Aalto University
Department of Computer Science

CS-C2160 Theory of Computation
Solved Example Problems, Spring 2022

1 Finite Automata and Regular Languages

1. **Problem:**

   Describe the following languages **both** in terms of regular expressions **and** in terms of deterministic finite automata:

   (a) $\{w \in \{0,1\}^* \mid w \text{ contains 101 as a substring}\}$,

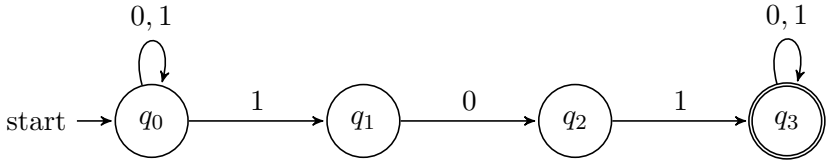
   (b) $\{w \in \{0,1\}^* \mid w \text{ does not contain 101 as a substring}\}$.

   Solution:

   (a) A regular expression for this language is easy to construct:

   $$(0|1)^*101(0|1)^*.$$

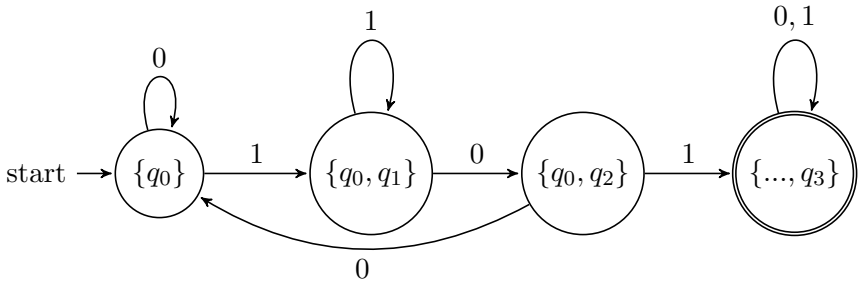
   A nondeterministic finite automaton is also easy to come up with:



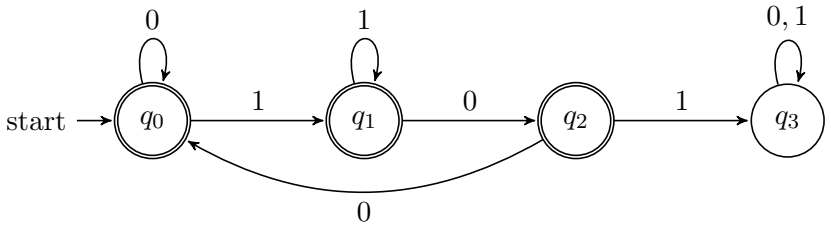
   Let us then determinise this using the subset construction:

		0	1
$\rightarrow$	$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
	$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_1\}$
	$\{q_0, q_2\}$	$\{q_0\}$	$\{q_0, q_1, q_3\}$
$\leftarrow$	$\{\ldots, q_3\}$	$\{\ldots, q_3\}$	$\{\ldots, q_3\}$

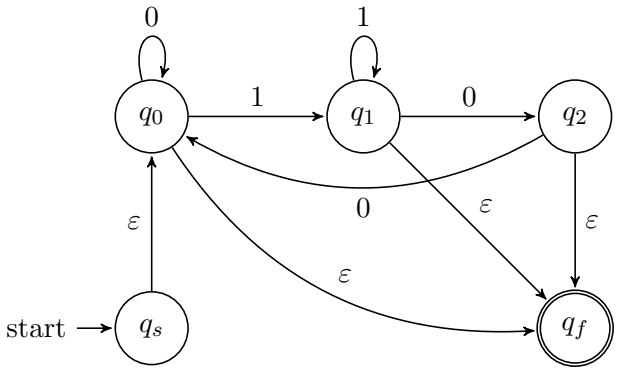
   The last row of the table has been simplified based on the observation that from a set containing the accepting state q_3, one always moves again to some set containing q_3, and so all such states are equivalent. Thus, we obtain the following deterministic automaton:



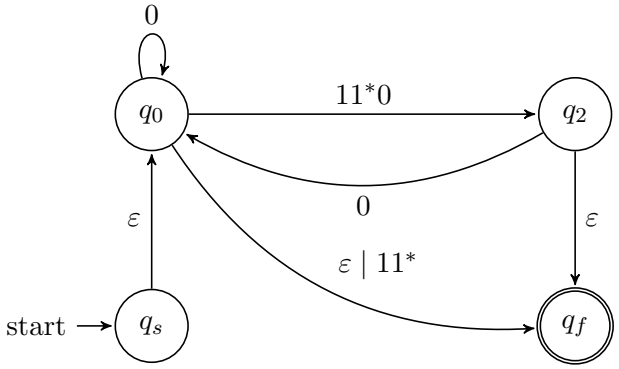
(b) Observe that the language here is the complement of the language in part (a), and the DFA provided in part (a) is complete, i.e. all possible transitions are explicitly listed. Therefore, complementing the DFA from part (a) yields a DFA for this language:



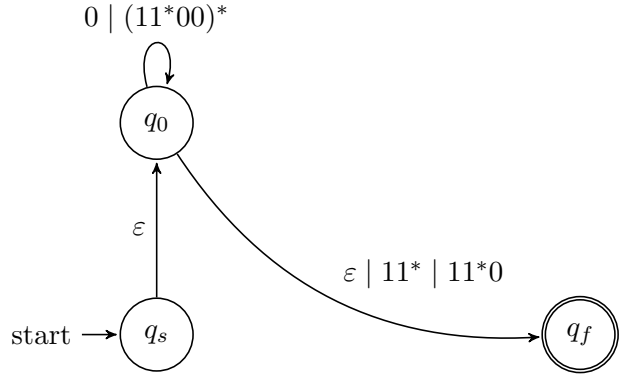
All accepting states are inaccessible from state q_3, thus we may ignore it. To find a corresponding regular expression, we first add a new initial and final state and their connecting ε-transitions:



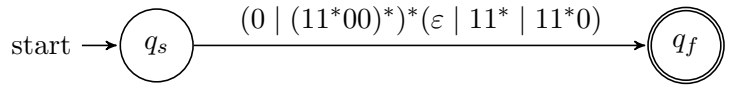
Remove state q_1 :



Remove state q_2:



Remove state q_0:



From the above, we may read off a regular expression describing the language:

$$(0 \mid (11^*00)^*)^*(\varepsilon \mid 11^* \mid 11^*0).$$

2. **Problem:**

Describe the following languages **both** in terms of regular expressions **and** in terms of deterministic finite automata:

(a) $\{w \in \{0,1\}^* \mid w$ contains an even number (possibly zero) of 1's$\}$

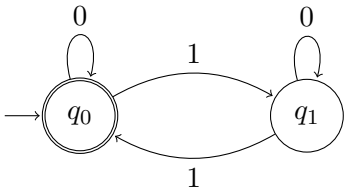
(b) $\{w \in \{0,1\}^* \mid w$ contains an odd number of 1's$\}$

(c) $\{wb \in \{0,1\}^* \mid$ either w contains an even number (possibly zero) of 1's and $b = 0$, or w contains an odd number of 1's and $b = 1\}$.

