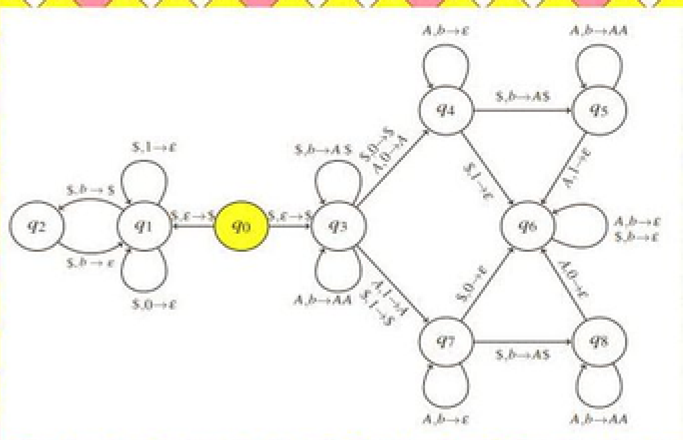
CONSTANTIN CIUBOTARU

LIMBAJE FORMALE ȘI AUTOMATE

AUTOMATE CU MEMORIE STIVĂ



Chișinău 2019

Universitatea de Stat "Dimitrie Cantemir"

Institutul de Matematică și Informatică "Vladimir Andrunachievici"

Constantin Ciubotaru

LIMBAJE FORMALE ȘI AUTOMATE

AUTOMATE CU MEMORIE STIVĂ

Chișinău 2019

CZU _ _ _
N _ _ _

În manual sunt incluse practic toate aspectele automatelor cu memorie stivă: funcționarea, diverse variante (cu acceptare prin stări finale, prin stivă vidă, deterministe), echivalența lor cu gramaticile independente de context, probleme și exerciții însoțite de soluții și explicații.

Manualul este destinat studenților, masteranzilor și doctoranzilor specialităților informatice, tuturor persoanelor interesate de bazele teoretice ale informaticii.

Aprobat la şedinţa departamentului "Chimie, Matematică și Informatică" din 13.04.2019, proces-verbal nr. 6

Recomandat pentru editare de Senatul Universităţii de Stat "Dimitrie Cantemir" și Consiliul științific al Institutului de Matematică și Informatică "Vladimir Andrunachievici".

Recenzent: dr. hab. în inf., m. c. AȘM Svetlana Cojocaru

Descrierea CIP a Camerei Naționale a Cărții

Ciubotaru Constantin.
Limbaje formale și automate. Automate cu memorie stivă/Constantin Ciubotaru; Universitatea de Stat "Dimitrie Cantemir", Institutul de Matematică și Informatică "Vladimir Andrunachievici". – Chișinău, 2019 (Tipogr. „Biotehdesign").
– 107 p.: fig., tab.
Bibliogr.: p. 99-100 (22 tit.). – 50 ex.
ISBN

Cuprins

Prefață

Intuitiv automatele cu memorie stivă (AS) reprezintă automate finite (de obicei cu ε-tranziții) extinse cu memorie de tip stivă. AS constituie un compartiment important al teoriei limbajelor formale și automatelor fiind echivalente cu gramaticile independente de context, care reprezintă modelul sintactic al limbajelor de programare.

Stiva este o structură de date frecvent utilizată în informatică. Să menționăm doar câteva domenii mai importante: funcții recursive, algoritmi de sortare, parcurgere și căutare, compilatoare, editoare de text, motoare de căutare, urmărirea și restabilirea versiunilor produselor program etc.

Stiva stă la baza realizării structurilor ierarhice ale limbajelor de programare, de exemplu, structuri de genul:

“*begin*“ ... “*begin*“ ... “*end*“ ... “*end*“
“*if*“ ... “*then*“ ... “*if*“ ... “*then*“ ... “*else*“ ... “*else*“
“(“ ... “(“ ... “)“ ... “)“ etc.

Un rol cu totul deosebit se atribuie AS deterministe și limbajelor deterministe, care constituie nucleul analizoarelor sintactice ale limbajelor de programare, fază obligatorie a oricărui compilator.

În lucrarea [18] D.Knuth introduce gramaticile $LR(k)$ și demonstrează echivalența acestor gramatici și a AS deterministe. Această clasă de gramatici și limbaje a contribuit esențial la automatizarea construirii compilatoarelor.

Această carte apare ca rezultat al cursurilor “Limbaje formale și automate“ și “Proiectarea compilatoarelor“ predate pe parcursul mai multor ani la Universitatea de Stat din Moldova, Universitatea Tehnică a Moldovei, Universitatea de Stat “Dimitrie Cantemir“. Având programe de studii diferite, dar și torente de studenți cu pregătire diferită, de fiecare dată apărea necesitatea adaptării cursului. Acest fapt a influențat și expunerea materialului. Vom întâlni explicații și exemple la un nivel intuitiv, simplu, dar și definiții formale, algoritmi, demonstrări de teoreme. Aceasta sugerează și un anumit algoritm de studiere a materialului. La prima lectură va fi suficient să se studieze doar partea intuitivă ce include definiții, algoritmi și exemple, lasând pentru un studiu mai aprofundat partea formală, inclusiv demonstrările.

Sunt incluse, practic, toate aspectele AS: funcționarea AS, variante de AS, echivalența AS și a gramaticilor independente de context, AS deterministe. De asemenea sunt incluse probleme și exerciții. Deoarece toate problemele și exercițiile sunt însoțite de soluții și explicații, ele ar putea fi privite ca parte componentă a materialului expus. Din aceste considerente este recomandabil să li se acorde atenție sporită.

Pentru orele practice (de laborator) sunt incluse două lucrări, fiecare conținând 25 de probleme, aceasta fiind comod dacă se lucrează cu grupe de studenți. Pentru fiecare tip de probleme practice găsim exemple similare în compartimentele respective, dar și indicații.

Vine să consolideze materialul expus și capitolul consacrat programării automatelor cu memorie stivă. Programarea algoritmilor se face în limbajul Common Lisp, limbaj al programării funcționale și al calculului simbolic. Aceasta facilitează înțelegerea programelor, dar poate fi și un bun exercițiu pentru cei care vor să se familiarizeze cu paradigma programării funcționale și cu limbajul Common Lisp.

1. Modelul stivei

Să precizăm mai întâi modelul și modul de funcționare a stivei acceptate la definirea AS. În general stiva reprezintă un mod de organizare a informației după principiul "ultimul-sosit-primul-plecat" (în engleză LIFO, Last-In-First-Out). În continuare vom reprezenta stiva ca o bandă de memorie infinită intr-o direcție și împărțită în celule (căsuțe), în fiecare celulă fiind posibilă înregistrarea unui singur simbol din $\Gamma \cup \{\varepsilon\}$, unde Γ este vocabularl stivei, $\Gamma \neq \varnothing$, $\varepsilon \notin \Gamma$. Prin ε vom nota simbolul (șirul) vid. Acces avem doar la prima poziție care se numește *topul stivei*.

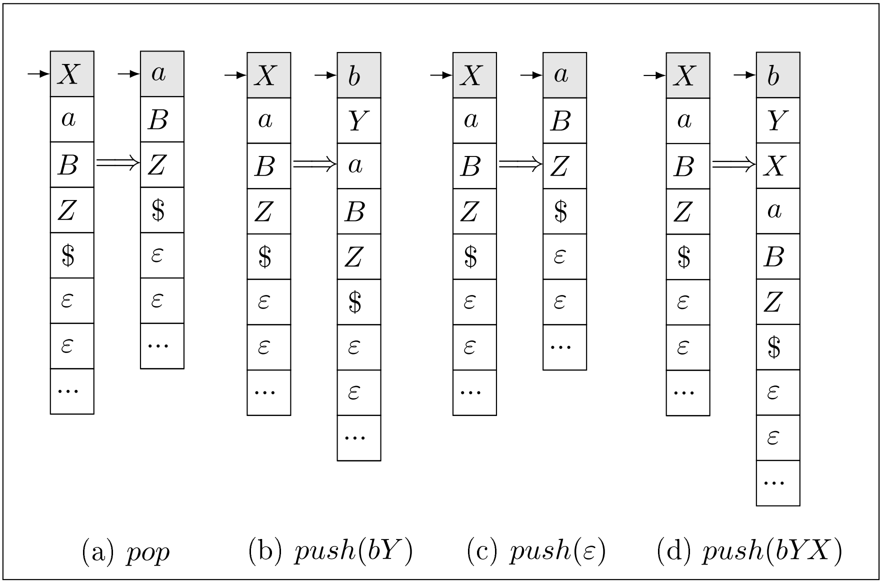


Figura 1: Funcționarea stivei

Anume elementul din topul stivei este elementul curent vizibil care poate fi extras din stivă. Există mai multe moduri de interpretare a funcționării stivei.

Se definesc două tipuri de operații asupra stivei: *pop* și *push*.

- *pop* - extrage elementul din top, toate celelalte elemente fiind deplasate cu o poziție mai aproape de top.
- $push(\gamma)$, $\gamma \in \Gamma^*$, $\gamma = Z_1 Z_2 \ldots Z_n$ - șterge elementul din top, deplasează toată informația din stivă cu n poziții mai

jos (mai la dreapta) de top și înregistrează $Z_1 Z_2 \ldots Z_n$ pe pozițiile eliberate.

Observăm că operația *push* întotdeauna șterge elementul din top. Dacă acest element, fie X, este necesar în continuare atunci el trebuie concatenat la γ, adică trebuie de efectuat $push(\gamma X)$. Este comod să considerăm că Γ conține un *element evidențiat*, fie \$, care tot timpul va fi ultimul element al stivei și toate elementele ce urmează după \$ sunt ε. Aceasta ne permite să urmărim evoluția stivei. În mod special ne va interesa situația când stiva devine vidă.

În Figura 1 aducem câteva exemple. Vom desena stiva pe verticală. O situație interesantă apare la efectuarea operației $push(\varepsilon)$. În acest caz elementul din top se șterge, iar în locul lui se înscrie ε. Așa cum ε reprezintă simbolul vid, tot conținutul stivei se va deplasa cu o poziție în sus. Astfel, $pop = push(\varepsilon)$ de unde rezultă că toate operațiile cu stiva pot fi efectuate doar cu ajutorul instrucțiunilor *push*.

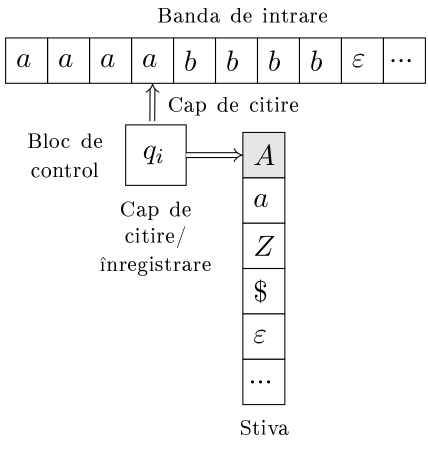


Figura 2: Schema automatului cu memorie stivă.

Pentru a defini un AS vom extinde automatul finit (AF) cu o stivă, un cap de citire/înregistrare, conectat la topul stivei și la blocul de control și un vocabular pentru stivă. Desigur, se va modifica și funcția de tranziție. Dacă în cazul AF funcția de tranziție era definită ca o aplicație $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$, atunci funcția de tranziție a AS va fi mai complicată. Orice tranziție a AS, de rând cu acțiunile proprii AF, trebuie să țină cont de elementul din topul stivei și să definească o acțiune *push* potrivită situației. Schema AS este reprezentată în Figura 2.

2. Funcționarea automatului stivă

AS începe funcționarea având pe bandă șirul care trebuie recunoscut, blocul de control fiind în starea q_0, iar în topul stivei fiind plasat \$. În continuare, în dependență de elementul curent de pe bandă, de starea curentă și de elementul din topul stivei sunt posibile următoarele acțiuni (sunt definite de funcția de tranziție):

- *Acceptare:*
  - citește simbolul de pe bandă,
  - se deplasează cu o poziție la dreapta,
  - își schimbă starea sau rămâne în aceeași stare,
  - înregistrează un șir nou în stivă sau șterge elementul din top, dacă acest șir este ε,
  - dacă se citesc toate simbolurile de pe bandă și automatul trece în una din stările finale, atunci se semnalează acceptarea sau recunoașterea șirului de pe bandă.
- *Impas (Respingere):* se blochează semnalând imposibilitatea recunoașterii șirului de pe bandă.

Menționăm că, ca și în cazul AF, AS poate deplasa capul de citire pentru banda de intrare doar la dreapta, fără reveniri. Astfel, doar acțiunile cu stiva pot extinde puterea de recunoaștere a AS.

Exemplul 2.1 Să explicăm intuitiv acțiunile AS expuse mai

sus pentru limbajul $L_{ij} = \{a^i b^j | i \geq j \geq 1\}$ care nu este un limbaj regulat.

- inițial pe bandă se înregistrează șirul $a^i b^j$, în topul stivei se plasează \$, iar blocul de control se poziționează în starea q_0.
- cât timp de pe bandă se citește simbolul a, automatul va plasa în stivă A rămânând în starea q_0; se va ține cont de faptul că orice înregistrsre în stivă se produce peste elementul din top.
- când se întâlnește pe bandă primul b (un b trebuie să fie obligatoriu, deoarece $j \geq 1$), în topul stivei va fi A. În acestă situație AS citește b, șterge A din topul stivei și trece în starea q_1.
- cât timp în starea q_1 avem b pe bandă și A în top, AS rămâne în starea q_1 și avansează citind b și ștergând A.
- dacă în starea q_1 de pe bandă s-au citit toate simbolurile b (banda a devenit vidă) iar în top avem A sau \$, atunci AS acceptă șirul inițial iar starea q_1 este stare finală.
- toate celelalte situații blochează funcționarea automatului semnalând imposibilitatea recunoașterii șirului.

3. Definiții formale

Definiția 3.1

Vom numi AS obiectul $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$, F)$, unde

- Q - mulțime finită de stări,
- Σ - vocabularul de intrare,
- Γ - vocabularul stivei, $\varepsilon \notin \Gamma$,
- δ - funcția de tranziție, $\delta : Q \times \Gamma \times (\Sigma \cup \{\varepsilon\}) \longrightarrow 2^{Q \times \Gamma^*}$,
- q_0 - starea inițială,
- $\$$ - simbolul evidențiat al stivei, $\$ \in \Gamma$,
- F - mulțimea stărilor finale, $F \in Q$.

Să aducem câteva explicații referitor la funcția de tranziție. Menționăm că funcția δ este o funcție parțial definită. Din definiție urmează că toate tranzițiile au forma

$$\delta(q, Z, a) = \{(q_1, \gamma_1), (q_2, \gamma_2), \dots, (q_n, \gamma_n)\},$$

unde $a \in (\Sigma \cup \{\varepsilon\})$. Dacă $a = \varepsilon$, avem ε-tranziție și automatul va funcționa ignorând banda de intrare. În general, AS funcționează în mod nedeterminist. La orice pas se alege o singură variantă $(q_i, \gamma_i), 1 \leq i \leq n$, pentru care se va efectua tranziția. Așa cum al doilea argument al funcției este $Z \in \Gamma$ și $z \neq \varepsilon$, automatul nu poate funcționa cu stiva vidă. Imediat ce stiva drvine vidă, automatul se va opri. Se poate permite și funcționarea AS cu stiva vidă, adică $\delta : Q \times (\Gamma \cup \{\varepsilon\}) \times (\Sigma \cup \{\varepsilon\}) \longrightarrow 2^{Q \times \Gamma^*}$. Aceasta însă nu modifică puterea de recunoaștere a automatului (paragraful 5).
Încă o observație. Vom folosi notația $\delta(q, Z, a)$, nu $\delta(q, a, Z)$, o formă frecvent utilizată în alte manuale. Argumentarea ar fi următoarea:

a) principala aplicație a AS o constituie analizoarele sintactice, dar atât analizoarele descendente, cât și cele ascendente folosesc de obicei AS fără stări, stiva fiind principalul element în gestionarea analizoarelor; cu alte cuvinte, majoritatea analizoarelor sintactice folosesc $\delta(Z, a)$

b) automatele generate din gramatici, conform algoritmului de convertire (paragraful 6) au o singură stare care ar putea fi omisă.

Exemplul 3.1 Inserăm mai jos definiția formală a automatului care recunoaște limbajul $L_{ij} = \{a^i b^j \mid i \geq j \geq 1\}$ (Figura 3), explicat intuitiv în exemplul 2.1.

$$\begin{gathered} AS = (Q, \Sigma, \Gamma, \delta, q_0, \$, F),\ Q = \{q_0, q_1\}, \\ \Sigma = \{a, b\},\ \Gamma = \{\$, A\},\ F = \{q_1\}, \\ \delta(q_0, \$, a) = \{(q_0, A\$)\}, \quad \delta(q_0, A, a) = \{(q_0, AA)\}, \\ \delta(q_0, A, b) = \{(q_1, \varepsilon)\}, \quad \delta(q_1, A, b) = \{(q_1, \varepsilon)\} \end{gathered}$$

Figura 3: AS pentru limbajul $L_{ij} = \{a^i b^j \mid i \geq j \geq 1\}$

AS poate fi reprezentat grafic, dar, spre deosebire de reprezentarea grafică a automatului finit, diagrama AS nu ne spune nimic despre structura limbajului acceptat. Prezentăm în Figura 4 diagrama AS pentru limbajul $L_{ij} = \{a^i b^j \mid i \geq j \geq 1\}$

Definiția 3.2

Se numește *configurație* a AS obiectul (q, γ, x), unde q este starea curentă, γ - conținutul stivei, x - partea neanalizată a șirului de pe bandă. Configurația $(q_0, \$, x)$, unde x este

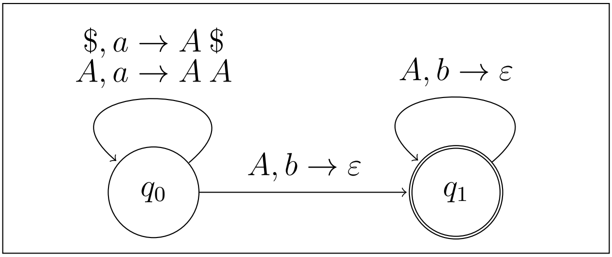


Figura 4: Reprezentarea grafică a AS

șirul inițial, se numește *configurație inițială*, iar configurația (q, γ, ε), unde $q \in F$ - *configurație de acceptare*.

De exemplu, $(q_0, \$, aaaabb)$, $(q_0, A\$, aaabb)$, $(q_1, AAAA\$, bb)$, $(q_1, \$, \varepsilon)$ (Figura 5).

Definiția 3.3

AS *trece direct* din configurația (q_1, γ_1, x_1) în configurația (q_2, γ_2, x_2), dacă:

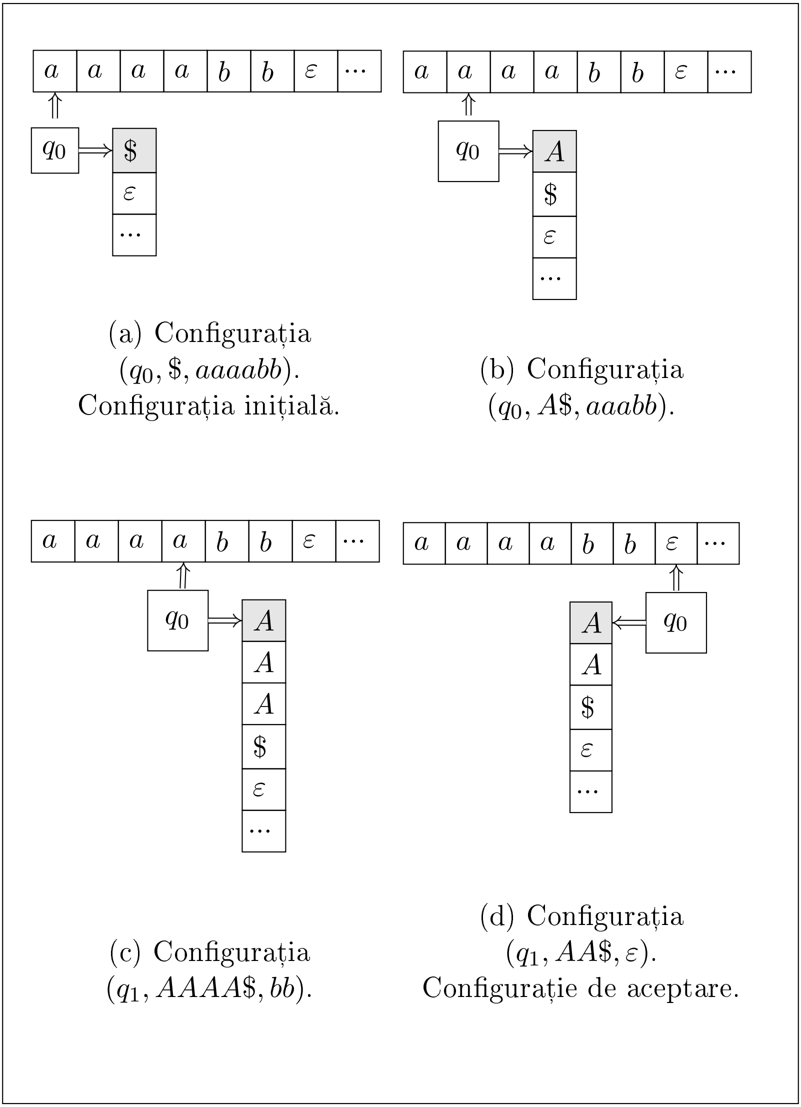
- $\gamma_1 = Z_1\gamma_{11}$, $\gamma_1 \neq \varepsilon$,
- $x_1 = ax_2$, $a \in \Sigma$ sau $a = \varepsilon$,
- $\delta(q_1, Z_1, a) \ni (q_2, \gamma_{22})$,
- $\gamma_2 = \gamma_{22}\gamma_{11}$.

Vom nota această acțiune prin $(q_1, \gamma_1, x_1) \vdash_{AS} (q_2, \gamma_2, x_2)$. Semnul $\vdash_{AS}$ se citește "trece direct" sau "trece la un pas". Dacă nu apar ambiguități în privința automatului, putem utiliza simplu $\vdash$. Expunerea explicită a acțiunii din definiție,

$$(q_1, \gamma_1, x_1) = (q_1, Z_1\gamma_{11}, ax_2) \vdash (q_2, \gamma_{22}\gamma_{11}, x_2) = (q_2, \gamma_2, x_2),$$

ne ajută să o înțelegem mai bine. Dacă $a = \varepsilon$ atunci $x_1 = x_2$ și obținem:

$$(q_1, \gamma_1, x_1) = (q_1, Z_1\gamma_{11}, x_1) \vdash (q_2, \gamma_{22}\gamma_{11}, x_1) = (q_2, \gamma_2, x_1).$$



(a) Configurația $(q_0, \$, aaaabb)$. Configurația inițială.

(b) Configurația $(q_0, A\$, aaabb)$.

(c) Configurația $(q_1, AAAA\$, bb)$.

(d) Configurația $(q_1, AA\$, \varepsilon)$. Configurație de acceptare.

Figura 5: Funcționarea automatului cu memorie stivă

În acest caz automatul efectuează o ε-tranziție modificând stiva și ignorând banda de intrare.

Definiția 3.4

AS trece la n pași ($n \geq 1$) din configurația (q_1, γ_1, x_1) în configurația $(q_{n+1}, \gamma_{n+1}, x_{n+1})$, dacă există (q_2, γ_2, x_2), (q_3, γ_3, x_3), ..., (q_n, γ_n, x_n) și

- $(q_1, \gamma_1, x_1) \vdash (q_2, \gamma_2, x_2)$,
- $(q_2, \gamma_2, x_2) \vdash (q_3, \gamma_3, x_3)$,
  $\ldots$
- $(q_n, \gamma_n, x_n) \vdash (q_{n+1}, \gamma_{n+1}, x_{n+1})$.

Vom nota aceasta prin $(q_1, \gamma_1, x_1) \overset{n}{\vdash} (q_{n+1}, \gamma_{n+1}, x_{n+1})$. Observăm că $(q_i, \gamma_i, x_i) \overset{1}{\vdash} (q_j, \gamma_j, x_j)$ reprezintă trecerea directă a automatului din configurație în configurație (Definiția 3.3), astfel în asemenea situații vom scrie $(q_i, \gamma_i, x_i) \vdash (q_j, \gamma_j, x_j)$.
Pentru simplificare vom nota (q_i, γ_i, x_i) prin c_i. Uneori o să fie comod să scriem

$(q_1, \gamma_1, x_1) \vdash (q_2, \gamma_2, x_2) \vdash \ldots \vdash (q_n, \gamma_n, x_n) \vdash (q_{n+1}, \gamma_{n+1}, x_{n+1})$
sau $c_1 \vdash c_2 \vdash \ldots \vdash c_n \vdash c_{n+1}$.

Cum am putea explica $(q_i, \gamma_i, x_i) \overset{0}{\vdash} (q_j, \gamma_j, x_j)$? Este firesc să se considere $q_i = q_j$, $\gamma_i = \gamma_j$, $x_i = x_j$. În așa mod, pentru "$\overset{0}{\vdash}$" automatul se află în stare de repaus, nu efectuează nici o acțiune. Extindem în continuare acțiunea "$\vdash$" în modul următor:

Definiția 3.5

AS trece din configurația (q_i, γ_i, x_i) în configurația (q_j, γ_j, x_j), vom nota $(q_i, \gamma_i, x_i) \overset{*}{\vdash} (q_j, \gamma_j, x_j)$, dacă există $n \geq 0$ pentru care $(q_i, \gamma_i, x_i) \overset{n}{\vdash} (q_j, \gamma_j, x_j)$.

În multe situații va fi comod să utilizăm notațiile

$$(q_i, \gamma_i, x_i) \overset{\geq n}{\vdash} (q_j, \gamma_j, x_j),\ (q_i, \gamma_i, x_i) \overset{\leq n}{\vdash} (q_j, \gamma_j, x_j)$$

semnificația cărora este evidentă.

Exemplul 3.2

Ne vom referi în continuare la Exemplul 3.1 și Figura 5. Fie $x = aaaabb$ șirul inițial. Configurația inițială va fi $(q_0, \$, aaaabb)$ (Figura 5a). Așa cum $\delta(q_0, \$, a) \ni (q_0, A\$)$, avem trecerea directă $(q_0, \$, aaaabb) \vdash (q_0, A\$, aaabb)$ (Figura 5b). În această situație AS vede simbolul a pe bandă și A în topul stivei. Deoarece $\delta(q_0, A, a) \ni (q_0, AA)$, se efectuează tranziția $(q_0, A\$, aaabb) \vdash (q_0, AA\$, aabb)$. Procesul poate fi continuat. După 4 pași obținem:
$(q_0, \$, aaaabb) \vdash (q_0, A\$, aaabb) \vdash (q_0, AA\$, aabb) \vdash (q_0, AAA\$, abb) \vdash (q_0, AAAA\$, bb)$ (Figura 5c). În acest noment automatul vede b pe bandă și A în topul stivei. Consultând funcția δ, observăm că $\delta(q_0, A, b) \ni (q_1, \varepsilon)$, astfel $(q_0, AAAA\$, bb) \vdash (q_1, AAA\$, b)$. Efectuând încă un pas, obținem: $(q_1, AAA\$, b) \vdash (q_1, AA\$, \varepsilon)$ (Figura 5d). Așa cum în acest moment s-a citit toată banda, iar AS se află în starea q_1 care este stare finală, șirul de pe bandă este acceptat (recunoscut). Combinând toate trazițiile efectuate, obținem: $(q_0, \$, aaaabb) \vdash (q_0, A\$, aaabb) \vdash (q_0, AA\$, aabb) \vdash (q_0, AAA\$, abb) \vdash (q_0, AAAA\$, bb) \vdash (q_1, AAA\$, b) \vdash (q_1, AA\$, \varepsilon)$. Putem scrie, deci, $(q_0, \$, aaaabb) \overset{6}{\vdash} (q_1, AA\$, \varepsilon)$. Dacă numărul de tranziții este irelevant, atunci se poate scrie $(q_0, \$, aaaabb) \overset{*}{\vdash} (q_1, AA\$, \varepsilon)$.

După cum s-a menționat mai devreme, automatele stivă reprezintă modele de recunoaștere a limbajelor formale. Următoarea definiție vine să confirme acest lucru.

Definiția 3.6

Se numește limbaj recunoscut (acceptat) de către $AS = (Q, \Sigma,$

Γ, δ, $q_0, \$, F$), mulțimea
$L(AS) = \{x | x \in \Sigma^*, (q_0, \$, x) \models^* (q_f, \gamma, \varepsilon), q_f \in F\}$

4. Automate cu acceptare prin stivă vidă

Există mai multe variante de automate cu memorie stivă. Am definit mai sus AS care acceptă prin stări finale. Altă variantă reprezintă AS care acceptă prin stivă vidă. Am convenit mai sus că AS nu poate funcționa cu stiva vidă, dar uneori, în mod intenționat, putem goli stiva, iar dacă in acest moment este vidă și banda, putem considera că automatul acceptă șirul de pe bandă. În asemenea situații automatul nu mai are nevoie de stări finale și vom considera $F = \varnothing$. Astfel, putem aduce următoarea definiție:

Definiția 4.1

$AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$ acceptă prin stivă vidă limbajul
$L(AS) = \{x | x \in \Sigma^*, (q_0, \$, x) \models^* (q, \varepsilon, \varepsilon), q \in Q\}$

Exemplul 4.1

Fie x un șir arbitrar peste $V_T = \{1, 0\}$, $x \in V_T^*$. Să notăm prin $n_0(x)$ numărul de zerouri, iar prin $n_1(x)$ - numărul de unități din x. Să examinăm în contunuare limbajul L_{01},

$$L_{01} = \{x | x \in \{0, 1\}^*, n_0(x) = n_1(x)\}.$$

Să observăm din start că $\varepsilon \in L_{01}$, deoarece $n_0(\varepsilon) = n_1(\varepsilon)\} = 0$. Dacă $x_1, x_2 \in L_{01}$, atunci și $0x_11, 1x_10, 0x_21, 1x_20, x_1x_2, x_2x_1 \in L_{01}$.

$$AS_{01} = (Q, \Sigma, \Gamma, \delta, q_0, \$),\ Q = \{q_0, q_1\},\ \Sigma = \{0,1\},$$
$$\Gamma = \{0,1,\$\},\ V_T = \{a,b\},\ \delta = \{$$
$$\delta(q_0,\$,\varepsilon) = \{(q_1,\$)\},\ \ \delta(q_1,\$,0) = \{(q_1,0\$)\},\ \delta(q_1,\$,1) = \{(q_1,1\$)\},$$
$$\delta(q_1,1,1) = \{(q_1,11)\},\ \delta(q_1,0,0) = \{(q_1,00)\},\ \delta(q_1,0,1) = \{(q_1,\varepsilon)\},$$
$$\delta(q_1,1,0) = \{(q_1,\varepsilon)\},\quad \delta(q_1,\$,\varepsilon) = \{(q_1,\varepsilon)\}$$

Figura 6: AS pentru limbajul L_{01}.

$$(q_0,\$,101010) \vdash (q_1,\$,101010) \begin{cases} (q_1,\varepsilon,101010) \text{ - i} \\ (q_1,1\$,01010) \vdash (q_1,\$,1010) \stackrel{0}{\vdash} \end{cases}$$

$$(q_1,\$,1010) \begin{cases} (q_1,\varepsilon,1010) \text{ - i} \\ (q_1,1\$,010) \vdash (q_1,\$,10) \begin{cases} (q_1,\varepsilon,10) \text{ - i} \\ (q_1,1\$,0) \stackrel{*}{\vdash} (q_1,\varepsilon,\varepsilon) \text{ - a} \end{cases} \end{cases}$$

Figura 7: AS_{01}. Recunoașterea șirului 101010.

