

**UNIVERSITATEA BABEȘ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
CENTRUL DE FORMARE CONTINUĂ ȘI ÎNVĂȚĂMÎNT LA DISTANȚĂ**

GRIGOR MOLDOVAN

MIHAELA LUPEA

VASILE CIOBAN

LIMBAJE FORMALE ȘI AUTOMATE

Culegere de probleme

2000

2

1. NOȚIUNI FUNDAMENTALE ȘI PROBLEME PROPUSE.....	3
1.1. GRAMATICI ȘI LIMBAJE.....	3
1.1.1. Probleme propuse.....	5
1.1.2. Indicații și soluții.....	8
1.2. AUTOMATE FINITE.....	20
1.2.1. Automate finite deterministe (AFD), automate finite nedeterministe (AFN).....	20
1.2.2. Reprezentare.....	20
1.2.3. Configurații și relații de tranziție.....	21
1.2.4. Limbaj acceptat de un AF.....	22
1.2.5. Funcționare.....	22
1.2.6. Echivalența dintre AFD și AFN.....	24
1.2.7. Minimizarea automatelor finite.....	24
1.2.8. Construcția AFD redus.....	25
1.2.9. Automate finite cu ϵ -mişcări.....	26
1.2.10. Probleme propuse.....	26
1.2.11. Indicații și soluții.....	31
1.3. EXPRESII REGULARE.....	35
1.3.1. Probleme rezolvate.....	39
1.3.2. Probleme propuse.....	43
1.3.3. Indicații și soluții.....	45
1.4. GRAMATICI ȘI LIMBAJE REGULARE.....	46
1.4.1. Legătura dintre gramaticile regulate și automatele finite.....	46
1.4.2. Proprietăți de închidere ale limbajelor regulate.....	46
1.4.3. Lema de pompare pentru limbaje regulate.....	47
1.4.4. Probleme propuse.....	48
1.4.5. Indicații și soluții.....	50
1.5. GRAMATICI ȘI LIMBAJE INDEPENDENTE DE CONTEXT.....	54
1.5.1. Proprietăți de închidere ale limbajelor independente de context.....	54
1.5.2. Arbori de derivare.....	54
1.5.3. Simplificarea gramaticilor independente de context.....	56
1.5.4. Forma normală Chomsky.....	59
1.5.5. Forma normală Greibach.....	59
1.5.6. Leme de pompare pentru limbaje independente de context.....	60
1.5.7. Probleme propuse.....	60
1.5.8. Indicații și soluții.....	63
1.6. AUTOMATE PUSH-DOWN.....	70
1.6.1. Reprezentare.....	71
1.6.2. Limbaj acceptat de un APD.....	71
1.6.3. Probleme propuse.....	72
1.6.4. Indicații și soluții.....	73
1.7. TRANSLATARE ȘI TRANSLATOARE.....	84
1.7.1. Translator finit.....	84
1.7.2. Translator push-down.....	85
1.7.3. Probleme propuse.....	86
1.7.4. Indicații și soluții.....	86
2. PROBLEME ȘI PROGRAME COMPLEXE.....	88
2.1. PROBLEME PROPUSE.....	88
2.1. INDICAȚII ȘI SOLUȚII.....	90
BIBLIOGRAFIE.....	103

1. NOȚIUNI FUNDAMENTALE ȘI PROBLEME PROPUSE

1.1. GRAMATICI ȘI LIMBAJE

Fie Σ o mulțime de elemente, finită și nevidă. O astfel de mulțime Σ o numim *alfabet*, iar elementele ei le numim *simboluri*.

O succesiune de simboluri ale alfabetului Σ o numim *secvență* peste alfabetul Σ , iar o mulțime de simboluri consecutive dintr-o secvență se numește *subsecvență*.

Exemplu: Dacă $w = a_1a_2\dots a_n$; atunci w este o secvență peste alfabetul Σ , iar dacă $w = a_1a_2a_3a_4a_5$ atunci $w' = a_3a_4$ este o subsecvență a secvenței w .

Dacă $x = a_1a_2\dots a_n$ iar $y = b_1b_2\dots b_m$ atunci $z = xy$, adică $z = a_1\dots a_nb_1\dots b_m$ reprezintă *concatenarea* secvențelor x și y .

Dacă x, y, z sunt secvențe peste alfabetul Σ , atunci x este un *prefix* al secvenței xy , y un *suffix* al secvenței xy , iar y în secvența xyz este o *subsecvență*.

De asemenea, adesea vom folosi notația $a^n = a\dots a$ (de n ori).

Lungimea unei secvențe w , peste un alfabet Σ , notată $|w|$, este numărul simbolurilor din care este alcătuită. Secvența de lungime 0, o notăm cu ϵ , deci $|\epsilon| = 0$, și se numește *secvența vidă*.

Exemplu: Dacă $w = a_1a_2\dots a_n$ atunci $|w| = n$.

Notăm cu Σ^* mulțimea tuturor secvențelor peste alfabetul Σ .

Exemplu: Dacă $\Sigma = \{a\}$, atunci $\Sigma^* = \{\epsilon, a, a^2, \dots, a^n, \dots\}$

Observație: O secvență peste alfabetul Σ , poate fi considerată și ca un element al monoidului liber (Σ^*, \cdot) generat de Σ , iar operația " \cdot " este operația de concatenare

$\cdot : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ definită mai sus. Operația de concatenare este asociativă și $\epsilon \in \Sigma^*$ este elementul neutru. Cele două proprietăți ale concatenării definesc structura de monoid. Pentru operația de concatenare *nu* folosim simboluri speciale.

Numim *limbaj* peste alfabetul Σ , o submulțime L a mulțimii Σ^* , deci $L \subseteq \Sigma^*$. Elementele unui limbaj le numim *cuvinte*. Un limbaj poate să fie finit, adică să conțină un număr finit de cuvinte, sau să fie infinit.

Dacă L_1 este un limbaj peste alfabetul Σ_1 , iar L_2 este un limbaj peste alfabetul Σ_2 atunci prin operațiile:

- $L_1 \cup L_2$ (reuniune);
- $L_1 \cap L_2$ (intersecție);
- $L_1 - L_2$ (diferența);
- $L_1 L_2 = \{uv \mid u \in L_1, v \in L_2\}$ (concatenarea);

obținem, de asemenea, niște limbaje peste alfabete alese corespunzător.

Fie L, L_1, L_2 limbaje peste un alfabet Σ , atunci putem defini următoarele limbaje:

- $\bar{L} = \{x \in \Sigma^* \mid x \notin L\}$ (complementara);
- $L^* = \bigcup_{n \geq 0} L^n$ unde $L^n = LL^{n-1}$, $L^0 = \{\epsilon\}$; (închiderea reflexivă și tranzitivă);
- $L^+ = \bigcup_{n \geq 1} L^n$ sau $L^+ = LL^* = L^+L$, $L^* = L^+ \cup \{\epsilon\}$; (închiderea tranzitivă);
- $L_1 / L_2 = \{w \in \Sigma^* \mid \exists y \in L_2: wy \in L_1\}$ (câtul la dreapta);
- $L_1 \setminus L_2 = \{w \in \Sigma^* \mid \exists y \in L_2: yw \in L_1\}$ (câtul la stânga);

Un ansamblu $G = (N, \Sigma, P, S)$ unde:

- N este un alfabet de simboluri *neterminale*;
- Σ este un alfabet de simboluri *terminale*;
- $P \subseteq (N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$, P este o mulțime finită de *producții*;
- $S \in N$, S este *simbolul inițial*;

se numește *gramatică*.

Dacă $(\alpha, \beta) \in P$, atunci convenim să notăm această producție astfel: $\alpha \rightarrow \beta$ și înseamnă că α se înlocuiește cu β .

Observație: Din definiția unei gramatici rezultă evident că $N \cap \Sigma = \emptyset$.

O gramatică reprezintă o modalitate de *specificare* a unui limbaj. Fie gramatica $G = (N, \Sigma, P, S)$. Pe mulțimea $(N \cup \Sigma)^*$ se definesc relațiile binare:

\Rightarrow (derivare directă);

$\stackrel{k}{\Rightarrow}$ (k derivare);

$\stackrel{+}{\Rightarrow}$ (+ derivare);

$\stackrel{*}{\Rightarrow}$ (* derivare).

Avem $\gamma \Rightarrow \delta \Leftrightarrow \exists \gamma_1, \gamma_2, \alpha, \beta \in (N \cup \Sigma)^*$ astfel încât $\gamma = \gamma_1 \alpha \gamma_2$, $\delta = \gamma_1 \beta \gamma_2$, iar $(\alpha \rightarrow \beta) \in P$.

O "*k derivare*" este o succesiune de *k* derivări directe, adică $\gamma \stackrel{k}{=} \delta \Leftrightarrow$

$$\exists \alpha_1, \dots, \alpha_{k-1} \in (N \cup \Sigma)^* \text{ astfel încât } \gamma \Rightarrow \alpha_1 \Rightarrow \dots \Rightarrow \alpha_{k-1} \Rightarrow \delta.$$

Între două secvențe avem o "*+ derivare*" dacă $\exists k > 0$ astfel încât ele să fie într-o relație de "*k derivare*".

Între două secvențe avem o "** derivare*" dacă $\gamma \Rightarrow \delta$ sau $\gamma \stackrel{+}{=} \delta$.

Limbajul generat de o gramatică $G = (N, \Sigma, P, S)$ este:

$$L(G) = \{w \mid w \in \Sigma^*, S \stackrel{*}{=} w\}$$

Fie gramatica $G = (N, \Sigma, P, S)$. O secvență $x \in (N \cup \Sigma)^*$ astfel încât $S \stackrel{*}{=} x$, se numește **formă propozițională**. O formă propozițională care nu conține neterminale se numește **propoziție**. O propoziție este un **cuvânt** al limbajului generat de gramatica G .

Două gramatici G_1 și G_2 sunt **echivalente** dacă ele generează același limbaj, deci $L(G_1) = L(G_2)$.

După forma produțiilor gramaticilor, acestea se clasifică (după **Chomsky**) în gramatici de tip **0,1,2,3**.

Gramaticile de tip **0** sunt gramaticile care nu au nici o restricție referitoare la forma produțiilor.

Gramaticile de tip **1** sunt gramaticile în care pentru orice producție $\alpha \rightarrow \beta$ din P avem $|\alpha| \leq |\beta|$. Dacă producția $S \rightarrow \varepsilon$ aparține gramaticii, atunci S nu apare în membrul drept al nici unei producții.

Gramaticile de tip **2** sunt gramaticile în care produțiile sunt de forma $A \rightarrow \alpha$, $A \in N$, $\alpha \in (N \cup \Sigma)^*$.

Gramaticile de tip **3** sunt gramaticile în care orice producție are numai una din formele următoare: $A \rightarrow aB$ sau $A \rightarrow b$, unde $A, B \in N$ și $a, b \in \Sigma$, iar dacă producția $S \rightarrow \varepsilon$ aparține gramaticii, atunci S nu apare în membrul drept al nici unei producții.

Gramaticile de tip **1** se mai numesc **gramatici dependente de context**, cele de tip **2** sunt **gramatici independente de context**, iar cele de tip **3** **gramatici regulate**.

1.1.1. Probleme propuse

1.1.1.

Să se definească mulțimea numerelor naturale \mathbf{N} ca limbaj.

1.1.2.

Fie gramatica $G = (N, \Sigma, P, S)$ unde $N = \{S, C\}$, $\Sigma = \{a, b\}$, $P = \{S \rightarrow ab \mid aCSb, C \rightarrow S \mid bSb, CS \rightarrow b\}$. Să se arate că $ab(ab^2)^2 \in L(G)$.

1.1.3.

Se dă gramatica $G = (N, \Sigma, P, S)$ unde:

$$N = \{S, A\};$$

$$\Sigma = \{\wedge, \vee, \neg, p, q, r, ', (,)\};$$

$$P = \{S \rightarrow (S) \wedge (S) \mid (S) \vee (S) \mid \neg(S) \mid A, A \rightarrow A' \mid p \mid q \mid r\}.$$

Să se verifice că următoarele secvențe aparțin limbajului generat de această gramatică:

a) $((p') \wedge (p)) \vee (r'')$;

b) $\neg((p) \vee (q'))$;

c) r''' .

1.1.4.

Se dă gramatica $G = (N, \Sigma, P, S)$ unde:

$$N = \{S, A\};$$

$$\Sigma = \{\wedge, \vee, \neg, p, q, r, '\};$$

$$P = \{S \rightarrow \wedge SS \mid \vee SS \mid \neg S \mid A, A \rightarrow A' \mid p \mid q \mid r\};$$

Să se verifice că $\wedge \vee \wedge \vee pqr \wedge pqr \in L(G)$.

1.1.5.

Fie gramatica $G = (\{S\}, \{a, b, c\}, \{S \rightarrow a^2S \mid bc\}, S)$. Să se arate că: $L(G) = \{a^{2n}bc \mid n \geq 0\}$.

1.1.6.

Se dă gramatica $G = (\{S, H\}, \{b, c, d, e\}, \{S \rightarrow bbSe \mid H, H \rightarrow cHdd \mid cd\}, S)$.

Care este limbajul generat de această gramatică ?

1.1.7.

Se dă gramatica independentă de context $G = (\{S, A, B\}, \{a, b, c\}, P, S)$ unde:

$$P = \{S \rightarrow AB, A \rightarrow aAb \mid ab, B \rightarrow cB \mid c\}.$$

a) Care este limbajul generat de această gramatică ?

b) Să se reprezinte sub formă de vectori, listă ramificată și tablou gramatica dată.

1.1.8.

Să se construiască gramatici pentru generarea limbajelor:

a) $L = \{a^{2n-1} \mid n \geq 1\}$;

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

1.1.9.

Să se construiască gramatici care generează limbajul $L = \{x^n y^n \mid n \geq 1\}$.

1.1.10.

Să se construiască o gramatică dependentă de context care generează limbajul:
 $L = \{a^n b^n c^n \mid n \geq 1\}$.

1.1.11.

Să se construiască o gramatică dependentă de context care generează limbajul:
 $L = \{a^n b^n c^n d^n \mid n \geq 1\}$.

1.1.12.

Să se dea o gramatică care generează limbajul $L = \{a^n b^n c^n d^n e^n \mid n \geq 1\}$.

1.1.13.

Să se construiască gramaticile care generează limbajele:

- a) $L_1 = \{ww \mid w \in \{a,b\}^*\}$;
- b) $L_2 = \{a^m b^n a^m b^n \mid m, n \geq 1\}$;
- c) $L_3 = \{wxw \mid w, x \in \{a,b\}^*\}$.

1.1.14.

Să se construiască gramaticile care generează limbajele:

- a) $L_1 = \{a^{2^n} \mid n \geq 0\}$;
- b) $L_2 = \{a^{k^n} \mid n \geq 0, k \in \mathbb{N}, \text{fixat}\}$.

1.1.15.

Să se construiască gramaticile care generează:

- a) expresiile algebrice ce conțin operandul a și operatorii: $+$, $*$, $(,)$;
- b) expresiile algebrice ce conțin operandul a și operatorii: $+$, $-$, $*$, $:$, $(,)$.

1.1.16.

Să se construiască o gramatică care generează limbajul $L = \{a^{2m} b^{n+1} c^{2n+1} d^m \mid m \geq 1, n \geq 0\}$.

1.1.17.

Să se construiască gramatici care generează limbajele:

- a) $L_1 = \{a^n b^n c^m d^m \mid n, m \geq 1\}$;
- b) $L_2 = \{a^n b^n a^m b^m \mid n, m \geq 1\}$.

1.1.18.

Să se construiască o gramatică care generează limbajul $L = \{a^n b^n c^n d^m e^m \mid n, m \geq 1\}$.

1.1.19.

Să se construiască gramatici dependente de context care generează limbajele:

- a) $L_1 = \{a^n b^n c^m \mid n \geq m \geq 0\}$;
- b) $L_2 = \{a^n b^n c^m \mid m \geq n \geq 0\}$;
- c) $L_3 = \{a^n b^m c^l \mid n \geq m \geq l \geq 0\}$;
- d) $L_4 = \{a^n b^n c^m \mid 0 \leq n \leq m \leq 2n\}$.

1.1.20.

Pentru o secvență w să notăm numărul de simboluri s din această secvență cu $nr_s(w)$. Să se construiască gramatici care generează limbajele:

- a) $L_1 = \{w \in \{a,b,c\}^* \mid nr_a(w) + nr_b(w) = nr_c(w)\}$;
- b) $L_2 = \{w \in \{a,b,c\}^* \mid nr_a(w) = nr_b(w) = nr_c(w)\}$.

1.1.21.

Să se demonstreze că pentru câtul la dreapta dintre limbaje avem:

- a) $L_1 / (L_2 \cup L_3) = (L_1 / L_2) \cup (L_1 / L_3)$;
- b) $(L_1 \cup L_2) / L_3 = (L_1 / L_3) \cup (L_2 / L_3)$;
- c) $L_1 / (L_2 L_3) = (L_1 / L_3) / L_2$;
- d) $(L_1 L_2) / L_3 = L_1 / (L_2 / L_3) \cup L_1 / (L_3 / L_2)$;
- e) $L_1 / (L_2 \cap L_3) = (L_1 / L_2) \cap (L_1 / L_3)$;
- f) $(L_1 \cap L_2) / L_3 = L_1 / L_3 \cap L_2 / L_3$;

Proprietăți similare există și pentru câtul la stânga.

1.1.2. Indicații și soluții

În rezolvarea problemelor de determinare a unei gramatici care generează un limbaj, esențială este stabilirea mulțimii P a producțiilor. De aceea de multe ori mulțimea P o vom da în mai multe variante; totodată de multe ori schimbând producțiile se pot introduce noi neterminale, deci gramaticile care generează limbajul să fie complet diferite.

Să reținem și alte amănunte legate de recursivitatea producțiilor:

- pentru familiarizarea cu definițiile recursive trebuie încercat să se definescă foarte multe noțiuni matematice și informatice în acest mod pornind de la noțiuni simple cum ar fi cea de **identificator** și anume:

$\langle \text{identificator} \rangle := \langle \text{literă} \rangle \mid \langle \text{identificator} \rangle \langle \text{literă} \rangle \mid \langle \text{identificator} \rangle \langle \text{cifră} \rangle$;
 $\langle \text{literă} \rangle := \text{mulțimea literelor alfabetului latin}$;
 $\langle \text{cifră} \rangle := \text{mulțimea cifrelor zecimale, ș.a.m.d.}$

- să reținem apoi că pentru gramaticile de tipul 1, 2 și 3 după fiecare derivare, lungimea secvenței din $(N \cup \Sigma)^*$ se mărește;

- definițiile recursive nu sunt unice întotdeauna, fapt ce se va vedea și în continuare (pentru aceeași problemă în multe cazuri se vor da gramatici diferite).

1.1.1.

Avem $\Sigma = \{0,1,2,3,4,5,6,7,8,9\}$ iar

$$\Sigma^* \supset \{w \mid w=aw_1, a \in \{1,2,3,4,5,6,7,8,9\}, w_1 \in \Sigma^*\} \cup \{0\} = N$$

1.1.2.

Trebuie să arătăm că după un număr finit de derivări directe, \Rightarrow , din simbolul inițial S se obține cuvântul $ab(ab^2)^2=ababbabb$, adică $S \xRightarrow{*} ab(ab^2)^2$. Numerotăm producțiile gramaticii date, astfel:

$$1. S \rightarrow ab \quad 2. S \rightarrow aCSb \quad 3. C \rightarrow bSb \quad 4. C \rightarrow S \quad 5. CS \rightarrow b$$

Vom scrie sub relația de derivare directă, \Rightarrow , numărul regulii de producție care se aplică în trecerea de la o secvență γ la o secvență δ , de exemplu, $\gamma \xRightarrow{3} \delta$ sau dacă folosim succesiv, de la stânga la dreapta, de exemplu producțiile i, j, k mai scurt vom scrie $\xRightarrow{+}_{i,j,k}$.

$$\text{Avem astfel: } S \xRightarrow{2} aCSb \xRightarrow{3} abSbSb \xRightarrow{1} ababbSb \xRightarrow{1} ababbabb$$

$$\text{sau } S \xRightarrow{2} aCSb \xRightarrow{+}_{3,1,1} ababbabb.$$

Observăm că producțiile 4 și 5 nu au fost folosite.

1.1.3.

Procedând analog cu rezolvarea problemei precedente numerotăm producțiile astfel:

$$\begin{array}{ll} 1. S \rightarrow (S) \wedge (S) & 5. A \rightarrow A' \\ 2. S \rightarrow (S) \vee (S) & 6. A \rightarrow p \\ 3. S \rightarrow \neg(S) & 7. A \rightarrow q \\ 4. S \rightarrow A & 8. A \rightarrow r \end{array}$$

și atunci avem:

$$S \xRightarrow{2} (S) \vee (S) \xRightarrow{1} ((S) \wedge (S)) \vee (S) \xRightarrow{+}_{4,4,4} ((A) \wedge (A)) \vee (A) \xRightarrow{5} ((A') \wedge (A)) \vee (A) \xRightarrow{+}_{6,6,5}$$

$$((p') \wedge (p)) \vee (A') \xRightarrow{5} ((p') \wedge (p)) \vee (A'') \xRightarrow{8} ((p') \wedge (p)) \vee (r').$$

b) și c) analog.

1.1.4.

Procedăm analog cu rezolvarea problemei precedente. Numerotăm producțiile astfel:

- | | | |
|------------------------------|-----------------------|----------------------|
| 1. $S \rightarrow \wedge SS$ | 4. $S \rightarrow A$ | 7. $A \rightarrow q$ |
| 2. $S \rightarrow \vee SS$ | 5. $A \rightarrow A'$ | 8. $A \rightarrow r$ |
| 3. $S \rightarrow \neg S$ | 6. $A \rightarrow p$ | |

și atunci avem:

$$\begin{aligned}
 S &= \underset{1}{>} \wedge \underset{2}{SS} = \underset{1,2}{>} \wedge \underset{1,2}{\vee} \underset{1,2}{SSS} = \underset{1,2}{>} \wedge \underset{1,2}{\vee} \underset{1,2}{\wedge} \underset{1,2}{\vee} \underset{1,2}{SSSSS} = \underset{4,4,4}{>} \wedge \underset{4,4,4}{\vee} \underset{4,4,4}{\wedge} \underset{4,4,4}{\vee} \underset{4,4,4}{AAASS} = \underset{6,7,8}{>} \wedge \underset{6,7,8}{\vee} \underset{6,7,8}{\wedge} \underset{6,7,8}{\vee} \underset{6,7,8}{pqrSS} = \underset{1}{>} \\
 &\wedge \underset{1}{\vee} \underset{1}{\wedge} \underset{1}{\vee} \underset{1}{pqr} \wedge \underset{4,4,4}{SSS} = \underset{4,4,4}{>} \wedge \underset{4,4,4}{\vee} \underset{4,4,4}{\wedge} \underset{4,4,4}{\vee} \underset{4,4,4}{pqr} \wedge \underset{4,4,4}{AAA} = \underset{6,7,8}{>} \wedge \underset{6,7,8}{\vee} \underset{6,7,8}{\wedge} \underset{6,7,8}{\vee} \underset{6,7,8}{pqr} \wedge \underset{6,7,8}{pqr}.
 \end{aligned}$$

1.1.5.

Egalitatea mulțimilor, $L(G) = \{a^{2n}bc \mid n \geq 0\}$ se demonstrează prin dublă incluziune:

a) $\{a^{2n}bc \mid n \geq 0\} \subseteq L(G)$, pe scurt " \subseteq ".

Fie $w = a^{2n}bc$, n fixat. Trebuie să arătăm că $w \in L(G)$.

Într-adevăr, folosind de n ori producția 1. $S \rightarrow a^2S$ și odată producția 2. $S \rightarrow bc$, avem:

$$S = \underset{1}{>} \underset{1}{(a^2)} \underset{2}{S} = \underset{2}{>} a^{2n}bc, \text{ adică } a^{2n}bc \in L(G).$$

b) A demonstra incluziunea inversă (" \supseteq "), adică $L(G) \subseteq \{a^{2n}bc \mid n \geq 0\}$, revine la a arăta că gramatica G generează numai cuvinte de forma $a^{2n}bc$. Pentru a demonstra acest lucru să considerăm propoziția care depinde de numărul natural n , $P(n)$:

"Folosind de n ori producțiile $S \rightarrow a^2S$, $S \rightarrow bc$ se obțin numai secvențe de forma $a^{2n}S$ sau $a^{2(n-1)}bc$ ".

Demonstrăm proprietatea $P(n)$ prin inducție matematică.

1) Verificare.

Dacă $n=1$, deci folosind o singură producție, obținem secvența a^2S sau bc .

2) Demonstrația.

Presupunem că $P(k)$ este adevărată și trebuie să demonstrăm că și $P(k+1)$ este adevărată. Secvențele $a^{2(k+1)}S$, $a^{2k}bc$ se obțin folosind câte una din cele două producții, pornind de la secvența $a^{2k}S$.

Din proprietatea $P(n)$ rezultă că singurele cuvinte generate de gramatică sunt de forma $a^{2n}b$, $n \geq 0$ și este adevărată incluziunea " \supseteq ".

1.1.6.

$$L(G) = \{b^{2n}c^m d^{2m-1}e^n \mid m \geq 1, n \geq 0\}.$$

1.1.7.

a) $L(G) = \{a^i b^j c^k \mid i \geq 1, j \geq 1\}$;

b)

1) **vectorial**;

vom construi vectorii:

N:

S	A	B	\$
---	---	---	----

Σ :

a	b	c	\$
---	---	---	----

P:

S	A	B	#	A	a	A	b		a	b	#	B	c	B		c	#	\$
---	---	---	---	---	---	---	---	--	---	---	---	---	---	---	--	---	---	----

Am marcat sfârșitul șirului de caractere ce definesc cei trei vectori cu \$, iar sfârșitul regulilor de producție cu același membru stâng, cu #.

2) listă ramificată înlănțuită

Pentru fiecare membru drept al fiecărei producții se construiește o sublistă liniară. Considerăm nodurile listei de forma:

a		
b	c	d

unde câmpurile au următoarele semnificații:

a - simbolul terminal sau neterminal al gramaticii (numai din membrul drept al fiecărei producții);

b - pointer sau întreg astfel:

b := 0 dacă $a \in \Sigma$;

b := pointer spre capul primei subliste atașate regulii de producție cu membrul stâng a.

c - pointer sau întreg;

Considerăm următoarele propoziții:

p - a este primul simbol al membrului doi dintr-o producție;

u - a este ultimul simbol al membrului doi dintre toate regulile de producție cu același membru stâng;

x - producția curentă este ultima din șirul producțiilor cu același membru stâng;

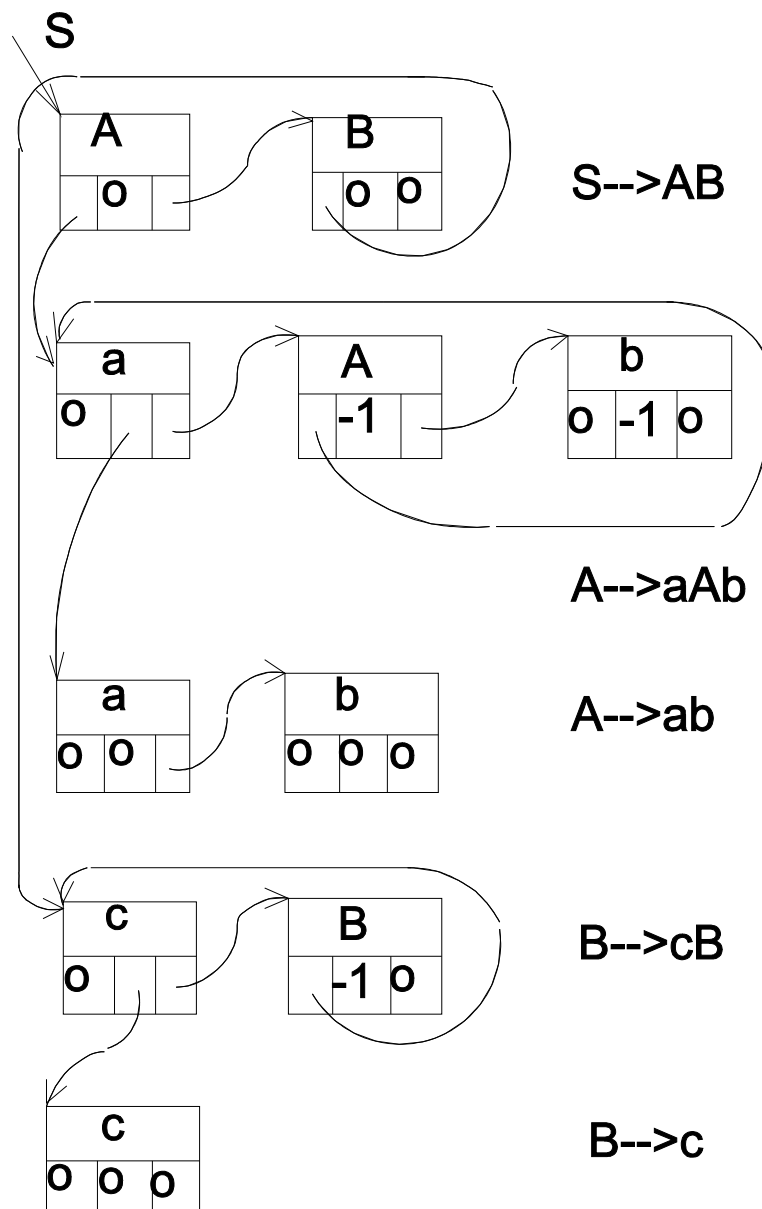
L - legătura spre capul următoarei subliste asociată regulii de producție cu același membru stâng;

c := *dacă* p *atunci* { *dacă* x *atunci* 0 *altfel* L }
 altfel { *dacă* u *atunci* 0 *altfel* -1 }

d - pointer sau întreg

dacă **a** este ultimul simbol al membrului drept al unei producții
 atunci **d** := 0
 altfel **d** este pointer spre următorul nod al sublistei.

Pentru gramatica dată în problemă avem următoarea reprezentare:



3) **tabelar** cu notațiile de la reprezentarea prin listă ramificată obținem:

	a	B	c	D
1	A	3	0	2
2	B	8	0	0
3	a	0	6	4
4	A	3	-1	5
5	b	0	-1	0
6	a	0	0	7
7	b	0	0	0
8	c	0	10	9
9	B	8	-1	0
10	c	0	0	0

1.1.8.

a) Gramatici care generează limbajul $\{a, a^3, a^5, \dots\}$:

$G = (\{S\}, \{a\}, P, S)$ cu $P = \{S \rightarrow aSa \mid a\}$ sau $P = \{S \rightarrow aaS \mid a\}$ sau $P = \{S \rightarrow Saa \mid a\}$ etc.

b) Pentru limbajul $\{a^2, a^4, \dots\}$ avem gramaticile:

$G = (\{S\}, \{a\}, P, S)$ cu $P = \{S \rightarrow aSa \mid aa\}$ sau $P = \{S \rightarrow aaS \mid aa\}$ sau $P = \{S \rightarrow Saa \mid aa\}$.

1.1.9.

Vom da mai multe gramatici care generează acest limbaj:

a) $G_1 = (\{S\}, \{x, y\}, P, S)$ cu $P = \{S \rightarrow xSy \mid xy\}$;

Demonstrăm că $L(G_1) = L$ prin dublă incluziune:

" \supseteq " este imediată:

Fie $w = x^n y^n$, $n > 0$, deci $w \in L$;

Folosind producția $S \rightarrow xSy$ de $n-1$ ori avem: $S \xRightarrow{n-1} x^{n-1} S y^{n-1}$;

Folosind a doua producție ($S \rightarrow xy$) obținem: $x^{n-1} S y^{n-1} \Rightarrow x^n y^n$, deci $S \xRightarrow{*} x^n y^n$ și cu alte cuvinte $w \in L(G)$;

" \subseteq ".

Demonstrația acestei incluziuni, revine la stabilirea faptului că gramatica G_1 generează numai cuvinte de forma $x^n y^n$. Pentru a demonstra acest lucru să considerăm propoziția "Folosind de n ori producțiile gramaticii G_1 , se obțin **numai** secvențe de forma $x^n S y^n$ respectiv $x^n y^n$ ". Această propoziție se demonstrează ușor prin inducție matematică. De aici rezultă că este adevărată incluziunea " \subseteq ".

b) $G_2 = (\{S, A\}, \{x, y\}, P, S)$ cu $P = \{S \rightarrow xAy, xA \rightarrow xxAy, A \rightarrow \varepsilon\}$.

c) $G_3 = (\{S, B\}, \{x, y\}, P, S)$ cu $P = \{S \rightarrow xBy, By \rightarrow Byy, B \rightarrow \varepsilon\}$.

d) $G_4 = (\{S, A, B\}, \{x, y\}, P, S)$

cu $P = \{S \rightarrow xA, A \rightarrow By, B \rightarrow S \mid \varepsilon\}$ sau $P = \{S \rightarrow xA \mid xy, A \rightarrow By, B \rightarrow S\}$.

e) $G_5 = (\{S, A, B\}, \{x, y\}, P, S)$

cu $P = \{S \rightarrow By, B \rightarrow xA, A \rightarrow S \mid \varepsilon\}$ sau $P = \{S \rightarrow By \mid xy, B \rightarrow xA, A \rightarrow S\}$.

