Tutorial No. :2
CFG, PDA

1. Construct context-free grammars that generate each of these languages:
   (a) $\{wcw^R : w \in \{a, b\}^*\}$
   (b) $\{ww^R : w \in \{a, b\}^*\}$
   (c) $\{w \in \{a, b\}^* : w = w^R\}$

Solutions

(a) $G = (V, \Sigma, R, S)$ with $V = \{S, a, b, c\}$, $\Sigma = \{a, b, c\}$, $R = \{$
$S \rightarrow aSa$
$S \rightarrow bSb$
$S \rightarrow c$ $\}$.

(b) Same as (a) except remove c from V and Σ and replace the last rule, $S \rightarrow c$, by $S \rightarrow \varepsilon$.

(c) This language very similar to the language of (b). (b) was all even length palindromes; this is all palindromes. We can use the same grammar as (b) except that we must add two rules:
$S \rightarrow a$
$S \rightarrow b$

2. Consider the grammar $G = (V, \Sigma, R, S)$, where
$V = \{a, b, S, A\}$,
$\Sigma = \{a, b\}$,
$R = \{ \; S \rightarrow AA,$
$A \rightarrow AAA, \quad A \rightarrow a, \; A \rightarrow bA, A \rightarrow Ab$
$\}$.
   (a) Which strings of L(G) can be produced by derivations of four or fewer steps?
   (b) Give at least four distinct derivations for the string babbab.
   (c) For any m, n, p > 0, describe a derivation in G of the string $b^m ab^n ab^p$.

Solutions

(a) We can do an exhaustive search of all derivations of length no more than 4:

$S \Rightarrow AA \Rightarrow aA \Rightarrow aa$

$S \Rightarrow AA \Rightarrow aA \Rightarrow abA \Rightarrow aba$

$S \Rightarrow AA \Rightarrow aA \Rightarrow aAb \Rightarrow aab$

$S \Rightarrow AA \Rightarrow bAA \Rightarrow baA \Rightarrow baa$

$S \Rightarrow AA \Rightarrow bAA \Rightarrow bAa \Rightarrow baa$

$S \Rightarrow AA \Rightarrow AbA \Rightarrow abA \Rightarrow aba$

$S \Rightarrow AA \Rightarrow AbA \Rightarrow Aba \Rightarrow aba$

$S \Rightarrow AA \Rightarrow Aa \Rightarrow aa$

$S \Rightarrow AA \Rightarrow Aa \Rightarrow bAa \Rightarrow baa$

$S \Rightarrow AA \Rightarrow Aa \Rightarrow Aba \Rightarrow aba$

$S \Rightarrow AA \Rightarrow AbA \Rightarrow abA \Rightarrow aba$

$S \Rightarrow AA \Rightarrow AbA \Rightarrow Aba \Rightarrow aba$

$S \Rightarrow AA \Rightarrow AAb \Rightarrow aAb \Rightarrow aab$

$S \Rightarrow AA \Rightarrow AAb \Rightarrow Aab \Rightarrow aab$

Many of these correspond to the same parse trees, just applying the rules in different orders. In any case, the strings that can be generated are: aa, aab, aba, baa.

(b) Notice that $A \Rightarrow bA \Rightarrow bAb \Rightarrow bab$, and also that $A \Rightarrow Ab \Rightarrow bAb \Rightarrow bab$. This suggests 8 distinct derivations:

$S \Rightarrow AA \Rightarrow AbA \Rightarrow AbAb \Rightarrow Abab \Rightarrow^* babbab$

$S \Rightarrow AA \Rightarrow AAb \Rightarrow AbAb \Rightarrow Abab \Rightarrow^* babbab$

$S \Rightarrow AA \Rightarrow bAA \Rightarrow bAbA \Rightarrow babA \Rightarrow^* babbab$

$S \Rightarrow AA \Rightarrow AbA \Rightarrow bAbA \Rightarrow babA \Rightarrow^* babbab$

Where each of these four has 2 ways to reach babbab in the last steps. And, of course, one could interleave the productions rather than doing all of the first A, then all of the second A, or vice versa.

(c) This is a matter of formally describing a sequence of applications of the rules in terms of m, n, p that will produce the string $b^m ab^n ab^p$.

