LIMBAJE FORMALE ŞI TRANSLATOARE - NOTIȚE DE CURS

MIHAI-LICĂ PURA

Departamentul de Calculatoare şi Securitate Cibernetică
Facultatea de Sisteme Informatice şi Securitate Cibernetică
Academia Tehnică Militară F̈erdinand I¨

CUPRINS

1

INTRODUCERE ÎN TEORIA LIMBAJELOR FORMALE

1.1 LIMBAJE FORMALE

Limbajul este o modalitate sistematică de comunicare care utilizează sunete sau simboluri convenţionale. Referindu-ne la limbajele de programare, simbolurile convenţionale sunt şiruri de caractere.

Pentru a defini un şir de caractere trebuie să pornim de la un alfabet.

Definitie 1 *Un **alfabet** este o mulţime finită de simboluri.*

Exemple:

- ★ $\Sigma_1 = \{a, ă, â, b, c, d, \ldots, z\}$ este mulţimea literelor din alfabetul limbii române.
- ★ $\Sigma_2 = \{0, 1, \ldots, 9\}$ este mulţimea cifrelor în baza 10.
- ★ $\Sigma_3 = \{a, b, \ldots, z, \#\}$ este mulţimea literelor din alfabetul latin, plus simbolul special #.

Definitie 2 *O succesiune finită de simboluri din alfabetul Σ se numeşte **şir de simboluri** peste alfabetul dat.*

Exemple:

- ★ *abfbz* este şir de caractere peste $\Sigma_1 = \{a, b, c, d, \ldots, z\}$,
- ★ *9021* este şir de caractere peste $\Sigma_2 = \{0, 1, \ldots, 9\}$,
- ★ *ab#bc* este şir de caractere peste $\Sigma_3 = \{a, b, \ldots, z, \#\}$.

Definitie 3 *Un **limbaj** este o mulţime de şiruri (finită sau infinită) de simboluri peste acelaşi alfabet.*

Dacă limbajul este finit atunci el poate să fie definit prin **enumerare**. Cum poate fi însă definit un limbaj infinit? Există două mecanisme distincte de definire a limbajelor: prin **generare** sau prin **recunoaştere**.

În primul caz este vorba de un "dispozitiv" care ştie să genereze toate propoziţiile din limbaj (şi numai pe acestea), astfel încât alegând o propoziţie din limbaj, dispozitivul va ajunge sa genereze propoziţia respectivă într-un interval finit de timp. Din această categorie fac parte gramaticile formale.

În al doilea caz este vorba de un "dispozitiv" care ştie să recunoască (să accepte ca fiind corecte) propoziţiile limbajului dat (şi numai pe acestea). Din această categorie fac parte expresiile regulate şi automatele finite.

2

GRAMATICI FORMALE

Pentru a vedea care este intuiţia din spatele conceptului de gramatici formale, vom porni de la exemplul următor. Să presupunem că se dă o mulţime de cuvinte:

{ casa, străluceşte, repede, copila, stă, frumos, câinele, aleargă, bine }

Şi trebuie să formăm propoziţii de câte trei cuvinte care să aibă înţeles în limba română. Dacă am alege la întâmplare cele trei cuvinte, atunci avem toate şansele de a forma propoziţii care nu au niciun înţeles, cum ar fi, de exemplu:

"repede casa bine"

Pentru a elimina astfel de situaţii, am putea proceda în felul următor. Ştim că o propoziţie are sens dacă ea exprimă o acţiune, adică dacă are ceea ce gramatica limbii române numeşte *predicat*. Această acţiune ar putea fi realizată de către cineva. Gramatica limbii române îl numeşte pe acest cineva *subiect*. În plus, acţiunea ar putea să fie caracterizată într-un anume fel, gramatica limbii române numind această caracterizare *complement circumstanţial de mod*. Printr-o astfel de judecată, am stabilit practic o structură a propoziţiilor cu sens, pe care le putem forma pornind de la o mulţime de cuvinte. Să notăm atunci această structură sub forma unei înlocuiri:

Start $\rightarrow$ *Subiect Predicat Complement*

Apoi, să grupăm cuvintele din mulţimea dată, în funcţie de partea propoziţiei care ar putea să fie. De exemplu, $casa, copila, câinele$ ar putea fi subiecte. Putem scrie aceasta tot sub forma unei înlocuiri:

Subiect $\rightarrow$ *casa* | *copila* | *câinele*

Analog putem proceda şi pentru predicat, şi pentru complement. În acest mod vom obţine patru înlocuiri cu ajutorul cărora, pornind de la Start, putem obţine o propoziţie cu sens folosind cuvintele din mulţimea dată.

Start $\rightarrow$ *Subiect Predicat Complement*
Subiect $\rightarrow$ *casa* | *copila* | *câinele*
Predicat $\rightarrow$ *străluceşte* | *stă* | *aleargă*
Complement $\rightarrow$ *repede* | *frumos* | *bine*

Mulţimea de cuvinte de la care am plecat, împreună cu mulţimea de elemente ajutătoare (*{ start, subiect, predicat, complement}*) şi împreună cu cele patru înlocuiri de mai sus formează o gramatică formală. Observăm deci că o gramatică formală este un instrument care permite descrierea şi analiza unui limbaj, pe baza unei mulţimi de reguli, care prezintă modul în care se construiesc propoziţiile valide (şi numai acestea) din limbajul ţintă.

2.1 GRAMATICI FORMALE

În continuare vom prezenta sintaxa gramaticilor formale. Pentru aceasta este nevoie să definim operatorii steluţa Kleene şi plusul Kleene.

Fie V o mulţime de simboluri (un alfabet). Definim, prin inducţie, următoarele mulţimi:

- $V_0 = \{\epsilon\}$,
- $V_1 = V$,
- ...

★ $V_{i+1} = \{wv | w \in V_i \ , \ v \in V\}, \forall i > 0.$

Definitie 4 *Operatorul **steluţa Kleene** asupra unei mulţimi V este definit astfel:*

$$V^* = \bigcup_{i \in \mathbb{N}} V_i = \{\epsilon\} \cup V_1 \cup V_2 \cup V_3 \cup \dots$$

Definitie 5 *Definiţia operatorului **plusul Kleene** asupra unei mulţime V este:*

$$V^+ = \bigcup_{i \in \mathbb{N} \setminus \{0\}} V_i = V_1 \cup V_2 \cup V_3 \cup \dots$$

Altfel spus, V^* este mulţimea tuturor şirurilor de simboluri peste V, iar V^+ este mulţimea tuturor şirurilor de simboluri peste V cu excepţia şirului vid.

Definitie 6 *O **gramatică** formală este un cvadruplu* $G = (V_N, \Sigma, S, P)$*, unde:*

★ V_N *se numeşte mulţimea neterminalelor,*

★ Σ *se numeşte mulţimea terminalelor,*

★ *S este simbolul iniţial (sau simbolul de start),* $S \in V_N$*,*

★ *P este mulţimea regulilor de producţie, ale cărei elemente au forma:*
$V^* V_N V^* \to V^*$*, unde* $V = V_N \cup \Sigma$ *se numeşte alfabetul gramaticii.*

Exemplu:
Fie gramatica $G = (V_N, \Sigma, S, P)$, astfel încât:

★ V_N = {Subiect, Predicat, Propoziţie},

★ Σ = {Ana, Ion, învaţă, doarme, ., ' '},

★ S este simbolul de start și anume Propoziţie,

★ P = {
  - Propoziţie → Subiect ' ' Predicat .,
  - Subiect → Ana | Ion,
  - Predicat → învaţă | doarme

}

Iată convenţiile de notaţie care vor fi folosite de-a lungul lucrării:

★ literele mici de la începutul alfabetului latin (a, b, c, ...) reprezintă elemente din mulţimea Σ (simboluri terminale),

★ literele mici de la sfârşitul alfabetului latin (u, v, x, ...) reprezintă elemente din mulţimea Σ^* (şiruri de simboluri terminale),

★ literele mari de la începutul alfabetului latin (A, B, C, ...) reprezintă elemente din V_N (simboluri neterminale),

★ literele mari de la sfârşitul alfabetului latin (U, V, X, ...) reprezintă elemente din $V_N \cup \Sigma$ (simboluri terminale sau neterminale),

★ literele alfabetului grecesc reprezintă şiruri din $(V_N \cup \Sigma)^*$ (şiruri de simboluri terminale şi neterminale).

Din exemplele de gramatici furnizate se poate observa că o gramatică formală definește numai lexicul (vocabularul) și sintaxa propozițiilor unui limbaj. Ea nu permite definirea semanticii propozițiilor unui limbaj. De asemenea, din definiția unei gramatici se poate observa că aceasta cuprinde lexicul (vocabularul) acestui meta-limbaj, precum și sintaxa regulilor de producție, însă nu cuprinde și semantica regulilor de producție. Aceasta va fi definită implicit prin definirea relației de derivare. În continuare vom prezenta semantica gramaticilor formale.

Fie M o mulţime. Orice mulţime de perechi ordonate $R = \{(a, b) | \ a, b \in M\}$, se numeşte relaţie binară peste M ($R \subseteq A \times A$).

