Practice 4

Theory of languages and machines

Dr. movaghar

Sara Azarnoush

98170668

Contents

1

1.1

a) $L = \{a^n b^m \mid n \neq 2m\}$

can be either

- A string with too many *a*`s, or
- A string with too few *a*'s

$S \to A|B$

$A \to$ justA TwoB

$B \to$ TwoB justB

justA $\to$ a justA |a

justB $\to$ b justB |b

TwoB $\to$ a TwoB bb

b) $L = \{\omega \in \{a, b\}* \mid \forall\ v \in Pref(\omega)\ .\ n_a(v) \geq n_b(v)\}$

$S \to aS|abS|\ \varepsilon$

c) $L = \{\omega \in \{a, b\}* \mid n_a(\omega) = 2n_b(\omega) + 1\}$

$S \to bSaa|baSa|baaS|Sbaa|aSab|aaSb|aabS|Saab|aSba|abSa|abaS|Saba$

1.2

a) $L = \{a^n b^n \mid n \geq 1\}$

$S \to aX$

$X \to aXb|b$

b) $L = \{a^n b^{n+1} \mid n \geq 2\}$

$S \to aXb$

$X \to aYb$

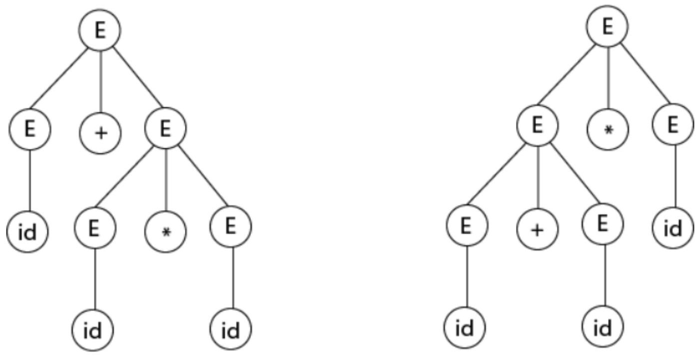
$Y \to aYb|b$

B) both. We can only make one choice at each step of the parsing process.

As a result, they are unambiguous.

1.3

a)



1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow id$

b)

1.4

2

2.1

1)

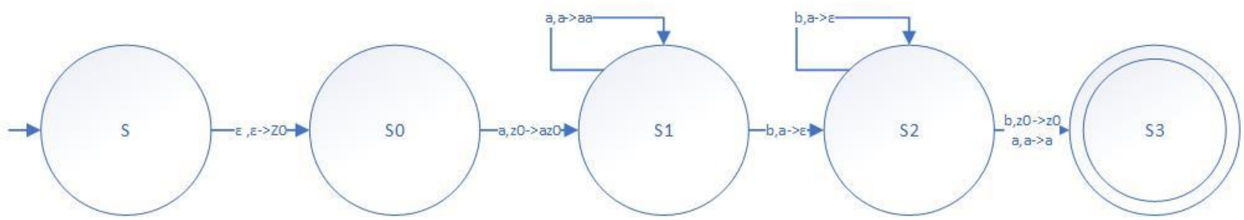
$L = \{x\omega x : \omega \in \{a, b\}* \wedge x \in \{a, b\}\}$

2)

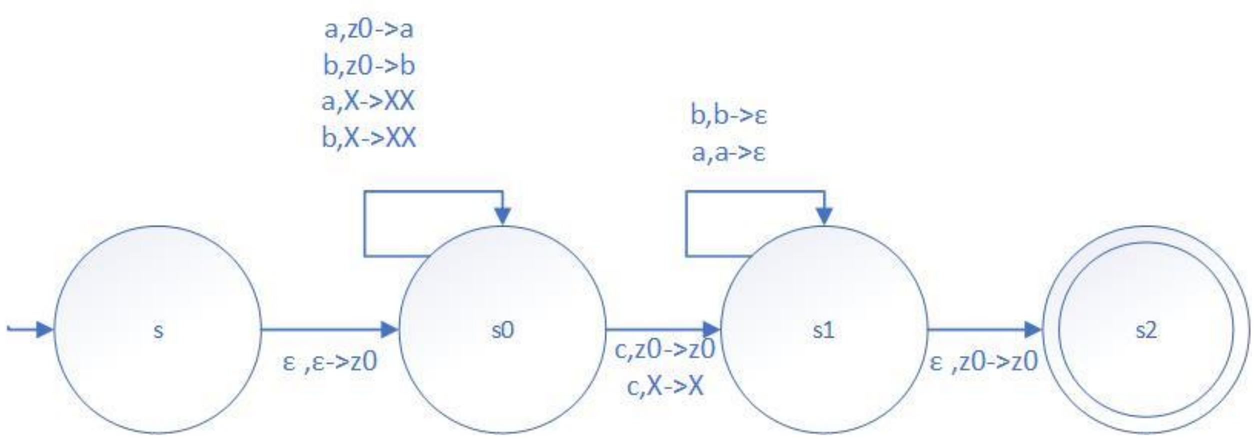
$L = \{\omega c\omega' : \omega, \omega' \in \{a, b\}* \wedge |\omega| = |\omega'|\}$