S

$\Rightarrow$ /* by rule $S \rightarrow AA$ */

AA

$\Rightarrow^*$ /* by m applications of rule $A \rightarrow bA$ */

$b^m AA$

$\Rightarrow$ /* by rule $A \rightarrow a$ */

$b^m aA$

$\Rightarrow^*$ /* by n applications of rule $A \rightarrow bA$ */

$b^m ab^n A$

$\Rightarrow^*$ by p applications of rule $A \rightarrow Ab$ */

$b^m ab^n Ab^p$

$\Rightarrow$ /* by rule $A \rightarrow a$ */

$b^m ab^n ab^p$

Clearly this derivation (and some variations on it) produce $b^m ab^n ab^p$ for each m, n, p.

3. Consider the alphabet $\Sigma = \{a, b, (,), \cup, *, \varnothing\}$. Construct a context-free grammar that generates all strings in Σ^* that are regular expressions over {a, b}.

Solutions

3. This is easy. Recall the inductive definition of regular expressions that was given in class :

1. $\varnothing$ and each member of Σ is a regular expression.
2. If α , β are regular expressions, then so is $\alpha\beta$
3. If α , β are regular expressions, then so is $\alpha \cup \beta$.
4. If α is a regular expression, then so is α^*.
5. If α is a regular expression, then so is (α).
6. Nothing else is a regular expression.

This definition provides the basis for a grammar for regular expressions:

G = (V, Σ, R, S) with V = {S, a, b, (,), ∪, *, ∅}, Σ = { a, b, (,), ∪, *, ∅}, R= {

$S \rightarrow \varnothing$	/* part of rule 1, above
$S \rightarrow a$	/* "
$S \rightarrow b$	/* "
$S \rightarrow SS$	/* rule 2
$S \rightarrow S \cup S$	/* rule 3
$S \rightarrow S^*$	/* rule 4
$S \rightarrow (S)$	/* rule 5 }

4. Consider the following grammar (the start symbol is S; the alphabets are implicit in the rules):

$S \rightarrow SS \mid AAA \mid \varepsilon$

$A \rightarrow aA \mid Aa \mid b$

(a) Describe the language generated by this grammar.

(b) Give a left-most derivation for the terminal string abbaba.

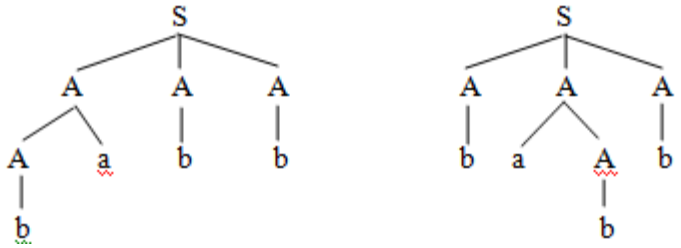
(c) Show that the grammar is ambiguous by exhibiting two distinct derivation trees for some terminal string.

Solution

(a) (a*ba*ba*ba*)*

(b) S ⇒ AAA ⇒ aAAA ⇒ abAA ⇒ abAaA ⇒ abbaA ⇒ abbaAa ⇒ abbaba

(c)



5. Given a grammar G with production rules

S →aB , S →bA ,

A→ aS , A →bAA , A →a

B→ bS , B →aBB , B →b

Obtain the (i) leftmost derivation, and (ii) rightmost derivation for the string "aaabbabbba".

Solutions

(i) *Leftmost derivation:*

$$S \Rightarrow aB \Rightarrow aaBB \Rightarrow aaaBBB \Rightarrow aaabBB \Rightarrow aaabbB$$
$$\Rightarrow aaabbabB \Rightarrow aaabbabbB \Rightarrow aaabbabbbS \Rightarrow aaabbabbba$$
$$\Rightarrow aaabbabb$$

(ii) *Rightmost derivation:*

$$S \Rightarrow aB \Rightarrow aaBB \Rightarrow aaBbS \Rightarrow aaBbbA \Rightarrow aaaBBbba$$
$$\Rightarrow aaabBbba \Rightarrow aaabbSbba \Rightarrow aaabbaBbba \Rightarrow aaabbabbba$$

6. Show that the grammar G with production

S →a /aAb/ abSb

A →aAAb /bS is ambiguous.