Definitie 7 *Fie o gramatică* $G = (V_N, \Sigma, S, P)$. ***Derivarea într-un pas (derivarea directă)*** *este o relaţie binară între şiruri din* V^* *notată cu* $\Rightarrow$ *(*$\Rightarrow \subseteq V^* \times V^*$*) şi definită astfel:*
$\alpha \Rightarrow \beta$ *dacă şi numai dacă* $\alpha = xpy$ *şi* $\beta = rqt$, $x, p, y, r, q, t \in V^*$ *şi* $\exists\ p \to q \in P$.

Exemplu (pe baza gramaticii anterioare):
Subiect ' ' Predicat . $\Rightarrow$ Ana ' ' Predicat .

Definitie 8 *Fie o gramatică* $G = (V_N, \Sigma, S, P)$. ***Derivarea în k paşi*** *este o relaţie binară între şiruri din* V^* *notată cu* $\Rightarrow^k$ *şi definită astfel:*
$\gamma \Rightarrow^k \delta$ *dacă şi numai dacă* $\exists\ \alpha_1, \alpha_2, \ldots \alpha_{k+1}$ *astfel încât* $\alpha_1 = \gamma, \alpha_{k+1} = \delta$ *şi* $\alpha_i \Rightarrow \alpha_{i+1}, \forall\ i \in [1, k]$.

Exemplu (pe baza gramaticii anterioare):
Subiect ' ' Predicat . $\Rightarrow$ Ana ' ' Predicat .

Definitie 9 ***Închiderea tranzitivă a relaţiei de derivare*** *se notează* $\Rightarrow^+$ *şi se defineşte folosind relaţia de derivare în k paşi, astfel:*
$\gamma \Rightarrow^+ \delta$ *dacă şi numai dacă* $\exists\ k \geqslant 1$ *astfel încât* $\gamma \Rightarrow^k \delta$.

Definitie 10 ***Închiderea tranzitivă şi reflexivă a relaţiei de derivare*** *se notează* $\Rightarrow^*$ *şi se defineşte folosind închiderea trazitivă a relaţiei de derivare, astfel:*
$\gamma \Rightarrow^* \delta$ *dacă şi numai dacă* $\gamma = \delta$ *sau* $\gamma \Rightarrow^+ \delta$.

Definitie 11 *O* ***formă propoziţională*** *peste o gramatica* $G = (V_N, \Sigma, S, P)$ *se defineşte recursiv (inductiv) în modul următor:*

1. ***S*** *este o formă propoziţională.*
2. *Dacă* ***aBt*** *este o formă propoziţională şi dacă există o regulă de producţie* $B \to r \in P$, *atunci* ***art*** *este o formă propoziţională.*

Exemplu:
Propoziţie; Subiect Predicat.; Ana Predicat.; Subiect învaţă.; Ana învaţă.

Definitie 12 *O forma propoziţională peste o gramatică G, care conţine numai simboluri terminale, se numeşte* ***propoziţie*** *generata de G.*

Exemplu:
Ana învaţă.; Ion doarme.
Forma propoziţională şi propoziţia pot fi definite şi folosind relaţia de derivare. Dăm mai jos şi aceste două definiţii.

Definitie 13 *O **formă propoziţională** peste o gramatica* $G = (V_N, \Sigma, S, P)$ *este orice şir* $\alpha \in V^*$ *cu proprietatea că* $S \Rightarrow^* \alpha$.

Forma propoziţională este deci orice succesiune de simboluri terminale şi/sau neterminale care poate fi obţinută pornind de la simbolul de start prin o sau mai multe derivări.

Definitie 14 *O **propoziţie** peste o gramatica* $G = (V_N, \Sigma, S, P)$ *este orice şir* $x \in \Sigma^*$ *cu proprietatea că* $S \Rightarrow^+ x$.

Propoziţia este deci orice succesiune de simboluri terminale care poate fi obţinută pornind de la simbolul de start prin cel puţin o derivare.

Definitie 15 *Limbajul generat de o gramatica* $G = (V_N, \Sigma, S, P)$ *se notează cu L(G) şi reprezintă mulţimea tuturor propoziţiilor care pot fi generate de către această gramatică:*
$L(G) = \{x \in \Sigma^* | S \Rightarrow^+ x\}$.

Exemplu:
$L(G) = \{$Ana învaţă., Ion învaţă., Ana doarme., Ion doarme.$\}$
O gramatica este o reprezentare finită (toate elementele sale sunt finite) a unui limbaj care poate să fie infinit. Nu orice limbaj are o reprezentare finită, cu alte cuvinte nu pentru orice limbaj există o gramatică care să îl reprezinte. Dacă este posibilă o astfel de construcţie, atunci pentru un acelaşi limbaj dat, se pot construi mai multe gramatici distincte care să îl genereze.

Singura modalitate prin care, pe baza unei gramatici, se pot genera propoziții cu un număr oarecare de simboluri este recursivitatea. Aceasta poate să apară la nivelul regulilor de producție și poate să fie recursivitate stanga ($A \Rightarrow^* A\alpha$) sau recursivitate dreapta ($A \Rightarrow^* \alpha A$), recursivitate directă ($A \to A\alpha$ și $A \to \alpha A$) sau recursivitate indirectă ($A \to \gamma B \beta, B \Rightarrow^* A\alpha$ și analog pentru recursivitatea dreapta).

De exemplu, fie gramatica $G = (V_N, \Sigma, S, P)$, astfel încât:

- ★ V_N = {Subiect, Predicat, Propoziţie, Frază},
- ★ Σ = {Ana, Ion, învaţă, doarme, ., ' '},
- ★ S este simbolul de start și anume Frază,
- ★ P = {
  - Frază → Propoziţie Frază | Propoziţie
  - Propoziţie → Subiect ' ' Predicat .,
  - Subiect → Ana | Ion,
  - Predicat → învaţă | doarme

  }

Pe baza acestei gramatici se pot genera fraze care sunt formate din una, două sau un număr oarecare de propoziții.

Atunci când derivăm o formă propoziţională, putem ajunge în situaţia în care aceasta să conţină mai multe simboluri neterminale. Prin urmare va trebui să alegem pentru care dintre simbolurile neterminale vom aplica următoarea regulă de producţie pentru a face o derivare directă. Dacă într-o astfel de situaţie se alege întotdeauna neterminalul cel mai din stânga, atunci derivarea se va numi *derivare stânga*. Dacă îl alegem întotdeauna pe neterminalul cel mai din dreapta, atunci derivarea se va numi *derivare dreapta*.

Exemplu:
Fie gramatica $G = (V_N, \Sigma, S, P)$, astfel încât:

- ★ V_N = {E},
- ★ Σ = {+, -, *, /, (,), id},
- ★ S este neterminalul E,
- ★ P = {
  - E → E + E,
  - E → E - E,
  - E → E * E
  - E → E / E
  - E → (E)
  - E → id

  }

Următorul şir de derivări, care reprezintă generarea propoziţiei *id * (id + id)*, este format din derivări stânga.

$$S \Rightarrow E * E \Rightarrow \mathbf{id} * E \Rightarrow id * (E) \Rightarrow id * (E + E) \Rightarrow id * (\mathbf{id} + E) \Rightarrow id * (id + id)$$

Pentru aceeaşi propoziţie, următorul şir de derivări este format din derivări dreapta.

$$S \Rightarrow E * E \Rightarrow E * \mathbf{(E)} \Rightarrow E * (\mathbf{E + E}) \Rightarrow E * (E + \mathbf{id}) \Rightarrow E * (\mathbf{id} + id) \Rightarrow \mathbf{id} * (id + id)$$

2.2 IERARHIA CHOMSKY

Ierarhia Chomsky este o ierarhie de clase de limbaje formale, care arată ce relaţie de incluziune există între acestea. Ierarhia a fost descrisă pentru prima dată de către lingvistul Naom Chomsky, într-un articol publicat în anul 1956. Apartenenţa unui limbaj la unul dintre tipurile ierarhiei se determină în funcţie de forma pe care o au regulile de producţie ale unei gramatici care generează limbajul respectiv. Din această cauză, ierarhia Chomsky poate fi văzută şi ca o ierarhie de clase de gramatici formale.

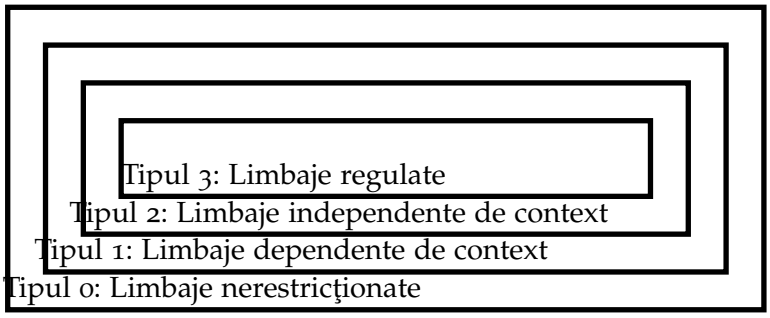


Figura 1: Ierarhia Chomsky

Primul tip de gramatici din ierarhia Chomsky este tipul 0 şi este reprezentat de către **gramaticile nerestricţionate**. Nicio restricţie nu este impusă asupra regulilor de producţie ale unei astfel de gramatici.