(*Hint:* In part (c) it may, depending on your solution method, be useful to first design a nondeterministic automaton.)

Solution:

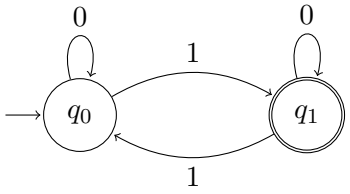
(a) Language is recognised by the deterministic finite automaton



and described by the following regular expression

$$(0^*10^*1)^*0^*$$

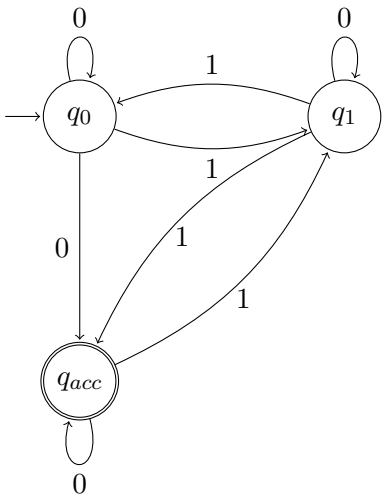
(b) This language is the complement of the one considered in part (a). We can use a lemma derived earlier in the course: the complement of a regular language can be arrived at by constructing a new DFA for which one simply changes the rejecting states of the original DFA to accepting states and vice versa.



We modify the regular expression of part (a) accordingly:

$$0^*1(0^*10^*1)^*0^*$$

(c) We use the DFA introduced in part (a) as a starting point, and add new complementary non-deterministic transitions from q_0 and q_1 to a new unique accepting state q_{acc}. However we also need to add transitions for the situations where the DFA have already transitioned to state q_{acc} but we still have more alphabet-symbols left in the string. As a result we get the following **NFA**:



We determinise the NFA by applying the subset construction for which the space of possible resulting states is the powerset of the number of original states in the NFA. In our case this yields a space of $2^3 = 8$ states for the resulting DFA.

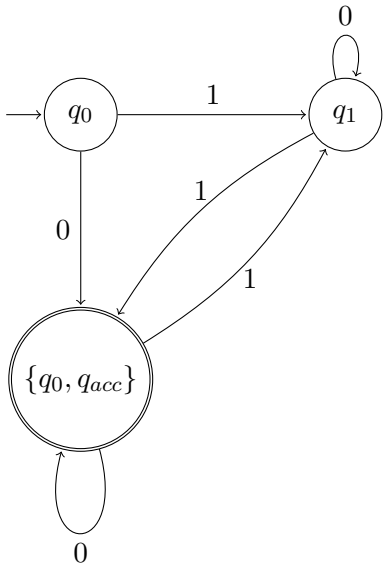
However when we derive new transitions for the DFA, we can simplify the procedure by only including those states that can be eventually reached from the starting state q_0. Transitions for the new transition function $\hat{\delta}$ are thus:

		0	1
$\rightarrow$	$\{q_0\}$	$\{q_0, q_{acc}\}$	$\{q_1\}$
	$\{q_1\}$	$\{q_1\}$	$\{q_0, q_{acc}\}$
$\leftarrow$	$\{q_0, q_{acc}\}$	$\{q_0, q_{acc}\}$	$\{q_1\}$

The resulting set of states in the determinised DFA is then

$$\hat{Q} = \{q_0, q_1, \{q_0, q_{acc}\}\}$$

Resulting **DFA**:



First we concatenate the parity bit at the end of the regular expressions from parts (a) and (b). Then by using the knowledge that regular expressions are closed under union, we derive the whole regular expression as a union of two parts:

$$(0^*10^*1)^*0^*0 \mid 0^*1(0^*10^*1)^*0^*1$$

3. **Problem:** Design deterministic finite automata accepting each of the following languages:

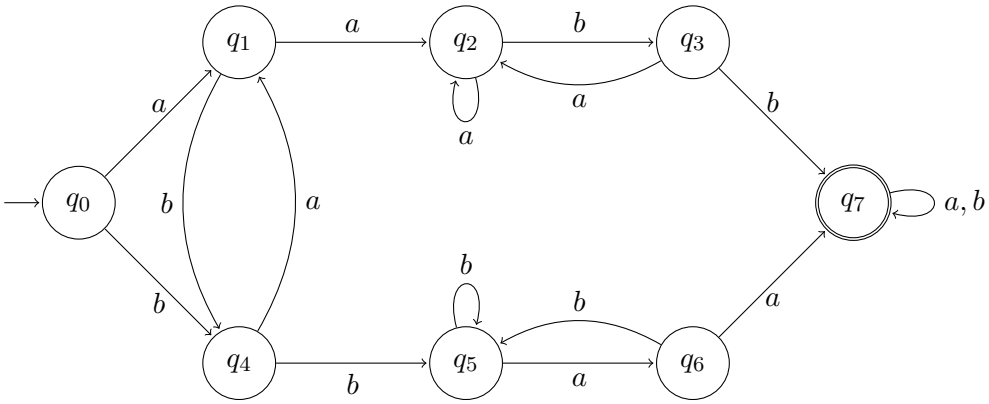
   (a) $\{w \in \{a, b\}^* \mid w$ contains both aa and bb as substrings$\}$;

   (b) $\{w \in \{a, b\}^* \mid w$ does not contain the substring $aba\}$;

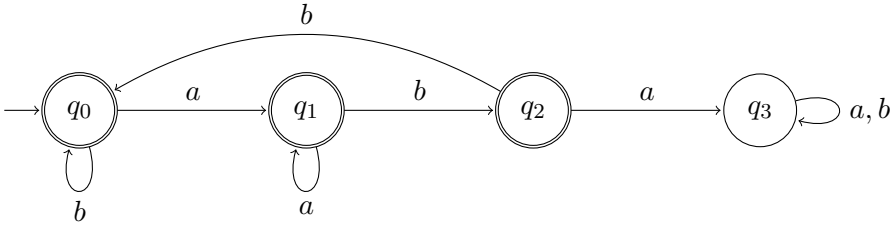
   (c) $\{w \in \{a, b\}^* \mid w$ contains a number of a's that is an exact multiple of three$\}$.

Solution:

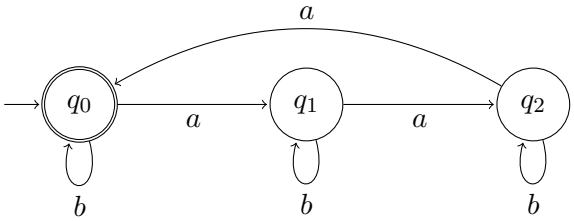
(a) The substrings can come in any order, so we need to consider two cases. The upper part accepts strings where 'aa' is before 'bb', and the lower part accepts ones where 'bb' is before 'aa'.



(b) If we find the substring "aba", we go to the error state q_3. All other states are accept states.



(c) Here state q_i has a meaning: "There have been i mod 3 a's so far". Whenever we are in state q_0, the number of a's so far is a multiple of three.