Să construim AS care acceptă acest limbaj prin stivă vidă. Ideea automatului este simplă: dacă simbolul curent de pe bandă este 0 (sau 1) și simbolul din topul stivei este 0 (sau 1), atunci simbolul de pe bandă se citește, iar în stivă se înregistrează 00 (sau 11). Dacă, însă, pe bandă avem 0 (sau 1), iar în topul stivei 1 (sau 0), atunci se citește simbolul de pe bandă și se șterge simbolul din topul stivei. Dacă șirul inițial conține un număr egal de 0 și 1, atunci acest șir se va citi, iar în topul stivei va rămâne \$. În această situație putem șterge \$ din top și accepta șirul. În caz contrar automatul se va bloca și va respinge șirul.

Prezentăm în Figura 6 acest automat. Vom nota prin "i" - situațiile de impas, iar prin "a" - acceptarea. Explicațiile de mai sus ne permit să afirmăm că $L_{01} = L(AS_{01})$.

Tranziția $\delta(q_1, \$, \varepsilon) = \{(q_1, \varepsilon)\}$ șterge \$ din topul stivei, stiva devenind astfel vidă. Această ε-tranziție condiționează comportamentul nedeterminist al automatului și va fi ultima acțiune la recunoașterea șirurilor corecte, inclusiv ε. Secvența de acceptare pentru șirul $x = \varepsilon$ va fi: $(q_0, \$, \varepsilon) \vdash (q_1, \$, \varepsilon) \vdash (q_1, \varepsilon, \varepsilon)$.

În Figura 7 prezentăm tranzițiile automatului AS_{01} la recunoașterea șirului 101010. Configurațiile de impas $(q_1, \varepsilon, 101010)$, $(q_1, \varepsilon, 1010)$, $(q_1, \varepsilon, 10)$ se datorează faptului că șirul 101100 conține prefixele ε, 10, 1010 care de asemenea aparțin limbajului L_{01}.

Exemplul 4.2

Palindromul este un șir (cuvânt, frază), care se citește la fel de la stânga la dreapta și de la dreapta la stânga. De exemplu: $ABBA$, *abbabbabba*, *cojoc*, *elefaccafele* (ele fac cafele). Să examinăm în continuare palindroamele de lungime pară peste vocabularul $\{a, b\}$. Vom construi un AS care recunoaște aceste palindroame prin stivă vidă. O proprietate interesantă a palindromului este operația de reversare sau oglindire. De exemplu, șirul $x = abbabbabba$ poate fi reprezentat ca concatenarea a două șiruri: $x_1 = abbab$ și $x_2 = babba$, $x = x_1x_2$. Șirul x_2 ese reversul sau imaginea oglindită a șirului x_1. Vom nota aceasta prin $x_2 = x_1^R$. Putem considera că intre simbolurile centrale *bb* se plasează oglinda, adică $abbab|babba$. Această oglindă ne poate ajuta la recunoașterea palindroamelor. AS înregistrează x_1 în stvă (partea șirului până la oglindă), apoi verifică stiva cu șirul rămas pe bandă. Dacă aceste șiruri coincid, AS acceptă șirul, în caz contrar îl respinge. Problema dificilă care apare constă în determinarea poziției oglinzii. De exemplu, șirul *abba* este palindrom și prefix al șirului x. Așa cum automatul nu cunoaște lungimea șirului x, va trebui să presupună că poziția oglinzii poate fi între acești *bb*, adică $ab|ba$. Faptul că această oglindă este falsă va fi depistat după citirea șirului *abba*, automatul mai având simboluri necitite pe bandă. Astfel, automatul trebuie să verifice în mod nedeterminist toate oglinzile posibile, acceptând oglinda reală, dacă ea există, și respingând oglinzile false. Poziția de oglindă poate fi între orice două simboluri vecune *aa* sau *bb*. Pentru șirul x sunt posibile următoarele oglinzi: $ab|bab|bab|ba$, două fiind false. Urmând aceste explicații prezentăm mai jos (Figura 8) AS care acceptă prin stivă vidă limbajul palindroamlor peste $\{a,b\}$; îl vom nota prin L_R, $L_R = \{xx^R | x \in \{a,b\}^*, x \neq \varepsilon\}$. Observăm că tranzițiile

$$AS_R = (Q, \Sigma, \Gamma, \delta, q_0, \$),\ Q = \{q_0, q_1\},\ \Sigma = \{a, b\},\ \Gamma = \{\$, a, b\},$$
$$\begin{array}{ll} \delta(q_0, \$, a) = \{(q_0, a\$)\}, & \delta(q_0, \$, b) = \{(q_0, b\$)\}, \\ \delta(q_0, a, b) = \{(q_0, ba\}, & \delta(q_0, a, a) = \{(q_0, aa), (q_1, \varepsilon)\}, \\ \delta(q_0, b, a) = \{(q_0, ab)\}, & \delta(q_0, b, b) = \{(q_0, bb), (q_1, \varepsilon)\}, \\ \delta(q_1, a, a) = \{(q_1, \varepsilon)\}, & \delta(q_1, b, b) = \{(q_1, \varepsilon)\}, \\ \delta(q_1, \$, \varepsilon) = \{(q_1, \varepsilon)\} & \end{array}$$

Figura 8: AS pentru limbajul L_R.

$\delta(q_0, a, a)$ și $\delta(q_0, b, b)$ produc în mod nedeterminist două variante posibile de funcționare a automatului. O variantă generează o oglindă nouă și trece în regimul de ștergere (starea q_1), altă variantă continuă să înregistreze în stivă, ținând cont de faptul că oglinda generată poate fi falsă. Să urmărim acțiunile automatului la recunoașterea șirului *abbbba*, Figura 9.

Definiția 4.2

AS_1 este ecivalent cu AS_2, dacă $L(AS_1) = L(AS_2)$.

Vom arăta în continuare că pentru orice AS care acceptă prin stări finale (AS_F) se poate construi un AS echivalent care acceptă prin stivă vidă (AS_V), $L(AS_F) = L(AS_V)$. Ideea convertirii

$$(q_0,\$,abbbba) \vdash (q_0,a\$,bbbba) \vdash (q_0,ba\$,bbba) \begin{array}{l} \vdash (q_1,a\$,bba) \text{ - i} \\ \vdash (q_0,bba\$,bba) \vdash^{0} \end{array}$$

$$(q_0,bba\$,bba) \begin{array}{l} \vdash (q_1,ba\$,ba)\vdash(q_1,a\$,a)\vdash(q_1,\$,\varepsilon)\vdash(q_1,\varepsilon,\varepsilon) \text{ - a} \\ \vdash (q_0,bbba\$,ba) \begin{array}{l} \vdash (q_1,bba\$,a) \text{ - i} \\ \vdash (q_0,bbbba\$,a)\vdash(q_0,abbbba\$,\varepsilon) \text{ - i} \end{array} \end{array}$$

Figura 9: AS_R. Recunoașterea șirului *abbbba*.

AS_F în AS_V este simplă. AS_V va modela pas cu pas funcționarea AS_F, iar în momentul când AS_F ajunje într-o stare finală, AS_V trece într-o stare nouă și șterge toate simbolurile rămase în stivă. Dacă la momentul când stiva devine vidă este vidă și banda de intrare, considerăm că AS_V a ajuns într-o configurație de acceptare. Trebuie să se țină cont de faptul că AS_F poate ajunje într-o configurație $(q, \varepsilon, \varepsilon)$ cu $q \notin F$, care pentru el este o configurație de impas. Pentru a evita acceptarea acestei configurații de către AS_V (el modelează întocmai AS_F) se introduce un nou simbol evidențiat pentru stivă, de exemplu €, $\text{€} \notin \Gamma$, pe care îl poate accesa din stivă doar AS_V.

Să expunem în continuare algoritmul de convertire în detalii.

Algoritmul 4.1 (Convertirea AS_F în AS_V.)

0. ***start***

1. Este dat $AS_F = (Q, \Sigma, \Gamma, \delta, q_0, \$, F)$.
* Vom construi $AS_V = (Q', \Sigma, \Gamma', \delta', q_0, \text{€})$ *\

2. $Q' := Q \cup \{q_0', q_\varepsilon\}, q_0', q_\varepsilon \notin Q$

3. $\Gamma' := \Gamma \cup \{\text{€}\}, \text{€} \notin \Gamma$

4. Construim δ':

4.1. $\delta' := \delta$

4.2. $\delta'(q_0', €, \varepsilon) = \{(q_0, \$€)\}$

4.3. ***Pentru toate*** stările finale $q_f \in F$ și ***toate*** tranzițiile $\delta'(q_i, Z, a) \ni (q_f, \gamma)$ modificăm:
$\delta'(q_i, Z, a) := \delta'(q_i, Z, a) \cup \{(q_\varepsilon, \varepsilon)\}$

4.4. ***Pentru toți*** $Z \in \Gamma'$: $\delta'(q_\varepsilon, Z, \varepsilon) = \{(q_\varepsilon, \varepsilon)\}$

5. ***stop***

Pasul 4.2 din start înscrie în stivă simbolul €, care va fi tot timpul ultimul element al stivei și va putea fi șters doar în starea q_ε. Dacă AS_V ajunje în configurația $(q_i, Z\gamma€, ax)$ și $\delta'(q_i, Z, a) \ni (q_f, \gamma_1)$ (a poate fi și ε), atunci pașii 4.3 și 4.4 ne asigură tranzițiile:

$$(q_i, Z\gamma€, ax) \underset{AS_V}{\vdash} (q_\varepsilon, \gamma_1\gamma€, x) \underset{AS_V}{\overset{*}{\vdash}} (q_\varepsilon, €, x) \underset{AS_V}{\vdash} (q_\varepsilon, \varepsilon, x),$$

unde $q_f \in Q$.
Dacă $x = \varepsilon$, AS_V va ajunje în configurația de acceptare $(q_\varepsilon, \varepsilon, \varepsilon)$.

Demonstrarea echivalenței AS_F și AS_V, adică $L(AS_F) = L(AS_V)$, se bazează pe explicațiile și observațiile de mai sus.

Exemplul 4.3 Să construim AS_V pentru automatul din Exemplul 3.1. Pentru a evita ambiguitățile la notare, vom redenumi q_0' prin q_{00} și vom omite din descrierea AS_V semnul "$'$". Definiția ASV este prezentată în Figura 10.

$AS_V = (Q, \Sigma, \Gamma, \delta, q_{00}, €), Q = \{q_{00}, q_0, q_1, q_\varepsilon\}, \Sigma = \{a, b\}$,
$\Gamma = \{€, \$, A\}$,

$\delta(q_{00}, €, \varepsilon) = \{(q_0, \$€)\}$	$\delta(q_0, \$, a) = \{(q_0, A\$)\}$
$\delta(q_0, A, a) = \{(q_0, AA)\}$	$\delta(q_0, A, b) = \{(q_1, \varepsilon)\}$
$\delta(q_1, A, b) = \{(q_1, \varepsilon)\}$	$\delta(q_0, A, b = \{(q_\varepsilon, \varepsilon)\}$
$\delta(q_1, A, b) = \{(q_\varepsilon, \varepsilon)\}$	$\delta(q_\varepsilon, €, \varepsilon) = \{(q_\varepsilon, \varepsilon)\}$
$\delta(q_\varepsilon, A, \varepsilon) = \{(q_\varepsilon, \varepsilon)\}$	$\delta(q_\varepsilon, \$, \varepsilon) = \{(q_\varepsilon, \varepsilon)\}$

Figura 10: AS_V pentru limbajul $L_{ij} = \{a^i b^j \mid i \geq j \geq 1\}$

În Figura 11 prezentăm procesarea șirului *aaaabbb* de către AS_V.

$(q_{00}, €, aaaabbb) \vdash (q_0, \$€, aaaabbb) \overset{*}{\vdash} (q_0, AAAA\$€, bbb) \overset{0}{\vdash}$

$(q_0, AAAA\$€, bbb) \vdash (q_1, AAA\$€, bb)$
- $(q_1, AA\$€, b) \overset{0}{\vdash}$
- $(q_\varepsilon, AA\$€, bb) \overset{*}{\vdash} (q_\varepsilon, \varepsilon, bb)$ - i

$(q_1, AA\$€, b)$
- $(q_1, A\$€, \varepsilon) \vdash (q_\varepsilon, \$€, \varepsilon) \overset{*}{\vdash} (q_\varepsilon, \varepsilon, \varepsilon)$ - a
- $(q_\varepsilon, A\$€, b) \overset{*}{\vdash} (q_\varepsilon, \varepsilon, b)$ - i

Figura 11: Funcționarea automatului AS_V pentru limbajul $L_{ij} = \{a^i b^j \mid i \geq j \geq 1\}$

Există un algoritm analogic pentru convertirea AS_V în AS_F (Algoritmul 13.1, Paragraful "Soluții, indicații, răspunsuri").

5. Alte variante de automate cu memorie stivă

1. Modelele examinate mai sus nu admit tranziții cu stiva vidă. Se pot defini și modele care admit așa tranziții. Pentru aceasta se extinde funcția de tranziție în modul următor:

$$\delta : Q \times (\Gamma \cup \{\varepsilon\}) \times (\Sigma \cup \{\varepsilon\}) \longrightarrow 2^{Q \times \Gamma^*}$$

În acest caz pot fi definite tranziții de tipul:

$$\delta(q, \varepsilon, a) = \{(q_1, \gamma_1), (q_2, \gamma_2), \ldots, (q_n, \gamma_n)\} \qquad (1)$$

Pentru $\delta(q, \varepsilon, a) \ni (q_i, \gamma_i)$, de exemplu, automatul citește a de pe bandă, trece în starea q_i și, menținând simbolul din top, înregistrează în stivă γ_i, deplasând toată stiva cu $|\gamma_i|$ poziții. Astfel, definiția 1 poate fi interpretată ca $\delta(q, Z, a) = \{(q_1, \gamma_1 Z), (q_2, \gamma_2 Z), \dots, (q_n, \gamma_n Z)\}$, pentru toți $Z \in \Gamma$.

2. Dacă admitem tranziții cu stiva vidă, pot fi definite trei operații atomare (elementare):

avans - $\delta(q_i, \varepsilon, a) = (q_j, \varepsilon)$. *AS* citeșe simbolul curent de pe bandă ignorând stiva;

pop - $\delta(q_i, Z, \varepsilon) = (q_j, \varepsilon)$. *AS* șterge simbolul din topul stivei ignorând banda;

push Z - $\delta(q_i, \varepsilon, \varepsilon) = (q_j, Z)$. *AS* înregistrează Z în top, menținând conținutul stivei și ignorând banda.

Tranziția $\delta(q_i, Z, a) = (q_j, Z_1 Z_2 Z_3 Z_4)$, de exemplu, poate fi substituită prin:

$$\begin{aligned}
&\delta(q_i, \varepsilon, a) = (p_1, \varepsilon), && \textit{avans},\\
&\delta(p_1, \varepsilon, \varepsilon) = (p_2, Z_4), && \textit{push } Z_4,\\
&\delta(p_2, \varepsilon, \varepsilon) = (p_3, Z_3), && \textit{push } Z_3,\\
&\delta(p_3, \varepsilon, \varepsilon) = (p_4, Z_2), && \textit{push } Z_2,\\
&\delta(p_4, \varepsilon, \varepsilon) = (q_j, Z_1), && \textit{push } Z_1,
\end{aligned}$$

iar $\delta(q_i, Z, a) = (q_j, \varepsilon)$ - prin:

$$\begin{aligned}
&\delta(q_i, \varepsilon, a) = (p_5, \varepsilon), && \textit{avans},\\
&\delta(p_5, Z, \varepsilon) = (q_j, \varepsilon), && \textit{pop},
\end{aligned}$$

unde p_1, p_2, p_3, p_4, p_5 sunt stări noi care nu aparțin mulțimii Q.

Se poate afirma că pentru orice *AS* se poate construi un *AS* echivalent care are doar operații atomare.

Menționăm că tranzițiile prin stivă vidă reprezintă o sursă suplimentară de nedeterminism. De exemplu, pentru tranzițiile

$$\delta(q_1, \varepsilon, a) = (q_2, \gamma_2),\ \delta(q_1, Z_1, a) = (q_3, \gamma_3),\ \delta(q_1, \varepsilon, \varepsilon) = (q_4, \gamma_4)$$

și configurația $(q_1, Z_1\gamma_1, ax)$ avem trei ramificări posibile:

$$\begin{aligned}
&(q_1, Z_1\gamma_1, ax) \vdash (q_2, \gamma_2 Z_1 \gamma 1, x),\\
&(q_1, Z_1\gamma_1, ax) \vdash (q_3, \gamma_3 \gamma_1, x),\\
&(q_1, Z_1\gamma_1, ax) \vdash (q_4, \gamma_4 Z_1 \gamma 1, ax).
\end{aligned}$$

Astfel, la elaborarea aplicațiilor bazate pe automate cu memorie stivă este necesar de fiecare dată de precizat modelul *AS* utilizat.

6. Echivalența AS și a GIC. Algoritmul și teorema GAS

Definiția 6.1

Automatul cu memorie stivă AS este echivalent cu gramatica independentă de context G, dacă $L(AS) = L(G)$.

Vom arăta în continuare că pentru orice gramatică independentă de context (GIC) se poate construi un automat stivă echivalent și invers.

Vom arăta mai întâi cum se poate construi un AS echivalent cu gramatica independentă de context $G = (V_N, V_T, P, S)$. Automatul construit va modela derivările stângi în gramatică și va accepta prin stivă vidă. Orice derivare stângă are forma $A \stackrel{*}{\Longrightarrow} xB\gamma \stackrel{*}{\Longrightarrow} xy$, unde $A, B \in V_N$, $x, y \in V_T^*$, $\gamma \in (V_N \cup V_T)^*$.

Ținând cont de faptul că A în final derivează xy, $A \stackrel{*}{\Longrightarrow} xy$, să presupunem că AS inițial se află în configurația (q, A, xy). Deoarece $A \stackrel{*}{\Longrightarrow} xB\gamma$, iar, conform lemei de ramificare, $B\gamma \stackrel{*}{\Longrightarrow} y$, putem considera că AS va efectua tranzițiile $(q, A, xy) \stackrel{*}{\vdash} (q, xB\gamma, xy)$. Observăm că prefixul x din topul stivei coincide cu prefixul șirului xy de pe bandă. Vom admite că acest prefix poate fi recunoscut de către AS. Obținem $(q, A, xy) \stackrel{*}{\vdash} (q, xB\gamma, xy) \stackrel{*}{\vdash} (q, B\gamma, y)$. În concluzie, putem afirma: 1) dacă simbolul terminal din topul stivei coincide cu simbolul curent de pe bandă, AS efectuează o operație de avansare, citind elementul de pe bandă și ștergând elementul din topul stivei și 2) toate substituțiile derivării pot fi efectuate folosind doar stiva. Dacă, de exemplu, AS ajunje în configurația (q, B, y) și gramatica G conține producția $B \to y$, automatul va modela această situație în modul următor: $(q, B, y) \vdash (q, y, y) \stackrel{*}{\vdash} (q, \varepsilon, \varepsilon)$. În baza acestor idei expunem mai jos algoritmul de convertire a gramaticii G în AS echivalent.

Algoritmul 6.1 (Algoritmul GAS)

0. ***start***

1. Este dată gramatica independentă de context $G=(V_N,V_T,P,S)$. * Vom construi $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$ cu acceptare prin stivă vidă echivalent cu G *\
2. $Q = \{q\}, \Sigma = V_T, \Gamma = V_N \cup V_T, q_0 = q, \$ = S$.
3. Construim δ:
  3.1. $\delta(q, A, \varepsilon) := \{\}$ pentru toți $A \in V_N$.
  3.2. $\delta(q, a, a) = \{(q, \varepsilon)\}$ pentru toți $a \in \Sigma$.
  3.3. Pentru toate producțiile $A \to \alpha$ din P modificăm:
    $\delta(q, A, \varepsilon) := \delta(q, A, \varepsilon) \cup \{(q, \alpha)\}$.
4. ***stop***

Să observăm că automatul construit are o singură stare. Tranzițiile definite la pasul 3.2 le vom numi *"avansări"*, iar cele de la pasul 3.3 - *"expandări"*.

Exemplul 6.1 $G = (V_N, V_T, P, S), V_N = \{S\}, V_T = \{a, b\},\ P = \{0.\ S \to aSb,\ 1.\ S \to \varepsilon\}$.
Este ușor de observat că $L(G) = L_{aibi} = \{a^i b^i | i \geq 0\}$. Să construim aplicând algoritmul 6.1 automatul $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$:
$Q = \{q\}, \Sigma = \{a, b\}, \Gamma = \{S, a, b\}, q_0 = q, \$ = S$.
$\delta(q, a, a) = \{(q, \varepsilon)\}, \delta(q, b, b) = \{(q, \varepsilon)\}$ - *"avansări"*.
$\delta(q, S, \varepsilon) = \{(q, aSb), (q, \varepsilon)\}$ - *"expandări"*.

În Figura 12a vedem funcționarea AS construit, secvența de acceptare (Figura 12b) și derivarea șirului $aabb$ (Figura 12c).

Exemplul 6.2

$G = (V_N, V_T, P, S), V_N = \{S, A, B\}, V_T = \{a, b\},\ P = \{$
$0.\ S \to AB \quad 1.\ S \to ASB \quad 2.\ A \to a \quad 3.\ A \to aA \quad 4.\ B \to b$
$\}$ Observăm că $L(G) = Lij = \{a^i b^j \mid i \geq j \geq 1\}$. Construim automatul $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$:
$Q = \{q\}, \Sigma = \{a, b\}, \Gamma = \{S, A, B, a, b\}, q_0 = q, \$ = S$.

$$avans\breve{a}ri - \begin{bmatrix} \delta(q, a, a) = \{(q, \varepsilon)\}, \\ \delta(q, b, b) = \{(q, \varepsilon)\}, \end{bmatrix}$$

$$(q, S, aabb) \begin{array}{l} \nearrow (q, aSb, aabb) \vdash (q, Sb, abb) \begin{array}{l} \nearrow (q, aSbb, abb) \vdash^{0} \\ \searrow (q, b, abb) \text{ - impas} \end{array} \\ \searrow (q, \varepsilon, aabb) \text{ - impas} \end{array}$$

$$(q, aSbb, abb) \vdash (q, Sbb, bb) \begin{array}{l} \nearrow (q, aSbbb, bb) \text{ - impas} \\ \searrow (q, bb, bb) \vdash^{*} (q, \varepsilon, \varepsilon) \text{ - acceptare} \end{array}$$

(a) Recunoașterea șirului *aabb*.

$$(q, S, aabb) \vdash (q, aSb, aabb) \vdash (q, aSbb, abb) \vdash (q, Sbb, bb) \vdash^{0}$$
$$(q, Sbb, bb) \vdash (q, bb, bb) \vdash^{*} (q, \varepsilon, \varepsilon) \text{ - acceptare}$$

(b) Secvența de acceptare a șirului *aabb*.

$$S \Longrightarrow aSb \Longrightarrow aaSbb \Longrightarrow aabb$$

(c) Derivarea șirului *aabb*.

Figura 12: Funcționarea AS echivalent cu gramatica G, $L(G) = L_{aibi} = \{a^i b^i | i \geq 0\}$

$$expandări - \begin{cases} \delta(q,S,\varepsilon) = \{(q,AB),(q,ASB)\}, \\ \delta(q,A,\varepsilon) = \{(q,aA),(q,a)\}, \\ \delta(q,B,\varepsilon) = \{(q,b)\} \end{cases}$$

Teorema 6.1 (Teorema GAS)

AS construit cu Algoritmul 6.1 este echivalent cu gramatica independentă de context G.

Demonstrare. Vom demonstra că $L(AS) = L(G)$.

(i) $L(G) \subseteq L(AS)$.

Fie y un șir arbitrar din $L(G)$, $S \overset{*}{\Longrightarrow} y$. Vom demonstra prin inducție după numărul de substituții k la derivarea stângă a șirului că $(q,S,y) \overset{*}{\vdash} (q,\varepsilon,\varepsilon)$. Pentru $k=1$ fie $S \Longrightarrow y$. În acest caz există producția $S \to y$ în P, $y \in V_T^*$, $y = a_1a_2\ldots a_n$ și $(q,S,y) \vdash (q, a_1a_2\ldots a_n, a_1a_2\ldots a_n) \overset{*}{\vdash} (q,\varepsilon,\varepsilon)$.

Să presupunem că afirmația este adevărată pentru $k = 1,2\ldots m, m \geq 1$, și o vom demonstra pentru $k = m+1$. Fie $S \overset{m+1}{\Longrightarrow} y$. Să punem în evidență prima substituție: $S \overset{1}{\Longrightarrow} X_1X_2\ldots X_n \overset{m}{\Longrightarrow} y$, unde $S \to X_1X_2\ldots X_n$ este producție din P, iar $X_j \in (V_N \cup V_T), j = 1,2,\ldots,n$. Aplicând lema ramificării, obținem $S \overset{1}{\Longrightarrow} X_1X_2\ldots X_n \overset{m}{\Longrightarrow} y_1y_2\ldots y_n = y$, unde $X_j \overset{\leq m}{\Longrightarrow} y_j$. În conformitate cu ipoteza inducției avem $(q,X_j,y_j) \overset{*}{\vdash} (q,\varepsilon,\varepsilon), j = 1,2,\ldots,n$. Aplicând aceste tranziții și expandarea $\delta(q,S,\varepsilon) \ni (q, X_1X_2\ldots X_n)$, obținem:
$(q,S,y) \vdash (q, X_1X_2\ldots X_n, y_1y_2\ldots y_n) \overset{*}{\vdash} (q, X_2X_3\ldots X_n, y_2y_3\ldots y_n) \overset{*}{\vdash} (q,X_n,y_n) \overset{*}{\vdash} (q,\varepsilon,\varepsilon)$.

(ii) $L(AS) \subseteq L(G)$.

Vom demonstra mai întâi prin inducție după numărul de tranziții k că, dacă $(q,A,y) \overset{*}{\vdash} (q,\varepsilon,\varepsilon)$ pentru orice $A \in V_N$, atunci $A \overset{*}{\Longrightarrow} y$. Fie pentru $k = 1$ $(q,A,y) \overset{1}{\vdash} (q,\varepsilon,\varepsilon)$. Aceasta este posibil doar în cazul când

$x = \varepsilon$ și avem definită tranziția $\delta(q, A, \varepsilon) \ni (q, \varepsilon)$. Conform Algoritmului *GAS* așa tranziție se include pentru producția $A \to \varepsilon$. Respectiv, $A \Longrightarrow y = \varepsilon$. Să presupunem că afirmația este adevărată pentru $k = 1, 2, \ldots, m, m \geq 1$. Adică, dacă $(q, A, y) \models^{\leq m} (q, \varepsilon, \varepsilon)$, atunci $A \overset{*}{\Longrightarrow} y$. Să demonstrăm aceasta și pentru $k = m + 1$. Fie $(q, A, y) \models^{m+1} (q, \varepsilon, \varepsilon)$. Să punem în evidență prima tranziție, $\delta(q, A, \varepsilon) \ni (q, X_1X_2 \ldots X_n)$: $(q, A, y) \models^{1} (q, X_1X_2 \ldots X_n, y) \models^{m} (q, \varepsilon, \varepsilon)$. Conform Algoritmului *GAS*, gramatica conține producția $A \to X_1X_2 \ldots X_n$, $X_i \in V_N \cup V_T, i = 1, 2, \ldots, n$. Pornind cu stiva $X_1X_2 \ldots X_n$, așa cum *AS* poate șterge doar câte un singur simbol din topul stivei, iar în final automatul va șterge toate elementele din stivă și va citi toată banda, vor exista subșirurile $y_1, y_2, \ldots, y_n$, $y = y_1y_2 \ldots y_n$, cu proprietatea: $(q, X_1 \ldots X_n, y_1 \ldots y_n) \models^{*} (q, X_2 \ldots X_n, y_2 \ldots y_n) \models^{*} \ldots \models^{*} (q, X_n, y_n) \models^{*} (q, \varepsilon, \varepsilon)$. În caz contrar automatul nu ar putea concomitent să șteargă toată stiva și să citească toată banda.

Putem afirma că $(q, X_i, y_i) \models^{\leq m} (q, \varepsilon, \varepsilon)$. Într-adevăr, dacă $X_i \in V_T$, atunci $X_i = y_i = a$ și $(q, a, a) \models^{1} (q, \varepsilon, \varepsilon)$ prin avansare. Dacă $X_i \in V_N$, atunci $(q, X_i, y_i) \models^{\leq m} (q, \varepsilon, \varepsilon)$ și, conform ipotezei, $X_i \overset{*}{\Longrightarrow} y_i$. Aplicând lema ramificării putem construi derivarea:
$A \overset{1}{\Longrightarrow} X_1X_2 \ldots X_n \overset{*}{\Longrightarrow} y_1y_2 \ldots y_n = y$. Pentru a încheia demonstrarea, vom considera $A = S$. ■

În Figura 13 aducem un exemplu care ilustrează Teorema *GAS*. Menționăm că $y_3 = \varepsilon, X_5 = y_5 = a$.

7. Echivalența *AS* și a *GIC*. Algoritmul și teorema *ASG*

În acest paragraf vom arăta că pentru orice *AS* se poate construi o *GIC* echivalentă. Să formulăm mai întâi ideea algoritmului. Fie *AS* un automat care acceptă prin stivă vidă. Șirul $y \in L(AS)$, dacă există $q_i \in Q$ și $(q_0, \$, y) \models^{*} (q_i, \varepsilon, \varepsilon)$. Deoarece automatul nu are stări finale, starea q_i poate fi determinată doar la momentul acceptării. Așa cum

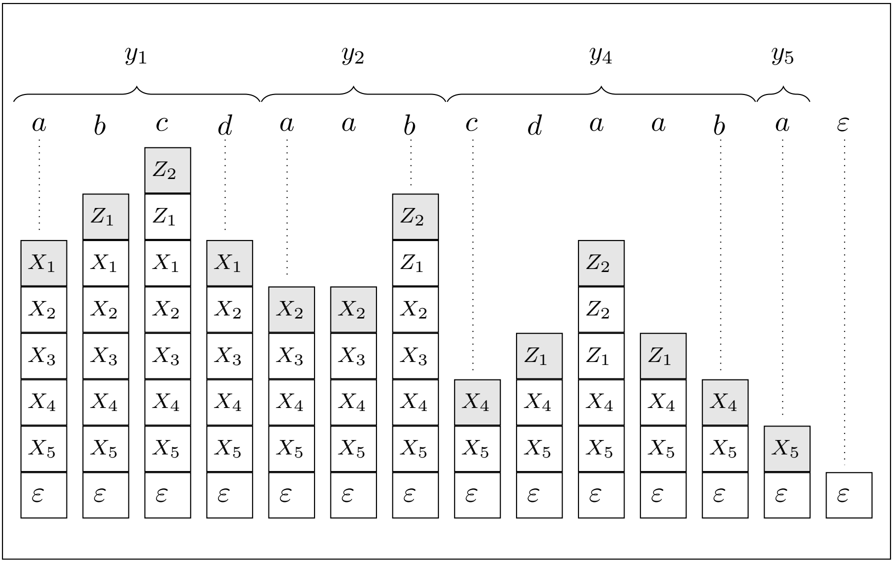


Figura 13: Ilustrare la Teorema GAS ($y_3 = \varepsilon, X_5 = y_5 = a$)

gramatica construită trebuie să deriveze y, $S \overset{*}{\Longrightarrow} y$, ea trebuie să țină cont de stările q_0, q_i. Aceste stări pentru gramatică vor apărea ca niște contexte ale simbolurilor neterminale. Vom nota aceste situații prin $[q_0\$q_i] \overset{*}{\Longrightarrow} y$. Vom nota prin S axioma gramaticii și vom introduce producțiile $S \to [q_0\$q_i]$ pentru toți $q_i \in Q$. Desigur, unele producții ar putea fi neproductive, dar acest fapt va putea fi determinat doar după construirea gramaticii.

