turing machine's tape head at the corresponding point in its computation.

100. Assume that H is a decider for the language

$$\mathit{Halt}_{TM} = \{\langle M, w\rangle \mid M \text{ is a Turing machine and } M \text{ halts when run on input } w\}$$

Here is a decider for A_{TM}:

> On input x:
> if x is not of the required form $\langle M, w\rangle$, reject;
> else run H on $\langle M, w\rangle$;
> if H rejects, reject;
> else run M on w and accept x if M accepts w;
> else reject.

Since H halts for all input, the above Turing machine also halts, and it accepts if and only if M accepts w. Since we know that A_{TM} is undecidable, we conclude that Halt_{TM} is undecidable as well.

101. Yes, $\{\langle G\rangle \mid G \text{ is a context-free grammar over } \{\mathtt{0},\mathtt{1}\} \text{ and } L(G) \cap \mathtt{1}^* \neq \emptyset\}$ is decidable. Since $\mathtt{1}^*$ is regular, the intersection $L(G) \cap \mathtt{1}^*$ is context-free, and we can build a context-free grammar G' for it. We then run the emptiness decider from Lecture 21 on G'. If that decider accepts, we reject, and vice versa.

102. Let A and B be decidable languages, let M_A be a decider for A and M_B a decider for B. We construct a decider for $A \cup B$ as a Turing machine which implements this routine:

> On input w:
> (1) Run M_A on input w and accept if M_A accepts.
> (2) Run M_B on input w and accept if M_B accepts; else reject.

103. Let M_A and M_B be recognisers for A and B, respectively. We want to construct a recogniser for $A \cup B$ but we can't use the construction from the last question. The reason is that there could be some string $w \in B$, which should obviously be accepted by the recogniser we construct, but for which M_A fails to terminate. That is, step (1) above never terminates.

Instead we construct a recogniser for $A \cup B$ by alternating the recognisers for A and B:

> On input w:
> (1) Let M_A take a single execution step; accept if M_A accepts;
> go to step (4) if M_A rejects.
> (2) Let M_B take a single execution step; accept if M_B accepts;
> go to step (5) if M_B rejects.
> (3) Go to step (1).
> (4) Run M_B alone; accept if M_B accepts; reject if it rejects.
> (5) Run M_A alone; accept if M_A accepts; reject if it rejects.

Note that this machine may fail to terminate on w, which will happen if both of M_A and M_B fail to terminate. That's okay, of course, as we are constructing a recogniser, not a decider.

104. This is easy enough. Let M be a decider for A. We get a decider for A^c simply by swapping the 'reject' and 'accept' states in M.

The construction won't work if all we know about M is that it is a recogniser for A. Namely, M may fail to terminate for some string $w \in A^c$.

105. We just show the case for concatenation. Let M_A and M_B be deciders for A and B, respectively. We want to construct a decider for $A \circ B$. It will make our task easier if we utilise nondeterminism. We can construct a nondeterministic Turing machine to implement this routine:

> On input w:
> (1) Split w nondeterministically so that $w = xy$.
> (2) Run M_A on x; reject w if M_A rejects x.
> (2) Run M_B on y; reject w if M_B rejects y.
> (3) Otherwise accept w.

This makes good use of the nondeterministic Turing machine's bias towards acceptance.

106. Assume $\mathcal{B}$ is countable. Then we can enumerate $\mathcal{B}$:

Element								
b_1	$\boxed{0}$	0	1	0	1	1	1	...
b_2	1	$\boxed{0}$	1	1	1	0	1	...
b_3	1	0	$\boxed{1}$	1	1	0	1	...
b_4	0	1	0	$\boxed{0}$	1	0	0	...
$\vdots$								

However, the infinite sequence which has

$$i\text{'th bit} = \begin{cases} 0 & \text{if the } i\text{th bit of } b_i \text{ is 1} \\ 1 & \text{if the } i\text{th bit of } b_i \text{ is 0} \end{cases}$$

is different from each of the b_i. Hence no enumeration can exist, and $\mathcal{B}$ is uncountable. This should not be surprising, because the set $\mathcal{B}$ is really the same as (or is isomorphic to) $\mathbb{N} \to \Sigma$.