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

Selected Problem Set Solutions, Week 11

P11.1 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 $w \in A^c$.

P11.2 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 t ards acceptance.

P11.3 Here is how the 2-PDA rec

(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 pu k 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.

P11.4 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.