Pentru un simbol arbitrar $A \in \Gamma$, dacă $(q_i, A, y) \overset{*}{\vdash} (q_j, \varepsilon, \varepsilon)$, vom considera că $A \overset{*}{\Longrightarrow} y$. Din nou avem nevoie de contexte: $[q_iAq_j] \overset{*}{\Longrightarrow} y$, unde q_j poate fi orice stare din Q.

Cum construim alte producții? Fie $\delta(q_i, A, a) \ni (q_j, X_1X_2 \ldots X_n)$, $a \in \Sigma \cup \{\varepsilon\}$. În acest caz $(q_i, A, ay) \overset{1}{\vdash} (q_j, X_1X_2 \ldots X_n), y \overset{*}{\vdash} (q_n, \varepsilon, \varepsilon)$. Bazându-ne pe raționamentele expuse în paragraful precedent, există

$y_1,\ y_2,\ \ldots, y_n,\ y = y_1 y_2 \ldots y_n$ și

$$\begin{aligned}(q_i, A, ay) &\overset{1}{\vdash} (q_j, X_1X_2\ldots X_n, y_1y_2\ldots y_n)\\ &\overset{*}{\vdash} (q_1, X_2X_3\ldots X_n, y_2y_3\ldots y_n)\\ &\overset{*}{\vdash} (q_2, X_3X_4\ldots X_n, y_3y_4\ldots y_n)\\ \ldots\\ &\overset{*}{\vdash} (q_{n-1}, X_n, y_n)\\ &\overset{*}{\vdash} (q_n, \varepsilon, \varepsilon)\end{aligned} \tag{2}$$

Observăm că la tranziția $(q_{k-1}, X_k\ldots X_n, y_k\ldots y_n)\ \overset{*}{\vdash}\ (q_k,\ X_{k+1}\ldots X_n, y_{k+1}\ldots y_n),\ 2\le k\le n$, în momentul când X_k este șters din stivă, AS trece în starea q_k, această stare fiind prima la recunoașterea subșirului y_{k+1} având X_{k+1} în topul stivei. Urmând ideea expusă mai sus, vor fi necesare neterminalele $[q_jX_1q_1]$, $[q_1X_2q_2]$, …, $[q_{n-1}X_nq_n]$ și producțiile

$$[q_iAq_n] \to a[q_jX_1q_1][q_1X_2q_2][q_2X_3q_3]\ldots[q_{n-1}X_nq_n] \tag{3}$$

Așa neterminale și producții vom construi pentru toate combinările posibile $q_1, q_2, \ldots, q_n,\ q_k\in Q, k = 1, 2, \ldots, n$. Mai precis,

$$(q_1, q_2, \ldots, q_n) \in \underbrace{Q\times Q\times\ldots\times Q}_{n}$$

Dacă $\delta(q_i, A, a)\ni(q_j, \varepsilon)$, atunci se va genera o singură producție: $[q_iAq_j]\to a$.

Prezentăm în Figura 14 o ilustrare a acestei idei.

De menționat că dimensiunea gramaticii astfel construite poate fi foarte mare. De exemplu, dacă $card(Q) = 5$, atunci pentru tranziția $\delta(q_i, A, a)\ni(q_j, X_1X_2X_3)$ vom obține $card(Q\times Q\times Q) = 125$ producții $[q_iAq_3]\to a[q_jX_1q_1][q_1X_2q_2][q_2X_3q_3],\ (q_1, q_2, q_3)\in Q\times Q\times Q$.

Expunem mai jos algoritmul de convertire a AS în gramatică echivalentă. Vom considera că AS acceptă prin stivă vidă și toate tranzițiile lui pentru $a\in\Sigma\cup\{\varepsilon\}$ au forma $\delta(q_i, X, a) = \{c_1, c_2, \ldots, c_n\}$,

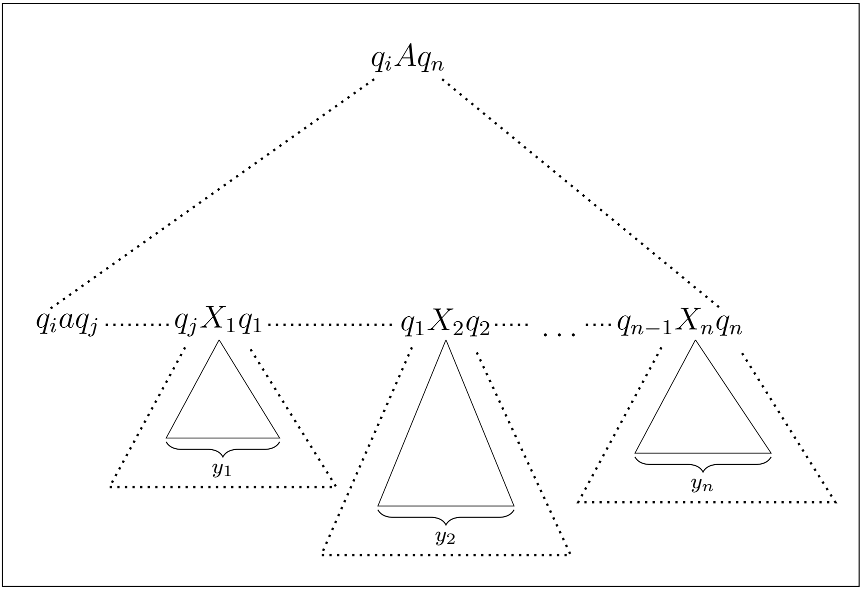


Figura 14: Ilustrare la Algoritmul *ASG*

unde

$$c_i = \begin{cases} (q_j, \varepsilon), \\ (q_j, X_1), \\ (q_j, X_1X_2) \end{cases} \tag{4}$$

pentru $i = 1, 2, \dots, n$.

Vom numi această formă de reprezentare a *AS* Formă Normală (FN)

În paragraful 5 am arătat că prin transformări echivalente orice *AS* poate fi redus la un *AS* echivalent care satisface acestor condiții.

Algoritmul 7.1 ALGORITMUL *ASG*.

0. ***Start***

1. Este dat $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$
* Vom construi gramatica independentă de context

$G=(V_N,V_T,P,S)$ echivalentă cu AS *\

2. $V_N = \{[q_iXq_j] \mid q_i, q_j \in Q, X \in \Gamma\}, V_T = \Sigma.$

3. Construim P.

3.1. $P := \{S \to [q_0\$q_j] \mid q_j \in Q\}.$

3.2. ***Pentru toți*** $\delta(q_i, X, a) \ni (q_j, \varepsilon)\}$, $P := P \cup \{[q_iXq_j] \to a\}.$

3.3. ***Pentru toți*** $\delta(q_i, X, a) \ni (q_j, X_1)\}$,
toți $q_1 \in Q$
$P := P \cup \{[q_iXq_1] \to a[q_jX_1q_1]\}.$

3.4. ***Pentru toți*** $\delta(q_i, X, a) \ni (q_j, X_1X_2)\}$,
toți $q_1 \in Q$,
toți $q_2 \in Q$
$P := P \cup \{[q_iXq_2] \to a[q_jX_1q_1][q_1X_2q_2]\}.$

4. ***Stop***

Exemplul 7.1

Este dat automatul
$AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$, $Q = \{q_0, q_1\}, \Sigma = \{a, b, c\}, \Gamma = \{\$, A, B\}$,
$\delta(q_0, \$, a) = \{(q_0, A, \$)\}$, $\delta(q_0, A, b) = \{(q_0, B)\}$
$\delta(q_0, B, c) = \{(q_1, \varepsilon)\}$, $\delta(q_1, \$, a) = \{(q_1, \varepsilon), (q_1, \$)\}$.
În rezultatul convertirii AS în gramatică echivalentă (Figura 15) obținem:
$G = (V_N, V_T, P, S), V_N = \{S, A, B, C, D, E, F, J, H\}$, $V_T = \{a, b, c\}$,
$P = \{$

0. $S \to A$	1. $S \to B$	2. $A \to aEA$	3. $A \to aFC$
4. $B \to aEB$	5. $B \to aFD$	6. $E \to bJ$	7. $F \to bH$
8. $H \to c$	9. $D \to a$	10. $C \to aC$	11. $D \to aD$ $\}$

Să simplificăm această gramatică. Calculăm mai întâi simbolurile productive și neproductive.

PRODUCTIVE $= \{H, D, F, B, S\}$, NEPRODUCTIVE $= \{A, C, E, J\}$.
După eliminarea simbolurilor neproductive obținem:
$G = (V_N, V_T, P, S)$, $V_N = \{S, B, F, D, H\}, V_T = \{a, b, c\}$,
$P = \{$ 0. $S \to B$ 1. $B \to aFD$ 2. $F \to bH$

$3.\ H \rightarrow c \qquad 4.\ D \rightarrow a \qquad 5.\ D \rightarrow aD\ \}.$

În continuare substituim H în producția 2, F în producția 1, B în producția 0 și eliminăm simbolurile inaccesibile. Obținem:

$$G=(V_N, V_T, P, S),\ V_N = \{S, D\}, V_T = \{a, b, c\},$$
$$P = \{0.\ S \rightarrow abcD \qquad 1.\ D \rightarrow a \qquad 2.\ D \rightarrow aD\ \}.$$

Este evident, $L(G) = \{abca^i \mid i \geq 1\}$. Să notăm $L(G)$ prin L_{abcai}.

<table>
<tr><th>Funcția δ</th><th colspan="2">Producțiile generate</th><th>Notații</th></tr>
<tr><td></td><td>$S \rightarrow [q_0\$q_0]$
$S \rightarrow [q_0\$q_1]$</td><td>$S \rightarrow A$
$S \rightarrow B$</td><td rowspan="6">$[q_0\$q_0]$ – A
$[q_0\$q_1]$ – B
$[q_1\$q_0]$ – C
$[q_1\$q_1]$ – D
$[q_0Aq_0]$ – E
$[q_0Aq_1]$ – F
$[q_0Bq_1]$ – H</td></tr>
<tr><td>$\delta(q_0, \$, a)=$
$\{(q_0, A\$)\}$</td><td>$[q_0\$q_0] \rightarrow a[q_0Aq_0][q_0\$q_0]$
$[q_0\$q_0] \rightarrow a[q_0Aq_1][q_1\$q_0]$
$[q_0\$q_1] \rightarrow a[q_0Aq_0][q_0\$q_1]$
$[q_0\$q_1] \rightarrow a[q_0Aq_1][q_1\$q_1]$</td><td>$A \rightarrow aEA$
$A \rightarrow aFC$
$B \rightarrow aEB$
$B \rightarrow aFD$</td></tr>
<tr><td>$\delta(q_0, A, b)=$
$\{(q_0, B)\}$</td><td>$[q_0Aq_0] \rightarrow b[q_0Bq_0]$
$[q_0Aq_1] \rightarrow b[q_0Bq_1]$</td><td>$E \rightarrow bJ$
$F \rightarrow bH$</td></tr>
<tr><td>$\delta(q_0, B, c)=$
$\{(q_1, \varepsilon)\}$</td><td>$[q_0Bq_1] \rightarrow c$</td><td>$H \rightarrow c$</td></tr>
<tr><td>$\delta(q_1, \$, a)=$
$\{(q_1, \varepsilon)\}$</td><td>$[q_1\$q_1] \rightarrow a$</td><td>$D \rightarrow a$</td></tr>
<tr><td>$\delta(q_1, \$, a)=$
$\{(q_1, \$)\}$</td><td>$[q_1\$q_0] \rightarrow a[q_1\$q_0]$
$[q_1\$q_1] \rightarrow a[q_1\$q_1]$</td><td>$C \rightarrow aC$
$D \rightarrow aD$</td></tr>
</table>

Figura 15: Algoritmul ASG. Exemplu.

Exemplul 7.2

Să construim gramatica echivalentă cu automatul de recunoaștere a palindroamelor peste $\{a, b\}$, exemplul 4.2. Prezentăm rezultatele algoritmului în Figura 16, iar gramatica obținută G_{R0} în Figura 17.

Se obține o gramatică destul de voluminoasă - 13 neterminale și 31 de producții. Asupra gramaticii G_{R0} vom aplica următoarea secvență de transformări echivalente:

1. Calculăm mulțimea simbolurilor neproductive și simplificăm gramatica G_{R0}.

   $$\text{PRODUCTIVE} = \{K, I, M, D, F, B, S\},$$
   $$\text{NEPRODUCTIVE} = \{A, C, E, H, J, L\}.$$

   Rezultatul este prezentat în Figura 18a, gramatica G_{R1}.

Funcția δ	Producțiile generate		Notații
	$S \to [q_0\$q_0]$ $S \to [q_0\$q_1]$	$S \to A$ $S \to B$	$[q_0\$q_0]$ – A $[q_0\$q_1]$ – B $[q_1\$q_0]$ – C $[q_1\$q_1]$ – D $[q_0aq_0]$ – E $[q_0aq_1]$ – F $[q_1aq_0]$ – H $[q_1aq_1]$ – I $[q_0bq_0]$ – J $[q_0bq_1]$ – K $[q_1bq_0]$ – L $[q_1bq_1]$ – M
$\delta(q_0,\$,a)=$ $\{(q_0,a\$)\}$	$[q_0\$q_0] \to a[q_0aq_0][q_0\$q_0]$ $[q_0\$q_0] \to a[q_0aq_1][q_1\$q_0]$ $[q_0\$q_1] \to a[q_0aq_0][q_0\$q_1]$ $[q_0\$q_1] \to a[q_0aq_1][q_1\$q_1]$	$A \to aEA$ $A \to aFC$ $B \to aEB$ $B \to aFD$	
$\delta(q_0,\$,b)=$ $\{(q_0,b\$)\}$	$[q_0\$q_0] \to b[q_0bq_0][q_0\$q_0]$ $[q_0\$q_0] \to b[q_0bq_1][q_1\$q_0]$ $[q_0\$q_1] \to b[q_0bq_0][q_0\$q_1]$ $[q_0\$q_1] \to b[q_0bq_1][q_1\$q_1]$	$A \to bJA$ $A \to bKC$ $B \to bJB$ $B \to bKD$	
$\delta(q_0,a,b)=$ $\{(q_0,ba)\}$	$[q_0aq_0] \to b[q_0bq_0][q_0aq_0]$ $[q_0aq_0] \to b[q_0bq_1][q_1aq_0]$ $[q_0aq_1] \to b[q_0bq_0][q_0aq_1]$ $[q_0aq_1] \to b[q_0bq_1][q_1aq_1]$	$E \to bJE$ $E \to bKH$ $F \to bJF$ $F \to bKI$	
$\delta(q_0,b,a)=$ $\{(q_0,ab)\}$	$[q_0bq_0] \to a[q_0aq_0][q_0bq_0]$ $[q_0bq_0] \to a[q_0aq_1][q_1bq_0]$ $[q_0bq_1] \to a[q_0aq_0][q_0bq_1]$ $[q_0bq_1] \to a[q_0aq_1][q_1bq_1]$	$J \to aEJ$ $J \to aFL$ $K \to aEK$ $K \to aFM$	
$\delta(q_0,a,a)=$ $\{(q_0,aa)\}$	$[q_0aq_0] \to a[q_0aq_0][q_0aq_0]$ $[q_0aq_0] \to a[q_0aq_1][q_1aq_0]$ $[q_0aq_1] \to a[q_0aq_0][q_0aq_1]$ $[q_0aq_1] \to a[q_0aq_1][q_1aq_1]$	$E \to aEE$ $E \to aFH$ $F \to aEF$ $F \to aFI$	
$\delta(q_0,a,a)=$ $\{(q_1,\varepsilon)\}$	$[q_0aq_1] \to a$	$F \to a$	
$\delta(q_0,b,b)=$ $\{(q_0,bb)\}$	$[q_0bq_0] \to b[q_0bq_0][q_0bq_0]$ $[q_0bq_0] \to b[q_0bq_1][q_1bq_0]$ $[q_0bq_1] \to b[q_0bq_0][q_0bq_1]$ $[q_0bq_1] \to b[q_0bq_1][q_1bq_1]$	$J \to bJJ$ $J \to bKL$ $K \to bJK$ $K \to bKM$	
$\delta(q_0,b,b)=$ $\{(q_1,\varepsilon)\}$	$[q_0bq_1] \to b$	$K \to b$	
$\delta(q_1,a,a)=$ $\{(q_1,\varepsilon)\}$	$[q_1aq_1] \to a$	$I \to a$	
$\delta(q_1,b,b)=$ $\{(q_1,\varepsilon)\}$	$[q_1bq_1] \to b$	$M \to b$	
$\delta(q_1,\$,\varepsilon)=$ $\{(q_1,\varepsilon)\}$	$[q_1\$q_1] \to \varepsilon$	$D \to \varepsilon$	

Figura 16: Gramatica echivalentă cu automatul palindroamelor.

2. Substituim B in producția 0, apoi I, M, D în producțiile 1,2,3, 4,5,8 și eliminăm simbolurile inaccesibile B, I, M, D. Obținem

$G_{R0} = (V_N, V_T, P, S)$, $V_N = \{S, A, B, C, D, E, F, H, I, J, K, L, M\}$,
$V_T = \{a, b\}$, $P = \{$

0. $S \to A$	1. $S \to B$	2. $A \to aEA$	3. $A \to aFC$
4. $B \to aEB$	5. $B \to aFD$	6. $A \to bJA$	7. $A \to bKC$
8. $B \to bJB$	9. $B \to bKD$	10. $E \to bJE$	11. $E \to bKH$
12. $F \to bJF$	13. $F \to bKI$	14. $J \to aEJ$	15. $J \to aFL$
16. $K \to aEK$	17. $K \to aFM$	18. $E \to aEE$	19. $E \to aFH$
20. $F \to aEF$	21. $F \to aFI$	22. $F \to a$	23. $J \to bJJ$
24. $J \to bKL$	25. $K \to bJK$	26. $K \to bKM$	27. $K \to b$
28. $I \to a$	29. $M \to b$	30. $D \to \varepsilon$ $\}$	

Figura 17: Gramatica G_{R0}.

gramatica G_{R2}, Figura 18b.

3. Factorizăm prefixele commune pentru F și K. Obținem gramatica G_{R3}, Figura 18c.
4. Eliminăm ε-producția $Z \to \varepsilon$. Obținem G_{R4}, Figura 18d.
5. Substituim F și K în producțiile 0,1,6 și 7. Eliminăm simbolurile inaccesibile F și K. Obținem G_{R5}, Figura 18e.
6. Pentru G_{R5} se demonstrează simplu că, daca $S \stackrel{*}{\Longrightarrow} x$, atunci $Z \stackrel{*}{\Longrightarrow} x$, și invers. Astfel, gramatica finală este G_{R6}, echivalentă cu G_{R0}, Figura 18f.

Exemplul 7.3

Să construim pentru automatul cu memorie stivă AS_{01}, Exemplul 4.1, gramatica independentă de context echivalentă, aplicând algoritmul ASG (Figura 19). Obținem gramatica G_5.

G_5=(V_N, V_T, P, S), $V_N = \{S, A, B, C\}$, $V_T = \{1, 0\}$, $P = \{$

0. $S \to C$	1. $C \to \varepsilon$	2. $C \to 0BC$	3. $C \to 1AC$
4. $A \to 1AA$	5. $B \to 0BB$	6. $A \to 0$	7. $B \to 1$

Deoarece $S \longrightarrow C$ este unica producție pentru S, $L(G) = \{x \mid S \stackrel{*}{\Longrightarrow} x\}$= $\{x \mid C \stackrel{*}{\Longrightarrow} x\}$. Rezultă că prima producție poate fi eliminată, iar

(a) G_{R1}	(b) G_{R2}	(c) G_{R3}
0. $S \to B$	0. $S \to aF$	0. $S \to aF$
1. $B \to aFD$	1. $S \to bK$	1. $S \to bK$
2. $B \to bKD$	2. $F \to bKa$	2. $F \to Za$
3. $F \to bKI$	3. $F \to aFa$	3. $K \to Zb$
4. $K \to aFM$	4. $F \to a$	4. $Z \to aF$
5. $F \to aFI$	5. $K \to aFb$	5. $Z \to bK$
6. $F \to a$	6. $K \to bKb$	6. $Z \to \varepsilon$
7. $K \to bKM$	7. $K \to b$	
8. $K \to b$		
9. $I \to a$		
10. $M \to b$		
11. $D \to \varepsilon$		

(d) G_{R4}	(e) G_{R5}	(f) G_{R6}
0. $S \to aF$	0. $S \to aZa$	0. $S \to aSa$
1. $S \to bK$	1. $S \to aa$	1. $S \to aa$
2. $F \to Za$	2. $S \to bZb$	2. $S \to bSb$
3. $F \to a$	3. $S \to bb$	3. $S \to bb$
4. $K \to Zb$	4. $Z \to aZa$	
5. $K \to b$	5. $Z \to aa$	
6. $Z \to aF$	6. $Z \to bZb$	
7. $Z \to bK$	7. $Z \to bb$	

Figura 18: Transformări echivalente
asupra gramaticii G_{R0}

C devine noua axiomă a gramaticii. În rezultat obținem gramatica G_{01}.

$G_{01}=(V_N, V_T, P, S)$, $V_N = \{S, A, B\}$, $V_T = \{1, 0\}$, $P = \{$

0. $S \to \varepsilon$ 1. $S \to 0BS$ 2. $S \to 1AS$ 3. $B \to 0BB$
4. $A \to 1AA$ 5. $A \to 0$ 6. $B \to 1$ $\}$

Algoritmul de convertire asigură echivalența automatului AS_{01} și a gramaticii G_1.

Funcția δ	Producții generate	
	$S \rightarrow [q_0\$q_0]$	$S \rightarrow C$
$\delta(q_0,\$,\varepsilon)=\{(q_0,\varepsilon)\}$	$[q_0\$q_0] \rightarrow \varepsilon$	$C \rightarrow \varepsilon$
$\delta(q_0,\$,0)=\{(q_0,0\$)\}$	$[q_0\$q_0] \rightarrow 0[q_0 0 q_0][q_0\$q_0]$	$C \rightarrow 0BC$
$\delta(q_0,\$,1)=\{(q_0,1\$)\}$	$[q_0\$q_0] \rightarrow 1[q_0 1 q_0][q_0\$q_0]$	$C \rightarrow 1AC$
$\delta(q_0,0,0)=\{(q_0,00)\}$	$[q_0 0 q_0] \rightarrow 0[q_0 0 q_0][q_0 0 q_0]$	$B \rightarrow 0BB$
$\delta(q_0,0,0)=\{(q_0,00)\}$	$[q_0 0 q_0] \rightarrow 0[q_0 0 q_0][q_0 0 q_0]$	$B \rightarrow 0BB$
$\delta(q_0,1,1)=\{(q_0,11)\}$	$[q_0 1 q_0] \rightarrow 1[q_0 1 q_0][q_0 1 q_0]$	$A \rightarrow 1AA$
$\delta(q_0,0,1)=\{(q_0,\varepsilon)\}$	$[q_0 0 q_0] \rightarrow 1$	$B \rightarrow 1$
$\delta(q_0,1,0)=\{(q_0,\varepsilon)\}$	$[q_0 0 q_0] \rightarrow 0$	$A \rightarrow 0$
Notații: $[q_0 1 q_0]$ - A $[q_0 0 q_0]$ - B $[q_0\$q_0]$ - C		

Figura 19: Gramatica G_1 echivalentă cu AS_{01}.

Teorema 7.2 (Teorema ASG)

Gramatica independentă de context G construită cu Algoritmul 7.1 este echivalentă cu automatul cu memorie stivă AS.

Demonstrare. Vom demonstra că $L(AS)=L(G)$.

(i) $L(G) \subseteq L(AS)$.

Fie y un șir arbitrar din $L(G)$, există producția $S \xrightarrow{1} [q_0\$q_2]$ și derivarea $S \overset{1}{\Longrightarrow} [q_0\$q_2] \overset{*}{\Longrightarrow} y$. Vom arăta că $(q_0,\$,y) \overset{*}{\vdash} (q_2,\varepsilon,\varepsilon)$, adică $y \in L(AS)$. Să demonstrăm mai întâi prin inducție după numărul de substituții k la derivarea stângă a șirului următoarea afirmație: dacă $[q_0Aq_2] \overset{*}{\Longrightarrow} y$, atunci $(q_0,A,y) \overset{*}{\vdash} (q_2,\varepsilon,\varepsilon)$. Pentru $k=1$ $[q_0Aq_2] \overset{1}{\Longrightarrow} y$ este posibil, reieșind din structura gramaticii, doar pentru $y=a$, $a \in \Sigma \cup \{\varepsilon\}$ $(\Sigma = V_T)$, $A=\$$ și producția $[q_0\$q_2] \longrightarrow a$. Această producție se generează la pasul 3.2 al algoritmului ASG din tranziția $\delta(q_0, \$, a) \ni (q_2, \varepsilon)$. Pentru AS obținem $(q_0, A, y) = (q_0, \$, a) \overset{1}{\vdash} (q_2, \varepsilon, \varepsilon)$.

Presupunem că afirmația este adevărată pentru $k = 1, 2, \ldots, m, m \geq 1$, adică, dacă $[q_i A q_2] \overset{\leq m}{\Longrightarrow} y$, atunci $(q_i, A, y) \overset{*}{\vdash} (q_2, \varepsilon, \varepsilon)$. Să demonstrăm afirmația pentru $k = m+1$. Fie $[q_i A q_2] \overset{m+1}{\Longrightarrow} y$. Să punem în evidență primul pas al derivării:

(a) $[q_i A q_2] \overset{1}{\Longrightarrow} a[q_j X_1 q_1][q_1 X_2 q_2] \overset{m}{\Longrightarrow} y$, dacă $\delta(q_0, A, a) \ni (q_j, X_1 X_2)$, $(q_1, q_2) \in Q \times Q$.

(b) $[q_i A q_2] \overset{1}{\Longrightarrow} a[q_j X_2 q_2] \overset{m}{\Longrightarrow} y$, dacă $\delta(q_0, A, a) \ni (q_j, X_2)$, $q_2 \in Q$.

Pentru varianta (a), conform lemei ramificării, există y_1, y_2, $y = a y_1 y_2$ și $a[q_j X_1 q_1][q_1 X_1 q_2] \overset{m}{\Longrightarrow} a y_1 y_2 = y$, unde $[q_j X_1 q_1] \overset{\leq m}{\Longrightarrow} y_1$, $[q_1 X_2 q_2] \overset{\leq m}{\Longrightarrow} y_2$. Astfel, conform ipotezei, $(q_j, X_1, y_1] \overset{*}{\vdash} (q_1, X_2, y_2)$ și $(q_1, X_2, y_2] \overset{*}{\vdash} (q_2, \varepsilon, \varepsilon)$. Obținem
$(q_i, A, y) = (q_i, A, a y_1 y_2) \overset{*}{\vdash} (q_j, X_1 X_2, y_1 y_2) \overset{*}{\vdash} (q_1, X_2, y_2) \overset{*}{\vdash} (q_2, \varepsilon, \varepsilon)$.
Pentru a finaliza demonstrarea luăm $A = \$$, $q_i = q_0$ și producția $S \longrightarrow [q_0 \$ q_2]$. Varianta (b) se examinează analogic.

(ii) $L(AS) \subseteq L(G)$.

Vom arăta că, dacă $(q_0, \$, y) \overset{*}{\vdash} (q_2, \varepsilon, \varepsilon)$, atunci există producția $S \overset{1}{\longrightarrow} [q_0 \$ q_2]$ și $S \Longrightarrow [q_0 \$ q_2] \overset{*}{\Longrightarrow} y$. Să demonstrăm mai întâi prin inducție după numărul de tranziții k următoarea afirmație: dacă $(q_i, A, y] \overset{*}{\vdash} (q_2, \varepsilon, \varepsilon)$, atunci $[q_i A q_2] \overset{*}{\Longrightarrow} y$. Pentru $k = 1$ fie $(q_i, A, y] \overset{1}{\vdash} (q_2, \varepsilon, \varepsilon)$. Aceasta este posibil doar pentru $y = a$, $a \in \Sigma \cup \{\varepsilon\}$, și $\delta(q_i, A, a) \ni (q_2, \varepsilon)$. În acest caz gramatica conține producția $[q_i A q_2] \longrightarrow a$ și este posibilă derivarea $[q_i A q_2] \Longrightarrow a$.
Presupunem că afirmația este adevărată pentru orice $k \leq m, m \geq 1$: dacă $(q_i, A, y) \overset{\leq m}{\models} (q_2, \varepsilon, \varepsilon)$, atunci $[q_i A q_2] \overset{*}{\Longrightarrow} y$. Să demonstrăm afirmația pentru $k = m+1$. Fie $(q_i, A, y) \overset{m+1}{\models} (q_2, \varepsilon, \varepsilon)$, $y = ax$, $a \in \Sigma \cup \{\varepsilon\}$. Să punem în evidență prima tranziție:

(a) $(q_i, A, ax) \overset{1}{\models} (q_j, X_1 X_2, x) \overset{n}{\models} (q_2, \varepsilon, \varepsilon)$, dacă $\delta(q_i, A, a) \ni (q_j, X_1 X_2)$,

(b) $(q_i, A, ax) \overset{1}{\models} (q_j, X_2, x) \overset{n}{\models} (q_2, \varepsilon, \varepsilon)$, dacă $\delta(q_i, A, a) \ni (q_j, X_2)$.

Pentru varianta (a), urmând raționamentele din Teorema GAS și ținând cont de faptul că AS acceptă prin stivă vidă, există subșirurile y_1, y_2 pentru care $x = y_1y_2$ și $(q_j, X_1, y_1) \overset{\leq m}{\vdash} (q_1, \varepsilon, \varepsilon)$, $(q_1, X_2, y_2) \overset{\leq m}{\vdash} (q_2, \varepsilon, \varepsilon)$. Respectiv, conform ipotezei, $[q_jX_1q_1] \overset{*}{\Longrightarrow} y_1$, $[q_1X_2q_2] \overset{*}{\Longrightarrow} y_2$.
Deoarece $\delta(q_i, A, a) \ni (q_j, X_1X_2)$, la pasul 3.4 al Algoritmului ASG se va genera producția $[q_iAq_2] \Longrightarrow a[q_jX_1q_1][q_1X_2q_2]$. Astfel, putem construi derivarea $[q_iAq_2] \overset{1}{\Longrightarrow} a[q_jX_1q_1][q_1X_2q_2] \overset{*}{\Longrightarrow} ay_1[q_1X_2q_2] \overset{*}{\Longrightarrow} ay_1y_2$.

Pentru a finaliza demonstrarea luăm $A = \$$, $q_i = q_0$ și producția $S \longrightarrow [q_0\$q_2]$. Varianta (b) se examinează analogic. ∎

8. Automate cu memorie stivă deterministe

Ca și în cazul automatelor finite, putem defini AS deterministe. Intuitiv aceasta înseamnă că la fiecare tranziție există doar o singură acțiune posibilă sau niciuna. Prezența masivă a ε-tranzițiilor, care induc nedeterminism, complică definiția AS determinist. Prezența unei ε-tranziții, de exemplu, $\delta(q, Z, \varepsilon) \ni (q_1, \gamma_1)$, chiar dacă ea este unică, permite aplicarea ei pentru orice simbol curent de pe bandă $a \in \Sigma \cup \{\varepsilon\}$. Aceasta ne convinge că nu este suficient să cerem $card(\delta(q, Z, a)) \leq 1$, prin analogie cu cazul automatelor finite. Avem astfel definiția 8.1.

Definiția 8.1

Automatul cu memorie stivă AS este determinist (ASD), dacă pentru orice $q \in Q$, $Z \in \Gamma$, $a \in \Sigma \cup \{\varepsilon\}$ se indeplinesc condițiile:

1. $\delta(q, Z, a)$ conține cel mult o valoare, adică $card(\delta(q, Z, a)) \leq 1$.
2. dacă $\delta(q, Z, a) \neq \{\}$, atunci $\delta(q, Z, \varepsilon) = \{\}$.