Solutions

$$S \Rightarrow abSb \qquad (\because S \rightarrow abSb)$$
$$\Rightarrow abab \qquad (\because S \rightarrow a)$$

Similarly,

$$S \Rightarrow aAb \qquad (\because S \rightarrow aAb)$$
$$\Rightarrow abSb \qquad (\because A \rightarrow bS)$$
$$\Rightarrow abab$$

Since '*abab*' has two different derivations, the grammar *G* is ambiguous.

7. Consider the language L = {w ∈ {a, b}* : w contains equal numbers of a's and b's}
   a) Write a context-free grammar G for L.
   b) Show two derivations (if possible) for the string aabbab using G. Show at least one leftmost derivation.
   c) Do all your derivations result in the same parse tree? If so, see if you can find other parse trees or convince yourself there are none.
   d) If G is ambiguous (i.e., you found multiple parse trees), remove the ambiguity. (Hint: look out for two recursive occurrences of the same nonterminal in the right side of a rule, e.g, X → XX)
   e) See how many parse trees you get for aabbab using the grammar developed in (d).

Solutions

(a) G = (V, Σ, R, S) with V = {S }, Σ = {a, b }, R = {

S → aSb

S → bSa

S → ε

S → SS }

(b) (i) S ⇒ SS ⇒ aSbS ⇒ aaSbbS ⇒ aabbaSb ⇒ aabbab /* This is the leftmost derivation of the most "sensible" parse.

(ii) S ⇒ SS ⇒ SSS ⇒ aSbSS ⇒ aaSbbSS ⇒ aabbSS ⇒ aabbaSbS ⇒ aabbabS ⇒ aabbab /* This is the leftmost derivation of a parse that introduced an unnecessary S in the first step, which was then eliminated by rewriting it as ε in the final step.

(c) No. The two derivations shown here have different parse trees. They do, however, have the same bracketing, [a[ab]b][ab].(In other words, they have similar essential structures.) They differ only in how S is introduced and then eliminated. But there are other derivations that correspond to additional parse

trees, and some of them correspond to a completely different bracketing, [a[ab][ba]b]. One derivation that does this is

(ii) $S \Rightarrow aSb \Rightarrow aSSb \Rightarrow aabSb \Rightarrow aabbab$

(d) This is tricky. Recall that we were able to eliminate ambiguity in the case of the balanced parentheses language just by getting rid of ε except at the very top to allow for the empty string. If we do the same thing here, we get R = { $S \rightarrow \varepsilon$

$S \rightarrow T$

$T \rightarrow ab$

$T \rightarrow aTb$

$T \rightarrow ba$

$T \rightarrow bTa$

$T \rightarrow TT$

But aabbab still has multiple parses in this grammar. This language is different from balanced parens since we can go back and forth between being ahead on a's and being ahead on b's (whereas, in the paren language, we must always either be even or be ahead on open paren). So the two parses correspond to the bracketings [aabb][ab] and [a [ab] [ba] b]. The trouble is the rule $T \rightarrow TT$, which can get applied at the very top of the tree (as in the case of the first bracketing shown here), or anywhere further down (as in the case of the second one). We clearly need some capability for forming a string by concatenating a perfectly balanced string with another one, since, without that, we'll get no parse for the string abba. Just nesting won't work. We have to be able to combine nesting and concatenation, but we have to control it. It's tempting to think that maybe an unambiguous grammar doesn't exist, but it's pretty easy to see how to build a deterministic pda (with a bottom of stack symbol) to accept this language, so there must be an unambiguous grammar. What we need is the notion of an A region, in which we are currently ahead on a's, and a B region, in which we are currently ahead on b's. Then at the top level, we can allow an A region, followed by a B region, followed by an A region and so forth. Think of switching between regions as what will happen when the stack is empty and we're completely even on the number of a's and b's that we've seen so far. For example, [ab][ba] is one A region followed by one B region. Once we enter an A region, we stay in it, always generating an a followed (possibly after something else embedded in the middle) by a b. After all, the definition of an A region, is that we're always ahead on a's. Only when we are even, can we switch to a B region. Until then, if we want to generate a b, we don't need to do a pair starting with b. We know we're ahead on a's, so make any desired b's go with an a we already have. Once we are even, we must either quit or move to a B region. If we allow for two A regions to be concatenated at the top, there will be ambiguity between concatenating two A regions at the top vs. staying in a single one. We must, however, allow two A regions to be concatenated once we're inside an A region. Consider [a[ab][ab]b] Each [ab] is a perfectly

balanced A region and they are concatenated inside the A region whose boundaries are the first a and the final b. So we must distinguish between concatenation within a region (which only happens with regions of the same type, e.g, two A's within an A) and concatenation at the very top level, which only happens between different types.