Un limbaj este limbaj de tipul 0, dacă şi numai dacă el este generat de către o gramatică de tipul 0.

Gramaticile de tipul 1 sunt gramaticile dependente de context.

Definitie 16 *O gramatică* $G = (V_N, \Sigma, S, P)$ *se numeşte* ***gramatică dependentă de context*** *dacă şi numai dacă* ***toate*** *regulile sale de producţie au forma* $\alpha A \beta \to \alpha \gamma \beta$, *unde* $A \in V_N$, $\alpha, \beta \in V^*$ *şi* $\gamma \in V^+$.

Observaţie: Unii autori permit şi regulile de producţie de forma $S \to \epsilon$ în cadrul mulţimii regulilor de producţie ale unei gramatici dependente de context, însă cu condiţia ca S să nu apară în partea dreaptă a niciunei reguli de producţie.

Un limbaj este dependent de context, dacă şi numai dacă el este generat de către o gramatică dependentă de context.

Exemplu:

Fie gramatica $G = (V_N, \Sigma, S, P)$, unde:

- ★ V_N = { Propoziţie, Subiect, Atribut, Predicat, Complement, Substantiv, Verb, Adjectiv, Adverb},
- ★ Σ = { a merge, a cânta, este, face, copilul, câinele, frumos, rău, bine, mult, ., ' '},
- ★ S este Propoziţie,
- ★ P = {
  - Propoziţie → Subiect [Atribut] Predicat [Complement].,
  - Subiect → Substantiv | Verb,
  - Atribut → Adjectiv,
  - Predicat → Verb,
  - Complement → Adverb,
  - Verb Atribut → a merge Atribut | a cânta Atribut,
  - Verb Complement → este Complement | face Complement,
  - Substantiv → copilul | câinele,
  - Adjectiv → frumos | rău,
  - Adverb → bine | mult

  }

Gramaticile de tipul 2 sunt gramaticile independente de context.

Definitie 17 *O gramatică* $G = (V_N, \Sigma, S, P)$ *se numeşte* ***gramatică independentă de context*** *dacă şi numai dacă* ***toate*** *regulile sale de producţie au forma* $A \to u$, *unde* $A \in V_N$ *şi* $u \in V^*$.

Un limbaj este independent de context, dacă şi numai dacă el este generat de către o gramatică independentă de context.

Gramaticile regulate reprezintă gramaticile de tipul 3.

Definitie 18 *O gramatică* $G = (V_N, \Sigma, S, P)$ *se numeşte* ***gramatică liniară dreapta*** *dacă este o gramatică independentă de context în care fiecare regulă de producţie este de forma* $A \to a$ *sau de forma* $A \to aB$ *sau de forma* $A \to \epsilon$, *unde* $a \in \Sigma$, *iar* $A, B \in V_N$.

Definitie 19 *O gramatică* $G = (V_N, \Sigma, S, P)$ *se numeşte* ***gramatică liniară stânga*** *dacă este o gramatică independentă de context în care fiecare regulă de producţie este de forma* $A \to a$ *sau de forma* $A \to Ba$ *sau de forma* $A \to \epsilon$, *unde* $u \in \Sigma$, *iar* $A, B \in V_N$.

Definitie 20 *O gramatică se numeşte **gramatică regulată** dacă este o gramatică liniară dreapta sau stânga.*

Un limbaj este regulat dacă şi numai dacă el este generat de către o gramatică regulată.

Acum că am văzut definiţia pentru fiecare tip de limbaj/gramatică în parte, putem să explicăm semnificaţia ierarhiei Chomsky. După cum am precizat la începutul secţiunii, ierarhia Chomsky arată relaţia de incluziune dintre cele patru clase de limbaje definite. Prin urmare putem spune că: orice limbaj regulat este şi un limbaj independent de context; orice limbaj independent de context este şi un limbaj dependent de context; şi orice limbaj dependent de context este şi un limbaj nerestricţionat. Analog, putem spune că: nu orice limbaj nerestricţionat este şi un limbaj dependent de context; nu orice limbaj dependent de context este şi limbaj independent de context; şi nu orice limbaj independent de context este şi limbaj regulat.

Având în vedere restricţiile impuse asupra regulilor de producţie ale fiecărui tip de gramatici formale, se observă că limbajele regulate sunt cele mai simple, iar limbajele nerestricţionate sunt cele mai complexe. Cu alte cuvinte, complexitatea limbajelor scade de la tipul 0 către tipul 3. Iată câte un exemplu de limbaj pentru fiecare clasă în parte. Limbajele naturale sunt limbaje de tipul 0. Limbajele de programare sunt limbaje de tipul 2. (De fapt, unele aspecte ale limbajelor de programare le-ar clasifica ca şi limbaje de tipul 1, însă în practică se preferă definirea lor prin gramatici independente de context, adăugându-se câteva restricţii suplimentare.) Limbajul atomilor lexicali ai unui limbaj de programare (cuvintele cheie, numele de funcţii, variabile, structuri, clase, ş.a., constantele numerice, constantele literale, constantele de tip şir de caractere, operatorii, semnele de punctuaţie) este un limbaj de tipul 3.

Am văzut că gramaticile formale permit definirea limbajelor prin generare. Adică, pornind de la simbolul de start, prin derivări succesive, putem genera orice propoziţie a limbajului definit de gramatica respectivă. Cum se poate rezolva problema inversă? Fiind dată o propoziţie oarecare, cum se poate stabili dacă ea aparţine unui anumit limbaj (definit de o gramatică dată)? Pentru aceasta se foloseşte un "dispozitiv" capabil sa recunoască dacă o propoziţie dată aparţine sau nu limbajului pe care el îl defineşte. În capitolul introductiv am arătat că astfel de dispozitive sunt automatele finite.

Pentru fiecare clasă de limbaje în parte, s-a propus câte un model matematic al unui dispozitiv capabil să facă acest lucru, astfel: pentru limbajele regulate, acest lucru se poate face cu ajutorul unui automat finit, pentru limbajele independente de context este nevoie de un automat finit cu stivă, pentru limbajele dependente de context este nevoie de o maşină Turing cu bandă limitată, iar pentru limbajele nerestricţionate este nevoie de o maşină Turing.

Scopul acestui curs este prezentarea teoriei traducerii/compilării pentru limbajele de programare. Limbajele de programare pot fi definite prin gramatici independente de context, iar vocabularul unui limbaj de programare este un limbaj regulat. De aceea ne vom opri numai asupra analizei pentru limbajele regulate şi pentru cele independente de context.

2.3 ARBORI SINTACTICI

Fie M o mulţime de noduri şi R o relaţie binară peste M. Atunci $G = (M, R)$ se numeşte **graf orientat**. Dacă $(a, b) \in R$, atunci definim următoarele mulţimi:

- ★ $input(a) = \{b|(b, a) \in R\}$,
- ★ $output(a) = \{b|(a, b) \in R\}$.

Un arbore este un graf orientat (M, R) cu următoarele restricţii:

★ $\exists\, r \in M$ unic, astfel încât $input(r) = \emptyset$,

★ $\forall\, a \in M,\ a \neq r,\ card(input(a)) = 1$,

★ $\forall\, a \in M,\ (r, a) \in R^*$.

Definitie 21 *Fie o gramatică formală* $G = (V_N, \Sigma, S, P)$. *Un* ***arbore de derivare*** *(arbore sintactic)* $A = (M, R)$ *este un arbore etichetat ordonat de la stânga la dreapta cu proprietăţile:*

1. *rădăcina arborelui este etichetată cu* S,
2. *pentru orice nod* $a \in M$ *cu descendenţi, astfel încât A este eticheta nodului* a, *iar* $X_1, X_2, \ldots, X_i$ *sunt etichetele descendenţilor lui a,* $\exists\, A \to X_1 X_2 \ldots X_i \in P$.

Exemplu:
Arborele sintactic corespunzător propoziţiei *Ana învaţă.* este următorul:

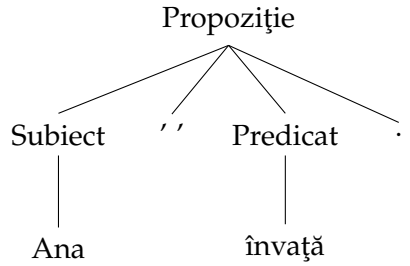


Figura 2: Arborele sintactic descendent pentru propoziţia *Ana învaţă.*

Arborii sintactici sunt o reprezentare arborescentă a succesiunii de derivări prin care se generează o anumită propoziţie. Fiecare nod al arborelui care nu este frunză, este etichetat cu un neterminal. Fiecare frunză a arborelui este etichetată cu un terminal.

Citind etichetele nodurilor arborelui (de la stânga la dreapta) de pe un nivel dat se obţine o formă propoziţională.