De exemplu, automatele 3.1 și 4.1 sunt deterministe. Automatul 4.2 este nedeterminist, deoarece $\delta(q_0, a, a) = \{(q_0, aa), (q_1, \varepsilon)\}$ (nu se îndeplinește prima condiție). Nedeterminist este și automatul 4.3, deoarece $\delta(q_1, A, b) = \{(q_1, \varepsilon)\}$, $\delta(q_1, A, \varepsilon) = \{(q_\varepsilon, \varepsilon)\}$ (nu se îndeplinește condiția a doua).

Definiția 8.2

Se numește limbaj determinist orice limbaj $L = L(ASD)$.

Noțiunea de limbaj determinist este una foarte importantă în studierea limbajelor formale, în special la proiectarea compilatoarelor. Spre deosebire de automatele finite, unde pentru orice automat se poate construi un automat determinist echivalent, pentru automatele cu memorie stivă aceasta nu mai este adevărat. Există AS pentru care nu poate fi construită o variantă deterministă echivalentă. Să menționăm următoarele: AS din exemplul 6.1 este determinist cu toate că este bine cunoscut faptul că limbajul $L_{aibi} = \{a^i b^i | i \geq 0\}$ nu este regulat. Astfel, există limbaje deterministe care nu sunt regulate. Vom demonstra în continuare că

(1) toate limbajele regulate sunt limbaje deterministe,

(2) există limbaje independente de context nedeterministe.

Mai întâi vom arăta că pentru orice AF se poate construi un AS echivalent.

Algoritmul 8.1 ALGORITMUL $AFAS$.

0. ***Start***

1. Este dat $AF = (Q, \Sigma, \delta, q_0, F)$
* Vom construi $AS = (Q, \Sigma, \Gamma, \delta', q_0, \$, F)$ cu acceptare prin stări finale echivalent cu AF *\

2. $\Gamma = \{\$\}$

3. Construim δ'. Inițial $\delta' = \varnothing$.

3.1. ***Pentru toate*** valorile $\delta(q,a) = \{q_1, q_2, \ldots, q_n\}$ definim $\delta' := \delta' \cup \{\delta'(q,\$,a) = (q_1,\$), (q_2,\$), \ldots, (q_n,\$)\}$.

4. ***Stop***

AS astfel construit modelează pas cu pas tranzițiile AF menținând în stivă \$. Se poate arăta ușor, ca dacă $(q_0, x) \vdash_{AF}^{*} (q_f, \varepsilon)$, atunci $(q_0, \$, x) \vdash_{AS}^{*} (q_f, \$, x)$ și invers. Astfel, pentru orice AF se poate construi un AS echivalent. Să observăm, că dacă AF este determinist (AFD), atunci și AS va fi determinist (ASD). Conform definiției AFD, toate tranzițiile lui au forma $\delta(q,a) = q_1$ sau $\delta(q,a) = \varnothing$. Respectiv, toate tranzițiile AS construit au forma $\delta(q,\$,a) = (q_1,\$)$ sau $\delta(q,\$,a) = \varnothing$, pentru toți $q \in Q$, $a \in \Sigma$. De menționat că AFD nu poate să conțină ε-tranziții. Astfel, AS construit va satisface definiția ASD.

Ținând cont de faptul că pentru orice AF se poate construi un AFD echivalent, are loc următoarea

Teorema 8.3

Limbajele regulate sunt limbaje deterministe.

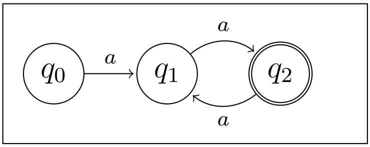


Figura 20: AF pentru $L_{a2n} = \{a^{2n} | n \geq 1\}$.

Să remarcăm că pentru orice AFD putem construi un AS determinist echivalent cu acceptare prin stări finale. Însă nu pentru orice AFD putem construi un AS determinist echivalent cu acceptare prin stivă vidă. Vom spune că limbajul L nu este prefixat, dacă pentru orice șir $x \in L$ nu există niciun prefix y al șirului x, $y \neq x$, care să aparțină limbajului L. În caz contrar vom spune că L este prefixat. Se poate demonstra că pentru limbajele prefixate nu se poate construi un ASD cu acceptare prin stivă vidă care să accepte acest limbaj.

De exemplu, limbajul $L_{a2n} = \{a^{2n} \mid n \geq 1\}$ este prefixat, deoarece pentru șirul $x = aaaaaa \in L$ avem prefixele aa și $aaaa$, care deasemenea aparțin limbajului L_{a2n}. L_{a2n} este un limbaj regulat, De exemplu, automatul finit reprezentat în Figura 20 recunoaște acest limbaj.

Conform algoritmului 8.1 putem construi un ASD echivalent cu acceptare prin stări finale, dar nu vom putea construi un ASD echivalent cu acceptare prin stivă vidă. Dacă vom presupune că există așa un automat, atunci la recunoașterea șirului $x = aaaa$, după citirea primelor două simboluri "a" automatul trebuie să golească stiva, acceptând aa. Deoarece AS este determinist și nu poate funcționa cu stiva vidă, nu va mai putea continua recunoașterea șirului $aaaa$.

Această problemă se rezolvă simplu. Redefinim limbajul L_{a2n} prin $L_{a2n} = \{a^{2n}€ \mid n \geq 1\}$. În acest caz nu mai pot exista prefixe ale șirului $a^{2n}€$ care să aparțină limbajului L, deoarece toate șirurile limbajului trebuie să se termine cu €. Pentru limbajul redefinit prezentăm mai jos ASD echivalent cu acceptare prin stivă vidă:

$$\delta(q_0, \$, a) = (q_1, \$), \qquad \delta(q_1, \$, a) = (q_2, \$),$$
$$\delta(q_2, \$, a) = (q_1, \$), \qquad \delta(q_2, \$, €) = (q_2, \varepsilon).$$

Să ne convingem în continuare că există limbaje independente de context care nu sunt deterministe. Să examinăm limbajul $L_{abb} = \{a^i b^i \mid i \geq 1\} \cup \{a^i b^i b^i \mid i \geq 1\}$. Acest limbaj nu este reguat, dar este independent de context. El poate fi generat, de exemplu, de gramatica

$G = (V_N, V_T, P, S), V_N = \{S, A, B\}, V_T = \{a, b\}$,

$P = \{$ 0. $S \to A$ 1. $S \to B$ 2. $A \to ab$
3. $A \to aAb$ 4. $B \to abb$ 5. $B \to aBbb$ $\}$

Este adevărată următoarea afirmație:

Teorema 8.4 (Teorema ASND)

Limbajul L_{abb} nu este determinist.

Demonstrare.

Vom demonstra prin reducere la absurd că acest limbaj nu este determinist, adică nu există ASD care ar recunoaște limbajul L_{abb}.

Să presupunem că L_{abb} este determinist. Atunci există ASD $M_1 = (Q, \Sigma, \Gamma, \delta, q_0, \$, F)$, care acceptă acest limbaj prin stări finale, $L_{abb} = L(M_1)$.

Construim o copie (clonă) a acestui automat, $M_2 = (Q', \Sigma, \Gamma, \delta', q'_0, \$, F')$. La clonare M_2 se obține din M_1 substituind toate stările $q_i \in Q$ prin $q'_i \in Q'$. Stările q_i și q'_i le vom numi stări "gemene". Să construim în continuare un automat nou $M = (Q_M, \Sigma_M, \Gamma, \delta_M, q_{0M}, \$, F_M)$, unde $Q_M = Q \cup Q'$, $\Sigma_M = \{a, b, c\}$, $q_{0M} = q_0$, $F_M = F'$, iar funcția δ_M se definește în modul următor:

- modificăm funcția δ substituind toate tranzițiile $\delta(q, Z, b) = (q_f, \gamma)$ pentru $q_f \in F$, $Z \in \Gamma$, $\gamma \in \Gamma^*$, $a \in \Sigma$ prin $\delta(q, Z, b) = (q'_f, \gamma)$, adică le direcționăm în starea geamănă a automatului M_2.
- modificăm funcția δ' substituind toate tranzițiile $\delta'(q'_i, Z, b) = (q'_j, \gamma)$ din M_2 prin $\delta'(q'_i, Z, c) = (q'_j, \gamma)$.
- δ_M se obține prin reuniunea funcțiilor δ și δ' astfel modificate.

Reieșind din cele expuse mai sus și din faptul că M_1 și M_2 sunt deterministe, și M va fi determinist.

Să urmărim mai întâi comportamentul automatului M_1 la recunoașterea șirului $a^n b^n b^n$:

$$(q_0, \$, a^n b^n b^n) \mathrel{\vdash^{*}_{M_1}} (q_1, Z\gamma_1, bb^n) \mathrel{\vdash^{1}_{M_1}} (q_2, \gamma_2\gamma_1, b^n) \mathrel{\vdash^{*}_{M_1}} (q_3, \gamma_3, \varepsilon),$$

q_2 și q_3 - stări finale. Configurația $(q_2, \gamma_2\gamma_1, b^n)$ ar fi una de acceptare, dacă în acest moment banda ar fi vidă, adică pentru șirul $a^n b^n$.

Să urmărim acum comportamentul automatului M la recunoașterea șirului $a^n b^n c^n$:

$$(q_0, \$, a^n b^n c^n) \mathrel{\vdash^{*}_{M}} (\vdash^{*}_{M_1}) \ (q_1, Z\gamma_1, bc^n) \mathrel{\vdash^{1}_{M}} (q'_2, \gamma_2\gamma_1, c^n) \mathrel{\vdash^{*}_{M}} (q'_3, \gamma_3, \varepsilon).$$

Deoarece M_1 și M sunt deterministe, există o unică posibilitate de a obține $(q_0, \$, a^n b^n b^n) \vdash^{*}_{M_1} (q_2, \gamma_2\gamma_1, b^n)$ și $(q_0, \$, a^n b^n c^n) \vdash^{*}_{M} (q'_2, \gamma_2\gamma_1, c^n)$, q'_2 fiind geamăna stării q_2. Tranzițiile $(q_2, \gamma_2\gamma_1, b^n) \vdash^{*}_{M_1} (q_3, \gamma_3, \varepsilon)$ și $(q'_2, \gamma_2\gamma_1, c^n) \vdash^{*}_{M} (q'_3, \gamma_3, \varepsilon)$ se deosebesc doar prin substituțiile: "b" prin "c", q_2 prin q'_2 și q_3 prin q'_3, asigurate prin construcția funcției δ_M.

În așa mod automatul M va accepta limbajul $L_{abc} = \{a^i b^i c^i \mid i \geq 1\}$, aceasta fiind imposibil, deoarece limbajul L_{abc} nu este independent de context. Obținem o absurditate, ceea ce înseamnă că presupunerea noastră referitor la existența automatului M_1 este falsă. ■

Pentru multe limbaje independente de context poate fi construită varianta deterministă. Să aducem un exemplu.

Exemplul 8.1 Fie dată gramatica:

$G = (V_N, V_T, P, S), V_N = \{S, L\}, V_T = \{a, i, [,]\}$,
$P = \{\ 0.\, S \to a\, L \qquad 1.\, L \to [i] \qquad 2.\, L \to [i\, L]\ \}$.
Se poate observa că limbajul generat este $L_{[i]} = \{a([i)^n]^n \mid n \geq 1\} = \{a[i], a[i[i]], a[i[i[i]]], \ldots\}$.

Cunoscând structura limbajului, în cazul dat se poate ușor construi un ASD care acceptă acest limbaj. De exemplu,

$$AS = (Q, \Sigma, \Gamma, \delta, q_0, \$),\ Q = \{q_0, q_1, q_2\},$$
$$\Sigma = \{a, i, [,]\},\ \Gamma = \{\$, [\}$$
$$\begin{array}{ll} \delta(q_0, \$, a) = \{(q_0, \$)\}, & \delta(q_0, \$, [) = \{(q_1, [)\}, \\ \delta(q_1, [, i) = \{(q_2, [\}, & \delta(q_2, [, [) = \{(q_2, [[)\}, \\ \delta(q_2, [, i) = \{(q_2, [)\}, & \delta(q_2, [,]) = \{(q_2, \varepsilon)\} \end{array}$$

Nu întotdeauna structura limbajului este destul de transparentă. Gramatica generatoare poate fi foarte voluminoasă, de exemplu, pentru limbajele de programare. În astfel de situații, automatul determinist ar putea fi construit aplicând transformări echivalente asupra gramaticii și algoritmul GAS. Să exemplificăm mai jos aceste afirmații.

Construim aplicând algoritmul GAS, automatul echivalent cu gramatica G.

$AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$:
$Q = \{q\}, \Sigma = \{a, i, [,]\}, \Gamma = \{S, L, a, i, [,]\}, q_0 = q, \$ = S$,

$$avansări - \left[\begin{array}{l} \delta(q, a, a) = \{(q, \varepsilon)\}, \\ \delta(q, i, i) = \{(q, \varepsilon)\}, \\ \delta(q, [, [) = \{(q, \varepsilon)\}, \\ \delta(q,],]) = \{(q, \varepsilon)\}, \end{array} \right.$$

$$expand\breve{a}ri - \begin{bmatrix} \delta(q, S, \varepsilon) = \{(q, a\, L), \}, \\ \delta(q, L, \varepsilon) = \{(q, [i]), (q, [i\, L])\} \end{bmatrix}$$

Automatul construit este nedeterminist, cauza principală fiind prezența ε-tranzițiilor. Vom încerca să modificăm automatul eliminând situațiile de nedeterminism. Putem observa că șirurile corecte întotdeauna vor începe cu a, astfel tranziția $\delta(q, S, \varepsilon) = \{(q, aL)\}$ poate fi substituită prin $\delta(q, S, a) = \{(q, L)\}$. Pentru toate celelalte simboluri din Σ automatul se va bloca având S în topul stivei. Pentru a înlătura ambiguitățile generate de tranzițiile $\delta(q, L, \varepsilon) = \{(q, [i]), (q, [i\, L])\}$ transformăm gramatica efectuând factorizarea producțiilor 1 și 2 și substituirea neterminalului L. Obținem:

$G = (V_N, V_T, P, S), V_N = \{S, A\}, V_T = \{a, i, [,]\}$,
$P = \{\ 0.\ S \to a[iA \quad 1.\ A \to] \quad 2.\ A \to [iA]\ \}$

Construim din nou AS echivalent, $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$:
$Q = \{q\}, \Sigma = \{a, i, [,]\}, \Gamma = \{S, A, a, i, [,]\}, q_0 = q, \$ = S$,

$$avans\breve{a}ri - \begin{bmatrix} \delta(q, a, a) = \{(q, \varepsilon)\}, \\ \delta(q, i, i) = \{(q, \varepsilon)\}, \\ \delta(q, [, [) = \{(q, \varepsilon)\}, \\ \delta(q,],]) = \{(q, \varepsilon)\}, \end{bmatrix}$$

$$expand\breve{a}ri - \begin{bmatrix} \delta(q, S, \varepsilon) = \{(q, a[iA)\}, \\ \delta(q, A, \varepsilon) = \{(q,]), (q, [iA])\} \end{bmatrix}$$

Tranziția $\delta(q, S, \varepsilon) = \{(q, a[iA)\}$ poate fi substituită prin $\delta(q, S, a) = \{(q, [iA)\}$, tranziția $\delta(q, A, \varepsilon) = \{(q,])\}$ poate fi substituită prin $\delta(q, A,]) = \{(q, \varepsilon)\}$, iar $\delta(q, A, \varepsilon) = \{(q, [iA])\}$ - prin $\delta(q, A, [) = \{(q, iA])\}$. În rezultat obținem automatul prezentat în Figura 21a.

Evident, în definiția funcției δ prezența stării "q" este inutilă. Dacă o excludem și reprezentăm funcția δ sub forma unui tabel, T, unde prin "V" vom nota "*aVansare*", prin "A" - "*Acceptare*", iar prin $T(S, a) = [iA,\ T(A,]) = \varepsilon,\ T(A, [) = iA]$, - expandările, vom obține tabelul din Figura 21b.

Prezentăm mai jos secvența de acceptare pentru șirul "$a[i[i]]$". Am

avansări
$$\left[\begin{array}{l}\delta(q,a,a)=\{(q,\varepsilon)\},\\ \delta(q,i,i)=\{(q,\varepsilon)\},\\ \delta(q,[,[)=\{(q,\varepsilon)\},\\ \delta(q,],])=\{(q,\varepsilon)\},\end{array}\right.$$

expandări
$$\left[\begin{array}{l}0.\ \delta(q,S,a)=\{(q,[iA)\},\\ 1.\ \delta(q,A,])=\{(q,\varepsilon)\}\\ 2.\ \delta(q,A,[)=\{(q,iA])\}.\end{array}\right.$$

(a) reprezentarea analitică

	a	i	$[$	$]$	ε
a	V				
i		V			
$[$			V		
$]$				V	
ε					A
S	$[iA$				
A			$iA]$	ε	

(b) reprezentarea tabelară

Figura 21: *ASD* pentru limbajul
$L=\{a([i)^n]^n \mid n\geq 1\}=\{a[i],a[i[i]],a[i[i[i]]],\ldots\}$

notat prin “$\stackrel{V}{\vdash}$” avansările, prin “$\stackrel{A}{\vdash}$” - acceptarea, iar prin “$\stackrel{0}{\vdash}$, $\stackrel{1}{\vdash}$, $\stackrel{2}{\vdash}$” - expandările.

$(S,a[i[i]])\stackrel{0}{\vdash}([iA,[i[i]])\stackrel{V}{\vdash}(iA,i[i]])\stackrel{V}{\vdash}(A,[i]])\stackrel{2}{\vdash}(iA],i]])\stackrel{V}{\vdash}(A],]])\stackrel{1}{\vdash}(],])\stackrel{V}{\vdash}(\varepsilon,\varepsilon)\stackrel{A}{\vdash}$.

Pentru un șir greșit, de exemplu, $a[i[ii])$, obținem: $(S,a[i[ii])\stackrel{0}{\vdash}([iA,[i[ii])\stackrel{V}{\vdash}(iA,i[ii])\stackrel{V}{\vdash}(A,[ii])\stackrel{2}{\vdash}(iA]],i]])\stackrel{V}{\vdash}(A],i])\vdash$ impas. Pentru a continua procesul de recunoaștere este necesar ca simbolul curent de pe bandă să fie "[" sau "]", nu "i". În asemenea situații, de obicei, se afișează un mesaj de eroare, cum ar fi: “Eroare – $a[i[i\,\boxed{i}\,]$ – se așteaptă "[" sau "]"”.

Exemplul de mai sus ne arată că utilizarea gramaticilor, datorită arsenalului bogat de transformări echivalente, este mai flexibilă, intuitiv mai clară.

Vom mai menționa că reprezentarea tabelară a *ASD* constituie nucleul analizorului sintactic descendent bazat pe gramatici $LL(1)$, dar aceste subiecte țin deja de proiectarea compilatoarelor.

9. Programarea automatelor cu memorie stivă

Acest paragraf este destinat programării algoritmilor expuși pe parcurs. În calitate de limbaj de programare am ales limbajul programării funcționale Common Lisp [19, 20], limbaj potrivit pentru procesare simbolică. Toate funcțiile (programele) și exemplele au fost testate folosind versiunea CLISP - ANSI Common Lisp 2.49 [21] și mediul integrat LispIDE [22] și incluse în regim `"verbatim"`, astfel, cititorul poate să le testeze, dar și să le modifice după dorință. Aceasta ar putea fi un bun exercițiu pentru cei care vor să se familiarizeze cu paradigma programării funcționale și cu limbajul Common Lisp.

Reprezentarea *AS*

Automatul $AS = (Q, \Sigma, \Gamma, \delta, q_0, F)$ se va reprezenta sub forma a 5 liste:

- `*Q*` - multimea de stări, *Q*$=(q_0\ q_1\ \dots\ q_n)$. Starea este reprezentată printr-un atom simbolic, de obicei $q_0, q_1, \dots, q_n$, unde q_0 întotdeauna va fi unica stare inițială. De exemplu,
  `(setq *Q* '(q0 q1 q2 q3))`.
- `*SIGMA*` - vocabularul de intrare. Elementele vocabularului vor fi șiruri arbitrare. De exemplu,
  `(setq *SIGMA* '("a" "b" "begin" "" "0" "1"))`.
- `*GAMMA*`-vocabularul stivei, multime arbitrară nevidă de atomi. De exemplu,
  `(setq *GAMMA* '($ A "b"))`.
- funcția δ o vom reprezenta prin lista *AS*$=(d_1\ d_2\ \dots\ d_m)$, unde fiecare d_i reprezintă valoarea unei tranziții conform definiției. Astfel, $d_i = (a_0\ t_1\ \dots\ t_k)$. Prin a_0 vom nota argumentul funcției, $a_0 = (q\ Z\ "a")$ sau $a_0 = (q\ Z)$ pentru ε-tranziții, iar t_j este o listă, care reprezintă valorile funcției, $t_j = ((q_1\ \gamma_1)(q_2\ \gamma_2)\dots\ (q_l\ \gamma_l))$. Dacă $\gamma_i = \varepsilon$, atunci valoarea respectivă va fi (q_i). Exemplu:

```
((q0 A "a") (q0 (A A $))(q1) (q1 (A "b" A $))).
```

- `*F*` - mulțimea stărilor finale, `*F*`$\subseteq$ `*Q*`.

Am folosit notațiile *nume*, se mai poate spune „nume cu ochelari”, deoarece pentru programele noastre aceste variabile sunt globale mutabile. Valorile lor pot fi modificate și utilizate ca variabile globale fară a le transmite prin transfer de parametri, „ochelarii” servind doar pentru a atrage atenția.

Am notat funcția δ prin *AS* deoarece, dacă *AS* este definit corect și $F = \varnothing$, atunci toate celelalte componente ale automatului se pot extrage din definiția funcției δ.

Pentru limbajul $L_{i2jk} = \{a^i b^{2j} c^k | i \geq k \geq 1, j \geq 1\}$ avem reprezentarea:

AS pentru $L_{i2jk} = \{a^i b^{2j} c^k | i \geq k \geq 1, j \geq 1\}$

```
(setq *Q* '(q0 q1 q2 q3))
(setq *SIGMA* '("a" "b" "c"))
(setq *GAMMA* '($ A B))
(setq *F* '(q3))
(setq *AS* '(
           ((q0 $ "a") (q0 (A $)))
           ((q0 A "a") (q0 (A A)))
           ((q0 A "b") (q1 (B A)))
           ((q1 B "b") (q1 (B B)) (q2))
           ((q2 B "b") (q2))
           ((q2 A "c") (q3))
           ((q3 A "c") (q3))
          )
)
```

Validarea *AS*

Validarea este un algoritm simplu de verificare a *AS*. Se verifică conform definiției

- corectitudinea argumentului funcției δ.
- corectitudinea tranzițiilor t_i.
- inadmisibilitatea utilizării multiple a stărilor $q00$, qe și a simbo-

lului *EURO* din vocabularul stivei, folosite doar la convertirea ASF în ASV. În caz de ambiguitate, se redenumesc unele stări, unele simboluri.

Pentru validarea *AS* sunt definite trei funcții:

Funcția *valid−head* verifică corectitudinea argumentului funcției δ. Acesta poate fi (q Z "a") sau (q Z) pentru ε-producții, $q \in Q$, $Z \in \Gamma$, $a \in \Sigma$.

Functia *valid−tail* verifica corectitudinea tranzițiilor posibile definite cu δ. Toate valorile unui argument trebuie sa fie de forma: $(q_1\ \gamma_1)\ \ (q_2\ \gamma_2)\ ...\ (q_n\ \gamma_n)$.

Funcția *validare* aplică consecutiv *valid–head* și *valid–tail* pentru toate definitiile δ. Daca este gresită structura listei (eroare de sintaxă) sau se utillizează elemente nedefinite, vom obține un mesaj de eroare, de exemplu, "Delta inadmisibil" sau "Simboluri nedefinite".

Linearizarea AS

Pentru toate aplicațiile realizate vom folosi varianta *AS* linearizată cu acceptare prin stivă vidă, adică cu $F = \varnothing$. Mai întâi vom converti automatele cu acceptare prin stări finale în automate echivalente cu acceptare prin stivă vidă. Concomitent vom efectua și validarea *AS*.

Procesul de linearizare constă în următoarele: fiecare tranziție $d_i = (a_0\ t_1\ \ldots\ t_k)$ se substituie prin $(a_0\ t_1)$, $(a_0\ t_2)$, ... , $(a_0\ t_k)$. Fiecare a_0 se va substitui prin trei componente: (q Z "a") sau (q Z "") pentru ε-tranziții. Fiecare t_j se va prezenta ca $(q_j\ Z_1\ Z_2 \ldots Z_r)$, dacă $\gamma_j = Z_1\ Z_2 \ldots Z_r$ sau (q_j), dacă $\gamma_j = \varepsilon$. Deoarece nu pot apărea ambiguități, parantezele pot fi omise: $(a_0\ t_k)$=(q Z "a" q_j Z_1 $Z_2 \ldots$ Z_r). De exemplu, automatul prezentat mai sus, în rezultatul linearizării se va transforma în:

AS linearizat pentru L_{i2jk}

```
*Q*=(Q0 Q1 Q2 Q3)
  *SIGMA*=("a" "b" "c")
  *GAMMA*=($ A B)
  *F*=(Q3)
```

```
*AS*=
    (Q0 $ "a" Q0 A $)
    (Q0 A "a" Q0 A A)
    (Q0 A "b" Q1 B A)
    (Q1 B "b" Q1 B B)
    (Q1 B "b" Q2)
    (Q2 B "b" Q2)
    (Q2 A "c" Q3)
    (Q3 A "c" Q3)
```

Prezentăm în continuare listingul programelor descrise mai sus.

AS: reprezentare, validare, linearizare

```
(defun memberq (e l)(member e l :test #'equal))

;;  Functia valid-head verifica corectitudinea argumentului functiei
;;  delta: (q Z "a") sau (q Z)
(defun valid-head(hd)
 (if(<(length hd)2)(return-from valid-head(format t "~%Delta
      inadmisibil: ~S " hd)))
 (let((q(car hd))(Z(cadr hd))(a(caddr hd)))
  (append(if(not(memberq q *Q*))(list q))
      (if(not(memberq Z *GAMMA*))(list Z))
      (if(not(memberq a *SIGMA*))(list a))))
)

;; Functia valid-tail verifica corectitudinea tranzitiilor
;; posibile definite cu delta: (q1 gamma1) (q2 gamma2)...(qn gamman)
(defun valid-tail(x)
  (append(set-difference(cadr x) *GAMMA* :test #'equal)
     (if(not(memberq (car x) *Q*))(list(car x))))
)

;;  Functia validare aplica consecutiv valid-head si valid-tail
;;  pentru toate definitiile delta
(defun validare(&aux val)
  (dolist (ti *AS*)
```

```
  (setq val(append val (valid-head(car ti))
          (mapcar #'valid-tail (cdr ti)))))
 (setq val(remove nil val))
 (if(null val) (return-from validare "AS validat."))
 (return-from validare(format t "~%Simboluri nedefinite: ~S" val))
)

;; Functia linearizare substituie delta(q Z "a")=((q1 gamma1) (q2
     gamma2)...(qn gamman))
;; prin  (q Z "a" q1 gamma1) (q Z "a" q2 gamma2)...(q Z "a" qn
     gamman)
(defun linearizare(&aux as)
 (dolist (el *AS*)
  (let((d1 (car el))(dn (cdr el)))
   (if(equal 2(length d1))(setq d1(append d1 '(""))))
   (dolist (d dn)
      (setq d (append (list(car d))(cadr d)))
      (setq as (cons (append d1 d) as)))))
 (return-from linearizare (reverse as))
)

;; Functia print-lin-as tipareste AS linearizat
(defun print-lin-as()
 (format t "~%(setq *AS* '(")
 (dolist (el *AS*)(format t "~% ~S" el))
 (format t "))")
)

;; Exemplu, AS pentru Li2jk, ASi2jk

(setq *Q* '(q0 q1 q2 q3))
(setq *SIGMA* '("a" "b" "c"))
(setq *GAMMA* '($ A B))
(setq *F* '(q3))
(setq *AS* '(
            ((q0 $ "a") (q0 (A $)))
            ((q0 A "a") (q0 (A A)))
            ((q0 A "b") (q1 (B A)))
```

```
            ((q1 B "b") (q1 (B B)) (q2))
            ((q2 B "b") (q2))
            ((q2 A "c") (q3))
            ((q3 A "c") (q3))
           )
)

(validare)
(setq *AS* (linearizare))

(format t
  "~%~3:T*Q*=~S *SIGMA*=~S *GAMMA*=~S *F*=~S
   *AS*=~{~%~8:T~S~}" *Q* *SIGMA* *GAMMA* *F* *AS*)

;;  Rezultat, ASi2jk linearizat

   *Q*=(Q0 Q1 Q2 Q3)
   *SIGMA*=("a" "b" "c")
   *GAMMA*=($ A B)
   *F*=(Q3)
   *AS*=
        (Q0 $ "a" Q0 A $)
        (Q0 A "a" Q0 A A)
        (Q0 A "b" Q1 B A)
        (Q1 B "b" Q1 B B)
        (Q1 B "b" Q2)
        (Q2 B "b" Q2)
        (Q2 A "c" Q3)
        (Q3 A "c" Q3)
```

Convertirea AS_F în AS_V echivalent

Funcția *convertire–asf–asv* convertește automatul linearizat ASF în AS_V echivalent conform algoritmului descris în paragraful 3.

Dacă mulțimea de stări finale ecte vidă, se va considera că automatul este deja un AS_V. În continuare funcția $convertire-asf-asv$ verifică prezența stărilor `q00,qe` și a simbolului `EURO` în `*Q*` și, respectiv, în `*GAMMA*`. Aceasta se produce în scopul evitării utilizării multiple a acestor simboluri. Dacă aceste simboluri nu se întâlnesc în mulțimile respective, se modifică `*Q*` și `*GAMMA*` adăugând stările noi `q00,qe` si, respectiv, simbolul nou `EURO`. În caz contrar se afișează un mesaj de eroare cu propunerea de a redenumi unele simboluri. Starea `q00` va fi noua stare inițială, iar `EURO` - noul simbol evidențiat al stivei. Starea `qe` va corespunde stării q_ε conform algoritmului.

Prezentăm în continuare listingul funcției descrise mai sus.