Also, we must be careful of any rule of the form $X \rightarrow XX$ for another reason. Suppose we have a string that corresponds to XXX. Is that the first X being rewritten as two, or the second one being rewritten as two. We need to force a single associativity.

All of this leads to the following set of rules R:

$S \rightarrow \varepsilon$

$S \rightarrow T_a$ /* start with an A region, then optionally a B, then an A, and so forth

$S \rightarrow T_b$ /* start with a B region, then optionally an A, then a B, and so forth

$T_a \rightarrow A$ /* just a single A region

$T_a \rightarrow AB$ /* two regions, an A followed by a B

$T_a \rightarrow ABT_a$ /* we write this instead of $T_a \rightarrow T_aT_a$ to allow an arbitrary number of regions,

but force associativity,

$T_b \rightarrow B$ /* these next three rules are the same as the previous three but starting with b

$T_b \rightarrow BA$

$T_b \rightarrow BAT_b$

$A \rightarrow A_1$ /* this A region can be composed of a single balanced set of a's and b's

$A \rightarrow A_1A$ /* or it can have arbitrarily many such balanced sets.

$A_1 \rightarrow aAb$ /* a balanced set is a string of A regions inside a matching a, b pair

$A_1 \rightarrow ab$ /* or it bottoms out to just the pair a, b

$B \rightarrow B_1$ /* these next four rules are the same as the previous four but for B regions

$B \rightarrow B_1B$

$B_1 \rightarrow bBa$

$B_1 \rightarrow ba$

(e) The string aabbab is a single A region, and has only one parse tree in this grammar, corresponding to [[aabb][ab]]. You may also want to try abab, abba, and abaababb to see how G works.

8. Consider the grammar G = ({+, *, (,), id, T, F, E}, {+, *, (,), id}, R, E}, where

$R = \{E \rightarrow E+T,\ E \rightarrow T,\ T \rightarrow T * F,\ T \rightarrow F,\ F \rightarrow (E),\ F \rightarrow id\}$.

Give two derivations of the string id * id + id, one of which is leftmost and one of which is not leftmost.

Solution

9. Draw parse trees for each of the following:

(a) The grammar of Problem 8 and the strings id + (id + id) * id and (id * id + id * id).

10. Using the principles of CFG, capture expression (x1+x2/x1) *(x1*x1+x2). Also draw its parse tree.

11. Consider the pushdown automaton M = (K, Σ, Γ, Δ, s, F), where

K = {s, f},
F = {f},
Σ = {a, b},
Γ = {a},
Δ = {((s, a, ε), (s, a)), ((s, b, ε), (s, a)), ((s, a, ε), (f, ε)), ((f, a, a), (f, ε)), ((f, b, a), (f, ε))}.

(a) Trace all possible sequences of transitions of M on input aba.

(b) Show that aba, aa, abb ∉ L(M), but baa, bab, baaaa ∈ L(M).

(c) Describe L(M) in English.

Solutions

(a) There are three possible computations of M on aba:

(s, aba, ε) |- (s, ba, a) |- (s, a, aa) |- (s, ε, aaa)
(s, aba, ε) |- (s, ba, a) |- (s, a, aa) |- (f, ε, aa)
(s, aba, ε) |- (f, ba, ε)

None of these is an accepting configuration.

(b) This is done by tracing the computation of M on each of the strings, as shown in (a).

(c) L(M) is the set of strings whose middle symbol is a. In other words,

$L(M) = \{xay \in \{a, b\}^* : |x| = |y|\}$.

12. Construct pushdown automata that accept each of the following:

(a) L = the language generated by the grammar G = (V, Σ, R, S), where

$V = \{S, (,), [,]\}$, $\Sigma = \{(,), [,]\}$, $R = \{\ S \rightarrow \varepsilon,\ S \rightarrow SS,\ S \rightarrow [S],\ S \rightarrow (S)\}$.

Solution

Notice that the square brackets and the parentheses must be properly nested. So the strategy will be to push the open brackets and parens and pop them against matching close brackets and parens as they are read in. We only need one state, since all the counting will be done on the stack. Since $\varepsilon \in L$, the start state can be final.