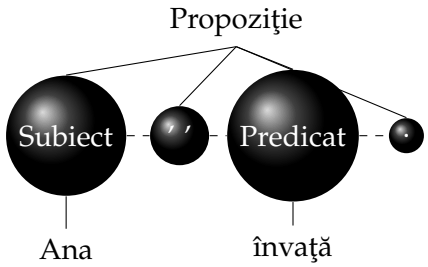


Figura 3: Forma propoziţională *Subiect ' ' Predicat .*

Citind etichetele frunzelor arborelui (de la stânga la dreapta) se obţine propoziţia pentru care a fost construit arborele.

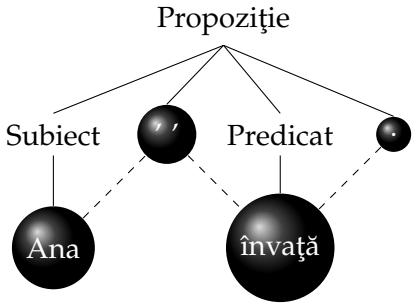


Figura 4: Propoziţia *Ana învaţă.*

Construcţia arborelui sintactic pentru o propoziţie dată se poate face în două moduri. Se poate porni de la rădăcină către frunze, adică de la simbolul de start către propoziţie. Această manieră se numeşte construcţie descendentă. Sau se poate porni de la frunze către rădăcină, adică de la propoziţie către simbolul de start. Această manieră de construcţie se numeşte ascendentă.

Verificarea dacă o propoziţie dată aparţine limbajului definit de o gramatică formală se numeşte analiză sintactică. Ea constă în încercarea de a construi arborele sintactic al propoziţiei date, conform gramaticii care defineşte limbajul respectiv. Dacă se poate construi un arbore sintactic pentru propoziţie, atunci înseamnă că ea poate fi generată de către gramatică şi deci aparţine limbajului. Dacă nu se poate construi un astfel de arbore, atunci înseamnă că propoziţia nu aparţine limbajului. Dacă analiza încearcă construirea arborelui sintactic în maniera descendentă, atunci ea se numeşte analiză sintactică descendentă. Dacă încearcă construirea arborelui în sens ascendent, atunci se numeşte analiză sintactică ascendentă. Evident arborele sintactic descendent al unei propoziţii este identic cu arborele sintactic ascendent al aceleiaşi propoziţii.

Exemplu:

Arborele sintactic din exemplul precedent este un arbore sintactic descendent. Iată arborele sintactic ascendent pentru aceeaşi propoziţie.

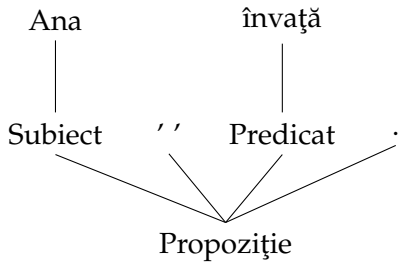


Figura 5: Arborele sintactic ascendent pentru propoziţia *Ana învaţă.*

2.4 EXERCIŢII

Exerciţiul 1:

Fie gramatica $G = (V_N, \Sigma, S, P)$. Determinaţi regulile de producţie, mulţimea terminalelor şi a neterminalelor şi simbolul de start, astfel încât gramatica să definească limbajul numerelor naturale.

★ $V_N = \{N, L_c, C\}$,

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

★ S este neterminalul N,

★ P = {
  - $N \rightarrow L_c$,
  - $L_c \rightarrow CL_c|C$,
  - $C \rightarrow 0|1|2|3|4|5|6|7|8|9$

}

Alfabetul limbajului este format din toate simbolurile care pot apărea în cadrul propoziţiilor limbajului, adică din toate cifrele de la 0 la 9.

Pentru construirea regulilor de producţie, am folosit următorul raţionament. Se cunoaşte că o gramatică defineşte sintaxa unui limbaj, adică forma propoziţiilor acestuia. Deci, ne-am pus întrebarea, cum arată propoziţiile din limbajul pentru care trebuie să construim gramatica? Aceste propoziţii sunt de fapt numerele naturale. Un număr natural este o succesiune de cifre formată din cel puţin o cifră (C). Deci forma propoziţiilor este cea a unei liste de cifre (L_c). Cum se poate defini o listă de cifre care poate să aibă o infinitate de elemente? Singurul mecanism pe care regulile de producţie ni-l pun la dispoziţie pentru a putea defini o succesiune infinită de simboluri este recursivitatea. Atunci trebuie să stabilim cum putem defini recursiv o listă de cifre. O listă de cifre este formată din primul ei element, care este o cifră, urmat de restul listei de cifre, care este tot o listă de cifre. Iar restul listei de cifre este format din primul său element, care este o cifră, urmat de restul restului listei de cifre, care este tot o listă de cifre, ş.a.m.d. până la ultimul element al listei, care este o listă de cifre format dintr-un singur element, adică o cifră. Prin urmare, cazul de bază al recursivităţii este faptul că cea mai simplă listă de cifre este formată dintr-o singură cifră (de unde a rezultat regula de producţie $L_c \rightarrow C$), iar formula de recurenţă este cea exprimată prin regula de producţie $L_c \rightarrow CL_c$. În plus, într-o gramatică, regulile de producţie ale simbolului de start vor arăta întotdeauna toate formele pe care le pot avea propoziţiile unui limbaj. În acest caz, este vorba de o singură formă ($N \rightarrow L_c$).

Următorul şir de derivări, care reprezintă generarea propoziţiei *2573*, este format din derivări stânga.

$$N \Rightarrow L_c \Rightarrow CL_c \Rightarrow 2L_c \Rightarrow 2CL_c \Rightarrow 25L_c \Rightarrow 25CL_c \Rightarrow 257L_c \Rightarrow 257C \Rightarrow 2573$$

Arborele sintactic descendent corespunzător propoziţiei este:

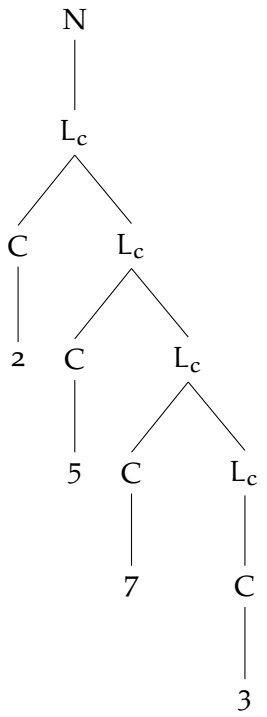


Figura 6: Arborele sintactic descendent pentru propoziţia *2573*

Arborele sintactic ascendent corespunzător propoziţiei este:

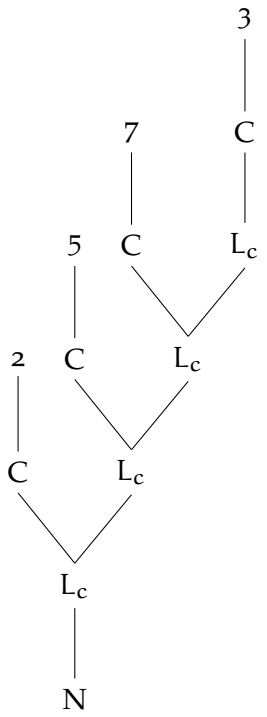


Figura 7: Arborele sintactic ascendent pentru propoziţia *2573*

Analizând gramatica definită mai sus, se poate constata că ea permite generarea numerelor de forma: orice succesiune de zerouri urmată de un număr natural oarecare (de exemplu 00013). Cum se poate modifica gramatica, astfel încât aceste propoziţii să nu mai poată fi generate? Oferim mai jos una dintre modalităţile în care se poate realiza acest lucru.

Fie gramatica $G = (V_N, \Sigma, S, P)$, astfel încât:

- ★ $V_N = \{N, L_c, C, D\}$,
- ★ $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$,
- ★ S este neterminalul N,
- ★ P = {
  - $N \to DL_c|C$,
  - $Lc \to CL_c|C$,
  - $C \to 0|1|2|3|4|5|6|7|8|9$
  - $D \to 1|2|3|4|5|6|7|8|9$

  }

Următorul şir de derivări, care reprezintă generarea propoziţiei *2573*, este format din derivări dreapta.

$N \Rightarrow DL_c \Rightarrow DCL_c \Rightarrow DCCL_c \Rightarrow DCCC \Rightarrow DCC3 \Rightarrow DC73 \Rightarrow D573 \Rightarrow 2573$

Următorul şir de derivări, care reprezintă generarea propoziţiei *2573*, este format din derivări stânga.

$N \Rightarrow DL_c \Rightarrow 2L_c \Rightarrow 2CL_c \Rightarrow 25L_c \Rightarrow 25CL_c \Rightarrow 257L_c \Rightarrow 257C \Rightarrow 2573$

Se observă că ambele tipuri de derivări, adică atât şirul derivărilor stânga, cât şi şirul derivărilor dreapta, pleacă de la aceeaşi formă propoziţională (adică de la simbolul de start) şi ajung la propoziţie. Diferenţa dintre cele două şiruri de derivări constă în formele propoziţionale intermediare, care sunt diferite.

Cu toate acestea arborele corespunzător propoziţiei, fie că este realizat folosind derivări dreapta, fie că este realizat folosind derivări stânga, este unic.