AS: convertire AS_F în AS_V

```
(defun memberq(l1 l2)(member l1 l2 :test #'equal))

;; Funcia convertire-asf-asv converteste automatul linearizat
;; ASF in ASV echivalent
(defun convertire-asf-asv(&optional (as *AS*))
   (if (null *F*) (return-from convertire-asf-asv *AS*))
   (let((s(remove nil(list (car(memberq 'q00  *Q*))
                           (car(memberq 'qe   *Q*))
                           (car(memberq 'EURO *GAMMA*))))))
       (if s (return-from convertire-asf-asv
            (format t "~%Utilizare multipla. De redenumit: ~S " s))))
  (setq *Q* (append '(q00 qe) *Q*))
  (setq *GAMMA* (append '(EURO) *GAMMA* ))
  (dolist (qf *F*)
     (dolist (ti *AS*)
        (when (equal qf (cadddr ti))
              (setq as(append as(list(list(car ti)(cadr ti)(caddr
      ti)'qe)))))))
  (setq *AS* as *F* nil)
  (dolist(z *GAMMA*)
      (setq *AS*(append *AS*(list(list'qe z "" 'qe)))))
  (setq *AS*(cons'(q00 EURO "" q0 $ EURO)*AS*))
)

;; Exemplu, ASi2jk linearizat
```

```
(setq *Q* '(q0 q1 q2 q3))
(setq *SIGMA* '("a" "b" "c"))
(setq *GAMMA* '($ A B))
(setq *F* '(q3))
(setq *AS* '(
        (Q0 $ "a" Q0 A $)
        (Q0 A "a" Q0 A A)
        (Q0 A "b" Q1 B A)
        (Q1 B "b" Q1 B B)
        (Q1 B "b" Q2)
        (Q2 B "b" Q2)
        (Q2 A "c" Q3)
        (Q3 A "c" Q3)
             )
)

(setq *AS* (convertire-asf-asv))

(format t
  "~%~3:T*Q*=~S
   *SIGMA*=~S
   *GAMMA*=~S
   *F*=~S
   *AS*=~{~%~8:T~S~}" *Q* *SIGMA* *GAMMA* *F* *AS*)

;; Rezultat, ASV echivalent cu ASi2jk

*Q*=(Q00 QE Q0 Q1 Q2 Q3)
   *SIGMA*=("a" "b" "c")
   *GAMMA*=(EURO $ A B)
   *F*=NIL
   *AS*=
        (Q00 EURO "" Q0 $ EURO)
        (Q0 $ "a" Q0 A $)
        (Q0 A "a" Q0 A A)
        (Q0 A "b" Q1 B A)
        (Q1 B "b" Q1 B B)
        (Q1 B "b" Q2)
```

```
(Q2 B "b" Q2)
(Q2 A "c" Q3)
(Q3 A "c" Q3)
(Q2 A "c" QE)
(Q3 A "c" QE)
(QE EURO "" QE)
(QE $ "" QE)
(QE A "" QE)
(QE B "" QE)
```

Modelarea automatelor stivă

Deoarece întotdeauna putem construi varianta echivalentă a AS cu acceptare prin stivă vidă, AS_V, vom modela doar automate AS_V. Modelarea funcționării unui AS constă în construirea tuturor configurațiilor de acceptare și/sau impas pentru un șir dat.

Toate configurațiile de acceptare le vom stoca in variabila globală mutabilă `*ACCEPT*`, iar cele de impas - in `*IMPAS*`. Vom mai folosi variabila `*NOI*` pentru a gestiona iterațiile algoritmului. Inițial `*ACCEPT*` = `*IMPAS*` = nil, iar `*NOI*=((q0 ($) x (nil)))` - configurația inițială. Parantezele duble apar deoarecepe pe parcurs se va construi o listă de configurații (listă de liste). Aceasta este valabil și pentru `*ACCEPT*` si `*IMPAS*`.

Conform materialului expus în paragraful 3, configurația este un obiect de forma (q, γ, x), unde q este starea curentă, γ - conținutul stivei, x - partea neanalizată a șirului de pe bandă. Vom adăuga încă un element - calea parcursă la modelare. Astfel, toate configurațiile vor fi de forma: $(q, \gamma, x, cale)$. Calea reprezintă consecutivitatea tranzițiilor (numerele de ordine ale tranzitiilor din definiția funcției δ) efectuate la moment pornind de la configurația inițială.

Pentru a explica funcționarea ASV ne vom referi la automatul linearizat care acceptă limbajul $L_{ab23} = \{a^i b^j | 2i \leq j \leq 3i, i \geq 0\}$ prezentat în listingul de mai jos.

AS pentru $L_{ab23} = \{a^i b^j | 2i \leq j \leq 3i, i \geq 0\}$

```
(setq *AS* '(
        (q0 $ "" q3)
        (q0 $ "a" q1 A A A $)
        (q0 $ "a" q1 A A $)
        (q1 A "a" q1 A A A A)
        (q1 A "a" q1 A A A)
        (q1 A "" q2 A)
        (q2 A "b" q2)
        (q2 A "b" q3)
        (q3 $ "" q3) ) )
```

Prezentăm și poziționarea tranzițiilor definite pentru *AS* folosită la reprezentarea secvențelor de tranziții ale configurației.

Poziționarea tranzițiilor pentru L_{ab23}

```
*AS*=
        0. (Q0 $ "" Q3)
        1. (Q0 $ "a" Q1 A A A $)
        2. (Q0 $ "a" Q1 A A $)
        3. (Q1 A "a" Q1 A A A A)
        4. (Q1 A "a" Q1 A A A)
        5. (Q1 A "" Q2 A)
        6. (Q2 A "b" Q2)
        7. (Q2 A "b" Q3)
        8. (Q3 $ "" Q3)
```

Algoritmul de modelare constă din două funcții principale: *tranzitie* și *configuratii*. Funcția *tranzitie* are două argumente: *configuratie* și *delta*. Destinația funcției este de a verifica dacă este posibilă trecerea directă din *configuratie* aplicând tranziția *delta*. Sunt posibile situațiile:

- configurația verificată este una de acceptare; în acest caz funcția va returna "accept" fără a modifica configurația;
- tranziția *delta* poate fi aplicată; în acest caz se va returna "valid" și configurația nouă, obținută la aplicarea tranziției *delta*.

- tranziția *delta* nu poate fi aplicată; în acest caz se va returna configurația nemodificată, dar marcată ca candidat la o configurație de impas.

De exemplu, configurația `(Q3 NIL "" (1 4 5 6 6 6 6 7 8))` este una de acceptare pentru șirul $z =$ ”*aabbbbb*”. Pentru același șir, configurația `(Q1 A A A $ "abbbbb" (1))` este validă pentru tranziția `4.(Q1 A "a" Q1 A A A)`, producând configurația nouă
`(Q1 A A A A A $ "bbbbb" (1 4))`.

Funcția *configuratii* gestionează integral procesul de modelare. Pornind cu `*NOI*=((q0 ($) x (nil)))`, adică cu configurația inițială, procesul de modelare va continua cât timp vor apărea configurații noi în `*NOI*`. Orice iterație nouă va parcurge toate configurațiile din `*NOI*` și toate tranzițiile din `*AS*`, construind configurații noi. Dacă aceasta nu este posibil, procesul se termină, afișând toate configurațiile de acceptare și de impas.

Variabila `flaga`=*true*, daca configuratia curentă este una de acceptare ea fiind inclusă în `*ACCEPT*`. Dacă nici o tranziție nu poate fi aplicată asupra configurației curente, atunci `flagi`=*true* și această configurație va fi inclusă în `*IMPAS*`.

Să precizăm în continuare notațiile facute în program stabilind corespunderea lor notațiilor din text. Astfel, configurației $(q, \gamma, x, cale)$ îi va corespunde lista `(qcurent stiva x cale)` cu `top= (car stiva)`, `gammac=(cdr stiva)`, `scurent` - primul element al șirului `x` sau `""`, `x1` - restul șirului `x` sau `""`.

Pentru $\delta(q_i, Z, a) = (q_j, \gamma)$ avem notațiile:
`(qidelta zdelta sdelta qjdelta gammad)`. Prin `l,l1` vom nota componentele noii configurații.

Funcționarea automatului ar putea să cicleze. Se propune o metodă simplă de verificare specificând numărul maximal de iterații admisibile. Pentru aceasta se folosește variabila `*ITER*`.

În program se mai utilizează funcțiile ajutătoare $print - config$, $print - accept - impas$ și $print - as$ pentru imprimarea rezultatelor.

Prezentăm în continuare listingul programului cu rezultatele testării șirului $z =$ ”*aabbbbb*” pentru limbajul L_{ab23}.

Modelarea AS_V

```
;; Functia tranzitie(configuratie delta) verifica daca este posibila
;; trecerea directa din  configuratie aplicand tranzitia delta
(defun tranzitie(configuratie delta)
 (let*((qcurent (nth 0 configuratie))
   (stiva(nth 1 configuratie))
         (top (car stiva))
         (gammac (cdr stiva))
   (x(nth 2 configuratie))
         (scurent(if (equal x "") x (subseq x 0 1)))
         (x1(if(equal x "") x (subseq x 1)))
    (cale (nth 3 configuratie))
    (qidelta (nth 0 delta))
    (zdelta (nth 1 delta))
    (sdelta (nth 2 delta))
    (qjdelta (nth 3 delta))
    (gammad (cddddr delta))
    (l(list qjdelta(remove nil(append gammad(cdr(nth 1
      configuratie))))))
    (l1(cons(cons(position delta *AS* :test #'equal) cale)nil)))
    (cond((null top)(if(equal "" x)(list "accept" configuratie)))
      ((and(equal qcurent qidelta)(equal top zdelta))
      (cond((equal sdelta scurent)(list "valid"(append l(cons x1
      nil)l1)))
         ((equal sdelta "")(list "valid"(append l(cons x nil) l1)))))))
)

;; Functia configuratii gestioneaza integral procesul de modelare
(defun configuratii(&aux noi ncf flagi flaga (iteratii 0))
  (loop while *NOI* do
   (setq noi nil)(incf iteratii)
   (dolist(configuratie *NOI*) do
    (setq flagi t flaga nil)
    (dolist(delta *AS*)
     (setq ncf(tranzitie configuratie delta))
     (when(> iteratii *ITER*)(return-from configuratii
      (format t "~%Numarul de iteratii depaseste limita admisibila=~S"
```

```
     *ITER*)))
    (cond((equal "valid" (car ncf))(push (cadr ncf) noi)(setq flagi nil))
         ((equal "accept"(car ncf))(setq flaga t flagi nil))))
    (if flagi (push configuratie *IMPAS*))
    (if flaga (pushnew configuratie *ACCEPT* :test #'equal)))
  (setq *NOI* noi))
)

;; Functia print-config tipareste o lista de configuratii
(defun print-config(lconfig)
  (dolist (conf lconfig)
    (format t "~%(~S ~S ~S ~S)"
       (car conf)(cadr conf)(caddr conf)(reverse(cadddr conf))))
)

;; Functia print-accept-impas tipareste listele configuratiilor de
;; acceptare *ACCEPT* si de impas *IMPAS*
(defun print-accept-impas(z)
  (cond((null *ACCEPT*)(format t "~%Sir neacceptat z=~S" z))
       (t(format t "~%Sir acceptat z=~S" z)
         (format t "~%Configuratii de acceptare pentru z=~S" z)
         (print-config *ACCEPT*)))
 (format t "~%Configuratii de impas pentru z=~S" z)
 (print-config *IMPAS*)
)

;; Functia print-as tipareste *AS*
(defun print-as()
 (format t "~%*AS*=")
 (dolist (el *AS*)
  (format t "~%   ~S. ~S"  (position el *AS* :test #'equal)el))
)

;; Exemplu, AS linearizat pentru Lab23
(setq *AS* '((q0 $ "" q3)
             (q0 $ "a" q1 A A A $)
             (q0 $ "a" q1 A A $)
             (q1 A "a" q1 A A A A)
```

```
          (q1 A "a" q1 A A A)
          (q1 A "" q2 A)
          (q2 A "b" q2)
          (q2 A "b" q3)
          (q3 $ "" q3)))


(setq z "aabbbbb")
(setq *ITER* 100)
(setq *IMPAS* nil)
(setq *ACCEPT* nil)
(setq *NOI* (list(append '(q0 ($)) (cons z nil) '(nil))))
(configuratii)
(print-as)
(print-accept-impas z)

;; Rezultate, modelarea functionarii AS pentru Lab23

*AS*=
  0. (Q0 $ "" Q3)
  1. (Q0 $ "a" Q1 A A A $)
  2. (Q0 $ "a" Q1 A A $)
  3. (Q1 A "a" Q1 A A A A)
  4. (Q1 A "a" Q1 A A A)
  5. (Q1 A "" Q2 A)
  6. (Q2 A "b" Q2)
  7. (Q2 A "b" Q3)
  8. (Q3 $ "" Q3)

Sir acceptat z="aabbbbb"
Configuratii de acceptare pentru z="aabbbbb"
(Q3 NIL "" (1 4 5 6 6 6 6 7 8))
(Q3 NIL "" (2 3 5 6 6 6 6 7 8))
Configuratii de impas pentru z="aabbbbb"
(Q2 ($) "" (2 3 5 6 6 6 6 6))
(Q3 NIL "b" (2 4 5 6 6 6 7 8))
(Q2 (A $) "" (1 3 5 6 6 6 6 6))
(Q3 (A $) "" (1 3 5 6 6 6 6 7))
```

```
(Q2 ($) "" (1 4 5 6 6 6 6 6))
(Q3 (A $) "b" (1 4 5 6 6 6 7))
(Q3 (A A $) "b" (1 3 5 6 6 6 7))
(Q2 ($) "b" (2 4 5 6 6 6 6))
(Q3 (A $) "b" (2 3 5 6 6 6 7))
(Q3 (A A $) "bb" (2 3 5 6 6 7))
(Q3 (A $) "bb" (2 4 5 6 6 7))
(Q3 (A A A $) "bb" (1 3 5 6 6 7))
(Q3 (A A $) "bb" (1 4 5 6 6 7))
(Q3 (A A A $) "bbb" (1 4 5 6 7))
(Q3 (A A A A $) "bbb" (1 3 5 6 7))
(Q3 (A A $) "bbb" (2 4 5 6 7))
(Q3 (A A A $) "bbb" (2 3 5 6 7))
(Q3 (A A A A $) "bbbb" (2 3 5 7))
(Q3 (A A A $) "bbbb" (2 4 5 7))
(Q3 (A A A A A $) "bbbb" (1 3 5 7))
(Q3 (A A A A $) "bbbb" (1 4 5 7))
(Q2 (A A $) "abbbbb" (2 5))
(Q2 (A A A $) "abbbbb" (1 5))
(Q3 NIL "aabbbbb" (0))
```

Algoritmul ASG

Algoritmul **ASG** transformă *ASV* în gramatică independentă de context echivalentă în conformitate cu algoritmul descris în paragraful 7. *ASV* se va lua în Formă Normală linearizat. Descriem în continuare realizarea algoritmului comentând funcțiile utilizate și unele exemple.

- Deoarece funcțiile primitive `adjoin, remove, position, set- -difference` utilizează opțiunea de test `eql`, iar aplicațiile noastre operează cu subliste și șiruri de caractere, le vom extinde cu opțiunea `equal`.
- Funcția `cartesianasg` va genera produsul cartezian `*Q*`$\times$`*Q*`.
- Funcția `prodx1x2(qi x a qj x1 x2)` generează toate producțiile posibile $[q_iXq_2] \to a[q_jX_1q_1][q_1X_2q_2]$ pentru toți $\delta(q_i, X, a) \ni (q_j, X_1X_2)\}$, toți $(q_1, q_2) \in$ `*Q*`$\times$`*Q*`. În program producția se va

reprezenta ca `((Qi X Q2) ("a") (Qj X1 Q1) (Q1 X2 Q2))`.

- Funcția `prodx1(qi x a qj x1)` generează toate producțiile posibile $[q_iXq_1] \to a[q_jX_1q_1]$ pentru toți $\delta(q_i, X, a) \ni (q_j, X_1)$, toți $q_1 \in$ `*Q*`.
- Funcția `genergic` este funcția principală care construiește gramatica apelând iterativ funcțiile `prodx1x2(qi x a qj x1 x2)` și `prodx1(qi x a qj x1)`. De rând cu aceasta funcția `genergic`:
  - construiește mulțimea `*Q*`,
  - generează producțiile $S \to [q_0\$q_j]$ pentru toți $q_j \in$ `*Q*`,
  - elimină șirurile vide `""` (a=`""`) din producțiile
    $[q_iXq_2] \to ""[q_jX_1q_1][q_1X_2q_2]$ și $[q_iXq_1] \to ""[q_jX_1q_1]$.
- Funcția `vn-vt` calculează mulțimile simbolurilor terminale și neterminale: `*VT*` și `*VN*`.
- Funcția `codificare-gensym` generează utilizând funcția primitivă `gensym` o mulțime de simboluri neterminale noi `*VNGSYM*`=`(A1 A2...AN)`. Astfel, fiecărui element $[q_iXq_j]$ din `*VN*`, i se pune în în corespondență un element din `*VNGSYM*`. În continuare codifică gramatica `*GIC*` substituind simbolurile $[q_iXq_j]$ prin simbolurile corespunzătoare din `*VNGSYM*`. Gramatica astfel codificată se va nota prin `*NGIC*`. Axioma `S` va rămâne neschimbată.
- Funcția `prod-nprod` calculează simbolurile productive `*PROD*` și simbolurile neproductive `*NPROD*`. Simbolurile terminale se includ în `*PROD*`.
- Funcția `simplificare-gic` modifică gramatica `*NGIC*` eliminând simbolurile neproductive.
- Funcția `alpha-gic` substituie elementele neterminale din `*NGIC*` cu litere din `*ALPHA*`= `(s a b c d ... )`, obținând o reprezentare mai obișnuită pentru gramatică utilizată pentru simplificări ulterioare. Litera `s` se va rezerva pentru axiomă. Gramatica astfel obținută se notează prin `*NNGIC*`. Funcția nu se va apela dacă numărul de elemente neterminale din `*VNGSYM*` depășește numărul de elemente din `*ALPHA*`
- Funcția `asgmain` întrunește apelurile tuturor funcțiilor expuse mai sus, dar și imprimarea rezultatelor intermediare după fiecare apel.

Inserăm în continuare listingul programului și un exemplu.

Convertirea AS_V în GIC

```
;; Functii auxiliare
(defun adjoinq(element lst)(adjoin element lst :test #'equal))
(defun removeq(element lst)(remove element lst :test #'equal))
(defun positionq(element lst)(position element lst :test #'equal))
(defun set−differenceq(lst1 lst2)(set−difference lst1 lst2 :test #'equal))

;; Functia cartesianasg calculeaza produsul cartesian A X A
(defun cartesianasg(A)
  (loop for e1 in A append
    (loop for e2 in A collect(list e1 e2)))
)

;; Functia prodx1x2 genereaza productii pentru delta(qi,x,a)=
;; (qj,x1 x2) conform algoritmului ASG
(defun prodx1x2(qi x a qj x1 x2)
 (dolist(comb (cartesianasg *Q*))
  (let((q1(car comb))(q2(cadr comb)))
    (push(list (list qi x q1)(cons a nil)(list qj x1 q2)(list q2 x2
      q1))*GIC*)))
)

;; Functia prodz1 genereaza productii pentru delta(qi,x,a)=
;; (qj,x1) conform algoritmului ASG
(defun prodx1(qi x a qj x1)
  (dolist (q1 *Q*)
    (push(list(list qi x q1) (cons a nil) (list qj x1 q1))*GIC*))
)

;; Functia genergic genereaza gramatica *GIC*, algoritmul ASG
(defun genergic()
  (setq *GIC* nil)
  (setq *Q* (remove−duplicates
       (loop for el in *AS* append(list(car el)(cadddr el)))))
  (dolist (q *Q*)
    (push(list '(S)(append '(q0 $)(cons q nil)))*GIC*)
```

```
  )
  (dolist(tr (reverse *AS*))
    (let((qi (nth 0 tr))(x (nth 1 tr))(a (nth 2 tr))
         (qj (nth 3 tr))(x1 (nth 4 tr))(x2 (nth 5 tr)))
    (cond((eq(length tr) 4)(push(list(list qi x qj)(list a)) *GIC*))
         ((eq(length tr) 5)(prodx1 qi x a qj x1))
         ((eq(length tr) 6)(prodx1x2 qi x a qj x1 x2)))))
    (setq *GIC* (loop for el in *GIC* append
        (list(if(>(length el)2)(removeq '("")el)el))))
)

;;  Functia vn-vt calculeaza multimile simbolurilor terminale
;;  si neterminale: *VT* si *VN*
(defun vn-vt()
 (setq *VN* nil *VT* nil)
 (dolist(prod *GIC*)
   (dolist(el prod)
    (if(stringp (car el))(setq *VT*(adjoinq (car el) *VT*))
                         (setq *VN*(adjoinq el *VN*)))))
)

;; Functia codificare-gensym codifica elementele neterminale prin
;; S,A1, A2, A3,...
(defun codificare-gensym ()
 (setq *gensym-counter* 1)
 (setq *VNGSYM* (cons 'S
  (loop for el in *VN* collect
      (read-from-string (symbol-name (gensym "A"))))))
 (setq *GICSYM*
    (loop for el in (reverse *GIC*) collect
     (loop for e in el collect
       (if(stringp(car e))(car e)(nth(positionq e *VN*)*VNGSYM*)))))
)

;; Functia prod-nprod calculeaza multimile simbolurilor productive
;; si neproductive: *PROD* si *NPROD*
(defun prod-nprod()
 (setq *PROD* *VT* *NPROD* nil)
```

```
 (let((flag t)(temp))
  (loop while flag do
   (setq flag nil)
   (dolist(prod *GICSYM*) (setq temp *PROD*)
     (let((x0(car prod))(xc(cdr prod)))
       (when(not(set-differenceq xc *PROD*))
               (setq temp (adjoinq x0 *PROD*)))
      (when(not(equal temp *PROD*))(setq *PROD* temp)(setq flag
      t)))
    )))
 (setq *NPROD* (set-differenceq *VNGSYM* *PROD*))
)

;; Functia simplificare-gic elimina simbolurile neproductive.
;; *NGIC* - gramatica simplificata
(defun simplificare-gic()
  (setq *NGIC* nil)
  (dolist(prod *GICSYM*)
    (when(not(set-differenceq prod *PROD*))
             (setq *NGIC* (adjoinq prod *NGIC*))))
  (setq *NGIC* (reverse *NGIC*))
)

;; Functia alpha-gic substituie in *NGIC* neterminalele prin litere
;; din *ALPHA*. Rezultatul va fi *NNGIC*
(defun alpha-gic()
 (setq *NNGIC* nil)
 (dolist (el *NGIC*)
  (let((xel))
   (dolist(x el)
     (cond((equal x 'S)(push 'S xel))
          ((stringp x)(push x xel))
          (t (push(nth(positionq x *PROD*)*ALPHA*)xel))))
   (push (reverse xel) *NNGIC*)))
 (setq *NNGIC* (reverse *NNGIC*))
)

;; Functia asgmain intruneste apelurile tuturor functiilor expuse
```

```
;; mai sus, asigura imprimarea rezultatelor intermediare
;; dupa fiecare apel.
(defun asgmain()
 (genergic)
 (format t "~%Productii generate - ~S" (length *GIC*))
 (dolist (el *GIC*)
   (format t "~%~S~A ~S ~A ~S"
     (position el *GIC*) "." (car el) "-->" (cdr el)))

 (vn-vt)

 (codificare-gensym)
 (format t "~%Productii codificate - ~S" (length *GICSYM*))
 (dolist (el *GICSYM*)
   (format t "~%~S.~S"(position el *GICSYM*)el))
 (format t "~%Simboluri neterminale - ~S
      ~&*VN*=~S
      ~&Simboluri terminale - ~S
      ~&*VT*=~S"
      (length *VNGSYM*) *VNGSYM* (length *VT*) *VT*)

(prod-nprod)
 (if(null(member 'S *PROD* :test #'equal))
    (return "Axioma neproductiva. Limbajul generat este vid")
      (setq *PROD*(append '(S)(remove 'S *PROD* :test #'equal))))
 (format t "~%Simboluri productive+terminale - ~S
           ~&*PROD*=~S
           ~%Simboluri neproductive - ~S
           ~%*NPROD*=~S"
       (length *PROD*) *PROD* (length *NPROD*) *NPROD*)

(simplificare-gic)
 (format t "~%Gramatica simplificata - ~S productii" (length
      *NGIC*))
 (dolist (el *NGIC*)
  (format t "~%~S. ~S" (position el *NGIC*)el) )

 (setq *ALPHA* '(s a b c d e f g h i j k l m n o p q r t u v w x y z
```

```
    a1 b1 c1 d1 e1 f1 g1 h1 i1 j1 k1 l1 m1 n1 o1 p1 q1 r1 s1 t1 u1 v1 w1
      x1 y1 z1
    a2 b2 c2 d2 e2 f2 g2 h2 i2 j2 k2 l2 m2 n2 o2 p2 q2 r2 s2 t2 u2 v2 w2
      x2 y2 z2))
 (cond((<(length *PROD*)(length *ALPHA*))
      (alpha-gic)
      (format t "~%Gramatica alpha-simplificata - ~S productii"
      (length *NNGIC*))
      (dolist (el *NNGIC*)
      (format t "~%~S. ~S" (position el *NNGIC*) (if (>(length
      el)2)(removeq "" el)el))))
  (t(format t "~%Mai multe neterminale decat simboluri in
      *ALPHA*")) )
) ;   end asgmain

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;  Exemplu, convertirea ASi2jk in GIC echivalenta
(setq *AS* '(
        (Q0 $ "" Q1 C $)
        (Q1 A "b" Q2 B A)
        (Q1 A "a" Q1 A A)
        (Q1 C "a" Q1 A C)
        (Q2 B "b" Q3)
        (Q2 B "b" Q2 B B)
        (Q3 A "c" Q4)
        (Q3 B "b" Q3)
        (Q3 A "c" Q5)
        (Q4 A "c" Q5)
        (Q4 A "c" Q4)
        (Q5 $ "" Q5)
        (Q5 B "" Q5)
        (Q5 A "" Q5)
        (Q5 C "" Q5)))
(asgmain)

Productii generate - 196
Simboluri neterminale - 145
Simboluri terminale - 4
```

```
*VT*=("" "c" "a" "b")
Simboluri productive+terminale - 19
Simboluri neproductive - 131
Gramatica simplificata - 19 productii
0. (S A144)
1. (A60 "")
2. (A100 "")
3. (A30 "")
4. (A142 "")
5. (A88 "c")
6. (A98 "c")
7. (A96 "c")
8. (A18 "b")
9. (A87 "c")
10. (A97 "b" A97 A18)
11. (A97 "b")
12. (A143 "a" A102 A60)
13. (A90 "a" A90 A88)
14. (A102 "a" A90 A98)
15. (A102 "a" A102 A100)
16. (A90 "b" A97 A87)
17. (A102 "b" A97 A96)
18. (A144 A143 A142)
Gramatica alpha-simplificata - 19 productii
0. (S A)
1. (N "")
2. (M "")
3. (L "")
4. (K "")
5. (J "c")
6. (I "c")
7. (H "c")
8. (G "b")
9. (F "c")
10. (E "b" E G)
11. (E "b")
12. (B "a" C N)
13. (D "a" D J)
```

```
14. (C "a" D I)
15. (C "a" C M)
16. (D "b" E F)
17. (C "b" E H)
18. (A B K)
```

Drept exemplu am luat automatul (versiunea AS_V obținută la convertirea AS_F în AS_V) care recunoaște limbajul $L_{i2jk} = \{a^i b^{2j} c^k | i \geq k \geq 1, j \geq 1\}$. În Figura 22 inserăm gramatica obținută în rezultatul convertirii și simplificată prin eliminarea simbolurile neproductive.

$$\begin{array}{l}
G_{i2jk} = (V_N, V_T, P, S), \\
V_N = \{S, A, B, C, D, E, F, H, I, J, K, L, M, N\}, \\
V_T = \{a, b, c\},\ P = \{ \\
\begin{array}{llll}
0.\ S \to A & 1.\ N \to \varepsilon & 2.\ M \to \varepsilon & 3.\ L \to \varepsilon \\
4.\ K \to \varepsilon & 5.\ J \to c & 6.\ I \to c & 7.\ H \to c \\
8.\ G \to b & 9.\ F \to c & 10.\ E \to bEG & 11.\ E \to b \\
12.\ B \to aCN & 13.\ D \to aDJ & 14.\ C \to aDI & 15.\ C \to aCM \\
16.\ D \to bEF & 17.\ C \to bEH & 18.\ A \to BK & \}
\end{array}
\end{array}$$

Figura 22: Gramatica G_{i2jk}.

După eliminarea ε-producțiilor și substituirea neterminalelor de unică folosință obținem gramatica din Figura 23 (a), iar după factorizarea producțiilor comune pentru neterminalele C și D, obținem gramatica din Figura 23 (b).

Să ne convingem că $L(G) = L_{i2jk}$. Mai întâi observăm că $E \overset{*}{\Longrightarrow} b^{2j+1}, j \geq 0$. Pentru neterminaul D obținem:

1) $D \Longrightarrow bEc \overset{*}{\Longrightarrow} bb^{2j+1}c, j \geq 0$ sau $D \overset{*}{\Longrightarrow} b^{2j}c, j \geq 1$.

2) $D \Longrightarrow aDc \overset{*}{\Longrightarrow} a^k D c^k \overset{*}{\Longrightarrow} a^k b^{2j} c c^k = a^k b^{2j} c^{k+1}, k \geq 1, j \geq 1$.

Îmbinând 1) cu 2) obținem: $D \overset{*}{\Longrightarrow} a^k b^{2j} c^{k+1}, k \geq 0, j \geq 1$.

Pentru neterminalul C obținem:

(a)	(b)
0. $S \to aC$	0. $S \to aC$
1. $C \to aC$	1. $C \to aC$
2. $C \to aDc$	2. $C \to D$
3. $C \to bEc$	3. $D \to aDc$
4. $D \to aDc$	4. $D \to bEc$
5. $D \to bEc$	5. $E \to bEb$
6. $E \to bEb$	6. $E \to b$
7. $E \to b$	

Figura 23: Transformări echivalente asupra gramaticii G_{i2jk}

1) $C \Longrightarrow D \stackrel{*}{\Longrightarrow} a^k b^{2j} c^{k+1}, k \geq 0, j \geq 1.$

2) $C \Longrightarrow aC \stackrel{*}{\Longrightarrow} a^i C \stackrel{*}{\Longrightarrow} a^i a^k b^{2j} c^{k+1} = a^{i+k} b^{2j} c^{k+1}, i \geq 1, j \geq 1, k \geq 0.$

Îmbinând 1) cu 2) obținem: $C \stackrel{*}{\Longrightarrow} a^{i+k} b^{2j} c^{k+1}, i \geq 0, j \geq 1, k \geq 0$. În final, $S \Longrightarrow aC \stackrel{*}{\Longrightarrow} aa^{i+k} b^{2j} c^{k+1} = a^{i+k+1} b^{2j} c^{k+1}, i \geq 0, j \geq 1, k \geq 0$ sau $S \stackrel{*}{\Longrightarrow} a^{i+k} b^{2j} c^k, i \geq 0, j \geq 1, k \geq 1$. Evident, $i + k \geq k$.

10. Notițe bibliografice

Automatele cu memorie stivă pentru prima dată au fost menționate în lucrarea [14], fiind propuse ca mecanism pentru implementarea analizoarelor sintactice. Aplicațiii analogice sunt expuse și în lucrarea [17]. Echivalența gramaticilor independente de context și a automatelor cu memorie stivă a fost demonstrată în [15, 16]. Alte (mai multe) aspecte, proprietăți și exerciții pentru AS întâlnim în [1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 13, 9, 12]

11. Probleme și exerciții

1. Fie x un șir arbitrar peste $\Sigma=\{a,b\}$, $x\in\Sigma^*$. Să notăm prin $n_a(x)$ numărul de simboluri "a", iar prin $n_b(x)$ - numărul de simboluri "b" din x. Să se construiască AS care acceptă prin stivă vidă limbajul:
   (a) $L_{ba1}=\{x|x\in\{a,b\}^*, n_b(x)=n_a(x)+1\}$
   (b) $L_{b2a}=\{x|x\in\{a,b\}^*, n_b(x)=2n_a(x)\}$
   (c) $L_{b3a}=\{x|x\in\{a,b\}^*, n_b(x)=3n_a(x)\}$
   (d) $L_{2ab3a}=\{x|x\in\{a,b\}^*, 2n_a(x)\le n_b(x)\le 3n_a(x)\}$
2. Să se construiască AS care acceptă prin stivă vidă limbajul: $L_{nm}=\{x|x\in\{a,b\}^*, x=a^nb^m, n\ge 0, 2n\le m\le 3n$.
3. Pentru automatul definit mai jos să se construiască automatul echivalent în formă atomară.
   $AS=(Q,\Sigma,\Gamma,\delta,q_0,\$)$,
   $Q=\{q_0,q_1,q_2,q_3\},\Sigma=\{a,b\},\Gamma=\{\$,A\}$,
   $\delta(q_0,\$,a)=\{(q_1,AAA\$)\}$, $\delta(q_0,\$,\varepsilon)=\{(q_3,\varepsilon)\}$,
   $\delta(q_1,A,a)=\{(q_1,AAAA)\}$, $\delta(q_1,A,b)=\{(q_2,\varepsilon)\}$,
   $\delta(q_2,A,b)=\{(q_2,\varepsilon),(q_3,\varepsilon)\}$, $\delta(q_3,A,\varepsilon)=\{(q_3,\varepsilon)\}$,
   $\delta(q_3,\$,\varepsilon)=\{(q_3,\varepsilon)\}$.
4. (a) Să se construiască algoritmul de convertire a AS_V în AS_F echivalent.
   (b) Să se construiască aplicând acest algoritm AS_F pentru automatul:
   $AS=(Q,\Sigma,\Gamma,\delta,q_0,\$)$, $Q=\{q_0,q_1,q_2\},\Sigma=\{a,b\},\Gamma=\{\$,A\}$,
   $\delta(q_0,\$,a)=\{(q_0,A)\}$, $\delta(q_0,A,a)=\{(q_0,AA)\}$,
   $\delta(q_0,A,b)=\{(q_1,\varepsilon))\}$, $\delta(q_1,A,b)=\{(q_1,\varepsilon)\}$,
   $\delta(q_1,A,\varepsilon)=\{(q_1,\varepsilon))\}$.
5. (a) Pentru AS_V definit mai jos să se construiască, aplicând algoritmul ASG, gramatica echivalentă.
   $AS=(Q,\Sigma,\Gamma,\delta,q_0,\$)$, $Q=\{q_0\},\Sigma=\{a,b\},\Gamma=\{\$,A\}$,
   $\delta(q_0,\$,a)=\{(q_0,AAA\$)\}$, $\delta(q_0,A,b)=\{(q_0,\varepsilon)\}$,
   $\delta(q_0,\$,\varepsilon)=\{(q_0,\varepsilon)\}$.