Thus we have $M = (K, \Sigma, T, \Delta, s, F)$, where

K= {s}

Σ= { (,), [,] }

T= { (, [}

Δ = {

1. ((s, (, ε), (s, ()), /* push (*/
2. ((s, [, ε), (s, []), /* push [*/
3. ((s,), (),(s, ε)), /* if the input character is) and the top of the stack is (, they match */
4. ((s,], [), (s, ε)) /* same for matching square brackets */

}

S= s

F= {s}

If we run out of input and stack at the same time, we'll accept.

(b) $L = \{a^m b^n : m \le n \le 2m\}$.

Solution

Clearly we need to use the stack to count the a's and then compare that count to the b's as they're read in. The complication here is that for every a, there may be either one or two b's. So we'll need nondeterminism. Every string in L has two regions, the a region followed by the b region (okay, they're hard to tell apart in the case of ε, but trivially, this even true there). So we need a machine with at least two states.

There are two ways we could deal with the fact that, each time we see an a, we don't know whether it will be matched by one b or two. The first is to push either one or two characters onto the stack. In this case, once we get to the b's, we'll pop one character for every b we see. A nondeterministic machine that follows all paths of combinations of one or two pushed characters will find at least one match for every string in L. The alternative is to push a single character for every a and then to get nondeterministic when we're processing the b's: For each stack character, we accept either one b or two. Here's a PDA that takes the second approach. You may want to try writing one that does it the other way. This machine actually needs three states since it needs two states for processing b's to allow for the case where two b's are read but only a single a is popped. So M = ({s, f, g}, {a, b}, {a}, Δ, s, {f, g}), where

Δ = {((s, a, ε), (s, a)), /* Read an a and push one onto the stack */

((s, ε, ε), (f, ε)), /* Jump to the b reading state */

((f, b, a), (f, ε)), /* Read a single b and pop an a */

((f, b, a), (g, ε)),/* Read a single b and pop an a but get ready to read a second one */

((g, b, ε), (f, ε))}. /* Read a b without popping an a */

(c) L = {w ∈ {a, b}* : w = w^R}.

Solution

For the WWR

M = (K, Σ, Γ, Δ, s, F), where:

K = {s, f} the states

Σ = {a, b, c} the input alphabet

Γ = {a, b} the stack alphabet

F = {f} the final states

Δ contains:

((s, a, ε), (s, a))
((s, b, ε), (s, b))
((s, ε, ε), (f, ε))
((f, a, a), (f, ε))
((f, b, b), (f, ε))

A PDA that accepts {w : w = w^R} is just a variation of the PDA that accepts {ww^R} . You can modify that PDA by adding two transitions ((s, a, ε), (f, ε)) and ((s, b, ε), (f, ε)), which have the effect of making odd length palindromes accepted by skipping their middle symbol.

So Δ contains:

((s, a, ε), (s, a))
((s, b, ε), (s, b))
((s, ε, ε), (f, ε))
((f, a, a), (f, ε))
((f, b, b), (f, ε))

((s, a, ε), (f, ε)) and
((s, b, ε), (f, ε)),

(d) L = {w ∈ {a, b}* : w has equal numbers of a's and b's}.- See A k panday Book

(e) L = {w ∈ {a, b}* : w has twice as many a's as b's}.- See A k Panday Book

(f) L = {$a^m b^n$: m ≥ n }

(g) L= { wcwR ∈ {a, b}* }

Solution

M = (K, Σ, Γ, Δ, s, F), where:

K = {s, f} the states
Σ = {a, b, c} the input alphabet
Γ = {a, b} the stack alphabet
F = {f} the final states
Δ contains:

((s, a, ε), (s, a))
((s, b, ε), (s, b))
((s, c, ε), (f, ε))
((f, a, a), (f, ε))
((f, b, b), (f, ε))

(See on note Chapter 3) - this can be just two state diagram only here initially $ sign is pushed into Stack as end marker of stack and *1, 0 → e is read as on input 1 and stack top symbol is 0 then pop stack symbol)*

(h) L= { wwR ∈ {a, b}* }

Solution

M = (K, Σ, Γ, Δ, s, F), where:

K = {s, f} the states
Σ = {a, b, c} the input alphabet
Γ = {a, b} the stack alphabet
F = {f} the final states
Δ contains:

((s, a, ε), (s, a))
((s, b, ε), (s, b))
((s, ε, ε), (f, ε))
((f, a, a), (f, ε))
((f, b, b), (f, ε))

(See on note Chapter 3) - this can be just two state diagram only here initially $ sign is pushed into Stack as end marker of stack and *1, 0 → e is read as on input 1 and stack top symbol is 0 then pop stack symbol)*

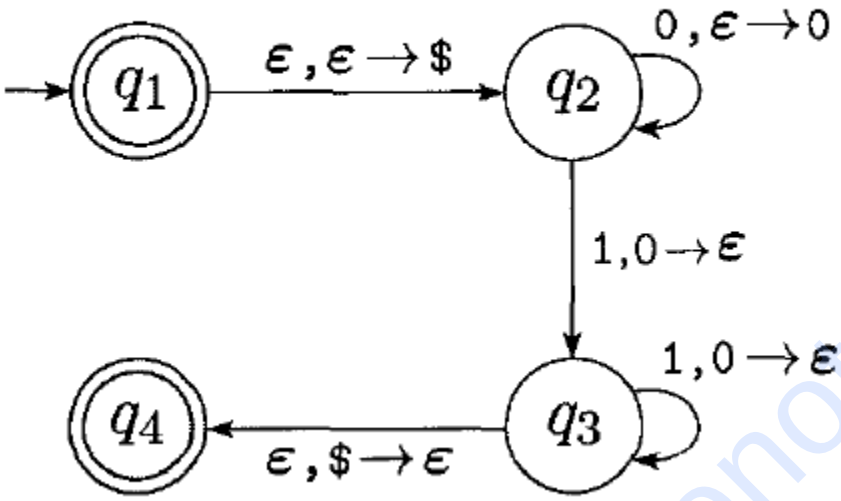
(i) $L= \{ a^n b^{2n} \in \{a, b\}^* : n > 0 \}$

(j) $L = \{0^n 1^n \quad n \geq 0 : \ L \in \{0, 1\}^* \}$.

Solution

(See on note Chapter 3) - this can be just two state diagram only

here initially $ sign is pushed into Stack as end marker of stack and *1, 0 → e is read as on input 1 and stack top symbol is 0 then pop stack symbol)*

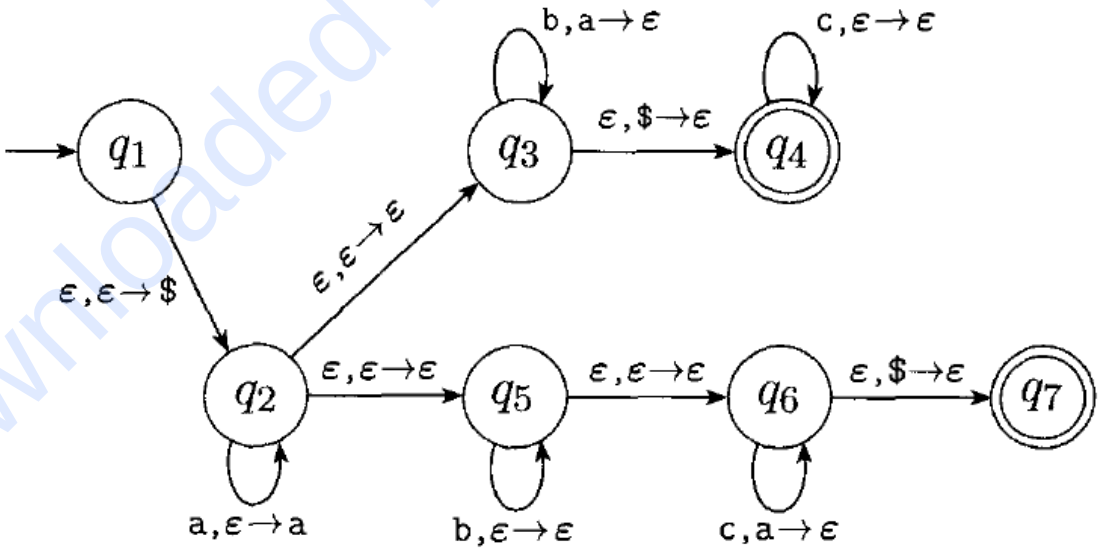


(k) L= { $a^i b^j c^k$: i=j or i=k L ∈ {a, b}* }

Solution

(See on note Chapter 3) -

here initially $ sign is pushed into Stack as end marker of stack and *1, 0 → e is read as on input 1 and stack top symbol is 0 then pop stack symbol)*