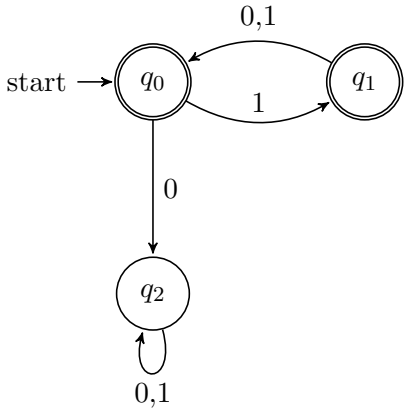
4. **Problem:** Let

$$L = \{w \mid w = a_1 a_2 \dots a_n, n \geq 0, a_i \in \{0, 1\}, a_1 = a_3 = a_5 = \dots = 1\}.$$

In other words, L consists of those binary strings that have 1's in odd-numbered positions. Show that L is regular.

Solution: By definition, a language is regular if there exists a finite automaton that recognises it. Let's try to construct such automaton. In our language, every symbol in an odd-numbered position must be 1, and even-numbered positions can take any symbol.

Let's keep track of the position with two states: q_0 for even-numbered and q_1 for odd-numbered. Both of these states can be accept states if we make sure that the transitions between them follow the rules of our language, especially that transitions to state q_1 are labelled with a 1. If we encounter a rule-breaking input, we enter our third state, q_2, which is not an accept state. All transitions from q_2 point to itself.



This finite automaton now recognises our language L, and we can say that L is regular.

Another approach could be to construct a regular expression that describes the language. Such a regular expression would consist of pairs of symbols, where first input symbol of the pair is always 1 (odd-numbered positions) and the second can be either 0 or 1. If the length of the string is odd, we end the string with symbol 1. With strings of even length we don't need to add anything. A regular expression fulfilling these conditions is: $(1(0 \cup 1))^*(1 \cup \varepsilon)$.

5. **Problem:**

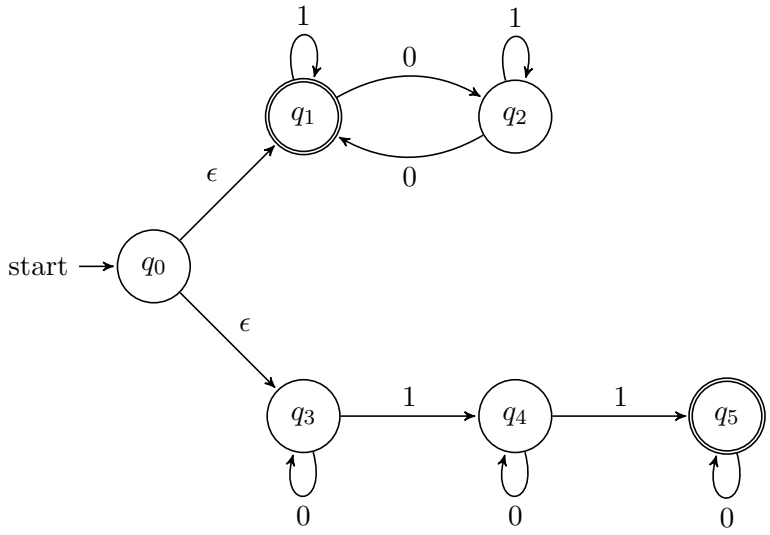
Let

$$L = \{w \in \{0,1\}^* \mid w \text{ contains an even number of 0's or exactly two 1's.}\}.$$

Show that L is regular.

Solution: We can show a language L is regular by designing a corresponding DFA (NFA as well, because we can convert an NFA to a DFA) or regular expression for it. Therefore, let's design the corresponding automaton for L.

We want L to have either (i) an even number of 0's or (ii) exactly two 1's. Therefore, we introduce a choice-making an ϵ-transition from the starting state to disjoint subparts of the automaton that are fairly easy to design. If we choose to have a string according to (i), we are immediately in a final state (q_1) and will come back there after every two zeros. On the other hand, to generate a string according to (ii), after generating two 1's we will be in a final state (and there is no continuation on a 1 from there, because we need exactly two 1's).



It is also quite easy to design a corresponding regular expression, which is $((1^*01^*0)^* \cup (0^*10^*10^*))$.

II Context-Free Grammars and Languages

1. **Problem:**

   (a) Design a context-free grammar for the language

   $$L = \{a^n b^m \mid n \geq 0 \text{ and } m = n \text{ or } m = 2n\}.$$

   Draw the corresponding parse tree for the sentence *aabbbb*.

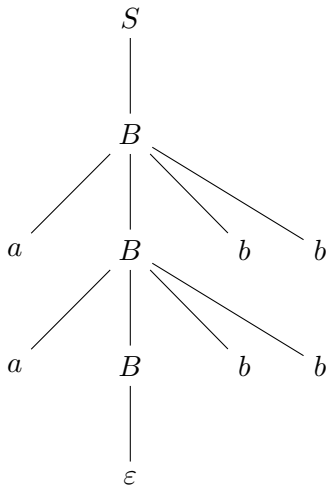
   (b) Prove (precisely!) that the language L in part (a) is not regular.

   Solution:

   (a) We can represent case $m = n$ with variable A and case $m = 2n$ with variable B. The grammar is then:

   $$\begin{aligned} S &\rightarrow A \mid B \\ A &\rightarrow aAb \mid \varepsilon \\ B &\rightarrow aBbb \mid \varepsilon \end{aligned}$$

   The parse tree for the sentence *aabbbb* is:



   (b) Suppose that L is regular. The pumping lemma says that for some $p \geq 1$ the following holds: if $x \in L, |x| \geq p$, then x can be divided into three parts $x = uvw$ so that $|uv| \leq p$, $|v| \geq 1$, and $uv^n w \in L$ for all $n \geq 0$.

   Take the string $x = a^p b^{2p}$. Clearly $|x| \geq p$, so we know that the three parts must be $u = a^i$, $v = a^j$ and $w = a^{p-i-j} b^p$ for some $i + j \leq p$, $i \geq 0$, $j \geq 1$. Now by the pumping lemma we should have $uv^0 w \in L$. But

   $$uv^0 w = uw = a^i a^{p-i-j} b^{2p} = a^{p-j} b^{2p}$$

   Which is not in L, because $j \geq 1$ and so $2(p - j) \neq 2p$. This is a contradiction, so L cannot be regular.

2. **Problem:**

   (a) Design context-free grammars for the languages $L_{\leq} = \{a^i b^j \mid 0 \leq i \leq j\}$ and $L_{\neq} = \{a^i b^j \mid i \neq j\}$. (*Hint:* Note that $i \neq j$ if and only if $i < j$ or $i > j$.)

(b) Prove (precisely!) that the language $L_{\leq}$ in part (a) is not regular.

(c) Prove (precisely!) that the language $L_{\neq}$ in part (a) is not regular.

Solution:

(a) The language $L_{\leq}$ consists of strings where the number of a's is at most number of b's, and the symbols are in alphabetical order. Grammar:

$$S \rightarrow aSb \mid Sb \mid \varepsilon$$

The first production allows us to add any number of a's we want, but ensures that we then have equally many b's. The second production is for adding extra b's.

The language $L_{\neq}$ consists of strings with unequal numbers of a's and b's, again in alphabetical order. As it is hinted, we can do this by considering strings where $i < j$ or $i > j$. Grammar:

$$\begin{aligned} S &\rightarrow A \mid B \\ A &\rightarrow aAb \mid aA \mid a \\ B &\rightarrow aBb \mid Bb \mid b \end{aligned}$$

The production $S \rightarrow A$ describes situations where $i > j$, ie. we have more a's. $S \rightarrow B$ on the other hand is for strings with more b's. The productions for variables A and B are constructed in a similar manner as with $L_{\leq}$, but now we replace the ε-production with terminal a or b to make sure that we don't have an equal number of a's and b's.