   (b) Să se simplifice gramatica construită și să se determine structura limbajului generat.
6. Să se construiască AS care acceptă limbajul:
   $L_{i<>j} = \{x | x \in \{a, b\}^*, x = a^i b^j, i \geq 0, j \geq 0, i \neq j$.
7. (a) Să se construiască AS determinist cu acceptare prin stiva vidă care recunoaște limbajul:
   $L_{123} = \{ba^i b^i | i \geq 1\} \cup \{bba^i b^{2i} | i \geq 1\} \cup \{bbba^i b^{3i} | i \geq 1\}$.
   Să se reducă acest automat la Forma Normală.
   (b) Să se construiască gramatica independentă de context care generează limbajul L_{123}.

8. (a) Să se construiască, aplicând algoritmul GAS, AS care acceptă limbajul expresiilor aritmetice generate de gramatica:
   $G = (V_N, V_T, P, S), V_N = \{S, E, T, F\}$,
   $V_T = \{a, +, *, (,), ;\}$, $P = \{$

$$
\begin{array}{lll}
0. S \to E; & 1. E \to E + T & 2. E \to T \\
3. T \to F & 4. T \to T * F & 5. F \to a \\
6. F \to (E) \quad \} & &
\end{array}
$$

   Să se arate toate configurațiile de acceptare (impas) pentru expresiile "$a + a * a;$" și "$a + a)$".
   (b) Pentru același limbaj să se construiască AS determinist cu acceptare prin stivă vidă. Să se arate configurația de acceptare (impas) pentru expresiile "$a + a * a;$" și "$a + a)$".
9. Să se construiască AS cu acceptare prin stivă vidă pentru limbajul
   $L = \{x | x \in \{a, b\}^*, x = x_1 x_2, n_b(x_1) > n_a(x_1)\}$. Prin $n_a(x)$ am notat numărul de simboluri "a", iar prin $n_b(x)$ - numărul de simboluri "b" din x.
10. Este dat limbajul $L_{xx} = \{xx \mid x \in \{0, 1\}^*\}$. Să se construiască AS_{xx} cu acceptare prin stivă vidă pentru limbajul $\overline{L_{xx}}$, limbajul complementar pentru L_{xx}.

12. Lucrări practice

Lucrarea 1.

a) Construiţi automatul cu memorie stivă determinist (ASD) care acceptă limbajul L.

b) Pentru trei șiruri arbitrare x_1, x_2 și x_3 care aparțin limbajului $L(ASD)$ arătaţi secvenţele de acceptare $(q_0, \$, x_i) \overset{*}{\vdash} (q, \varepsilon, \varepsilon)$, $i = 1, 2, 3$.

1.1. $L = \{a^n bc^{2n-1} \mid n \geq 1\}$
1.2. $L = \{a^{n-1} b^2 c^{n+1} \mid n \geq 1\}$
1.3. $L = \{a^{2n-1} bc^n \mid n \geq 1\}$.
1.4. $L = \{a^{n-1} bdc^n \mid n \geq 1\}$.
1.5. $L = \{a^{3n} c^{n+3} \mid n \geq 1\}$.
1.6. $L = \{a^{2n-1} b^2 c \mid n \geq 1\}$.
1.7. $L = \{a^n bac^{3n} \mid n \geq 1\}$.
1.8. $L = \{a^{2n} bc^{2n-1} \mid n \geq 1\}$.
1.9. $L = \{b^{3n} c^{n+2} \mid n \geq 1\}$.
1.10. $L = \{a^n bc^{2n+1} \mid n \geq 1\}$.
1.11. $L = \{a^n bbc^{2n} \mid n \geq 1\}$.
1.12. $L = \{aba^n c^{2n} \mid n \geq 1\}$.
1.13. $L = \{b^{2n-1} ac^n \mid n \geq 1\}$.
1.14. $L = \{b^{3n} ad^n c \mid n \geq 1\}$.
1.15. $L = \{b^{3n} ca^{2n} \mid n \geq 1\}$.
1.16. $L = \{b^{2n+1} ac^{2n} \mid n \geq 1\}$.
1.17. $L = \{b^n ac^{2n+1} \mid n \geq 1\}$.
1.18. $L = \{b^n c^{3n} \mid n \geq 1\}$.
1.19. $L = \{b^{3n} cab^2 \mid n \geq 1\}$.
1.20. $L = \{b^{2n+1} ca^{n+2} \mid n \geq 1\}$.
1.21. $L = \{b^{n+3} aba^{n+2} \mid n \geq 1\}$.
1.22. $L = \{b^{3n+1} c^n \mid n \geq 1\}$.
1.23. $L = \{b^{2n-1} ca^{2n+1} \mid n \geq 1\}$.
1.24. $L = \{b^{n+1} cba^{2n+1} \mid n \geq 1\}$.

1.25. $L=\{b^{3n}c^{n+1} \mid n \geq 1\}$.

Lucrarea 2.

Este dat automatul AS cu acceptare prin stivă vidă.

a) Construiţi aplicând algoritmul ASG gramatica independentă de context G echivalentă.

b) Simplificaţi gramatica G şi determinaţi structura limbajului $L(G)$.

2.1. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1, q_2\}, \Sigma = \{a, b\}, \Gamma = \{\$, A\}$,
$\delta(q_0, \$, a)=\{(q_1, A)\}$, $\delta(q_1, A, a)=\{(q_1, AA)\}$,
$\delta(q_1, A, b)=\{(q_2, A)\}$, $\delta(q_2, A, a)=\{(q_2, \varepsilon)\}$.

2.2. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1\}, \Sigma = \{a, b, c\}, \Gamma = \{\$, A, B, C\}$,
$\delta(q_0, \$, a)=\{(q_0, A\$)$, $\delta(q_0, A, b)=\{(q_0, B)\}$,
$\delta(q_0, B, c)=\{(q_0, C)\}$, $\delta(q_0, C, a)=\{(q_1, \varepsilon)\}$,
$\delta(q_1, \$, a)=\{(q_1, \varepsilon), (q_1, \$)\}$.

2.3. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1, q_2\}, \Sigma = \{b, c\}, \Gamma = \{\$, B\}$,
$\delta(q_0, \$, b)=\{(q_1, BB)\}$, $\delta(q_1, B, b)=\{(q_1, BB)\}$,
$\delta(q_1, B, c)=\{(q_2, \varepsilon)\}$, $\delta(q_2, B, c)=\{(q_2, \varepsilon)\}$,

2.4. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1, q_2\}, \Sigma = \{a, b\}, \Gamma = \{\$, A\}$,
$\delta(q_0, \$, b)=\{(q_1, \$)\}$, $\delta(q_1, \$, a)=\{(q_1, A)\}$,
$\delta(q_1, A, a)=\{(q_1, AA)\}$, $\delta(q_1, A, b)=\{(q_2, A)\}$,
$\delta(q_2, A, a)=\{(q_2, \varepsilon)\}$,

2.5. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1\}, \Sigma = \{a, c\}, \Gamma = \{\$, A\}$,
$\delta(q_0, \$, a)=\{(q_0, A\$)\}$, $\delta(q_0, A, a)=\{(q_1, A)\}$,
$\delta(q_1, A, c)=\{(q_1, \varepsilon)\}$, $\delta(q_1, \$, a)=\{(q_1, \varepsilon), (q_1, \$)\}$,

2.6. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,

$Q = \{q_0, q_1, q_2\}, \Sigma = \{a, b, d\}, \Gamma = \{\$\}$,
$\delta(q_0, \$, a) = \{(q_1, \$), (q_3, \varepsilon)\}$, $\delta(q_1, \$, d) = \{(q_2, \$)\}$,
$\delta(q_2, \$, b) = \{(q_3, \varepsilon)\}$, $\delta(q_2, \$, d) = \{(q_3, \$)\}$,
$\delta(q_3, \$, b) = \{(q_3, \varepsilon)\}$,

2.7. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1, q_2\}, \Sigma = \{a, b\}, \Gamma = \{\$, A\}$,
$\delta(q_0, \$, a) = \{(q_0, A)\}$, $\delta(q_0, A, a) = \{(q_0, AA)\}$,
$\delta(q_0, A, b) = \{(q_1, \varepsilon)\}$, $\delta(q_1, A, b) = \{(q_1, \varepsilon), (q_2, \varepsilon)\}$,
$\delta(q_2, A, \varepsilon) = \{(q_2, \varepsilon)\}$.

2.8. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1\}, \Sigma = \{c, d\}, \Gamma = \{\$, A, C, D\}$,
$\delta(q_0, \$, d) = \{(q_0, D\$)\}$, $\delta(q_0, D, a) = \{(q_0, A)\}$,
$\delta(q_0, A, c) = \{(q_0, C)\}$, $\delta(q_0, C, d) = \{(q_1, \varepsilon)\}$,
$\delta(q_1, \$, d) = \{(q_1, \varepsilon), (q_1, \$)\}$.

2.9. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1, q_2\}, \Sigma = \{a, b, c\}, \Gamma = \{\$, A, B, C\}$,
$\delta(q_0, \$, a) = \{(q_0, A\$)\}$, $\delta(q_0, A, b) = \{(q_0, B)\}$,
$\delta(q_0, B, c) = \{(q_1, C), (q_2, \varepsilon)\}$, $\delta(q_1, C, a) = \{(q_0, A)\}$,
$\delta(q_2, \$, \varepsilon) = \{(q_2, \varepsilon)\}$.

2.10. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1\}, \Sigma = \{a, b\}, \Gamma = \{\$, A\}$,
$\delta(q_0, \$, a) = \{(q_0, A)\}$, $\delta(q_0, A, a) = \{(q_0, AA)\}$,
$\delta(q_0, A, b) = \{(q_1, \varepsilon), (q_1, \$)\}$, $\delta(q_1, A, b) = \{(q_1, \varepsilon)\}$,
$\delta(q_1, \$, b) = \{(q_1, \$), (q_1, \varepsilon)\}$.

2.11. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1\}, \Sigma = \{a, b, c\}, \Gamma = \{\$, A, B, C\}$,
$\delta(q_0, \$, b) = \{(q_0, B\$)\}$, $\delta(q_0, B, a) = \{(q_0, A)\}$,
$\delta(q_0, A, c) = \{(q_0, C)\}$, $\delta(q_0, C, b) = \{(q_1, \varepsilon)\}$,
$\delta(q_1, \$, b) = \{(q_1, \varepsilon), (q_1, \$)\}$,

2.12. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1\}, \Sigma = \{a, b\}, \Gamma = \{\$, A, B\}$,

$\delta(q_0, \$, a) = \{(q_0, A\$)\}$, $\delta(q_0, A, b) = \{(q_1, B), (q_1, \varepsilon)\}$,
$\delta(q_1, B, a) = \{(q_0, A)\}$, $\delta(q_1, \$, \varepsilon) = \{(q_1, \varepsilon)\}$.

2.13. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1\}, \Sigma = \{a, b, d\}, \Gamma = \{\$, A, B, D\}$,
$\delta(q_0, \$, d) = \{(q_0, D)\}$, $\delta(q_0, \$, a) = \{(q_1, AB\}$,
$\delta(q_0, D, d) = \{(q_0, \varepsilon), (q_0, \$)\}$, $\delta(q_1, A, a) = \{(q_1, \varepsilon)\}$,
$\delta(q_1, B, b) = \{(q_1, \varepsilon)\}$.

2.14. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1\}, \Sigma = \{a, b, c\}, \Gamma = \{\$, A, B\}$.
$\delta(q_0, \$, a) = \{(q_0, A\$)\}$, $\delta(q_0, A, b) = \{(q_1, \varepsilon)\}$,
$\delta(q_1, \$, b) = \{(q_1, \$)\}$, $\delta(q_1, \$, c) = \{(q_1, \$)\}$,
$\delta(q_1, \$, a) = \{(q_1, \varepsilon)\}$.

2.15. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1\}, \Sigma = \{a, b\}, \Gamma = \{\$, A, B\}$,
$\delta(q_0, \$, a) = \{(q_0, A\$)\}$, $\delta(q_0, A, a) = \{(q_1, \varepsilon)\}$,
$\delta(q_1, \$, b) = \{(q_1, B)\}$, $\delta(q_1, B, b) = \{(q_1, B), (q_1, \varepsilon)\}$.

2.16. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1, q_2\}, \Sigma = \{a, b, d\}, \Gamma = \{\$, D\}$,
$\delta(q_0, \$, a) = \{(q_1, \$), (q_1, \varepsilon)\}$, $\delta(q_1, \$, d) = \{(q_1, D)$,
$\delta(q_1, D, d) = \{(q_1, DD)\}$, $\delta(q_1, D, b) = \{(q_2, \varepsilon)\}$,
$\delta(q_2, D, b) = \{(q_2, \varepsilon)\}$.

2.17. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1\}, \Sigma = \{a, c, d\}, \Gamma = \{\$, A\}$,
$\delta(q_0, \$, a) = \{(q_0, A\$)\}$, $\delta(q_0, A, d) = \{(q_1, \varepsilon)\}$,
$\delta(q_1, \$, d) = \{(q_1, \$)\}$, $\delta(q_1, \$, c) = \{(q_1, \$)\}$,
$\delta(q_1, \$, a) = \{(q_1, \varepsilon)\}$.

2.18. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1\}, \Sigma = \{a, b, c\}, \Gamma = \{\$, A, C\}$,
$\delta(q_0, \$, a) = \{(q_0, A\$)\}$, $\delta(q_0, A, a) = \{(q_1, \varepsilon)\}$,
$\delta(q_1, \$, b) = \{(q_1, \$)\}$, $\delta(q_1, \$, c) = \{(q_1, C)\}$,
$\delta(q_1, C, c) = \{(q_1, \varepsilon)\}$.

2.19. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1, q_2\}, \Sigma = \{a, b, c\}, \Gamma = \{\$, A\}$,
$\delta(q_0, \$, a) = \{(q_1, A)\}$, $\delta(q_1, A, a) = \{(q_1, AA)\}$,
$\delta(q_1, A, b) = \{(q_2, AA)\}$, $\delta(q_2, A, c) = \{(q_2, \varepsilon)\}$.

2.20. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1\}, \Sigma = \{b, c, d\}, \Gamma = \{\$, B, D\}$,
$\delta(q_0, \$, b) = \{(q_0, B\$\},)$ $\delta(q_0, B, d) = \{(q_0, D)$,
$\delta(q_0, D, d) = \{(q_0, D)$, $\delta(q_0, D, c) = \{(q_1, \varepsilon)\}$,
$\delta(q_1, \$, b) = \{(q_1, \varepsilon)\}$.

2.21. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1\}, \Sigma = \{a, b, c, d\}, \Gamma = \{\$, B\}$,
$\delta(q_0, \$, b) = \{(q_0, B\$\}$, $\delta(q_0, B, d) = \{(q_1, \varepsilon)\}$,
$\delta(q_1, \$, d) = \{(q_1, \$)\}$, $\delta(q_1, \$, c) = \{(q_1, \$)\}$,
$\delta(q_1, \$, a) = \{(q_1, \varepsilon)\}$.

2.22. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1, q_2\}, \Sigma = \{a, b\}, \Gamma = \{\$, A\}$,
$\delta(q_0, \$, a) = \{(q_1, \$)\}$, $\delta(q_1, \$, a) = \{(q_1, A)\}$,
$\delta(q_1, A, a) = \{(q_1, AA)\}$, $\delta(q_1, A, b) = \{(q_2, \varepsilon)\}$,
$\delta(q_2, A, b) = \{(q_2, \varepsilon)\}$.

2.23. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1\}, \Sigma = \{a, c, d\}, \Gamma = \{\$, A, D\}$,
$\delta(q_0, \$, a) = \{(q_0, A\$)\}$, $\delta(q_0, A, d) = \{(q_0, D)$,
$\delta(q_0, D, d) = \{(q_0, D)$, $\delta(q_0, D, c) = \{(q_1, \varepsilon)\}$,
$\delta(q_1, \$, a) = \{(q_1, \varepsilon)\}$.

2.24. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1\}, \Sigma = \{a, b, c\}, \Gamma = \{\$, B\}$,
$\delta(q_0, \$, b) = \{(q_0, B\$\}$, $\delta(q_0, B, a) = \{(q_1, \varepsilon)\}$,
$\delta(q_1, \$, a) = \{(q_1, \$)\}$, $\delta(q_1, \$, c) = \{(q_1, \$)\}$,
$\delta(q_1, \$, b) = \{(q_1, \varepsilon)\}$.

2.25. $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1, q_2\}, \Sigma = \{a, b\}, \Gamma = \{\$, A\}$,

$\delta(q_0,\$,a)=\{(q_0,A\$)\}$, $\delta(q_0,A,a)=\{(q_1,\varepsilon)\}$,
$\delta(q_1,\$,b)=\{(q_2,A)\}$, $\delta(q_2,A,b)=\{(q_1,\$)\}$,
$\delta(q_1,\$,\varepsilon)=\{(q_1,\varepsilon)\}$.

13. Soluții, indicații, răspunsuri

1.(a) $L_{ba1}=\{x \mid x \in \{a,b\}^*, n_b(x)=n_a(x)+1\}$. Vom folosi modelul automatului AS_{01} construit în Exemplul 4.1 care acceptă limbajul $L_{01}=\{x|x \in \{0,1\}^*, n_0(x)=n_1(x)\}$. Obținem automatul:
$AS_{ba1}=(Q,\Sigma,\Gamma,\delta,q_0,\$)$,
$Q=\{q_0,q_1\}, \Sigma=\{a,b\}, \Gamma=\{\$,A,B\}$,
$\delta(q_0,\$,a)=\{(q_0,A\$)\}$, $\delta(q_0,\$,b)=\{(q_0,B\$)\}$,
$\delta(q_0,A,a)=\{(q_0,AA)\}$, $\delta(q_0,A,b)=\{(q_0,\varepsilon)\}$,
$\delta(q_0,B,b)=\{(q_0,BB)\}$, $\delta(q_0,B,a)=\{(q_0,\varepsilon)\}$,
$\delta(q_0,B,\varepsilon)=\{(q_1,\varepsilon)\}$, $\delta(q_1,\$,\varepsilon)=\{(q_1,\varepsilon)\}$.

Spre deosebie de AS_{01}, orice șir acceptat de AS_{ba1} trebuie să conțină cu un simbol "b" mai mult decât "a". Astfel, AS_{ba1} pentru orice șir corect va ajunje în configurția $(q_0,B\$,\varepsilon)$ și va accepta prin ε-tranzițiile: $(q_0,B\$,\varepsilon) \vdash (q_1,\$,\varepsilon) \vdash (q_1,\varepsilon,\varepsilon)$.
Menționăm că automatul va accepta și șirul $x=b$ $(n_a(x)=0)$:
$(q_0,\$,b) \vdash (q_0,B\$,\varepsilon) \vdash (q_1,\$,\varepsilon) \vdash (q_1,\varepsilon,\varepsilon)$.

(b) $L_{b2a}=\{x|x \in \{a,b\}^*, n_a(x) \geq 0,\ n_b(x)=2n_a(x)\}$. Vom folosi ideile automatului din Exemplul 4.1. Dacă fiecare "a" de pe bandă se va substitui prin "aa", problema se reduce la 4.1. Acest procedeu se va realiza urmărind acțiunile cu stiva. Pentru fiecare configurație corectă trebuie să se îndeplinească condiția: n_A(stivă)$+2n_a$(bandă)$=n_B$(stivă)$+n_b$(bandă). Inserăm mai jos o variantă a automatului.
$AS_{b2a}=(Q,\Sigma,\Gamma,\delta,q_0,\$)$,
$Q=\{q_0,q_1\}, \Sigma=\{a,b\}, \Gamma=\{\$,A,B\}$,
$\delta(q_0,\$,\varepsilon)=\{(q_0,\varepsilon)\}$, $\delta(q_0,\$,a)=\{(q_0,AA\$)\}$,
$\delta(q_0,\$,b)=\{(q_0,B\$)\}$, $\delta(q_0,\$,\varepsilon)=\{(q_0,\varepsilon)\}$,
$\delta(q_0,A,a)=\{(q_0,AAA)\}$, $\delta(q_0,A,b)=\{(q_0,\varepsilon)\}$,

$\delta(q_0, B, b) = \{(q_0, BB)\}$, $\delta(q_0, B, a) = \{(q_1, \varepsilon)\}$,
$\delta(q_1, B, \varepsilon) = \{(q_0, \varepsilon)\}$, $\delta(q_1, A, \varepsilon) = \{(q_0, AA)\}$,
$\delta(q_1, \$, \varepsilon) = \{(q_0, A\$)\}$.

Automatul funcționează simplu, dacă simbolurile "a" preced simbolurile "b". Pentru fiecare "a" de pe bandă în stivă se va înregistra "AA". În continuare, pentru fiecare "b" de pe bandă se va șterge un "A" din stivă. De exemplu, $(q_0, \$, aabbbb) \vdash (q_0, AA\$, abbbb) \vdash (q_0, AAAA\$, bbbb \overset{*}{\vdash} (q_0, \$, \varepsilon) \vdash (q_0, \varepsilon, \varepsilon)$. Situația se complică puțin, dacă simboluri "b" apar înaintea simbolurilor "a". În acest caz automatul trebuie să țină cont de acești "b", înregistrându-i în stivă ca "B". În continuare simbolului "a" de pe bandă ar putea să-i corespundă combinația "BB" din topul stivei. În acest caz autoomatul va avansa, citind "a" și ștergând "BB". Dacă în topul stivei găsim "BA" sau "$B\$$", atunci simbolul "a" de pe bandă se va citi, iar în stiva se va înregistra "AA" sau "$A\$$" pentru a menține balanța menționată mai sus. De exemplu, $(q_0, \$, bba) \vdash (q_0, B\$, ba) \vdash (q_0, BB\$, a) \vdash (q_1, B\$, \varepsilon) \vdash (q_0, \$, \varepsilon) \vdash (q_0, \varepsilon, \varepsilon)$.

(c) $L_{b3a} = \{x | x \in \{a, b\}^*, n_b(x) = 3n_a(x)\}$. Vom folosi procedeul aplicat la soluționarea problemei 1(b). În acest caz pentru fiecare configurație corectă trebuie să se îndeplinească condiția: $n_A(\text{stivă}) + 3n_a(\text{bandă}) = n_B(\text{stivă}) + n_b(\text{bandă})$. Inserăm mai jos automatul construit.
$AS_{b3a} = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1, q_2\}, \Sigma = \{a, b\}, \Gamma = \{\$, A, B\}$,

$\delta(q_0, \$, a) = \{(q_0, AAA\$)\}$, $\delta(q_0, \$, a) = \{(q_0, AA\$)\}$,
$\delta(q_0, \$, b) = \{(q_0, B\$)\}$, $\delta(q_0, \$, \varepsilon) = \{(q_0, \varepsilon)\}$,
$\delta(q_0, A, a) = \{(q_0, AAAA)\}$, $\delta(q_0, A, b) = \{(q_0, \varepsilon)\}$,
$\delta(q_0, B, b) = \{(q_0, BB)\}$, $\delta(q_0, B, a) = \{(q_2, \varepsilon)\}$,
$\delta(q_2, B, \varepsilon) = \{(q_1,)\}$, $\delta(q_2, \$, \varepsilon) = \{(q_0, AA\$)\}$,
$\delta(q_2, A, \varepsilon) = \{(q_0, AAA)\}$, $\delta(q_1, B, \varepsilon) = \{(q_0, \varepsilon)\}$,
$\delta(q_0, B, a) = \{(q_1, \varepsilon)\}$, $\delta(q_1, A, \varepsilon) = \{(q_0, AA)\}$,
$\delta(q_1, \$, \varepsilon) = \{(q_0, A\$)\}$,

(d) $L_{2ab3a}= \{x|x \in \{a,b\}^*, n_a(x) \geq 0,\ 2n_a(x) \leq n_b(x) \leq\ 3n_a(x)\}$.

Soluția se obține prin compoziția nedeterministă a automatelor construite pentru problemele 1(b) și 1(c). Aceasta este posibil deoarece orice număr întreg $n_b(x)$, $2n_a(x) \leq n_b(x) \leq 3n_a(x)$, $n_a(x) \geq 0$, poate fi reprezentat ca $n_b(x) = 2i + 3j$, $i, j \geq 0$. Să menționăm că pentru $3n_a(x) \leq n_b(x) \leq 5n_a(x)$, de exemplu, această proprietate nu se îndeplinește. Astfel, dacă $n_a(x) = 2$ și $n_b(x) = 7$, $6 \leq 7 \leq 10$, dar 7 nu poate fi reprezentat ca $7 = 3i + 5j$.

Inserăm mai jos automatul.
$AS_{2ab3a} = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1, q_2\}, \Sigma = \{a, b\}, \Gamma = \{\$, A, B\}$,

$\delta(q_0, \$, \varepsilon) = \{(q_0, \varepsilon)\}$,	$\delta(q_0, \$, a) = \{(q_0, AAA\$)\}$,
$\delta(q_0, \$, a) = \{(q_0, AA\$)\}$,	$\delta(q_0, \$, b) = \{(q_0, B\$)\}$,
$\delta(q_0, \$, \varepsilon) = \{(q_0, \varepsilon)\}$,	$\delta(q_0, A, a) = \{(q_0, AAAA)\}$,
$\delta(q_0, A, b) = \{(q_0, \varepsilon)\}$,	$\delta(q_0, B, b) = \{(q_0, BB)\}$,
$\delta(q_0, B, a) = \{(q_2, \varepsilon)\}$,	$\delta(q_2, B, \varepsilon) = \{(q_1,)\}$,
$\delta(q_2, \$, \varepsilon) = \{(q_0, AA\$)\}$,	$\delta(q_2, A, \varepsilon) = \{(q_0, AAA)\}$,
$\delta(q_1, B, \varepsilon) = \{(q_0, \varepsilon)\}$,	$\delta(q_0, B, a) = \{(q_1, \varepsilon)\}$,
$\delta(q_1, A, \varepsilon) = \{(q_0, AA)\}$,	$\delta(q_1, \$, \varepsilon) = \{(q_0, A\$)\}$.

Inserăm mai jos rezultatele modelării automatului aplicând algoritmul din paragraful 9.
Pentru șirul $z = babbbbbbbbaaba$ $(n_a = 4, n_b = 10)$ se obțin 6 configurații de acceptare și 14 configurații de impas. Există 2 moduri de reprezentare a numărului 10 sub forma $10 = 2 \cdot i + 3 \cdot j$: 10=2· 5+0· 3, 10=2· 2+2· 3. Se obțin 6 configurații de acces deoarece pentru automat este importantă și ordinea factorilor: 1) 10=2+2+2+2+2, 2) 10=2+2+3+3, 3) 10=2+3+2+3, 4)10=2+3+3+2, 5) 10=3+ 2+3+2, 6) 10=3+3+2+2.

Modelarea automatului AS_{2ab3a}

Automatul linearizat ∗AS∗=

```
0. (Q0 $ "" Q0)
1. (Q0 $ "a" Q0 A A A $)
2. (Q0 $ "a" Q0 A A $)
3. (Q0 $ "b" Q0 B $)
4. (Q0 A "a" Q0 A A A A)
5. (Q0 A "b" Q0)
6. (Q0 B "b" Q0 B B)
7. (Q0 B "a" Q2)
8. (Q2 B "" Q1)
9. (Q2 $ "" Q0 A A $)
10. (Q2 A "" Q0 A A A)
11. (Q1 B "" Q0)
12. (Q0 B "a" Q1)
13. (Q1 A "" Q0 A A)
14. (Q1 $ "" Q0 A $)

   Sir acceptat z="babbbbbbbbaaba"
   Configuratii de acceptare pentru z="babbbbbbbbaaba"
   (Q0 NIL "" (3 12 14 5 3 6 6 6 6 6 6 12 11 7 8 11 6 7 8 11 0))
   (Q0 NIL "" (3 12 14 5 3 6 6 6 6 6 6 7 8 11 7 8 11 6 12 11 0))
   (Q0 NIL "" (3 12 14 5 3 6 6 6 6 6 6 7 8 11 12 11 6 7 8 11 0))
   (Q0 NIL "" (3 7 9 5 5 3 6 6 6 6 6 7 8 11 12 11 6 12 11 0))
   (Q0 NIL "" (3 7 9 5 5 3 6 6 6 6 6 12 11 7 8 11 6 12 11 0))
   (Q0 NIL "" (3 7 9 5 5 3 6 6 6 6 6 12 11 12 11 6 7 8 11 0))
   Configuratii de impas pentru z="babbbbbbbbaaba"
   (Q0 (A $)""(3 12 14 5 3 6 6 6 6 6 6 7 8 11 7 8 11 6 7 8 14))
   (Q0 (A $) "" (3 7 9 5 5 3 6 6 6 6 6 7 8 11 12 11 6 7 8 14))
   (Q0 (A $) "" (3 7 9 5 5 3 6 6 6 6 6 7 8 11 7 8 11 3 12 14))
   (Q0 (A A $) "" (3 7 9 5 5 3 6 6 6 6 6 7 8 11 7 8 11 3 7 9))
   (Q0 (A $) "" (3 7 9 5 5 3 6 6 6 6 6 12 11 7 8 11 6 7 8 14))
   (Q0 (B $) "" (3 12 14 5 3 6 6 6 6 6 6 12 11 12 11 6 7 8 11))
   (Q0 (B $) "" (3 12 14 5 3 6 6 6 6 6 6 12 11 7 8 11 6 12 11))
   (Q0 (B $) "" (3 12 14 5 3 6 6 6 6 6 6 7 8 11 12 11 6 12 11))
   (Q0 NIL "ba" (3 7 9 5 5 3 6 6 6 6 6 7 8 11 7 8 11 0))
   (Q0 (B $) "" (3 7 9 5 5 3 6 6 6 6 6 12 11 12 11 6 12 11))
   (Q0 (B B $)""(3 12 14 5 3 6 6 6 6 6 6 12 11 12 11 6 12 11))
   (Q0 NIL "bbbbbbaaba" (3 7 9 5 5 0))
   (Q0 NIL "bbbbbbbaaba" (3 12 14 5 0))
```

```
(Q0 NIL "babbbbbbbbaaba" (0))
```

2. Să se construiască AS care acceptă prin stivă vidă limbajul: $L_{nm}=\{x|x \in \{a,b\}^*, x = a^n b^m, n \geq 0,\ 2n \leq m \leq 3n$. Automatul va funcțina în mod nedeterminist bazându-se pe procedeul folosit la exercițiul 1(d). Inserăm mai jos automatul.
   $AS_{nm} = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
   $Q = \{q_0, q_1, q_2, q_3\}, \Sigma = \{a, b\}, \Gamma = \{\$, A\}$,
   $\delta(q_0, \$, \varepsilon) = \{(q_3, \varepsilon)\}$, $\delta(q_0, \$, a) = \{(q_1, AAA\$)\}$,
   $\delta(q_0, \$, a) = \{(q_1, AA\$)\}$, $\delta(q_1, A, a) = \{(q_1, AAAA)\}$,
   $\delta(q_1, A, a) = \{(q_1, AAA)\}$, $\delta(q_1, A, \varepsilon) = \{(q_2, A)\}$,
   $\delta(q_2, A, b) = \{(q_2, \varepsilon), (q_3, \varepsilon)\}$, $\delta(q_3, \$, \varepsilon) = \{(q_3, \varepsilon)\}$.