(See on note Chapter 3)

(L) L= { $a^n bc^n : n \geq 0$ L ∈ {a, b}* }

13. Construct a Push Down Automata (PDA) accepting L= {$a^n b^m a^n$ |m, n ≥1 } by empty stack
14. Show that the following languages are not context-free.
    (a) L = { a^{n^2} : n ≥ 0}
    (b) L = {www : w ∈ {a, b}*}
    (c) L = {w ∈ {a, b, c}* : w has equal numbers of a's, b's, and c's}
    (d) L = {$a^n b^m a^n$: n ≥ m}
    (e) L = {$a^n b^m c^n d^{(n+m)}$: m, n ≥ 0}
    (f) L = {$a^n b^n c^n$: n ≥ 0}
15. Show that $L = \{a^p | p \text{ is a prime number}\}$ is not a context free language.
16. Design a CFG for the language L= { WCW^R : W ϵ {a, b}* }. Also convert the designed CFG to PDA.
17. Consider the following CFG,

$$S \longrightarrow aA$$
$$A \longrightarrow aABC|bB|a$$
$$B \longrightarrow b$$
$$C \longrightarrow c$$

where S is the start symbol.

Design a pushdown automata corresponding to the above grammar.

18. Design a PDA for the grammar G=(V, E, R,S) where V= { S, a,b,c } ,E= { a, b,c} and R is given by S→ aSa, S→ bSb, S→ c.
19. Consider the **regular grammar** G = (V,Σ, *R, S)*, where
    V = {a,b,A,B,S}
    Σ = *{a,b}*
    R = {S → abA,
    S → B,
    S → baB,
    S → e,
    A → bS,
    B → as,
    A→b}.

Construct a nondeterministic finite automaton M such that *L(M) = L(G).*

Trace the transitions of M that lead to the acceptance of the string *abba*, and compare with a derivation of the same string in G.

20. Given a CFG with P given by

S →AB/ a

A→ b

Eliminate the useless symbols to obtain an equivalent grammar.

Solution

Given $S \rightarrow AB \mid a$

$A \rightarrow b$

B is a non-generating symbol. a and b generate themselves. S generates a and A generates b.

When B is eliminated, $S \rightarrow AB$ is eliminated. Therefore we have

$$S \rightarrow a$$
$$A \rightarrow b$$

S and a are only reachable from S. Therefore we eliminate A and b, therefore we have

$$S \rightarrow a$$

as the new $\hat{P}$ for equivalent grammar.

21. Given the CFG with P given by

S →AB

A →aAA/λ

B →bBB/λ

Eliminate the λ-productions to obtain P for an equivalent CFG.

Solution

A and B are "Nullable Symbols" as they have λ-productions. S is also "Nullable", because it has the production $S \rightarrow AB$, which has only "Nullable symbols", A and B.

For $S \rightarrow AB$, we have three ways viz.,

$$S \rightarrow AB \mid A \mid B$$

For $A \rightarrow aAA$, we have four ways viz.,

$$A \rightarrow aAA \mid aA \mid aA \mid a$$

For $B \rightarrow bBB$, we have three ways viz.,

$$B \rightarrow bBB \mid bB \mid b$$

Therefore, the new set of productions $\hat{P}$ for the grammar equivalent to the given CFG is

$$S \rightarrow AB \mid A \mid B$$
$$S \rightarrow aAA \mid aA \mid a$$
$$B \rightarrow bBB \mid bB \mid b.$$

21. Obtain a grammar in Chomsky Normal Form (CNF) equivalent to the grammar G with productions P given

S →aAbB

A →aA/ a

B →bB/ b

Solution

(i) There are no unit productions in the given set of P.

(ii) Amongst the given productions, we have

$$A \to a,$$
$$B \to b$$

which are in proper form.