Se observă că gramaticile G_1, G_4, G_5 , sunt independente de context (de tipul 2 după Chomsky) iar G_2 și G_3 sunt dependente de context (tipul 1).

1.1.10.

$G = (N, \Sigma, P, S)$

$N = \{S, B\}, \Sigma = \{a, b, c\}; P = \{S \rightarrow aSBc \mid abc, cB \rightarrow Bc, bB \rightarrow bb\}$

Numerotăm producțiile:

1. $S \rightarrow aSBc$; 2. $S \rightarrow abc$; 3. $cB \rightarrow Bc$; 4. $bB \rightarrow bb$.

Vom da câteva explicații:

- folosind prima producție $\{S \rightarrow aSBc\}$ de $n-1$ ori se obține secvența $a^{n-1} S (Bc)^{n-1}$;
- folosind apoi producția a 2-a obținem după încă o derivare secvența $a^n bc (Bc)^{n-1}$;
- în continuare se folosește a 3-a producție de $1+2+\dots+n-1$ ori pentru a obține $a^n b B^{n-1} c^n$;
- se folosește în final ultima producție de $n-1$ ori și se obține $a^n b^n c^n$.

Exemplu: să încercăm să obținem cuvântul $a^4 b^4 c^4$:

$$\underline{S} \xRightarrow{1} a \underline{S} B c \xRightarrow{1} aa \underline{S} B c B c \xRightarrow{1} a^2 a \underline{S} B c B c B c \xRightarrow{2} a^4 bc B c B c B c \xRightarrow{3} a^4 bc B c \underline{B} B c c \xRightarrow{3}$$

$$a^4 \underline{bc} \underline{BBc} \underline{Bcc} =_3 > a^4 \underline{bc} \underline{BBBccc} =_3 > a^4 \underline{bBc} \underline{BBc}^3 =_3 > a^4 \underline{bBBc} \underline{Bc}^3 =_3 > a^4 \underline{bBBBc}^4 =_4 > a^4 \underline{bbBBc}^4 =_4 > a^4 \underline{bbbBc}^4 =_4 > a^4 \underline{b^4c^4}.$$

Am scris sub notația de derivare numărul producției folosite și am subliniat membrul stâng al fiecărei producții folosite.

Să remarcăm, în finalul rezolvării, că se pot da și alte gramatici care generează limbajul cerut:

1) o gramatică cu producțiile oarecum simetrice față de gramatica anterioară:

$$G = (N, \Sigma, P, S)$$

$$N = \{S, B\}, \Sigma = \{a, b, c\},$$

$$P = \{S \rightarrow aBSc \mid abc, Ba \rightarrow aB, Bb \rightarrow bb\}.$$

2) o gramatică cu mai multe neterminale și producții:

$$G = (N, \Sigma, P, S)$$

$$N = \{S, B, C\}, \Sigma = \{a, b, c\},$$

$$P = \{S \rightarrow aBSC \mid aBC, CB \rightarrow BC, bC \rightarrow bc, cC \rightarrow cc, aB \rightarrow ab, bB \rightarrow bb\}.$$

1.1.11.

Vom da două gramatici echivalente bazate pe ideea de la problema precedentă:

a) $G = (N, \Sigma, P, S)$

$$N = \{S, B, C\}, \Sigma = \{a, b, c, d\},$$

$$P = \{S \rightarrow aSBCd \mid aBCd, dB \rightarrow Bd, dC \rightarrow Cd, CB \rightarrow BC, bB \rightarrow bb, cC \rightarrow cc, aB \rightarrow ab\}.$$

b) $G = (N, \Sigma, P, S)$

$$N = \{S, B, C, D\}, \Sigma = \{a, b, c, d\}$$

$$P = \{S \rightarrow aSBCD \mid aBCD, DB \rightarrow BD, CB \rightarrow BC, aB \rightarrow ab, bC \rightarrow bc, cD \rightarrow cd, bB \rightarrow bb, cC \rightarrow cc, dD \rightarrow dd\}.$$

1.1.12.

$$G = (N, \Sigma, P, S) \text{ cu } N = \{S, B, C, D, E\}, \Sigma = \{a, b, c, d, e\},$$

Producțiile sunt:

$$S \rightarrow aSBCDE \mid aBCDE$$

$$EB \rightarrow BE, EC \rightarrow CE, aB \rightarrow ab, cC \rightarrow cc, cD \rightarrow cc, dE \rightarrow de, DB \rightarrow BD, DC \rightarrow CD, bB \rightarrow bb, bC \rightarrow bc, dD \rightarrow dd, eE \rightarrow ee, CB \rightarrow BC, ED \rightarrow DE.$$

1.1.13.

Gramaticile construite sunt dependente de context.

a) $G = (N, \Sigma, P, S)$

$$N = \{S, A, B, C, D\}, \Sigma = \{a, b\},$$

$$P = \{S \rightarrow CD, C \rightarrow aCA \mid bCB, AD \rightarrow aD, BD \rightarrow bD, Aa \rightarrow aA, Ab \rightarrow bA, Ba \rightarrow aB,$$

$$Bb \rightarrow bB, C \rightarrow \epsilon, D \rightarrow \epsilon\}.$$

C=marcator central al cuvântului; D=marcator drept al cuvântului;

-producțiile cu membrul stâng C generează de fapt secvențe de forma $wC\overline{w}D$ (unde \overline{w} este oglinditul lui w dar cu litere mari, adică dacă $w=abb$ atunci $\overline{w}=BBA$) după care C se înlocuiește cu ϵ .

-secvența \overline{w} trebuie inversată, neterminalul de la sfârșitul său se înlocuiește cu terminalul corespunzător (folosind producțiile $AD \rightarrow aD$ sau $BD \rightarrow bD$) și acesta se deplasează spre stânga la locul său (folosind producțiile $Aa \rightarrow aA$, $Ab \rightarrow bA$, $Ba \rightarrow aB$, $Bb \rightarrow bB$);

-când toate neterminalele au ajuns pe rând lângă D cuvântul dorit a fost generat în stânga lui D, iar acesta se înlocuiește cu ϵ .

Exemplu:

$x=ww$, unde $w=abb$, deci $x=abbabb$. Avem:

$$\begin{aligned} S & \xrightarrow{1} CD \xrightarrow{2} aCAD \xrightarrow{3} abCBAD \xrightarrow{3} abbCBBAD \xrightarrow{10} abbBBAD \xrightarrow{4} abbBBaD \xrightarrow{8} abbBaBD \\ & \xrightarrow{8} abbaBBD \xrightarrow{5} abbaBbD \xrightarrow{9} abbabBD \xrightarrow{5} abbabbD \xrightarrow{11} abbabb. \end{aligned}$$

(Am numerotat producțiile în ordinea scrierii lor în mulțimea P).

$$b) G = (N, \Sigma, P, S)$$

$$N = \{S, A, B, C, D, E\}, \Sigma = \{a, b\},$$

$$P = \{S \rightarrow CD, C \rightarrow aCA \mid E, E \rightarrow bEB \mid bB, AD \rightarrow aD, BD \rightarrow bD, Aa \rightarrow aA, Ab \rightarrow bA, Ba \rightarrow aB, Bb \rightarrow bB, D \rightarrow \epsilon\};$$

-vezi a) în care $w=a^m b^n$.

Exemplu:

$x=aabaab$. Avem:

$$\begin{aligned} S & \xrightarrow{1} CD \xrightarrow{2} aCAD \xrightarrow{2} aaCAAD \xrightarrow{3} aaEAAD \xrightarrow{5} aabBAAD \xrightarrow{6} aabBAaD \xrightarrow{8} aabBaAD \\ & \xrightarrow{8} abbaBBD \xrightarrow{5} abbaBbD \xrightarrow{9} abbabBD \xrightarrow{5} abbabbD \xrightarrow{11} abbabb. \end{aligned}$$

(Am numerotat producțiile în ordinea scrierii lor în mulțimea P).

$$c) G = (N, \Sigma, P, S)$$

$$N = \{S, A, B, C, D, X\}, \Sigma = \{a, b\},$$

$P = \{S \rightarrow CD, C \rightarrow aCA \mid bCB \mid X, X \rightarrow aX \mid bX \mid \epsilon, AD \rightarrow aD, BD \rightarrow bD, Aa \rightarrow aA, Ab \rightarrow bA, Ba \rightarrow aB, Bb \rightarrow bB, D \rightarrow \epsilon\};$

-vezi a) cu modificarea că C-ul după ce și-a îndeplinit rolul de mijloc al cuvântului va genera prin trecerea în X o secvență oarecare de simboluri a,b.

1.1.14.

a) $G = (N, \Sigma, P, S), N = \{S, A, B, C\}, \Sigma = \{a\},$

$P = \{S \rightarrow BAB, BA \rightarrow BC, CA \rightarrow AAC, CB \rightarrow AAB, A \rightarrow a, B \rightarrow \epsilon\}.$

Practic de la forma propozițională BAB se înlocuiește A cu doi de A cu ajutorul lui C trecându-l pe acesta de la stânga la dreapta. Se ajunge la forma propozițională BAAB de unde se poate ajunge la:

1°) cuvântul aa folosind ultimele două producții;

2°) sau la forma propozițională BAAAAB folosind producțiile:

$BA \rightarrow BC, CA \rightarrow AAC, CB \rightarrow AAB.$

Procesul se poate continua pentru a obține orice cuvânt din limbajul cerut.

Evident că există și o altă gramatică cu producțiile simetrice care generează același limbaj:

$P = \{S \rightarrow BAB, AB \rightarrow CB, AC \rightarrow CAA, BC \rightarrow BAA, A \rightarrow a, B \rightarrow \epsilon\}.$

sau

a) $G = (N, \Sigma, P, S)$

$N = \{S, A, B, C\}, \Sigma = \{a\};$

$P = \{S \rightarrow BAB, BA \rightarrow BC, CA \rightarrow AAC, CB \rightarrow AAB, A \rightarrow a, B \rightarrow \epsilon\}.$

-B=marcatori de stânga și dreapta ai cuvântului;

-A-ul cel mai din stânga e înlocuit cu C (s-a pierdut temporar un A);

-C parcurge de la stânga la dreapta șirul de A-uri și transformă fiecare A în AA;

-când C-ul ajunge la marginea dreaptă se reface A-ul pierdut prin AA.

exemplu: $w=a^4$

$S \underset{1}{=} > \underset{2}{BAB} = > \underset{4}{BCB} = > \underset{2}{BAAB} = > \underset{3}{BCAB} = > \underset{4}{BAACB} = > \underset{6}{BAAAAB} = > \underset{5}{AAAAAB} = > \underset{5}{aAAAB} = > \underset{5}{aaAAB} = > \underset{5}{aaaAB} = > \underset{6}{aaaaB} = > \underset{6}{aaaa}.$

(Am numerotat producțiile în ordinea scrierii lor în mulțimea P).

b) Folosind aceeași idee de la punctul a) obținem:

$G = (N, \Sigma, P, S)$

$N = \{S, A, B, C\}, \Sigma = \{a\},$

$P = \{S \rightarrow BAB, BA \rightarrow BC, CA \rightarrow A^k C, CB \rightarrow A^k B, A \rightarrow a, B \rightarrow \epsilon\}.$

1.1.15.

a) Dăm două gramatici echivalente:

$G_1 = (\{E\}, \{a\}, P, E)$ cu $P = \{E \rightarrow E+E \mid E^*E \mid (E) \mid a\}$ și
 $G_2 = (\{E, T, F\}, \{a\}, P, E)$ cu $P = \{E \rightarrow E+T \mid T, F \rightarrow (E) \mid a, T \rightarrow T^*F \mid F\}$.

b) Analog două gramatici echivalente:

$G_3 = (\{E\}, \{a\}, P, E)$ cu $P = \{E \rightarrow E+E \mid E-E \mid E^*E \mid E:E \mid (E) \mid a\}$ și
 $G_4 = (\{E, T, F\}, \{a\}, P, E)$ cu $P = \{E \rightarrow E+T \mid E-T \mid T, T \rightarrow T^*F \mid T:F \mid F, F \rightarrow (E) \mid a\}$.

1.1.16.

$G = (\{S, H\}, \{a, b, c, d\}, P, S)$ cu $P = \{S \rightarrow aaSd \mid X, X \rightarrow bXcc \mid bc\}$.

1.1.17.

a) Ideea folosită este următoarea: considerăm cuvintele formate din două secvențe. Prima secvență $a^n b^n$ iar a doua $c^m d^m$ apoi vom concatena cele două secvențe. Practic avem gramatica:

$G_1 = (\{S, X, Y\}, \{a, b, c, d\}, P, S)$ cu $P = \{S \rightarrow XY, X \rightarrow aXb \mid ab, Y \rightarrow cYd \mid cd\}$.

Neterminalul X generează secvențele $a^n b^n$, iar neterminalul Y generează secvențele $c^m d^m$.

b) Reluând ideea de la punctul a) avem gramatica:

$G_2 = (\{S, X, Y\}, \{a, b\}, P, S)$ cu $P = \{S \rightarrow XY, X \rightarrow aSb \mid ab, Y \rightarrow aYb \mid ab\}$.

1.1.18.

Folosind ideea de la problema 1.1.17. avem gramatica:

$G = (\{S, X, Y\}, \{a, b, c, d, e\}, P, S)$ cu
 $P = \{S \rightarrow XY, X \rightarrow aSBc \mid abc, cB \rightarrow Bc, bB \rightarrow bb, Y \rightarrow dYe \mid de\}$.

1.1.19.

a) $G = (N, \Sigma, P, S)$, $N = \{S, A, B\}$, $\Sigma = \{a, b, c\}$

$P = \{S \rightarrow aSBc \mid A, cB \rightarrow Bc, bB \rightarrow bb, aB \rightarrow ab, A \rightarrow aAb \mid \epsilon\}$.

exemplu: fie $w = a^3 b^3 c$

$S \xrightarrow{1} aSBc \xrightarrow{2} aABc \xrightarrow{6} aaAbBc \xrightarrow{6} aaaAbbBc \xrightarrow{7} a^3 bbBc \xrightarrow{4} a^3 b^3 c^3$.

(Am numerotat producțiile în ordinea scrierii lor în mulțimea P).

Prima producție generează formele propoziționale de forma $a^k S(Bc)^k$.

Producțiile numerotate cu 2 și 6 asigură multiplicarea $a^k b^k$ și astfel ne asigurăm că $n \geq m$.

Producția a 3-a inter schimbă c cu B .

Producțiile 4 și 5 transformă neterminalele B în b .

Producția a 7-a ajută la ștergerea lui A .

Se poate demonstra ușor că nu se generează și alte cuvinte decât cele cerute în enunț.

b) $G = (N, \Sigma, P, S)$, $N = \{S, B, C\}$, $\Sigma = \{a, b, c\}$,

$$P = \{S \rightarrow aSBc \mid C, aB \rightarrow ab, cB \rightarrow Bc, bB \rightarrow bb, C \rightarrow Cc \mid \epsilon\}.$$

exemplu: fie $w = a^2b^2c^4$

$$S \underset{1}{=} > aSBc \underset{1}{=} > a^2SBcBc \underset{2}{=} > a^2CBcBc \underset{6}{=} > a^2CcBcBc \underset{6}{=} > a^2CccBcBc \overset{+}{=} > a^2b^2c^4$$

(Am numerotat producțiile în ordinea scrierii lor în mulțimea P).

Idea folosită este asemănătoare cu cea folosită la punctul a).

Producțiile 2 și 6 permit să obținem relația $m \geq n$.

$$c) G = (N, \Sigma, P, S), \quad N = \{S, A, B, X\}, \quad \Sigma = \{a, b, c\},$$

$$P = \{S \rightarrow aSBc \mid X, cB \rightarrow Bc, bB \rightarrow bb, aB \rightarrow ab, X \rightarrow aXb \mid A, A \rightarrow aA \mid \epsilon\}.$$

exemplu: fie $w = a^3b^2c$

$$S \underset{1}{=} > aSBc \underset{2}{=} > aXBc \underset{6}{=} > aaXBcBc \underset{7}{=} > aaAbBc \underset{8}{=} > a^3AbBc \underset{9}{=} > a^3bBc \underset{4}{=} > a^3b^2c.$$

(Am numerotat producțiile în ordinea scrierii lor în mulțimea P asemănător punctului a).

$$d) G = (N, \Sigma, P, S), \quad N = \{S, B\}, \quad \Sigma = \{a, b, c\},$$

$$P = \{S \rightarrow aSBc \mid aSBcc \mid \epsilon, aB \rightarrow ab, cB \rightarrow Bc, Bc \rightarrow bc, bB \rightarrow bb\}$$

exemplu: fie $w = a^2b^2c^3$

$$S \underset{1}{=} > aSBc \underset{2}{=} > aaSBccBc \underset{3}{=} aaBccBc \underset{6}{=} > aabccBc \underset{5}{=} > a^2bcBcc \underset{5}{=} > a^2bBccc \underset{7}{=} > a^2b^2c^3.$$

(Am numerotat producțiile în ordinea scrierii lor în mulțimea P asemănător punctului a).

1.1.20.

$$a) G = (N, \Sigma, P, S)$$

$$N = \{S\}, \quad \Sigma = \{a, b, c\}, \quad P = \{S \rightarrow aSc \mid cSa \mid bSc \mid cSb \mid SS \mid \epsilon\};$$

Primele două producții permit ca atunci când în forma propozitională se introduce un **a** să se introducă și câte un **c** în orice ordine.

Producțiile 3 și 4 permit ca atunci când în forma propozitională se introduce un **b** să se introducă și câte un **c** în orice ordine.

Producția a 6-a permite eliminarea lui S.

Exemplu: $w = acbc$.

$$S \underset{1}{=} > aSc \underset{4}{=} > acSbc \underset{6}{=} > acbc.$$

(Am numerotat producțiile în ordinea scrierii lor în mulțimea P).

$$b) G = (N, \Sigma, P, S)$$

$$N = \{S, A, B, C\}, \quad \Sigma = \{a, b, c\},$$

$$P = \{S \rightarrow ASBC \mid \epsilon, AB \rightarrow BA, BA \rightarrow AB, AC \rightarrow CA, CA \rightarrow AC, CB \rightarrow BC, BC \rightarrow CB, A \rightarrow a, B \rightarrow b, C \rightarrow c\}.$$

exemplu: $w=cba$.

$$S \underset{1}{=} > ASBC \underset{2}{=} > ABC \underset{3}{=} > BAC \underset{5}{=} > BCA \underset{8}{=} > CBA \underset{9,10,11}{=}^+ > cba.$$

(Am numerotat producțiile în ordinea scrierii lor în mulțimea P).

1.1.21.

c) Demonstrăm prin dublă incluziune egalitatea celor două mulțimi mergând pe echivalențe:

$$x \in L_1/(L_2L_3) \Leftrightarrow \exists y \in L_2L_3 \text{ a.î. } xy \in L_1 \Leftrightarrow \exists y_2 \in L_2 \text{ și } y_3 \in L_3 \text{ a.î. } xy_2y_3 \in L_1 \Leftrightarrow \exists y_2 \in L_2 \text{ și } xy_2 \in L_1/L_3 \Leftrightarrow x \in (L_1/L_3)/L_2.$$

1.2. AUTOMATE FINITE

1.2.1. Automate finite deterministe (AFD), automate finite nedeterministe (AFN)

Un **automat finit (acceptor)** este un ansamblu $M = (Q, \Sigma, \delta, q_0, F)$ în care:

- Q este o mulțime finită și nevidă de elemente numite **stări**;
- Σ este o mulțime finită și nevidă numită **alfabet de intrare**;
- $\delta : Q \times \Sigma \rightarrow P(Q)$ este o aplicație numită **funcție de tranziție**;
- $q_0 \in Q$, q_0 este **stare inițială**;
- $F \subseteq Q$, F este o mulțime nevidă numită **mulțimea stărilor finale**.

Observație. În definiția de mai sus $P(Q)$ este mulțimea tuturor părților (submulțimilor) lui Q . Deci, în particular, $\emptyset \in P(Q)$ și $Q \in P(Q)$. Apoi, $\forall q \in Q, \forall a \in \Sigma$ avem $\delta(q,a) = Q_1 \subseteq Q$.

Fie M un automat finit (AF). Dacă $\forall q \in Q, \forall a \in \Sigma$ avem $|\delta(q,a)| \leq 1$ atunci automatul M se numește **automat finit determinist (AFD)**, iar în caz contrar automatul se numește **automat finit nedeterminist (AFN)**. Dacă $\forall q \in Q, \forall a \in \Sigma$ avem $|\delta(q,a)| = 1$, atunci automatul finit M se numește **automat finit determinist complet (total) definit**. În acest caz avem întotdeauna $\forall q \in Q, \forall a \in \Sigma \delta(q,a) = \{p\}, p \in Q$.

Observații: 1°. Fie $M=(Q, \Sigma, \delta, q_0, F)$ un AFD. Avem același automat și dacă funcția de tranziție $\delta: Q \times \Sigma \rightarrow P(Q)$ se înlocuiește printr-o **funcție parțial definită** pe $Q \times \Sigma$ de forma:

$$\delta': Q \times \Sigma \rightarrow Q \text{ unde } \forall q \in Q, \forall a \in \Sigma \text{ avem } \delta'(q,a) = \begin{cases} p, & \text{dacă } \delta(q,a) = \{p\} \\ -, & \text{dacă } \delta(q,a) = \emptyset \end{cases}.$$

2°. Dacă M este un AFD complet definit atunci evident funcția de tranziție δ' menționată la 1° va fi și ea complet definită.

3°. Întotdeauna și invers, de la funcția δ' definită la 1° se poate trece la δ .

1.2.2. Reprezentare

Un automat finit poate fi *reprezentat* sub formă de tablou sau sub forma unui graf orientat. Astfel, tabloul alăturat ne furnizează toate elementele care definesc un automat finit $M=(Q,\Sigma,\delta,q_0,F)$, unde $Q=\{q_0,\dots,q_n\}$, $\Sigma=\{a_1,\dots,a_m\}$ și în care se consideră:

$Q \backslash$	a_1	...	a_j	...	a_m
q_0					
...					
q_i	...		(q_i, a_j)	...	z_i
...					
q_n					

O altă reprezentare asociată automatului $M = (Q, \Sigma, \delta, q_0, F)$, este graful orientat cu noduri și arce etichetate astfel:

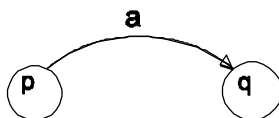
- mulțimea vârfurilor se etichetează cu stări, elemente ale lui Q ;
- starea inițială se reprezintă printr-un nod de forma:



- stările finale (deci $q \in F$) se reprezintă prin noduri de forma:



- dacă $\delta(p,a) \ni q$ atunci avem arcul, (p,q) care se etichetează cu a , adică:



1.2.3. Configurații și relații de tranziție

Fie $M = (Q, \Sigma, \delta, q_0, F)$. Mulțimea **configurațiilor** acestui automat este formată din $Q \times \Sigma^*$. Pe mulțimea configurațiilor automatului finit M se definesc relațiile binare:

- \mapsto tranziția directă;
- \xrightarrow{k} k tranziția;
- $\xrightarrow{+}$ + tranziția;
- $\xrightarrow{*}$ * tranziția.

Avem $(p, aw) \mapsto (q, w) \iff \delta(p, a) \ni q; p, q \in Q, a \in \Sigma, w \in \Sigma^*$.

Apoi $(p, w_1) \xrightarrow{k} (q, w_2)$ dacă și numai dacă de la prima configurație se poate ajunge la a doua configurație folosind k tranziții directe; $p, q \in Q; w_1, w_2 \in \Sigma^*$.

Între două configurații avem o "+ tranziție", dacă și numai dacă $\exists k > 0$ astfel încât între ele să avem o "k-tranziție", iar

$$(p, w_1) \xrightarrow{*} (q, w_2) \iff p = q, w_1 = w_2 \text{ adică } (p, w_1) = (p, w_2) \text{ sau } (p, w_1) \xrightarrow{+} (q, w_2); p, q \in Q, w_1, w_2 \in \Sigma^*.$$

Configurația (q_0, w) se numește **configurație inițială** iar $(q, \epsilon), q \in F$ se numește **configurație finală**.

1.2.4. Limbaj acceptat de un AF

Limbajul acceptat de automatul $M = (Q, \Sigma, \delta, q_0, F)$ este:

$$T(M) = \{w \mid w \in \Sigma^*, (q_0, w) \xrightarrow{*} (q, \epsilon), q \in F\}.$$

Două automate M_1 și M_2 sunt **echivalente** dacă ele acceptă același limbaj, adică $T(M_1) = T(M_2)$.

Automatele finite sunt o altă formă de **specificare** a unor limbaje. Relația de tranziție directă definește o **mișcare** a AF. Apoi, $w \in T(M)$, dacă și numai dacă în graful care definește automatul M , există un drum de la starea inițială la o stare finală și arcele sunt etichetate succesiv cu simbolurile cuvântului w .

1.2.5. Funcționare

Funcționarea unui automat finit $M = (Q, \Sigma, \delta, q_0, F)$ se petrece în *timp* și este definită de relația \mapsto^* .

Funcționarea automatului finit M care la momentul de timp t_i are configurația (q, w) , $w = aw'$, presupune trecerea la momentul de timp t_{i+1} sau oprirea acestuia. În acest proces de trecere deosebim următoarele situații:

- 1° *mișcare*, dacă $(q, aw') \mapsto (p, w')$ adică $\delta(q, a) \ni p$, deci la momentul t_{i+1} are configurația (p, w') ;
- 2° *blocare*, dacă $\delta(q, a) = \emptyset$;
- 3° *staționare*, dacă $(q, w) \mapsto^* (q, w)$.

La momentul inițial t_0 configurația automatului este (q_0, w) . Apoi, la funcționarea **AF** trebuie să avem în vedere că:

- momentele de timp luate în considerare sunt discrete, adică operațiile acestor mașini sunt *sincronizate*;
- trecerea de la o stare la alta a **AF** nu se face întâmplător;
- automatul răspunde (trece eventual într-o nouă stare) întotdeauna unui *simbol de intrare*;
- există un *număr finit de stări*, iar la un moment de timp dat, **AF** se poate găsi într-o singură stare.

O stare q a unui automat $M = (Q, \Sigma, \delta, q_0, F)$ este o *stare inaccesibilă* dacă și numai dacă nu există nici o secvență $w \in \Sigma^*$ astfel încât:

$$(q_0, w) \mapsto^* (q, \varepsilon);$$

în caz contrar *starea q este accesibilă*.

Automatul M poate fi transformat într-un **AF** echivalent, M' , care să conțină numai stări accesibile din starea inițială q_0 . Pentru determinarea lui $M' = (Q', \Sigma, \delta', q_0, F')$ se poate folosi algoritmul:

- a) $S_0 = \{q_0\}$, $i := 0$;
- b) $S_{i+1} = S_i \cup \{q \mid q \in Q, \delta(p, a) \ni q, \forall p \in S_i, a \in \Sigma\}$;
- c) Dacă $S_{i+1} \neq S_i$, atunci $i := i + 1$, salt la b);
- d) $Q' = S_{i+1}$, $F' = F \cap S_{i+1}$ iar $\delta'(q, a) = \delta(q, a)$, $\forall q \in Q', \forall a \in \Sigma$.

Fie $M = (Q, \Sigma, \delta, q_0, F)$. O stare $q \in Q$ este o *stare neproductivă* dacă și numai dacă nu există $w \in \Sigma^*$ astfel încât:

$$(q, w) \mapsto^* (p, \varepsilon), p \in F;$$

în caz contrar starea q este o *stare productivă*.

Automatul M poate fi transformat într-un AF echivalent, M' , care să conțină numai stări productive. Pentru determinarea lui $M' = (Q', \Sigma, \delta', q_0, F')$ se poate folosi algoritmul:

- a) $A_0 = F, i := 0$;
- b) $A_{i+1} = A_i \cup \{p \mid p \in Q, q \in \delta(p, a), a \in \Sigma, q \in A_i\}$;
- c) dacă $A_{i+1} \neq A_i$, atunci $i := i+1$, salt la b);
- d) $Q' = A_{i+1}, F' = F, \delta'(q, a) = \{p \in Q' \mid p \in \delta(q, a)\}, \forall q \in Q', \forall a \in \Sigma$.

Observație: Dacă $q_0 \notin Q'$ atunci $T(M) = \emptyset$, adică automatul M nu acceptă nici o secvență.

1.2.6. Echivalența dintre AFD și AFN

Teoremă. Fie M_1 un automat finit (nedeterminist). Întotdeauna există un automat finit determinist M_2 , astfel încât cele două automate să fie echivalente, adică:

$$T(M_1) = T(M_2).$$

Construcția AFD.

Se dă $M_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$. Atunci AFD echivalent cu M_1 este $M_2 = (Q_2, \Sigma, \delta_2, q'_0, F_2)$ unde:

$$Q_2 = P(Q_1), \quad q'_0 = \{q_0\},$$

$$F_2 = \{S \mid S \subseteq Q_1, S \cap F_1 \neq \emptyset\},$$

$$\text{iar } \forall a \in \Sigma \text{ și } \forall S \subseteq Q_1, S \neq \emptyset, \text{ avem } \delta_2(S, a) = \bigcup_{q \in S} \delta_1(q, a), \text{ și } \delta_2(\emptyset, a) = \emptyset$$

unde $\delta_2: Q_2 \times \Sigma \rightarrow Q_2$.

Observații.

1°. Funcția de tranziție δ_2 a AFD, M_2 , dat mai sus, este complet definită.

2°. Elementele $S_k, k \geq 0$ (în număr finit) ale mulțimii Q_2 și valorile funcției $\delta_2: Q_2 \times \Sigma \rightarrow Q_2$ care definesc automatul M_2 se pot obține din automatul M_1 și astfel:

- $S_0 = \{q_0\}, \delta_2(S_0, a) = \delta_1(q_0, a), a \in \Sigma$.
- fiecare submulțime $\delta_1(q, a)$ a lui Q_1 , dacă este diferită de submulțimile deja determinate se notează cu $S_i, i \geq 1$.
- pentru fiecare nouă stare S_k a automatului M_2 se calculează:

$$\delta_2(S_k, a) = \bigcup_{q \in S_k} \delta_1(q, a), \quad \forall a \in \Sigma;$$

această stare S_k este un candidat la mulțimea stărilor lui M_2 în sensul precizat mai sus; procesul se termină când nu mai obținem stări noi; evident că $F_2 = \{S \mid S \subseteq Q_2, S \cap F_1 \neq \emptyset\}$.

3°. Teorema de mai sus justifică considerarea în continuare a automatelor finite deterministe complet definite, fără a avea o limitare a unor rezultate teoretice.

1.2.7. Minimizarea automatelor finite

Fie automatul finit determinist (AFD) complet definit: $M = (Q, \Sigma, \delta, q_0, F)$.

Două stări distincte q_1 și q_2 din Q sunt **diferențiate** de $x, x \in \Sigma^*$, dacă în relațiile:

$$(q_1, x) \xrightarrow{*} (q_3, \epsilon); (q_2, x) \xrightarrow{*} (q_4, \epsilon),$$

q_3 și q_4 nu aparțin simultan lui F sau $Q - F$, ori dacă cel puțin una din aceste relații nu are loc.

Două stări distincte $q_1, q_2 \in Q$ sunt **k-diferențiate** de $x, x \in \Sigma^*$ dacă și numai dacă au loc condițiile de mai sus și $|x| \leq k$.

Două stări distincte $q_1, q_2 \in Q$ sunt **echivalente** dacă și numai dacă nu există nici o secvență care să le diferențieze; notăm acest fapt prin $q_1 \equiv q_2$.

Două stări distincte $q_1, q_2 \in Q$ sunt **k-echivalente** dacă și numai dacă nu există nici o secvență $x, |x| \leq k$ care să le diferențieze; notăm acest fapt prin:

$$q_1 \overset{k}{\equiv} q_2.$$

Observație. Dacă $q_1 \overset{0}{\equiv} q_2$ atunci q_1 și q_2 aparțin simultan lui F sau lui $Q - F$.

Lemă. Fie M un AFD cu n stări. Stările q_1 și q_2 sunt echivalente dacă și numai dacă ele sunt $(n-2)$ -echivalente.