   Să aducem în continuare rezultatele modelării automatului pentru șirul $z =$ ”$aaabbbbbbb$”. Se obțin 3 configurații de acceptare și 67 configurații de impas. Pentru configurațiile de acceptare automatul inițial va înregistra în stivă 7 simboluri "A". Aceasta se poate face în 3 moduri diferite: 1)2+3+2, 2)2+2+3, 3)3+2+2.

Modelarea automatului AS_{nm}

```
Automatul linearizat *AS*=
  0. (Q0 $ "" Q3 $)
  1. (Q0 $ "a" Q1 A A A $)
  2. (Q0 $ "a" Q1 A A $)
  3. (Q1 A "a" Q1 A A A A)
  4. (Q1 A "a" Q1 A A A)
  5. (Q1 A "" Q2 A)
  6. (Q2 A "b" Q2)
  7. (Q2 A "b" Q3)
  8. (Q3 $ "" Q3)

Sir acceptat z="aaabbbbbbb"
Configuratii de acceptare pentru z="aaabbbbbbb"
(Q3 NIL "" (2 3 4 5 6 6 6 6 6 6 7 8))
(Q3 NIL "" (2 4 3 5 6 6 6 6 6 6 7 8))
(Q3 NIL "" (1 4 4 5 6 6 6 6 6 6 7 8))
```

```
Configuratii de impas pentru z="aaabbbbbbb"
(Q2 (A $) "" (1 4 3 5 6 6 6 6 6 6))
(Q3 (A $) "" (1 4 3 5 6 6 6 6 6 7))
(Q2 ($) "" (1 4 4 5 6 6 6 6 6 6))
(Q2 (A A $) "" (1 3 3 5 6 6 6 6 6 6))
(Q3 (A A $) "" (1 3 3 5 6 6 6 6 6 7))
(Q2 (A $) "" (1 3 4 5 6 6 6 6 6 6))
(Q3 (A $) "" (1 3 4 5 6 6 6 6 6 7))
(Q2 ($) "" (2 4 3 5 6 6 6 6 6 6))
(Q3 NIL "b" (2 4 4 5 6 6 6 6 7 8))
(Q2 (A $) "" (2 3 3 5 6 6 6 6 6 6))
(Q3 (A $) "" (2 3 3 5 6 6 6 6 6 7))
(Q2 ($) "" (2 3 4 5 6 6 6 6 6 6))
(Q3 (A $) "b" (2 3 4 5 6 6 6 6 7))
(Q3 (A A $) "b" (2 3 3 5 6 6 6 6 7))
(Q2 ($) "b" (2 4 4 5 6 6 6 6 6))
(Q3 (A $) "b" (2 4 3 5 6 6 6 6 7))
(Q3 (A A $) "b" (1 3 4 5 6 6 6 6 7))
(Q3 (A A A $) "b" (1 3 3 5 6 6 6 6 7))
(Q3 (A $) "b" (1 4 4 5 6 6 6 6 7))
(Q3 (A A $) "b" (1 4 3 5 6 6 6 6 7))
(Q3 (A A A $) "bb" (1 4 3 5 6 6 6 7))
(Q3 (A A $) "bb" (1 4 4 5 6 6 6 7))
(Q3 (A A A A $) "bb" (1 3 3 5 6 6 6 7))
(Q3 (A A A $) "bb" (1 3 4 5 6 6 6 7))
(Q3 (A A $) "bb" (2 4 3 5 6 6 6 7))
(Q3 (A $) "bb" (2 4 4 5 6 6 6 7))
(Q3 (A A A $) "bb" (2 3 3 5 6 6 6 7))
(Q3 (A A $) "bb" (2 3 4 5 6 6 6 7))
(Q3 (A A A $) "bbb" (2 3 4 5 6 6 7))
(Q3 (A A A A $) "bbb" (2 3 3 5 6 6 7))
(Q3 (A A $) "bbb" (2 4 4 5 6 6 7))
(Q3 (A A A $) "bbb" (2 4 3 5 6 6 7))
(Q3 (A A A A $) "bbb" (1 3 4 5 6 6 7))
(Q3 (A A A A A $) "bbb" (1 3 3 5 6 6 7))
(Q3 (A A A $) "bbb" (1 4 4 5 6 6 7))
(Q3 (A A A A $) "bbb" (1 4 3 5 6 6 7))
(Q3 (A A A A A $) "bbbb" (1 4 3 5 6 7))
```

```
(Q3 (A A A A $) "bbbb" (1 4 4 5 6 6 7))
(Q3 (A A A A A A $) "bbbb" (1 3 3 5 6 6 7))
(Q3 (A A A A A $) "bbbb" (1 3 4 5 6 6 7))
(Q3 (A A A A $) "bbbb" (2 4 3 5 6 6 7))
(Q3 (A A A $) "bbbb" (2 4 4 5 6 6 7))
(Q3 (A A A A A $) "bbbb" (2 3 3 5 6 6 7))
(Q3 (A A A A $) "bbbb" (2 3 4 5 6 6 7))
(Q3 (A A A A A $) "bbbbb" (2 3 4 5 6 7))
(Q3 (A A A A A A $) "bbbbb" (2 3 3 5 6 7))
(Q3 (A A A A $) "bbbbb" (2 4 4 5 6 7))
(Q3 (A A A A A $) "bbbbb" (2 4 3 5 6 7))
(Q3 (A A A A A A $) "bbbbb" (1 3 4 5 6 7))
(Q3 (A A A A A A A $) "bbbbb" (1 3 3 5 6 7))
(Q3 (A A A A A $) "bbbbb" (1 4 4 5 6 7))
(Q3 (A A A A A A $) "bbbbb" (1 4 3 5 6 7))
(Q3 (A A A A A A A $) "bbbbbb" (1 4 3 5 7))
(Q3 (A A A A A A $) "bbbbbb" (1 4 4 5 7))
(Q3 (A A A A A A A A $) "bbbbbb" (1 3 3 5 7))
(Q3 (A A A A A A A $) "bbbbbb" (1 3 4 5 7))
(Q3 (A A A A A A $) "bbbbbb" (2 4 3 5 7))
(Q3 (A A A A A $) "bbbbbb" (2 4 4 5 7))
(Q3 (A A A A A A A $) "bbbbbb" (2 3 3 5 7))
(Q3 (A A A A A A $) "bbbbbb" (2 3 4 5 7))
(Q2 (A A A A A $) "abbbbbbb" (1 4 5))
(Q2 (A A A A A A $) "abbbbbbb" (1 3 5))
(Q2 (A A A A $) "abbbbbbb" (2 4 5))
(Q2 (A A A A A $) "abbbbbbb" (2 3 5))
(Q2 (A A $) "aabbbbbbb" (2 5))
(Q2 (A A A $) "aabbbbbbb" (1 5))
(Q3 NIL "aaabbbbbbb" (0 8))
```

3. Vom folosi procedeele propuse în capitolul 5.
$AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$,
$Q = \{q_0, q_1, q_2, q_3, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}\}$,
$\Sigma = \{a, b\}, \Gamma = \{\$, A\}$,

$$\delta(q_0, \varepsilon, a) = \{(p_1, \varepsilon)\}, \qquad \delta(p_1, \varepsilon, \varepsilon) = \{(p_2, A)\},$$
$$\delta(p_2, \varepsilon, \varepsilon) = \{(p_3, A)\}, \qquad \delta(p_3, \varepsilon, \varepsilon) = \{(q_1, A)\},$$

$\delta(q_0,\$,\varepsilon)=\{(q_3,\varepsilon)\}$, $\delta(q_1,\varepsilon,a)=\{(p_4,\varepsilon)\}$,
$\delta(p_4,\varepsilon,\varepsilon)=\{(p_5,A)\}$, $\delta(p_5,\varepsilon,\varepsilon)=\{(p_6,A)\}$,
$\delta(p_6,\varepsilon,\varepsilon)=\{(p_7,A)\}$, $\delta(p_7,\varepsilon,\varepsilon)=\{(q_1,A)\}$,
$\delta(q_1,\varepsilon,b)=\{(p_8,\varepsilon)\}$, $\delta(p_8,A,\varepsilon)=\{(q_2,\varepsilon)\}$,
$\delta(q_2,\varepsilon,b)=\{(p_9,\varepsilon)\}$, $\delta(p_9,A,\varepsilon)=\{(q_2,\varepsilon)\}$,
$\delta(q_2,\varepsilon,b)=\{(p_{10},\varepsilon)\}$, $\delta(p_{10},A,\varepsilon)=\{(q_3,\varepsilon)\}$,
$\delta(q_3,A,\varepsilon)=\{(q_3,\varepsilon)\}$, $\delta(q_3,\$,\varepsilon)=\{(q_3,\varepsilon)\}$.

4.(a) Conform definiției AS_V, orice configurație $(q_i,\varepsilon,\varepsilon)$ este o configurație de acceptare, $q_i \in Q$. Vom cere ca primul pas al AS_F construit să fie $(q_{00},€,x)\underset{AS_F}{\vdash}(q_0,\$€,x)$, unde q_{00} va fi starea inițială, iar € - simbolul evidențiat al automatului construit. Astfel, automatul AS_F, modelând AS_V, pentru orice configurație de acceptare a AS_V va ajunje în configurația

$$(q_{00},€,x)\underset{AS_F}{\vdash}(q_0,\$€,x)\overset{*}{\underset{AS_V}{\vdash}}(q_i,€,\varepsilon).$$

Pentru a accepta, AS_F trebuie doar să treacă în starea q_f. Prezentăm mai jos algoritmul.

Algoritmul 13.1 (Convertirea AS_V în AS_F.)

0. ***start***
1. Este dat $AS_V=(Q,\Sigma,\Gamma,\delta,q_0,\$)$.
* Vom construi $AS_F=(Q',\Sigma,\Gamma',\delta',q_{00},€,F')$ *\
2. $Q' := Q\cup\{q_{00},q_f\}$, $q_{00},q_f\notin Q$
3. $\Gamma' := \Gamma\cup\{€\}$, $€\notin\Gamma$
4. $F' := \{q_f\}$
5. Construim δ':
5.1. $\delta' := \{\delta'(q_{00},€,\varepsilon)=\{(q_0,\$€)\}\}$
5.2. $\delta' := \delta'\cup\delta$
5.3. ***Pentru toate*** stările $q_i\in Q$:
$\delta' := \delta'\cup\{\delta'(q_i,€,\varepsilon)=\{(q_f,\varepsilon)\}\}$
6. ***stop***

(b) La construirea automatului AS_F au fost eliminate semnele "$'$", deoarece nu au apărut conflicte cu notațiile AS_V.
$AS_F=(Q,\Sigma,\Gamma,\delta,q_{00},€,F)$,
$Q=\{q_{00},q_0,q_1,q_2,q_f\},\Sigma=\{a,b\},\Gamma=\{\$,A\},\ F=\{q_f\}$,
$\delta(q_{00},€,\varepsilon)=\{(q_0,\$€)\}$, $\delta(q_0,\$,a)=\{(q_0,A)\}$,

$\delta(q_0, A, a) = \{(q_0, AA)\}$, $\delta(q_0, A, b) = \{(q_1, \varepsilon))\}$,
$\delta(q_1, A, b) = \{(q_1, \varepsilon)\}$, $\delta(q_1, A, \varepsilon) = \{(q_1, \varepsilon)\}$,
$\delta(q_0, €, \varepsilon) = \{(q_f, \varepsilon)\}$, $\delta(q_1, €, \varepsilon) = \{(q_f, \varepsilon)\}$.

Să menționăm că pot exista variante AS_F echivalente mai simple decât cel construit cu ajutorul algoritmului. De exemplu:
$AS = (Q, \Sigma, \Gamma, \delta, q_0, \$, F)$,
$Q = \{q_0, q_1\}, \Sigma = \{a, b\}, \Gamma = \{\$, A\},\ F = \{q_1\}$,
$\delta(q_0, \$, a) = \{(q_0, A)\}$, $\delta(q_0, A, a) = \{(q_0, AA)\}$,
$\delta(q_0, A, b) = \{(q_1, \varepsilon)\}$, $\delta(q_1, A, b) = \{(q_1, \varepsilon)\}$.

5. Pentru a aplica algoritmul ASG aducem mai întâi automatul dat la forma normală, conform transformărilor definite în paragraful 5 Prezentăm mai jos automatul obținut.

   $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$),\ Q = \{q_0\}, \Sigma = \{a, b\}, \Gamma = \{\$, A\}$,
   $\delta(q_0, \$, a) = \{(p_1, A\$)\}$, $\delta(p_1, A, \varepsilon) = \{(p_2, AA)\}$,
   $\delta(p_2, A, \varepsilon) = \{(q_0, AA)\}$, $\delta(q_0, A, b) = \{(q_0, \varepsilon)\}$,
   $\delta(q_0, \$, \varepsilon) = \{(q_0, \varepsilon)\}$.

   Rezultatele algoritmului ASG sunt inclise în Figura 24. Gramatica construită conține 32 producții și 19 simboluri neterminale (14 simboluri neproductive). După eliminarea simbolurilor neproductive obținem:
   $G = (V_N, V_T, P, S), V_N = \{S, A, B, C, D\},\ V_T = \{a, b\},\ P = \{$
   $0.\ S \to C \quad 1.\ C \to aAC \quad 2.\ A \to BD \quad 3.\ B \to DD$
   $4.\ D \to b \quad 5.\ C \to \varepsilon\ \}$
   Aplicăm teorema substituțiilor pentru B și D în producțiile 2,3,4. Obținem:
   $G = (V_N, V_T, P, S), V_N = \{S, A, C\},\ V_T = \{a, b\},\ P = \{$
   $0.\ S \to C \quad 1.\ C \to aAC \quad 2.\ A \to bbb \quad 3.\ C \to \varepsilon\}$.
   Observăm că $L(G) = \{x \mid S \overset{*}{\Longrightarrow} x\} = \{x \mid C \overset{*}{\Longrightarrow} x\}$. Substituind C prin S obținem în final:
   $G = (V_N, V_T, P, S), V_N = \{S, A\},\ V_T = \{a, b\},\ P = \{$
   $0.\ S \to aAS \quad 1.\ A \to bbb \quad 2.\ S \to \varepsilon\}$.
   Rezultă că $L(G) = \{(abbb)^i \mid i \geq 0\}$.

<table>
<tr><th>Funcția δ</th><th colspan="2">Producțiile generate</th><th>Notații</th></tr>
<tr><td></td><td>$S \to [q_0\$p_2]$
$S \to [q_0\$p_1]$
$S \to [q_0\$q_0]$</td><td>$0.S \to R$
$1.S \to Q$
$2.S \to C$</td><td rowspan="6">$[p_1Aq_0]$ – A
$[p_2Aq_0]$ – B
$[q_0\$q_0]$ – C
$[q_0Aq_0]$ – D
$[q_0Ap_2]$ – E
$[q_0Ap_1]$ – F
$[p_2Ap_2]$ – G
$[p_2Ap_1]$ – H
$[p_2\$p_2]$ – I
$[p_1\$p_2]$ – J
$[p_2\$p_1]$ – K
$[p_1\$p_1]$ – L
$[p_2\$q_0]$ – M
$[p_1Ap_2]$ – N
$[p_1\$q_0]$ – O
$[p_1Ap_1]$ – P
$[q_0\$p_1]$ – Q
$[q_0\$p_2]$ – R</td></tr>
<tr><td>$\delta(q_0,\$,a)=$
$\{(p_1,A\$)\}$</td><td>$[q_0\$q_0] \to a[p_1Aq_0][q_0\$q_0]$
$[q_0\$q_0] \to a[p_1Ap_1][p_1\$q_0]$
$[q_0\$q_0] \to a[p_1Ap_2][p_2\$q_0]$
$[q_0\$p_1] \to a[p_1Aq_0][q_0\$p_1]$
$[q_0\$p_1] \to a[p_1Ap_1][p_1\$p_1]$
$[q_0\$p_1] \to a[p_1Ap_2][p_2\$p_1]$
$[q_0\$p_2] \to a[p_1Aq_0][q_0\$p_2]$
$[q_0\$p_2] \to a[p_1Ap_1][p_1\$p_2]$
$[q_0\$p_2] \to a[p_1Ap_2][p_2\$p_2]$</td><td>$3.C \to aAC$
$4.C \to aPO$
$5.C \to aNM$
$6.Q \to aAQ$
$7.Q \to aPL$
$8.Q \to aNK$
$9.R \to aAR$
$10.R \to aPJ$
$11.R \to aNI$</td></tr>
<tr><td>$\delta(p_1,A,\varepsilon)=$
$\{(p_2,AA)\}$</td><td>$[p_1Aq_0] \to [p_2Aq_0][q_0Aq_0]$
$[p_1Aq_0] \to [p_2Ap_1][p_1Aq_0]$
$[p_1Aq_0] \to [p_2Ap_2][p_2Aq_0]$
$[p_1Ap_1] \to [p_2Aq_0][q_0Ap_1]$
$[p_1Ap_1] \to [p_2Ap_1][p_1Ap_1]$
$[p_1Ap_1] \to [p_2Ap_2][p_2Ap_1]$
$[p_1Ap_2] \to [p_2Aq_0][q_0Ap_2]$
$[p_1Ap_2] \to [p_2Ap_1][p_1Ap_2]$
$[p_1Ap_2] \to [p_2Ap_2][p_2Ap_2]$</td><td>$12.A \to BD$
$13.A \to HA$
$14.A \to GB$
$15.P \to BF$
$16.P \to HP$
$17.P \to GH$
$18.N \to BE$
$19.N \to HN$
$20.N \to GG$</td></tr>
<tr><td>$\delta(p_2,A,\varepsilon)=$
$\{(q_0,AA)\}$</td><td>$[p_2Aq_0] \to [q_0Aq_0][q_0Aq_0]$
$[p_2Aq_0] \to [q_0Ap_1][p_1Aq_0]$
$[p_2Aq_0] \to [q_0Ap_2][p_2Aq_0]$
$[p_2Ap_1] \to [q_0Aq_0][q_0Ap_1]$
$[p_2Ap_1] \to [q_0Ap_1][p_1Ap_1]$
$[p_2Ap_1] \to [q_0Ap_2][p_2Ap_1]$
$[p_2Ap_2] \to [q_0Aq_0][q_0Ap_2]$
$[p_2Ap_2] \to [q_0Ap_1][p_1Ap_2]$
$[p_2Ap_2] \to [q_0Ap_2][p_2Ap_2]$</td><td>$21.B \to DD$
$22.B \to FA$
$23.B \to EB$
$24.H \to DF$
$25.H \to FP$
$26.H \to EH$
$27.G \to DE$
$28.G \to FN$
$29.G \to EG$</td></tr>
<tr><td>$\delta(q_0,A,b)=$
$\{(q_0,\varepsilon)\}$</td><td>$[q_0Aq_0] \to b$</td><td>$30.D \to b$</td></tr>
<tr><td>$\delta(q_0,\$,\varepsilon)=$
$\{(q_0,\varepsilon)\}$</td><td>$[q_0\$q_0] \to \varepsilon$</td><td>$31.C \to \varepsilon$</td></tr>
</table>

Figura 24: Algoritmul ASG.
Limbajul $L(G) = \{(abbb)^i \mid i \geq 0\}$.

6. Să se construiască AS care acceptă limbajul:
$L_{ij} = \{x|x \in \{a,b\}^*, x = a^ib^j, i \geq 0, j \geq 0, i \neq j\}$.
Construim mai întâi automatele AS pentru limbajele
$L_{i>j} = \{x|x \in \{a,b\}^*, x = a^ib^j, i > j, j \geq 0,\}$ și

$L_{j>i}= \{x|x \in \{a,b\}^*, x = a^ib^j, j > i, i \geq 0,\}$, apoi prin compoziție construim automatul final:

$AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$, $Q = \{q_0, q_1, q_2, q_3, q_a, q_b\}$, $\Sigma = \{a, b\}$, $\Gamma = \{\$, A\}$.

$\delta(q_0, \$, \varepsilon) = \{(q_a, \$), (q_b, \$)\}$, $\delta(q_a, \$, a) = \{(q_a, A\$), (q_a, \$), (q_1, \$)\}$,
$\delta(q_a, A, a) = \{(q_a, AA)\}$, $\delta(q_a, A, b) = \{(q_2, \varepsilon)\}$,
$\delta(q_2, A, b) = \{(q_2, \varepsilon)\}$, $\delta(q_2, A, \varepsilon) = \{(q_1, \varepsilon\}$,
$\delta(q_1, \$, \varepsilon) = \{(q_1, \varepsilon)\}$, $\delta(q_b, \$, a) = \{(q_b, A\$)\}$,
$\delta(q_b, \$, b) = \{(q_3, \$)\}$, $\delta(q_b, A, a) = \{(q_b, AA)\}$,
$\delta(q_b, A, b) = \{(q_3, \varepsilon)\}$, $\delta(q_3, A, b) = \{(q_3, \$)\}$,
$\delta(q_3, \$, b) = \{(q_3, \$), (q_3, \varepsilon)\}$.

Să menționăm că automatul construit acceptă șirurile $x = a^i, i \geq 1$ și $x = b^j, j \geq 1$.

7.(a) Construcția automatului este destul de transparentă: în dependență de prefixul șirului ("b", "bb" sau "bbb") se alege una din 3 alternative caracterizate prin stările q_{1b}, q_{2b}, q_{3b}. Inserăm mai jos automatul construit.

$AS_{123} = (Q, \Sigma, \Gamma, \delta, q_0, \$)$, $Q = \{q_0, q_{1b}, q_{2b}, q_{3b}, q_1\}$, $\Sigma = \{a, b\}$
$\Gamma = \{\$, A\}$,

$\delta(q_0, \$, b) = \{(q_{1b}, \$)\}$ $\delta(q_{1b}, \$, b) = \{(q_{2b}, \$)\}$
$\delta(q_{2b}, \$, b) = \{(q_{3b}, \$)\}$ $\delta(q_{1b}, \$, a) = \{(q_{1b}, A\$)\}$
$\delta(q_{1b}, A, a) = \{(q_{1b}, AA)\}$ $\delta(q_{1b}, A, b) = \{(q_1, \varepsilon)\}$
$\delta(q_{2b}, \$, a) = \{(q_{2b}, AA\$)\}$ $\delta(q_{2b}, A, a) = \{(q_{2b}, AAA)\}$
$\delta(q_{2b}, A, b) = \{(q_1, \varepsilon)\}$ $\delta(q_{3b}, \$, a) = \{(q_{3b}, AAA\$)\}$
$\delta(q_{3b}, A, a) = \{(q_{3b}, AAAA)\}$ $\delta(q_{3b}, A, b) = \{(q_1, \varepsilon)\}$
$\delta(q_1, A, b) = \{(q_1, \varepsilon)\})$ $\delta(q_1, \$, \varepsilon) = \{(q_1, \varepsilon)\}$.

Pentru a obține Forma Normală substituim
$\delta(q_{2b}, \$, a) = \{(q_{2b}, AA\$)\}$ și $\delta(q_{2b}, A, a) = \{(q_{2b}, AAA)\}$ prin:

$\delta(q_{2b}, \$, a) = \{(p_2, A\$)\}$,
$\delta(q_{2b}, A, a) = \{(p_2, AA)\}$,
$\delta(p_2, A, \varepsilon) = \{(q_{2b}, AA)\}$,

iar
$\delta(q_{3b}, \$, a) = \{(q_{3b}, AAA\$)\}$ şi $\delta(q_{3b}, A, a) = \{(q_{3b}, AAAA)\}$ prin:

$$\delta(q_{3b}, \$, a) = \{(p_3, A\$)\},$$
$$\delta(q_{3b}, A, a) = \{(p_3, AA)\},$$
$$\delta(p_3, A, \varepsilon) = \{(p_4, AA)\},$$
$$\delta(p_4, A, \varepsilon) = \{(q_{3b}, AA)\}.$$

(b) Construcţia gramaticii este la fel de transparentă. Prezentăm mai jos gramatica.

$G = (V_N, V_T, P, S), V_N = \{S, A, B, C\},\ V_T = \{a, b\},\ P = \{$
0. $S \to bA$ 1. $S \to bbB$ 2. $S \to bbbC$ 3. $A \to ab$
4. $A \to aAb$ 5. $B \to abb$ 6. $B \to aBbb$ 7. $C \to abbb$
8. $C \to aCbbb$ $\}$

Se poate construi gramatica echivalentă aplicând algoritmul ASG asupra Formei Normale a automatului AS_{123}. Algoritmul ASG generează o gramatică cu 613 producţii şi 130 simboluri neterminale. După eliminarea simbolurilor neproductive (117 simboluri) se obţine o gramatică mai simplă (doar 18 producţii şi 13 neterminale):

$G = (V_N, V_T, P, S), V_N = \{S, A, B, C, D, E, F, G, H, I, J, K, L\},$
$V_T = \{a, b\},\ P = \{$
0. $S \to A$ 1. $L \to \varepsilon$ 2. $K \to b$ 3. $J \to b$
4. $J \to aHK$ 5. $I \to JK$ 6. $H \to IK$ 7. $G \to aHL$
8. $F \to b$ 9. $F \to aEK$ 10. $E \to FK$ 11. $D \to aEL$
12. $C \to b$ 13. $C \to aCK$ 14. $B \to aCL$ 15. $D \to bG$
16. $B \to bD$ 17. $A \to bB\}$

După eliminarea ε-producţiei 1. $L \to \varepsilon$ şi efectuarea unei serii de substituţii şi eliminări ale simbolurilor inaccesibile obţinem gramatica: $G = (V_N, V_T, P, S), V_N = \{S, C, F, J\},$
$V_T = \{a, b\},\ P = \{$
0. $S \to bbbaJbb$ 1. $S \to bbaFb$ 2. $S \to baC$ 3. $J \to aJbbb$
4. $J \to b$ 5. $F \to aFbb$ 6. $F \to b$ 7. $C \to aCb$

$8.\, C \to b \quad \}$

Se observă ușor că $L(G) = L_{123}$.

8.(a) Alicând algoritmul GAS construim AS_E:

$$
\begin{aligned}
&AS_E = (Q, \Sigma, \Gamma, \delta, q_0, S),\ Q = \{q_0\},\\
&\Sigma = \{\text{"}a\text{"}, \text{"}+\text{"}, \text{"}*\text{"}, \text{"}(\text{"}, \text{"})\text{"}, \text{"};\text{"}\},\\
&\Gamma = \{S, E, T, F, a, \text{"}+\text{"}, \text{"}*\text{"}, \text{"}(\text{"}, \text{"})\text{"}, \text{"};\text{"}\},
\end{aligned}
$$

$$
\begin{array}{ll}
\delta(q_0, \text{"}a\text{"}, \text{"}a\text{"}) = \{(q_0, \varepsilon)\} & \delta(q_0, \text{"}+\text{"}, \text{"}+\text{"}) = \{(q_0, \varepsilon)\}\\
\delta(q_0, \text{"}*\text{"}, \text{"}*\text{"}) = \{(q_0, \varepsilon)\} & \delta(q_0, \text{"}(\text{"}, \text{"}(\text{"}) = \{(q_0, \varepsilon)\}\\
\delta(q_0, \text{"}(\text{"}, \text{"})\text{"}) = \{(q_0, \varepsilon)\} & \delta(q_0, \text{"};\text{"}, \text{"};\text{"}) = \{(q_0, \varepsilon)\}
\end{array}
$$

$$
\begin{aligned}
&\delta(q_0, S, \varepsilon) = \{(q_0, E\,\text{"};\text{"})\}\\
&\delta(q_0, E, \varepsilon) = \{(q_0, E\,\text{"}+\text{"}\,T), (q_0, T)\}\\
&\delta(q_0, T, \varepsilon) = \{(q_0, T\,\text{"}*\text{"}\,F), (q_0, F)\}\\
&\delta(q_0, F, \varepsilon) = \{(q_0, a), (q_0, \text{"}(\text{"}\,E\,\text{"})\text{"})\}.
\end{aligned}
$$

Am folosit ghilimelele "" pentru a nu confunda simbolurile din Σ cu simbolurile utilizate la descrierea automatului, de exemplu, parantezele.

Datorită faptului că gramatica nu conține ε-producții, adică gramatica este nedescrescătoare, există un număr finit de derivări (configurații) distincte de lungime mai mică sau egală cu lungimea expresiei analizate. Astfel, în calitate de criteriu suplimentar pentru oprire vom folosi și lungimea configurațiilor. Pentru comoditate vom folosi varianta linearizată a automatului (paragraful 9). Astfel, AS_E se reduce la:

$$
\begin{aligned}
&AS_E = (Q, \Sigma, \Gamma, \delta, q_0, S),\ Q = \{q_0\},\\
&\Sigma = \{\text{"}a\text{"}, \text{"}+\text{"}, \text{"}*\text{"}, \text{"}(\text{"}, \text{"})\text{"}, \text{"};\text{"}\},\\
&\Gamma = \{S, E, T, F, \text{"}a\text{"}, \text{"}+\text{"}, \text{"}*\text{"}, \text{"}(\text{"}, \text{"})\text{"}, \text{"};\text{"}\},
\end{aligned}
$$

$$
\begin{array}{ll}
0.\ \delta(q_0, \text{"}a\text{"}, \text{"}a\text{"}) = (q_0, \varepsilon) & 1.\ \delta(q_0, \text{"}+\text{"}, \text{"}+\text{"}) = (q_0, \varepsilon)\\
2.\ \delta(q_0, \text{"}*\text{"}, \text{"}*\text{"}) = (q_0, \varepsilon) & 3.\ \delta(q_0, \text{"}(\text{"}, \text{"}(\text{"}) = (q_0, \varepsilon)\\
4.\ \delta(q_0, \text{"})\text{"}, \text{"})\text{"}) = (q_0, \varepsilon) & 5.\ \delta(q_0, \text{"};\text{"}, \text{"};\text{"}) = (q_0, \varepsilon)\\
6.\ \delta(q_0, S, \varepsilon) = (q_0, E\,\text{"};\text{"}) & 7.\ \delta(q_0, E, \varepsilon) = (q_0, E\,\text{"}+\text{"}\,T)\\
8.\ \delta(q_0, E, \varepsilon) = (q_0, T) & 9.\ \delta(q_0, T, \varepsilon) = (q_0, T\,\text{"}*\text{"}\,F)
\end{array}
$$

10. $\delta(q_0, T, \varepsilon) = (q_0, F)$ 11. $\delta(q_0, F, \varepsilon) = (q_0, "a")$
12. $\delta(q_0, F, \varepsilon) = (q_0, "(" \; E \; ")")$.

Pentru expresia "$a + a * a;$" avem $|a + a * a;| = 6$. La modelare vom obține 23 secvențe de configurații posibile, 22 dintre care vor corespunde situațiilor de impas. Putem reprezenta aceste secvențe sub forma unui arbore (Figura 25), unde nodurile terminale corespund situațiilor de *impas* (notate prin i) și de *acceptare* (notate prin a). Celelalte noduri marchează o tranziție a automatului. Astfel, fiecare drum, pornind de la rădăcină și terminând cu un nod terminal, reprezintă o secvență de configurații posibile de lungime mai mică sau egală cu 6.

