Exercise sessions #4
Context free languages and Turing machines

prof. B. Demoen W. Van Onsem

March-April 2015

Exercise 1. Give for every of the following languages a context-free grammar that describes that language.

1. $\{w \in \{0,1\}^* \mid \text{the length of } w \text{ is odd and the symbol in the middle is a } 0\}$
2. $\{w \in \{0,1\}^* \mid w \text{ contains strictly more 1's than 0's}\}$
3. $\{w \in \{0,1\}^* \mid w \text{ contains twice as much 1's as 0's}\}$
4. $\{w_1\#w_2\#\ldots\#w_n \mid n \geq 1, \text{ every } w_i \in \{0,1\}^* \text{ and there exists an } i \text{ and } j \text{ such that } w_i = w_j^{\mathcal{R}}\}$

Give a push-down automaton that decides the first two languages.

Answer.

1. $\{w \in \{0,1\}^* \mid \text{the length of } w \text{ is odd and the symbol in the middle is a } 0\}$; see Grammar 1 and Figure 1.

$\langle S\rangle \rightarrow \mathbf{0}$
$\langle S\rangle \rightarrow \langle A\rangle\langle S\rangle\langle A\rangle$
$\langle A\rangle \rightarrow \mathbf{0}$
$\langle A\rangle \rightarrow \mathbf{1}$

Grammatica 1: Odd length, 0 in the middle.

$$\begin{tikzcd}
\text{start} \arrow[r] & q_0 \arrow[r, "\epsilon/\epsilon/\$"] & q_1 \arrow[loop above, "0/\epsilon/a{,}\ 1/\epsilon/a"] \arrow[r, "0/\epsilon/\epsilon"] & q_2 \arrow[loop above, "0/a/\epsilon{,}\ 1/a/\epsilon"] \arrow[r, "\epsilon/\$/\epsilon"] & q_3
\end{tikzcd}$$

Figure 1: $\{w \in \{0,1\}^* \mid \text{the length of } w \text{ is odd and the symbol in the middle is a } 0\}$.

2. $\{w \in \{0,1\}^* \mid w \text{ contains strictly more 1's than 0's}\}$
3. $\{w \in \{0,1\}^* \mid w \text{ contains twice as much 1's as 0's}\}$
4. $\{w_1\#w_2\#\ldots\#w_n \mid n \geq 1, \text{ every } w_i \in \{0,1\}^* \text{ and there exists an } i \text{ and } j \text{ such that } w_i = w_j^{\mathcal{R}}\}$

$\diamond$

Exercise 2. Let G be a context-free grammar in Chomsky normal form that contains exactly b nonterminals.

1. Show that for every $w \in L_G$ with length $n \geq 1$, every derivation of w consists of exactly $2n-1$ steps.

$\langle S\rangle \to \mathbf{1}$
$\langle S\rangle \to \mathbf{1}\langle A\rangle\langle S\rangle$
$\langle S\rangle \to \langle A\rangle\mathbf{1}\langle S\rangle$
$\langle S\rangle \to \mathbf{1}\langle S\rangle\langle A\rangle$
$\langle S\rangle \to \langle A\rangle\langle S\rangle\mathbf{1}$
$\langle S\rangle \to \langle S\rangle\mathbf{1}\langle A\rangle$
$\langle S\rangle \to \langle S\rangle\langle A\rangle\mathbf{1}$
$\langle A\rangle \to \mathbf{0}$
$\langle A\rangle \to \mathbf{1}$

Grammatica 2: More 1's than 0's.

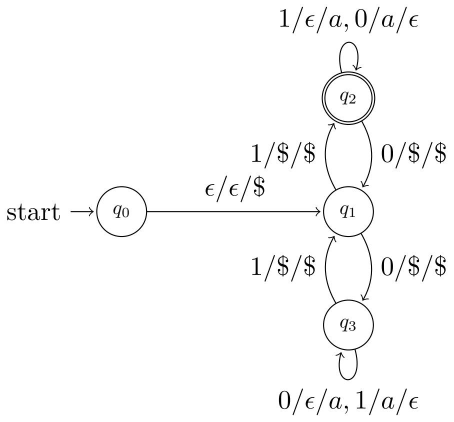


Figure 2: $\{w \in \{0,1\}^* \mid w$ contains strictly more 1's than 0's$\}$.

2. Show that, given L_G contains a string such that every derivation requires more than 2^b steps, L_G must contain an infinite amount of strings.

Answer. 1. A context-free grammar in Chomsky normal form has three possible rules:

   (a) $S \to \epsilon$; with S the start nonterminal. Since only the start nonterminal can be used, in such case $\epsilon \in L_G$; but since the empty string has a length of 0; this rule fails to meet the conditions, and thus is irrelevant.