(b) For a regular language we have the pumping lemma: there exists some $p \in \mathbb{N}$ such that all strings of length p or longer have a partition $x = uvw$, where $|uv| \leq p$, $|v| \geq 1$ and $uv^iw \in L$, $\forall i \in \mathbb{N}$.

Let's start our proof by assuming that $L_{\leq}$ is regular. Our goal is to find a contradiction to this statement. Now, after our assumption there should be some $p \in \mathbb{N}$, and we are able to choose a string from the language, for which the pumping lemma holds. Let's choose $x = a^pb^p$. Now as $|x| = 2p \geq p$, we have some partition $x = uvw$. Because we have the requirement that $|uv| \leq p$, uv must contain only a's: $uv = a^n$, where $n \leq p$. Let $v = m \geq 1$. Our partition is now $u = a^{n-m}$, $v = a^m$ and $w = a^{p-n}b^p$.

Now uv^iw should belong to the language with all i's. Let's choose $i = 2$. Now

$$uv^2w = a^{n-m}a^{2m}a^{p-n}b^p = a^{n-m+2m+p-n}b^p = a^{p+m}b^p$$

As $m \geq 1$, $p + m > p$, and so $uv^2w \notin L_{\leq}$. This is our contradiction, and $L_{\leq}$ is not regular.

(c) Let's again start by assuming that $L_{\neq}$ is regular. As the regular languages are closed under complement, ie. the complement language of a regular language is also regular, language

$$L_{\neq}^c = L_{=} = \{a^ib^j \mid i = j\}$$

must be regular, too. This can be shown for example by constructing a deterministic finite automaton M for one language, and then creating another one M' for the

complement language by changing the accept states into non-accept states and vice versa: $F' = \{q \in Q \mid q \notin F\}$. Now in M' we accept all the strings that are not accepted by M, and reject all that are accepted by M.

Now, as $L_=$ is regular, the pumping lemma should hold. We can choose the same string as in part (b), $x = a^p b^p \in L_=$, and notice that as $L_= \subset L_\leq$, we have $uv^2w \notin L_\leq \Rightarrow uv^2w \notin L_=$. This is our contradiction: $L_=$ is not regular as the pumping lemma doesn't hold, so $L_\neq$ cannot be regular either.

3. **Problem:**

(a) Design a context-free grammar for the language

$$L = \{ucvcw \mid u, v, w \in \{0,1\}^*,\ v = u^R \text{ or } v = w^R \text{ (or both)}\}.$$

(Notation x^R denotes the reverse of string x, i.e. string x written backwards.)

(b) Show that the grammar you gave in part (a) is ambiguous.

(c) Prove (precisely!) that the language in part (a) is not regular. (*Hint:* Consider e.g. strings of the form $1^n c 1^n c 0^n$.)

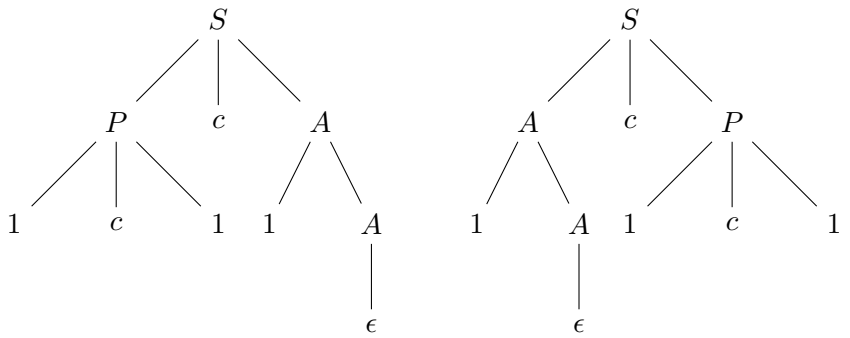
Solution:

(a)

$$\begin{aligned} S &\rightarrow PcA \mid AcP \\ P &\rightarrow 1P1 \mid 0P0 \mid c \\ A &\rightarrow 1A \mid 0A \mid \epsilon \end{aligned}$$

The two productions associated to start variable S let us choose if we want to derive a string in L of type (i) $v = u^R$ or (ii) $v = w^R$. (Note that even with a given choice, the other alternative may be valid as well, which leads to the ambiguity.) Then variable P generates an odd-length palindromic string over $\{0,1\}^*$ with a c in the middle, and variable A generates every possible string over $\{0,1\}^*$.

(b) To show the ambiguity of a grammar, it is enough to come up with one example which has more than one parse tree. Here, all strings which are of both types (i) and (ii) create such ambiguity. For instance, consider $x = 1c1c1$ which is one of the simplest examples. (Actually $x = cc$ would be the simplest, but does not make a very illustrative example.) This has the following parse trees:



(c) **Proof by contradiction.** Suppose that the language L is regular. Then, according to the *pumping lemma* there exists a $p \geq 1$ such that we can write any string $x \in L$ with length at least p in form $x = uvw$ where $|uv| \leq p$ and $|v| \geq 1$, so that all strings of form $x' = uv^k w$, $k = 0, 1, 2, \ldots$ are also in L.

Let $x = 1^p c 1^p c 0^p \in L$ which obviously is longer than p. Now, necessarily, $u = 1^r$, $v = 1^m$ and $w = 1^{p-r-m} c 1^p c 0^p$ where $m \geq 1$. Therefore, also strings of form $x' = 1^r 1^{km} 1^{p-r-m} c 1^p 0^p = 1^{p+(k-1)m} c 1^p c 0^p$ should belong to L for all values of k. But choose for instance $k = 2$. Then we get $x' = 1^{p+m} c 1^p c 0^p$ which is obviously not in L. We have reached a contradiction, which means that the original assumption is wrong and the language L is not regular.

4. **Problem:**

(a) Design a context-free grammar that generates the language

$$L = \{a^m b^n c^{m+n} \mid m, n \geq 0\}.$$

(b) Prove that the language L in part (a) is not regular.

Solution: Language L is generated by the following context-free grammar:

(a)

$$S \to aSc \mid A \mid \varepsilon$$
$$A \to bAc \mid \varepsilon$$

The grammar is split into two parts: variable S first generates matching pairs of a and c, and then variable A generates matching pairs of b and c. This arrangement guarantees that the total number of c's equals the sum of the numbers of a's and b's, and also that the symbols appear in the correct order.

(b) We will use the pumping lemma for regular languages to show that language L is not regular. For a contradiction, suppose that L is a regular language. Then by the pumping lemma there exists $p > 0$ such that every string $w \in L$ of length $|w| \geq p$ can be decomposed into three parts $w = xyz$ which satisfy the following conditions:

(i) $|xy| \leq p$,
(ii) $|y| \geq 1$,
(iii) for each $i \geq 0$, $xy^i z \in L$.