2.2

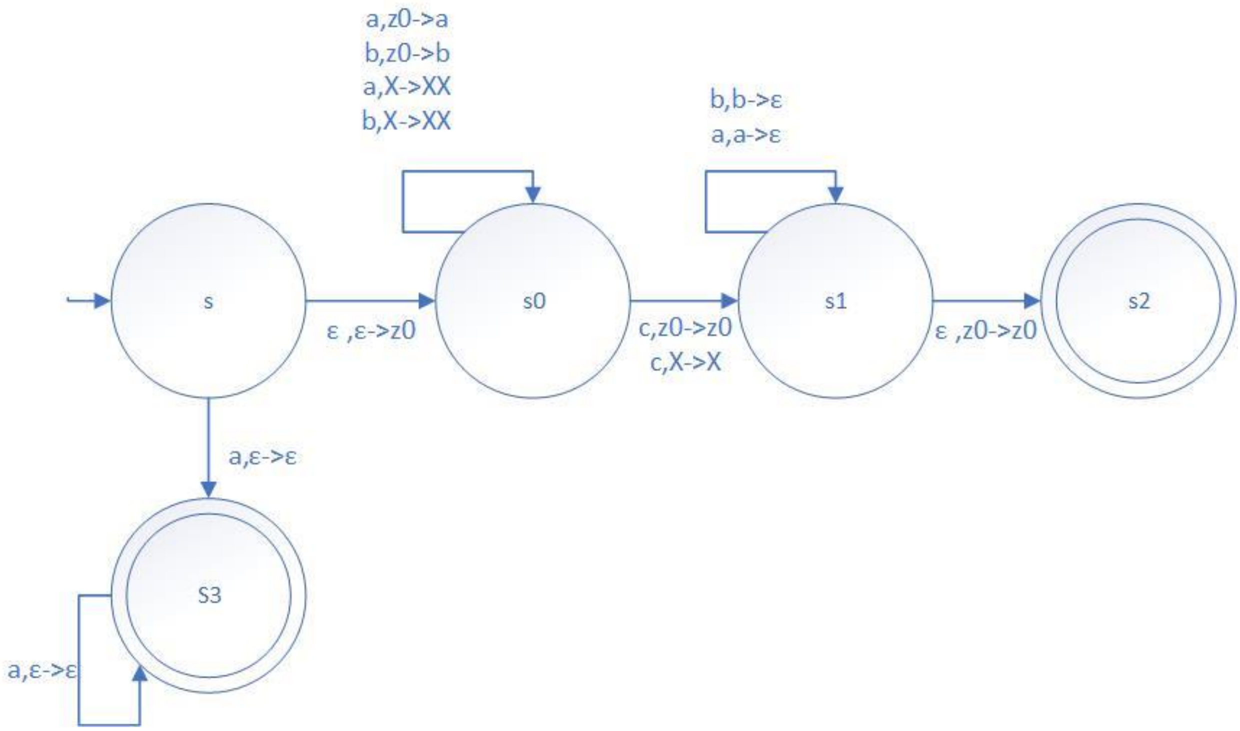
a) $L1 = \{a^n b^m : n > 0, n \neq m\}$



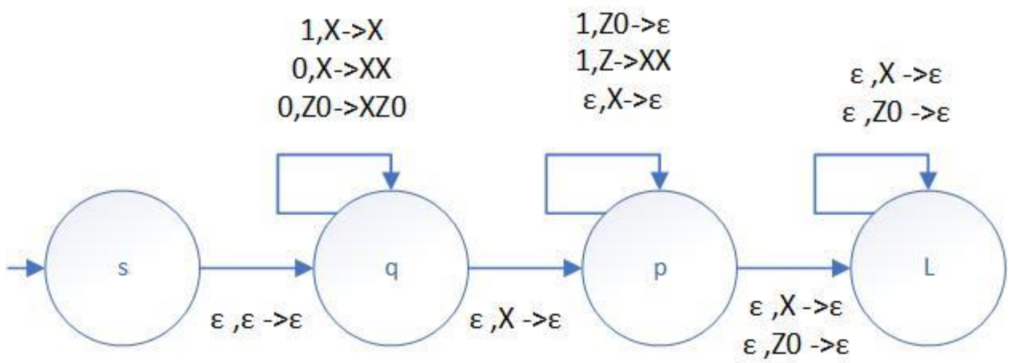
b) $L2 = \{w1cw2 : w1, w2 \in \{a, b\} * , w1 \neq w2^R \}\}$