Gramatica de mai sus este o gramatică de tipul 2, adică este o gramatică independentă de context. Dacă limbajul pentru care a fost construită o gramatică este un limbaj regulat, atunci există cel puţin o gramatică regulată care să definească limbajul respectiv. Limbajul numerelor naturale este un limbaj regulat şi, prin urmare, se poate construi o gramatică regulată care să definească acest limbaj. Oferim mai jos o astfel de gramatică.

Fie gramatica $G = (V_N, \Sigma, S, P)$, astfel încât:

- ★ $V_N = \{N, L_c, C\}$,
- ★ $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$,
- ★ S este neterminalul N,
- ★ P = {
  - $N \to 1L_c|2L_c|3L_c|4L_c|5L_c|6L_c|7L_c|8L_c|9L_c|C$,
  - $Lc \to 0L_c|1L_c|2L_c|3L_c|4L_c|5L_c|6L_c|7L_c|8L_c|9L_c|C$,
  - $C \to 0|1|2|3|4|5|6|7|8|9$

  }

O proprietate interesantă a gramaticilor regulate este faptul că, pentru orice propoziţie din limbaj, şirul derivărilor stânga coincide cu şirul derivărilor dreapta. Acest lucru se datorează faptului că formele propoziţionale intermediare între simbolul de start şi propoziţie conţin

un singur neterminal (deoarece la orice derivare se utilizează o singură regulă de producţie, iar partea dreaptă a regulilor de producţie ale unei gramatici regulate conţine un singur neterminal). Şi atunci, indiferent dacă înlocuim mai întâi neterminalul cel mai din stânga sau pe cel mai din dreapta, vom acţiona de fapt asupra aceluiaşi neterminal.

Următorul şir de derivări, care reprezintă generarea propoziţiei *2573*, este format din derivări dreapta.

$N \Rightarrow 2L_c \Rightarrow 25L_c \Rightarrow 257L_c \Rightarrow 257C \Rightarrow 2573$

Următorul şir de derivări, care reprezintă generarea propoziţiei *2573*, este format din derivări stânga.

$N \Rightarrow 2L_c \Rightarrow 25L_c \Rightarrow 257L_c \Rightarrow 257C \Rightarrow 2573$

Se observă că cele două şiruri de derivări sunt identice.

Exerciţiul 2:

Fie gramatica $G = (V_N, \Sigma, S, P)$. Scrieţi regulile de producţie, mulţimea terminalelor şi a neterminalelor şi determinaţi simbolul de start, astfel încât gramatica să definească limbajul numerelor întregi.

Fie gramatica $G = (V_N, \Sigma, S, P)$, astfel încât:

- ★ V_N = {N,L_c,C,D,Z},
- ★ Σ = {+, -, 0, 1, 2, 3,4,5,6,7,8,9},
- ★ S este neterminalul N,
- ★ P = {
  - $N \rightarrow ZDL_c|ZC$,
  - $L_c \rightarrow CL_c|C$,
  - $C \rightarrow 0|1|2|3|4|5|6|7|8|9$
  - $D \rightarrow 1|2|3|4|5|6|7|8|9$
  - $Z \rightarrow +|-$

  }

Alfabetul limbajului este format din toate simbolurile care pot apărea în cadrul propoziţiilor limbajului, adică din toate cifrele de la 0 la 9 şi cele două semne posibile, plus şi minus.

În construcţia regulilor de producţie s-a procedat la fel ca şi în exemplul anterior. S-a plecat de la forma propoziţiilor din limbajul de definit: numerele întregi sunt numere naturale cu semn. Având deci ca punct de plecare gramatica exemplului anterior, s-a adăugat neterminalul Z care reprezintă semnul numărului, neterminal care a fost folosit la începutul părţii drepte a tuturor regulilor de producţie ale simbolului de start.

Următorul şir de derivări, care reprezintă generarea propoziţiei *-50*, este format din derivări stânga.

$N \Rightarrow ZDLc \Rightarrow -DLc \Rightarrow -5Lc \Rightarrow -5C \Rightarrow -50$

Care este şirul derivărilor dreapta, pentru aceeaşi propoziţie?

Arborele sintactic descendent corespunzător propoziţiei este:

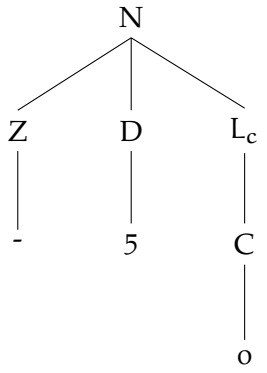


Figura 8: Arborele sintactic descendent pentru propoziţia *-50*

Care este arborele sintactic ascendent corespunzător aceleiaşi propoziţii?

Care este tipul acestei gramatici? Construiţi o gramatică regulată care să definească acelaşi limbaj (limbajul numerelor întregi este un limbaj regulat).

Exerciţiul 3:

Fie gramatica $G = (V_N, \Sigma, S, P)$. Determinaţi regulile de producţie, mulţimea terminalelor şi a neterminalelor şi simbolul de start, astfel încât gramatica să definească limbajul numerelor raţionale.

Fie gramatica $G = (V_N, \Sigma, S, P)$, astfel încât:

★ V_N = {P,N,M,L_c,C,D,Z},

★ Σ = {+, -, /, 0, 1, 2, 3,4,5,6,7,8,9},

★ S este neterminalul P,

★ P = {
  - $P \to N/M$,
  - $N \to ZDL_c|0|ZD$,
  - $M \to ZDL_c|ZD$,
  - $L_c \to CL_c|C$,
  - $C \to 0|1|2|3|4|5|6|7|8|9$
  - $D \to 1|2|3|4|5|6|7|8|9$
  - $Z \to +|-$

}

Alfabetul limbajului este format din toate simbolurile care pot apărea în cadrul propoziţiilor limbajului, adică din toate cifrele de la 0 la 9, semnele plus şi minus şi semnul de fracţie.

Punctul de plecare pentru enumerarea acestor elemente a fost sintaxa propoziţiilor limbajului de definit. O propoziţie a limbajului (adică un număr raţional) este de forma n/m, unde n şi m sunt numere întregi, iar m este diferit de zero. De aici rezultă regula de producţie pentru simbolul de start ($P \to N/M$), precum şi regulile de producţie pentru celelalte neterminale.

Următorul şir de derivări, care reprezintă generarea propoziţiei *-51/-301*, este format din derivări dreapta.

$P \Rightarrow N/M \Rightarrow N/ZDL_c \Rightarrow N/ZDCL_c \Rightarrow N/ZDCC \Rightarrow N/ZDC1 \Rightarrow N/ZD01 \Rightarrow N/Z301 \Rightarrow N/-301 \Rightarrow ZDL_c/-301 \Rightarrow ZDC/-301 \Rightarrow ZD1/-301 \Rightarrow Z51/-301 \Rightarrow -51/-301$

Care este şirul derivărilor stânga, pentru aceeaşi propoziţie?

Arborele sintactic ascendent corespunzător propoziţiei este:

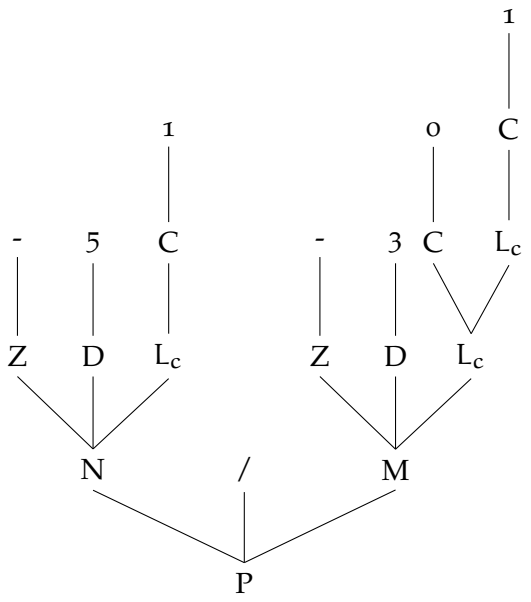


Figura 9: Arborele sintactic ascendent pentru propoziţia *-51/-301*

Care este arborele sintactic descendent corespunzător aceleiaşi propoziţii?

Care este tipul acestei gramatici? Construiţi o gramatică regulată care să definească acelaşi limbaj (limbajul numerelor raţionale este un limbaj regulat).

Exerciţiul 4:

Fie gramatica $G = (V_N, \Sigma, S, P)$. Enumeraţi regulile de producţie, mulţimea terminalelor şi a neterminalelor şi determinaţi simbolul de start, astfel încât gramatica să definească limbajul numerelor reale.

Fie gramatica $G = (V_N, \Sigma, S, P)$, astfel încât:

★ V_N = {R,N,M,L_c,C,D,Z},

★ Σ = {+, -, ., 0, 1, 2, 3,4,5,6,7,8,9},

★ S este neterminalul R,

★ P = {
  - R → N.M,
  - N → ZDL_c|ZD|Z0,
  - M → L_c|C,
  - Lc → CL_c|C,
  - C → 0|1|2|3|4|5|6|7|8|9
  - D → 1|2|3|4|5|6|7|8|9
  - Z → +|−

}

Alfabetul limbajului este format din toate simbolurile care pot apărea în cadrul propoziţiilor limbajului, adică din toate cifrele de la 0 la 9, semnele plus şi minus şi punctul.