Observație. Avem $\overset{n-2}{\equiv} \subseteq \overset{n-3}{\equiv} \subseteq \dots \subseteq \overset{1}{\equiv} \subseteq \overset{0}{\equiv}$.

Un AFD este un **automat redus** dacă nu conține stări inaccesibile, stări neproductive și nu conține perechi de stări distincte echivalente.

Teoremă. Oricare ar fi un automat finit M_1 , întotdeauna există un automat finit redus M' astfel încât: $T(M_1) = T(M')$.

1.2.8. Construcția AFD redus

Se dă $M_1 = (Q_1, \Sigma, \delta_1, q'_0, F_1)$.

- Transformăm acest automat, într-un AFD fără stări inaccesibile și neproductive (vezi algoritmi anteriori); notăm automatul astfel obținut $M = (Q, \Sigma, \delta, q_0, F)$;
- Construim relațiile de echivalență $\overset{0}{\equiv}, \overset{1}{\equiv}, \dots$ și folosim lema enunțată mai sus;

- Automatul redus $M' = (Q', \Sigma, \delta', q'_0, F')$ are:
 - $Q' = Q/\equiv$ mulțimea claselor de echivalență;
notăm clasa care conține starea p astfel: $[p]$.
 - $\delta'([p], a) = [q]$ dacă $\delta(p, a) = q$;
 - $q'_0 = [q_0]$;
 - $F' = \{[q] \mid q \in F\}$.

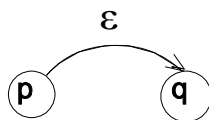
Teoremă. Automatul redus are numărul minim de stări, dintre toate automatele deterministe echivalente.

1.2.9. Automate finite cu ε -mișcări

Un AFN cu ε -mișcări este un ansamblu $M = (Q, \Sigma, \delta, q_0, F)$ ale cărei componente au aceleași semnificații ca la un AF oarecare, doar că funcția de tranziție este definită astfel:

$$\delta: Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow P(Q).$$

Avem o **ε -tranziție** între două configurații (p, w) (q, w) dacă și numai dacă $\delta(p, \varepsilon) \ni q$; $p \in Q$, $q \in Q$, $w \in \Sigma^*$; adică în reprezentarea grafică avem o legătură de următoarea formă:



Într-un AFN cu ε -mișcări se pot pune în evidență printr-un procedeu analog cu cel de la observația 2° de mai sus, toate stările accesibile dintr-o stare q , folosind numai ε -tranziții.

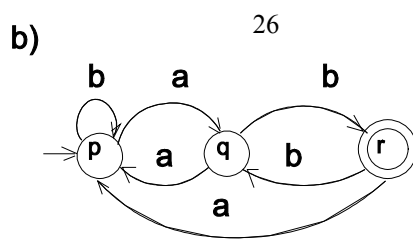
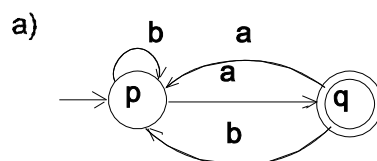
Teoremă. Dacă L este un limbaj acceptat de un AFN cu ε -mișcări, M_1 , întotdeauna există un AFN fără ε -mișcări, M_2 , echivalent cu M_1 , adică:

$$L = T(M_1) = T(M_2).$$

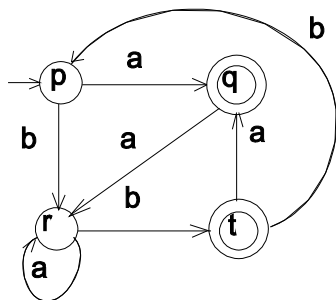
1.2.10. Probleme propuse

1.2.1.

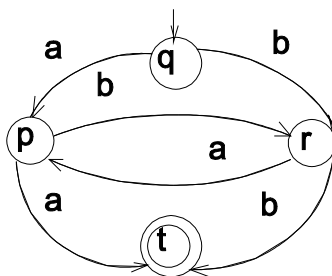
Să se reprezinte tabelar următoarele AF:



c)



d)

**1.2.2.**

Să se reprezinte sub formă de graf următoarele AF:

a) $M = (\{p, q, r\}, \{a, b\}, \delta, p, \{p\})$

	A	b	
p	Q	p	1
q	R	p	0
r	P	r	0

b) $M = (\{p, q, r, t\}, \{a, b, c\}, \delta, p, \{q, r\})$

	a	B	c	
p	q	P	r	0
q	p	T	p	1
r	t	R	p	1
t	q	p	q	0

1.2.3.

Să se reprezinte sub formă de graf automatul finit: $M = (Q, \Sigma, \delta, q_0, \{q_0\})$

unde $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$, iar funcția $\delta: Q \times \Sigma \rightarrow Q$ este dată de tabloul următor:

	0	1
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Verificați apoi pe graful obținut că:

a) secvențele 1010, 1100 sunt acceptate de automat;

b) secvența 1011 nu este acceptată de automat.

1.2.4.

Să se construiască AF care acceptă limbajul L definit peste alfabetul $\{0, 1\}$ și care are proprietatea că orice cuvânt al limbajului conține cel puțin două zerouri consecutive.

1.2.5.

Să se construiască automatul finit care acceptă limbajul:

$$L = \{0^n 1^m \mid n \geq 0, m \geq 1\} \cup \{1^n 0^m \mid n \geq 0, m \geq 1\}.$$

1.2.6.

Să se construiască automatele care acceptă limbajele:

a) $L_1 = \{0(10)^n \mid n \geq 0\};$

b) $L_2 = \{1(01)^n \mid n \geq 0\};$

Să se precizeze apoi, automatul redus care acceptă limbajul $L_1 L_2$.

1.2.7.

Să se construiască automatele care acceptă următoarele limbaje (peste $\{a,b\}^*$) descrise narativ:

a) L_1 format din secvențe care conțin un număr par de **a**-uri și număr par de **b**-uri;

b) L_2 format din secvențe care conțin un număr par de **a**-uri și număr impar de **b**-uri;

c) L_3 format din secvențe care conțin un număr impar de **a**-uri și număr par de **b**-uri;

d) L_4 format din secvențe care conțin un număr impar de **a**-uri și număr impar de **b**-uri;

1.2.8.

Să se construiască automatele care acceptă următoarele limbaje (peste $\{a,b,c\}^*$) descrise narativ:

a) L_1 format din secvențe care conțin un număr par de **a**-uri și număr par de **b**-uri;

b) L_2 format din secvențe care conțin un număr par de **a**-uri și număr impar de **b**-uri;

c) L_3 format din secvențe care conțin un număr impar de **a**-uri și număr par de **b**-uri;

d) L_4 format din secvențe care conțin un număr impar de **a**-uri și număr impar de **b**-uri;

1.2.9.

Să se construiască automatele care acceptă următoarele limbaje (peste $\{a,b,c\}^*$) descrise narativ:

a) L_1 format din secvențe care conțin un număr par de **a**-uri, **b**-uri și **c**-uri;

b) L_2 format din secvențe care conțin un număr par de **a**-uri și **b**-uri și un număr impar de **c**-uri;

c) L_3 format din secvențe care conțin un număr impar de **a**-uri și un număr par de **b**-uri și **c**-uri;

d) L_4 format din secvențe care conțin un număr impar de **a**-uri **b**-uri și **c**-uri;

1.2.10.

Să se construiască automatele M care acceptă următoarele limbaje:

a) $T(M) = \{w_1 a a w_2 \mid w_1 \in \{b, ab\}^*, w_2 \in \{a, b\}^*\};$

b) $T(M) = \{w_1 a a a w_2 \mid w_1, w_2 \in \{a, b\}^*\};$

c) $T(M) = \{1^n 0^m 1 u \mid n \geq 0, m \geq 1, u \in \{0, 1\}^*\};$

d) $T(M) = \{0^n 1^m 0^q \mid n, m, q \geq 1\};$

e) $T(M) = \{0^n 1^m 0^q \mid n, m \geq 1, q \geq 0\};$

1.2.11.

Să se construiască automatele M care acceptă următoarele limbaje:

a) $T(M) = \{w \mid w \in \{b,ccc\}^*\};$

b) $T(M) = \{c^{3n} \mid n \geq 1\};$

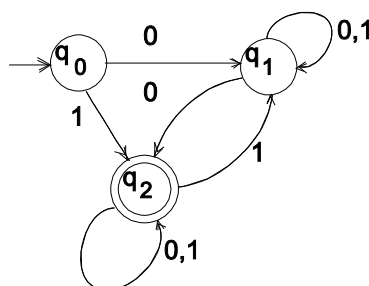
c) $T(M) = \{a,bc\}^*;$

d) $T(M) = \{bc, c, d\}^*;$

e) $T(M) = \{0(10)^n 01^m \mid n \geq 0, m \geq 0\} \cup \{(10)^n 01^m \mid n \geq 1, m \geq 0\}$ cu cel mult 4 stări;

1.2.12.

Se da automatul finit nedeterminist reprezentat mai jos:



Să se construiască AFD echivalent.

1.2.13.

Fie L_i mulțimea de secvențe acceptate de automatele finite deterministe $M_i = (Q_i, \Sigma, \delta_i, q_i^0, F_i)$ unde $\delta_i: Q_i \times \Sigma \rightarrow Q_i; i=1,2$.

Fie $M = (Q_1 \times Q_2, \Sigma, \delta, q_0, F)$ unde $\delta((p,q),a) = (\delta_1(p,a), \delta_2(q,a))$ iar $q_0 = (q_1^0, q_2^0)$,

$F = \{(p,q) \mid p \in F_1 \text{ și } q \in F_2\}$.

Arătați că $T(M) = L_1 \cap L_2$.

1.2.14.

Fie L_i mulțimea de secvențe acceptate de automatul finit determinist $M_i = (Q_i, \Sigma, \delta_i, q_i^0, F_i)$ unde $\delta_i: Q_i \times \Sigma \rightarrow Q_i, i=1,2$.

Fie $M = (Q_1 \times Q_2, \Sigma, \delta, q_0, F)$ unde $\delta((p,q),a) = (\delta_1(p,a), \delta_2(q,a))$ iar $q_0 = (q_1^0, q_2^0)$,

$F = \{(p,q) \mid p \in F_1 \text{ sau } q \in F_2\}$.

Arătați că $T(M) = L_1 \cup L_2$.

1.2.15.

Să se construiască automate care acceptă următoarele limbaje:

a) numerele întregi definite în limbajele C și PASCAL;

b) numerele reale definite în limbajele C și PASCAL;

c) declarațiile tablourilor în C;

d) declarațiile tablourilor în PASCAL;

1.2.16.

Să se construiască un automat finit peste alfabetul $\{0,1,\dots,9\}$ care să accepte numai cuvinte cu proprietățile:

- primul caracter este 0;
- al i -lea caracter este o cifră mai mare sau egală decât oricare din cele $i-1$ cifre anterioare.

1.2.17.

Să se construiască automate finite care acceptă limbajele următoare:

- a) mulțimea cuvintelor peste alfabetul $\{a,b\}$ în care orice două a -uri sunt separate printr-o secvență de lungime $4k$, $k > 0$ de simboluri b .
- b) mulțimea cuvintelor peste alfabetul $\{0,1\}$ cu cel mult două zerouri consecutive și cel mult două 1-uri consecutive.
- c) mulțimea cuvintelor peste alfabetul $\{0,1\}$ în care orice pereche de 0-uri alăturate apare înaintea oricărei perechi de 1 alăturate.
- d) mulțimea cuvintelor peste alfabetul $\{a,b\}$ a căror lungime este divizibilă cu trei.
- e) mulțimea cuvintelor care încep cu 1 și reprezintă scrierea în binar a tuturor numerelor divizibile cu cinci.
- f) mulțimea cuvintelor peste alfabetul $\{a,b,c\}$ care au aceeași valoare când le evaluăm de la stânga la dreapta ca și de la dreapta la stânga conform următoarei table de operații:

	a	b	c
a	a	a	c
b	c	a	b
c	b	c	a

1.2.18.

Să se construiască automatele finite care acceptă numai cuvinte peste alfabetul $\{a,b,c\}$ cu proprietatea că:

- a) simbolul cu care se termină cuvântul mai apare cel puțin o dată în cuvânt;
- b) simbolul cu care începe cuvântul este același cu cel cu care se termină cuvântul;
- c) simbolul cu care începe cuvântul mai apare cel puțin o dată în cuvânt;
- d) există un simbol în cuvânt care mai apare cel puțin o dată în cuvânt.

1.2.19.

Câte limbaje diferite definite peste alfabetul $\{0,1\}$ sunt acceptate de toate automatele finite cu două stări dacă:

- a) ele sunt deterministe;
- b) ele sunt nedeterministe;

1.2.11. Indicații și soluții

1.2.1.

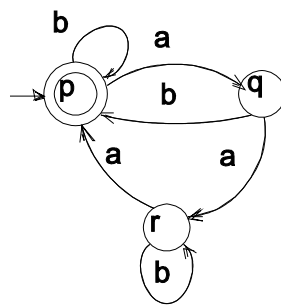
a) Automatul este complet definit iar reprezentarea tabelară este următoarea:

Q	a	b	
P	q	p	0
Q	p	p	1

b) c) și d) se reprezintă analog.

1.2.2.

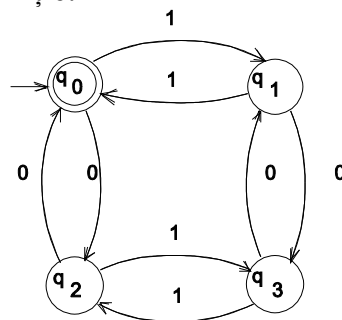
a) graful de tranziție este următorul:



Limbajul acceptat de automat este $T(M) = \{ \{b, ab, a^2b^n a\}^*, n \geq 0 \}$.

1.2.3.

Avem următorul graf de tranziție:

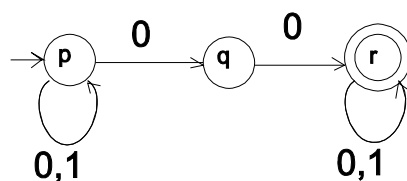


a) De la q_0 la q_0 există drumuri cu arcele etichetate 1010 respectiv 1100;

b) Nu există drum de la q_0 la q_0 cu arce etichetate în ordinea 1011.

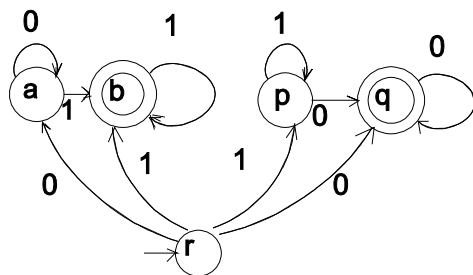
1.2.4.

Dăm reprezentarea automatului prin graful de tranziție:



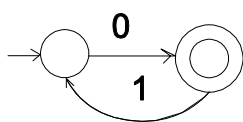
1.2.5.

Dăm reprezentarea automatului prin graful de tranziție:

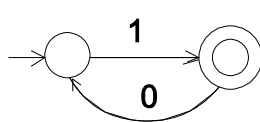


1.2.6.

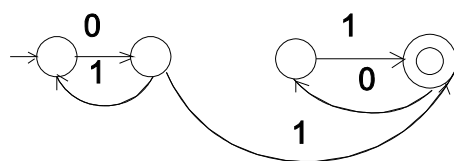
a)



b)

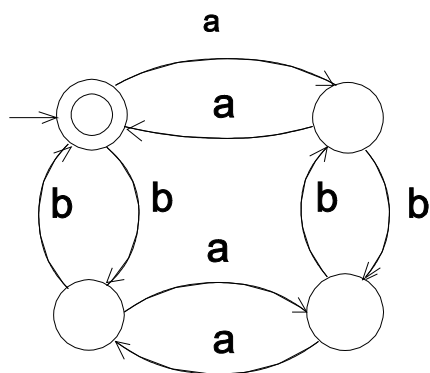


Limbajul L_1L_2 este acceptat de automatul de mai jos:



1.2.7.

Vom rezolva doar acest punct deoarece pentru a obține celelalte automate se modifică doar mulțimea stărilor finale ale automatului următor:

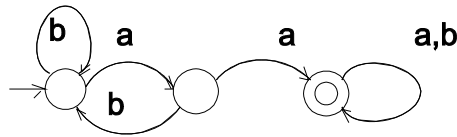


a) $F = \{p\}$; b) $F = \{t\}$; c) $F = \{q\}$; d) $F = \{r\}$.

1.2.8. și 1.2.9. Vezi problema 1.2.7., grafurile vor fi spațiale de forma unui cub. muchiile sale fiind etichetate cu simbolurile **a**, **b**, **c**.

1.2.10.

Vom rezolva doar punctul a), graful de tranziție al automatului care acceptă limbajul este:



1.2.11.

Vezi problemele 1.2.5. și 1.2.6.

1.2.12.

Avem $M = (Q, \Sigma, \delta, q_0, F)$ unde $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{0,1\}$, $F = \{q_2\}$, $\delta: Q \times \Sigma \rightarrow Q$ funcția de tranziție corespunzătoare. Vom nota cu $M' = (Q', \Sigma', \delta', q'_0, F')$ AFD astfel încât: $T(M) = T(M')$.

Construcția lui M' se face considerând:

$Q' = P(Q)$, $q'_0 = \{q_0\}$, $F' = \{S \mid S \subset Q, S \cap F \neq \emptyset\}$, $\delta': Q' \times \Sigma \rightarrow Q'$ cu

$$\delta'(S, a) = \{p \mid p \in Q, q \in S, a \in \Sigma, \delta(q, a) \ni p\} = \bigcup_{q \in S} \delta(q, a).$$

Se poate însă considera ca mulțimea a stărilor automatului M' , mulțimea $Q' \subseteq P(Q)$ construită pornind de la submulțimea $\{q_0\}$, căci altfel ar trebui să considerăm $|P(Q)| = 2^{|Q|}$ elemente.

Avem deci $S_0 = \{q_0\}$;

$$\delta'(S_0, 0) = \{q_1\} = S_1; \quad \delta'(S_0, 1) = \{q_2\} = S_2;$$

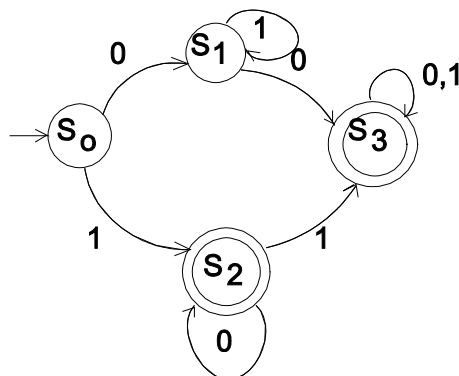
$$\delta'(S_1, 0) = \{q_1, q_2\} = S_3; \quad \delta'(S_1, 1) = \{q_1\} = S_1;$$

$$\delta'(S_2, 0) = \{q_2\} = S_2; \quad \delta'(S_2, 1) = \{q_1, q_2\} = S_3;$$

$$\delta'(S_3, 0) = \{q_1, q_2\} = S_3; \quad \delta'(S_3, 1) = \{q_1, q_2\} = S_3;$$

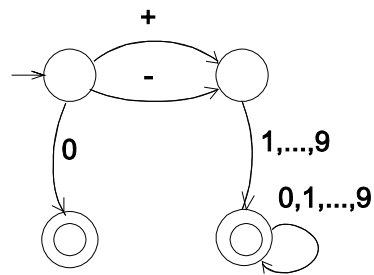
Astfel, $Q' = \{S_0, S_1, S_2, S_3\}$, $F' = \{S_2, S_3\}$

Reprezentarea grafului de tranziție este următoare:



1.2.15.

a) Vom da graful de tranziție al automatului ce acceptă numerele întregi:



1.2.16.

$M = (Q, \Sigma, \delta, q_0, F)$ unde:

$Q = \{q_0, q_1, \dots, q_{10}\}$, $\Sigma = \{0, 1, \dots, 9\}$, $F = Q \setminus \{q_0\}$

Funcția de tranziție este definită astfel:

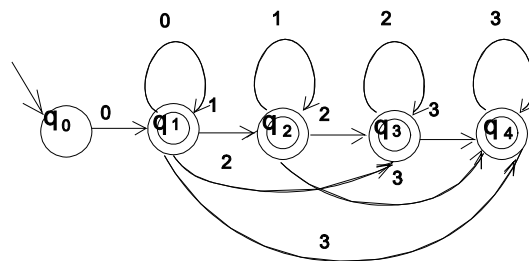
$\delta(q_0, 0) = q_1$;

$\delta(q_i, i-1) = q_i$, $i=1, \dots, 10$;

$\delta(q_i, j) = q_{j+1}$, $i=1, \dots, 9$, $j=i, \dots, 9$;

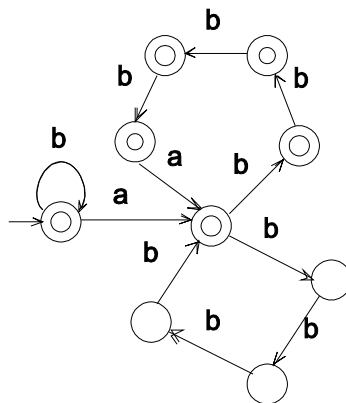
δ nu este definită în restul cazurilor.

Dacă $\Sigma = \{0,1,2,3\}$ atunci avem graful de tranziție:



1.2.17.

a) Vom da rezolvarea sub forma unui graf de tranziție:



1.3. EXPRESII REGULARE

Fie Σ un alfabet. **Mulțimile regulate** peste Σ se definesc recursiv astfel:

- 1) \emptyset este o mulțime regulată peste Σ ;
- 2) $\{\epsilon\}$ este o mulțime regulată peste Σ ;
- 3) Oricare ar fi $a \in \Sigma$, $\{a\}$ este mulțime regulată peste Σ ;
- 4) Dacă P și Q sunt mulțimi regulate peste Σ , atunci mulțimile:
 $P \cup Q$, PQ , și P^* sunt, de asemenea mulțimi regulate;
- 5) Orice altă mulțime regulată se obține aplicând de un număr finit de ori regulile 1)-4).

Fie Σ un alfabet. Definim recursiv **expresiile regulate** și **mulțimile regulate reprezentate de acestea**, astfel:

- 1) \emptyset este o expresie regulată reprezentând limbajul \emptyset ;
- 2) ϵ este o expresie regulată reprezentând limbajul $\{\epsilon\}$;
- 3) Dacă $a \in \Sigma$, atunci a este expresie regulată reprezentând limbajul a ;
- 4) Dacă p și q sunt expresii regulate reprezentând limbajele P și Q atunci:
 $(p+q)$ este expresie regulată reprezentând limbajul $P \cup Q$;
 (pq) este expresie regulată reprezentând limbajul PQ ;
 p^* este expresie regulată reprezentând limbajul P^* ;
- 5) Orice altă expresie regulată se obține aplicând de un număr finit de ori regulile 1) -4).

Observații. 1°. Se constată imediat că expresiile regulate peste alfabetul Σ sunt secvențe obținute prin concatenare de simboluri din alfabetul $\Sigma \cup \{\emptyset, \epsilon, +, *, (,)\}$;

2°. Limbajele asociate expresiilor regulate sunt mulțimi regulate;

3°. De multe ori simbolul "+" în scrierea expresiilor regulate se înlocuiește cu "|";

4°. Orice expresie regulată peste alfabetul Σ reprezintă un limbaj regulat peste alfabetul Σ .

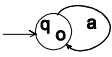

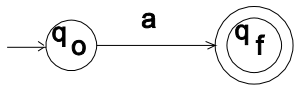
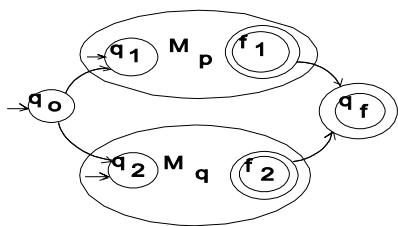
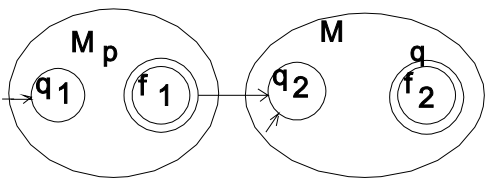
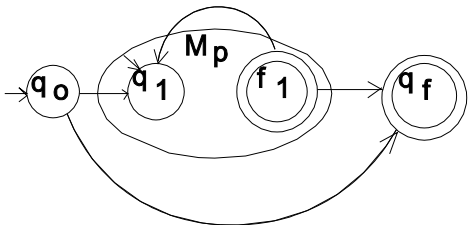
5°. Au loc următoarele identități (se pot demonstra):

- | | | |
|--------------------------|--------------------------------------|---|
| a) $r+s = s+r$; | f) $\emptyset^* = \epsilon$; | j) $\emptyset r = r\emptyset = \emptyset$; |
| b) $(r+s)+t = r+(s+t)$; | g) $(r^*)^* = r^*$; | k) $(r^*s^*) = (r+s)^*$; |
| c) $(rs)t = r(st)$; | h) $(\epsilon+r)^* = r^*$; | l) $r^*+\epsilon = \epsilon+r^*=r^*$; |
| d) $r(s+t) = rs+rt$; | i) $r+\emptyset = \emptyset+r = r$; | m) $\epsilon r = r\epsilon = r$ |
| e) $(r+s)t = rt+st$; | | |

Două expresii regulate sunt **egale** dacă limbajele reprezentate de acestea sunt egale.

Teoremă. Dacă r este o expresie regulată, atunci există un AFN care acceptă limbajul reprezentat de această expresie regulată și reciproc.

Construcția. Se construiește AFN cu ε -mișcări din aproape în aproape după următorul procedeu:

Expresia regulată	AFN	Observații
		
		
$a, a \in \Sigma$		<i>Expresia regulată a reprezintă limbajul {a} care este acceptat de acest automat</i>
$p+q$		<i>f_1 și f_2 nu vor mai fi stări finale; noua stare finală q_f, iar noua stare inițială este q_0</i>
pq		<i>f_1 nu mai este stare finală noua stare inițială rămâne q_1, noua stare finală f_2</i>
p^*		<i>f_1 nu mai este stare finală; noua stare inițială este q_0, noua stare finală q_f</i>

AFN cu ε -mișcări astfel obținut se poate transforma într-un AFN fără ε -mișcări. Invers, dacă se dă un AFD, M , determinarea expresiei regulate ce reprezintă limbajul acceptat de acest automat se poate face prin două metode:

1) Fie $M = (\{q_1, q_2, \dots, q_n\}, \Sigma, \delta, q_1, F)$

Se notează cu

$$R_{ij}^k = \{\alpha \mid (q_i, \alpha) \xrightarrow{*} (q_j, \varepsilon) \wedge [\forall \beta, \alpha = \beta\gamma, 0 < |\beta| < |\alpha|, (q_i, \beta) \xrightarrow{*} (q_s, \varepsilon) \Rightarrow s \leq k]\}$$

pentru care avem proprietatea:

$$R_{ij}^k = R_{ik}^{k-1} (R_{kk}^{k-1})^* R_{kj}^{k-1} \cup R_{ij}^{k-1}, \quad k \in \{1, 2, \dots, n\} \quad \text{iar} \quad R_{ij}^0 = \{a \mid \delta(q_i, a) \ni q_j\} \cup A \quad \text{unde}$$

$$A = \begin{cases} \emptyset, & \text{daca } i \neq j \\ \{\varepsilon\}, & \text{daca } i = j \end{cases}$$

În toate aceste relații $i = \overline{1, n}, j = \overline{1, n}$.

Apoi, R_{ij}^k este formată din toate cuvintele peste alfabetul Σ care duc automatul M din starea q_i în starea q_j fără să treacă prin nici o stare de indice mai mare decât k . În particular, R_{ij}^n este formată din toate cuvintele care duc automatul din starea q_i în starea q_j :

R_{ij}^n unde $q_j \in F$, este formată din toate cuvintele acceptate de automat în starea finală q_j .

$$\text{Avem } T(M) = \bigcup_{q_j \in F} R_{1j}^n$$

Fiecărei mulțimi regulate R_{ij}^k îi corespunde o expresie regulată r_{ij}^k ; aceste expresii regulate pot fi calculate folosind definiția recursivă a mulțimilor R_{ij}^k după formulele:

$$r_{ij}^0 = a_1 + \dots + a_s + x,$$

$$\text{unde } q_j \in \delta(q_i, a_m), m \in \{1, 2, \dots, s\} \quad \text{iar} \quad x = \begin{cases} \emptyset, & \text{daca } i \neq j \\ \varepsilon, & \text{daca } i = j \end{cases}; \quad \forall i, j \in \{1, 2, \dots, n\};$$

$$r_{ij}^k = r_{ik}^{k-1} (r_{kk}^{k-1})^* r_{kj}^{k-1} + r_{ij}^{k-1}, \quad \forall i, j, k \in \{1, 2, \dots, n\}.$$

În aceste relații se observă că a_1, a_2, \dots, a_s sunt toate simbolurile din Σ care duc automatul din starea q_i în starea q_j .

Limbajul acceptat de automatul M se poate specifica cu ajutorul expresiei regulate:

$$r_{1j_1}^n + r_{1j_2}^n + \dots + r_{1j_p}^n \quad \text{unde } F = \{q_{j_1}, q_{j_2}, \dots, q_{j_p}\}.$$

Observație: Expresia regulată obținută R_{ij}^k nu depinde de ordinea de numerotare a stărilor automatului, singura cerință este ca starea inițială să fie q_1 .

2) Fie $M = (\{q_1, q_2, \dots, q_n\}, \Sigma, \delta, q_1, F)$ un automat finit cu n stări și care nu conține stări inaccesibile.

Acestui automat i se va atașa următorul sistem de n ecuații liniare cu n necunoscute expresii regulate și coeficienți expresii regulate:

$$X_i = X_1 \alpha_1^i + \dots + X_n \alpha_n^i + \beta_i \quad \forall i \in \{1, 2, \dots, n\}, \quad \text{unde}$$

$$\alpha_k^i = a_{k_1}^i + \dots + a_{k_p}^i; \quad \text{cu } q_i \in \delta(q_k, a_{k_j}^i) \quad \forall j \in \{1, \dots, p\}, \quad \forall k \in \{1, \dots, n\}$$

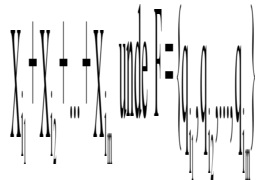
$$(a_{k_1}^i, \dots, a_{k_p}^i); \quad \text{sunt toate etichetele de pe arcu de la } q_k \text{ la } q_i;$$

$\alpha_k = \emptyset$ dacă nu există $a \in \Sigma$ astfel încât $q_i \in \delta(q_k, a)$;

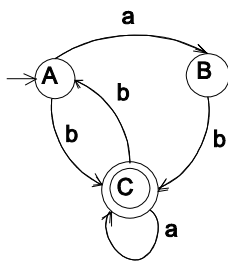
$$\beta_i = \begin{cases} \varepsilon & \text{dacă } q_i \text{ este stare inițială} \\ \emptyset & \text{altfel} \end{cases}$$

Acest sistem se poate rezolva folosind metoda eliminării a lui Gauss și reducând totul la rezolvarea de ecuații cu o singură necunoscută.

Expresia regulată atașată automatului va fi:



Exemplu: Fie automatul finit reprezentat mai jos. Să observăm că șirurile de simboluri ce aduc automatul în starea A sunt: ε , apoi $+$ (adică reunit cu) șirurile de forma $X_C b$ unde prin X_C am notat toate secvențele ce aduc automatul în starea C.



Avem: $X_A = \varepsilon + X_C b$

Analog obținem:
$$\begin{aligned} X_B &= X_A a \\ X_C &= X_B b + X_A b + X_C a \end{aligned}$$

Avem astfel un sistem de ecuații algebrice în care coeficienții se află la dreapta necunoscutelor.

Deducem succesiv valoarea lui X_C ce reprezintă chiar expresia regulată corespunzătoare limbajului acceptat de automatul finit dat:

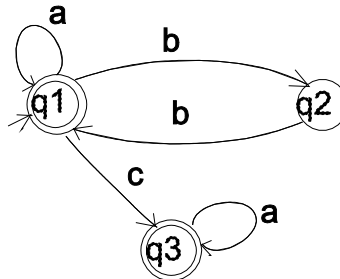
$$\begin{aligned} X_C &= X_A ab + X_A b + X_C a = X_A (ab + b) + X_C a = (\varepsilon + X_C b) (ab + b) + X_C a = \\ &= ab + b + X_C (bab + bb) + X_C a = ab + b + X_C (bab + bb + a). \end{aligned}$$

De unde $X_C = (ab + b)(bab + bb + a)^*$.

În final avem o ecuație de forma $X = X\alpha + \beta$. Soluția acestei ecuații este $X = \beta\alpha^*$ și ea este unică dacă $\varepsilon \notin \alpha$ (verificare: $\beta\alpha^* = \beta\alpha^* \alpha + \beta$ adică $\beta\alpha^* = \beta(\alpha^+ + \varepsilon)$ adică $\beta\alpha^* = \beta\alpha^*$). În cazul considerat avem $\alpha = bab + bb + a$ iar $\beta = ab + b$.