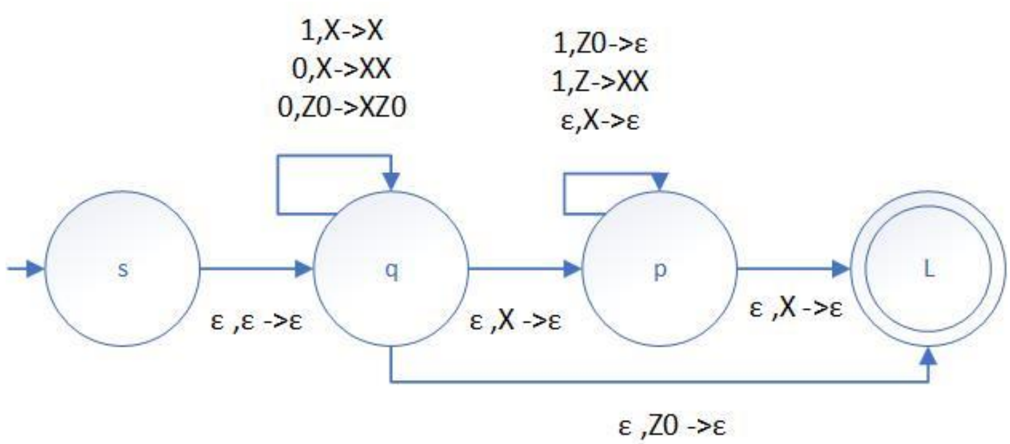


c) L3 = concatenation of L(a∗) and L2


a,z0->a
b,z0->b
a,X->XX
b,X->XX
b,b->ε
a,a->ε
s
s0
s1
s2
ε ,ε->z0
c,z0->z0
c,X->X
ε ,z0->z0
a,ε->ε
S3
a,ε->ε

2.3


1,X->X
0,X->XX
0,Z0->XZ0
1,Z0->ε
1,Z->XX
ε,X->ε
ε ,X ->ε
ε ,Z0 ->ε
s
q
p
L
ε ,ε ->ε
ε ,X ->ε
ε ,X ->ε
ε ,Z0 ->ε


1,X->X
0,X->XX
0,Z0->XZ0
1,Z0->ε
1,Z->XX
ε,X->ε
s
q
p
L
ε ,ε ->ε
ε ,X ->ε
ε ,X ->ε
ε ,Z0 ->ε

2.4

a)

there is a DFA for every regular language L. so we convert DFA to PDA. In each transition, push and pop a fixed symbol. We can create a deterministic PDA that is equivalent to the supplied DFA by making this alteration to the present DFA.

b)

For any input and stack symbol, we add a new trap state with all of its output transitions to itself. Then we add a new transition to the trap state with input symbol # for each non-final state of the old automata. For each end state of the previous automaton, we must also add a transition to itself using the symbol #.

If x is in the old language, the new automaton will proceed to the next state, which will parse the xy string in the old language. if x is not in the old language, it will move to the trap state when it sees the symbol # in the string x#y.

3

3.1

a) L = {a^k | k is a prime}

Assume L is context-free and is generated by a context-free grammar G.

some constant p dependent on G such that for all strings w in L of length at least p the Pumping

Choose w= a^k for some prime number $k \geq p$.

$w = a^i\, a^j\, a^r\, a^s$ at where $k = i + j + r + s + t$, $j + s \geq 1$, $v=a^j$, $y= a^s$.

Pumping n+1 times:

$a^i\, (a^j\,)^{n+1}\, a^r\, (a^s)^{n+1}\, a^t = a^i\, a^r\, a^t\, (a^j\,)^{n+1}\, (a^s)^{n+1}= a^{k-j-s\, +\, (j+s)(n+1)}$

Let x= j+s.

Pumping n+1 times yields: a^{k+xn}

$w = a^i\, a^j\, a^r\, a^s\, a^t$

Let x= j+s.

