Computability and Complexity

Instituto de Matemáticas

Exercises

Context-free languages

Exercise 1 : *Elementary training.*
Let $\Sigma = \{a, b\}$. For each of the following languages, give a context-free grammar that generates it:

1. $\{w \in \Sigma^* \mid w$ contains at least three $b\}$;
2. $\{w \in \Sigma^* \mid w$ starts and ends with the same symbol$\}$;
3. $\{w \in \Sigma^* \mid w$ has odd length$\}$;
4. $\{w \in \Sigma^* \mid w$ has odd length and its middle symbol is $a\}$;
5. $\{w \mid w$ is a palindrome$\}$;
6. $\{w \mid w$ contains as many a as $b\}$.

Answer:

1. $S \to RbRbRbR$
   $R \to \epsilon \mid Ra \mid Rb$
2. $S \to aRa \mid bRb \mid \epsilon$
   $R \to \epsilon \mid aR \mid bR$
3. $S \to aA \mid bA$
   $A \to \epsilon \mid aS \mid bS$
4. $S \to a \mid aSa \mid aSb \mid bSa \mid bSb$
5. $S \to \epsilon \mid a \mid b \mid aSa \mid bSb$
6. $S \to \epsilon \mid SaSbS \mid SbSaS$

□

Exercise 2 : *Ambiguous elementary school.*
Let $\Sigma = \{a, b, c\}$. Give a context-free grammar that generates the language

$$\{a^i b^j c^k \mid i = j \text{ or } j = k;\ i, j, k \geq 0\}.$$

Is your grammar ambiguous? Justify.
Answer: The following grammar ensures that either there are as many a than b, or that there are as many b than c:

$$\begin{aligned} S &\to \epsilon \mid R_{ij}C \mid AR_{jk} \\ R_{ij} &\to \epsilon \mid aR_{ij}b \\ C &\to \epsilon \mid cC \\ R_{jk} &\to \epsilon \mid bR_{jk}c \\ A &\to \epsilon \mid aA \end{aligned}$$

However, this grammar is ambiguous. Indeed, the string abc can be generated by the two following leftmost derivations:

$$S \Rightarrow R_{ij}C \Rightarrow aR_{ij}bC \Rightarrow abC \Rightarrow abcC \Rightarrow abc$$

and

$$S \Rightarrow AR_{jk} \Rightarrow aAR_{jk} \Rightarrow aR_{jk} \Rightarrow abR_{jk}c \Rightarrow abc.$$

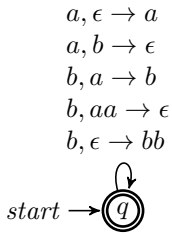
□

Exercise 3 : *Pushdown automatas are not pushovers!*.
For each of the following languages, construct a pushdown automata that recognizes it:

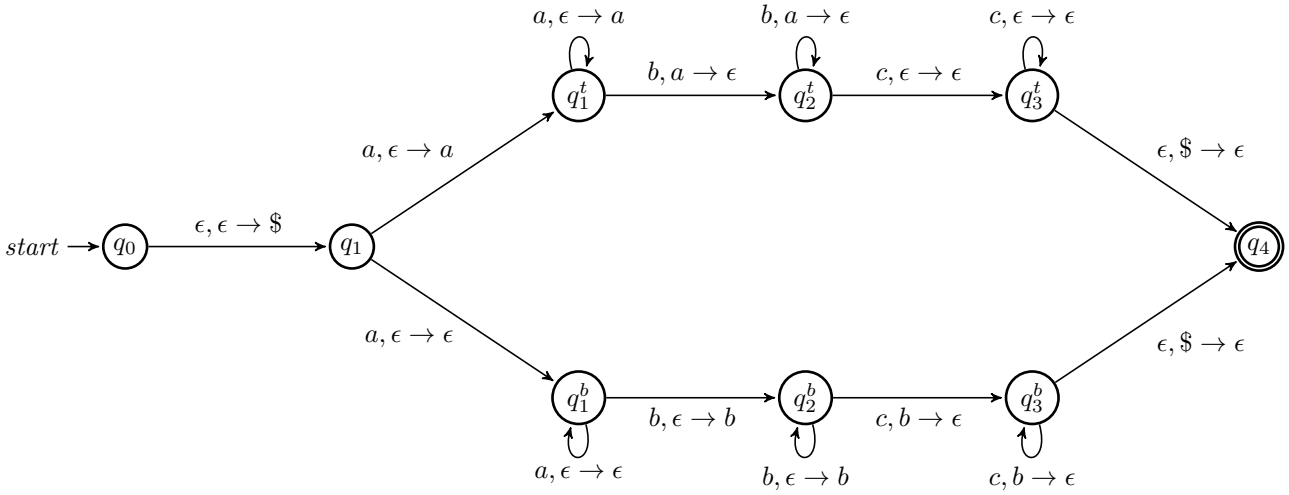
1. $\{w \in \{a,b\}^* \mid w \text{ has twice as many } a \text{ than } b\}$;
2. $\{a^i b^j c^k \mid i = j \text{ or } j = k;\ i,j,k > 0\}$;
3. $\{a^m b^n c^{2(m+n)} \mid m,n \geq 0\}$.

Answer: We consider that we can push and pop several symbols at the same time. As we saw in the course, this can be done by adding new states.

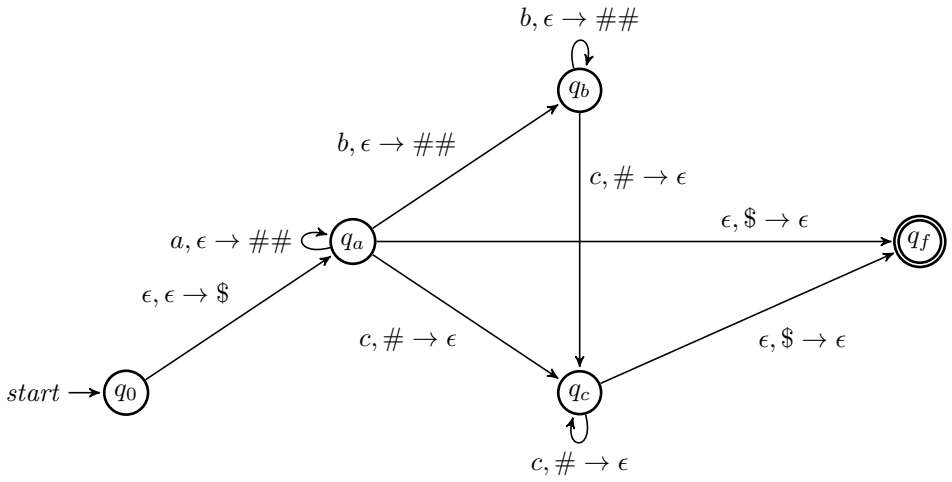
1. *The automata will read the word and use the stack to "count" a and b. If it reads an a and the top of the stack is a b, it will pop it; otherwise it will push an a. If it reads a b, there are several cases: if there are two a on top of the stack, it will pop them; if there is an a, it will replace it by a b, and otherwise, it will push two b. It is easy to check that the word will be accepted (by empty stack) if there are twice as many a than b.*



2. *The automata will be nondeterministic. The top path will be the case where we will check if $i = j$, and the bottom path will be the case where we will check if $j = k$.*



3. *We will use the stack to count the a and b, by pushing two characters for each of those. Then, we will read c while popping the characters on the stack. The automata will accept the word if, at the end, the stack is empty. Note that we have to consider the where the word is empty.*



□

Exercise 4 : *Closing exercise.*
Prove that the class of context-free languages is closed under union, concatenation, star and reversal[1].
Answer:

1. *Let L_1 and L_2 be two context-free languages, and let $G_1 = (V_1, \Sigma_1, R_1, S_1)$ and $G_2 = (V_2, \Sigma_2, R_2, S_2)$ be the context-free grammars such that $L(G_1) = L_1$ and $L(G_2) = L_2$. We construct $G = (V, \Sigma, R, S)$ such that $L(G) = L_1 \cup L_2$:*
   - $\Sigma = \Sigma_1 \cup \Sigma_2$;
   - $V = V_1 \cup V_2 \cup \{S\}$;
   - $R = R_1 \cup R_2 \cup \{S \to S_1 \mid S_2\}$.
2. *Let L_1 and L_2 be two context-free languages, and let $G_1 = (V_1, \Sigma_1, R_1, S_1)$ and $G_2 = (V_2, \Sigma_2, R_2, S_2)$ be the context-free grammars such that $L(G_1) = L_1$ and $L(G_2) = L_2$. We construct $G = (V, \Sigma, R, S)$ such that $L(G) = L_1 L_2$:*
   - $\Sigma = \Sigma_1 \cup \Sigma_2$;
   - $V = V_1 \cup V_2 \cup \{S\}$;
   - $R = R_1 \cup R_2 \cup \{S \to S_1 S_2\}$.
3. *Let L be a context-free language, and let $G = (V, \Sigma, R, S)$ such that $L(G) = L$. We construct $G^* = (V^*, \Sigma^*, R^*, S^*)$ such that $L(G^*) = L^*$:*
   - $\Sigma^* = \Sigma$;
   - $V^* = V \cup \{S^*\}$;
   - $R^* = R \cup \{S^* \to \epsilon \mid S^* S\}$.
4. *Let L be a context-free language, and let $G = (V, \Sigma, R, S)$ such that $L(G) = L$. The grammar $G^R = (V, \Sigma, R^R, S)$ (where R^R is R where every right-hand side of each rule is reversed) verifies $L(G^R) = L^R$.*

□

Exercise 5 : *More pumping!.*
Use the pumping lemma to prove that the following languages are not context-free:

1. $\{a^n b^n a^n b^n \mid n \geq 0\}$;
2. $\{a^n b a^{2n} b a^{3n} \mid n \geq 0\}$;

[1]Recall that, given L, its reverse language L^R is the language containing all the words in L in reverse order.

3. $\{w\#t \mid w, t \in \{a,b\}^*$ and w is a substring of $t\}$.

Answer: In all the following answers, we assume by contradiction that the language is context-free, and apply the pumping lemma. Let p be the pumping length.

1. *Let $s = a^p b^p a^p b^p = uvxyz$. Now, since $|vxy| \leq p$, there are several possibilities:*
   - *If vxy is composed only of a or b, then the string uv^2xy^2z will not be in the language (since $|vy| \geq 1$), a contradiction.*
   - *If vxy is in the first (resp. second) $a^p b^p$, then $s' = uv^2xy^2z = a^{p+k}b^{p+\ell}a^p b^p$ (resp. $a^p b^p a^{p+k} b^{p+\ell}$) with at least one of k and ℓ positive, and thus s' is not in the language, a contradiction.*
   - *If vxy contains the middle point of s, then $s' = uv^2xy^2z = a^p b^{p+k} a^{p+\ell} b^p$ with at least one of k and ℓ positive, and thus s' is not in the language, a contradiction.*
2. *Let $s = a^p b a^{2p} b a^{3p} = uvxyz$. Note that neither v nor y can contain a b, since otherwise uv^0xy^0z would contain less than three b and thus not belong to the language, a contradiction. By dividing the word into three segments of consecutive a, it is clear that v and y belong to at most two of these segments. Then, in $s' = uv^2xy^2z$, the $1:2:3$ length ratio between the segments is not maintained, since at most two segments see their length increase. Thus, s' is not in the language, a contradiction.*
3. *Let $s = a^p b^p \# a^p b^p = uvxyz$. Note that neither v nor y can contain $\#$, since otherwise uv^0xy^0z does not contain $\#$ and thus is not in the language, a contradiction. There are several cases to consider:*
   - *If both v and y are nonempty, and are in w, then uv^2xy^2z is not in the language, since the left part will contain more characters than the right part, a contradiction.*
   - *If both v and y are nonempty, and are in t, then uv^0xy^0z is not in the language, since the right part will contain less characters than the left part, a contradiction.*
   - *If one of v and y is empty, then consider that they are in the same part (w or t) and apply one of the above.*
   - *If both v and y are nonempty, v is in w and y is in t, then since $|vxy| \leq p$, we have $v = b^k$ and $y = a^\ell$ with $k, \ell > 0$. But now $s' = uv^2xy^2z = a^p b^{p+k} \# a^{p+\ell} b^p$, and the left part is not a substring of the right part. Thus, s' is not in the language, a contradiction.*