1.3.1. Probleme rezolvate

Problema 1. Să se construiască o expresie regulată corespunzătoare automatului finit dat prin graful de mai jos.



Vom folosi cele două metode:

Metoda 1:

Avem în vedere cele prezentate în partea de teorie privind mulțimile R_{ij}^k , expresiile r_{ij}^k etc.

Limbajul acceptat de automat se poate specifica cu ajutorul expresiei regulate: $r_{11}^3 + r_{13}^3$.

Vom construi tabelul:

	k=0	k=1	k=2	k=3
r_{11}^k	$a + \varepsilon$	a^*	$(a + bb)^*$	$(a + bb)^*$
r_{12}^k	b	a^*b	$(a + bb)^*b$	
r_{13}^k	c	a^*c	$(a + bb)^*c$	$(a + bb)^*ca$
r_{21}^k	b	ba^*	$b(a + bb)^*$	
r_{22}^k	ε	$ba^*b + \varepsilon$	$(ba^*b)^+ + \varepsilon$	
r_{23}^k	\emptyset	ba^*c	$b(a + bb)^*c$	
r_{31}^k	\emptyset	\emptyset	\emptyset	
r_{32}^k	\emptyset	\emptyset	\emptyset	
r_{33}^k	$a + \varepsilon$	$a + \varepsilon$	$a + \varepsilon$	

$$r_{11}^0 = a + \varepsilon, r_{12}^0 = b, r_{13}^0 = c, r_{21}^0 = b, r_{22}^0 = \varepsilon, r_{23}^0 = \emptyset, r_{31}^0 = \emptyset, r_{32}^0 = \emptyset, r_{33}^0 = a + \varepsilon;$$

$$\begin{aligned} r_{11}^1 &= r_{11}^0(r_{11}^0)^* + r_{11}^0 = (a + \varepsilon)(a + \varepsilon)^*(a + \varepsilon) + (a + \varepsilon) = (a + \varepsilon)((a + \varepsilon)^+ + \varepsilon) = \\ &= (a + \varepsilon)(a + \varepsilon)^* = (a + \varepsilon)a^* = a^+ + a^* = a^*; \end{aligned}$$

$$\begin{aligned} r_{12}^1 &= r_{11}^0 (r_{11}^0)^* r_{12}^0 + r_{12}^0 = (a + \varepsilon)(a + \varepsilon)^* b + b = (a + \varepsilon)^+ b + b = ((a + \varepsilon)^+ + \varepsilon) b = (a + \varepsilon)^* b = a^* b; \\ &= (a + \varepsilon)^+ c + c = ((a + \varepsilon)^+ + \varepsilon) c = (a + \varepsilon)^* c = a^* c; \end{aligned}$$

$$r_{21}^1 = r_{21}^0 (r_{11}^0)^* + r_{21}^0 = b(a + \varepsilon)^* (a + \varepsilon) + b = b(a + \varepsilon)^+ + b = b((a + \varepsilon)^+ + \varepsilon) = b(a + \varepsilon)^* = ba^*;$$

$$r_{22}^1 = r_{21}^0 (r_{11}^0)^* r_{12}^0 + r_{22}^0 = b(a + \varepsilon)^* b + \varepsilon = b(a + \varepsilon)^* + \varepsilon = ba^* b + \varepsilon;$$

$$r_{23}^1 = r_{21}^0 (r_{11}^0)^* r_{13}^0 + r_{23}^0 = b(a + \varepsilon)^* c + \emptyset = ba^* c;$$

$$r_{31}^1 = r_{31}^0 (r_{11}^0)^* r_{11}^0 + r_{31}^0 = \emptyset(a + \varepsilon)^* c + \emptyset = \emptyset + \emptyset = \emptyset;$$

$$r_{32}^1 = r_{31}^0 (r_{11}^0)^* r_{12}^0 + r_{32}^0 = \emptyset(a + \varepsilon)^* b + \emptyset = \emptyset + \emptyset = \emptyset;$$

$$r_{33}^1 = r_{31}^0 (r_{11}^0)^* r_{13}^0 + r_{33}^0 = \emptyset(a + \varepsilon)^* b + a + \varepsilon = \emptyset + a + \varepsilon = a + \varepsilon;$$

$$\begin{aligned} r_{11}^2 &= r_{12}^1 (r_{22}^1)^* r_{21}^1 + r_{11}^1 = a^* b(ba^* b + \varepsilon)^* ba^* + a^* = a^* b(ba^* b)^* ba^* + a^* = a^* (bba^*)^+ + a^* = \\ &= a^* (bba^*)^* = (a + bb)^* b; \end{aligned}$$

$$\begin{aligned} r_{12}^2 &= r_{12}^1 (r_{22}^1)^* r_{22}^1 + r_{12}^1 = a^* b(ba^* b + \varepsilon)^* (ba^* b + \varepsilon) + a^* b = a^* b((ba^* b + \varepsilon)^+ + \varepsilon) = a^* b(ba^* b + \varepsilon)^* = \\ &= a^* b(ba^* b)^* = (a + bb)^* b; \end{aligned}$$

$$r_{13}^2 = r_{12}^1 (r_{22}^1)^* r_{13}^1 + r_{12}^1 = a^* b(ba^* b + \varepsilon)^* ba^* c + a^* c = a^* b(ba^* b)ba^* c + a^* c = (a + bb)^* c;$$

$$r_{21}^2 = r_{22}^1 (r_{22}^1)^* r_{21}^1 + r_{21}^1 = (ba^* b + \varepsilon)(ba^* b + \varepsilon)^* ba^* + ba^* = (ba^* b)^* ba^* = b(a + bb)^*;$$

$$\begin{aligned} r_{22}^2 &= r_{22}^1 (r_{22}^1)^* r_{22}^1 + r_{22}^1 = (ba^* b + \varepsilon)(ba^* b + \varepsilon)^* (ba^* b + \varepsilon) + (ba^* b + \varepsilon) = (ba^* b + \varepsilon)((ba^* b + \varepsilon)^+ + \varepsilon) \\ &= (ba^* b + \varepsilon)(ba^* b + \varepsilon)^* = (ba^* b + \varepsilon) = (ba^* b)^+ + \varepsilon; \end{aligned}$$

$$\begin{aligned} r_{23}^2 &= r_{22}^1 (r_{22}^1)^* r_{23}^1 + r_{23}^1 = \\ &= (ba^* b + \varepsilon)(ba^* b + \varepsilon)^* ba^* c + ba^* c = (ba^* b + \varepsilon)^* ba^* c = (ba^* b)^* ba^* c = \\ &= b(a + bb)^* c; \end{aligned}$$

$$r_{31}^2 = r_{32}^1 (r_{22}^1)^* r_{12}^1 + r_{31}^1 = \emptyset(ba^* b + \varepsilon)^* ba^* + \emptyset = \emptyset + \emptyset = \emptyset;$$

$$r_{32}^2 = r_{32}^1 (r_{22}^1)^* r_{22}^1 + r_{32}^1 = \emptyset(ba^* b + \varepsilon)^* (ba^* b + \varepsilon) + \emptyset = \emptyset + \emptyset = \emptyset;$$

$$r_{33}^2 = r_{32}^1 (r_{22}^1)^* r_{23}^1 + r_{33}^1 = \emptyset(ba^* b + \varepsilon)^* ba^* c + a + \varepsilon = \emptyset + a + \varepsilon = a + \varepsilon;$$

$$r_{11}^3 = r_{13}^2 (r_{33}^2)^* r_{31}^2 + r_{11}^2 = (a + bb)^* c(a + \varepsilon)^* \emptyset + (a + bb)^* = (a + bb)^*;$$

$$\begin{aligned} r_{13}^3 &= r_{13}^2 (r_{33}^2)^* r_{33}^2 + r_{13}^2 = (a + bb)^* c(a + \varepsilon)^* (a + \varepsilon) + (a + bb)^* c = (a + bb)^* c(a + \varepsilon)^* = \\ &= (a + bb)^* ca^*; \end{aligned}$$

Expresia regulară corespunzătoare automatului dat este:

$$r_{11}^3 + r_{13}^3 = (a + bb)^* + (a + bb)^* ca^* = (a + bb)^* (\varepsilon + ca^*);$$

Metoda 2:

Această metodă presupune că automatul nu are stări inaccesibile. Atașăm automatului dat următorul sistem de ecuații liniare având coeficienți și necunoscute expresii regulate; fiecare ecuație corespunde unei stări a automatului:

$$\begin{cases} X = Xa + Yb + \varepsilon \\ Y = Xb \\ Z = Xc + Za \end{cases}$$

cu următoarele semnificații:

X reprezintă expresia regulară atașată mulțimii cuvintelor care aduc automatul în starea q_1 ;

Y reprezintă expresia regulară atașată mulțimii cuvintelor care aduc automatul în starea q_2 ;

Z reprezintă expresia regulară atașată mulțimii cuvintelor care aduc automatul în starea q_3 ;

Xa reprezintă X concatenat cu a;

+ reprezintă reuniunea;

X=Xa+Yb+ε are semnificația următoare : un cuvânt care aduce automatul în starea q_1 se poate obține în unul din următoarele moduri:

- dintr-un cuvânt care aduce automatul în starea q_1 concatenat cu simbolul "a";

Xa (bucla pe q_1 etichetată cu "a");

- dintr-un cuvânt care aduce automatul în starea q_2 concatenat cu simbolul "b";

Yb (arcul de la q_2 la q_1 etichetat cu "b");

- secvența vidă lasă automatul în starea q_1 pentru că q_1 este stare inițială.

Se va rezolva sistemul în necunoscutele **X,Y,Z**, iar expresia regulară atașată automatului va fi **X+Z** deoarece acestea corespund stărilor finale q_1 și q_3 .

Rezolvarea unui astfel de sistem ține cont de rezolvarea unei ecuații de forma următoare:

$$X = X\alpha + \beta, \text{ unde } \alpha \text{ și } \beta \text{ sunt expresii regulate}$$

Soluția acestei ecuații este **X = βα*** și ea este unică dacă $\varepsilon \notin \alpha$.

Rezolvăm sistemul astfel: înlocuim **Y** în prima ecuație și obținem **X = Xa + Xbb + ε** adică **X = X(a + bb) + ε** și soluția acestei ecuații este **X = ε(a + bb)*** adică **X = (a + bb)***.

Înlocuind valoarea lui **X** în ultima ecuație obținem **Z=Za+(a+bb)*c** adică **Z=(a+bb)*ca***.

Nu este nevoie să mai calculăm **Y**.

Deci expresia regulară atașată automatului dat este :

$$X + Z = (a + bb)^* + (a + bb)^* ca^* = (a + bb)^* (\varepsilon + ca^*)$$

și este aceeași cu cea obținută folosind prima metodă.

Problema 2:

Să se construiască o expresia regulară corespunzătoare automatului finit dat tabelar:

$Q \backslash \Sigma$	A	B	C	
Q_1	q_2	q_2	-	1
Q_2	q_3	-	q_2	0
Q_3	q_2	-	q_3	1

Folosim metoda a doua și atașăm acestui automat sistemul:

$$\begin{cases} X_1 = \varepsilon \\ X_2 = X_1(a + b) + X_2c + X_3a \\ X_3 = X_2a + X_3c \end{cases}$$

cu următoarele semnificații:

X_1 - reprezintă expresia regulară corespunzătoare mulțimii cuvintelor care aduc automatul în starea q_1 ; $X_1 = \varepsilon$ pentru că în starea q_1 e acceptată doar secvența vidă (q_1 este stare inițială);

X_2 - reprezintă expresia regulară corespunzătoare mulțimii cuvintelor care aduc automatul în starea q_2 ;

X_3 - reprezintă expresia regulară corespunzătoare mulțimii cuvintelor care aduc automatul în starea q_3 ;

Rezolvăm sistemul prin metoda substituției, înlocuim X_1 în ecuația a doua și obținem :

$X_2 = (a+b) + X_2c + X_3a$ și de aici $X_2 = (a+b+X_3a)c^*$, iar apoi înlocuim în ultima ecuație $X_3 = (a+b+X_3a)c^*a + X_3c$ și obținem:

$$X_3 = (a+b)c^*a(ac^*a+c)^*$$

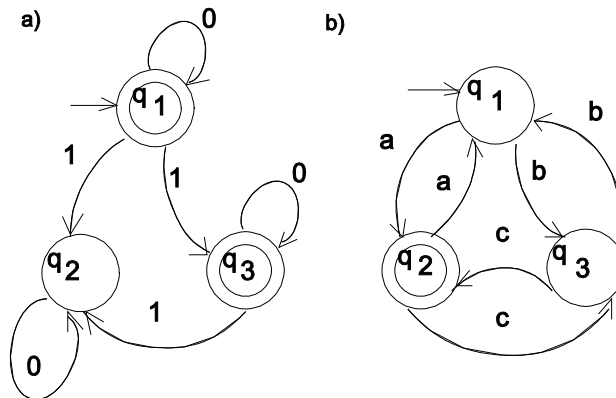
Nu mai calculăm valoarea efectivă a lui X_2 deoarece ea nu apare în expresia regulară finală (q_2 nu e stare finală). Expresia regulară corespunzătoare automatului dat este

$$X_1 + X_3 = \varepsilon + (a + b)c^*a(ac^*a + c)^*.$$

1.3.2. Probleme propuse

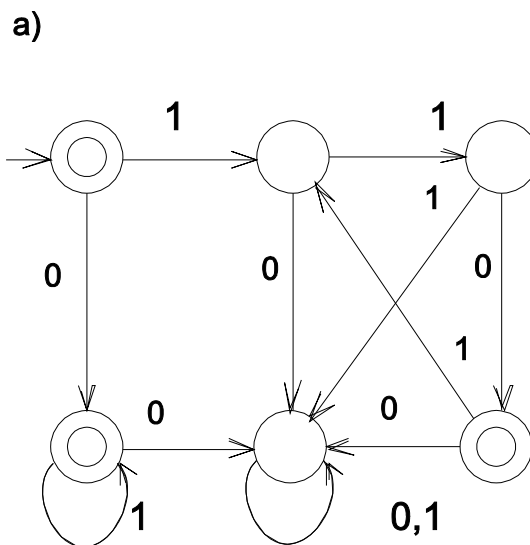
1.3.1.

Să se construiască expresii regulate care reprezintă limbaje acceptate de automatele finite următoare:

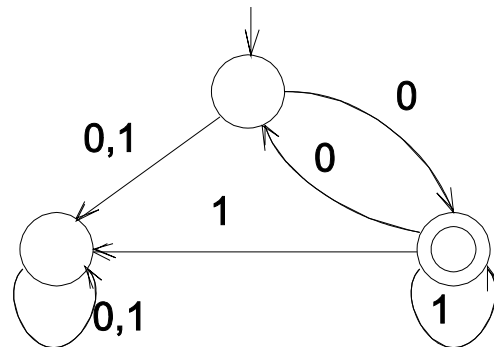


1.3.2.

Să se scrie expresiile regulate care reprezintă limbajele acceptate de automatele:



b)



1.3.3.

Dați o expresie regulată care să reprezinte mulțimea:

- a) $\{101, 1001, 10001, 100001, \dots\}$;
- b) tuturor secvențelor formate cu simbolurile **a** și **b** care conțin cel puțin un **a**;
- c) toate secvențele cu **a**-uri și **b**-uri care au cel puțin două **a**-uri consecutive.

1.3.4.

Precizați dacă secvențele ce urmează sunt elemente ale mulțimilor regulate reprezentate de expresiile regulate alăturate:

- a) 01110111; $(1^*01)^*(11+0)^*$;
- b) 11100111; $(1^*0)^*+(0^*11)^*$;
- c) 011100101; $01^*01^*(11^*0)^*$;
- d) 1000011; $(10^*+11)^*(0^*1)^*$;

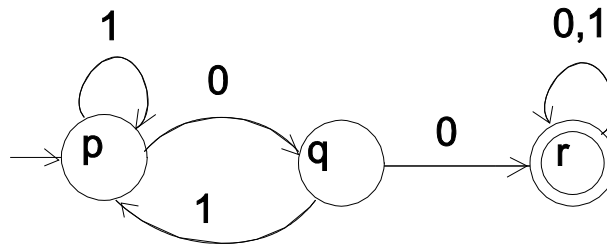
1.3.5.

Să se construiască automatele finite fără ϵ -tranziții corespunzătoare următoarelor expresii regulate:

- a) $(aa+b^*)c^+ + ((\epsilon+bc)aa)^*$;
- b) $\epsilon + (a+b+c)^* + ((ab+bc)(a^*+c^+))^+$;
- c) $01((10)^*+111)^*+0)^*1$;
- d) $(11+00)^*(01+10)^*$;
- e) $(aa+bb^*a+abb^*a)^*$;
- f) $10+(0+11)0^*1+1^+(10+01)$.

1.3.6.

Să se arate că automatul dat prin diagrama de tranziție de mai jos, acceptă limbajul specificat prin expresia regulată: $(01+1)^*00(0+1)^*$.



1.3.7.

Se dă AFN, $M_1 = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_0\})$ unde $\delta(q_0, 0) = \{q_0, q_1\}$, $\delta(q_0, 1) = \{q_1\}$, $\delta(q_1, 0) = \{q_0\}$, $\delta(q_1, 1) = \{q_0, q_1\}$. Găsiți AFD, M astfel ca $T(M) = T(M_1)$ și scrieți o expresie regulată care să reprezinte limbajul acceptat de cele două automate.

1.3.8.

Stabiliți dacă sunt adevărate proprietățile:

- a) $(rs+r)^*r = r(sr+r)^*$;
- b) $s(rs+s)^*r = rr^*s(rr^*s)^*$;
- c) $(r+s)^* = r^* + s^*$;

1.3.9.

Să se arate că dacă un limbaj este regulat, atunci există o infinitate de expresii regulate care îl specifică.

1.3.3. Indicații și soluții

1.3.2.

- a) $01^*(110)^*$; b) $0(00+1)^*$.

1.3.3.

- a) 100^*1 sau 10^+1 ;
- b) $(a+b)^*a(a+b)^*$;
- c) $(a+b)^*aa(a+b)^*$.

1.3.4.

- a) da; b) nu; c) nu; d) da.

1.3.7.

M	0	1	
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_1\}$	1
$\{q_1\}$	$\{q_0\}$	$\{q_0, q_1\}$	0
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_1\}$	1

expresia regulată: $(0^* + (0+1)1^*(0+1))^*$;

1.4. GRAMATICI ȘI LIMBAJE REGULARE

Producțiile unei *gramatici regulate* $G = (N, \Sigma, P, S)$ diferite de $S \rightarrow \varepsilon$ sunt numai de forma $A \rightarrow aB$, $A \rightarrow b$.

Limbajul generat de o gramatică regulată îl numim *limbaj regulat*.

1.4.1. Legătura dintre gramaticile regulate și automatele finite

Teoremă. Oricare ar fi o gramatică regulată $G = (N, \Sigma, P, S)$, există un automat finit $M = (Q, \Sigma, \delta, q_0, F)$ astfel încât limbajul acceptat de M să coincidă cu limbajul generat de G , adică $T(M) = L(G)$.

Construcția.

Se dă $G = (N, \Sigma, P, S)$. Automatul $M = (Q, \Sigma, \delta, q_0, F)$ cu proprietatea $T(M) = L(G)$ are:

$$Q = N \cup \{k\}, k \notin N;$$

Σ - același cu al gramaticii date;

$$q_0 = S;$$

$$F = \begin{cases} \{k\}, & \text{daca } (S \rightarrow \varepsilon) \notin P \\ \{S, k\}, & \text{daca } (S \rightarrow \varepsilon) \in P \end{cases};$$

$$\delta: Q \times \Sigma \rightarrow P(Q) \text{ cu } \delta(A, a) = \{B \mid (A \rightarrow aB) \in P\} \cup K \text{ unde}$$

$$K = \begin{cases} \{k\}, & \text{daca } (A \rightarrow a) \in P \\ \emptyset, & \text{altfel} \end{cases}; \quad \forall A, B \in N \text{ si } \forall a \in \Sigma.$$

$$\delta(k, a) = \emptyset, \forall a \in \Sigma.$$

Teoremă. (reciproca teoremei precedente)

Oricare ar fi un automat finit $M = (Q, \Sigma, \delta, q_0, F)$, există o gramatică regulată regulată $G = (N, \Sigma, P, S)$ astfel încât limbajul generat de G să coincidă cu limbajul acceptat de M , adică $L(G) = T(M)$.

Construcția.

Se dă $M = (Q, \Sigma, \delta, q_0, F)$. Gramatica $G = (N, \Sigma, P, S)$ cu proprietatea $L(G) = T(M)$ se construiește astfel:

$$N = Q;$$

$$S = q_0;$$

Σ - același cu al automatului dat;

$$P = \{A \rightarrow aB \mid \delta(A, a) \ni B\} \cup \{A \rightarrow a \mid \delta(A, a) \ni B, B \in F\}.$$

1.4.2. Proprietăți de închidere ale limbajelor regulate

Teoremă. Dacă L_1, L_2 sunt limbaje regulate peste alfabetul Σ , atunci

$$L_1 \cup L_2, L_1 \cap L_2, L_1 L_2, L_1^* = \bigcup_{n \geq 0} L_1^n, \overline{L_1} = \Sigma^* - L_1, \quad \text{sunt limbaje}$$

regulare.

Construcția.

Fie automatele: $M_1 = (Q_1, \Sigma, \delta_1, q'_0, F_1)$; $M_2 = (Q_2, \Sigma, \delta_2, q''_0, F_2)$; cu $L_1 = T(M_1)$, $L_2 = T(M_2)$.

a) Automatul care acceptă limbajul $L_1 \cup L_2$ este $M = (Q, \Sigma, \delta, q, F)$ unde:

$$Q = Q_1 \cup Q_2 \cup \{q_0\}, q_0 \notin Q_1 \cup Q_2;$$

$$F = \begin{cases} F_1 \cup F_2, & \text{daca } \varepsilon \notin L_1 \cup L_2 \\ F_1 \cup F_2 \cup \{q\}, & \text{daca } \varepsilon \in L_1 \cup L_2 \end{cases};$$

$$\delta(q_0, a) = \delta_1(q'_0, a) \cup \delta_2(q''_0, a), \forall a \in \Sigma;$$

$$\delta(q, a) = \begin{cases} \delta_1(q, a), & \text{daca } q \in Q_1 \\ \delta_2(q, a), & \text{daca } q \in Q_2 \end{cases}$$

b) Automatul care acceptă limbajul $L_1 L_2$ este $M = (Q, \Sigma, \delta, q_0, F)$ unde:

$$Q = Q_1 \cup Q_2;$$

$$q_0 = q'_0;$$

$$F = \begin{cases} F_2, & \text{daca } q''_0 \notin F_2 \\ F_1 \cup F_2, & \text{daca } q''_0 \in F_2 \end{cases}$$

$$\delta(q, a) = \begin{cases} \delta_1(q, a), & \text{daca } q \in Q_1 - \\ \delta_1(q, a) \cup \delta_2(q''_0, a), & \text{daca } q = q'_0 \\ \delta_2(q, a), & \text{daca } q \in Q_2 \end{cases}$$

c) Automatul care acceptă limbajul L_1^* este $M = (Q, \Sigma, \delta, q_0, F)$ unde:

$$Q = Q_1 \cup \{q_0\};$$

$$F = F_1 \cup \{q_0\};$$

$$\delta(q_0, a) = \delta_1(q'_0, a);$$

$$\delta(q,a) = \begin{cases} \delta_1(q,a), & \text{daca } q \in Q_1 - F_1 \\ \delta_1(q,a) \cup \delta_1(q_0'',a) & \text{daca } q \in F_1 \end{cases}.$$

d) Automatul care acceptă limbajul $\bar{L}_1 = \Sigma^* - L_1$ are în vedere faptul că automatul M_1 care acceptă limbajul L_1 trebuie să aibă proprietatea: $|\delta(s,a)|=1, \forall (s,a) \in Q \times \Sigma$, adică să fie total definit.

Atunci automatul care acceptă limbajul \bar{L}_1 notat $M = (Q, \Sigma, \delta, q_0, F)$ are $Q=Q_1$, $q_0 = q_0'$, $\delta = \delta_1$, $F = Q - F_1$.

Observație. Dăm o metodă de a construi automatul M_1 total definit echivalent cu M de la punctul d):

Fie $M = (Q, \Sigma, \delta, q_0, F)$ atunci automatul total definit echivalent cu M este:

$M_1 = (Q_1, \Sigma, \delta_1, q_0', F_1)$ astfel ca $T(M) = T(M_1)$, și are $Q_1 = Q \cup \{k\}$, $F_1 = F$ iar funcția

$\delta_1: Q_1 \times \Sigma \rightarrow P(Q_1)$, este definită astfel:

$$\delta_1(q,a) = \begin{cases} \delta(q,a), & \text{daca } |\delta(q,a)| \neq 0 \\ \{k\}, & \text{daca } |\delta(q,a)| = 0 \end{cases} \quad q \in Q, \forall a \in \Sigma.$$

$$\delta_1(q,k) = \{k\}, \forall a \in \Sigma.$$

Deci, pentru simbolurile $a \in \Sigma$ pentru care nu există arce corespunzătoare dintr-o stare q se construiesc arce spre k (o stare nou introdusă); totodată avem buclă pe k , $\forall a \in \Sigma$; starea k va fi nereproductivă.

e) Pentru construirea automatului care acceptă limbajul $L_1 \cap L_2$ se ține seama de relația $L_1 \cap L_2 = \overline{\bar{L}_1 \cup \bar{L}_2}$.

Există și altă metodă:

considerăm automatele care acceptă cele două limbaje

$$M_1 = (Q_1, \Sigma, \delta_1, q_0', F_1), M_2 = (Q_2, \Sigma, \delta_2, q_0'', F_2)$$

care sunt deterministe și total definite cu $\delta_i: Q_i \times \Sigma \rightarrow Q_i$; $i=1,2$. Atunci automatul M cu proprietatea că $T(M_1) \cap T(M_2) = T(M)$ este $M = (Q_1 \times Q_2, \Sigma, \delta, (q_0', q_0''), F_1 \times F_2)$ unde

$$\delta((p_1, p_2), a) = (\delta_1(p_1, a), \delta_2(p_2, a)) \quad \forall a \in \Sigma, p_1 \in Q_1, p_2 \in Q_2.$$

1.4.3. Lema de pompare pentru limbaje regulate

Propoziție. Dacă L este un limbaj regulat, atunci există un număr natural p astfel încât,

orice cuvânt $w \in L$ de lungime cel puțin p ($|w| \geq p$) să aibă o descompunere de forma $w=xyz$, unde $0 < |y| \leq p$ cu proprietatea că $xy^i z \in L, \forall i \in \mathbb{N}$.

O interpretare pe graful de tranziție: dacă cuvântul $w \in L$ are lungimea mai mare decât numărul stărilor (deci $p \geq |Q|$) atunci există un ciclu în graf.

Observație. În practică pentru a se demonstra că un limbaj nu este regulat se utilizează negația lemei de pompare. Dacă nu are loc lema de pompare limbajul nu este regulat. Atenție, lema dă o condiție necesară dar nu și suficientă: dacă condițiile din concluzie au loc nu e sigur că limbajul e regulat.

1.4.4. Probleme propuse

1.4.1.

Fie limbajele $L_1 = \{01^n 0 \mid n \geq 1\}$ și $L_2 = \{10^n 1 \mid n \geq 1\}$. Să se construiască automatele care acceptă limbajele:

$$L_1 \cup L_2, L_1 \cap L_2, L_1 L_2, L_1^*, L_2^*, \overline{L_1} = \Sigma^* - L_1, \overline{L_2} = \Sigma^* - L_2 \text{ unde } \Sigma = \{0,1\}.$$

1.4.2.

Să se arate că următoarele limbaje nu sunt regulate:

- $L = \{a^{n^2} \mid n \geq 1\}$;
- $L = \{a^k \mid k = \text{număr prim}\}$;
- $L = \{a^i b^j \mid \text{cmmdc}(i,j)=1\}$;
- $L = \{a^{2^n} \mid n \geq 0\}$;
- $L = \{a^n b^n \mid n \geq 1\}$.

1.4.3.

Să se arate că o gramatică care are numai producții de forma $A \rightarrow Ba$ și $A \rightarrow a$ este regulată.

1.4.4.

Să se arate că dacă L este un limbaj regulat, atunci $\tilde{L} = \{\tilde{w} \mid w \in L\}$ este regulat.

(\tilde{w} este oglinditul lui w , adică cuvântul citit de la sfârșit spre început).

1.4.5.

Să se arate că, dacă L (definit peste alfabetul Σ) este un limbaj regulat, atunci și următoarele limbaje sunt regulate:

- $\text{PRE}(L) = \{w \in \Sigma^* \mid \exists x \in \Sigma^*, wx \in L\}$ = mulțimea prefixelor cuvintelor din L ;
- $\text{SUF}(L) = \{w \in \Sigma^* \mid \exists x \in \Sigma^*, xw \in L\}$ = mulțimea sufixelor cuvintelor din L ;
- $\text{INT}(L) = \{w \in \Sigma^* \mid \exists x, y \in \Sigma^*, xwy \in L\}$ = mulțimea secvențelor interioare cuvintelor

din L .

1.4.6.

Fie L un limbaj regular acceptat de un automat finit determinist cu n stări. Atunci au loc:

1. $L \neq \emptyset$, L finit $\Leftrightarrow \exists w \in L$ cu $|w| < n$;
2. L infinit $\Leftrightarrow \exists w \in L$ cu $n \leq |w| < 2 \cdot n$.

Să se construiască un algoritm care să decidă dacă un limbaj regular specificat printr-un automat finit este finit sau nu.

1.4.7.

Să se construiască automate finite care acceptă limbajele de mai jos, apoi să se construiască pentru fiecare automat gramatica regulată corespondentă:

- a) mulțimea cuvintelor peste alfabetul $\{a, b\}$ în care orice două a -uri sunt separate printr-o secvență de lungime $4k$, $k \geq 1$ de simboluri b .
- b) mulțimea cuvintelor peste alfabetul $\{0, 1\}$ cu cel mult două zerouri consecutive și cel mult două 1 -uri consecutive.
- c) mulțimea cuvintelor peste alfabetul $\{0, 1\}$ în care orice pereche de 0 -uri alăturate apare înaintea oricărei perechi de 1 alăturate.
- d) mulțimea cuvintelor peste alfabetul $\{a, b\}$ cu lungimea divizibilă cu 3 .
- e) mulțimea cuvintelor care încep cu 1 și reprezintă scrierea în binar a tuturor numerelor divizibile cu 5 .
- f) mulțimea tuturor constantelor reale din limbajul " C ".
- g) mulțimea cuvintelor peste alfabetul $\{a, b, c\}$ care au aceeași valoare când le evaluăm de la stânga la dreapta ca și de la dreapta la stânga conform următoarei table de operații:

	a	b	c
a	a	a	c
b	c	a	b
c	b	c	a

1.4.8.

Să se construiască automatele finite care acceptă limbajele de mai jos, apoi să se construiască gramaticile regulate corespondente:

- a) $L_1 = \{ w \in \{a\}^* \mid nr_a(w) = \text{par} \}$;
- b) $L_2 = \{ w \in \{a, b\}^* \mid nr_a(w) = \text{par} \text{ și } nr_b(w) = \text{impar} \}$;
- c) $L_3 = \{ w \in \{a, b, c\}^* \mid nr_a(w) = \text{par} \text{ și } nr_b(w) = \text{impar} \text{ și } nr_c(w) = \text{par} \}$.

1.4.9.

Să se construiască un automat finit peste alfabetul $\{0, 1, \dots, 9\}$ care să accepte numai cuvinte cu proprietățile:

- primul caracter este 0 ;
- la i -lea caracter este o cifră mai mare sau egală decât oricare din cele $i-1$ cifre interioare.

Să se construiască gramatica regulară corespondentă.

1.4.10.

Să se construiască automatele finite care acceptă numai cuvinte peste alfabetul $\{a,b,c\}$ cu proprietatea că:

- a) simbolul cu care se termină cuvântul mai apare cel puțin o dată în cuvânt;
- b) primul simbol al cuvântului este același cu cel cu care se termină cuvântul;
- c) simbolul cu care începe cuvântul mai apare cel puțin o dată în cuvânt;
- d) există un simbol în cuvânt care mai apare cel puțin o dată în cuvânt.

Să se construiască în fiecare caz gramatica regulară corespondentă.

1.4.11.

