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

Selected Tutorial Solutions, Week 9

P9.1 (i) $\{w \mid w$ has length at least 3 and its third symbol is `0`$\}$: $(0 \cup 1)(0 \cup 1)0(0 \cup 1)^*$

(ii) $\{w \mid$ every odd position of w is a `1`$\}$: $(1(0 \cup 1))^*(\epsilon \cup 1)$

(iii) $\{w \mid w$ contains at least two `0`s and at most one `1`$\}$: $0^*(00 \cup 001 \cup 010 \cup 100)0^*$

(iv) $\{\epsilon, \mathtt{0}\}$: $\epsilon \cup 0$

(v) The empty set: $\emptyset$

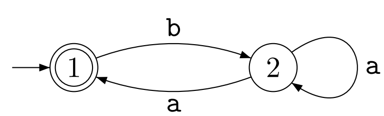
P9.2 If A is regular then *suffix*(A) is regular. Namely, let $D = (Q, \Sigma, \delta, q_0, F)$ be a DFA for A. Assume every state in Q is *reachable* from q_0. Then we can turn D into an NFA N for *suffix*(A) by adding a new state q_{-1} which becomes the NFA's start state. For each state $q \in Q$, we add an epsilon transition from q_{-1} to q.

That is, we define N to be $(Q \cup \{q_{-1}\}, \Sigma, \delta', q_{-1}, F)$, with transition function

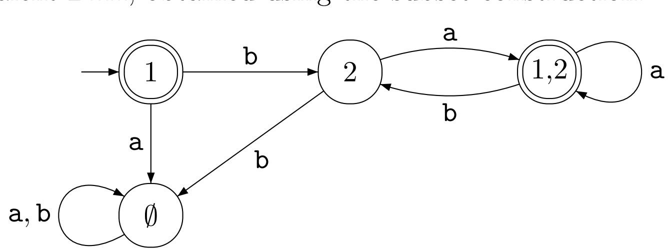
$$\delta'(q, x) = \begin{cases} \{\delta(q, x)\} & \text{for } q \in Q \text{ and } x \in \Sigma \\ Q & \text{for } q = q_{-1} \text{ and } x = \epsilon \\ \emptyset & \text{for } q \neq q_{-1} \text{ and } x = \epsilon \end{cases}$$

The restriction we assumed, that all of D's states are reachable, is not a severe one. It is easy to identify unreachable states and eliminate them (which of course does not change the language of the DFA). To see why we need to eliminate unreachable states before generating N in the suggested way, consider what happens to this DFA for $\{\epsilon\}$: $(\{q_0, q_1, q_2\}, \{\mathtt{a}\}, \delta, q_0, \{q_0\})$, where $\delta(q_0, \mathtt{a}) = \delta(q_1, \mathtt{a}) = q_1$ and $\delta(q_2, \mathtt{a}) = q_0$.

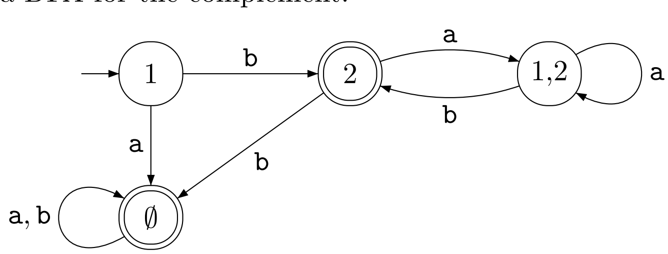
P9.3 (a) Here is an NFA for $(\mathtt{ba}^*\mathtt{a})^*$:



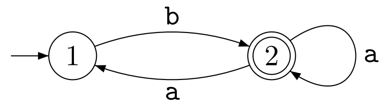
(b) Here is an equivalent DFA, obtained using the subset construction:



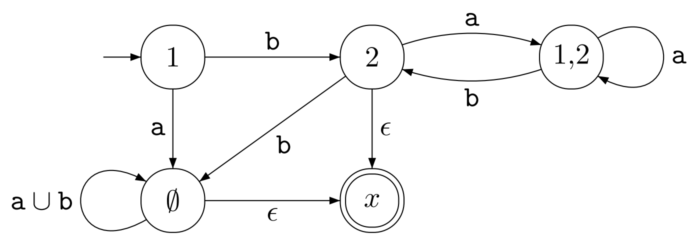
(c) It is easy to get a DFA for the complement:



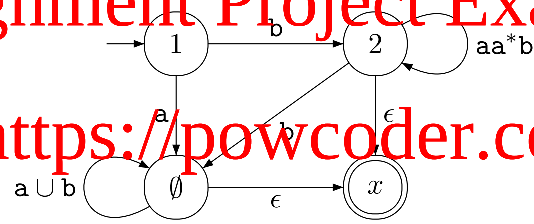
(d) It would be problematic to do the "complement trick" on the NFA, as it is only guaranteed to work on DFAs. We would get the following, which accepts, for example, `baa`:



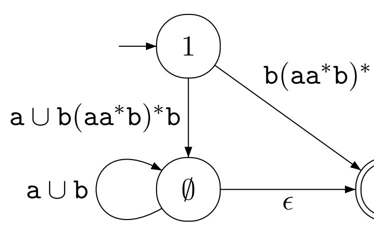
(e) Starting from the DFA that we found in (c), we make sure that we have just one accept state:



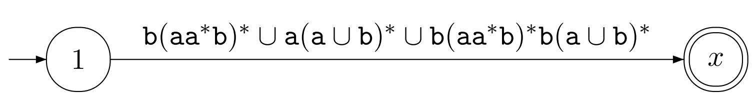
Let us first remove the state labeled 1,2:



Now we can remove state 2:



Finally, eliminating the state labelled $\emptyset$, we are left with:



The resulting regular expression can be read from that diagram.

P9.4 (a) We use the pumping lemma to show that B is not regular. Assume that it were. Let p be the pumping length, and consider $\mathtt{a}^{p+1}\mathtt{b}\mathtt{a}^p$. This string is in B and of length $2p+2$. By the pumping lemma, there are strings x, y, and z such that $\mathtt{a}^{p+1}\mathtt{b}\mathtt{a}^p = xyz$, with $y \neq \epsilon$, $|xy| \leq p$, and $xy^iz \in B$ for all $i \in \mathbb{N}$. By the first two conditions, y must be a non-empty string consisting of `a`s only. But then, pumping down, we get xz, in which the number of `a`s on the left no longer is strictly larger than the number of `a`s on the right. Hence we have a contradiction, and we conclude that B is not regular.

(b) We want to show that $C = \{w \in \{\mathtt{a},\mathtt{b}\}^* \mid w \text{ is not a palindrome}\}$ is not regular. But since regular languages are closed under complement, it will suffice to show that $C^c = \{w \in \{\mathtt{a},\mathtt{b}\}^* \mid w \text{ is a palindrome}\}$ is not regular. Assume that C^c *is* regular, and let p be the pumping length. Consider $a^p b a^p \in C^c$. Since $|s| \geq p$, by the pumping lemma, s can be written $s = xyz$, so that $y \neq \epsilon$, $|xy| \leq p$, and $xy^iz \in C^c$ for all $i \geq 0$. But since $|xy| \leq p$, y must contain as only. Hence $xz \notin C^c$. We have a contradiction, and we conclude that C^c is not regular. Now if C was regular, C^c would be regular too. Hence C cannot be regular.

P9.5 Here are the context-free grammars:

(a) $\{w \mid w$ starts and ends with the same symbol$\}$:

$$\begin{array}{rcl} S & \rightarrow & 0\,T\,0 \mid 1\,T\,1 \mid 0 \mid 1 \\ T & \rightarrow & 0\,T \mid 1\,T \mid \epsilon \end{array}$$

