

Problem 1

For each of the following languages, give *both* a context-free grammar and a pushdown automaton. Provide a short description of what the variables and rules in your CFG represent, and a short and clear description of how the PDA works.

- (a) $L_1 = \{wyw^R : y, w \in \Sigma^*\}, \Sigma = \{a, b\}$.
- (b) $L_2 = \{a^{2i}b^{3i} : i \geq 0\}, \Sigma = \{a, b\}$.
- (c) $L_3 = \{x : x \text{ has fewer } a\text{s than } b\text{s and } c\text{s together}\}, \Sigma = \{a, b, c\}$.
- (d) $L_4 = \{a^i b^j c^k : i = j \text{ or } j = k \text{ or } i = k\}, \Sigma = \{a, b, c\}$.

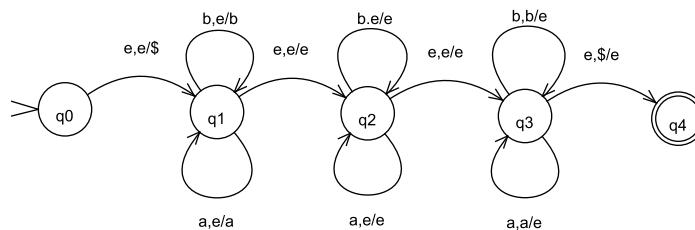
Solution

- (a) This is a “trick question” as the language of L_1 is the set Σ^* of all strings. So L_1 is described by the CFG $S \rightarrow aS \mid bS \mid \varepsilon$. You can also draw a 1-state PDA for L_1 .

Alternatively, you can just follow the pattern in the definition of L_1 and come up with the following CFG and PDA.

$$S \rightarrow aSa \mid bSb \mid Y$$

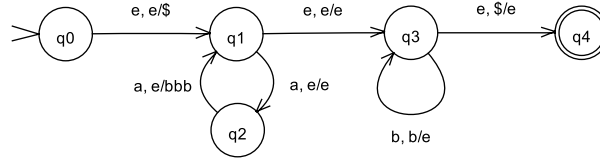
$$Y \rightarrow aY \mid bY \mid \varepsilon$$



- (b) For the CFG, we want to match every pair aa on the left to a triple bbb on the right:

$$S \rightarrow aaSbbb \mid \varepsilon$$

For the PDA, each time we see a pair of as , we push three bs on the stack (this happens in states q_1 and q_2). Later, we just count that the number of pushed bs against the bs from the input.



- (c) To write a CFG, let Y represent a b or a c . Suppose we had a variable E that derived strings with the same number of as and Ys . Strings with one more Y than a are then of the form EYE ; for two more Ys than as , we have $EYEYE$, and so on. To generate all these strings we can use the rule

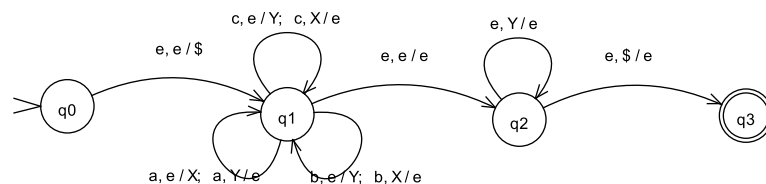
$$S \rightarrow EYS \mid EYE$$

It remains to describe strings with the same number of as and Ys . We gave such a CFG in class but here is another one. If the string starts with an a , then this a can be matched with an Y somewhere so that the remaining segments have the same number of as and Ys . Similarly, if it starts with an Y , this Y can be matched with a later a . So we can generate the strings with the same number of as and Ys via the rule

$$E \rightarrow aEYE \mid YEaE \mid \varepsilon$$

Finally, we add the rule $X \rightarrow b \mid c$.

For the PDA, we use an X on the stack to record the excess number of as , and we use Y to record the excess joint number of bs and cs . When we see an a , we have a choice: Either we can push an X or we can pop a Y (if it is available). When we see a b , we can either pop an X or push a Y . If there are at least as many bs and cs together as there are as , then the number of Ys on the stack should exceed the number of Xs . Moreover, there is always a way to run the PDA so there are always only Ys (zero or more of them) left on the stack. On the other hand, if there are more as than bs and cs together, it is not possible to have only Ys (zero or more) left on the stack. So at the end, the PDA simply checks that all that is left on the stack are Ys .

