Homework problems

P10.1 Show that the class of decidable languages is closed under complement. Why can't we use the same argument to show that the class of Turing recognisable languages is closed under complement?

Solution: 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 $w \in A^c$.

P10.2 Show that the class of decidable languages is closed under concatenation.

Solution: 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.

P10.3 Show that the class of decidable languages is closed under Kleene star.

P10.4 A 2-PDA is a pushdown automaton that has two stacks instead of one. In each transition step it may consume an input symbol, pop and/or push to stack 1, and pop an/or push to stack 2. It can also leave out any of these options (using ϵ moves) just like the standard PDA.

In the lectures, we used the pumping lemma for context-free languages to established that the language $B = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \in \mathbb{N}\}$ is not context-free. However, B has a 2-PDA that recognises it. Outline in English or pseudo-code how that 2-PDA operates.

Solution: 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.

P10.5 In fact, a 2-PDA is as powerful as a Turing machine. Outline an argument for this proposition by showing how a 2-PDA can simulate a given Turing machine. *Hint:* arrange things so that, at any point during simulation, the two stacks together hold the contents of the Turing machine's tape, and the symbol under the tape head sits on top of one of the stacks.

Solution: 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.

P10.6 For each of the following languages, write an algorithm in pseudocode which describes a Turing machine that decides the following languages. Assume that $\Sigma = \{\texttt{0}, \texttt{1}\}$

(i) $\{w \mid w$ has an equal number of `0`'s and `1`'s$\}$

(ii) $\{w \mid w$ has twice as many `0`'s as `1`'s$\}$

(iii) $\{w \mid w$ does not have twice as many `0`'s as `1`'s$\}$

P10.7 For each language in P10.6, draw a Turing machine that carries out the pseudocode. Then write down the sequence of configurations your TM goes through on an interesting input.

P10.8 Show that the problem of whether the language of a DFA is empty, is decidable. That is, show that the language

$$E_{DFA} = \{\langle D \rangle \mid D \text{ is a DFA and } L(D) = \emptyset\}$$

is decidable. *Hint:* write pseudocode for an algorithm which analyses the graph of the DFA, and argue that your algorithm will not run forever on any input DFA $\langle D \rangle$.

P10.9 Show that the problem of whether the language of a CFG is empty, is decidable. That is, show that the language

$$E_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$$

is decidable. *Hint:* write pseudocode for an algorithm which analyses the rules of the CFG, and argue that your algorithm will not run forever on any input CFG $\langle G \rangle$.