For $S \to aAbB$, we have

$$S \to B_a AB_b B,$$
$$B_a \to a$$
$$B_b \to b.$$

For $A \to aA$, we have

$$A \to B_a A$$

For $B \to bB$, we have

$$B \to B_b B.$$

Therefore, we have G_1 given by

$$G_1 = (\{S, A, B, B_a, B_b\}, \{a, b\}, P', S)$$

where P' has the productions

$$\begin{aligned} S &\to B_a A B_b B \\ A &\to B_a A \\ B &\to B_b B \\ B_a &\to a \\ B_b &\to b \\ A &\to a \\ B &\to b \end{aligned}$$

(iii) In P' above, we have only

$$S \to B_a A B_b B$$

not in proper form.

Hence we assume new variables D_1 and D_2 and the productions

$$\begin{aligned} S &\to B_a D_1 \\ D_1 &\to A D_2 \\ D_2 &\to B_b B \end{aligned}$$

Therefore the grammar in Chomsky Normal Form (CNF) is G_2 with the productions given by

$$\begin{aligned} S &\to B_a D_1, \\ D_1 &\to A D_2, \\ D_2 &\to B_b B, \\ A &\to B_a A, \\ B &\to B_b B, \\ B_a &\to a, \\ B_b &\to b, \\ A &\to a, \\ \text{and} \quad B &\to b. \end{aligned}$$

22. Reduce the given CFG with P given by
S → abSb/a /aAb and A→ bS / aAAb to Chomsky Normal Form (CNF).
Solution

(i) There are neither λ-productions nor unit product in the given set of P.

(ii) Among the given productions, we have

$$S \to a$$

in proper form.
For $S \to abSb$, we have

$$S \to B_a B_b S B_b, \quad B_a \to a, \quad \text{and } B_b \to b.$$

For $S \to aAb$, we have

$$S \to B_a A B_b.$$

For $A \to bS$, we have

$$A \to B_b S.$$

For $A \to aAAb$, we have

$$A \to B_a A A B_b.$$

Therefore, we have G_1 given by

$$G_1 = (\{S, A, B_a, B_b\}, \{a, b\}, P', S)$$

which has P' given by

$$\begin{aligned} S &\to B_a B_b S B_b \\ S &\to B_a A B_b \\ A &\to B_a A A B_b \\ A &\to B_b S \\ B_a &\to a \\ B_b &\to b \end{aligned}$$

and $S \to a$.

(iii) In P' above, we have

$$\begin{aligned} S &\to B_a B_b S B_b \\ S &\to B_a A B_b \end{aligned}$$

and $A \rightarrow B_a AAB_b$

not in proper form.

Hence we assume new variables D_1, D_2, D_3, D_4 and D_5 with productions given as below:

For $S \rightarrow B_a B_b SB_b$, we have

$$S \rightarrow B_a D_1, \quad D_1 \rightarrow B_b D_2, \quad D_2 \rightarrow SB_b$$

For $S \rightarrow B_a AB_b$, we have

$$S \rightarrow B_a D_3$$
$$D_3 \rightarrow AB_b$$

For $A \rightarrow B_a AAB_b$, we have

$$A \rightarrow B_a D_4$$
$$D_4 \rightarrow AD_5$$
$$D_5 \rightarrow AB_b$$

Therefore, the grammar in Chomsky Normal Form (CNF) is G_2 with production given by

$$S \rightarrow B_a D_1$$
$$D_1 \rightarrow B_b D_2$$
$$D_2 \rightarrow SB_b$$
$$S \rightarrow B_a D_3$$
$$D_3 \rightarrow AB_b$$
$$A \rightarrow B_a D_4$$
$$D_4 \rightarrow AD_5$$
$$D_5 \rightarrow AB_b$$
$$A \rightarrow B_b S$$
$$B_a \rightarrow a$$
$$B_b \rightarrow b$$

and $S \rightarrow a$.

23. Convert CFG which is given below into CNF form
    S → bA/aB
    A→ bAA / aS/a
    A→ aBB / bS/b
24. Design a CFG for the language L= $\{a^n b^m c^m d^n : n \geq 1\ m \geq 1\}$. Also derive the string aabbbcccdd from the same grammar that you have design. (See Ak Panday Book)
25. Write a CFG for the language L= $\{0^i 1^j 2^k : i=j \text{ or } i=k\}$ (Hint: Make CFG for L1= $0^i 1^j 2^k$ i=j , L2= $0^i 1^j 2^k$ i=k and make union of $L_1 \cup L_2$)
    (See Ak Panday Book)