Pumping n+1 times yields: a^{k+xn}

Pump k+1 times: $a^{k+xk} = a^{k(x+1)}$

Since x >= 1, (x+1) >= 2 and so k (x+1) Cannot be prime.

L is not context-free.

b) L = {a^{2n} | n ≥ 0}

Assume L is context-free

$s = a^{2p} \in L$

$s = a^{2p-2}\{z\}a\{v\}\varepsilon\{x\}\varepsilon\{y\}a\{z\}z \in L$

$s' = uv^2xy^2z$

$|s'| > |s|$

$|s'|$ isn't $2^n \Rightarrow s' \notin L$

L is not context-free.

c) L = {$a^n b^{2n} a^n$ | n ≥ 0}

Assume L is context-free

$s = a^p b^{2p} a^p$

$s = a^p\{u\}b\{v\}b\{x\}b\{y\}b^{2p-3}a^p\{z\} \in L$

$s' = a^p b^2 b b^2 b^{2p-3} a^p$

$s' = a^p b^{2p+2} a^p$

$2p + 2 \neq 2p \Rightarrow s' \notin L$

L is not context-free.

d) L = {ω ∈ {a, b, c} ∗ | na(ω) = max {nb(ω), nc(ω)}}

3.2

The class DCFL is not closed under concatenation

L1={aibjck|i≠j} and L2={$a^i b^j c^k$|j≠k}; both are DCFL and L3=0L1∪L2 is DCFL, too.

L0=0∗ is DCFL (regular) But L_{conc}=L0·L3=0∗L3 is not DCFL.

Suppose that L_{conc} (which is the concatenation of two DCFLs) is DCFL.

If we intersect L_{conc} with the regular language $0a^*b^*c^*$, we should get a DCFL language:

$L_{conc} \cap \{0a^*b^*c^*\} = 0L1 \cup 0L2$.

suppose $0L1 \cup 0L2$ is a DCFL, so $L1 \cup L2$ should be a DCFL, too but:

$L1 \cup L2 = (L1' \cap L2')' = (\{a^i b^i c^i\})'$

which is not DCFL ⇒ contradiction.

The class DCFL is closed under complement
Let M = (Q, Σ, Γ, δ, q0 , Z 0 , F) be a DPDA such that L(M) = L, and assume that M always reads its input completely.
Unfortunately, L^c does in general not coincide with the language accepted by the DPDA M' = (Q, Σ, Γ, δ, q0 , Z 0 , Q\ F), as M may have a computation of the following form:

$(q0, Z0, w) \vdash^*_M (q, \alpha, \varepsilon) \vdash_M (q', \beta, \varepsilon)$ for some $q \in F$ and $q' \notin F$

The class DCFL is not closed under union

If they were, then, by DeMorgan's Laws, using closure under complementation, they would have to be closed under intersection.
(The class DCFL is closed under intersection with regular languages.
Let L1 ∈ DCFL. The there exists a DPDA M that accepts L1 and that reads each input completely. For L2 ∈ REG, there exsists a DFA A such that L(A) = L2.
From M and A, one can construct a DPDA for L1 ∩ L2 , that is, L1 ∩ L2 ∈ DCFL.)

3.3
A)

Let P be the PDA that accepts L, and let M be the DFA that accepts R. A new PDA P′ will simulate P and M simultaneously on the same input and accept if both accept. Then P′ accepts L ∩ R.
- The stack of P′ is the stack of P
- The state of P′ at any time is the pair (state of P , state of M)
- These determine the transition function of P′
- The final states of P′ are those in which both the state of P and state of M are

accepting.
More formally, let $M = (Q1, \Sigma, \delta 1, q1, F1)$ be a DFA such that $L(M) = R$, and $P = (Q2, \Sigma, \Gamma, \delta 2, q2, F2)$
be a PDA such that $L(P) = L$. Then consider $P' = (Q, \Sigma, \Gamma, \delta, q0, F)$ such that

- $Q = Q1 \times Q2$
- $q0 = (q1, q2)$
- $F = F1 \times F2$
- $\delta((p, q), x, a) = \{((p', q'), b) \mid p' = \delta 1(p, x) \text{ and } (q', b) \in \delta 2(q, x, a)\}$.