   (b) $A \to a$ with A a nonterminal and a a character. In that case we generate a string of length 1 using 1 rule. Since $2 \cdot 1 - 1 = 1$; the rule holds.

   (c) $A \to BC$ with A a nonterminal and B and C nonterminals different from S. Since B and C are not the start nonterminal, the can't result in the empty string. So both B and C will derive to a nonempty string with lengths n_B and n_C and evidently $n_A = n_B + n_C$. Since the derivation of B requires $2 \cdot n_B - 1$ steps and the derivation of C requires $2 \cdot n_C + 1$ steps. Since it requires one additional step to split A into B and C, the total number of steps is $2 \cdot n_B - 1 + 2 \cdot n_C - 1 + 1 = 2 \cdot n_A - 1$. The theorem thus holds.

2. Since each rule $A \to BC$ splits a nonterminal in two new nonterminals, we construct a binary tree. If the number of steps is more than 2^b, this means there is at least one path from the root

$\langle S\rangle \to \epsilon$
$\langle S\rangle \to \langle S\rangle\mathbf{1}\langle S\rangle\mathbf{1}\langle S\rangle\mathbf{0}\langle S\rangle$
$\langle S\rangle \to \langle S\rangle\mathbf{1}\langle S\rangle\mathbf{0}\langle S\rangle\mathbf{1}\langle S\rangle$
$\langle S\rangle \to \langle S\rangle\mathbf{0}\langle S\rangle\mathbf{1}\langle S\rangle\mathbf{1}\langle S\rangle$

Grammatica 3: Twice as much 1's as 0's.

$\langle S\rangle \rightarrow \langle A\rangle\langle P\rangle\langle B\rangle$

$\langle P\rangle \rightarrow \epsilon$
$\langle P\rangle \rightarrow \mathbf{0}\langle P\rangle\mathbf{0}$
$\langle P\rangle \rightarrow \mathbf{1}\langle P\rangle\mathbf{1}$
$\langle P\rangle \rightarrow \langle D\rangle$
$\langle P\rangle \rightarrow \#$
$\langle P\rangle \rightarrow \#\langle R\rangle\#$

$\langle A\rangle \rightarrow \epsilon$
$\langle A\rangle \rightarrow \langle R\rangle\#$

$\langle B\rangle \rightarrow \epsilon$
$\langle B\rangle \rightarrow \#\langle R\rangle$

$\langle R\rangle \rightarrow \epsilon$
$\langle R\rangle \rightarrow \langle C\rangle\langle R\rangle$

$\langle C\rangle \rightarrow \#$
$\langle C\rangle \rightarrow \langle D\rangle$

$\langle D\rangle \rightarrow \mathbf{0}$
$\langle D\rangle \rightarrow \mathbf{1}$

Grammatica 4: Reverse group.

with a length more than b. Because of the pigeonhole argument, we know that this implies a nonterminal N occurs more than once in such path. This implies there exists a derivation sequence $N \rightarrow \ldots \rightarrow N$. This means we can repeat this derivation sequence an arbitrary number of times. For each derivation, we will introduce at least one extra nonterminal that differs from the starting nonterminal S. Since each such nonterminal results in a string with a length greater than zero, for each repetition, we will generate a different string. Since the number of repetitions is arbitrary, we can thus generate an arbitrary number of strings and thus is the number of strings in the language infinite.

$\diamond$

Exercise 3. Show that $\{x\#y \mid x, y \in \{0,1\}^* \text{ en } x \neq y\}$ is a context-free grammar over $\{0, 1, \#\}$.

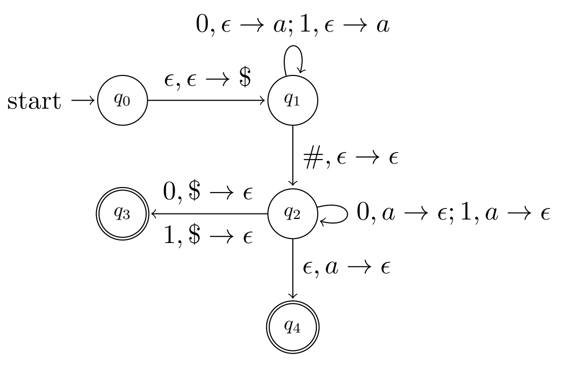
Answer. We can prove this by constructing a PDA. We construct a PDA using a union of two other PDA's[1] that decide the following languages:

1. the language $\left\{x\#y \mid x, y \in \{0,1\}^\star : |x| \neq |y|\right\}$, see Figure 3(a); and
2. the language $\left\{x\#y \mid x, y \in \{0,1\}^\star : \exists i \in \mathbb{N} : x_i \neq y_i\right\}$, see Figure 3(b).