Now let $p > 0$ be as stated above. Consider the word $w = a^p b^p c^{2p} \in L$ and decompose it into parts xyz as indicated. By condition (i) and the structure of the chosen w, it must be the case that both x and y contain only a's. Let thus $y = a^q$ for some $q \geq 1$ (condition (ii)). By condition (iii), the word $w' = xy^2 z = a^{p+q} b^p c^{2p}$ should then also be in L. But w' is not in L, as the number of a's and b's in it together do not equal the number of c's ($2p + q \neq 2p$) which would be required for $w' \in L$. We thus arrive at a contradiction, and conclude that L cannot be regular.

5. **Problem:**

Consider the following context-free grammar G:

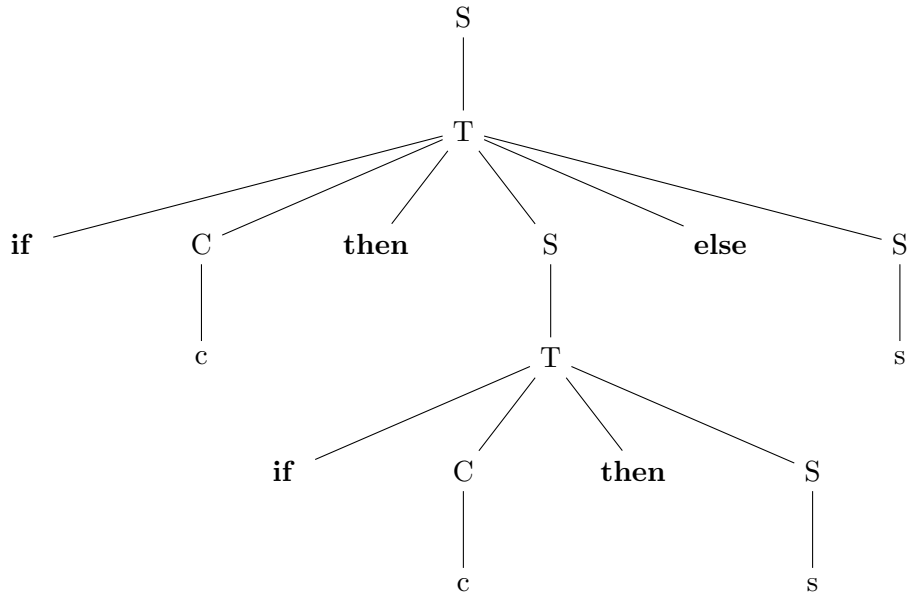
$$\begin{aligned} S &\rightarrow s \mid T \\ T &\rightarrow \textbf{if } C \textbf{ then } S \mid \textbf{ if } C \textbf{ then } S \textbf{ else } S \\ C &\rightarrow c \end{aligned}$$

(a) Give a parse tree for the string "**if** c **then if** c **then** s **else** s" in G.

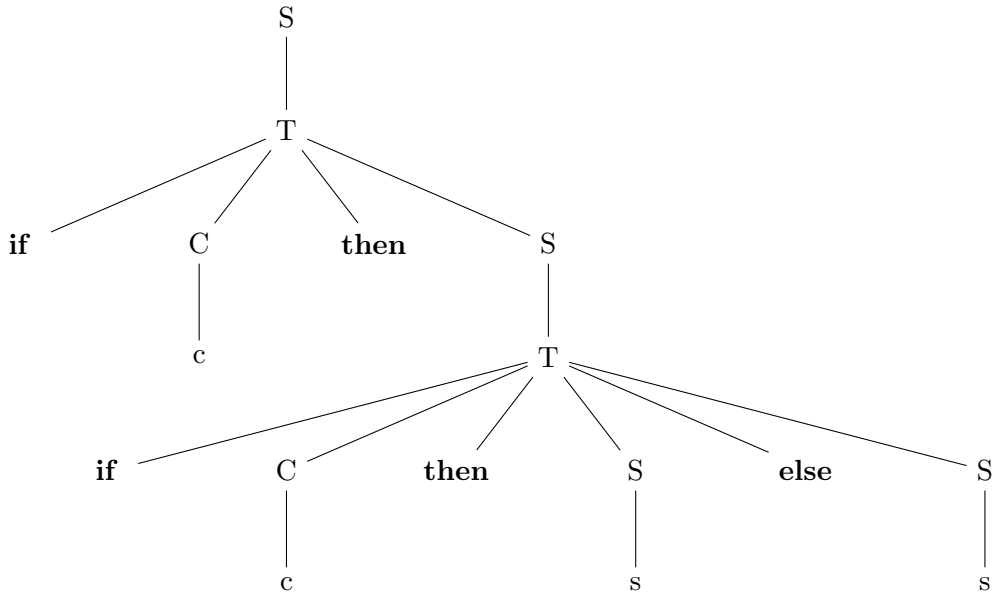
(b) Show that G is ambiguous.

Solution:

(a) A parse tree is drawn below:



(b) A grammar is ambiguous if there are two parse trees for some word in it. Therefore it suffices to present a second parse for the string in part (a):



6. **Problem:**

Prove that all regular languages are context-free, without appealing to the correspondence between context-free grammars and pushdown automata. (Using this correspondence would make the proof trivial, since finite state automata are a special case of pushdown automata.) Illustrate your proof with an example.

Solution:

In order to prove that every regular language is context-free, we show by construction how to transform a DFA accepting a regular language into a context-free grammar that generates the same language.

Let B be a regular language. By definition, there exists a DFA $M = (Q, \Sigma, \delta, q_0, F)$ such that $L(M) = B$. Based on M, we design a context-free grammar $G_B = (V, \Sigma, R, S)$ generating B in the following way:

- For each state $q \in Q$ introduce corresponding variable $R_q \in V$.
- For every transition rule $\delta(q_i, a) = (q_j)$, where $q_i, q_j \in Q$ and $a \in \Sigma$, add production $R_{q_i} \to aR_{q_j}$.
- For every accepting state $q \in F$, add production $R_q \to \varepsilon$.
- Define the start variable S as R_{q_0}.

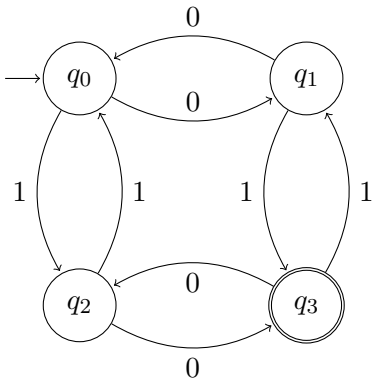
This "right-linear" context-free grammar G_B generates B, and thus B is a context-free language.

Let us demonstrate this method with the regular language

$$L = \{w \in \{0,1\}^* \mid \text{w contains odd number of 0's and odd number of 1's}\}.$$

We will show how to obtain a grammar for this language from its recognising DFA, and how to generate string $01001010 \in L$ using the grammar.

DFA:



We transform this DFA to a context-free grammar $G = (V, \Sigma, R, S)$:

- For set of states $Q = \{q_0, q_1, q_2, q_3\}$ we introduce set of variables $V = \{R_{q_0}, R_{q_1}, R_{q_2}, R_{q_3}\}$.