Să se arate că următoarele limbaje sunt limbaje regulate:

- a) $L = \{a^n \mid n \geq 0\}$;
- b) $L = \{a^{kn} \mid n \geq 0, k \text{ fixat}\}$;
- c) $L = \{a^n b^m \mid n \geq 0, m \geq 0\}$;
- d) $L = \{ab^n cd^m \mid n \geq 0, m \geq 0\}$;
- e) $L = \{(bcd)^m \mid m \geq 0\}$;
- f) $L = \{(b^m cd^k)^n \mid m, n, k \geq 0\}$;

1.4.12.

Să se studieze dacă sunt regulate următoarele limbaje:

- a) $L = \{0^n 1^{2n} \mid n \geq 1\}$;
- b) $L = \{0^n 10^n \mid n \geq 1\}$;
- c) $L = \{ww \mid w \in \{0,1\}^*\}$;
- d) $L = \{(ab)^n (ba)^m \mid m, n \geq 1\}$;
- e) $L = \{(ab)^n c (ba)^m \mid m, n \geq 1\}$;
- f) $L = \{w^n \mid w \in \{ab, ba\}^*, n \geq 1\}$;
- g) $L = \{(uw)^n \mid u \in \{a,b\}^*, w \in \{ab, ba\}^*, n \geq 1\}$;

Să se aplice, apoi, lema de pompare pentru diferite secvențe din aceste limbaje.

1.4.5. Indicații și soluții

1.4.2.

a) Folosim lema de pompare, adică condiția de necesitate. Presupunem că L este regulat și fie p dat de lema de pompare.

Alegem $w \in L$ cu $|w| = p^2 \geq p$ și conform lemei există o descompunere a lui w de forma $w = xyz$ cu $0 < |y| \leq p$ și $xy^iz \in L, \forall i \geq 0$.

Fie $i=2$ și avem $|xy^2z| = |xyz| + |y| = p^2 + |y|$; de aici $p^2 < |xy^2z| \leq p^2 + p < (p+1)^2$, lungimea lui w nu e un pătrat perfect deci $w = xy^2z \notin L$, ceea ce contrazice ipoteza de plecare. În concluzie L nu este regulat.

b) Folosim lema de pompare, adică condiția de necesitate. Presupunem că L este regulă și fie p dat de lema de pompare.

Alegem $w \in L$ cu $w = a^k$, $k \geq p$ și conform lemei există o descompunere a lui w de forma $w = xyz$ cu $0 < |y| = a^i \leq p$ și $xy^iz = a^{k-j+i*j} = a^{k'} \in L \forall i \geq 0$.

Dacă alegem $i = k+1$ atunci $k' = k - j + (k+1)*j = k - j + k*j + j = k + k*j = k*(j+1)$ se poate descompune în factori, deci k' nu e un număr prim și $xy^iz \notin L$ ceea ce contrazice ipoteza de plecare. În concluzie L nu este un limbaj regulă.

1.4.3.

Fie $G = (N, \Sigma, P, S)$ cu producții de forma celor din enunț.

Vom construi o gramatică $G' = (N, \Sigma, P', S)$ echivalentă cu gramatica dată și care are producții numai de forma $A \rightarrow aB$ și $A \rightarrow a$, păstrând aceleași neterminale și același simbol de start.

Avem următoarele reguli de transformare:

- dacă $(A \rightarrow Ba) \in P$ și $A \neq S$ se construiește producția $(B \rightarrow aA) \in P'$;
- dacă $(A \rightarrow a) \in P$ și $A \neq S$ se construiește $(S \rightarrow aA) \in P'$;
- dacă $(S \rightarrow Aa) \in P$ se construiește $(A \rightarrow a) \in P'$;
- dacă $(S \rightarrow a) \in P$ rămâne această producție și în P' : $(S \rightarrow a) \in P'$;
- dacă $(S \rightarrow \varepsilon) \in P$ rămâne această producție și în P' : $(S \rightarrow \varepsilon) \in P'$.

Regulile enunțate transformă în mod unic producțiile de prima formă în producții de a doua formă. De fapt în definiția unei gramatici regulate se poate lua sau numai producții de prima formă sau numai producții de forma a doua.

Observație: Gramatica care are producții doar de forma $A \rightarrow aB$, $A \rightarrow a$ generează cuvintele de la stânga la dreapta, pe când gramatica care are doar producții de forma $A \rightarrow Ba$, $A \rightarrow a$ generează aceleași cuvinte de la dreapta spre stânga.

Exemplu:

Fie gramatica $G = (\{S, A, B\}, \{a, b\}, P, S)$ cu producțiile:

$P = \{S \rightarrow Bb \mid Ab, B \rightarrow Bb \mid Ab, A \rightarrow Aa \mid a\}$ care generează limbajul $L = \{a^i b^j \mid i, j > 0\}$ considerăm producțiile numerotate în ordinea scrierii lor.

Se observă că un cuvânt al limbajului se generează de la sfârșit spre început :

Fie $w = a^3 b^2$

$S \xrightarrow{1} Bb \xrightarrow{4} Abb \xrightarrow{5} Aabb \xrightarrow{5} Aaabb \xrightarrow{6} aaabb$

Conform regulilor de transformare de mai sus se pot obține următoarele producții:

- din (1) $(S \rightarrow Bb) \in P$ se obține $(B \rightarrow b) \in P'$ (1')
- din (2) $(S \rightarrow Ab) \in P$ se obține $(A \rightarrow b) \in P'$ (2')
- din (3) $(B \rightarrow Bb) \in P$ se obține $(B \rightarrow bB) \in P'$ (3')
- din (4) $(B \rightarrow Ab) \in P$ se obține $(A \rightarrow bB) \in P'$ (4')
- din (5) $(A \rightarrow Aa) \in P$ se obține $(A \rightarrow aA) \in P'$ (5')
- din (6) $(A \rightarrow a) \in P$ se obține $(S \rightarrow aA) \in P'$ (6')

w se va genera de la stânga spre dreapta astfel:

$$S \xrightarrow{6'} aA \xrightarrow{5'} aaB \xrightarrow{5'} aaaA \xrightarrow{4'} aaabB \xrightarrow{1'} aaabb$$

Ordinea aplicării producțiilor corespunzătoare din cele două gramatici pentru generarea aceluiași cuvânt este inversă.

1.4.4.

Dacă L este regular atunci există o gramatică regulară $G = (N, \Sigma, P, S)$ care îl generează și are numai producții de forma $A \rightarrow aB$, $A \rightarrow a$, $S \rightarrow \varepsilon$.

Construim gramatica $G' = (N, \Sigma, P', S)$ cu producțiile

$$P' = \{A \rightarrow Ba \mid (A \rightarrow aB) \in P\} \cup \{A \rightarrow a \mid (A \rightarrow a) \in P\} \cup \{S \rightarrow \varepsilon \mid (S \rightarrow \varepsilon) \in P\}.$$

Se demonstrează ușor că această gramatică generează \tilde{L} , iar conform problemei precedente această gramatică este regulară, deci limbajul generat de ea \tilde{L} , este regular.

1.4.5.

Dacă L este regular atunci există un automat finit $M = (Q, \Sigma, \delta, q_0, F)$ fără stări inaccesibile și neproductive care îl acceptă.

a) construim un automat M' din automatul M astfel încât toate stările automatului M vor deveni în M' stări finale

$$M' = (Q, \Sigma, \delta, q_0, F'), \quad F' = Q$$

M' va accepta limbajul format din mulțimea prefixelor (nu neapărat proprii) cuvintelor limbajului L .

b) construim un automat M'' care acceptă reuniunea tuturor limbajelor acceptate de M_i , $i=0,1, \dots, n$. M_i se obțin din M astfel încât fiecare stare a automatului devine pe rând stare inițială.

$$\text{Fie } Q = \{q_0, q_1, q_2, \dots, q_n\}, \text{ construim } M_i = (Q, \Sigma, \delta, q_i, F), \quad i = 0, 1, \dots, n.$$

$$M_0 = M$$

$$\text{SUF}(L) = T(M'') = \bigcup_{i=0}^n T(M)_i$$

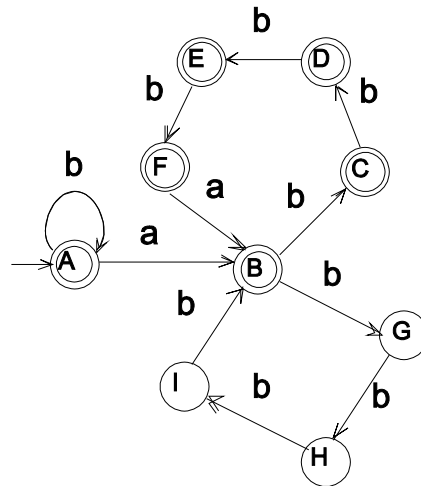
M'' va accepta limbajul format din mulțimea sufixelor (nu neapărat proprii) cuvintelor limbajului L .

Dacă dorim sufixele proprii eliminăm din reuniune M_0 ;

c) combinăm a) și b).

1.4.7.

a) Automatul construit este:



iar gramatica este $G = (\{A, B, C, D, E, F, G, H, I\}, \{a, b\}, P, A)$

unde $P: A \rightarrow bA \mid aB \mid a \mid b \mid \varepsilon$

$B \rightarrow bC \mid bG \mid b$

$C \rightarrow bD \mid b$

$D \rightarrow bE \mid b$

$E \rightarrow bF \mid b$

$F \rightarrow aB \mid a$

$G \rightarrow bH$

$H \rightarrow bI$

$I \rightarrow bB \mid b$

1.5. GRAMATICI ȘI LIMBAJE INDEPENDENTE DE CONTEXT

O gramatică $G = (N, \Sigma, P, S)$ este *independentă de context* dacă producțiile P ale acestei gramatici sunt numai de forma $A \rightarrow \alpha$, $A \in N$; $\alpha \in (N \cup \Sigma)^*$.

Gramaticile regulate sunt cazuri particulare de gramatici independente de context.

Limbajul generat de o gramatică independentă de context se numește *limbaj independent de context*.

1.5.1. Proprietăți de închidere ale limbajelor independente de context

Teoremă. Dacă L_1 și L_2 sunt limbaje independente de context atunci $L_1 \cup L_2$, $L_1 L_2$, L_1^* sunt limbaje independente de context.

Observații. Fie $G_i = (N_i, \Sigma_i, P_i, S_i)$, $i=1,2$, două gramatici independente de context și $L_i = L(G_i)$, $i=1,2$.

1°. Dacă $L = L_1 \cup L_2$, $L = L(G)$, $G = (N, \Sigma, P, S)$ atunci:

$$N = N_1 \cup N_2 \cup \{S\}, \Sigma = \Sigma_1 \cup \Sigma_2, P = P_1 \cup P_2 \cup \{S \rightarrow S_1 | S_2\}, S \notin N_1 \cup N_2.$$

2°. Dacă $L = L_1 L_2$, $L = L(G)$, $G = (N, \Sigma, P, S)$, atunci:

$$N = N_1 \cup N_2 \cup \{S\}, \Sigma = \Sigma_1 \cup \Sigma_2, P = P_1 \cup P_2 \cup \{S \rightarrow S_1 S_2\}, S \notin N_1 \cup N_2.$$

3°. Dacă $L = L_1^*$, $L = L(G)$, $G = (N, \Sigma, P, S)$, atunci:

$$G = (N_1 \cup \{S\}, \Sigma_1, P_1 \cup \{S \rightarrow S S_1 | \varepsilon\}, S).$$

4°. $L_1 \cap L_2$ nu e independent de context; dăm un contraexemplu

$$L_1 = \{a^m b^n c^n \mid m, n \geq 1\}$$

și $L_2 = \{a^m b^m c^n \mid m, n \geq 1\}$ sunt dependente de context, dar

$$L_1 \cap L_2 = \{a^m b^m c^m \mid m \geq 1\}$$

nu este independent de context.

5°. $\overline{L_1}$ nu e independent de context, pentru că dacă ar fi atunci conform relației

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}} \text{ ar rezulta că intersecția este independentă de context (ceea ce în general nu e adevărat).}$$

1.5.2. Arbori de derivare

Arborii de derivare sunt arborescențe ordonate cu vârfuri etichetate cu simboluri din mulțimea neterminalelor și mulțimea terminalelor; ei conțin o *radăcină*, **noduri interioare** și **noduri terminale** (care nu au succesori).

Fie $G = (N, \Sigma, P, S)$ o gramatică independentă de context. Numim *arbore de derivare* sau *arbore de analiză sintactică* (a.a.s) o arborescență cu următoarele proprietăți:

- 1°. Fiecare vârf are o *etichetă*, care este un simbol din $N \cup \Sigma \cup \{\epsilon\}$;
- 2°. Eticheta din rădăcină este S ;
- 3°. Dacă un nod este interior și are eticheta A atunci A trebuie să fie din N ;
- 4°. Dacă un nod are eticheta A iar nodurile succesoare acestuia, în ordine de la stânga la dreapta sunt etichetate cu X_1, X_2, \dots, X_n atunci $A \rightarrow X_1 X_2 \dots X_n$ trebuie să fie o producție din P .
- 5°. Dacă un nod are eticheta ϵ , atunci acesta nu are succesori.

Nodurile terminale formează *frontiera* arborelui și ea, în ordine de la stânga la dreapta, formează o secvență peste $\Sigma \cup \{\epsilon\}$.

Fie G o gramatică independentă de context. Dacă într-o derivare directă se înlocuiește cel mai din stânga neterminat atunci derivarea directă o numim *derivare de stânga* și o notăm \xRightarrow{s} . Dacă într-o derivare directă se înlocuiește cel mai din dreapta neterminat, atunci derivarea directă se numește *derivare de dreapta* și o notăm \xRightarrow{d} .

Fie gramatica independentă de context $G = (N, \Sigma, P, S)$ atunci pentru o secvență $w \in \Sigma^*$ succesiunea de derivări directe:

$$S \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \alpha_n = w$$

reprezintă o derivare pentru cuvântul w sau altfel spus o *analiză sintactică*. Dacă această succesiune de derivări directe se obține pornind de la S la w atunci avem o *analiză sintactică descendentă* și aceasta corespunde construirii a.a.s. de sus în jos. Dacă construirea a.a.s. se face de jos în sus, adică de la w la S , atunci avem o *analiză sintactică ascendentă*. Aceste derivări pot fi făcute numai cu derivări de stânga sau numai cu derivări de dreapta.

Teoremă. Fie $G = (N, \Sigma, P, S)$ o gramatică independentă de context. Un cuvânt w peste alfabetul Σ , deci din Σ^* , aparține limbajului generat de G , adică $w \in L(G)$, dacă și numai dacă w este frontul unui arbore de analiză sintactică.

O gramatică $G = (N, \Sigma, P, S)$ independentă de context este *ambiguă* dacă și numai dacă există cel puțin un cuvânt care admite doi a.a.s. distincți; în caz contrar gramatica este *neambiguă*.

Limbajul generat de o gramatică ambiguă (neambiguă) este un *limbaj ambiguu* (*neambiguu*).

Observații.

1°. Noțiunea de **ambiguitate** poate fi definită mai general, pentru orice tip de gramatică. Astfel o gramatică este **ambiguă** dacă există un cuvânt al limbajului generat de ea cu proprietatea că există cel puțin două derivări de stânga (sau de dreapta) distincte pentru el.

2°. Definiția anterioară este, de fapt, o particularizare relativă la gramaticile independente de context.

1.5.3. Simplificarea gramaticilor independente de context

Fie $G = (N, \Sigma, P, S)$ o gramatică independentă de context cu $L = L(G)$, $L \neq \emptyset$.

1. Simboluri inaccesibile

Un simbol $x \in N \cup \Sigma$ este **simbol inaccesibil** dacă nu există nici o derivare $S \xRightarrow{*} \alpha x \beta$ cu $\alpha, \beta \in (N \cup \Sigma)^*$, în caz contrar simbolul este **accesibil**.

Lema 1. Gramatica G este echivalentă cu o gramatică $G' = (N', \Sigma', P', S)$ fără simboluri inaccesibile.

Algoritm. Gramatica G' conține numai simboluri accesibile care se determină astfel:

1°. $V_0 := \{S\}$, $i := 1$;

2°. $V_i := V_{i-1} \cup \{x \mid (A \rightarrow \alpha x \beta) \in P, A \in V_{i-1}, \alpha, \beta \in (N \cup \Sigma)^*\}$;

3°. Dacă $V_i \neq V_{i-1}$ atunci $i := i+1$ salt la 2°
altfel stop.

$N' := N \cap V_i$, $\Sigma' := \Sigma \cap V_i$

Mulțimea producțiilor P' este: $\{A \rightarrow \alpha \mid (A \rightarrow \alpha) \in P, A \in N', \alpha \in (N' \cup \Sigma')^*\}$.

2. Simboluri neproductive

Un simbol A , $A \in N$ este **neproductiv** dacă nu există nici o derivare de forma $A \xRightarrow{*} x$, $x \in \Sigma^*$, în caz contrar A este **simbol productiv**.

Lema 2. Gramatica G este echivalentă cu o gramatică $G' = (N', \Sigma, P', S)$ care conține conține numai simboluri productive.

Algoritm. Simbolurile productive se determină astfel:

1°. $N_0 := \emptyset$, $i := 1$;

2°. $N_i := N_{i-1} \cup \{A \mid (A \rightarrow \alpha) \in P, \beta \in (N_{i-1} \cup \Sigma)^*\}$;

3°. Dacă $N_i \neq N_{i-1}$ atunci $i:=i+1$ salt la 2°
altfel $N':=N_i$ stop.

Mulțimea producățiilor P' este: $\{A \rightarrow \alpha \mid (A \rightarrow \alpha) \in P, A \in N', \alpha \in (N' \cup \Sigma)^*\}$.

Observație: trebuie ca $S \in N'$, pentru că dacă $S \notin N'$ atunci $L(G) = \emptyset$, adică gramatica inițială G nu generează nici o secvență.

3. Simboluri neutilizabile

Un simbol este **neutilizabil** dacă el este fie inaccesibil, fie neproductiv.

Lema 3. Gramatica G este echivalentă cu o gramatică $G' = (N', \Sigma', P', S)$ fără simboluri neutilizabile.

Observație. Pentru obținerea gramaticii G' se aplică:

- algoritmul corespunzător lemei 1 pentru G ;
- algoritmul corespunzător lemei 2 pentru gramatica obținută la a).

4. ϵ -producții

O producție de forma $A \rightarrow \epsilon$ se numește **ϵ -producție**.

Dacă $\epsilon \in L(G)$ atunci avem producția $S \rightarrow \epsilon$ și S nu apare în membrul drept al nici unei producții. Putem elimina pe S din celelalte producții considerând un nou simbol de start S' și producțiile $S' \rightarrow \epsilon \mid S$.

Lema 4. Gramatica G este echivalentă cu o gramatică $G' = (N', \Sigma, P', S')$ în care:

- dacă $\epsilon \notin L(G)$ atunci G' nu are ϵ -producții;
- dacă $\epsilon \in L(G)$ atunci avem o singură producție $S' \rightarrow \epsilon$ iar celelalte producții nu-l conțin în membrul drept pe S' .

Observație. Gramatica G' se numește ϵ -independentă.

Algoritm. Pentru obținerea gramaticii $G' = (N', \Sigma, P', S')$ fără ϵ -producții procedăm astfel:

1°. Construim mulțimea N_ϵ care are ca elemente acele neterminale care prin derivare conduc la ϵ adică $N_\epsilon = \{A \mid A \in N, A \Rightarrow^* \epsilon\}$.

Se poate folosi pentru determinarea mulțimii N_ϵ un algoritm similar cu cei utilizați pentru determinarea simbolurilor accesibile și a celor productive:

- $N_0 := \{A \mid (A \rightarrow \epsilon) \in P\}$, $i := 1$;
- $N_i := N_{i-1} \cup \{A \mid (A \rightarrow \alpha) \in P, \alpha \in N_{i-1}^*\}$;
- Dacă $N_i \neq N_{i-1}$ atunci $i:=i+1$, salt la b)
altfel $N_\epsilon := N_i$, stop.

- dacă $S \notin N_\varepsilon$ atunci: $N'=N$, $S'=S$, și P' se construiește conform 2°.
- dacă $S \in N_\varepsilon$ atunci:

$$N'=N \cup \{S'\}, P' = \{S' \rightarrow \varepsilon, S' \rightarrow S\} \cup \text{mulțimea construită conform 2°}.$$

2°. Fie în P producția $A \rightarrow \alpha_0 B_1 \alpha_1 B_2 \alpha_2 \dots B_k \alpha_k$, unde $k \geq 0$, $B_i \in N_\varepsilon$ iar în α_i nu există simboluri din N_ε , $1 \leq i \leq k$.

În P' includem toate producțiile $A \rightarrow \alpha_0 X_1 \alpha_1 X_2 \alpha_2 \dots X_k \alpha_k$, ce se obțin punând în toate modurile posibile:

$$X_i = \begin{cases} B_i & i = \overline{1, k}. \\ \varepsilon \end{cases}$$

și excluzând eventualele producții de forma $A \rightarrow \varepsilon$ care s-ar obține prin acest procedeu.

5. Redenumiri

O producție de forma $A \rightarrow B$ se numește *redenumire*.

Lema 5. Gramatica G este echivalentă cu o gramatică $G' = (N, \Sigma, P', S)$ fără redenumiri.

Algoritm

1° Pentru fiecare $A \in N$ se construiește mulțimea $N_A = \{B \mid A \xRightarrow{*} B\}$;

Construcția este iterativă:

- $N_0 := \{A\}$, $i:=1$;
- $N_i := N_{i-1} \cup \{C \mid (B \rightarrow C) \in P, B \in N_{i-1}\}$
- dacă $N_i \neq N_{i-1}$ atunci $i:=i+1$, salt la b)
altfel $N_A := N_i$.

2° Producțiile din P' se obțin după cum urmează:

dacă $(B \rightarrow \alpha) \in P$, atunci $\exists N_A$ astfel încât $B \in N_A$. În P' introducem toate producțiile $A \rightarrow \alpha$ cu proprietatea că $B \in N_A$. Producțiile de forma $A \rightarrow B$ din P nu se includ în P' .

O gramatică independentă de context fără simboluri neutilizabile, fără ε -producții și fără redenumiri se numește *gramatică proprie*.

Teoremă. Oricare ar fi o gramatică G independentă de context cu $L(G) \neq \emptyset$ și $\varepsilon \notin L(G)$,

există o gramatică proprie G' , echivalentă cu G , adică $L(G) = L(G')$.

6. Recursivitate

7.

O producție de forma $A \rightarrow A\alpha$ se numește *recursivă la stânga*.

Lema 6. Fie G o gramatică proprie cu producțiile P . Această gramatică G este echivalentă cu o gramatică $G' = (N', \Sigma, P', S)$ ale cărei producții P' nu conțin recursivități la stânga.

Pentru producțiile din P cu membrul stâng A , considerăm următoarea partiție:

$$M_1 = \{A \rightarrow A\alpha_1, A \rightarrow A\alpha_2, \dots, A \rightarrow A\alpha_r\}$$

$$M_2 = \{A \rightarrow \beta_1, A \rightarrow \beta_2, \dots, A \rightarrow \beta_s\}.$$

Atunci P' va fi formată din producțiile:

$$A \rightarrow \beta_i, A \rightarrow \beta_i Z; 1 \leq i \leq s \quad \text{apoi} \quad Z \rightarrow \alpha_i, Z \rightarrow \alpha_i Z; 1 \leq i \leq r.$$

Procedăm analog cu celelalte producții de acest fel ale mulțimii P . Atunci $N' = N \cup$ mulțimea neterminalelor Z astfel introduse.

Observație: Recursivitatea nu se poate elimina, ea se transformă din recursivitate la stânga în recursivitate la dreapta.

1.5.4. Forma normală Chomsky

O gramatică independentă de context $G = (N, \Sigma, P, S)$ se spune că este în *forma normală Chomsky (FNC)* dacă orice producție din P este de una din formele:

- a) $A \rightarrow BC, A, B, C \in N$;
- b) $A \rightarrow a, a \in \Sigma, A \in N$;
- c) dacă $\varepsilon \in L(G)$ atunci $(S \rightarrow \varepsilon) \in P$, iar S nu apare în membrul drept al nici unei producții.

Teoremă. Oricare ar fi $G = (N, \Sigma, P, S)$ o gramatică independentă de context, întotdeauna există o gramatică în forma normală Chomsky G' , astfel încât $L(G) = L(G')$.

1.5.5. Forma normală Greibach

O gramatică $G = (N, \Sigma, P, S)$ este în *forma normală Greibach (FNG)* dacă P are producții numai de forma:

- a) $A \rightarrow a\alpha, A \in N, a \in \Sigma, \alpha \in N^*$;
- b) dacă $\varepsilon \in L(G)$ atunci $(S \rightarrow \varepsilon) \in P$, iar S nu apare în membrul drept al nici unei producții.

Teoremă. Oricare ar fi $G = (N, \Sigma, P, S)$ o gramatică independentă de context,

întotdeauna există o gramatică în forma normală Greibach, astfel încât $L(G)=L(G')$.

1.5.6. Leme de pompare pentru limbaje independente de context

Propoziție. (Lema de pompare) Fie L un limbaj independent de context. Există atunci atunci o constantă n dependentă numai de L astfel că dacă $z \in L$ și $|z| \geq n$, atunci avem descompunerea $z=uvwxy$ cu proprietățile:

- a) $|vx| \geq 1$,
- b) $|vwx| \leq n$,
- c) $uv^iwx^iy \in L \ \forall i \geq 0$.

Propoziție. (Lema **Bar-Hillel**) Dacă L este un limbaj independent de context, atunci există $p > 0$, $q > 0$ astfel încât pentru orice $z \in L$, $|z| > p$ să avem o descompunere $z=uvwxy$ cu $|vwx| \leq q$ și $|v| + |x| \neq 0$ (v și x nu sunt simultan vide), iar $uv^iwx^iy \in L$, $\forall i \geq 0$.

1.5.7. Probleme propuse

1.5.1.

Să se construiască gramatica care generează expresiile regulate peste alfabetul $\{a,b\}$.

1.5.2.

Fie limbajele $L_1 = \{01^n0 \mid n \geq 1\}$ și $L_2 = \{10^n1 \mid n \geq 1\}$. Să se construiască gramaticile care generează limbajele:

$L_1 \cup L_2$, L_1L_2 , L_1^* , L_2^* .

1.5.3.

Să se definească o gramatică independentă de context care generează peste alfabetul $\{b,c\}$ toate cuvintele ce au același număr de b -uri ca și c -uri.

1.5.4.

Să se definească o gramatică independentă de context care generează peste alfabetul $\{b,c,d\}$ toate cuvintele în care bc nu este o subsecvență.

1.5.5.

Să se definească o gramatică independentă de context care generează mulțimea palindroamelor peste alfabetul $\{b,c\}$. (Un palindrom este un cuvânt identic cu oglinditul său).

1.5.6.

Să se construiască gramaticile independente de context care generează limbajele:

- a) $\{b^m c^{2m} d e^{n^m} \mid m, n \geq 1\}$;
- b) $\{b^m c^n d^n e^{m^p} (gh^k)^p \mid m \geq 2, k, n \geq 0, p \geq 1\}$;
- c) $\{b^m c^n \mid 1 \leq n \leq m \leq 2n\}$;
- d) $\{b^{m+p} c d^{m+n} e f^{m+p} \mid m, n, p \geq 0\}$.

1.5.7.

Să se construiască gramaticile independente de context care generează limbajele:

- a) $L_1 = \{a^n b^m \mid n \geq m \geq 0\}$; c) $L_3 = \{a^n b^k c^m \mid m \geq n \geq 0, k \geq 0\}$;
- b) $L_2 = \{a^n b^m \mid m \geq n \geq 0\}$; d) $L_4 = \{a^n b^k c^m \mid n \geq m \geq 0, k \geq 0\}$.

1.5.8.

Să se construiască gramaticile independente de context care generează limbajele:

- a) $L_1 = \{a^n b^m \mid 1 \leq n \leq m \leq 2n\}$; c) $L_3 = \{a^n b^m \mid 2n \leq m \leq 3n, n \geq 0\}$;
- b) $L_2 = \{a^n b^m \mid 1 \leq n \leq m \leq 5n\}$; d) $L_4 = \{a^n b^m \mid 3m \leq n \leq 5m, m \geq 0\}$.

1.5.9.

Să se construiască gramaticile independente de context care generează limbajele:

- a) $L_1 = \{a^n b^m c^m d^n \mid n, m \geq 0\}$; b) $L_2 = \{a^n b^k c^{n-k} \mid n \geq k \geq 0\}$;
- c) $L_3 = \{a^n b^k c^{n+k} \mid n \geq 0, k \geq 0\}$.

1.5.10.

Să se construiască gramaticile independente de context care generează limbajele:

- a) $L_1 = \{x \tilde{x} \mid x \in \{0,1\}^*\}$ \tilde{x} este oglinditul (reversul) lui x ;
- b) $L_2 = \{x w x \mid x, w \in \{0,1\}^*\}$;
- c) $L_3 = \{x \tilde{x} w \mid x, w \in \{0,1\}^*\}$;
- d) $L_4 = \{w x x \mid x, w \in \{0,1\}^*\}$.

1.5.11.

Să se construiască gramaticile independente de context care generează limbajele:

- a) $L_1 = \{w \in \{a,b\}^* \mid nr_a(w) = nr_b(w)\}$; b) $L_2 = \{w \in \{a,b\}^* \mid nr_a(w) \geq nr_b(w)\}$.

1.5.12.

Să se construiască gramaticile care generează limbajele:

- a) $L_1 = \{w \in \{a,b\}^* \mid nr_a(w) = 2 * nr_b(w), \text{ iar simbolurile "a" apar perechi}\}$
ex: $aabbbbaaaa \in L_1$;
 $abbaabaaa \notin L_1$;
- b) $L_2 = \{w \in \{a,b\}^* \mid nr_a(w) = k * nr_b(w), k \in \mathbb{N}, \text{ iar simbolurile "a" apar în grupuri de multiplu de k elemente}\}$;
- c) $L_3 = \{w \in \{a,b\}^* \mid 2 * nr_a(w) = 3 * nr_b(w), \text{ simbolurile "a" apar în grupuri de multiplu de 3 elemente, iar simbolurile "b" apar în grupuri de multiplu de 2 elemente}\}$;
ex: $aaabbbbbaaa \in L_3$.

1.5.13.

Să se construiască gramaticile care generează expresii aritmetice având doar operanzi notați cu "a" și operatorii binari +, -, /, *

- a) în forma poloneză prefixată;
- b) în forma poloneză postfixată;
- c) în forma poloneză infixată (cu paranteze).

1.5.14.

Să se arate că gramaticile care conțin producțiile următoare sunt ambigue și să se găsească câte o gramatică echivalentă neambiguă:

- a) $S \rightarrow aS \mid Sb \mid c$;
- b) $S \rightarrow \text{if } b \text{ then } S \text{ else } S \mid \text{if } b \text{ then } S \mid a$;
- c) $S \rightarrow a \mid aB, B \rightarrow a \mid aB \mid SB$;
- d) $S \rightarrow S \mid S \mid (S) \mid 1$.

1.5.15.

Să se arate că gramaticile care conțin producțiile următoare sunt ambigue:

- a) $A \rightarrow A\alpha A \mid a$;
- b) $A \rightarrow \alpha A \mid A\beta \mid a$;
- c) $A \rightarrow \alpha A \mid \alpha A\beta \mid a$.

1.5.16.

Să se arate că gramatica $G = (\{S, B, C\}, \{a, b, c\}, P, S)$, cu $P = \{S \rightarrow abC \mid aB, B \rightarrow bc, bC \rightarrow bc\}$ este ambiguă.

1.5.17.

Fie gramatica $G = (\{S, A, B\}, \{a, b\}, P, S)$ și având producțiile următoare:

$$S \rightarrow aB \mid bA$$

$$A \rightarrow a \mid aS \mid bAA$$

$$B \rightarrow b \mid bS \mid aBB$$

și $w = aaabbabbba$

- a) găsiți o derivare de stânga pentru w;
- b) găsiți o derivare de dreapta pentru w;
- c) construiți arborele de derivare cu frontul w;
- d) este această gramatică neambiguă?
- e) se poate descompune w conform lemei de pompare?

1.5.18.

Descrieți limbajul generat de gramatica $G = (\{S\}, \{a, b\}, P, S)$, $P = \{S \rightarrow bSS \mid a\}$.

1.5.19.

Să se descrie limbajul generat de gramatica $G = (\{S, A, B\}, \{a, b, c, d\}, P, S)$ cu producțiile:

$$S \rightarrow Ac \mid Bd$$

$$A \rightarrow aAb \mid ab$$

$$B \rightarrow aBbb \mid abb$$

1.5.20.

Arătați că $L(G)$ este mulțimea secvențelor peste alfabetul $\{0,1\}$ care conțin un număr multiplu de 3 de simboluri 0, unde:

$G = (\{S,A,B\}, \{0,1\}, P, S)$, $P = \{S \rightarrow 0A \mid 1S \mid \varepsilon, A \rightarrow 0B \mid 1A, B \rightarrow 0S \mid 1B\}, S$.