The first PDA simply counts the number of characters of x by pushing on to the stack, after it has read #, it starts popping from the stack per character from y. If at the end of the process, the stack does not correspond with the initial marker ($), the PDA accepts.

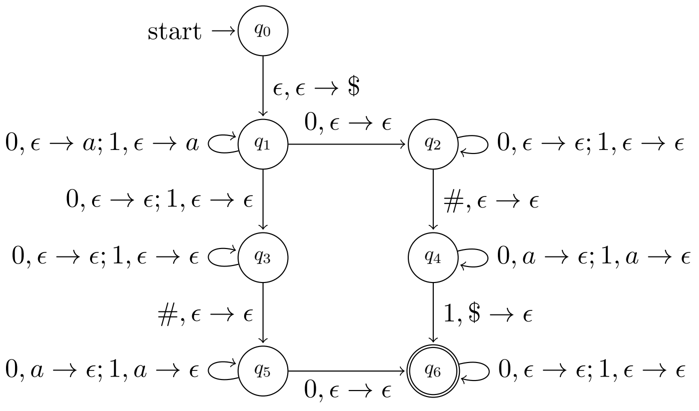
The second PDA "guesses" at which position the characters will differ. First it pushes an initial marker ($) to the stack, next it starts skipping a certain number of characters from x and pushes a's on to the stack until it decides to make a guess. The next character of x is then stored in the state of the DFA. Next it skips all the remaining characters of x. After it has read the split character #, it starts popping

[1]In contrast to the intersection and complement, the union is closed for context-free languages



$0, \epsilon \to a; 1, \epsilon \to a$
start
q_0
$\epsilon, \epsilon \to \$$
q_1
$\#, \epsilon \to \epsilon$
$0, \$ \to \epsilon$
$1, \$ \to \epsilon$
q_3
q_2
$0, a \to \epsilon; 1, a \to \epsilon$
$\epsilon, a \to \epsilon$
q_4

(a) $\{x\#y \mid x, y \in \{0,1\}^\star : |x| \neq |y|\}$



start
q_0
$\epsilon, \epsilon \to \$$
$0, \epsilon \to a; 1, \epsilon \to a$
q_1
$0, \epsilon \to \epsilon$
q_2
$0, \epsilon \to \epsilon; 1, \epsilon \to \epsilon$
$0, \epsilon \to \epsilon; 1, \epsilon \to \epsilon$
$\#, \epsilon \to \epsilon$
$0, \epsilon \to \epsilon; 1, \epsilon \to \epsilon$
q_3
q_4
$0, a \to \epsilon; 1, a \to \epsilon$
$\#, \epsilon \to \epsilon$
$1, \$ \to \epsilon$
$0, a \to \epsilon; 1, a \to \epsilon$
q_5
q_6
$0, \epsilon \to \epsilon; 1, \epsilon \to \epsilon$
$0, \epsilon \to \epsilon$

(b) $\{x\#y \mid x, y \in \{0,1\}^\star : \exists i \in \mathbb{N} : x_i \neq y_i\}$

Figure 3: sub-PDA's to decide the language $\big\{x\#y \mid x, y \in \{0,1\}^\star : x \neq y\big\}$.

from the stack for each character of y. The stack is used to align the strings, so that we inspect the character at the same position as the one of x. When we have popped the entire stack, we inspect whether the character of y differs. If so, we accept, otherwise we reject. Because of the nondeterminism of the PDA, we guess for every position. $\diamond$

Exercise 4. Which of the following languages are context-free? Prove your answer.

1. $\{w \in \{0,1\}^* \mid w \text{ is a palindrome}\}$
2. $\{0^n\#0^{2n}\#0^{3n} \mid n \geq 0\}$
3. $\{w \in \{a,b,c\}^* \mid w \text{ contains an equal amount of } a\text{'s}, b\text{'s and } c\text{'s}\}$
4. $\{xy \mid x, y \in \{0,1\}^* \text{ and } |x| = |y|, \text{ but } x \neq y\}$
5. $\{w\#x \mid w, x \in \{0,1\}^* \text{ en } w \text{ is a substring of } x\}$
6. $\{a^i b^j c^k \mid 0 \leq i \leq j \leq k\}$

Answer.

1. **context-free**: We can decide the language with the following context free grammar.
2. **not context-free**: We can prove this using the pumping lemma.