Vom nota mai jos prin $\stackrel{i}{\vdash}$ aplicarea tranziției cu numărul i. De exemplu, drumul 6,7,8,10,11,0,1,10,12 corespunde secvenței de configurații:

$$
\begin{array}{ll}
(q_0, S, "a + a * a;") & \stackrel{6}{\vdash} (q_0, E \, ";", "a + a * a;") \stackrel{7}{\vdash} \\
(q_0, E \, T \, ";", "a + a * a;") & \stackrel{8}{\vdash} (q_0, T \, "+" \, T \, ";", "a + a * a;") \stackrel{10}{\vdash} \\
(q_0, F \, "+" \, T \, ";", "a+a * a;") & \stackrel{11}{\vdash} (q_0, "a +" \, T \, ";", "a+a * a;") \stackrel{0}{\vdash} \\
(q_0, "+" \, T \, ";", "+a * a;") & \stackrel{1}{\vdash} (q_0, T \, ";", "a * a;") \stackrel{10}{\vdash} \\
(q_0, F \, ";", "a * a;") & \stackrel{12}{\vdash} (q_0, "(" \; E \; ");", "a * a;") \text{ - } \textit{impas},
\end{array}
$$

iar drumul 6,7,8,10,11,0,1,9,10,11,0,2,11,0,5 - secvenței:

$$
\begin{array}{ll}
(q_0, S, "a + a * a;") & \stackrel{6}{\vdash} (q_0, E \, ";", "a + a * a;") \stackrel{7}{\vdash} \\
(q_0, E \, "+" T \, ";", "a + a * a;") & \stackrel{8}{\vdash} (q_0, T \, "+" T \, ";", "a + a * a;") \stackrel{10}{\vdash} \\
(q_0, F \, "+" \, T \, ";", "a + a * a;") & \stackrel{11}{\vdash} (q_0, "a +" \, T \, ";", "a + a * a;") \stackrel{0}{\vdash} \\
(q_0, \; "+" \, T \, ";", " + a * a;") & \stackrel{1}{\vdash} (q_0, T; , "a * a;") \stackrel{9}{\vdash} \\
(q_0, T \, "*" \, F; , "a * a;") & \stackrel{10}{\vdash} (q_0, F \, "*" \, F; , "a * a;") \stackrel{11}{\vdash} \\
(q_0, "a *" \, F; , "a * a;") & \stackrel{0}{\vdash} (q_0, "*" \, F; , " * a;") \stackrel{2}{\vdash} \\
(q_0, F \, ";", "a;") & \stackrel{11}{\vdash} (q_0, "a;", "a;") \stackrel{0}{\vdash} \\
(q_0, ";", ";") & \stackrel{5}{\vdash} (q_0, \varepsilon, \varepsilon) \text{ - } \textit{acceptare}.
\end{array}
$$

Dacă vom considera în continuare o expresie greșită, de exemplu, $a + a)$, vom obține un arbore (Figura 26) cu 11 secvențe, toate corespunzând situațiilor de impas.

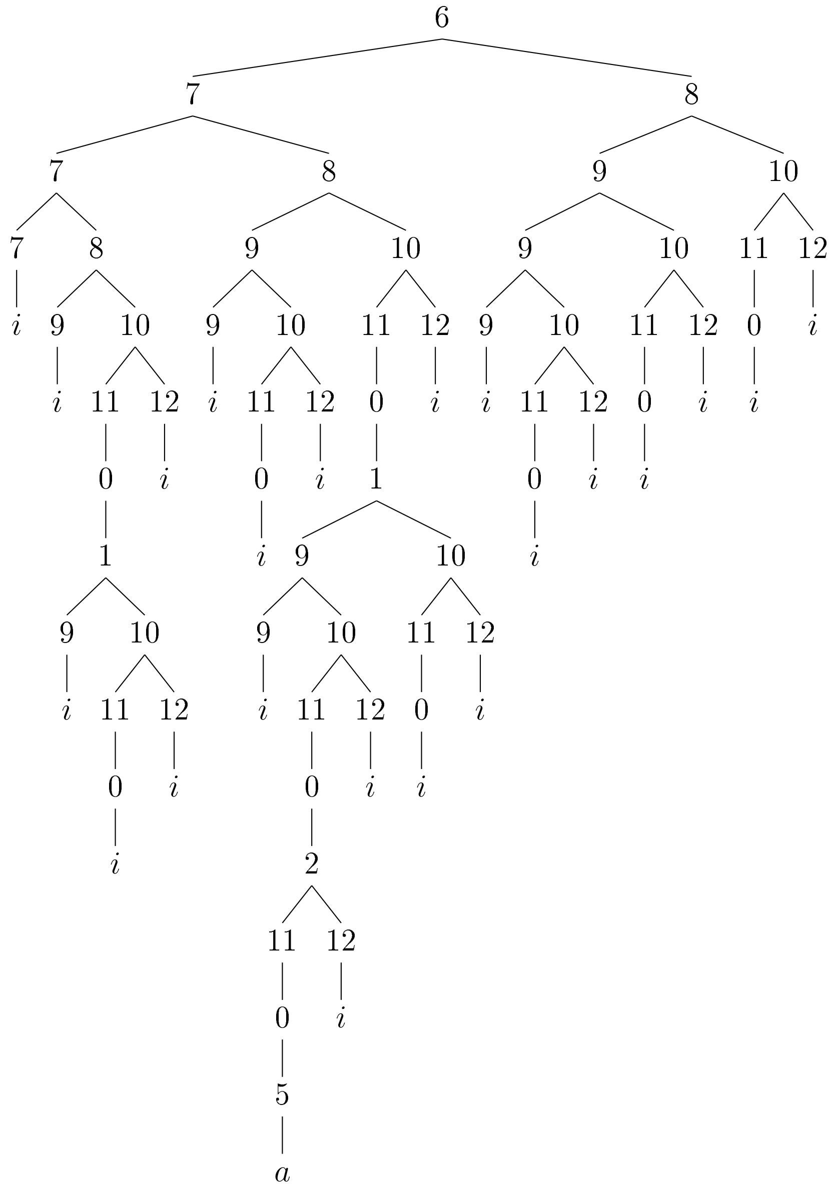


Figura 25: Arborele configurațiilor AS_E pentru expresia $a + a * a;$.

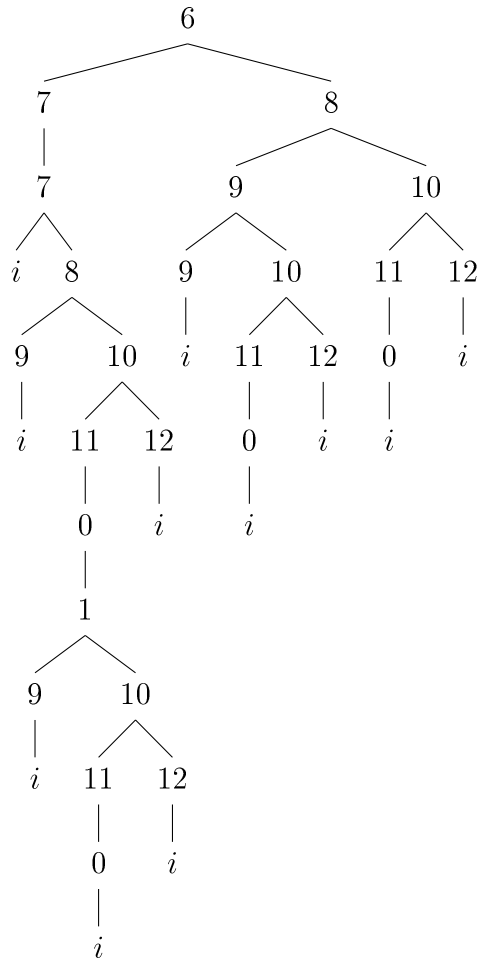


Figura 26: Arborele configurațiilor AS_E pentru expresia $a + a$).

Pentru a ne convinge că acest proces de modelare este foarte anevoios, să mai examinăm o expresie, relativ simplă, $a*(a+a);$. Pentru această expresie la modelare se va obține un arbore cu 47 secvențe (46 - *impas*), Figura 27.

(b) Pentru a construi varianta deterministă, care nu întotdeauna există, vom transforma puțin gramatica expresiilor eliminând recursia stângă cu implicarea ε-producțiilor. Obținem:

$G=(V_N,V_T,P,S), V_N=\{S,E,T,F,X,Y\}$,
$V_T=\{a,+,*,(,),;\}$, $P=\{$

$0.\ S\to E;$	$1.\ E\to TX$
$2.\ X\to\varepsilon$	$3.\ X\to +TX$
$4.\ T\to FY$	$5.\ Y\to\varepsilon$
$6.\ Y\to *FY$	$7.\ F\to a$
$8.\ F\to (E)\quad\}$	

Aplicând procedee asemănătoare cu cele explicate în Exemplul 8.1, construim:

$AS_E=(Q,\Sigma,\Gamma,\delta,q_0,S)$, $Q=\{q_0\}$, $\Sigma=\{"a","+","*","(",")",";"\}$,
$\Gamma=\{S,E,T,F,X,Y,"a","+","*","(",")",";"\}$,

0. $\delta(q_0,S,\varepsilon)=(q_0,E;)$	1. $\delta(q_0,E,"a")=(q_0,YX)$
2. $\delta(q_0,E,"(")=(q_0,E\,")"\,YX)$	3. $\delta(q_0,T,"a")=(q_0,Y)$
4. $\delta(q_0,T,"(")=(q_0,E\,"("\,Y)$	5. $\delta(q_0,F,"(")=(q_0,E\,"(")$
6. $\delta(q_0,F,"a")=(q_0,\varepsilon)$	7. $\delta(q_0,Y,"*")=(q_0,FY)$
8. $\delta(q_0,Y,";")=(q_1,\varepsilon)$	9. $\delta(q_0,Y,"+")=(q_2,\varepsilon)$
10. $\delta(q_0,Y,")")=(q_3,\varepsilon)$	11. $\delta(q_1,";",\varepsilon)=(q_1,\varepsilon)$
12. $\delta(q_1,X,\varepsilon)=(q_1,\varepsilon)$	13. $\delta(q_2,X,\varepsilon)=(q_0,TX)$
14. $\delta(q_3,X,\varepsilon)=(q_4,\varepsilon)$	15. $\delta(q_4,")",\varepsilon)=(q_0,\varepsilon)$.

Astfel,

- pentru expresia "$a+a*a;$" există o singură secvență de acceptare:
  0,1,9,13,3,7,6,8,12,11
- pentru "$a+a)$" - o singură secvență de impas:
  0,1,9,13

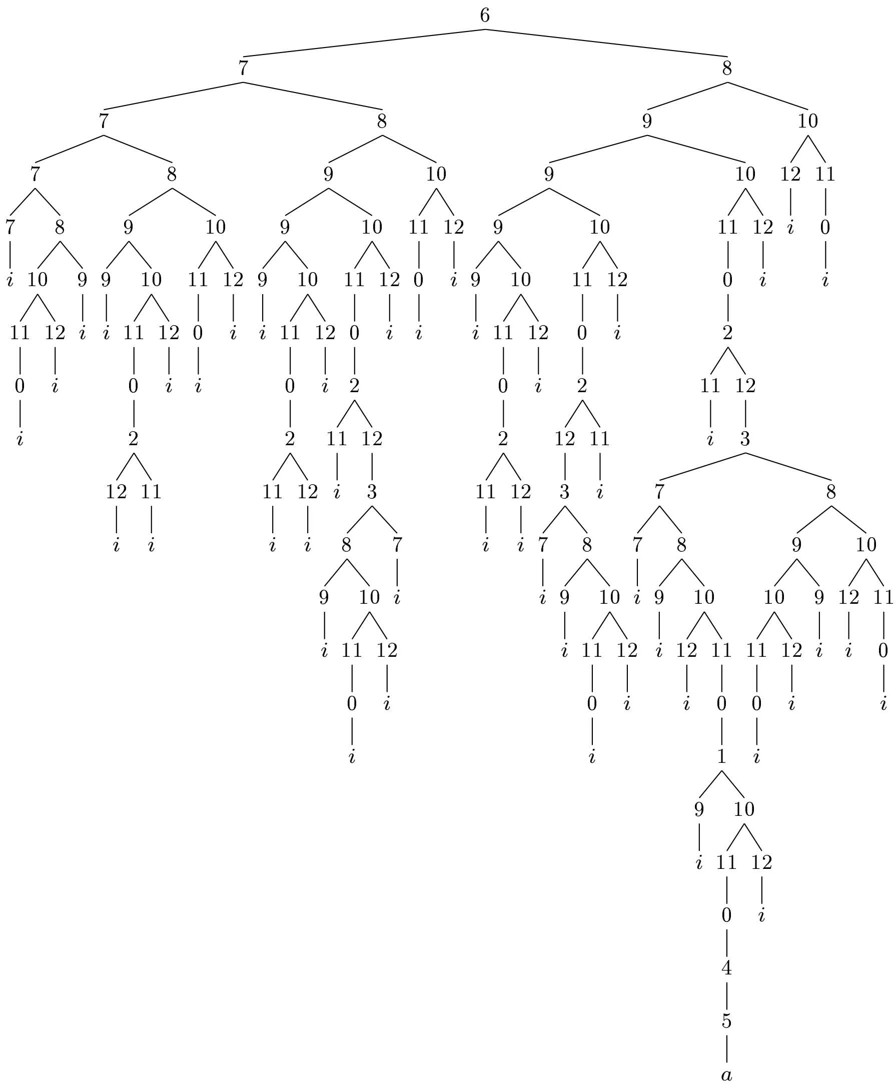


Figura 27: Arborele configurațiilor AS_E pentru expresia $a * (a + a)$;

- pentru "$a * (a + a);$" - o singură secvență de acceptare: 0,1,7,5,1,9,13,3,10,14,15,8,12,11

9. Să se construiască AS cu acceptare prin stivă vidă pentru limbajul $L = \{x | x \in \{a, b\}^*, x = x_1 x_2, n_b(x_1) > n_a(x_1)\}$. Vom nota prin $n_a(x)$ numărul de simboluri "a", iar prin $n_b(x)$ - numărul de simboluri "b" din x.

   Observăm că limbajul L conține toate șirurile diferite de ε peste $\{a, b\}$ care au cel puțin un prefix cu mai mulți "b" decât "a". De exemplu, șirurile "b", "abb", "$abba$","$abbbabba$" aparțin limbajului, iar șirurile "ab", "aaa", "$abababab$" nu aparțin. Șirul "$abbbabba$" conține 5 prefixe cu proprietatea menționată: "abb", "$abbb$", "$abbbab$", "$abbbabb$", "$abbbabba$". Ultimul prefix este chiar șirul dat.

   Prezentăm mai jos AS astfel construit.

   $AS = (Q, \Sigma, \Gamma, \delta, q_0, \$)$, $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{\$, A\}$,
   $\delta(q_0, \$, b) = \{(q_1, \$)\}$, $\delta(q_0, \$, a) = \{(q_2, A\$)\}$,
   $\delta(q_1, \$, a) = \{(q_1, \$), (q_1, \varepsilon)\}$, $\delta(q_1, \$, b) = \{(q_1, \$), (q_1, \varepsilon)\}$,
   $\delta(q_1, \$, \varepsilon) = \{(q_1, \varepsilon)\}$, $\delta(q_2, A, a) = \{(q_2, AA)\}$,
   $\delta(q_2, A, b) = \{(q_2, \varepsilon)\}$, $\delta(q_2, \$, b) = \{(q_1, \$)\}$,
   $\delta(q_2, \$, a) = \{(q_2, A\$)\}$.

   Dacă șirul x începe cu "b", atunci x aparține limbajului, automatul trece în starea q_1 și citeste toate simbolurile "a" și "b" rămase pe bandă fără nici o verificare. În caz contrar, automatul va înregistra în stivă "A" pentru fiecare "a" citit de pe bandă și va șterge un "A" pentru fiecare "b" de pe bandă. Astfel, dacă numărul de "b" depășește numărul de "a", automatul va ajunje în configurația $(q_2, \$, bx)$ și revine la regimul de citire a simbolurilor "a" și "b" rămase pe bandă.

10. Conform definiției $\overline{L_{xx}}$=$\{0, 1\}^* \setminus L_{xx}$. Fie v un șir arbitrar peste $\{0, 1\}^*$. Dacă lungimea lui v, $|v|$, este număr impar, atunci v aparține limbajului $\overline{L_{xx}}$. Automatul garantează aceste acceptări

prin utilizarea repetată a starilor q_1 și q_2. De exemplu, $(q_0, \$, 01110) \vdash (q_1, \$, 01110) \vdash (q_2, \$, 1110) \vdash (q_1, \$, 110) \vdash (q_2, \$, 10) \vdash (q_1, \$, 0) \vdash (q_1, \varepsilon, \varepsilon)$.

Dacă $|v|$ este număr par, $|v|$=$2n$, $n \geq 1$, atunci v poate fi reprezentat ca v=v_1v_2, $|v_1| = |v_2| = n$. Pentru ca v sa aparțină limbajului $\overline{L_{xx}}$ este necesar ca $v_1 \neq v_2$. Asta înseamnă că există cel puțin o poziție $i, 1 \leq i \leq n$, pentru care $v_1 = v_{11}\boxed{0}v_{12}$, iar $v_2 = v_{21}\boxed{1}v_{22}$ (sau invers, $v_1 = v_{11}\boxed{1}v_{12}$, iar $v_2 = v_{21}\boxed{0}v_{22}$), unde $|v_{11}| = |v_{21}| = i - 1$, $|v_{12}| = |v_{22}| = n - i$. Scopul AS_{xx} constă în a găsi, funcționând în mod nedeterminist, așa o poziție. Dacă așa poziție există, șirul v aparține limbajului $\overline{L_{xx}}$, în caz contrar - nu aparține.

Să notăm prin b un simbol arbitrar din $\{0, 1\}$. AS_{xx} trebuie să verifice dacă șirul v are, de exemplu, forma:

$$v = v_1v_2 = v_{11}\boxed{0}v_{12}v_{21}\boxed{1}v_{22}$$

Schematic

$$v = \underbrace{b \ldots b}_{i-1}\ \boxed{0}\ \underbrace{b \ldots b}_{n-i}\ \underbrace{b \ldots b}_{i-1}\ \boxed{1}\ \underbrace{b \ldots b}_{n-i}$$

Schema funcționării automatului:

1. Alege în mod nedeterminist poziția $i, 1 \leq i \leq n$.
2. Citește $i - 1$ simboluri de pe bandă și înregistrează $i - 1$ simboluri "A" în stivă. Starea curentă a benzii devine:
   $$\boxed{0}\ \underbrace{b \ldots b}_{n-i}\ \underbrace{b \ldots b}_{i-1}\ \boxed{1}\ \underbrace{b \ldots b}_{n-i}$$
3. Citește 0 de pe bandă fără a modifica stiva. Starea curentă a benzii devine:
   $$\underbrace{b \ldots b}_{n-i}\ \underbrace{b \ldots b}_{i-1}\ \boxed{1}\ \underbrace{b \ldots b}_{n-i}$$
4. Citește $i - 1$ simboluri de pe bandă și șterge $i - 1$ simboluri "A" din stivă. De menționat că i nu poate fi mai mare ca n. Starea curentă a benzii devine:
   $$\underbrace{b \ldots b}_{n-i}\ \boxed{1}\ \underbrace{b \ldots b}_{n-i}$$

5. Verifică, utilizând procedeul folosit la recunoașterea palindroamelor, dacă mijlocul șirului rămas pe bandă este 1.

Inserăm mai jos AS_{xx} și reprezentarea lui grafică (Figura 28).

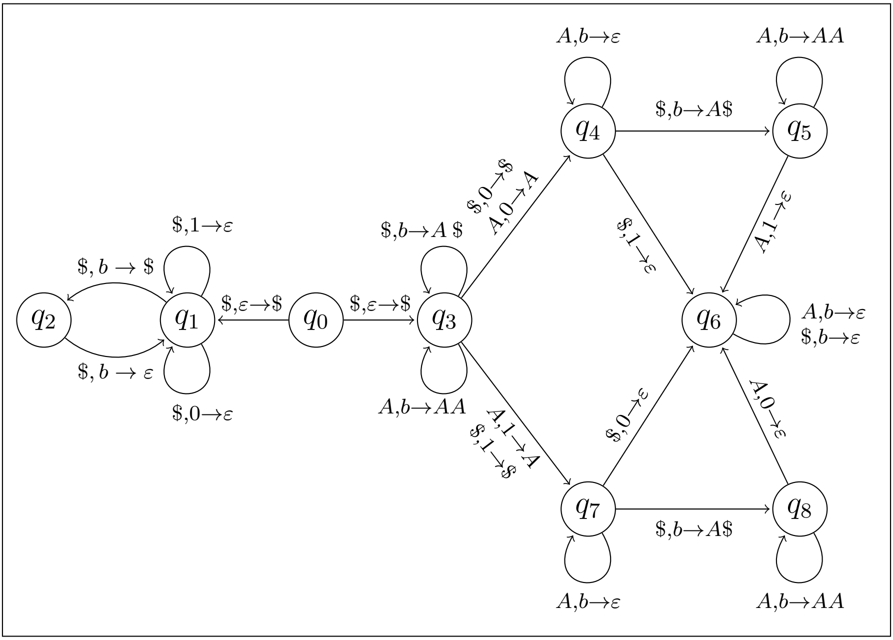


Figura 28: Reprezentarea grafică a AS pentru $\overline{L_{xx}}$

$AS_{xx} = (Q, \Sigma, \Gamma, \delta, q_0, \$),\ Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\},$
$\Sigma = \{0, 1\},\ \Gamma = \{\$, A\},$

$\delta(q_0, \$, \varepsilon) = \{(q_1, \$), (q_3, \$)\},$ $\quad\delta(q_1, \$, 0) = \{(q_1, \varepsilon)(q_2, \$)\},$
$\delta(q_1, \$, 1) = \{(q_1, \varepsilon)(q_2, \$)\},)$ $\quad\delta(q_2, \$, 0) = \{(q_1, \$)\},$
$\delta(q_2, \$, 1) = \{(q_1, \$)\},$ $\quad\delta(q_3, \$, 0) = \{(q_3, A\$)(q_4, \$)\},$
$\delta(q_3, \$, 1) = \{(q_3, A\$)(q_7, \$)\},$ $\quad\delta(q_3, A, 0) = \{(q_3, AA)(q_4, A)\},$
$\delta(q_3, A, 1) = \{(q_3, AA)(q_7, A)\},$ $\quad\delta(q_4, A, 1) = \{(q_4, \varepsilon,)\},$
$\delta(q_4, A, 0) = \{(q_4, \varepsilon)\},$ $\quad\delta(q_4, \$, 0) = \{(q_5, A\$)\},$

$\delta(q_4,\$,1)=\{(q_5,A\$),(q_6,\varepsilon)\}$, $\delta(q_7,A,0)=\{(q_7,\varepsilon)\}$,
$\delta(q_7,A,1)=\{(q_7,\varepsilon)\}$, $\delta(q_7,\$,1)=\{(q_8,A\$)\}$,
$\delta(q_7,\$,0)=\{(q_8,A\$),(q_6,\varepsilon)\}$, $\delta(q_5,A,0)=\{(q_5,AA)\}$,
$\delta(q_5,A,1)=\{(q_5,AA),(q_6,\varepsilon)\}$, $\delta(q_8,A,1)=\{(q_8,AA)\}$,
$\delta(q_8,A,0)=\{(q_8,AA),(q_6,\varepsilon)\}$, $\delta(q_6,A,0)=\{(q_6,\varepsilon)\}$,
$\delta(q_6,A,1)=\{(q_6,\varepsilon)\}$, $\delta(q_6,\$,0)=\{(q_6,\varepsilon)\}$,
$\delta(q_6,\$,1)=\{(q_6,\varepsilon)\}$.

Să urmărim în continuare configurațiile de acceptare pentru șirurile $z_1 = 01\boxed{1}01\boxed{0}$ și $z_2 = 01\boxed{0}01\boxed{1}$.

• $(q_0,\$,011010) \vdash (q_3,\$,011010) \vdash (q_3,A\$,11010) \vdash (q_3,AA\$,1010) \vdash (q_7,AA\$,010) \vdash (q_7,A\$,10) \vdash (q_7,\$,0) \vdash (q_6,\varepsilon,\varepsilon)$ – acceptare.

• $(q_0,\$,010011) \vdash (q_3,\$,010011) \vdash (q_3,A\$,10011) \vdash (q_3,AA\$,0011) \vdash (q_4,AA\$,011) \vdash (q_4,A\$,11) \vdash (q_4,\$,1) \vdash (q_6,\varepsilon,\varepsilon)$ – acceptare.

14. Indicații la lucrările practice

Pentru prima lucrare se recomandă materialul și exemplele expuse în paragrafele 2, 3, 4. Pentru lucrarea 2 se recomandă paragraful 7 și Tabelul 29, unde sunt prezentate câteva repere utile la realizarea lucrării.

Numărul lucrării	Producții generate	Simboluri neterminale	Simboluri neproductive	Producții după eliminarea simbolurilor neproductive	Limbajul generat
1	19	13	9	5	$L = \{a^n ba^n \mid n \geq 1\}$
2	14	11	5	7	$L = \{abca^n \mid n \geq 2\}$
3	23	13	9	5	$L = \{b^n c^{n+1} \mid n \geq 1\}$
4	22	16	11	6	$L = \{ba^n ba^n \mid n \geq 1\}$
5	12	9	4	6	$L = \{aaca^n \mid n \geq 1\}$
6	19	17	12	7	$L = \{a, adb, addb\}$
7	11	7	3	6	$L = \{a^{m+n} b^m \mid m \geq 1, n \geq 0\}$
8	14	11	5	7	$L = \{dacd^n \mid n \geq 2\}$
9	23	19	13	7	$L = \{(abc)^n \mid n \geq 1\}$
10	15	9	4	8	$L = \{a^m b^n \mid n \geq m \geq 1\}$
11	14	11	5	7	$L = \{bacb^n \mid n \geq 2\}$
12	12	9	4	6	$L = \{(ab)^n \mid n \geq 1\}$
13	13	11	4	10	$L = \{d^{2n}\{aab, dd\} \mid n \geq 0\}$
14	12	7	3	6	$L = \{ab\{b, c\}^* a\}$
15	12	9	4	6	$L = \{aab^n \mid n \geq 2\}$
16	21	16	10	8	$L = \{ad^n b^n \mid n \geq 0\}$
17	12	7	3	6	$L = \{ad\{c, d\}^* a\}$
18	12	9	4	6	$L = \{aab^n cc \mid n \geq 0\}$
19	25	13	9	5	$L = \{a^n bc^{n+1} \mid n \geq 1\}$
20	12	9	4	6	$L = \{bd^n cd \mid n \geq 1\}$
21	12	7	3	6	$L = \{bd\{c, d\}^* a\}$
22	20	16	11	6	$L = \{a^{n+1} b^n \mid n \geq 1\}$
23	12	9	4	6	$L = \{ad^n ca \mid n \geq 1\}$
24	12	7	3	6	$L = \{ba\{a, c\}^* b\}$
25	20	16	11	6	$L = \{aab^{2n} \mid n \geq 0\}$

Figura 29: Indicații la lucrarea practică numărul 2

Bibliografie

[1] Hopcroft J. E., Ullman J. D. Formal languages and their relation to automata. Addison-Wesley Educational Publishers Inc., 1969, 288 p. 70

[2] Hopcroft, J. E., Motwani, R., Ullman, J.D. Introduction to Automata Theory, Languages, and Computation, Addison Wesley, 1979, 427 p. 70

[3] Sipser, M. Introduction to the Theory of Computation (2nd Edition), Thomson Course Technology, 2006, 453 p. 70

[4] Linz, P. An Introduction to Formal Languages and Automata. Third Edition, Jones and Bartlett Publishers, 2001, 410 p. 70

[5] Harrison M. A. Introduction to Formal Language Theory, Addison–Wesley, 1978, 594 p. 70

[6] Salomaa, A. Formal Languages, Academic Press, 1973, 332 p. 70

[7] Sudkamp, T.A. Languages and Machines: An Introduction to the Theory of Computer Science, 3rd Edition, Addison Wesley, 2006, 654 p. 70

[8] Căzănescu, V. E. Introducere în teoria limbajelor formale. Editura Academiei, 1983, 165 p. 70

[9] Atanasiu, A. Limbaje formale și automate, Editura InfoData Cluj, 2007, 174 p. 70

[10] Jucan, T. Limbaje formale si automate. Ed. MatrixRom, Bucuresti, 1999 70

[11] Grigoraș, Gh. Limbaje formale și tehnici de compilare, Tipografia Universității "Alexandru Ioan Cuza", Iași, 1984, 260 p. 70

[12] Atanasiu A., Mateescu A. Limbaje formale. Culegere de probleme. Universitatea Bucureşti, 1990, 307 p. 70

[13] Atanasiu, A. Bazele informaticii.- Universitatea Bucuresti, 1987. 70

[14] Oettinger, A. G. Syntactic analysis and the pushdown store. - Proceedings of the 12th Symposia in Applied Mathematics, Providence, RI, 1961, American Mathematical Society, 104-109. 70

[15] Chomsky, N. Context-free grammars and pushdown storage.- Quarterly Progress Report 65, MIT Research Laboratories of Electronics, 1962, 187-194. 70

[16] Schützenberger, M. "On context-free languages and pushdown automata," Information and Control 6, 1963, 246-264. 70

[17] Evey, J. Application of pushdown store machines. -Proceedings 1963 Fall Joint Computer Conference, Montvale, NJ: AFIPS Press, 1963, 215-227. 70

[18] Knuth, D. E. On the translation of languages from left to right. - Information and Control. 8 (6), 1965, 607–639. 3

[19] Steele, Guy L. Common Lisp the Language, 2nd edition, Thinking Machines, Inc. Digital Press, ISBN 1-55558-041-6, 1990, 1029 pp. 46

[20] `https://clisp.sourceforge.io/`, 46

[21] `http://sourceforge.net/projects/clisp/files/clisp/2.49/clisp-2.49-win32-mingw-big.exe/download` 46

[22] `http://www.daansystems.com/lispide/` 46

Notații și abrevieri

$\boldsymbol{AF}$	Automat finit
$\boldsymbol{AFD}$	Automat finit determinist
$\boldsymbol{ASD}$	Automat cu memorie stivă determinist
$\boldsymbol{AS_F}$	Automat cu memorie stivă cu acceptare prin stări finale
$\boldsymbol{AS_V}$	Automat cu memorie stivă cu acceptare prin stivă vidă
$\boldsymbol{AS}$	Automat cu memorie stivă
$\boldsymbol{L(AS)}$	Limbaj acceptat (recunoscut) de către AS
$\boldsymbol{GIC}$	Gramatică independentă de comtext
$\boldsymbol{LIC}$	Limbaj independent de comtext
$\boldsymbol{L(G)}$	Limbaj generat de către gramatica G
$\boldsymbol{L(GIC)}$	Limbaj generat de către gramatica GIC, limbaj independent de context
LIFO	Last-In-First-Out (ultimul-sosit-primul-plecat)
$\boldsymbol{c_i \vdash_{AS} c_j}$, $\boldsymbol{c_i \vdash c_j}$	AS trece direct din configurația $\boldsymbol{c_i}$ în configurația $\boldsymbol{c_j}$
$\boldsymbol{c_i \overset{\leq n}{\vdash} c_j}$	AS trece la k pași din configurația c_i în configurația c_j, $0 \leq k \leq n$
$\boldsymbol{c_i \overset{n}{\vdash} c_j}$, $\boldsymbol{c_i \overset{*}{\vdash} c_j}$	AS trece la n pași din configurația c_i în configurația c_j, $n \geq 0$

Glosar