Sintaxa propoziţiilor limbajului (sintaxa numerelor reale) este cea care ne ajută să definim elementele gramaticii. Un număr real este format din partea întreagă şi partea zecimală, separate prin simbolul punct; partea întreagă este un număr întreg, iar partea zecimală este o listă oarecare de cifre. De aici rezultă expresia regulii de producţie pentru simbolul de start, precum şi pentru regulile de producţie ale celorlalte neterminale.

Următorul şir de derivări, care reprezintă generarea propoziţiei *-123.089*, este format din derivări stânga.

$R \Rightarrow N.M \Rightarrow ZDL_c.M \Rightarrow -1CL_c.M \Rightarrow -12C.M \Rightarrow -123.M \Rightarrow -123.L_c \Rightarrow -123.CL_c \Rightarrow -123.0CL_c \Rightarrow -123.09C \Rightarrow -123.098$

Care este şirul derivărilor dreapta, pentru aceeaşi propoziţie?

Arborele sintactic descendent corespunzător propoziţiei este:

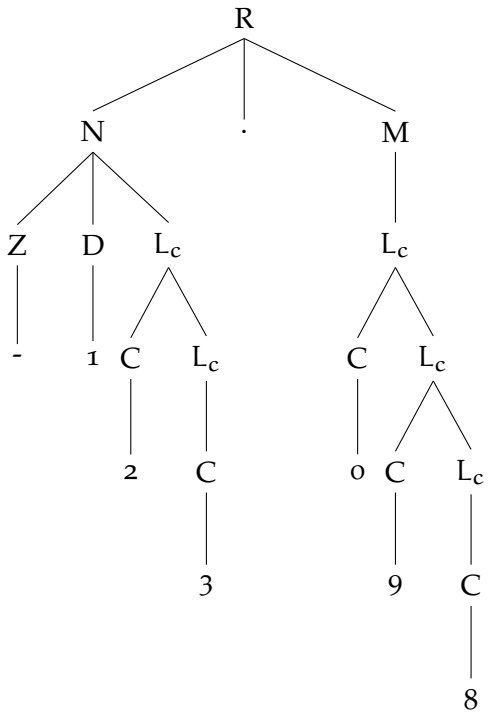


Figura 10: Arborele sintactic descendent pentru propoziţia *-123.089*

Care este arborele sintactic ascendent corespunzător aceleiaşi propoziţii?

Care este tipul acestei gramatici? Construiţi o gramatică regulată care să definească acelaşi limbaj (limbajul numerelor reale este un limbaj regulat).

Exerciţiul 5:

Fie gramatica $G = (V_N, \Sigma, S, P)$. Enumeraţi regulile de producţie, mulţimea terminalelor şi a neterminalelor şi determinaţi simbolul de start, astfel încât gramatica să definească limbajul numerelor reale scrise folosind separatorul de mii (virgula).

Fie gramatica $G = (V_N, \Sigma, S, P)$, astfel încât:

- V_N = {R,N,M,L_c,C,D,Z},
- Σ = {+, -, . , , , 0, 1, 2, 3, 4, 5, 6, 7, 8, 9},
- S este neterminalul R,
- P = {
  - $R \rightarrow ZN.M$,

- $N \rightarrow D,G \mid DC,G \mid DCC,G \mid D \mid DC \mid DCC \mid 0$,
- $G \rightarrow CCC \mid G.CCC$,
- $M \rightarrow L_c|C$,
- $L_c \rightarrow CL_c|C$,
- $C \rightarrow 0|1|2|3|4|5|6|7|8|9$
- $D \rightarrow 1|2|3|4|5|6|7|8|9$
- $Z \rightarrow +|-$

}

Următorul şir de derivări, care reprezintă generarea propoziţiei *-1,003.8*, este format din derivări stânga.

$R \Rightarrow ZN.M \Rightarrow -D,G.M \Rightarrow -1,CCC.M \Rightarrow -1,003.M \Rightarrow -1,003.C \Rightarrow -1,003.8$

Următorul şir de derivări, care reprezintă generarea propoziţiei *-1,000,000.35*, este format din derivări stânga.

$R \Rightarrow ZN.M \Rightarrow -N.M \Rightarrow -D,G.M \Rightarrow -D,G,CCC.M \Rightarrow -D,CCC,CCC.M \Rightarrow -1,000,000.L_c \Rightarrow -1,000,000.CL_c \Rightarrow -1,000,000.3C \Rightarrow -1,000,000.35$

Care sunt şirurile derivărilor dreapta, pentru cele două propoziţii?

Arborele sintactic ascendent corespunzător primei propoziţiei este:

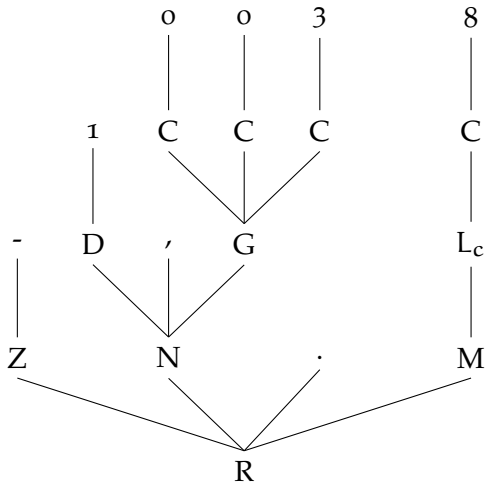


Figura 11: Arborele sintactic ascendent pentru propoziţia *-1.003,8*

Care este arborele sintactic descendent corespunzător aceleiaşi propoziţii?

Exerciţiul 6:

Pornind de la următorul exemplu de program, să se construiască o gramatică care să definească un limbaj de programare care să permită scrierea de programe cu structura şi instrucţiunile regăsite în exemplu.

int $a = 3$;
float b;
program test
begin
float c;
$c = a * b + 6.0$;
print c;
end

Se observă că limbajul permite stabilirea numelui programului, delimitarea corpului programului prin cuvintele "begin" şi "end", declararea de variabile globale (înainte de linia "program test") sau locale, cu sau fără iniţializarea valorii acestora, utilizarea expresiilor aritmetice, instrucţiuni de atribuire, afişarea valorii unei variabile.

Definirea limbajului de programare pentru care programul de mai sus este un exemplu, se va face în două etape: **definirea lexicului (vocabularului)** şi **definirea sintaxei**. Pentru ambele etape se vor folosi gramaticile formale.

Definirea lexicului constă în definirea atomilor lexicali care pot apărea în propoziţiile limbajului. Atomii lexicali reprezintă cele mai mici unităţi componente ale unei propoziţii care au sens prin ei înşişi. Ei constă din succesiuni de simboluri formate de baza alfabetului limbajului (literele mici şi mari, cifrele, alte simboluri cu caracter special). În mod asemănător vorbim despre lexicul sau vocabularul unei limbi care reprezintă toate cuvintele care fac parte din limba respectivă. Aceste cuvinte sunt succesiuni de simboluri din mulţimea alfabetului limbii şi constituie de fapt atomii lexicali ai limbii respective, fiind cele mai mici unităţi de vorbire care au sens. Ea se va face prin definirea a câte unei gramatici pentru fiecare dintre categoriile lexicale care pot apărea în cadrul limbajului:

1. cuvinte cheie: int, float, program, begin, end, print.

2. identificatori (numele programului şi numele variabilelor). Această categorie conţine o infinitate de elemente şi anume toate succesiunile de simboluri formate din litere şi cifre care încep cu o literă. De aceea pentru a reprezenta un identificator se va folosi simbolul "ID", care însă este un terminal, iar nu un neterminal, deoarece ţine loc oricărui element din mulţimea infinită a numelor posibile. Practic categoria identificatorilor este ea însăşi un limbaj în sine, ale cărui propoziţii se notează cu "ID".

   Gramatica care defineşte identificatorii este $G_1 = (V_N, \Sigma, S, P)$, astfel încât:

   - $V_N = \{ID, Li, L, C\}$,
   - $\Sigma = \{a, b, \ldots, z, A, B, \ldots, Z, 0, 1, \ldots, 9\}$,
   - S este neterminalul ID,
   - $P = \{$
     - $ID \to L \mid LLi$
     - $Li \to LLi \mid CLi \mid L \mid C$
     - $L \to a \mid b \mid \ldots \mid z \mid A \mid B \mid \ldots \mid Z$
     - $C \to 0 \mid 1 \mid \ldots \mid 9$

     $\}$

3. operatori: =, +, -, *, /.

4. semne de punctuaţie: ;, spaţiul (Linia nouă nu va fi luată în considerare, deoarece faptul că programul este scris pe mai multe linii sau pe o singură linie nu are importanţă. Scrierea pe mai multe linii are doar avantajul de a oferi o lizibilitate mărită.)

5. constantele

   a) constante de tip numere întregi. Această categorie conţine un număr foarte mare de elemente şi anume toate numere întregi care pot fi reprezentate pe un număr de octeţi egal cu cel folosit pentru a stoca astfel de valori. De aceea pentru a reprezenta o constantă întreagă se va folosi simbolul "CTI", care însă este un terminal, iar nu

un neterminal, deoarece ţine loc oricărui element din mulţimea numerelor întregi reprezentabile. Practic şi categoria constantelor întregi este ea însăşi un limbaj în sine, ale cărui propoziţii se notează cu "CTI".