   Proof. Consider the string $s = 0^p\#0^{2p}\#0^{3p}$ with p the unknown pumping length. It is clear that $|s| \geq p$ holds. For every possible breakdown $s = uvxyz$ satisfying conditions (2) and (3) it holds: that v and y can't contain a #: otherwise we would introduce additional # characters by "pumping"

$\langle S\rangle\to\epsilon$
$\langle S\rangle\to\mathbf{0}$
$\langle S\rangle\to\mathbf{1}$
$\langle S\rangle\to\mathbf{0}\langle S\rangle\mathbf{0}$
$\langle S\rangle\to\mathbf{1}\langle S\rangle\mathbf{1}$

Grammatica 5: Context-free grammar to decide $\left\{w\in\{0,1\}^\star \mid w \text{ is a palindrome}\right\}$.

the string, such strings are not part of the language. There are two possible cases: (1) v and y are part of the **same** group of 0's, or (2) they are part of two consecutive groups. In the first case, by "pumping" the string, we will introduce more 0's in that group (since $|vy|>0$), therefore the number of 0's in the groups no longer satisfies the constraint. In the second case, it is possible to make sure that the constraint between the two groups is satisfies, but the remaining group will not comtain more characters if we pump the string. Every breakdown thus leads to contradiction. □

3. **not context-free**: We can prove this using the puming lemma.

   Proof. Consider the string $s=a^pb^pc^p$ with p the pumping length. When we break down s in $s=uvxyz$ such that the last two conditions hold. This means that v en y either contain one type of characters (for instance $v,y\in a^\star$), or two types of consecutive characters, (for instance $u\in aa^\star$ en $y\in a^\star bb^\star$). It is impossible that v en y consists out of the three characters. By puming the string, the amount of at least one type of characters thus won't increase. Therefore the constraint on the language is no longer satisfied thus contradiction. □

4. **context-free**: This language is the intersection of $\left\{x \mid x\in\{0,1\}^\star : \text{even}\left(|x|\right)\right\}$ – a regular language – and $\overline{\left\{ww \mid w\in\{0,1\}^\star\right\}}$, a context-free language. Since the intersection of a regular language and a context-free language is a context-free language, the resulting language is context free.

5. **Not context-free**: We first take the intersection with the regular language $01^\star0^\star1\#01^\star0^\star1$, the resulting language is therefore $L'=\{01^p0^q1\#01^p0^q1 \mid p,q\in\mathbb{N}\}$. Because the intersection of a regular language with a context-free language is a context-free language, it suffices to show that the resulting language is not context-free.

   Proof. Consider the string $s=01^p0^p1\#01^p0^p1$. For every break down of s that satisfies the last two conditions, either v and y belong to the same part (for instance $v,y\in 0^\star$); or two two consecutive parts. In the first case, we generate strings where only the number of 0's increases, Because the first 0 and the last 1 before and after the # can't increase, we need to pick parts for v and y such that s, is repeated p times. Because this segment needs to be repeated before and after # the same amount of times, we cannot pump these components. The same arguments holds for the second case: if u and y are assigned to a different part. □

6. **Not context-free**:

   Proof. Consider the string $s=a^pb^pc^p$ and every possible break down into $s=uvxyz$. Evidently v and y each consists out of a undefined number of the same character: otherwise we would pump several segments of the same character in our string. Because we obtain $s'=uv^2xy^2z$ by pumping once, we know it is impossible that v can contain a's: otherwise, y can only contain a's or b's, and therefore the constraint $j\leq k$ would be broken. Because $|vy|>0$ we know that vy needs to contain at least one c: indeed vy can't contain any a's and if vy would only contain b's, by pumping the string, the number of b's would increase while the number of c's would remain the same. Therefore we would eventually generate more b's than c's.

We can apply the same reasoning the other way: we can pump one time negatively as well. The string $s'' = uxz$ needs to be part of the language as well. Because we know that $|vxy| \leq p$, we know that s'' is not an empty string and s'' needs to contain more c's than b's. Because v and y don't contain any a's, a^p is part of $s'' = uxz$ as well. But this implies that b^p and c^p should be part of s'' as well (since there should be more b's and c's in the string than a's). We therefore conclude that $|vy| = 0$, but that is in contradiction with the second condition 2. □

◇

Exercise 5. A k-push-down automaton is a push-down automaton with k stacks. A 0-PDA is thus an NFA and a 1-PDA a simple PDA. Show that 3-PDA's can decide the same set of languages as 2-PDA's.