1.5.21.

Fie $G = (\{S,A,B,C\}, \{a,b\}, P, S)$ cu mulțimea producțiilor:

$P = \{S \rightarrow AB \mid C, A \rightarrow a, B \rightarrow CB \mid A, C \rightarrow b\}$.

Să se arate că $L(G) = \{b\} \cup \{ab^n a \mid n \geq 0\}$.

1.5.22.

Să se aducă la forma normală Chomsky și la forma normală Greibach gramaticile de la problemele **1.5.16.**, **1.5.17.**, **1.5.19.**, **1.5.20.**

1.5.8. Indicații și soluții**1.5.1.**

$G = (N, \Sigma, P, S)$;

$N = \{S\}, \Sigma = \{a,b,+,*,(),\}, P = \{S \rightarrow S+S \mid SS \mid (S) \mid S^* \mid a \mid b \mid \varepsilon \mid \emptyset\}$.

1.5.3.

O astfel de gramatică este:

$G = (N, \Sigma, P, S)$ cu

$N = \{S\}, \Sigma = \{b,c\}, P = \{S \rightarrow bSc \mid cSb \mid SS \mid bc \mid cb\}$.

1.5.4.

$G = (N, \Sigma, P, S)$ cu

$N = \{S,E\}, \Sigma = \{b,c,d\}, P = \{S \rightarrow bE \mid cS \mid dS \mid b \mid c \mid d, E \rightarrow bE \mid dS \mid b \mid d\}$.

1.5.5.

$G = (N, \Sigma, P, S)$ cu

$N = \{S\}, \Sigma = \{b,c\}, P = \{S \rightarrow bSb \mid cSc \mid b \mid c \mid \varepsilon\}$.

1.5.6.

a) $G = (N, \Sigma, P, S)$

$N = \{S,J,K\}, \Sigma = \{b,c,d,e,f\}, P = \{S \rightarrow JdK, J \rightarrow bJcc \mid bcc, K \rightarrow eKf \mid ef\}$.

b) $G = (N, \Sigma, P, S)$

$N = \{S,J,L,K,Q\}, \Sigma = \{b,c,d,e,f,h,g\},$

$P = \{S \rightarrow JK, J \rightarrow bJe \mid bbee \mid bbLee, L \rightarrow cLd \mid cd, K \rightarrow fKQ \mid fQ, Q \rightarrow ghQ \mid gh\}$.

c) $G = (N, \Sigma, P, S)$

$N = \{S\}, \Sigma = \{b,c\}, P = \{S \rightarrow bSc \mid bbSc \mid bc \mid bbc\}$.

d) $G = (N, \Sigma, P, S)$

$N = \{S, G, H\}, \Sigma = \{b, c, d, e, f\}, P = \{S \rightarrow bSf \mid GH, G \rightarrow bGd \mid c, H \rightarrow dHf \mid e\}.$

Un cuvânt $w \in L(G)$ se poate scrie $w = b^{m+p}cd^{m+n}ef^{m+p} = b^p((b^mcd^m)(d^nef^m))f^p.$

1.5.7.

a) $G = (N, \Sigma, P, S)$ cu

$N = \{S, A\}, \Sigma = \{a, b\}, P = \{S \rightarrow aSb \mid A, A \rightarrow aA \mid \epsilon\}.$

b) $G = (N, \Sigma, P, S)$

$N = \{S, B\}, \Sigma = \{a, b\}, P = \{S \rightarrow aSb \mid B, B \rightarrow Bb \mid \epsilon\}.$

c) $G = (N, \Sigma, P, S)$

$N = \{S, B, C\}, \Sigma = \{a, b, c\}, P = \{S \rightarrow aSc \mid B, B \rightarrow bB \mid C, C \rightarrow Cc \mid \epsilon\}.$

d) $G = (N, \Sigma, P, S)$

$N = \{S, A, B\}, \Sigma = \{a, b, c\}, P = \{S \rightarrow aSc \mid B, B \rightarrow Bb \mid A, A \rightarrow aA \mid \epsilon\}.$

1.5.8.

a) $G = (N, \Sigma, P, S)$

$N = \{S\}, \Sigma = \{a, b\}, P = \{S \rightarrow aSb \mid aSbb \mid ab \mid abb\}.$

b) $G = (N, \Sigma, P, S)$

$N = \{S\}, \Sigma = \{a, b\}, P = \{S \rightarrow aSb \mid aSb^2 \mid aSb^3 \mid aSb^4 \mid aSb^5 \mid ab \mid ab^2 \mid ab^3 \mid ab^4 \mid ab^5\}.$

c) $G = (N, \Sigma, P, S)$

$N = \{S\}, \Sigma = \{a, b\}, P = \{S \rightarrow aSbb \mid aSbbb \mid \epsilon\}.$

d) $G = (N, \Sigma, P, S)$

$N = \{S\}, \Sigma = \{a, b\}, P = \{S \rightarrow a^3Sb \mid a^4Sb \mid a^5Sb \mid \epsilon\}.$

1.5.9.

a) $G = (N, \Sigma, P, S)$

$N = \{S, H\}, \Sigma = \{a, b, c, d\}, P = \{S \rightarrow aSd \mid H, H \rightarrow bHc \mid \epsilon\}.$

b) se observă că $a^n b^k c^{n-k}$ se poate scrie sub forma $a^{n-k} (a^k b^k) c^{n-k}$ și se reduce la problema de la a)

$G = (N, \Sigma, P, S)$

$N = \{S, H\}, \Sigma = \{a, b, c\}, P = \{S \rightarrow aSc \mid H, H \rightarrow aHb \mid \epsilon\}.$

c) se observă că $a^n b^k c^{n+k}$ se poate scrie sub forma $a^n (b^k c^k) c^n$ și se reduce la problema de la punctul a)

$G = (N, \Sigma, P, S)$

$N = \{S, H\}, \Sigma = \{a, b, c\}, P = \{S \rightarrow aSc \mid H, H \rightarrow bHc \mid \epsilon\}.$

1.5.10.a) $G = (N, \Sigma, P, S)$

$$N = \{S\}, \Sigma = \{0,1\}, P = \{S \rightarrow 0S0 \mid 1S1 \mid \epsilon\}.$$

b) $G = (N, \Sigma, P, S)$

$$N = \{S, X\}, \Sigma = \{0,1\}, P = \{S \rightarrow 0S0 \mid 1S1 \mid X, X \rightarrow 0X \mid 1X \mid \epsilon\}.$$

c) $G = (N, \Sigma, P, S)$

$$N = \{S, S_1, S_2\}, \Sigma = \{0,1\}, P = \{S \rightarrow S_1 S_2, S_1 \rightarrow 0S_1 0 \mid 1S_1 1 \mid \epsilon, S_2 \rightarrow 0S_2 \mid 1S_2 \mid \epsilon\}.$$

d) $G = (N, \Sigma, P, S)$

$$N = \{S, S_1, S_2\}, \Sigma = \{0,1\}, P = \{S \rightarrow S_2 S_1, S_1 \rightarrow 0S_1 0 \mid 1S_1 1 \mid \epsilon, S_2 \rightarrow 0S_2 \mid 1S_2 \mid \epsilon\}.$$

1.5.11.

a)-gramatică ambiguă

$$G = (N, \Sigma, P, S)$$

$$N = \{S\}, \Sigma = \{a,b\}, P = \{S \rightarrow aSb \mid bSa \mid SS \mid \epsilon\}.$$

-gramatică neambiguă

$$G = (N, \Sigma, P, S)$$

$$N = \{S, A, B\}, \Sigma = \{a,b\}, P = \{S \rightarrow aB \mid bA, A \rightarrow aS \mid bAA \mid a, B \rightarrow bS \mid aBB \mid b\}.$$

b)-gramatică ambiguă

$$G = (N, \Sigma, P, S)$$

$$N = \{S, A\}, \Sigma = \{a,b\}, P = \{S \rightarrow aSb \mid bSa \mid SS \mid A, A \rightarrow aA \mid a\}.$$

-gramatică neambiguă

$$G = (N, \Sigma, P, S)$$

$$N = \{S, A, B, X\}, \Sigma = \{a,b\},$$

$$P = \{S \rightarrow AB \mid bA \mid \epsilon, A \rightarrow AS \mid bAA \mid X, B \rightarrow bS \mid ABB \mid b, X \rightarrow aX \mid a\}.$$

1.5.12.a) $G = (N, \Sigma, P, S)$

$$N = \{S, A, B\}, \Sigma = \{a,b\},$$

$$P = \{S \rightarrow AB \mid BA \mid \epsilon, A \rightarrow AS \mid BAA \mid aa, B \rightarrow BS \mid ABB \mid b\}.$$

b) $G = (N, \Sigma, P, S)$

$$N = \{S, A, B\}, \Sigma = \{a,b\},$$

$$P = \{S \rightarrow AB \mid BA \mid \epsilon, A \rightarrow AS \mid BAA \mid a^k, B \rightarrow BS \mid ABB \mid b\}.$$

c) $G = (N, \Sigma, P, S)$

$$N = \{S, A, B\}, \Sigma = \{a,b\},$$

$$P = \{S \rightarrow AB \mid BA \mid \epsilon, A \rightarrow AS \mid BAA \mid a^3, B \rightarrow BS \mid ABB \mid b^2\}.$$

1.5.13.

Gramaticile de la a) și b) sunt neambigue, ele rezultă din definițiile recursive ale celor două forme poloneze prefixată și postfixată.

a) $G = (N, \Sigma, P, S)$

$N = \{S\}, \Sigma = \{a, +, -, /, *\}, P = \{S \rightarrow +SS \mid -SS \mid /SS \mid *SS \mid a\}.$

Exemplu:

$w = +*+aaa-a/aa$

$S \Rightarrow +SS \Rightarrow +*SSS \Rightarrow +*+SSSS \Rightarrow +*+aSSS \Rightarrow +*+aaSS \Rightarrow +*+aaaS \Rightarrow$
 $\Rightarrow +*+aaa-SS \Rightarrow +*+aaa-aS \Rightarrow +*+aaa-a/SS \Rightarrow +*+aaa-a/aS \Rightarrow +*+aaa-a/aa$

Pentru forma poloneză prefixată am folosit derivări de stânga pentru că ele sunt mai sugestive, fiind dirijate de forma cuvântului pornind de la stânga spre dreapta. Există o singură derivare de stânga a fiecărui cuvânt din limbaj, gramatica este neambiguă.

b) $G = (N, \Sigma, P, S)$

$N = \{S\}, \Sigma = \{a, +, -, /, *\}, P = \{S \rightarrow SS+ \mid SS- \mid SS/ \mid SS* \mid a\}.$

Exemplu:

$w = aa+a*aaa/-+$

$S \Rightarrow SS+ \Rightarrow SSS-+ \Rightarrow SSSS/-+ \Rightarrow SSSa/-+ \Rightarrow SSaa/-+ \Rightarrow Saaa/-+ \Rightarrow SS*aaa/-+ \Rightarrow$
 $\Rightarrow Sa*aaa/-+ \Rightarrow SS+a*aaa/-+ \Rightarrow Sa+a*aaa/-+ \Rightarrow aa+a*aaa/-+$

Pentru forma poloneză postfixată am folosit derivări de dreapta pentru că ele sunt mai sugestive, fiind dirijate de forma cuvântului pornind de la dreapta spre stânga. Există o singură derivare de dreapta a fiecărui cuvânt din limbaj, gramatica este neambiguă.

c) $G = (N, \Sigma, P, S)$

$N = \{S\}, \Sigma = \{a, +, -, /, *, (,)\}, P = \{S \rightarrow S+S \mid S-S \mid S/S \mid S*S \mid (S) \mid a\}.$

Această gramatică este ambiguă deoarece pentru unele cuvinte din limbaj există două derivări de stânga distincte.

Exemplu: pentru cuvântul $w = a+a*a$ avem derivările următoare:

$S \Rightarrow S+S \Rightarrow a+S \Rightarrow a+S*S \Rightarrow a+a*S \Rightarrow a+a*a$ și
 $S \Rightarrow S*S \Rightarrow S+S*S \Rightarrow a+S*S \Rightarrow a+a*S \Rightarrow a+a*a.$

Se poate construi și o gramatică neambiguă echivalentă cu cea de mai sus:

$G' = (N', \Sigma, P', E)$

$N' = \{E, T, F\}, \Sigma = \{a, +, -, /, *, (,)\},$

$P' = \{E \rightarrow E+T \mid E-T \mid T, T \rightarrow T*F \mid T/F \mid F, F \rightarrow E\} \mid a\}$

E = are semnificația de expresie

F = are semnificația de factor

T = are semnificația de termen

Exemplu: cuvântul $w = a+a*a$ se poate obține în mod unic prin derivare de stânga:

$E \Rightarrow E+T \Rightarrow T+T \Rightarrow F+T \Rightarrow a+T \Rightarrow a+T*F \Rightarrow a+F*F \Rightarrow a+a*F \Rightarrow a+a*a.$

1.5.14.

a) $w = aacb$ și cele două derivări de stânga pentru w sunt:

$S \Rightarrow aS \Rightarrow aaS \Rightarrow aaSb \Rightarrow aacb$ și

$S \Rightarrow Sb \Rightarrow aSb \Rightarrow aaSb \Rightarrow aacb$

Producțiile de mai sus se pot înlocui cu producțiile următoare:

$S \rightarrow aS \mid cB, B \rightarrow bB \mid \varepsilon$.

Această gramatică este neambiguă, generarea cuvintelor limbajului se face în mod unic de la stânga spre dreapta.

$w = aacb$

$S \Rightarrow aS \Rightarrow aaS \Rightarrow aacB \Rightarrow aacbB \Rightarrow aacb$

De fapt cele două gramatici generează limbajul $L = \{a^m cb^n \mid m, n \geq 0\}$.

b) Pentru $w = \text{if } b \text{ then if } b \text{ then } a \text{ else } a$ avem două derivări de stânga distincte:

$S \Rightarrow \text{if } b \text{ then } S \text{ else } S \Rightarrow \text{if } b \text{ then if } b \text{ then } S \text{ else } S \Rightarrow$
 $\text{if } b \text{ then if } b \text{ then } a \text{ else } S \Rightarrow \text{if } b \text{ then if } b \text{ then } a \text{ else } a$

și

$S \Rightarrow \text{if } b \text{ then } S \Rightarrow \text{if } b \text{ then if } b \text{ then } S \text{ else } S \Rightarrow$
 $\text{if } b \text{ then if } b \text{ then } a \text{ else } S \Rightarrow \text{if } b \text{ then if } b \text{ then } a \text{ else } a$.

Ambiguitatea rezultă din faptul că **else**-ul se poate atașa oricăruia din cele două **then**-uri. Pentru a înlătura ambiguitatea trebuie să facem o convenție de atașare a unui **else** la un **then**. Convenția va fi următoarea: **else**-ul se va atașa ultimului **then**.

Gramatica neambiguă care modelează **if...then..else...** va fi:

$G = (N, \Sigma, P, S)$

$N = \{S, S'\}, \Sigma = \{a, b, \text{if}, \text{then}, \text{else}\}$

$P = \{S \rightarrow \text{if } b \text{ then } S \mid \text{if } b \text{ then } S' \text{ else } S \mid a, S' \rightarrow \text{if } b \text{ then } S' \text{ else } S' \mid a\}$.

Faptul că doar S' precede un **else**, asigură că între o pereche **then-else** generată de orice producție trebuie să apară fie **a** fie un alt **else**.

c) Pentru $w = aaaa$ avem următoarele derivări:

$S \Rightarrow aB \Rightarrow aaB \Rightarrow aaaB \Rightarrow aaaa = w$

și

$S \Rightarrow aB \Rightarrow aSB \Rightarrow aaBB \Rightarrow aaaB \Rightarrow aaaa = w$.

Limbajul generat de această gramatică este $L = \{a^n \mid n \geq 1\}$, iar o gramatică neambiguă care îl generează este:

$G = (N, \Sigma, P, S)$

$N = \{S\}, \Sigma = \{a\}, P = \{S \rightarrow aS \mid a\}$.

d) Două derivări de stânga distincte pentru $w = ((1))$ sunt:

$S \Rightarrow (S \Rightarrow ((S \Rightarrow ((S) \Rightarrow ((1)) = w$

și

$S \Rightarrow (S \Rightarrow ((S) \Rightarrow ((1)) = w$

Gramatica generează limbajul $L = \{(1)^n \mid n \geq 0\}$;

O gramatică neambiguă care generează acest limbaj este:

$G = (N, \Sigma, P, S)$

$N = \{S, A\}, \Sigma = \{(1)\}$

$P = \{S \rightarrow (S \mid 1A, A \rightarrow A) \mid \varepsilon\}$

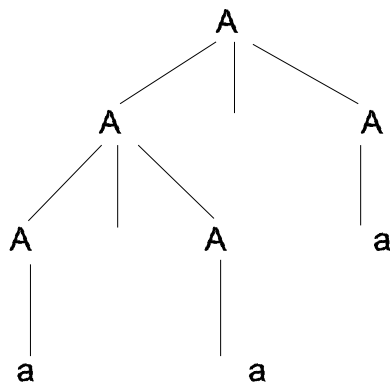
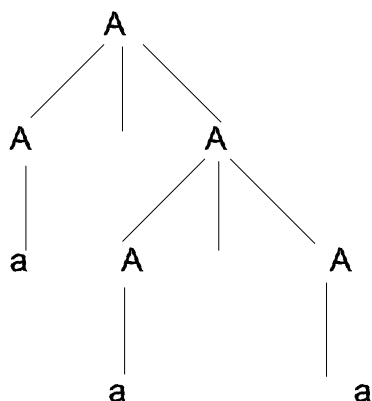
În acest caz cuvintele se generează în mod unic de la stânga la dreapta: întâi se generează parantezele deschise, apoi simbolul 1, iar la sfârșit parantezele închise.

$$S \Rightarrow (S \Rightarrow ((S \Rightarrow ((1A \Rightarrow ((1A) \Rightarrow ((1) = w.$$

1.5.15.

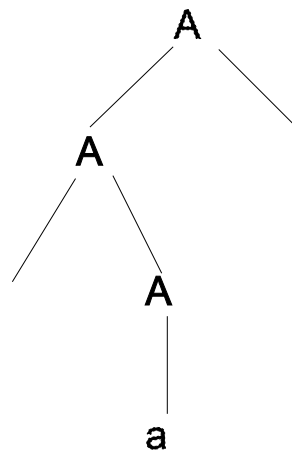
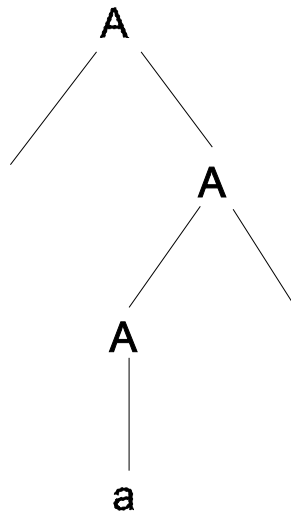
Gramaticile din enunț sunt independente de context.

a) Pentru $w = aaaaa \in L(G)$ cei doi arbori de derivare distincți cu frontul w sunt:

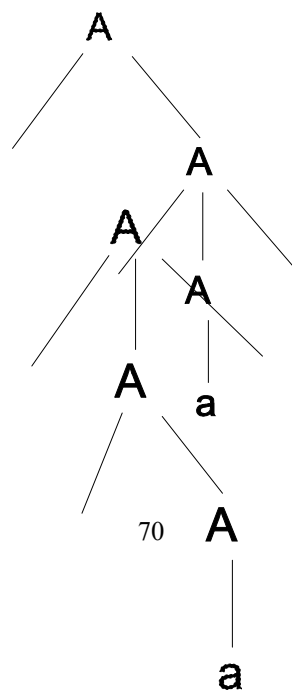


b) Fie cuvântul $w = \alpha a \beta \in L(G)$.

Cei doi arbori de derivare cu frontul w sunt:



c) Cei doi arbori de derivare cu frontul w pentru $w = \alpha\alpha a\beta$ sunt:



1.5.16.

Fie $w=abc \in L(G)$ și avem două derivări de stânga distincte pentru w :

$$a) S \xRightarrow{1} abC \xRightarrow{4} abc \text{ și}$$

$$b) S \xRightarrow{2} aB \xRightarrow{3} abc.$$

1.5.19.

$$L(G) = \{a^n b^n c \mid n \geq 1\} \cup \{a^n b^{2n} d \mid n \geq 1\}.$$

1.6. AUTOMATE PUSH-DOWN

Un *automat push-down* (APD) este un ansamblu $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ unde:

- Q este o mulțime finită și nevidă de elemente numite *stări*;
- Σ este un alfabet denumit *alfabetul de intrare*;
- Γ este un alfabet denumit *alfabetul memoriei stivă*;
- $q_0 \in Q$, q_0 este *stare inițială*;
- $Z_0 \in \Gamma$, Z_0 este *simbolul de start* al memoriei stivă;
- $F \subseteq Q$, F este *mulțimea stărilor finale*;
- $\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow P_0(Q \times \Gamma^*)$ este *funcția de tranziție* care are ca valori submulțimi finite din $Q \times \Gamma^*$ (în general parțial definită).

Fie APD $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$. Spunem că acest automat este *determinist* dacă:

- 1) $\forall q \in Q$ și $Z \in \Gamma$, în cazul că $|\delta(q, \epsilon, Z)| = 1$, atunci $\delta(q, a, Z) = \emptyset$, $\forall a \in \Sigma$;
- 2) $|\delta(q, a, Z)| \leq 1$, $\forall q \in Q$, $\forall a \in \Sigma \cup \{\epsilon\}$, $\forall Z \in \Gamma$.

În caz contrar APD este *nedeterminist*.

Observație.

Dacă $q \in Q$, $a \in \Sigma \cup \{\epsilon\}$, $Z \in \Gamma$ atunci $\delta(q, a, Z) = \{(p_1, \gamma_1), \dots, (p_n, \gamma_n)\}$ $p_k \in Q$, $\gamma_k \in \Gamma^*$, $k \in \{1, \dots, n\}$.

O *configurație* a automatului M este $(q, w, \alpha) \in Q \times \Sigma^* \times \Gamma^*$ care înseamnă că automatul se găsește în starea q , pe banda de intrare urmează să se citească (accepte) secvența w , iar în memoria stivă avem secvența α .

Pe mulțimea configurațiilor se definesc relațiile binare:

a) \mapsto *tranziție directă*

$$(q, ax, Z\alpha) \mapsto (p, x, \gamma\alpha) \iff \delta(q, a, Z) \ni (p, \gamma)$$

sau

$$(q, ax, Z\alpha) \mapsto (p, ax, \gamma\alpha) \iff \delta(q, \epsilon, Z) \ni (p, \gamma)$$

În ultimul caz spunem că a avut loc o ϵ -tranziție, adică nu s-a avansat pe banda de intrare (unde $p, q \in Q$, $a \in \Sigma$, $Z \in \Gamma$, $x \in \Sigma^*$, α, γ din Γ^*).

Tranziția directă înseamnă cu alte cuvinte următoarele:

- se trece (eventual) în altă stare;
- se avansează (sau nu) pe banda de intrare;
- se înlocuiește simbolul Z , din vârful stivei, cu o secvență γ de simboluri din Γ ;

b) \mapsto^k *k tranziția* (k tranziții directe);

c) \mapsto^+ *+ tranziția* (închiderea tranzitivă);

d) \mapsto^* ** tranziția* (închiderea reflexivă și tranzitivă)

1.6.1. Reprezentare

Fie $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ și $Q = \{q_0, q_1, \dots, q_n\}$, $\Gamma = \{Z_0, Z_1, \dots, Z_k\}$
Pentru reprezentare automatului se poate alcătui următorul *tabel*:

QxΓ		$\Sigma \cup \{\epsilon\}$			
		a	B	...	ε
q ₀	Z ₀	$\delta(q_0, a, Z_0)$	$\delta(q_0, b, Z_0)$		$\delta(q_0, \epsilon, Z_0)$

	Z _m	$\delta(q_0, a, Z_m)$ $\delta(q_0, b, Z_m)$		$\delta(q_0, \epsilon, Z_m)$	
...
q _n	Z ₀	$\delta(q_n, a, Z_0)$	$\delta(q_n, b, Z_0)$		$\delta(q_n, \epsilon, Z_0)$

	Z _m	$\delta(q_n, a, Z_m)$	$\delta(q_n, b, Z_m)$		$\delta(q_n, \epsilon, Z_m)$

1.6.2. Limbaj acceptat de un APD

Fie APD $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$. Configurația (q_0, w, Z_0) o numim *configurație inițială*. Configurația (q, ϵ, γ) , $q \in F$ o numim configurație finală după *criteriul stării finale*, iar configurația (q, ϵ, ϵ) se numește configurația finală după *criteriul stivei vide*.

Limbajul acceptat de APD, M , după *criteriul stării finale* este:

$$T_1(M) = \{w \mid w \in \Sigma^*, (q_0, w, Z_0) \xrightarrow{*} (q, \epsilon, \gamma), q \in F, \gamma \in \Gamma^*\}$$

iar *limbajul acceptat* de APD, M , după *criteriul stivei vide* este

$$T_2(M) = \{w \mid w \in \Sigma^*, (q_0, w, Z_0) \xrightarrow{*} (q, \epsilon, \epsilon), q \in Q\}$$

Observații.

1°. Pentru un limbaj acceptat de un APD după criteriul stivei vide nu are nici o importanță mulțimea stărilor finale F , se poate considera că în acest caz $F=\emptyset$; altfel în tabelul anterior se adaugă la Început o coloană cu elemente $z_i := \text{dacă } q_i \in F \text{ atunci } 1 \text{ altfel } 0, i=0,1,\dots,n$.

2°. Se folosește pentru $T_2(M)$ uneori notația $T_\varepsilon(M)$, deci $T_\varepsilon(M) = T_2(M)$.

3°. Cele două criterii de acceptare sunt echivalente.

Teoremă. Fie automatul push-down M . Există întotdeauna un automat push-down M' astfel încât $T_\varepsilon(M') = T_1(M)$; de asemenea și reciproc.

Teoremă. Oricare ar fi L un limbaj independent de context, există un automat push-down M astfel încât $T_\varepsilon(M) = L$.

Vom da un algoritm de trecere de la o gramatică independentă de context la automatul push-down care acceptă limbajul generat de gramatică.

Fie L un limbaj independent de context, atunci există o gramatică independentă de context $G = (N, \Sigma, P, S)$ astfel încât $L = L(G)$. Se construiește automatul push-down $M = (\{q\}, \Sigma, N \cup \Sigma, \delta, q, S, \emptyset)$ cu proprietatea $T(M) = L(G)$, astfel:

- a) dacă $(A \rightarrow \alpha) \in P$ atunci $(q, \alpha) \in \delta(q, \varepsilon, A)$;
(neterminalul din vârful stivei este expandat prin α);
- b) $\delta(q, a, a) := \{(q, \varepsilon)\} \quad \forall a \in \Sigma$;
(se șterge simbolul din vârful stivei dacă acesta coincide cu cel de pe banda de intrare)
- c) $\delta(.,.,.) = \emptyset$ în celelalte cazuri.

1.6.3. Probleme propuse

1.6.1.

Să se construiască un automat push-down care acceptă limbajul $L = \{a^n b^n \mid n \geq 0\}$.

1.6.2.

Să se construiască un automat push-down care acceptă limbajul $L = \{a^n b^{3n} \mid n \geq 0\}$.

1.6.3.

Să se construiască un automat push-down care acceptă limbajul $L = \{a^{3n} b^n \mid n \geq 0\}$.

1.6.4.

Să se construiască un automat push-down care acceptă limbajul $L = \{a^{2n} b^{3n} \mid n \geq 0\}$.

1.6.5.

Să se construiască un automat push-down care acceptă limbajul $L = \{a^n b^n \mid n \geq 1\} \cup$

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

1.6.6.

Să se construiască un automat push-down care acceptă limbajul $L = \{a^n b^m \mid 0 < n < m\}$.

1.6.7.

Să se construiască un automat push-down care acceptă limbajul $L = \{a^n b^m \mid n > m > 0\}$.

1.6.8.

Să se construiască un automat push-down care acceptă limbajul $L = \{a^n b^m \mid 1 \leq n \leq m \leq 2n\}$.

1.6.9.

Să se construiască un automat push-down care acceptă limbajul

$L = \{a^n b^m \mid 1 \leq pn \leq m \leq qn, p \leq q\}$.

1.6.10.

Să se construiască un automat push-down care acceptă limbajul $L = \{a^n b^m \mid 1 \leq m \leq n \leq 2m\}$.

1.6.11.

Să se construiască un automat push-down care acceptă limbajul

$L = \{w d \tilde{w} \mid w \in \{a, b, c\}^*\}$.

1.6.12.

Să se construiască un automat push-down care acceptă limbajul $L = \{w \tilde{w} \mid w \in \{a, b, c\}^*\}$.

1.6.13.

Să se arate că cele două criterii de acceptare ale unui APD sunt echivalente, adică:

a) Fiind dat un APD care acceptă un limbaj L după criteriul stivei vide, atunci \exists un automat APD care acceptă același limbaj după criteriul stării finale (Să se dea construcția);

b) analog, în sens invers.

1.6.14.

Să se arate că nu pentru orice APD nedeterminist există unul determinist echivalent cu el; adică există limbaje independente de context care nu sunt acceptate de un APD determinist ci doar de unul nedeterminist.

1.6.15.

Să se construiască un automat push-down care acceptă limbajul

$L = \{w \in \{a, b\}^* \mid nr_a(w) = nr_b(w)\}$.

1.6.4. Indicații și soluții

1.6.1.