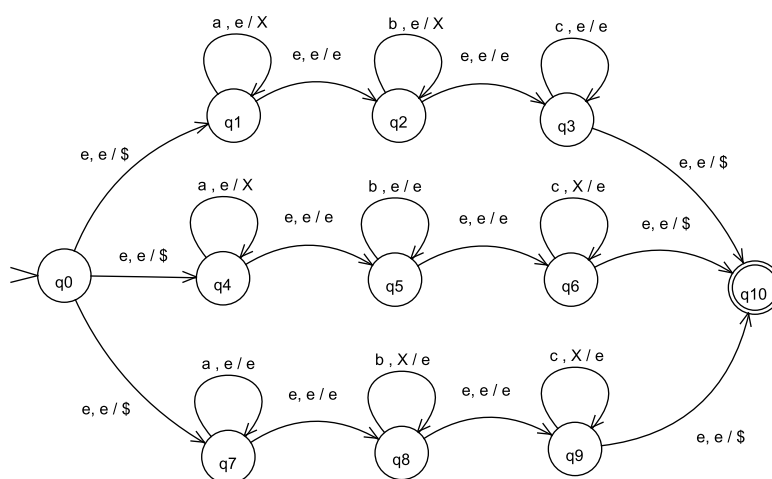


- (d) There are three different possibilities: $i = j$, $j = k$, and $k = i$. A, B, C generate the strings a^*, b^*, c^* respectively. X generates $a^i b^i$, thus XC represents the string $a^i b^i c^k$. Y generates $b^i c^i$, thus AY represents the string $a^k b^i c^i$. Z generates $a^i b^k c^i$. Thus the union of XC, AY, Z

represents L_4 .

$$\begin{aligned}
 S &\rightarrow XC \mid AY \mid Z \\
 X &\rightarrow aXb \mid \varepsilon \\
 Y &\rightarrow bYc \mid \varepsilon \\
 Z &\rightarrow aZb \mid B \\
 A &\rightarrow Aa \mid \varepsilon \\
 B &\rightarrow Bb \mid \varepsilon \\
 C &\rightarrow Cc \mid \varepsilon
 \end{aligned}$$

Similarly, to obtain a PDA, we draw three different PDAs for the three possibilities and connect them up via ε -transitions.



Problem 2

Consider the following context-free grammar G that describes (nontrivial) regular expressions over the alphabet $\{0, 1\}$:

$$R \rightarrow (R) \mid R+R \mid RR \mid R^* \mid 0 \mid 1 \mid \varepsilon$$

The alphabet of G consists of the symbols $(,), +, *, 0, 1,$ and ε . Here $+$ and $*$ describe the union and star operators, while ε describes the empty string.

- Convert G to Chomsky Normal Form.
- Apply the Cocke-Younger-Kasami algorithm (algorithm 2 from lecture 8) to obtain parse trees for the following strings: $(0+1)^*$, $\varepsilon + 0 + 1$, $\varepsilon + 01$. Some of these expressions several parse trees; which ones describe the intended meaning of the expression?
- Give a CFG G' that describes the same language as G but is not ambiguous. Moreover, each parse tree in G' should describe the intended meaning of the corresponding regular expression.

Solution

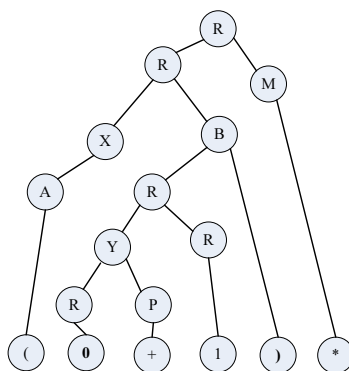
- (a) We notice that there are no unit productions and ε -productions in original CFG. So we just break up long sequences with new variables. We obtain the following grammar:

$$\begin{aligned}
 R &\rightarrow XB \mid YR \mid RR \mid RM \mid 0 \mid 1 \mid e \\
 X &\rightarrow AR \\
 Y &\rightarrow RP \\
 A &\rightarrow (\\
 B &\rightarrow) \\
 P &\rightarrow + \\
 M &\rightarrow *
 \end{aligned}$$

- (b) On input $(0+1)^*$, the run of the CYK algorithm looks like this:

R						
R	-					
X	-	-				
-	R	-	-			
X	Y	-	-	-		
A	R	P	R	B	M	
(0	+	1)	*	

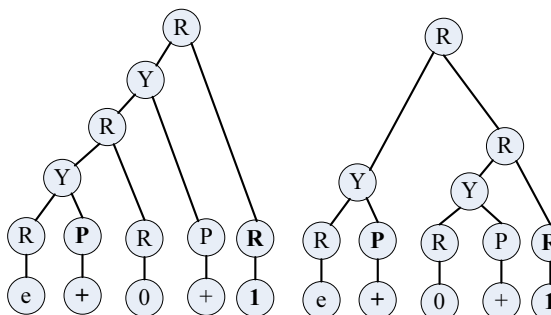
The table yields the following unique parse tree:



On input $e+0+1$, the CYK algorithm does the following:

R					
Y	-				
R	-	R			
Y	-	Y	-		
R	P	R	P	R	
e	$+$	0	$+$	1	

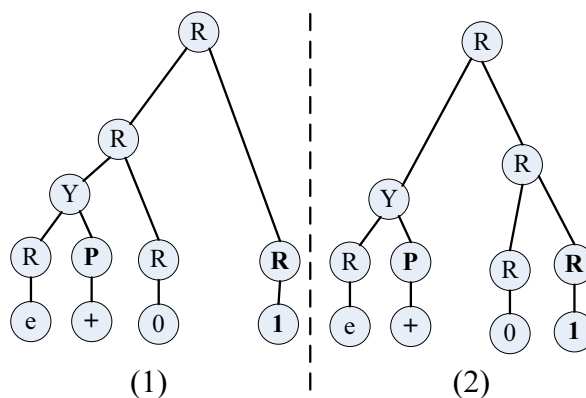
There are two parsing trees for $\epsilon+0+1$, see Fig 6. Both of them describe the intended meaning of the expression.



Finally, this is the run of the CYK algorithm on input $\epsilon+0+1$.

R				
R	-	R		
Y	-	R		
R	P	R	R	
ϵ	+	0	$+$	1

There are two parse for $\epsilon+0+1$. Only the right one describes the intended meaning of the expression. The left one describes the language “first take the union of ϵ and 0 , and then concatenate with 1 ”, which we would write as $(\epsilon+0)1$.



- (c) To disambiguate the grammar and ensure the parse tree has the intended meaning, we have to consider the order of precedence of the operators (union, concatenation, and star). Union has the least precedence, so we write a regular expression R as a union of one or more *terms* (which we represent by T). Next down the line is concatenation, so each term is a concatenation of one or more *factors* (F). Now each factor can be the star of another factor, or it can represent a whole regular expression, provided it is parenthesized. Finally, the “atomic” factors are the

constants 0, 1, and ϵ , which we represent by C .

$$\begin{aligned} R &\rightarrow R + T \mid T \\ T &\rightarrow TF \mid F \\ F &\rightarrow (R) \mid F* \mid C \\ C &\rightarrow 0 \mid 1 \mid \epsilon \end{aligned}$$

Problem 3

Consider the following languages. For each of the languages, say whether the language is (1) regular, (2) context-free but not regular, or (3) not context free. Explain your answer (e.g. give a DFA or argue why one exists, give a CFG, apply the appropriate pumping lemma).

- (a) $L_1 = \{xwyz : w \text{ has length at least 2, } x, y, z, w \in \Sigma^*\}, \Sigma = \{a, b\}$.
- (b) $L_2 = \{xyx^Ry^R : x, y \in \Sigma^*\}, \Sigma = \{a, b\}$.
- (c) $L_3 = \{a^ib^{2i}c^{3i} : i \geq 0\}, \Sigma = \{a, b, c\}$.
- (d) $L_4 = \{x\#y\#z : y^R \text{ is a substring of } xz; x, y, z \in \{a, b\}^*\}, \Sigma = \{a, b, \#\}$.
- (e) $L_5 = \{x\#y\#z : y^R \text{ is a substring of both } x \text{ and } z; x, y, z \in \{a, b\}^*\}, \Sigma = \{a, b, \#\}$.

Solution

- (a) L_1 is regular. To see this, it is convenient to rewrite the definition of L_1 in the following way:

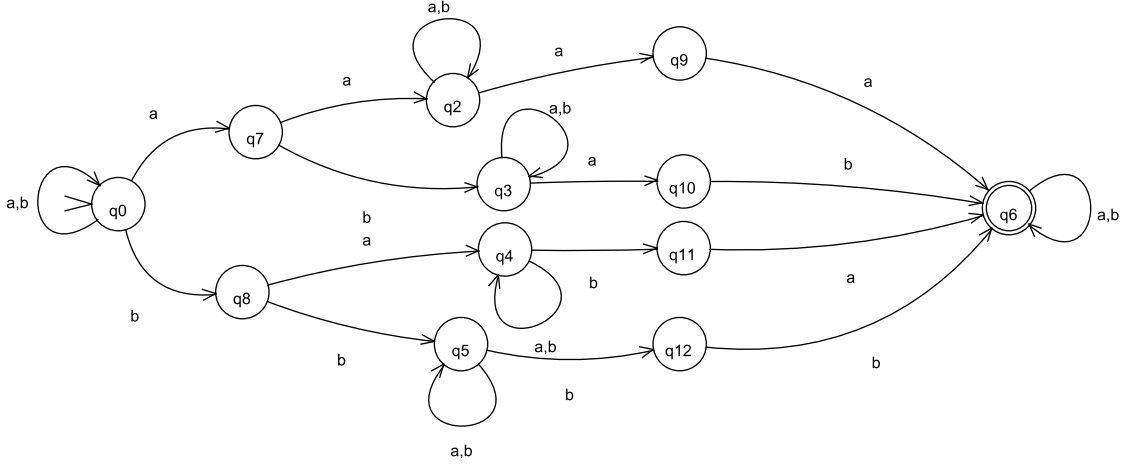
$$L'_1 = \{xwyz : w \text{ has length exactly 2, } x, y, z, w \in \Sigma^*\}$$

Observe that L_1 and L'_1 are the same: If there are two substrings w of length two or more, then in particular there are two substrings of length exactly two; just take the first two symbols in w .

Since there are only four distinct substrings of length two: aa , ab , ba , and bb , we can describe L_1 by the regular expression:

$$\begin{aligned} (a+b)^*aa(a+b)^*aa(a+b)^* &+ (a+b)^*ab(a+b)^*ab(a+b)^* \\ &+ (a+b)^*ba(a+b)^*ba(a+b)^* + (a+b)^*bb(a+b)^*bb(a+b)^*. \end{aligned}$$

Alternatively, we can draw the following NFA:



Yet another way to argue L_1 is regular is via the closure properties. We will argue that the complement $\overline{L_1}$ contains only finitely many strings, so it must be regular, and by closure L_1 is also regular. In fact, $\overline{L_1}$ does not contain any strings of length 10 or more. This is because a string of length 10 or more has at least 5 disjoint blocks of length 2. Since there are only four distinct blocks of length 2 – aa, ab, ba and bb – by the pigeonhole principle two of these blocks must be the same, and so the string is not in L_1 .

(b) L_2 is not context-free. We prove this by pumping lemma. Consider an arbitrary pumping length n , and choose $s = a^n b^n a^n b^n$. Then $s \in L_2$ but we will show that no matter how we write $s = uvwxy$ with $|vwx| \leq n, |vx| > 0$, we can pump it out of L_2 . We look at three cases:

- Case 1: vwx is in the *first half* of $a^n b^n a^n b^n$. We choose $i = 0$, thus uv^0wx^0y looks like $a^i b^j a^n b^n$ where $i < n$ or $j < n$ (or both). This is not of the form $xyx^R y^R$.
- Case 2: vwx is in the *middle part* of $a^n b^n a^n b^n$ (not intersecting the initial block of as and the final block of bs). We choose $i = 0$, thus uv^0wx^0y looks like $a^n b^i a^j b^n$ where $i < n$ or $j < n$ (or both). This is not the form $xyx^R y^R$.
- Case 3: vwx is in the *second half* of $a^n b^n a^n b^n$. We choose $i = 0$, thus uv^0wx^0y looks like $a^n b^n a^i b^j$ where $i < n$ or $j < n$ (or both). This is not the form $xyx^R y^R$.

This covers all the cases, so by the pumping lemma for context-free languages, L_2 is not context-free.

(c) L_3 is not context-free. We prove this by the pumping lemma. Let n be the pumping length and $s = a^n b^{2n} c^{3n}$, which is in L_3 . Now consider any splitting of s as $s = uvwxy$ with $|vwx| \leq n, |vx| > 0$. Since $|vwx| \leq n$, vwx can cover at most two symbols of $a^n b^{2n} c^{3n}$. We choose $i = 2$, the 1 : 2 : 3 ratio between as , bs , and cs cannot be maintained after pumping. So L_3 is not context-free.

(d) L_4 is not context-free. We prove this by the pumping lemma. Let n be the pumping length and $s = a^n \# b^n a^n \# b^n$, which is in L_4 . Now consider any splitting of s as $s = uvwxy$ with $|vwx| \leq n, |vx| > 0$.