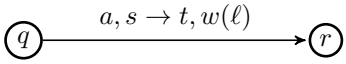
   All this implies that both v and y must be empty, which is a contradiction.

□

Exercise 6 : *Problem: doing additions.*
In this exercise, we add a *write* function to a pushdown automata. More formally, we define a pushdown writer-automata as a 7-uple: $A = (Q, \Sigma, \Gamma, W, \delta, q_0, F)$, where W is the writing alphabet, and

$$\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to \mathcal{P}(Q \times \Gamma_\epsilon \times W_\epsilon)$$

is the transition function. The way such an automata works is the following: it reads a character (which may be ϵ) of the input and may pop the top of the stack, then it changes a state while (maybe) pushing a character on top of the stack and (maybe) writing a character. The characters that are written cannot be modified afterwards. The following figure illustrates how we can draw such an automata:



Here, when in state q, reading the character a from the input and having s at the top of the stack, we will pop s, push t and write ℓ.
Other functions of the pushdown writer-automata are exactly the same than those of a classical pushdown automata, in particular the acceptance of an input.

1. Draw a pushdown writer-automata that recognizes the language $\{a^n b^n \mid n \geq 0\}$ and writes down the number of a in unary, followed by a #, followed by the number of b in unary. For instance, when reading *aaabbb*, it will write 111#111 and accept the word; and when reading *aabbb*, it will write 11#111 and reject the word.

2. Let w_1 and w_2 be two positive integers written in unary. Draw a pushdown writer-automata that writes their sum in unary. You have to decide the way to represent the two numbers as input. The automata must end its computation in a final state.

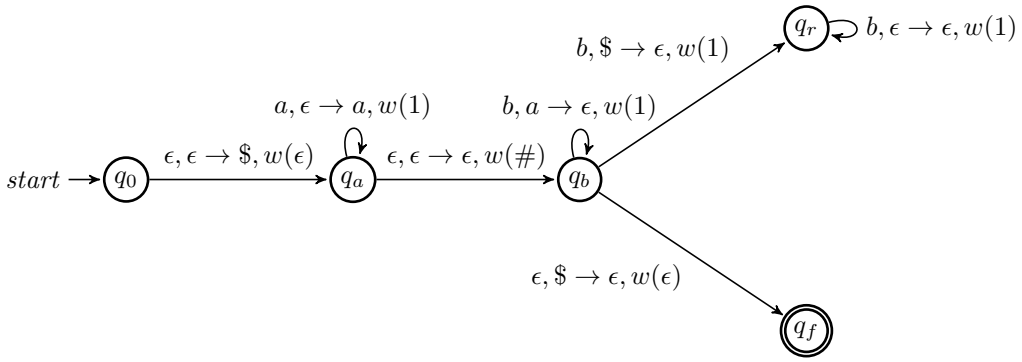
3. Let w_1 and w_2 be two positive integers written in binary, such that they have the same number of digits (if they do not, we can simply add 0s at the beginning of the smallest number until it is the case). Draw a pushdown writer-automata that writes their sum in binary. You have to decide the way to represent the two numbers as input. The automata must end its computation in a final state.
   Hint: You may have part of the input in reverse order, and write the result in reverse order too.