Answer.
1. It is trivial that every language that can be decided by a 2-PDA, can be decided by a 3-PDA. By simple ignoring the third stack.
2. A 2-PDA is equivalent with a Turing machine, where the first stack for instance holds the characters left from the head of the Turing machine, and the characters right of the head are stored in the second stack. If the Turing machine thus moves its head to the right, a character of the left stack is popped and pushed on the right stack, etc. We can now define three stacks in terms of a Turing machine. For instance the stacks can be written on to the tape interleaved. In that case we introduce an additional character that represents "empty" for the stack. In case we read empty, we need to move back into the history of the stacks until we reach the first non-empty position.

◇

Exercise 6. A *'right-only Turing machine'* differs from a normal Turing machine such that the head can only stay at the same position, or move to the right. Moving to the left is thus not possible. Show that this variant of Turing machines is not equivalent with normal Turing machines. Which languages can be decides by this variant?

Answer. This variant decides *regular languages.* Since this variant of Turing machines, can't go to the left, we don't have to take into account the characters on the left of the tape. Now we can also eliminate the *stay* instructions (instructions where the head doesn't move to the right). If there are three (possible the same) states s_1, s_2 and s_3 such that there is an instruction $s_1 \rightarrow^{a/b/S} s_2$ and $s_1 \rightarrow^{b/c/R} s_2$ with $a, b, c \in \Gamma$ the tape alphabet, we can remove these two instructions, and introduce an instruction $s_1 \rightarrow^{a/c/R} s_3$. By repeatedly performing such operations, we can eliminate stay instructions. If in the resulting Turing machine, there is still a stay instruction, this means (1) the Turing machine would crash because there is no next instruction that would work; or (2) the machine will get stuck into an infinite loop. In that case, we remove the *stay* instruction and replace it by a move to the right to a *trash state.* A trash state is a state that reads out the characters and moves to the right until it reaches the bank symbols.

Now the Turing machine only contains move to the *right instructions.* For right instructions, it doesn't matter what we write to the tape, since we move to the right: and can't move to the left, we are not able to ever read what we've written. So we can ignore what we write to the tape. Now we have produced a DFA because a DFA moves its cursor to the right as well and per step *eats* a character from the string.

One can also trivially construction a right-only Turing machine that decides the same language as the language decided by a given DFA: simply write an arbitrary character to the tape and move to the right. This shows that the right-only Turing machine and the DFA are equivalent and thus can decide the same set of languages: the regular languages. ◇

Exercise 7. Let $INF_{\text{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) \text{ is an infinite set}\}$. Show that INF_{DFA} is decidable. Is $FIVE_{\text{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) \text{ contains exactly five strings}\}$ decidable? Prove your answer.

Answer. 1. You can construct an algorithm that first minimizes the given DFA such that the unreachable states are no longer accessible. Next one runs a loop-detection algorithm on the DFA (this can be done in quadratic time in the number of states). If there is a loop in the minimized DFA, we can take this loop an arbitrary number of times. Since the loop consists of minimum one character (it is a DFA, not an NFA), each repetition of such loop would result in a new string and thus in an infinite number of possible strings. If there is no loop in the DFA. All paths have a maximum length of n with n the number of states (otherwise there is a loop). In that case the number of strings is evidently finite.

2. An inefficient algorithm first looks whether the language decided by the DFA is finite (see previous item). If the language contains an infinite amount of strings, evidently it does not contain exactly five strings, so we reject. If it contains a finite amount of strings, we can enumerate over all strings with a maximum length of n with n the number of states. If exactly five strings are accepted, we know the language decided by the DFA contains exactly five strings and thus we accept, otherwise we reject.

$\diamond$

Exercise 8. Show that the languages of Turing machines M for which there exists an input such that M will overwrite a non-blank symbol with a blank symbol for that input, is undecidable.

Answer. Proof. We make a reduction from the *emptiness problem* E_{TM} to our problem. We first modify the given Turing machine M to a Turing machine M' by *duplicating the blank symbol*. We introduce a new symbol b that stands for blank. All rules that read the blank symbol from the tape are duplicated by a rule that reads b from the tape as well. All rules that write a blank symbol to the tape now write a b to the tape. In other words the resulting machine can never overwrite a non-blank symbol by a blank symbol, because in that case it writes our alternative symbol to the tape. It furthermore still decides the same language. Now we modify the accepting state. If a string is accepted, the machine will first write a non-blank symbol to the tape and overwrite it with the real blank symbol. The modified machine M' will thus overwrite a non-blank symbol with a blank symbol if and only if it accepts at least one string, which is the complement of E_{TM}. Since we've designed a transformation from E_{TM} to the given problem and E_{TM} is undecidable, we know that the given problem is undecidable as well. $\square$

$\diamond$