Gramatica care defineşte constantele de tip numere întregi este $G_2 = (V_N, \Sigma, S, P)$, astfel încât:

- ★ V_N = {CTI,L_c,C,D,Z},
- ★ Σ = {+, -, 0, 1, 2, 3,4,5,6,7,8,9},
- ★ S este neterminalul CTI,
- ★ P = {
  - – $CTI \rightarrow ZDL_c|ZC$,
  - – $L_c \rightarrow CL_c|C$,
  - – $C \rightarrow 0|1|2|3|4|5|6|7|8|9$
  - – $D \rightarrow 1|2|3|4|5|6|7|8|9$
  - – $Z \rightarrow +|-$

  }

b) constante de tip numere reale. Această categorie conţine un număr foarte mare de elemente şi anume toate numere reale care pot fi reprezentate pe un număr de octeţi egal cu cel folosit pentru a stoca astfel de valori. De aceea pentru a reprezenta o constantă reală se va folosi simbolul "CTR", care însă este un terminal, iar nu un neterminal, deoarece ţine loc oricărui element din mulţimea numerelor reale reprezentabile. Practic şi categoria constantelor reale este ea însăşi un limbaj în sine, ale cărui propoziţii se notează cu "CTR".

Gramatica care defineşte constantele de tip numere reale este $G_3 = (V_N, \Sigma, S, P)$, astfel încât:

- ★ V_N = {CTR,N,M,L_c,C,D,Z},
- ★ Σ = {+, -, ., 0, 1, 2, 3,4,5,6,7,8,9},
- ★ S este neterminalul CTR,
- ★ P = {
  - – $R \rightarrow N.M$,
  - – $N \rightarrow ZDL_c|ZD|Z0$,
  - – $M \rightarrow L_c|C$,
  - – $Lc \rightarrow CL_c|C$,
  - – $C \rightarrow 0|1|2|3|4|5|6|7|8|9$
  - – $D \rightarrow 1|2|3|4|5|6|7|8|9$
  - – $Z \rightarrow +|-$

  }

Definirea sintaxei limbajului constă în definirea modului în care elementele lexicului (adică atomii lexicali) se grupează împreună pentru a forma propoziţiile limbajului.

Gramatica care defineşte sintaxa limbajului este $G = (V_N, \Sigma, S, P)$, astfel încât:

- ★ $V_N = \{Lg, P, D, Li, E, I, C, S\}$,

★ Σ = {int, float, program, begin, end, =, ; , print, ID, , CTI, CTR, +, -, *, /},

★ S este neterminalul S,

★ P = {
  - S → LgP | P
  - Lg → LgD; | D;
  - D → int ID | int ID=CTI | float ID | float ID=CTR
  - P → program ID begin Li end
  - Li → LiI; | I;
  - I → D | ID=E | print ID
  - E → E + E | E - E | E * E | E / E | ID | CTI | CTR

}

Se observă că în definirea alfabetului gramaticii care defineşte sintaxa limbajului au fost enumerate în mod direct elementele lexicale finite (cuvintele cheie, operatorii, semnele de punctuaţie), iar elementele lexicale infinite ca şi număr (identificatori şi constate) au fost reprezentate prin simbolul de start al gramaticii care le defineşte, conform explicaţiilor de mai sus.

Următorul şir de derivări, care reprezintă generarea propoziţiei dată ca şi exemplu, este format din derivări dreapta.

S →
LgP →
Lg **program ID begin Li end** →
Lg program ID begin **Li I ;** end →
Lg program ID begin Li **print ID** ; end →
Lg program ID begin **Li I ;** print ID ; end →
Lg program ID begin Li **ID = E** ; print ID ; end →
Lg program ID begin Li ID = **E + E** ; print ID ; end →
Lg program ID begin Li ID = E + **CTR** ; print ID ; end →
Lg program ID begin Li ID = **E * E** + CTR ; print ID ; end →
Lg program ID begin Li ID = E * **ID** + CTR ; print ID ; end →
Lg program ID begin Li ID = **ID** * ID + CTR ; print ID ; end →
Lg program ID begin **I ;** ID = ID * ID + CTR ; print ID ; end →
Lg program ID begin **D** ; ID = ID * ID + CTR ; print ID ; end →
Lg program ID begin **float ID** ; ID = ID * ID + CTR ; print ID ; end →
Lg D ; program ID begin float ID ; ID = ID * ID + CTR ; print ID ; end →
Lg **float ID** ; program ID begin float ID ; ID = ID * ID + CTR ; print ID ; end →
D ; float ID ; program ID begin float ID ; ID = ID * ID + CTR ; print ID ; end →
int ID = CTI ; float ID ; program ID begin float ID ; ID = ID * ID + CTR ; print ID ; end

Care este şirul derivărilor stânga, pentru aceeaşi propoziţie?

Arborele sintactic descendent corespunzător propoziţiei este:

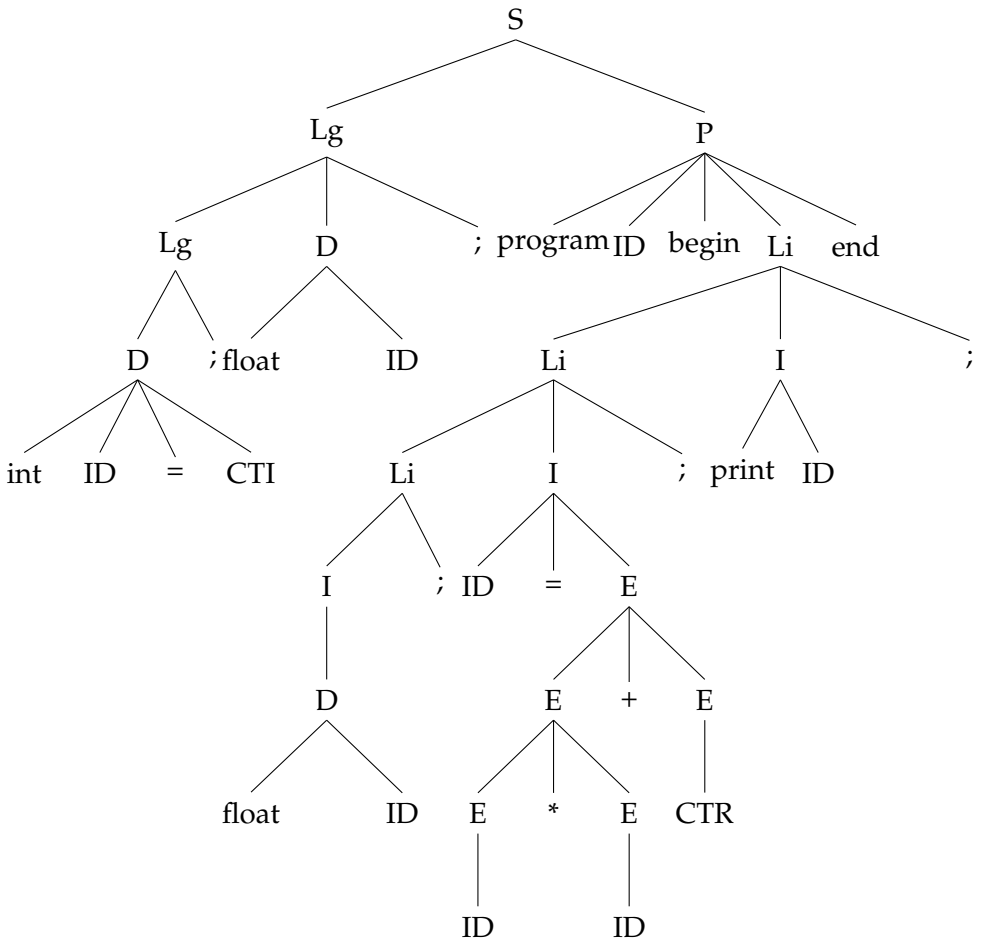


Figura 12: Arborele sintactic descendent pentru programul luat ca şi exemplu

Care este arborele sintactic ascendent corespunzător aceleiaşi propoziţii?

2.5 EXERCIŢII PROPUSE

3

LIMBAJE REGULATE

3.1 AUTOMATE FINITE

Automatele cu stări finite, sau pe scurt automatele finite, reprezintă un model matematic care permite descrierea proceselor de calcul. Un automat finit este gândit ca şi o maşină abstractă, care, la un moment dat, se poate afla într-o singură stare, dintr-o mulţime finită de stări date. Starea în care se află automatul la un moment dat se numeşte *stare curentă*. Starea în care se află automatul înainte de începerea procesului de calcul se numeşte *stare iniţială*. Automatul poate să treacă dintr-o stare în alta în funcţie de anumite intrări. Schimbarea stării curente a automatului (trecerea din starea curentă într-o altă stare, care devine noua stare curentă) se numeşte *tranziţie*. Dacă pentru o stare curentă şi pentru o intrare dată automatul poate să facă o singură tranziţie (există o singură stare în care poate să treacă, în baza intrării date), atunci el este un automat finit determinist. Dacă însă pentru o stare curentă şi o intrare dată un automat poate efectua mai multe tranziţii (există mai multe stări în care poate să treacă, în baza intrării date), atunci el este un automat finit nedeterminist.