- After adding new rules and renaming R_{q_0} as the start variable S we have the following grammar:

$$\begin{aligned} S &\to 0R_{q_1} \mid 1R_{q_2} \\ R_{q_1} &\to 0S \quad \mid 1R_{q_3} \\ R_{q_2} &\to 0R_{q_3} \mid 1S \\ R_{q_3} &\to 0R_{q_2} \mid 1R_{q_1} \mid \varepsilon \end{aligned}$$

This grammar generates string 01001010 in the following way:

$$\underline{S} \Rightarrow 0\underline{R}_{q_1} \Rightarrow 01\underline{R}_{q_3} \Rightarrow 010\underline{R}_{q_2} \Rightarrow 0100\underline{R}_{q_3} \Rightarrow 01001\underline{R}_{q_1} \Rightarrow 010010\underline{R}_{q_0} \Rightarrow 0100101\underline{R}_{q_2}$$
$$\Rightarrow 01001010\underline{R}_{q_3} \Rightarrow 01001010\varepsilon = 01001010$$

III Turing Machines and Computability

1. **Problem:** Design a deterministic single-tape Turing machine that computes the function $f(n) = n \bmod 3$. The machine receives the string 1^n as input, and at the end of the computation it must leave $n \bmod 3$ ones at the beginning of the tape, where $n \bmod 3$ denotes the remainder when n is divided by 3. The other ones must be overwritten with the symbol #.

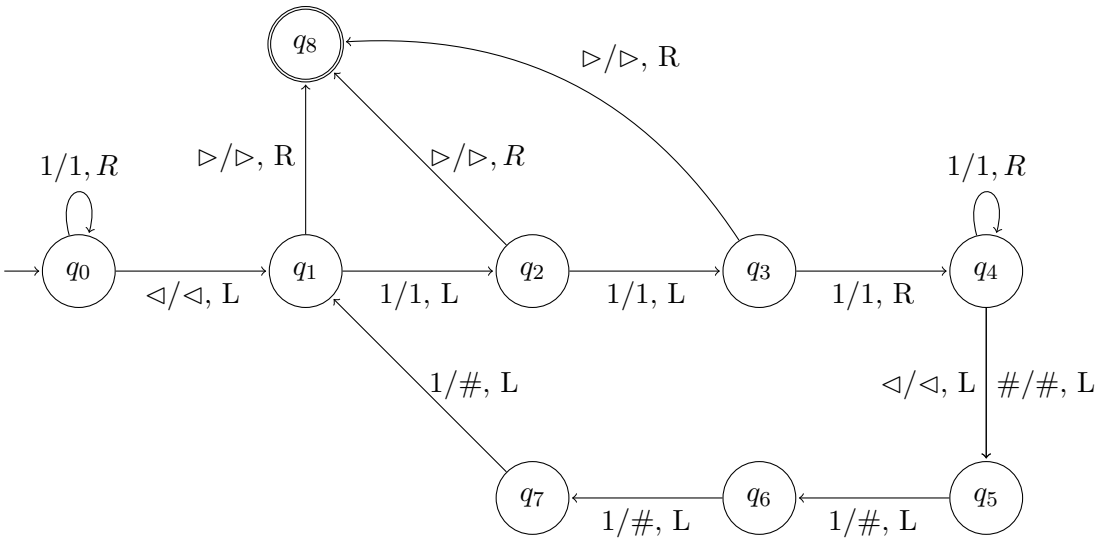
   (a) Present your Turing machine as a state diagram.

   (b) Write an overview of the method your machine uses.

   (c) Give the computation of your machine with the inputs 111 and 11111.

   Solution:

   (a)



   (b) The machine overwrites three 1's at a time in a loop, until there are fewer than three left.

   The remaining 1's should be at the start of the tape, so the machine initially moves to the end and starts overwriting from there. The overwriting happens in a loop: the machine checks whether there are at least three 1's, and then overwrites them if there are. If it finds the start marker when checking, that means there are fewer than three 1's left so it goes to the accept state and halts.

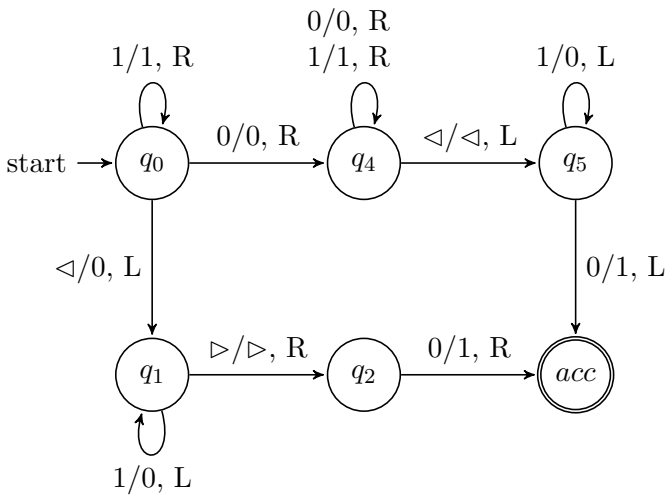
   (c) The computation on input 111:
   $(q_0, \underline{1}11) \vdash (q_0, 1\underline{1}1) \vdash (q_0, 11\underline{1}) \vdash (q_0, 111\underline{\varepsilon}) \vdash$
   $(q_1, 11\underline{1}) \vdash (q_2, 1\underline{1}1) \vdash (q_3, \underline{1}11) \vdash (q_4, 1\underline{1}1) \vdash (q_4, 11\underline{1}) \vdash (q_4, 111\underline{\varepsilon}) \vdash$
   $(q_5, 11\underline{1}) \vdash (q_6, 1\underline{1}\#) \vdash (q_7, \underline{1}\#\#) \vdash (q_1, \underline{\varepsilon}\#\#\#) \vdash (q_8, \underline{\#}\#\#)$
   The computation on input 11111:
   $(q_0, \underline{1}1111) \vdash (q_0, 1\underline{1}111) \vdash (q_0, 11\underline{1}11) \vdash (q_0, 111\underline{1}1) \vdash (q_0, 1111\underline{1}) \vdash (q_0, 11111\underline{\varepsilon}) \vdash$
   $(q_1, 1111\underline{1}) \vdash (q_2, 111\underline{1}1) \vdash (q_2, 11\underline{1}11) \vdash (q_3, 111\underline{1}1) \vdash (q_4, 1111\underline{1}) \vdash (q_4, 11111\underline{\varepsilon}) \vdash$
   $(q_5, 1111\underline{1}) \vdash (q_6, 111\underline{1}\#) \vdash (q_7, 11\underline{1}\#\#) \vdash (q_1, 1\underline{1}\#\#\#) \vdash$
   $(q_1, \underline{1}1\#\#\#) \vdash (q_1, \underline{\varepsilon}11\#\#\#) \vdash (q_8, \underline{1}1\#\#\#)$

2. **Problem:** Design a deterministic single-tape Turing machine that adds one to the binary number it receives as input. For example, the machine should replace input string 1001 by the string 1010 and the input string 11 by the string 100. Give a description of your algorithm, present the Turing machine as a state chart and give the computations of the machine with the input strings 1011 and 111.

   Solution: As the numbers are written most significant bit first, we might encounter a situation, where we need to move the input to the right. Luckily, as we only increase the number by one, the only scenario when this happens is when our input consists entirely of 1's, and the retyping of the input is kind of trivial: we add one 0 to the (right) end of the tape, and change all the 1's in the input to 0's, except the first (leftmost) one.