Ideea rezolvării constă în:

- la fiecare simbol a citit de pe banda de intrare se adaugă în stivă un simbol A ;

- la terminarea citirii tuturor simbolurilor **a** consecutive de pe banda de intrare, în stivă sunt atâtea simboluri **A** câte simboluri **b** trebuie să urmeze pe banda de intrare pentru a respecta forma cuvintelor din limbaj;
- la fiecare simbol **b** citit de pe banda de intrare se scoate din stivă un simbol **A**;
- după ce s-a citit un simbol **b** de pe banda de intrare, nu pot să mai apară simboluri **a** pe banda de intrare;
- dacă la terminarea citirii cuvântului de pe banda de intrare în stivă este doar simbolul **Z₀** atunci stiva se golește și cuvântul este acceptat după criteriul stivei vide;

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$;

$Q = \{q_0, q_1\}$, $\Sigma = \{a, b\}$, $\Gamma = \{Z_0, A\}$;

$F = \emptyset$ (automatul acceptă limbajul după criteriul stivei vide);

$\delta: \{q_0, q_1\} \times \{a, b, \epsilon\} \times \{Z_0, A\} \rightarrow P(\{q_0, q_1\} \times \{Z_0, A\}^*)$ și este definită astfel:

q_0 - este starea în care se citesc repetat simboluri **a** de pe banda de intrare adăugându-se câte un **A** în stivă;

q_1 - este starea în care se trece după citirea primului **b** de pe banda de intrare, apoi se vor citi repetat **b**-uri ștergându-se câte un **A** din stivă pentru fiecare **b**;

1. $\delta(q_0, \epsilon, Z_0) = \{(q_0, \epsilon)\}$

-este acceptată secvența vidă ϵ prin vidarea stivei;

2. $\delta(q_0, b, Z_0) = \emptyset$

-cuvântul nu poate să înceapă cu un simbol **b**;

3. $\delta(q_0, a, Z_0) = \{(q_0, A, Z_0)\}$

-s-a citit primul simbol **a** și s-a adăugat un **A** în stivă;

4. $\delta(q_0, \epsilon, A) = \emptyset$

-nu e o situație validă pentru că am terminat de citit cuvântul și nu am citit nici un **b**, deci forma cuvântului nu e cea dorită (cuvântul este format numai din simboluri **a**);

5. $\delta(q_0, a, A) = \{(q_0, AA)\}$

-se citește un **a** de pe banda de intrare și se adaugă un **A** în stivă;

6. $\delta(q_0, b, A) = \{(q_1, \epsilon)\}$

-s-a citit primul **b** și s-a șters un **A** din stivă;

-s-a trecut în starea q_1 pe care o putem numi stare de ștergere;

7. $\delta(q_1, \epsilon, A) = \emptyset$

-s-a terminat de citit cuvântul, dar în stivă au mai rămas simboluri **a**, deci cuvântul are mai multe simboluri **a** decât **b**;

8. $\delta(q_1, b, A) = \{(q_1, \epsilon)\}$

-suntem în starea de ștergere, se citește un **b** de pe banda de intrare, se șterge un **A** din stivă;

9. $\delta(q_1, a, A) = \emptyset$

-din această stare de ștergere nu se poate citi un **a** de pe banda de intrare pentru că forma cuvântului nu ar fi cea dorită (ar alterna **a**-urile și **b**-urile);

10. $\delta(q_1, \epsilon, Z_0) = \{(q_1, \epsilon)\}$

-în acest moment s-a terminat de citit cuvântul de pe banda de intrare, iar în stivă avem doar Z_0 , deci ștergem și acest simbol, stiva devine vidă, deci cuvântul este acceptat;

11. $\delta(q_1, a, Z_0) = \emptyset$

-din starea q_1 nu putem citi un **a** (forma cuvântului nu convine pentru că alternează **a**-urile și **b**-urile");

12. $\delta(q_1, b, Z_0) = \emptyset$

-nu e o situație validă pentru că citim un **b** dar în stivă nu mai există nici un **A** pentru a fi șters (cuvântul are mai multe simboluri **b** decât **a**);

Tabelar funcția δ se reprezintă astfel:

QxΓ		$\Sigma \cup \{\epsilon\}$		
		A	b	ϵ
Q ₀	Z ₀	$\{(q_0, AZ_0)\}$	\emptyset	$\{(q_0, \epsilon)\}$
	A	$\{(q_0, AA)\}$	$\{(q_1, \epsilon)\}$	\emptyset
Q ₁	Z ₀	\emptyset	\emptyset	$\{(q_1, \epsilon)\}$
	A	\emptyset	\emptyset	

Exemple :

1) w=aabb

$(q_0, aabb, Z_0) \xrightarrow{3} (q_0, abb, AZ_0) \xrightarrow{5} (q_0, bb, AAZ_0) \xrightarrow{6} (q_1, b, AZ_0) \xrightarrow{8} (q_1, \epsilon, Z_0) \xrightarrow{10} (q_1, \epsilon, \epsilon)$

- s-a ajuns la o configurație finală după criteriul stivei vide, deci cuvântul este acceptat de automat;

2) w= ϵ

$(q_0, \epsilon, Z_0) \xrightarrow{1} (q_0, \epsilon, \epsilon)$

-cuvântul este acceptat după criteriul stivei vide;

3) w=abab

$(q_0, abab, Z_0) \xrightarrow{3} (q_0, bab, AZ_0) \xrightarrow{6} (q_1, ab, Z_0) \xrightarrow{11}$ blocare

-cuvântul nu este acceptat, forma lui nu convine;

4) w=ba

$(q_0, ba, Z_0) \sqsubseteq_2$ blocare

-cuvântul nu este acceptat, forma lui nu convine;

5) $w=aab$

$(q_0, aab, Z_0) \sqsubseteq_3$ $(q_0, ab, AZ_0) \sqsubseteq_5$ $(q_0, b, AAZ_0) \sqsubseteq_6$ $(q_1, \varepsilon, AZ_0) \sqsubseteq_7$ blocare

-cuvântul nu este acceptat, forma lui nu convine;

6) $w=aaba$

$(q_0, aaba, Z_0) \sqsubseteq_3$ $(q_0, aba, AZ_0) \sqsubseteq_5$ $(q_0, ba, AAZ_0) \sqsubseteq_6$ $(q_1, a, AZ_0) \sqsubseteq_9$ blocare

-cuvântul nu este acceptat, forma lui nu convine;

7) $w=aa$

$(q_0, aa, Z_0) \sqsubseteq_3$ $(q_0, a, AZ_0) \sqsubseteq_5$ $(q_0, \varepsilon, AAZ_0) \sqsubseteq_4$ blocare

-cuvântul nu este acceptat, forma lui nu convine;

8) $w=abb$

$(q_0, abb, Z_0) \sqsubseteq_3$ $(q_0, bb, AZ_0) \sqsubseteq_6$ $(q_1, b, Z_0) \sqsubseteq_{12}$ blocare

-cuvântul nu este acceptat, forma lui nu convine;

1.6.2.

Vom prezenta două automate echivalente care acceptă limbajul L :

a) la fiecare simbol a citit de pe banda de intrare se adaugă în stivă 3 simboluri A ;

- după citirea tuturor simbolurilor a în stivă sunt atâtea simboluri A , câte simboluri b trebuie să urmeze pe banda de intrare;

- la fiecare b citit se scoate din stivă un A ;

- cuvintele din limbaj vor fi acceptate după criteriul stării finale;

- automatul este determinist

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

$Q = \{q_0, q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{Z_0, A\}$

$F = \{q_2\}$ criteriul de acceptare este cel al stării finale;

$\delta: \{q_0, q_1, q_2\} \times \{a, b, \varepsilon\} \times \{Z_0, A\} \rightarrow P(\{q_0, q_1, q_2\} \times \{Z_0, A\}^*)$

și este definită astfel:

$\delta(q_0, a, Z_0) = \{(q_0, AAAZ_0)\}$

$\delta(q_0, a, A) = \{(q_0, AAAA)\}$

$\delta(q_0, b, A) = \{(q_1, \varepsilon)\}$

$\delta(q_1, b, A) = \{(q_1, \varepsilon)\}$

$\delta(q_1, \varepsilon, Z_0) = \{(q_2, Z_0)\}$

- în acest moment s-a terminat de citit cuvântul de pe banda de intrare, iar în stivă este doar simbolul Z_0 ; (deci cuvântul este de forma dorită). Pentru ca automatul să accepte cuvântul după criteriul stării finale el trebuie să treacă într-o stare finală q_2 ;

$\delta(., ., .) = \emptyset$ în celelalte cazuri;

b) la fiecare simbol a citit se adaugă în stivă un simbol A ;

- la terminarea citirii simbolurilor a , în stivă vor fi de trei ori mai puține simboluri A decât numărul simbolurilor b care trebuie citite în continuare;

- la fiecare grup de 3 simboluri **b** citite se va scoate din stivă un **A**, aceasta se realizează folosind încă două stări intermediare;
- cuvintele vor fi acceptate după criteriul stivei vide;
- automatul este determinist;

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

$$Q = \{q_0, q_1, q_2, q_3\}, \Sigma = \{a, b\}, \Gamma = \{Z_0, A\}$$

$$F = \emptyset \text{ criteriul de acceptare este cel al stivei vide}$$

$$\delta: \{q_0, q_1, q_2, q_3\} \times \{a, b, \varepsilon\} \times \{Z_0, A\} \rightarrow P(\{q_0, q_1, q_2, q_3\} \times \{Z_0, A\}^*)$$

și este definită astfel:

$$\delta(q_0, a, Z_0) = \{(q_0, AZ_0)\}$$

$$\delta(q_0, a, A) = \{(q_0, AA)\}$$

$$\delta(q_0, b, A) = \{(q_1, A)\}$$

-s-a citit primul simbol **b**, stiva rămâne nemodificată, se trece în starea intermediară q_1 ;

$$\delta(q_1, b, A) = \{(q_2, A)\}$$

-s-a citit un număr multiplu de 3 plus 2 simboluri **b**, nu se modifică conținutul stivei, se trece în starea intermediară q_2 ;

$$\delta(q_2, b, A) = \{(q_3, \varepsilon)\}$$

-s-a citit un număr multiplu de 3 simboluri **b**, se șterge un **A** se revine la stare q_3 pentru a se citi un alt grup de 3 simboluri **b**;

$$\delta(q_3, b, A) = \{(q_1, A)\}$$

-s-a citit un număr multiplu de 3 plus 1 simboluri **b**, stiva rămâne nemodificată, se trece în starea intermediară q_1 ;

$$\delta(q_3, \varepsilon, Z_0) = \{(q_3, \varepsilon)\}$$

$$\delta(., ., .) = \emptyset \text{ în celelalte cazuri;}$$

Observații:

1°. Când automatul este în starea q_1 s-au citit un număr multiplu de 3 plus 1 simboluri **b**;

2°. Când automatul este în starea q_2 s-au citit un număr multiplu de 3 plus 2 simboluri **b**;

3°. Când automatul este în starea q_3 s-au citit un număr multiplu de 3 simboluri **b**.

1.6.3.

Vom prezenta două automate echivalente care acceptă limbajul $\{a^{3n}b^n\}$:

a) la un grup de trei simboluri **a** citite de pe banda de intrare se adaugă în stivă un simbol **A**, aceasta se realizează folosind încă două stări intermediare;

-după citirea tuturor simbolurilor **a**, în stivă sunt atâtea simboluri **A**, câte simboluri **b** trebuie să urmeze pe banda de intrare;

-la fiecare **b** citit se scoate din stivă un **A**;

-cuvintele din limbaj vor fi acceptate după criteriul stivei vide;

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

$$Q = \{q_0, q_1, q_2, q_3\}, \Sigma = \{a, b\}, \Gamma = \{A, Z_0\}$$

$$F = \emptyset \text{ criteriul de acceptare este cel al stivei vide}$$

$$\delta: \{q_0, q_1, q_2, q_3\} \times \{a, b, \varepsilon\} \times \{Z_0, A\} \rightarrow P(\{q_0, q_1, q_2, q_3\} \times \{Z_0, A\}^*)$$

și este definită astfel:

$$\begin{aligned}
\delta(q_0, \varepsilon, Z_0) &= \{(q_0, \varepsilon)\}; \\
\delta(q_0, a, Z_0) &= \{(q_1, AZ_0)\}; \\
\delta(q_1, a, A) &= \{(q_2, A)\}; \\
\delta(q_2, a, A) &= \{(q_0, A)\}; \\
\delta(q_0, a, A) &= \{(q_1, AA)\}; \\
\delta(q_0, b, A) &= \{(q_3, \varepsilon)\}; \\
\delta(q_3, b, A) &= \{(q_3, \varepsilon)\}; \\
\delta(q_3, \varepsilon, Z_0) &= \{(q_0, \varepsilon)\}; \\
\delta(., ., .) &= \emptyset \text{ în celelalte cazuri.}
\end{aligned}$$

b) la fiecare simbol **a** citit se adaugă în stivă un simbol **A**;

-la terminarea citirii simbolurilor **a**, În stivă vor fi de trei ori mai multe simboluri decât cele care trebuie citite în continuare;

-la fiecare **b** citit se vor scoate din stivă trei simboluri **A**. Deoarece numai simbolul din vârful stivei se poate șterge la un moment dat, ștergerea a trei simboluri se va face folosind încă două stări intermediare din care au loc doar ε -tranziții;

-cuvintele vor fi acceptate după criteriul stivei vide;

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

$$Q = \{q_0, q_1, q_2, q_3\}, \Sigma = \{a, b\}, \Gamma = \{Z_0, A\}$$

$$F = \emptyset \text{ criteriul de acceptare este cel al stivei vide}$$

$\delta: \{q_0, q_1, q_2, q_3\} \times \{a, b, \varepsilon\} \times \{Z_0, A\} \rightarrow P(\{q_0, q_1, q_2, q_3\} \times \{Z_0, A\}^*)$ și este definită astfel:

$$\delta(q_0, \varepsilon, Z_0) = \{(q_0, \varepsilon)\} \text{ secvența vidă este acceptată;}$$

$$\delta(q_0, a, Z_0) = \{(q_0, AZ_0)\}$$

-s-a citit primul **a** din cuvânt și s-a adăugat un **A** în stivă;

$$\delta(q_0, a, A) = \{(q_0, AA)\}$$

-s-a citit un **a** și s-a adăugat un **A** în stivă;

$$\delta(q_0, b, A) = \{(q_1, \varepsilon)\}$$

-s-a șters primul **A** corespunzător primului **b** citit;

$$\delta(q_1, \varepsilon, A) = \{(q_2, \varepsilon)\}$$

-s-a șters al doilea **A** corespunzător ultimului **b** citit fără a se înainta pe banda de intrare (are loc o ε -tranziție);

$$\delta(q_2, \varepsilon, A) = \{(q_3, \varepsilon)\}$$

-s-a șters al treilea **A** corespunzător ultimului **b** citit fără a se înainta pe banda de intrare (are loc o ε -tranziție);

$$\delta(q_3, b, A) = \{(q_1, \varepsilon)\}$$

-s-a șters primul **A** corespunzător **b**-ului care tocmai s-a citit;

$$\delta(q_3, \varepsilon, Z_0) = \{(q_3, \varepsilon)\}$$

-s-a golit stiva;

$$\delta(., ., .) = \emptyset \text{ în celelalte cazuri.}$$

Observații:

1°. Când automatul este în starea q_1 s-au citit un număr multiplu de 3 plus 1 simboluri **b**;

2o. Când automatul este în starea q_2 s-au citit un număr multiplu de 3 plus 2 simboluri b ;

3o. Când automatul este în starea q_3 s-au citit un număr multiplu de 3 simboluri b .

1.6.4.

Vom prezenta un automat care acceptă limbajul L construit folosind combinarea automatelor de la problemele 1.6.2. și 1.6.3:

- la fiecare al doilea a citit se adaugă în stivă un A ;
- la fiecare al treilea b citit se șterge din stivă un A ;
- cuvintele limbajului sunt acceptate după criteriul stivei vide;

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

$Q = \{q_0, q_1, q_2, q_3, q_4\}$, $\Sigma = \{a, b\}$, $\Gamma = \{Z_0, A\}$

$F = \emptyset$ criteriul de acceptare este cel al stivei vide

$\delta: \{q_0, q_1, q_2, q_3, q_4\} \times \{a, b, \varepsilon\} \times \{Z_0, A\} \rightarrow P(\{q_0, q_1, q_2, q_3, q_4\} \times \{Z_0, A\}^*)$

și este definită astfel:

$\delta(q_0, \varepsilon, Z_0) = \{(q_0, \varepsilon)\}$ $\delta(q_2, b, A) = \{(q_3, A)\}$

$\delta(q_0, a, Z_0) = \{(q_1, Z_0)\}$ $\delta(q_3, b, A) = \{(q_4, \varepsilon)\}$

$\delta(q_1, a, A) = \{(q_0, AA)\}$ $\delta(q_4, b, A) = \{(q_2, A)\}$

$\delta(q_0, a, A) = \{(q_1, A)\}$ $\delta(q_4, \varepsilon, Z_0) = \{(q_4, \varepsilon)\}$

$\delta(q_0, b, A) = \{(q_2, A)\}$

$\delta(., ., .) = \emptyset$ În celelalte cazuri.

1.6.5.

Vom prezenta un automat nedeterminist care acceptă limbajul L .

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

$Q = \{q_0, q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{Z_0, A\}$

$F = \emptyset$ criteriul de acceptare este cel al stivei vide

$\delta: \{q_0, q_1, q_2\} \times \{a, b, \varepsilon\} \times \{Z_0, A\} \rightarrow P(\{q_0, q_1, q_2\} \times \{Z_0, A\}^*)$

și este definită astfel:

$\delta(q_0, a, Z_0) = \{(q_0, AZ_0), (q_2, AAZ_0)\}$

$\delta(q_0, a, A) = \{(q_0, AA)\}$

$\delta(q_0, b, A) = \{(q_1, \varepsilon)\}$

$\delta(q_1, b, A) = \{(q_1, \varepsilon)\}$

$\delta(q_2, a, A) = \{(q_2, AAA)\}$

$\delta(q_2, b, A) = \{(q_1, \varepsilon)\}$

$\delta(q_1, \varepsilon, Z_0) = \{(q_1, \varepsilon)\}$

$\delta(., ., .) = \emptyset$ în celelalte cazuri.

1.6.6.

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

$Q = \{q_0, q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{Z_0, A\}$

$F = \{q_2\}$ criteriul de acceptare este cel al stării finale

$\delta: \{q_0, q_1, q_2\} \times \{a, b, \varepsilon\} \times \{Z_0, A\} \rightarrow P(\{q_0, q_1, q_2\} \times \{Z_0, A\}^*)$

și este definită astfel:

$\delta(q_0, a, Z_0) = \{(q_0, AZ_0)\}$

$\delta(q_0, a, A) = \{(q_0, AA)\}$

$$\delta(q_0, b, A) = \{(q_1, \varepsilon)\}$$

$$\delta(q_1, b, A) = \{(q_1, \varepsilon)\}$$

$$\delta(q_1, b, Z_0) = \{(q_2, Z_0)\}$$

-dacă se ajunge la starea q_1 și Z_0 în vârful stivei înseamnă că s-a citit subcuvântul de forma $a^n b^n$ și deci se pot citi în continuare oricâți de b ; pentru aceasta trecem într-o stare finală q_2 , stiva nu se modifică;

$$\delta(q_2, b, Z_0) = \{(q_2, Z_0)\}$$

-permite citirea unui număr oarecare de simboluri b , conținutul stivei nu se modifică;

$$\delta(., ., .) = \emptyset \text{ în celelalte cazuri.}$$

1.6.7.

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

$$Q = \{q_0, q_1, q_2\}, \Sigma = \{a, b\}, \Gamma = \{Z_0, A\}$$

$$F = \{q_2\} \text{ criteriul de acceptare este cel al stării finale}$$

$$\delta: \{q_0, q_1, q_2\} \times \{a, b, \varepsilon\} \times \{Z_0, A\} \rightarrow P(\{q_0, q_1, q_2\} \times \{Z_0, A\}^*)$$

și este definită astfel:

$$\delta(q_0, a, Z_0) = \{(q_0, AZ_0)\}$$

$$\delta(q_0, a, A) = \{(q_0, AA)\}$$

$$\delta(q_0, b, A) = \{(q_1, \varepsilon)\}$$

$$\delta(q_1, b, A) = \{(q_1, \varepsilon)\}$$

$$\delta(q_1, \varepsilon, A) = \{(q_2, A)\}$$

-dacă se ajunge la starea q_1 , A în vârful stivei și cuvântul a fost citit în întregime, înseamnă că s-au citit mai puține simboluri b decât a -uri, deci trecem într-o stare finală q_2 și cuvântul este acceptat;

$$\delta(., ., .) = \emptyset \text{ în celelalte cazuri.}$$

1.6.8.

a) automatul este nedeterminist

-la fiecare a citit se adaugă în stivă unul sau două simboluri A ;

-la fiecare b citit se șterge un A din stivă.

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

$$Q = \{q_0, q_1\}, \Sigma = \{a, b\}, \Gamma = \{Z_0, A\}$$

$$F = \emptyset \text{ criteriul de acceptare este cel al stivei vide}$$

$$\delta: \{q_0, q_1\} \times \{a, b, \varepsilon\} \times \{Z_0, A\} \rightarrow P(\{q_0, q_1\} \times \{Z_0, A\}^*)$$

și este definită astfel:

$$\delta(q_0, a, Z_0) = \{(q_0, AZ_0), (q_0, AAZ_0)\}$$

$$\delta(q_0, a, A) = \{(q_0, AA), (q_0, AAA)\}$$

$$\delta(q_0, b, A) = \{(q_1, \varepsilon)\}$$

$$\delta(q_1, b, A) = \{(q_1, \varepsilon)\}$$

$$\delta(q_1, \varepsilon, Z_0) = \{(q_1, \varepsilon)\}$$

$$\delta(., ., .) = \emptyset \text{ în celelalte cazuri.}$$

b)

-la fiecare a citit se adaugă în stivă un A ;

-la fiecare **b** citit sau la un grup de două **b**-uri citite se șterge din stivă un **A**;

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

$Q = \{q_0, q_1, q_2\}, \Sigma = \{a, b\}, \Gamma = \{Z_0, A\}$

$F = \emptyset$ criteriul de acceptare este cel al stivei vide

$\delta: \{q_0, q_1, q_2\} \times \{a, b, \varepsilon\} \times \{Z_0, A\} \rightarrow P(\{q_0, q_1, q_2\} \times \{Z_0, A\}^*)$

și este definită astfel:

$\delta(q_0, a, Z_0) = \{(q_0, AZ_0)\}$

$\delta(q_0, a, A) = \{(q_0, AA)\}$

$\delta(q_0, b, A) = \{(q_1, \varepsilon), (q_2, \varepsilon)\}$

$\delta(q_1, b, A) = \{(q_1, \varepsilon), (q_2, \varepsilon)\}$

$\delta(q_2, b, A) = \{(q_1, A)\}$

$\delta(q_1, \varepsilon, Z_0) = \{(q_1, \varepsilon)\}$

$\delta(., ., .) = \emptyset$ în celelalte cazuri.

1.6.9.

-automatul este nedeterminist;

-este o generalizare a problemei precedente;

-la fiecare **a** citit se adaugă în stivă p sau **p+1** sau ... **q** simboluri **A**;

-la fiecare **b** citit se șterge un **A** din stivă;

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

$Q = \{q_0, q_1\}, \Sigma = \{a, b\}, \Gamma = \{Z_0, A\}$

$F = \emptyset$ criteriul de acceptare este cel al stivei vide

$\delta: \{q_0, q_1\} \times \{a, b, \varepsilon\} \times \{Z_0, A\} \rightarrow P(\{q_0, q_1\} \times \{Z_0, A\}^*)$

și este definită astfel:

$\delta(q_0, a, Z_0) = \{(q_0, A^p Z_0), (q_0, A^{p+1} Z_0), \dots, (q_0, A^q Z_0)\}$

$\delta(q_0, a, A) = \{(q_0, A^p A), (q_0, A^{p+1} A), \dots, (q_0, A^q A)\}$

$\delta(q_0, b, A) = \{(q_1, \varepsilon)\}$

$\delta(q_1, b, A) = \{(q_1, \varepsilon)\}$

$\delta(q_1, \varepsilon, Z_0) = \{(q_1, \varepsilon)\}$

$\delta(., ., .) = \emptyset$ în celelalte cazuri.

1.6.10.

a)

-automatul este nedeterminist;

-la fiecare **a** citit se adaugă în stivă un **A**;

-la fiecare **b** citit se șterge din stivă unul sau două simboluri **A**;

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

$Q = \{q_0, q_1, q_2\}, \Sigma = \{a, b\}, \Gamma = \{Z_0, A\}$

$F = \emptyset$ criteriul de acceptare este cel al stivei vide

$\delta: \{q_0, q_1, q_2\} \times \{a, b, \varepsilon\} \times \{Z_0, A\} \rightarrow P(\{q_0, q_1, q_2\} \times \{Z_0, A\}^*)$

și este definită astfel:

$\delta(q_0, a, Z_0) = \{(q_0, AZ_0)\}$

$\delta(q_0, a, A) = \{(q_0, AA)\}$

$$\begin{aligned}\delta(q_0, b, A) &= \{(q_1, \varepsilon), (q_2, \varepsilon)\} \\ \delta(q_1, b, A) &= \{(q_1, \varepsilon), (q_2, \varepsilon)\} \\ \delta(q_2, \varepsilon, A) &= \{(q_1, \varepsilon)\} \\ \delta(q_1, \varepsilon, Z_0) &= \{(q_1, \varepsilon)\} \\ \delta(., ., .) &= \emptyset \text{ în celelalte cazuri.}\end{aligned}$$

b)

- la fiecare **a** citit sau la un grup de două simboluri **a** citite se adaugă în stivă un **A**;
- la fiecare **b** citit se șterge un **A** din stivă;

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

$$Q = \{q_0, q_1, q_2\}, \Sigma = \{a, b\}, \Gamma = \{Z_0, A\}$$

$F = \emptyset$ criteriul de acceptare este cel al stivei vide

$$\delta: \{q_0, q_1, q_2\} \times \{a, b, \varepsilon\} \times \{Z_0, A\} \rightarrow P(\{q_0, q_1, q_2\} \times \{Z_0, A\}^*)$$

și este definită astfel:

$$\begin{aligned}\delta(q_0, a, Z_0) &= \{(q_0, AZ_0), (q_1, AZ_0)\}; & \delta(q_0, b, A) &= \{(q_2, \varepsilon)\}; \\ \delta(q_0, a, A) &= \{(q_0, AA), (q_1, AA)\}; & \delta(q_2, b, A) &= \{(q_2, \varepsilon)\}; \\ \delta(q_1, a, A) &= \{(q_0, A)\}; & \delta(q_2, \varepsilon, Z_0) &= \{(q_2, \varepsilon)\}; \\ \delta(., ., .) &= \emptyset \text{ în celelalte cazuri.}\end{aligned}$$

1.6.11.

- automatul este determinist;
- se adaugă în stivă toate simbolurile care se citesc până la mijlocul cuvântului (marcat de simbolul "d");
- după citirea lui "d" dacă simbolul din vârful stivei coincide cu cel citit atunci se șterge un element din stivă;
- limbajul este acceptat după criteriul stivei vide;

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

$$Q = \{q_0, q_1\}, \Sigma = \{a, b, c, d\}, \Gamma = \{Z_0, a, b, c\}$$

$F = \emptyset$ criteriul de acceptare este cel al stivei vide

$$\delta: \{q_0, q_1\} \times \{a, b, c, d, \varepsilon\} \times \{Z_0, a, b, c\} \rightarrow P(\{q_0, q_1\} \times \{Z_0, a, b, c\}^*)$$

și este definită astfel:

$$\begin{aligned}\delta(q_0, x, Z_0) &= \{(q_0, xZ_0)\} & \forall x \in \{a, b, c\} \\ \delta(q_0, d, Z_0) &= \{(q_0, \varepsilon)\} \text{ este acceptat cuvântul } w=d \\ \delta(q_0, x, y) &= \{(q_0, xy)\} & \forall x, y \in \{a, b, c\} \\ \delta(q_0, d, x) &= \{(q_1, x)\} & \forall x \in \{a, b, c\}\end{aligned}$$

- s-a ajuns la mijlocul cuvântului și se trece într-o stare de ștergere; stiva rămâne nemodificată;

$$\delta(q_1, x, x) = \{(q_1, \varepsilon)\} \quad \forall x \in \{a, b, c\}$$

- dacă simbolul citit coincide cu cel din vârful stivei, acesta se șterge din stivă;

$$\delta(q_1, \varepsilon, Z_0) = \{(q_1, \varepsilon)\}$$

- vidarea stivei;

$$\delta(., ., .) = \emptyset \text{ în celelalte cazuri.}$$

1.6.12.

-automatul este nedeterminist;

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

$$Q = \{q_0, q_1\}, \Sigma = \{a, b, c\}, \Gamma = \{Z_0, a, b, c\}$$

$F = \emptyset$ criteriul de acceptare este cel al stivei vide

$$\delta: \{q_0, q_1\} \times \{a, b, c, \varepsilon\} \times \{Z_0, a, b, c\} \longrightarrow P(\{q_0, q_1\} \times \{Z_0, a, b, c\}^*)$$

și este definită astfel:

$$\delta(q_1, \varepsilon, Z_0) = \{(q_1, \varepsilon)\} \text{ este acceptată secvența vidă}$$

$$\delta(q_0, x, Z_0) = \{(q_0, xZ_0)\} \quad \forall x \in \{a, b, c\}$$

$$\delta(q_0, x, y) = \{(q_0, xy)\} \quad \forall x, y \in \{a, b, c\} \text{ și } x \neq y$$

$$\delta(q_0, x, x) = \{(q_0, xx), (q_1, \varepsilon)\} \quad \forall x \in \{a, b, c\}$$

-dacă simbolul citit coincide cu cel din vârful stivei sunt două variante (nu se poate determina varianta, de aici apare nedeterminismul):

a) nu s-a ajuns încă la mijlocul cuvântului și atunci adăugăm în stivă simbolul citit;

b) s-a ajuns la mijlocul cuvântului și atunci trebuie să începem ștergerea elementelor din stivă;

$$\delta(q_1, x, x) = \{(q_1, \varepsilon)\} \quad \forall x \in \{a, b, c\}$$

-după ce s-a trecut de mijlocul cuvântului (suntem în stare de ștergere q_1), dacă simbolul citit coincide cu cel din vârful stivei se șterge din stivă;

$$\delta(q_1, \varepsilon, Z_0) = \{(q_1, \varepsilon)\}$$

$$\delta(., ., .) = \emptyset \text{ în celelalte cazuri.}$$

1.6.15.

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

$$Q = \{q_0, q_1\}, \Sigma = \{a, b\}, \Gamma = \{Z_0, A, B\}$$

$F = \emptyset$ criteriul de acceptare este cel al stivei vide

$$\delta: \{q_0, q_1\} \times \{a, b, \varepsilon\} \times \{Z_0, A, B\} \longrightarrow P(\{q_0, q_1\} \times \{Z_0, A, B\}^*)$$

și este definită astfel:

$$\delta(q_0, \varepsilon, Z_0) = \{(q_0, \varepsilon)\}; \quad \delta(q_0, a, Z_0) = \{(q_0, AZ_0)\};$$

$$\delta(q_0, b, Z_0) = \{(q_1, BZ_0)\}; \quad \delta(q_0, a, A) = \{(q_0, AA)\};$$

$$\delta(q_0, b, A) = \{(q_0, \varepsilon)\}; \quad \delta(q_1, \varepsilon, Z_0) = \{(q_1, \varepsilon)\};$$

$$\delta(q_1, a, Z_0) = \{(q_0, AZ_0)\}; \quad \delta(q_1, b, Z_0) = \{(q_1, BZ_0)\};$$

$$\delta(q_1, a, B) = \{(q_1, \varepsilon)\}; \quad \delta(q_1, b, B) = \{(q_1, BB)\};$$

$$\delta(., ., .) = \emptyset \text{ în celelalte cazuri.}$$

1.7. TRANSLATARE ȘI TRANSLATOARE

Fie Σ un alfabet numit *alfabet de intrare* și D un alfabet numit *alfabet de ieșire*. Fie $L_1 \subset \Sigma^*$, $L_2 \subset D^*$ două limbaje (de intrare, respectiv de ieșire).

Numim *translatare* a limbajului L_1 în limbajul L_2 o relație T de la Σ^* la D^* a cărei domeniu de definiție este L_1 , iar imaginea lui T este L_2 .

Dacă pentru $y \in D^*$ există $x \in \Sigma^*$ astfel ca $(x, y) \in T$ atunci y este o *translatare a lui x*.

Definim *translator* pentru translatarea T ca fiind un "dispozitiv" care este capabil să producă pentru orice șir de intrare x un șir de ieșire y astfel încât $(x, y) \in T$.

1.7.1. Translator finit

Numim *translator finit* un ansamblu $M = (Q, \Sigma, D, \delta, q_0, F)$ unde:

- Q este o mulțime nevidă și finită formată din *stări*;
- Σ este alfabetul de *intrare*;
- D este alfabetul de *ieșire*;
- $q_0 \in Q$, q_0 este *starea inițială*;
- $F \subseteq Q$, F este mulțimea *stărilor finale*;
- $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow P(Q \times D^*)$ este *funcția de tranziție* cu valori în mulțimea părților finite.

Observație.

Dacă $q \in Q$ și $a \in \Sigma \cup \{\varepsilon\}$ atunci $\delta(q, a) = \{(p_1, \gamma_1), \dots, (p_n, \gamma_n)\}$, $(p_i, \gamma_i) \in Q \times D^*$, $i = \overline{1, n}$.

Tripletul $(q, x, y) \in Q \times \Sigma^* \times D^*$ unde q este starea translatorului, x este secvența ce urmează să fie citită la intrare, iar y este secvența scrisă la ieșire, se numește *configurație*.

Pe mulțimea configurațiilor $Q \times \Sigma^* \times D^*$ se definesc relațiile binare:

- a) \mapsto tranziție directă
 $(q, ax, y) \mapsto (p, x, yz) \iff ((qa) \ni (p, z), a \in \Sigma \cup \{\varepsilon\}, x \in \Sigma^*, y, z \in D^*).$
- b) \xrightarrow{k} k tranziția (o succesiune de k tranziții directe);
- c) \mapsto^+ + tranziția (închiderea tranzitivă a tranziției directe);
- d) \mapsto^* * tranziția (închiderea reflexivă și tranzitivă)

Dacă M este un translator finit, atunci y este o ieșire pentru intrarea x , dacă avem:

$$(q_0, x, \varepsilon) \vdash^* (q, \varepsilon, y), q \in F$$

adică din configurația inițială (q_0, x, ε) printr-un șir de tranziții directe ajungem în configurația finală (q, ε, y) , $q \in F$.

Translatarea definită de translatorul $M = (Q, \Sigma, D, \delta, q_0, F)$ este:

$$T(M) = \{(x, y) \mid x \in \Sigma^*, y \in D^*, (q_0, x, \varepsilon) \vdash^* (q, \varepsilon, y), q \in F\}$$

1.7.2. Translator push-down

Numim translator push-down un ansamblu $M = (Q, \Sigma, \Gamma, D, \delta, q_0, Z_0, F)$ unde:

- Q este alfabetul *stărilor*;
- Σ este alfabetul de *intrare*;
- Γ este alfabetul memoriei *stivă*;
- D este alfabetul de *ieșire*;
- $q_0 \in Q$, q_0 este *starea inițială*;
- $F \subseteq Q$, F este mulțimea *stărilor finale*;
- Z_0 este *simbolul inițial* al stivei;
- $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow P(Q \times \Gamma^* \times D^*)$ este *funcția de tranziție* cu valori în mulțimea părților finite.

Observație.