(b) $\{w \mid$ the length of w is odd$\}$:

$$S \rightarrow 0 \mid 1 \mid 0\,0\,S \mid 0\,1\,S \mid 1\,0\,S \mid 1\,1\,S$$

P9.6 Here is a context-free grammar for $\{\mathtt{a}^i\mathtt{b}\mathtt{a}^j \mid i > j \geq 0\}$:

$$\begin{array}{rcl} S & \rightarrow & A\ B \\ A & \rightarrow & \mathtt{a} \mid \mathtt{a}\ A \\ B & \rightarrow & \mathtt{b} \mid \mathtt{a}\ B\ \mathtt{a} \end{array}$$

P9.7 See Tutorial Sheet 10 solutions

P9.8 Here are some sentences generated from the grammar:

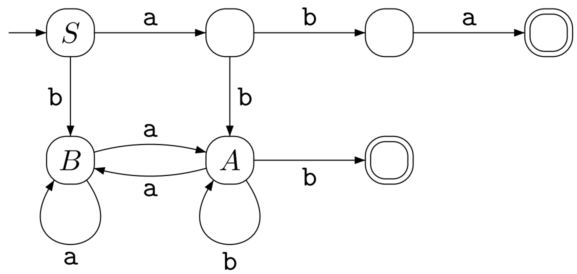
(a) A dog runs
(b) A dog likes a bone
(c) The quick dog chases the lazy cat
(d) A lazy bone chases a cat
(e) The lazy cat hides
(f) The lazy cat hides a bone

The grammar is concerned with the structure of well-formed sentences; it says nothing about meaning. A sentence such as "a lazy bone chases a cat" is syntactically correct—its structure makes sense; it could even be semantically correct, for example, "lazy bone" may be a derogatory characterisation of some person. But in general there is no guarantee that a well-formed sentence carries meaning.

P9.9 We can easily extend the grammar so that a sentence may end with an optional adverbial modifier:

$$\begin{array}{rcl} S & \rightarrow & NP\ \ VP\ \ PP \\ & \vdots & \\ PP & \rightarrow & \epsilon \\ PP & \rightarrow & \text{quietly} \\ PP & \rightarrow & \text{all day} \\ & \vdots & \end{array}$$

P9.10 Here is a suitable NFA:



P9.11 Let w be a string in $L(G)$, that is, w is derived from S. We will use structural induction to show a stronger statement than what was required; namely we show that, for every string $w \in L(G)$, w starts with neither `b` nor `abb`. That is, if w is derived from S then it starts with neither `b` nor `abb`. (To express the property formally, we may write $\forall w' \in \{\texttt{a}, \texttt{b}\}^*(w \neq \texttt{b}w' \wedge w \neq \texttt{abb}w')$.

There is one base case: If $w = \texttt{ab}$ then w does not start `b` and it does not start with `abb`.

For the first recursive case, let $w = \texttt{a}w'\texttt{b}$, where $w' \in L(G)$. By the induction assumption, w' starts with neither `b` nor `abb`. Hence, in this case, w does not start with `b` (because it starts with `a`), and it does not start with `abb` (because w' does not start with `b`).

For the second recursive case, let $w = w'w''$, with $w', w'' \in L(G)$. By the induction assumption, w' starts with neither `b` nor `abb` (similarly for w''). Let us do case analysis on the length of w'.

- $|w'| = 0$: If $w' = \epsilon$ then $w = w''$ which starts with neither `b` nor `abb`, by assumption.
- $|w'| = 1$: In this case we must have $w' = \texttt{a}$ since, by assumption, w' does not start with `b`. But then w doesn't start with `b`, and it doesn't start with `abb` either, because w'' does not start with `b`, by assumption.
- $|w'| \geq 2$: In this case w' must start with either `aa` or `ab`. That means w does not start with `b`. And w can only start with `abb` if $w' = \texttt{ab}$ and w'' starts with `b`. But the latter is impossible, by assumption.

Hence in no case does w start with `abb`.

P9.12 Too easy.

P9.13 Here is a context-free grammar that will do the job (S is the start symbol):

$$
\begin{array}{lcl}
S & \to & \epsilon \mid \texttt{a}\ A \\
A & \to & \texttt{a}\ A \mid \texttt{b}\ B \\
B & \to & \epsilon \mid \texttt{a}\ A \mid \texttt{b}\ B
\end{array}
$$