One can show by induction on the number of computation steps, that for any $w \in \Sigma*$

$\langle q0, \varepsilon \rangle \rightarrow P' \langle (p, q), \sigma \rangle$ iff $q1 \rightarrow M\ p$ and $\langle q2, \varepsilon \rangle \rightarrow P \langle q, \sigma \rangle$

Now as a consequence, we have $w \in L(P')$
iff $\langle q0, \varepsilon \rangle \rightarrow P' \langle (p, q), \sigma \rangle$ such that $(p, q) \in F$ (by definition of PDA acceptance) iff $\langle q0, \varepsilon \rangle \rightarrow P' \langle (p, q), \sigma \rangle$ such that $p \in F1$ and $q \in F2$ (by definition of F) iff $q1 \rightarrow M\ p$ and $\langle q2, \varepsilon \rangle \rightarrow P \langle q, \sigma \rangle$ and $p \in F1$ and $q \in F2$ (by the statement to be proved as exercise) iff $w \in L(M)$ and $w \in L(P)$ (by definition of DFA acceptance and PDA acceptance)

B)
An equivalent notation for context free languages is Backus Naur Form (BNF). In BNF the set of palindromes over {a, b} can be denoted as follows.
<palindromes> ::= <empty> | a | b | a<palindromes>a | b<palindromes>b
set of palindromes. The base cases are λ, a, and b. The recursive cases are that if we have a palidrome s, then s with an a concatenated at each end is a palindrome, and s with a b concatentated at each end is a palindrome.
As for a context free grammar, a BNF grammar has a finite set of terminal symbols, a finite set of nonterminal symbols, a start symbol (one of the nonterminal symbols), and a finite set of rules. Each rule has a lefthand side, which is one of the nonterminal symbols, and a righthand side, which is a finite string of terminal and nonterminal symbols, possibly empty (which we denoted by <empty> above.) The lefthand and righthand sides are separated by ::=.
In terms of the example above, the set of terminal symbols is {a, b}, the set of nonterminal symbols is {< palindromes >}, the start symbol is < palindromes >, and there are five rules:
<palindromes> ::= <empty>
<palindromes> ::= a

<palindromes> ::= b
<palindromes> ::= a<palindromes>a
<palindromes> ::= b<palindromes>b
There is a convention to abbreviate several rules with the same lefthand side by separating the different righthand sides with the symbol |, as shown above. (This convention is also used for context free grammars in standard linguistics notation.)

($S \rightarrow aSa|bSb|\varepsilon$)

3.4

a) $L = \{a^p b^q c^r d^s \mid p = 0 \text{ or } q = r = s\}$

C chooses an integer m≥0
N chooses the string st∈L: $ab^m c^m d^m$. |st|=3m+1>m. Mark the last mm positions in st, so that the first letter a is always not marked.
C chooses strings u,v,x,y,z where st=uvxyz, such that:

1. Vy has at least one marked position.
2. vxy has at most mm marked positions.

N can choose the integer i=2, and $uv^i xy^i z \notin L$. Proof:

1. C cannot choose v or y that involves more than one type of letter. Otherwise, the letter arrangement of the pumped string will be out of order, so the new string will not be ∈L
2. C cannot choose to pump letter b or c or d. Because C can only choose two types of letter, the resulting pumped string will not have equal length of b,c,d, and will not be ∈L.
3. Therefore, C can only choose to pump the first letter a to keep the resulting string valid. Therefore, C has two potential choices: v=a, y=ϵ; or v=ϵ, y=a. However, because a is not marked, neither choice satisfies the condition: vy has at least one marked position. Therefore, there is no choice that keeps the pumped string in L.

Now we can claim L is not context-free since such p doesn't exist in the Ogden's lemma.

b) $L = \{a^n b^n c^i \mid i \neq n\}$

choose $z = a^n b^n c^{n!+n}$ (where n is the constant from Ogden's lemma).

if v or x contain a mix of a's and b's, then see that with $i = 2$, the structure of the resulting grammar is no longer correct. where $v = a^{\alpha}$ and $x = b^{\beta}$. If $\alpha \neq \beta$ then can also see that the number of a's and b's will be different, therefore $\alpha = \beta$. We can now call $\gamma = \alpha = \beta$ and see that our final string will be of the form $a^{n+\gamma(i-1)}b^{n+\gamma(i-1)}c^{n!+n}$ Therefore, if we set the exponents of a or b equal to c, get:

$n + \gamma(i - 1) = n! + n$

$\gamma(i - 1) = n!$

$i - 1 = n!/\gamma$

Since $\gamma \leq n$ we know that the right side divides evenly and therefore we can pick an i that satisfies this constraint, therefore our original constraint is not satisfied, therefore we do not have a CFL.