   When we do have 0's in our input, we want to take a different approach. First we want to move to the right end of the tape. Then we want to replace every 1 we encounter with a 0, and first 0 with a 1. When we find the first zero, we can move to the accept state.

   We can determine which approach we want to take while we move from the left end to the right end of the tape, depending on if we encounter any 0's on our way.



   Let's handle input strings 1011 and 111 using our brand-new machine. Here is the computation on 1011:

$$(q_0, \underline{1}011) \vdash (q_0, 1\underline{0}11) \vdash (q_4, 10\underline{1}1) \vdash$$
$$(q_4, 101\underline{1}) \vdash (q_4, 1011\underline{\varepsilon}) \vdash (q_5, 101\underline{1}) \vdash$$
$$(q_5, 10\underline{1}0) \vdash (q_5, 1\underline{0}00) \vdash (acc, \underline{1}100).$$

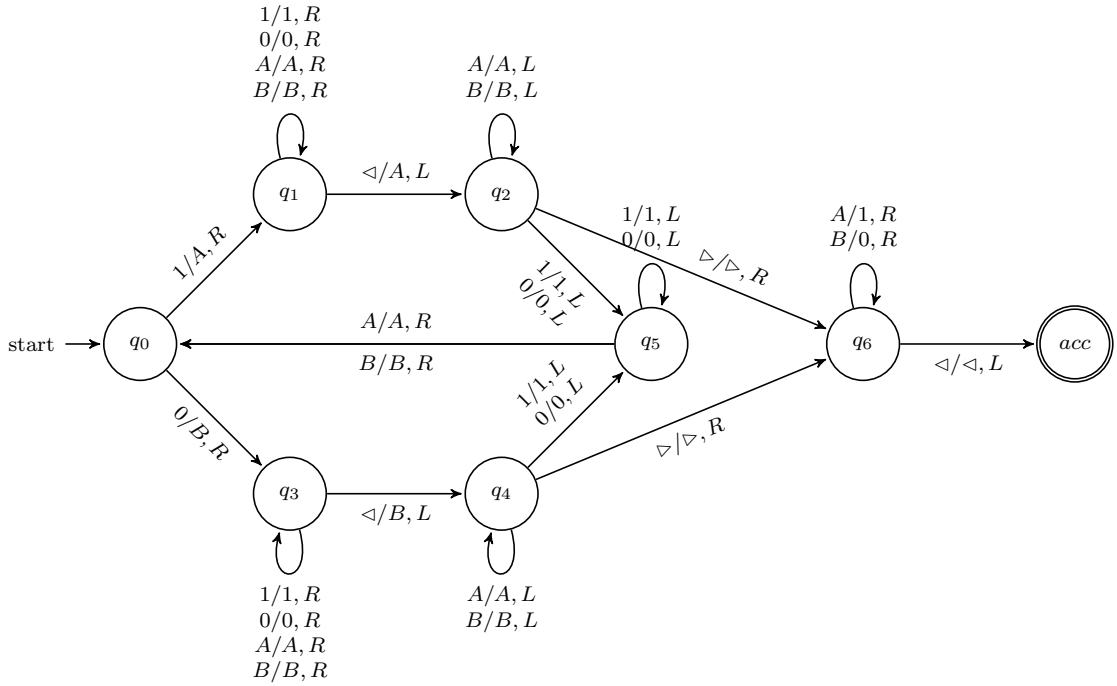
   And here on 111:

$$(q_0, \underline{1}11) \vdash (q_0, 1\underline{1}1) \vdash (q_0, 11\underline{1}) \vdash$$
$$(q_0, 111\underline{\varepsilon}) \vdash (q_1, 11\underline{1}0) \vdash (q_1, 1\underline{1}00) \vdash$$
$$(q_1, \underline{1}000) \vdash (q_1, \underline{\varepsilon}0000) \vdash (q_2, \underline{0}000) \vdash$$
$$(acc, 1\underline{0}00).$$

3. **Problem:**

Design a deterministic single-tape Turing machine that duplicates its input: if the tape initially contains a string $w \in \{0,1\}^*$, then when the machine halts the tape contains the string ww. Present your Turing machine as a state diagram, and give its computation sequence on input 10.

Solution: Briefly, the idea is to read one symbol and then copy that to the end. We should know which symbols have already been checked/copied, thus after reading a symbol we change it to A if it was 1 and B if it was 0. Then, we will write A or B at the end respectively for 1 and 0. When all the tape symbols become A or B, the input has been completely copied and we should change the symbols back to 1 and 0.



For input string 10 the computation sequence is the following:

$$
\begin{aligned}
&(q_0, \underline{1}0) \vdash (q_1, A\underline{0}) \vdash (q_1, A0\underline{\varepsilon}) \vdash \\
&(q_2, A\underline{0}A) \vdash (q_5, \underline{A}0A) \vdash (q_0, A\underline{0}A) \vdash \\
&(q_3, AB\underline{A}) \vdash (q_3, ABA\underline{\varepsilon}) \vdash (q_4, AB\underline{A}B) \vdash \\
&(q_4, A\underline{B}AB) \vdash (q_4, \underline{A}BAB) \vdash (q_4, \underline{\varepsilon}ABAB) \vdash \\
&(q_6, \underline{A}BAB) \vdash (q_6, 1\underline{B}AB) \vdash (q_6, 10\underline{A}) \vdash \\
&(q_6, 101\underline{B}) \vdash (q_6, 1010\underline{\varepsilon}) \vdash (acc, 101\underline{0})
\end{aligned}
$$

IV Unclassified

1. **Problem:**

   (a) Prove that if the languages $L \subseteq \{0, 1, \#\}^*$ and $L' \subseteq \{0, 1\}^*$ are context-free, then so is the language $L'' = L[L'] \subseteq \{0, 1\}^*$, whose words are obtained from the words in L by replacing each #-symbol by some word in L' (not necessarily always the same).

   (b) The same problem as in part (a), but with respect to semi-decidable (Turing-recognisable) rather than context-free languages.

   Solution:

   (a) Let $L \subseteq \{0, 1, \#\}$ and $L' \subseteq \{0, 1\}$ be context-free, and let G and G' be context-free grammars generating the languages L and L', respectively. Without loss of generality, we may assume that all the variables in G and G' are distinct. Let the start symbol of G be S and the start symbol of G' be $S' \neq S$. Using G and G', we can then construct a grammar, call it $G[G']$, that generates the language $L[L']$ as follows:

   - The start symbol of $G[G']$ is the start symbol S of G.
   - For each production $A \to \omega$ of G, replace each occurence of the terminal # in ω by the start symbol S' of G'. Add the modified productions to $G[G']$.
   - Add every production $A \to \omega$ of G' to $G[G']$.

   The construction ensures that every word of L can be also derived from $G[G']$ with each occurence of # replaced by S', and vice versa. From each S', any word in G' can be derived, therefore $G[G']$ generates the language $L[L']$

   (b) Let T and T' be Turing machines recognising the languages L and L', respectively. We may construct a non-deterministic Turing machine $T[T']$, which replaces non-deterministically arbitrary substrings w' (possibly $w' = \varepsilon$) of the input w by the character # if $w' \in L'$ (i.e. w' is accepted by T') to get a new string $\widetilde{w}$, and then simulates T with input $\widetilde{w}$. By construction, $T[T']$ recognises $L[L']$, and since deterministic and non-deterministic Turing-machines recognise the same languages, we conclude that also $L[L']$ is Turing-recognisable i.e. semi-decidable.

