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

Selected Tutorial Solutions, Week 10

73. (a) We show that $A = \{0^n 1^n 2^n \mid n \geq 0\}$ is not regular. Assume it is, and let p be the pumping length. Consider $s = 0^p 1^p 2^p \in A$. Since $|s| \geq p$, by the pumping lemma, s can be written $s = xyz$, so that $y \neq \epsilon$, $|xy| \leq p$, and $xy^i z \in A$ for all $i \geq 0$. But since $|xy| \leq p$, y must contain 0s only. Hence $xz \notin A$. Thus we have a contradiction, and we conclude that A is not regular.

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

    (c) 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 $xy^i z \in C^c$ 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.

74. Here are the context-free grammars:

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

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

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

$$S \to 0 \mid 1 \mid 0\,0\,S \mid 0\,1\,S \mid 1\,0\,S \mid 1\,1\,S$$

    (c) $\{w \mid$ the length of w is odd and its middle symbol is $\mathtt{0}\}$:

$$S \to 0 \mid 0\,S\,0 \mid 0\,S\,1 \mid 1\,S\,0 \mid 1\,S\,1$$

    (d) $\{w \mid w$ is a palindrome$\}$:

$$S \to 0\,S\,0 \mid 1\,S\,1 \mid 0 \mid 1 \mid \epsilon$$

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

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

76. The class of context-free languages is closed under the regular operations: union, concatenation, and Kleene star.

    Let G_1 and G_2 be context-free grammars generating L_1 and L_2, respectively. First, if necessary, rename variables in G_2 so that the two grammars have no variables in common. Let the start variables of G_1 and G_2 be S_1 and S_2, respectively. Then we get a context-free grammar for $L_1 \cup L_2$ by keeping the rules from G_1 and G_2, adding

$$\begin{array}{lcl} S & \rightarrow & S_1 \\ S & \rightarrow & S_2 \end{array}$$

    where S is a fresh variable, and making S the new start variable.

    We can do exactly the same sort of thing for $L_1 \circ L_2$. The only difference is that we now just add one rule:

$$S \rightarrow S_1\ S_2$$

    again making (the fresh) S the new start variable.

    Let G be a context-free grammar for L and let S be fresh. If we add two rules to those from G:

$$\begin{array}{lcl} S & \rightarrow & \epsilon \\ S & \rightarrow & S\ S' \end{array}$$

    where S' is G's start variable, then we have a context-free grammar for L^* (it has the fresh S as its start variable).

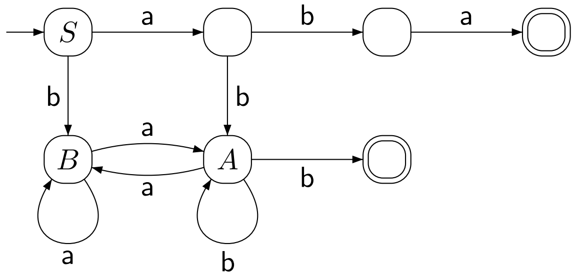
77. Here are some sentences generated from the grammar:

    (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.

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

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

79. Here is a suitable NFA: 

80. 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 $\forall w' \in \{\mathtt{a}, \mathtt{b}\}^*(w \neq \mathtt{b}w' \land w \neq \mathtt{abb}w')$.

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

For the first recursive case, let $w = \mathtt{a}w'\mathtt{b}$, 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 $w' = \mathtt{a}$ 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 $w' = \mathtt{ab}$ and w'' starts with `b`. But the latter is impossible, by assumption.

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

81. The induction hypothesis, for structural induction, is this:

$$s_1\ t + n = s\ n\ t \quad \text{for all } n \tag{1}$$

There are two base cases, namely $t = \mathtt{Void}$ and $t = \mathtt{Node}\ x\ \mathtt{Void}\ \mathtt{Void}$. In the first case, $s_1\ t + n = 0 + n = n = s\ n\ t$. In the second case, $s_1\ t + n = 1 + n = n + 1 = s\ n\ t$.

For the recursive case, let $t = \mathtt{Node}\ x\ \mathit{left}\ \mathit{right}$ and assume *left* and *right* satisfy the induction hypothesis, that is,

$$s_1\ \mathit{left} + n = s\ n\ \mathit{left} \quad \text{for all } n \tag{2}$$

$$s_1\ \mathit{right} + n = s\ n\ \mathit{right} \quad \text{for all } n \tag{3}$$

In this case,

$$\begin{array}{rcll} s_1\ t + n & = & s_1\ \mathit{left} + s_1\ \mathit{right} + n & \text{by the definition of } s_1 \\ & = & s_1\ \mathit{right} + s_1\ \mathit{left} + n & \text{by rearrangement} \\ & = & s\ (s_1\ \mathit{left} + n)\ \mathit{right} & \text{by (3)} \\ & = & s\ (s\ n\ \mathit{left})\ \mathit{right} & \text{by (2)} \\ & = & s\ n\ t & \text{by the definition of } s \end{array}$$

Hence we have established that (1) holds for all binary trees t. In particular, for all binary trees t we have: $s_1\ t = s_1\ t + 0 = s\ 0\ t = s_2\ t$.

82. Too easy.

83. The grammar is ambiguous because ab can be derived from A and also from B. However, it is the only string that can be derived from both, so we can make this grammar unambiguous simply by making sure that ab cannot be derived from A, or more precisely, making sure that the set of strings that can be derived from A is $\{\mathsf{a}^n\mathsf{b}^n \mid n > 1\}$. To do this, change the first rule for A like so:

$$\begin{array}{lcl} T & \rightarrow & A \mid B \\ A & \rightarrow & \mathsf{a\ a\ b\ b} \mid \mathsf{a}\ A\ \mathsf{b} \\ B & \rightarrow & \epsilon \mid \mathsf{a\ b}\ B \end{array}$$

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

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