Dacă $q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $Z \in \Gamma$ atunci

$$\delta(q, a, Z) = \{(p_1, \gamma_1, \beta_1), (p_2, \gamma_2, \beta_2), \dots, (p_n, \gamma_n, \beta_n)\}, \quad (p_i, \gamma_i, \beta_i) \in Q \times \Gamma^* \times D^*, \quad i = \overline{1, n}.$$

Apoi $(q, x, \alpha, y) \in Q \times \Sigma^* \times \Gamma^* \times D^*$ unde q este starea translatorului push-down, x este secvența ce urmează să fie citită la intrare, α este secvența din memoria stivă, iar y este secvența scrisă la ieșire, se numește *configurație*.

Pe mulțimea configurațiilor $Q \times \Sigma^* \times \Gamma^* \times D^*$ se definesc relațiile binare:

- a) \mapsto tranziție directă $(q, ax, Z\gamma, y) \mapsto (p, x, \alpha\gamma, yt) \iff \delta(q, a, Z) \ni (p, \alpha, t)$;
unde: $q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $x \in \Sigma^*$, $Z \in \Gamma$, iar $\alpha, \gamma \in \Gamma^*$ și $y, t \in D^*$.
- b) \mapsto^k k tranziția (o succesiune de k tranziții directe);
- c) \mapsto^+ $+$ tranziția (închiderea tranzitivă a tranziției directe);
- d) \mapsto^* $*$ tranziția (închiderea reflexivă și tranzitivă).

Secvența y este o ieșire pentru intrarea x după *criteriul stării finale* dacă (o trecere din configurația inițială în configurația finală) avem:

$$(q_0, x, Z_0, \varepsilon) \stackrel{*}{\rightarrow} (q, \varepsilon, \gamma, y), q \in F,$$

iar translatarea după acest criteriu este:

$$T_f(M) = \{(x, y) \mid x \in \Sigma^*, y \in D^*, (q_0, x, Z_0, \varepsilon) \stackrel{*}{\rightarrow} (q, \varepsilon, \gamma, y), q \in F\}.$$

Analog definim translatarea după criteriul stivei vide:

$$T_v(M) = \{(x, y) \mid x \in \Sigma^*, y \in D^*, (q_0, x, Z_0, \varepsilon) \stackrel{*}{\rightarrow} (q, \varepsilon, \varepsilon, y)\}.$$

1.7.3. Probleme propuse.

1.7.1.

Fie translatorul finit $M = (\{q_0, q_1\}, \{a\}, \{b\}, \delta, q_0, \{q_1\})$ unde

$$\delta(q_0, a) = \{(q_1, b)\},$$

$$\delta(q_1, \varepsilon) = \{(q_1, b)\},$$

$$\delta(.,.) = \emptyset \text{ în restul cazurilor.}$$

Să se determine translatarea definită de M .

1.7.2.

Să se construiască un translator push-down care transformă o expresie aritmetică de la forma poloneză prefixată în forma poloneză postfixată. Presupunem că expresia aritmetică conține operatorii binari $+$, $*$, și operanzii simbolizați prin a .

1.7.4. Indicații și soluții

1.7.1.

Translatarea obținută este $T(M) = \{(a, b^i) \mid i \geq 1\}$.

1.7.2.

Un translator push-down care funcționează după criteriul stivei vide și care transformă o expresie aritmetică din forma poloneză prefixată în forma poloneză postfixată este

$M = (Q, \Sigma, \Gamma, D, q_0, Z_0, \emptyset)$ unde:

$$Q = \{q\};$$

$$\Sigma = \{a, +, *\};$$

$$\Gamma = \{E, +, *\};$$

$$D = \{a, +, *\};$$

$$Z_0 = E;$$

$$q_0 = q;$$

funcția δ este dată prin:

$$\delta(q, a, E) = \{(q, \varepsilon, a)\};$$

$$\delta(q, +, E) = \{(q, EE+, \varepsilon)\};$$

$$\delta(q, *, E) = \{(q, EE*, \varepsilon)\};$$

$$\delta(q, \varepsilon, +) = \{(q, \varepsilon, +)\};$$

$\delta(q, \epsilon, *) = \{q, \epsilon, *\};$
 $\delta(., ., .) = \emptyset$; în celelalte cazuri.

Exemplu:

fie $w = +a*aa$ forma poloneză prefixată a expresiei aritmetice: $a+a*a$.

Avem:

$(q, +a*aa, E, \epsilon) \vdash (q, a*aa, EE+, \epsilon) \vdash (q, *aa, E+, a) \vdash (q, aa, EE*+, a) \vdash$
 $(q, a, E*+, aa) \mapsto (q, \epsilon, *+, aaa) \vdash (q, \epsilon, +, aaa*) \vdash (q, \epsilon, \epsilon, aaa*+)$

Cuplul $(+a*aa, aaa*+)$ este o traducere.

2. PROBLEME ȘI PROGRAME COMPLEXE

În acest capitol se propun câteva probleme legate de tematica primului capitol. Pentru unele din aceste probleme se dă rezolvarea completă inclusiv programele aferente în limbajul *C*.

2.1. PROBLEME PROPUSE

2.1.

Să se scrie programe în limbajul *C* pentru următoarele probleme:

a) reprezentarea unei gramatici independente de context sub formele:

- vectorială;
- tabelară;
- liste înlănțuite ramificate;

b) trecerea de la reprezentarea vectorială la reprezentarea cu ajutorul listelor înlănțuite.

2.2.

Să se scrie un program *C* pentru reprezentarea unui automat finit determinist și care verifică dacă o secvență peste alfabetul automatului este acceptată sau nu de automat.

2.3.

Să se scrie un program *C* pentru reprezentarea unui automat finit nedeterminist și care verifică dacă o secvență peste alfabetul automatului este acceptată sau nu de automat.

2.4.

Să se scrie o funcție *C* care face trecerea de la o gramatică regulară la automatul finit care acceptă limbajul generat de gramatica dată.

2.5.

Să se scrie o funcție *C* care face trecerea de la un automat finit la gramatica regulară care generează limbajul acceptat de automatul dat.

2.6.

Să se scrie o funcție *C* care face trecerea de la un automat finit nedeterminist dat la un automat finit determinist (echivalent cu cel inițial).

2.7.

Să se scrie o funcție *C* care face trecerea de la un automat finit dat la automatul finit redus echivalent.

2.8.

Având două automate finite M_1 care acceptă limbajul L_1 și M_2 care acceptă limbajul L_2 să se construiască automatele finite care acceptă limbajele:

- $L_1 \cup L_2$ (reuniunea) ;
- $L_1 \cap L_2$ (intersecția);
- $L_1 L_2$ (concatenarea);
- L_1^*, L_1^+ (închiderea reflexivă și tranzitivă, respectiv închiderea tranzitivă).

2.9.

Să se scrie funcții C pentru simplificarea gramaticilor independente de context pornind de la reprezentarea vectorială a acestora (pentru fiecare problemă câte o funcție C):

- eliminarea simbolurilor inaccesibile;
- eliminarea simbolurilor neproductive;
- eliminarea ϵ -producțiilor (epsilon producțiilor);
- eliminarea regulilor de redenumire;
- eliminarea recursivității de stânga.

2.10.

Să se scrie funcții C pentru aducerea unei gramatici independente de context la:

- forma normală Chomsky (FNC);
- forma normală Greibach (FNG).

2.11.

Să se construiască arborele de analiză sintactică corespunzător unei secvențe generate de o gramatică independentă de context (pornind de la reprezentarea tabelară în memorie a gramaticii).

2.12.

Să se construiască arborele binar de analiză sintactică corespunzător unei secvențe generate de o gramatică independentă de context aflată în formă normală Chomsky.

2.13.

Să se găsească o formă de reprezentare în memorie a unui automat push-down. Să se scrie o funcție C pentru reprezentarea găsită și pentru simularea funcționării sale (verificarea acceptării unor secvențe).

2.14.

Să se scrie o funcție C pentru trecerea de la o gramatică independentă de context la automatul push-down care acceptă limbajul generat de gramatica dată.

2.15.

Având o secvență generată de o gramatică regulată, să se dea toate descompunerile posibile ale ei sub forma "xyz" conform lemei de pompare.

2.16.

Având o expresie regulată, să se construiască automatul finit (cu sau fără epsilon mișcări) care acceptă limbajul specificat de expresia dată.

2.17.

Având un automat finit, să se construiască expresia regulată care specifică limbajul acceptat de automatul dat.

2.1. INDICAȚII ȘI SOLUȚII

2.2.

Reprezentarea și simularea funcționării unui automat finit determinist (verificarea dacă o secvență este acceptată sau nu de automat).

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <ctype.h>

int este(char x,char M[])
{ /* verifica daca simbolul x apartine multimii M si returneaza
    pozitia sa in multime sau -1 daca nu apartine */
    int i;
    for(i=0;M[i];i++) if(M[i]==x) return i;
    return(-1);
}

void citeste(char M[],char x)
{
    char *mesaj;
    if(x=='s') mesaj="simbolurilor:";
    else if(x=='q')
        mesaj="starilor (prima stare este starea initiala):\n";
    else mesaj="starilor finale:";
    printf("\nDati multimea %s ", mesaj);
    gets(M);
}

void citdelta(char delta[15][10],char Q[],char S[], char F[])
{
    // citeste functia de tranzitie delta din tabel
    int i,j;
    gotoxy(10,4);
    printf("Q\\ %c",179);
    for(i=0;i<strlen(S);i++)
        { gotoxy(10+5+i*3,4);
          printf("%c",S[i]);
        }
    printf(" %c Z",179);
    gotoxy(10,5);
    printf("%c%c%c%c",196,196,196,197);
    for(i=0;i<strlen(S);i++) printf("%c%c%c",196,196,196);
    printf("%c%c%c",197,196,196);
}
```

```

for(i=0;i<strlen(Q);i++)
{
    gotoxy(11,6+i);
    printf("%c %c",Q[i],179);
    gotoxy(14+strlen(S)*3,6+i);
    if(este(Q[i],F)!=-1) j=1;
    else j=0;
    printf("%c %d",179,j);
}
for(i=0;Q[i];i++)
for(j=0;S[j];j++)
{
    /* nu permite decit citirea unei stari */
    do
    { gotoxy(15+j*3,6+i);
      printf(" ");
      gotoxy(15+j*3,6+i);
      delta[i][j]=getche();
    } while(delta[i][j]!=' ' && este(delta[i][j],Q)==-1);
}
}

void main(void)
{
    char Q[15],S[10],F[15],delta[15][10];
    int ind,i,j,k;
    char secv[50],q;
    clrscr();
    printf("SIMULAREA FUNCTIONARII UNUI AUTOMAT FINIT");
    printf("\n          DETERMINIST");
    citeste(Q,'q'); // citeste multimea starilor automatului
    citeste(S,'s'); // citeste multimea simbolurilor

    do{
        b: citeste(F,'f'); // citeste multimea starilor finale
        for(i=0;i<strlen(F);i++)
            if(este(F[i],Q)==-1)
                { printf("\n!! stari eronate , reluati !!\n");
                  goto b;
                }
        break;
    }
    while(1);
    clrscr();
    printf("\nDati functia delta(spatiu daca nu e definita pt. o");
    printf("\npereche stare,simbol)");
    citdelta(delta,Q,S,F); // citeste functia de tranzitie
    printf("\n\n Doriti sa verificati secventa?(d/n):");
    while(tolower(getche())!='n')
    {
        q=Q[0];ind=1;i=0;
        printf("\n Dati secventa (pt secventa vida dati ENTER):\n");
        gets(secv);
        if(strlen(secv)==0)
            {printf("(%c,eps)",q);
             if(este(q,F)==-1)
                 printf("\n%c nu e stare finala, secventa nu e\n acceptata",q);
             else
                 printf("\n  secventa este acceptata");
             goto a;
            }
    }
}

```

```

while(secv[i])
{
    j=este(q,Q);
    k=este(secv[i],S);
    if(k==-1)
        { printf("\n caracter nepermis: %c",secv[i]);
          ind=-1;
          break;
        }
    printf("( %c, %s) | --",q,secv+i);
    q=delta[j][k];
    if(q==' ')
        { printf("\n tranzitie blocata");
          ind=-1;
          break;
        }
    else i++;
}
if(ind==1)
{ printf("( %c, eps)",q);
  if(este(q,F)!=-1)
    { printf("\n    %c este stare finala ",q);
      printf("\n    secventa este acceptata");
    }
  else
    { printf(" \n %c nu este stare finala ",q);
      ind=-1;
    }
}
if(ind==-1) printf("\n secventa nu este acceptata");
a: printf("\n\n Doriti sa verificati secventa?(d/n):");
}
}

```

Vom da în continuare un program care rezolvă aceeași problemă dar scris în C++, implementând clasa *automat* push-down.

```

#include <iostream.h>
#include <string.h>
#include <conio.h>
#define MAX 100
enum bool {False,True};

class automat          // M=( Q, Σ, δ, q, F )
{ private:
//----- reprezentare
    char Q[MAX];          // Multimea starilor
    char S[MAX];          // Alfabetul
    char qo;              // Starea initiala
    char F[MAX];          // Multimea starilor finale
    char delta[MAX][MAX]; // Functia δ: Q x Σ → P(Q)
//----- comportament privat
    bool Este_Stare(char c);
    bool Este_Finala(char);
//----- comportament public
public:
    void Citeste_Stari (void);
    void Citeste_Alfabet(void);
    void Citeste_qo (void);
    void Citeste_Finale (void);

```

```

    void Citeste_Delta    (void);
    void Afis_Automat    (void);
    bool Include          (char*);
    void Verifica         (void);
    void Tabel            (void);
};
inline void automat::Citeste_Stari (void)
{ cout<<"Q="; cin>>Q; }

void automat::Citeste_Alfabet(void)
{ for(;;)
    { bool temp=True;
      cout<<"Σ="; cin>>S;
      for(int i=0;i<strlen(S);i++)
        { if (Este_Stare(S[i]))
          { cout<<"simbolul "<<S[i]<<" este stare\n";
            temp=False;
            break;
          } //endif
        } // endfor
        if (temp) break;
      } // endfor
    }

void automat::Citeste_go (void)
{ for(;;)
    { cout<<"qo="; cin>>qo;
      if (Este_Stare(qo)) break;
      else
        {cout<<"n-ai dat o stare din Q\n";
          continue;
        } // endif
      } // endfor
    }

void automat::Citeste_Finale (void)
{ for(;;)
    { bool temp=True;
      cout<<"F="; cin>>F;
      for(int i=0;i<strlen(F);i++)
        { if (!Este_Stare(F[i]))
          { cout<<"simbolul "<<F[i]<<" nu este stare\n";
            temp=False;
            break;
          } //endif
        } // endfor
        if (temp) break;
      } // endfor
    }

void automat::Citeste_Delta(void)
{ int l,c;
  for(l=0;l<strlen(Q);l++)
    for(c=0;c<strlen(S);c++)
      { cout<<"δ("<<Q[l]<<","<< S[c]<<")=";
        cin>>delta[l][c];
      }
}

```

```

bool automat::Este_Stare(char c)
{ return strchr(Q,c) ? True : False; }

bool automat::Este_Finala(char c)
{ return strchr(F,c) ? True : False; }

bool automat::Include(char *sec)
{ int i;
  char l=go, rez;
  for(i=0; i<strlen(sec); i++)
  { rez=delta[strchr(Q,l)-Q][strchr(S, sec[i])-S];
    if(rez=='.') { cout<<"Secventa eronata...\n"; return False; }
    l=rez;
  }
  return Este_Finala(rez) ? True : False;
}

void automat::Verifica(void)
{ char s[MAX];
  cout<<"\n\nSecventa W = "; cin>>s;
  if(Include(s)) cout<<"Secventa "<<s<<" apartine T(M).";
  else cout<<"Secventa "<<s<<" nu apartine T(M).";
}

void automat::Tabel(void)
{ int i,j;
  cout<<"\n\n";
  cout<<"      r";
  for(i=0; i<strlen(S); i++) cout<<"——";
  cout<<"\n";
  cout<<"  δ |";
  for(i=0; i<strlen(S); i++) cout<<" "<<S[i]<<" ";
  cout<<"|\n";
  cout<<"┌——┐";
  for(i=0; i<strlen(S); i++) cout<<"——";
  cout<<"└|\n";
  for(i=0; i<strlen(Q); i++)
  { cout<<"| "<<Q[i]<<" |";
    for(j=0; j<strlen(S); j++) cout<<" "<<delta[i][j]<<" ";
    cout<<"|\n";
  }
  cout<<"┌——┐";
  for(i=0; i<strlen(S); i++) cout<<"——";
  cout<<"└|\n";
}

void automat::Afis_Automat (void)
{ cout<<"\nM=( Q={ "<<Q<<" } "<<" , ";
  cout<<" Σ={ "<<S<<" }, δ, "<<go<<" , ";
  cout<<" F={ "<<F<<" } )\n";
}

void main(void)
{ char opt;
  automat M;
  clrscr();
  M.Citeste_Stari();
  M.Citeste_Alfabet();
  M.Citeste_go();
  M.Citeste_Finala();
  M.Citeste_Delta();
}

```



```

do
{ clrscr();
  M.Afis_Automat();
  cout<<"\n";
  cout<<"\n V...Verificare acceptare secventa ";
  cout<<"\n T...Tabel δ ";
  cout<<"\n";
  cout<<"\n E...Esire ";
  cout<<"\n";
  cout<<"\nAlegeti: ";
  switch(opt=getche())
  { case 'V': M.Verifica(); break;
    case 'T': M.Tabel();
  }
  getch();
}
while(opt!='E');
cout<<"\n La Revedere...\n";
}

```

Programul anterior este scris în spiritul programării prin abstractizarea datelor și este mult mai clar și lizibil.

2.3.

Reprezentarea și simularea funcționării unui automat finit nedeterminist.

Rezolvarea consta în aplicarea nerecursivă a metodei backtracking.

```

#include <stdio.h>
#include <string.h>
#include <alloc.h>
#include <conio.h>
#include <ctype.h>
struct {
    char stare,simb; // starea si simbolul curent
    char *var; // toate variantele - stările în care
                // se poate ajunge din starea curentă
                // cu simbolul curent;
                // la început stările sunt simbolizate
                // cu litere mici, dar în momentul în
                // care se alege o variantă simbolul
                // pentru stare se transformă în literă mare
    } stiva[100]; // stiva este folosită pentru simularea
                // nerecursivă a metodei backtracking

char *delta[10][10]; // funcția de tranziție a automatului
                    // delta[i][j] conține mulțimea stărilor în
                    // care se poate ajunge din starea i cu
                    // simbolul j și este un șir de caractere de
                    // lungime variabilă pentru care se alocă
                    // dinamic zonă de memorie

char Q[15],S[10],F[15]; // Q = multimea starilor automatului
                        // (litere mici)
                        // S = multimea simbolurilor alfabetului
                        // de intrare
                        // F = multimea starilor finale ale
                        // automatului

```

```

int este(char x,char M[])          // verifica daca caracterul x se
                                   // afla în mulțimea M
{ int i;
  for(i=0;M[i];i++) if(M[i]==x) return i;
  return(-1);
}

void citeste(char M[],char x)
{ char *mesaj;
  if(x=='s')
    mesaj="simbolurilor";
  else
    if(x=='q')
      mesaj="starilor \n (litere mici,prima stare este starea\n initiala)";
    else mesaj="starilor finale";
  printf("\nDati multimea %s: ", mesaj);
  gets(M);
}

void citdelta()
{ int i,j;
  char x[10];
  printf("\nfunctia delta (daca nu e definita pentru o stare si");
  printf("\n un simbol tastati ENTER)");
  for(i=0;Q[i];i++)
    for(j=0;S[j];j++)
      {
        printf("\n delta[%c][%c]=",Q[i],S[j]);
        gets(x);
        if(strlen(x)==0) delta[i][j]=NULL;
        else
          {delta[i][j]=(char *) malloc(1+strlen(x));
            strcpy(delta[i][j],x);}
      }
}

void init(char stare,char simb,int i)          // initializeaza nivelul I
                                                // al stivei cu stare,
{                                                // simbol si variantele
                                                // corespunzatoare

  int j,k;
  stiva[i].stare=stare;
  stiva[i].simb=simb;
  j=este(stare,Q);
  if(simb==' '){ stiva[i].var=NULL;return;}
  k=este(simb,S);
  if(!delta[j][k])
    stiva[i].var=NULL;
  else
    { stiva[i].var=(char*)malloc(1+strlen(delta[j][k]));
      strcpy(stiva[i].var,delta[j][k]);
    }
}

```

```

void main(void)
{char secv[50],*p;
 int n,j,i=0;
 clrscr();
 citeste(Q,'q'); // citire mulțime stări
 citeste(S,'s'); // citire multime simboluri
 do{
  b: citeste(F,'f');// citeste multimea starilor finale
  for(i=0;i<strlen(F);i++)
   if(este(F[i],Q)==-1)
    {printf("\n!! stari eronate , reluati !!\n");
     goto b;
    }
  break;
 }
 while(1);
 citdelta(); // citire funcție de tranziție
 printf("\nDoriti sa verificati secventa?(d/n):");
 while(tolower(getche())!='n')
 { clrscr();
  printf("\nDati secventa (pentru secventa vida tastati\nENTER):");
  gets(secv);
  n=strlen(secv);
  if(n==0)
   if( este(Q[0],F)!=-1)
    { printf("\n!!!! secventa vida este acceptata deoarece");
      printf("\nstarea initiala este si finala !!!!!\n");
      printf("( %c,)",Q[0]);
    }
    else printf("\n!!!! secventa vida nu este acceptata");
      printf("\nstarea initiala nu e si finala");
  else
   {init(Q[0],secv[0],0);
    i=0;
    while(i>=0)
     { int q=0;
      if(stiva[i].simb==' ' && (q=este(stiva[i].stare,F))!=-1)
       {printf("\n!!!! secventa acceptata !!!!!\n");
        for(j=0;j<=i;j++) printf("( %c, %s) |--",stiva[j].stare,secv[j]);
        printf("\b\b\b "); break;
       }
      else
       {if(q==0) p=stiva[i].var;
        else p=stiva[--i].var;
        while(p && *p && isupper(*p))p++;
        if(!p || !*p) i--;
        else
         { *p=toupper(*p);
          i++;
          if(i==n) // introducerea in stiva a simbolului ' ' are
                   // semnificatia de sfarsit secventa de analizat
            init(tolower(*p),' ',i);
          else init(tolower(*p),secv[i],i);
         }
        }
      }
    }
   if(i<0) printf(" \n!!! secventa neacceptata !!!!!");
 }
 printf("\n\nDoriti sa verificati secventa?(d/n):");
 }
 }

```

2.13.

Verificarea dacă o secvență este acceptată sau nu de un automat push-down nedeterminist oricare ar fi criteriul de acceptare ales: criteriul stivei vide sau cel al stării finale.

Rezolvarea constă în implementarea metodei backtracking recursiv.

Programul este următorul:

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <alloc.h>
#include <ctype.h>

typedef struct
{
    char st;          // starea în care trece automatul
    char *sir;        // sirul de simboluri cu care se înlocuieste
                      // simbolul din varful stivei
} tip_delta;         // o vom numi varianta

tip_delta *delta[15][10][10]; // functia de tranzitie
                              // deoarece automatul push-down
                              // este nedeterminist fiecare
                              // element al tabloului delta
                              // este o adresa la care se vor
                              // memora dinamic
                              // variantele posibile

char Q[15];           // multimea starilor automatului
char S[10];           // multimea simbolurilor alfabetului de intrare
char Z[10];           // multimea simbolurilor alfabetului stivei
char F[15];           // multimea starilor finale ale automatului;
                      // este folosita doar daca criteriul de
                      // acceptare
                      // este cel al starii finale
char secv[50];        // secventa citita de pe banda de intrare
char crt;              // criteriul de acceptare
                      // crt='v' ( stiva vida)
                      // crt='f' ( stare finala)
char stiva[50];        // memoria stiva
int este(char x,char M[])
{
    // returneaza pozitia simbolului x in multimea M,
    // sau -1 daca x nu apartine lui M
    int i;
    for(i=0;M[i];i++)
        if(M[i]==x) return i;
    return(-1);
}
```

```

void citeste(char M[],char x)
{
    char *mesaj;
    switch(x)
    {
        case 's':
            mesaj="simbolurilor alfabetului de intrare";
            break;
        case 'q':
            mesaj="starilor (prima stare este starea initiala)\n";
            break;
        case 'f':
            mesaj="starilor finale";
            break;
        case 'p':
            mesaj="simbolurilor alfabetului memoriei stiva:";
            strcat(mesaj,"\n(primul simbol este simbolul initial al stivei)");
    }
    printf("\nDati multimea %s: ", mesaj);
    gets(M);
}

void citdelta()
{
    int i,j,k,l;
    tip_delta *pd;
    char sir1[50],*s,*s1;
    printf("\n exemplu pentru furnizarea functiei delta\n");
    printf("   delta[p,0,Z]=(p,&),(q,AA),(p,AZ)   \n\n");
    for(i=0;Q[i];i++)
        for(j=0;S[j];j++)
            for(k=0;Z[k];k++)
                {
                    printf("\n delta[%c,%c,%c]=",Q[i],S[j],Z[k]);
                    fflush(stdin);
                    gets(sir1);
                    if(strlen(sir1)==0)          // nu e definita functia pentru
                                                // tripletul Q[i],S[j],Z[k]
                        {delta[i][j][k]=NULL; continue;}
                    s=sir1;
                    l=0; //se numara variantele citite si se aloca dinamic zona necesara
                        //pentru memorarea lor ultima varianta introdusa va fi o
                        //varianta fictiva care va contine componenta sir=NULL si este
                        //utilizata pentru a depista sfarsitul variantelor
                    while(s=strchr(s,'('))
                        {l++;s++;}
                    pd=delta[i][j][k]=(tip_delta*)malloc((l+1)*sizeof(tip_delta));
                    s=sir1;
                    while(s=strchr(s,'('))
                        {pd->st=*(s+1);
                            s1=s+3;
                            s=strchr(s1,'('));
                            *s='\0';
                            pd->sir=(char *)malloc(strlen(s1)+1);
                            strcpy(pd->sir,s1);
                            s=s+1;
                            pd++;
                        }
                    pd->sir=NULL; // completarea variantei fictive
                }
}

```

```

int push(char *x,int vs)
{
    // inlocuieste simbolul din varful stivei cu
    // sirul de caractere de la adresa x
    // se actualizeaza varful stivei : vs

    char *p;
    vs--;
    if(*x!='&')
        { for(p=x;*p;p++);
          for(p--;p>=x;p--) stiva[vs++]=*p;
        }
    return vs;
}

int accept(char q,int i,int vs)
{
    // functia returneaza valoarea 1 daca secventa este acceptata
    // de automat si -1 in caz contrar
    // se modeleaza metoda backtracking recursiv pentru gasirea
    // solutiei
    char simb[2];
    int j,n,vf,k,l,m;
    tip_delta *p;
    n=0;
    if(crt=='v') // criteriul stivei vide
        {if(secv[i]=='\0')
          if(vs==0) return(1);
          else n=1;
        }
    else // criteriul starii finale
        {if(secv[i]=='\0')
          if(este(q,F)!=-1) return(1);
          else n=1;
        }
    simb[0]=secv[i];
    simb[1]='&';
    vf=vs;
    for(j=n;j<2;j++)
        {
            // se incerca cele doua posibilitati:
            // -daca simb[j]=secv[i] se incerca avansarea pe banda
            // de intrare cu simbolul secv[i]
            // -daca simb[j]='&' se incerca o epsilon tranzitie
            if((k=este(q,Q))===-1)break;
            if((l=este(simb[j],S))===-1) {j++;continue;}
            if(vf<1) break;
            if((m=este(stiva[vf-1],Z))===-1) break;
            p=delta[k][l][m];
            for(;p && p->sir;p++) // se parcurg toate variantele
                { vs=vf
                  vs=push(p->sir,vs);
                  if(accept(p->st,i+1-j,vs)!=-1) return(1);
                }
        }
    return(-1);
}

void main(void)
{
    int ind,i,j,k;
    int vs; // virf stiva ( vs=0 indica stiva vida )
    char q;
    clrscr();
    printf("\nSimularea functionarii unui automat push-down\n nedeterminist");
    citeste(Q,'q'); // citire stari automat
}

```

```

citeste(S,'s');          // citire alfabet de intrare
strcat(S,"&");           // se adauga la alfabetul de intrare epsilon
                           // simbolizat prin &
citeste(Z,'p');          // citire alfabet memorie stiva

printf("\n Dati criteriul de acceptare");
printf("\n stiva vida(v) ,stare finala (f) :");
if((crt=tolower(getche()))=='f')
    citeste(F,'f');      // citire stari finale
printf(" \n\n Functia de tranzitie delta:");
printf("\n (simbolul & reprezinta epsilon)");
printf("\n(daca delta nu e definita pentru un triplet\n tastati ENTER)\n");
citdelta();
printf("\n\n Doriti sa verificati secventa?(d/n):");
while(tolower(getche())!='n')
{printf("\n Dati secventa (pt secventa vida dati ENTER):\n");
 gets(secv);           // citirea secventei de pe banda de intrare
 stiva[0]=Z[0];        // initializarea stivei cu simbolul initial al
                       // stivei
 vs=1;                 // arata pozitia pe care se poate adauga un nou
                       // element in stiva;
                       // vs=0 indica stiva vida elementul din varful
                       // stivei este stiva[vs-1]
 if(accept(Q[0],0,vs)!=-1)
     printf("\n  secventa este acceptata");
 else
     printf("\n  secventa nu este acceptata");
 printf("\n\n Doriti sa verificati secventa?(d/n):");
}
}

```

BIBLIOGRAFIE

1. AHO,A.V., ULLMAN,J.D. "The Theory of Parsing, Translation, and Compiling",
Vol.1. Englewood Cliffs, N.J.:Prentice-Hall, 1972.
2. AHO,A.V., ULLMAN,J.D. "The Theory of Parsing, Translation, and Compiling",
Vol.2. Englewood Cliffs, N.J.:Prentice-Hall, 1973.
3. AHO,A.V., ULLMAN,J.D. "Principles of Compiler Desing"
Reading, Mass.: Addison-Wesley, 1977.
4. AHO,A.V., ULLMAN,J.D. "Concepts fondamentaux de l'informatique"
DUNOD, Paris, 1993 (Traducere din engleză. Carte apărută în 1992).
5. ALEKSANDER,I., HANNA F.K. "Automata Theory: An Engineering Approach",
New York : Crane Russak, 1975.
6. ATANASIU,A. "Bazele informaticii"
Universitatea din București, 1987.
7. BULAC,C. "Inițiere în Turbo C++ și Borland C"
Editura Teora, București, 1995.
8. GERSTING,J.L. "Mathematical Structures In Computer Science"
New York: W.M. Freeman and Company, 1987.
9. GINSBURG, S. "The Mathematical Theory of Context-free Languages"
New York: Mc.Graw-Hill, 1966.
10. GRIES, D. "Compiler Construction for Digital Computers"
New York: Wiley, 1971.
11. HOPCROFT,J.E., ULLMAN,J.D. "Introduction to Automata Theory, Languages,
and Computation"
Reading, Mass.: Addison-Wesley, 1979.

12. JOHNSONBAUGH,R. "Discrete Mathematics"
New York: Macmillan Publ. Co., 1990.
13. KERNIGHAN,B.W., RITCHIE,D.M. "The C programming languages"
Englewood. Cliffs, N.J.: Prentice-Hall, 1978.
14. LIVOVSCI,L. POPOVICI,C.P., GEORGESCU,M., ȚĂNDĂREANU,N.
"Bazele Informaticii"
Editura Didactică și Pedagogică, București, 1981.
15. McNAUGHTON,R. "Elementary Computability, Formal Languages,
and Automata"
Englewood. Cliffs, N.J.: Prentice-Hall, 1982.
16. MARCUȘ,S. "Gramatici și automate finite"
Editura Academiei RSR, București, 1964.
17. MOLDOVAN,Gr., CIOBAN,V., LUPEA,M.,
"Probleme pentru programare în limbajul C"
Universitatea "Babeș-Bolyai" Cluj-Napoca, 1995.
18. MOLDOVAN,Gr. "Bazele Informaticii II"
Universitatea "Babeș-Bolyai" Cluj-Napoca, 1985.
19. ORMAN,G. "Limbaje formale"
Editura Tehnică, București, 1982.
20. PAUN,Gh. "Gramatici contextuale"
Editura Academiei RSR, București, 1982.
21. SALOMAA,A. "Theory of Automata"
Oxford: Pergamon Press, 1969.
22. SALOMAA,A. "Formal Languages "
New York: Academic Press, 1993.

23. ȘERBĂNAȚI, L.D. "Limbaje de programare și compilatoare"
Editura Academiei RSR, București, 1987.

24. TOADERE, T. "Teoria grafelor"
Universitatea "Babeș-Bolyai" Cluj-Napoca, 1992.