4. Explain how you could use more stacks in the previous writer-automata to have no reverse order shenanigans.

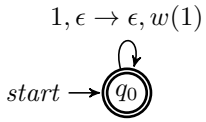
5. Explain how to construct a pushdown writer-automata to compute the sum of two positive integers written in decimal.

Answer:

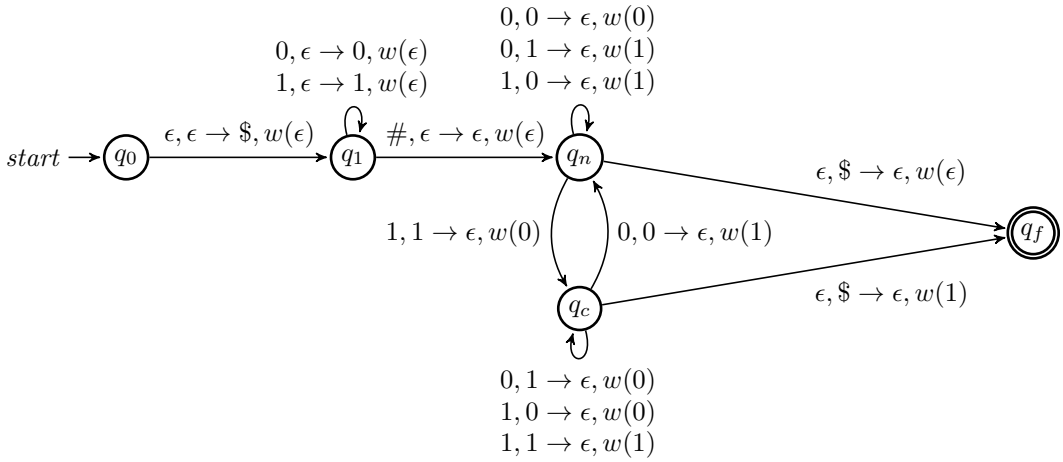
1. *We need to count the number of a, then the number of b. But if we have more b than a, then we need to reject the word, and so we create a rejecting state to continue counting the b (note that if we have more a than b, then we will be stuck in a non-accepting state).*



2. *If w_1 and w_2 are written in unary, then $w_1 + w_2 = w_1 w_2$. So the input will be $w_1 w_2$, and we simply have to write down the input. The stack is not necessary.*



3. *We will let the input be $w_1 \# w_2^R$. First, we will store w_1 in the stack, so that the highest weight digit will be at the bottom. Then, we will use two states (one for when we currently have no carry, and one for when we have one) to write down the sum of w_1 and w_2. It will obviously be in reverse order. Then, when going to the accept state, we may write a last 1 if we still had a carry.*



4. *We could use a second stack to store w_2, which would allow us to use $w_1\#w_2$ as an input. Then, we could use a third stack to store the result of the addition. When the stacks containing w_1 and w_2 become empty, we would simply write every character of the third stack, which would give us the result in the good order.*

5. *We can use the same template than in the previous questions. Indeed, when adding two numbers in decimal, the carry can be at most 1 (the highest possible sum is $9 + 9$ with a carry, which leaves a carry of 1). Now, we only need to be careful while writing the result. For example, when reading a 4 and popping a 2 in the no-carry state, we would write 6 and remain in the same state; when reading a 4 and popping a 6 in the no-carry state, we would write 0 and go to the carry state; when reading a 4 and popping a 5 in the carry state, we would write 0 and remain in the same state; and so on.*

   Like before, using three stacks allows us to have everything in the correct order.

□