- Case 0: If v or x contains a $\#$, then uv^2wx^2y has more than two $\#$ s, so it is not in L_4 .
- Case 1: $vw x$ is contained in the first half of s (the first $a^n \# b^n$). If vx contains any a s, we pump down: Then the string $uv^0wx^0y = uwy$ is of the form $a^i \# b^j a^n \# b^n$ with $i < n$, and it is not in L_4 , because now the middle part has fewer a s than the left and right parts (so its reverse cannot be a substring). If not, then vx contains some b s, and we pump up once: The string uv^2wx^2y is now of the form $a^i \# b^j a^n \# b^n$ with $j > n$, and now the middle part has more b s than the left and right parts, so uv^2wx^2y is not in L_4 again.
- Case 2: $vw x$ is contained in the second half of s (the second $a^n \# b^n$). The analysis is completely analogous to case 1. If anything in the last block is in vx we pump down, otherwise we pump up. In either case we end up outside L_4 .
- Case 3: $vw x$ is contained in the middle part of $vw x$ (the part between the two $\#$ s). Then after pumping up once, the string uv^2wx^2y has a middle part which is longer than the left and right parts together, so the reverse of the middle cannot be a substring of the left and right. So $uv^2wx^2y \notin L_4$.

This covers all the cases, so L_4 is not context-free.

- (e) L_5 is not context-free. We prove this by pumping lemma. Let n be the pumping length and $s = a^n \# a^n \# a^n$, which is in L_5 . Now consider any splitting of s as $s = uvwx y$ with $|vw x| \leq n$, $|vx| > 0$.

- Case 0: If v or x contains a $\#$, then uv^2wx^2y has more than two $\#$ s, so it is not in L_5 .
- Case 1: If either v or x is nonempty and intersects the middle block a^n , we pump up once: The string uv^2wx^2y has a middle block whose length is more than n . However, the substring $vw x$ cannot intersect both the left block a^n and the right block a^n . Therefore, in uv^2wx^2y at least one of these two blocks remains of the same length n . So the middle block is not a substring of it, and uv^2wx^2y is not in L_5 .
- Case 2: If $vw x$ is contained in the leftmost block a^n , then we pump down once, and notice that uv^0wx^0y is of the form $a^i \# a^n \# a^n$, where $i < n$. This is not in L_5 .
- Case 3: If $vw x$ is contained in the rightmost block a^n , by an analogous argument, $uv^0wx^0y \notin L_5$.

This covers all the cases, so L_5 is not context-free.

Problem 4

Context-free grammars are sometimes used to model natural languages. In this problem you will model a fragment of the English language using context-free grammars. Consider the following English sentences:

The girl is pretty.

The girl that the boy likes is pretty.

The girl that the boy that the clerk pushed likes is pretty.

The girl that the boy that the clerk that the girl knows pushed likes is pretty.

This is a special type of sentence built from a subject (**The girl**), a relative pronoun (**that**) followed by another sentence, a verb (**is**) and an adjective (**pretty**).

- (a) Give a context-free grammar G that models this special type of sentence. Your terminals should be words or sequences of words like **pretty** or **the girl**.
- (b) Is the language of G regular? If so, write a regular expression for it. If not, prove using the pumping lemma for regular languages.
- (c) Can you give an example of a sentence that is in G but does not make sense in common English?

Solution

- (a) The following grammar G describes this type of sentence:

$$\begin{aligned}
 \text{SENTENCE} &\rightarrow \text{SUBJ REL VERB ADJ} \\
 \text{REL} &\rightarrow \text{that SUBJ REL VERB} \mid \epsilon \\
 \text{SUBJ} &\rightarrow \text{the girl} \mid \text{the boy} \mid \text{the clerk} \\
 \text{VERB} &\rightarrow \text{is} \mid \text{likes} \mid \text{pushed} \mid \text{knows} \\
 \text{ADJ} &\rightarrow \text{pretty}
 \end{aligned}$$

- (b) This language is not regular. We prove this via the pumping lemma for regular languages. For any given n , the string

the girl (that the boy) ^{n} (knows) ^{n} is pretty

is in $L(G)$. It can be generated via the recursive rule for REL . Let uvw be any splitting of z where $|uv| \leq n, |v| > 0$. If v consists only of a single copy of “**that**”, then uv^2w is not in $L(G)$ since it contains the pattern “**that that**”, which is not allowed by the rules. Otherwise v contains at least one subject (“**the girl**” or “**the boy**”) but no verb. Then uv^2w has more subjects than verbs, which is not allowed by the rules of the grammar, where every subject must have a matching verb.

- (c) You can give different kinds of examples. For instance, “**The girl pushed pretty**” does not make sense in English. Neither does “**The boy that the boy is is pretty**”. There are also examples that make grammatical sense, but we would hardly use in practice, like

The girl that the boy that the clerk that the girl knows pushed likes is pretty.

This illustrates some of the difficulties in trying to design formal grammars for natural languages.