2. **Problem:**

   A language class C is *closed under complement*, if for every $L \in C$ also $\bar{L} \in C$.

   (a) Show that the class of regular languages is closed under complement.

   (b) Show that the class of context-free languages is not closed under complement. (*Hint:* The language $L = \{a^n b^n c^n \mid n \geq 0\}$ is not context-free.)

   (c) Show that the class of decidable languages is closed under complement.

   Solution:

(a) We want to prove that if a language L is regular, then $\bar{L} = \Sigma^* \setminus L$ is also regular. Assume then that L is regular. This means there is a finite automaton M that recognises L. Based on this machine we can construct a machine M' that recognises $\bar{L}$, thereby proving that $\bar{L}$ is regular. The construction is quite simple: M' is identical to M expect that its accept states are normal states, and its normal states accept states. Consider how M' behaves on some input x. If $x \in L$ then M accepts, meaning that the computation ends in an accepting state. But this state is not accepting in M' and therefore M' rejects as it should since $x \in L$ is equivalent to $x \notin \bar{L}$. If $x \notin L$ then M rejects, meaning that the computation ends in a state that is not accepting. But such a state is accepting in M' and therefore M' accepts, as it should since $x \notin L$ is equivalent to $x \in \bar{L}$.

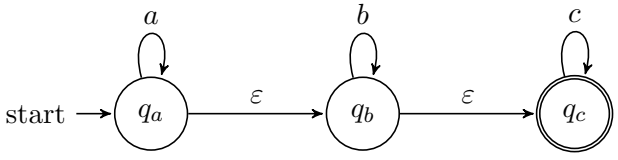
(b) We prove the statement by counterexample: we show that the complement $\bar{L}$ of the non-contextfree language $L = \{a^n b^n c^n \mid n \geq 0\}$ is in fact context-free. Hence there is a context-free language ($\bar{L}$) whose complement ($\bar{\bar{L}} = L$) is not context-free. Consider the language $\bar{L}$. It contains all strings that don't have the same amount of a's b's and c's appearing in order. We proceed to separate the language into parts that we can show are context-free, eventually showing the entire language is context-free.

Strings in $\bar{L}$ can be of two types: they can either have differing numbers of a's, b's and c's, or the letters can be in the wrong order. In other words $\bar{L} = \bar{L}_{\text{ord}} \cup L_{\text{num}}$ where $L_{\text{ord}} = \{a^i b^j c^k \mid i, j, k \in \mathbb{N}\}$ and $L_{\text{num}} = \{a^i b^j c^k \mid i \neq j \text{ or } j \neq k\}$. We can further separate $L_{\text{num}} = L_{a \neq b} \cup L_{b \neq c}$, where $L_{x \neq y}$ means the language where the numbers of letters x and y differ. We have then that

$$\bar{L} = \bar{L}_{\text{ord}} \cup L_{a \neq b} \cup L_{b \neq c}.$$

We will show that these component languages are context-free one by one, and then show that the class of context-free languages is closed under union, thus concluding that $\bar{L}$ is context-free.

$\bar{L}_{\textbf{ord}}$ We show that L_{ord} is a regular language, so its complement $\bar{L}_{\text{ord}}$ is also regular by part (a). Since the class of regular languages is a subset of the class of context-free languages, we can then conclude that $\bar{L}_{\text{ord}}$ is context-free. We show that L_{ord} is regular by describing a nondeterministic finite automaton that recognises it. The state diagram for such a machine is shown below.



$L_{a \neq b}$ We can describe this language with a context-free grammar. We can think of the language as composed of two cases: either there are more a's than b's or less a's than b's. The first rule of the grammar below expresses this idea.

$$\begin{aligned} S &\to MC \mid LC \\ M &\to aM \mid aE \\ L &\to Lb \mid Eb \\ E &\to aEb \mid \varepsilon \\ C &\to cC \mid \varepsilon \end{aligned}$$

If we use the production $S \to MC$ then the M first adds one or more a's to the beginning of the string and then an equal number of a's and b's, leading to an excess of a's. Then the C adds some number of c's at the end. In the production $S \to LC$ some b's are added at first, leading there to be fewer a's than b's.

$L_{b \neq c}$ The idea here is very similar to the previous part, the grammar is below

$$\begin{aligned} S &\to AM \mid AL \\ M &\to bM \mid bE \\ L &\to Lc \mid Ec \\ E &\to aEb \mid \varepsilon \\ A &\to aA \mid \varepsilon \end{aligned}$$

Finally to show that the class of context-free languages is closed under union, let L_1 and L_2 be two context-free languages, described by context-free grammars G_1 and G_2. A context-free grammar for $L_1 \cup L_2$ can then be constructed by combining all productions from G_1 and G_2 (renaming variables if necessary), renaming their start variables as S_1 and S_2 and adding a new start production $S \to S_1 \mid S_2$.

(c) This argument is very similar to that for regular languages. Assume that L is decidable, meaning that there is some Turing machine M that recognises it and halts on all inputs. We can construct a Turing machine M' that is identical to M expect that the accepting and rejecting states are swapped. Machine M' then also halts on every input, and gives the opposite answer from M. Clearly M' recognises the language $\bar{L}$.

3. **Problem:**

Show that if a language $L \subseteq \Sigma^*$ is semi-decidable but not decidable, then its complement language $\bar{L} = \Sigma^* - L$ is not semi-decidable. (You may assume as known any auxiliary results related to this claim that have been presented during the course.)

Solution: Let L be a language that is semi-decidable but not decidable, and suppose that its complement $\bar{L}$ is also semi-decidable. We will show that then L would in fact be decidable, a contradiction.

Since L is semi-decidable, there is by definition a Turing machine M_1 that recognises it, i.e. on any input $x \in L$, M_1 halts and accepts. Similarily, there is a machine M_2 that recognises $\bar{L}$. We construct from these two machines a Turing machine M that recognises L correctly and halts on all inputs, establishing that L is decidable, contrary to our assumption.

We will describe M on a general level, but an exact definition could be written based on this description. The machine M simulates the machines M_1 and M_2 in parallel on two independent tapes. Given an input x, M runs both M_1 and M_2 with input x and accepts if M_1 halts and accepts, and rejects if M_2 halts and accepts. Since an input x either is in the language L or it is not, machine M always halts and gives the correct answer on whether an input x is in language L or not. Hence L is decidable.

4. **Problem:**

   Give a brief but precise justification, based on results presented on the course, for each of the following statements: (i) all regular languages are context-free, (ii) all context-free languages are decidable.

   Solution:

   (i) Any regular language L can, by definition, be recognised by some finite automaton. Each finite automaton can be transformed into a (left- or right-recursive) context-free grammar recognising the same language. This means that the language L is also context-free.

   (ii) Any context-free language L is, by definition, generated by some context-free grammar G. Grammar G can be effectively transformed into an equivalent Chomsky normal form grammar G', which can then be used in the CYK parsing algorithm to decide whether a given input word is in the language L. The CYK-algorithm can be implemented for example in C, which by the Church-Turing thesis implies that the language L is decidable.