În teoria limbajelor formale, automatele finite sunt utilizate pentru a defini sau pentru a recunoaşte limbaje regulate. Cu alte cuvinte, expresivitatea unui anumit finit este aceeaşi cu cea a unei gramatici regulate. Pentru un astfel de automat, intrarea pe baza căreia se realizează tranziţiile este dată de către următorul simbol neanalizat din şirul de la intrare. Mulţimea şirurilor de simboluri pe care le recunoaşte un automat finit dat, se numeşte limbajul definit de către automat. Aşa cum am văzut în capitolul precedent, automatele finite sunt utilizate pentru a verifica dacă o propoziţie dată aparţine sau nu unui anumit limbaj regulat, adică dacă este o propoziţie corectă din punctul de vedere al limbajului regulat avut în vedere.

3.1.1 *Automate finite deterministe*

În continuare vom da definiţia formală a unui automat finit determinist, din punct de vedere al teoriei limbajelor formale.

Definitie 22 *Un **automat finit determinist** este un cvintuplu* $AFD = (\Sigma, Q, F, q_0, f)$*, în care:*

- ★ Σ *este o mulţime finită de simboluri şi se numeşte alfabetul limbajului de intrare al automatului;*
- ★ Q *este mulţimea finită de stări ale automatului;*
- ★ F *este mulţimea stărilor finale ale automatului,* $F \subseteq Q$*;*
- ★ q_0 *este starea iniţială a automatului,* $q_0 \in Q$*;*
- ★ f *este funcţia de tranziţie,* $f : Q \times \Sigma \to Q$*.*

Un automat finit poate fi reprezentat în două moduri: prin graful de tranziţie, sau prin tabela de tranziţie. Pentru a descrie cele două modalităţi de reprezentare, vom porni de la exemplul următor.

Exemplu:

Fie automatul finit determinist $AFD = (\Sigma, Q, F, q_0, f)$, unde:

- ★ $\Sigma = \{0, 1\}$,
- ★ $Q = \{q_0, q_1, q_2\}$,
- ★ $F = \{q_0, q_2\}$
- ★ starea iniţială este q_0,
- ★ iar funcţia de tranziţie f se deduce din graful de tranziţie, sau respectiv din tabelul de tranziţie.

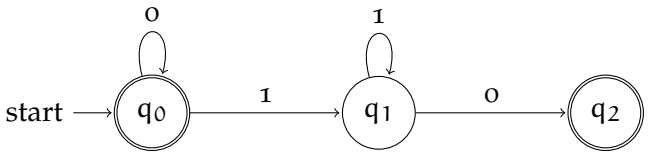


Figura 13: Graful de tranziţie

f	0	1
*q_0	q_0	q_1
q_1	q_2	q_1
*q_2	-	-

Figura 14: Tabela de tranziţie

Automatul finit determinist din exemplul de mai sus acceptă limbajul următor:
$L(AFD) = \{0^n1^m0 | n \geqslant 0,\ m \geqslant 1\}$, unde am notat cu a^t concatenarea simbolului a cu el însuşi de un număr de t ori.

Iată care sunt convenţiile pentru cele două maniere de reprezentare. Începem cu graful de tranziţie. Stările se reprezintă prin cercuri, care au înscrise în ele numele. Starea iniţială este marcată printr-o săgeată. Stările finale sunt marcate prin dublarea cercurilor. Tranziţiile sunt reprezentate prin săgeţi între stări, deasupra cărora se scrie simbolul/simbolurile de la intrare în baza cărora se face schimbarea stării. În cazul tabelelor de tranziţie avem o singură convenţie, şi anume stările finale se marchează printr-o steluţă.

Un automat citeşte un şir finit de simboluri $a_1a_2 \dots a_n$, $a_i \in \Sigma, \forall\ 1 \leqslant i \leqslant n$. În funcţie de fiecare simbolul citit şi de starea curentă în care se află automatul la citirea simbolului respectiv, acesta execută o tranziţie, în conformitate cu definiţia funcţiei de tranziţie. Automatul se opreşte după ce a citit toate simbolurile din propoziţia de intrare, sau atunci când pentru simbolul următor de la intrare şi pentru starea curentă în care se află, nu se poate executa nicio tranziţie.

Definitie 23 *Fie* $AFD = (\Sigma, Q, F, q_0, f)$ *un automat finit. O pereche* $(x,\ q), x \in \Sigma^*,\ q \in Q$ *se numeşte* ***configuraţie*** *a automatului finit.*

Observaţie: q este starea curentă a automatului, iar x este restul şirului de la intrarea automatului, care nu a fost încă analizat.

Definitie 24 *Fie* $AFD = (\Sigma, Q, F, q_0, f)$ *un automat finit.* ***Relaţia de mişcare*** *este o relaţie binară notată cu* $\vdash$ $(\vdash \subseteq (\Sigma^* \times Q) \times (\Sigma^* \times Q))$ *şi definită astfel:*
$(ax, q) \vdash (x, q') \Leftrightarrow f(q, a) = q'$, *unde* $a \in \Sigma$, $x \in \Sigma^*$, $q, q' \in Q$.

Definitie 25 *Fie* $AFD = (\Sigma, Q, F, q_0, f)$ *un automat finit.* ***Relaţia de mişcare în k paşi*** *este o relaţie binară notată cu* $\vdash^k$ ($\vdash^k \subseteq (\Sigma^* \times Q) \times (\Sigma^* \times Q)$) *şi definită astfel:*
$(a_1 a_2 \dots a_k x, q) \vdash^k (x, q') \Leftrightarrow \exists\, k+1$ *configuraţii astfel încât*
$(a_i a_{i+1} \dots a_k x, q_i) \vdash (a_{i+1} \dots a_k x, q_{i+1})$, $\forall\, 1 \leqslant i \leqslant k$, $q = q_1, q' = q_{k+1}$,
unde $a_1, a_2, \dots, a_k \in \Sigma$, $x \in \Sigma^*$, $q, q', q_1, q_2, \dots, q_{k+1} \in Q$.

Definitie 26 *Fie* $AFD = (\Sigma, Q, F, q_0, f)$ *un automat finit.* ***Relaţia generală de mişcare*** *este o relaţie binară notată cu* $\vdash^*$ ($\vdash^* \subseteq (\Sigma^* \times Q) \times (\Sigma^* \times Q)$) *şi definită astfel:*
$(x_1, q) \vdash^* (x_2, q') \Leftrightarrow (x_1 = x_2$ *şi* $q = q')$ *sau* $((x_1, q) \vdash^+ (x_2, q'))$, *unde* $x_1, x_2 \in \Sigma^*, q, q' \in Q$.

Definitie 27 *Fie* $AFD = (\Sigma, Q, F, q_0, f)$ *un automat finit. Se spune că* $x \in \Sigma^*$ *este o* ***propoziţie acceptată*** *de către automat, dacă există următoarea relaţie de mişcare:* $(x, q_0) \vdash^* (\epsilon, q')$, *unde* $q' \in F$.

O propoziţie acceptată de către un automat finit AFD este orice şir de simboluri din Σ, care, citit simbol cu simbol de la stânga spre dreapta, duce automatul din starea iniţială într-o stare finală.

Definitie 28 *Fie* $AFD = (\Sigma, Q, F, q_0, f)$ *un automat finit.* ***Limbajul acceptat*** *de către automatul AFD se notează cu* $L(AFD)$ *şi est definit în felul următor:*
$L(AFD) = \{x \in \Sigma^* | f^*(q_0, x) \in F\}$,
unde f *este extinsă la* f^* *astfel:*

- $f^*(q, \epsilon) = q$,
- $f^*(q, ax) = f^*(f(q, a), x)$.

Cu alte cuvinte, limbajul acceptat de către un automat finit AFD este mulţimea formată din toate şirurile de simboluri peste Σ care sunt acceptate de către automat.

Din punct de vedere filosofic, *determinismul* este credinţa potrivit căreia toate evenimentele care s-au petrecut au fost cauzate de către ceva care s-a întâmplat înaintea lor, şi deci oamenii nu au cu adevărat puterea de a alege, şi nici de a controla ceea ce se întâmplă. Atributul *determinist* aplicat unui automat finit, are exact aceeaşi semnificaţie. Dacă considerăm că un eveniment înseamnă realizarea unei tranziţii, şi ştiind că acest lucru este cauzat de către starea în care se află automatul şi de către următorul simbol din şirul de intrare, vedem că în cazul unui automat finit determinist el nu poate să aleagă/să controleze ce tranziţie să execute. Tranziţia este cauzată exclusiv de starea curentă a automatului şi de simbolul citit de la intrare.

Exemplu:
Fie automatul finit $M = (\Sigma, Q, F, q_0, f)$, unde:

- $\Sigma = \{a, b\}$
- $Q = \{q_0, q_1, q_2, q_3\}$
- $F = \{q_3\}$
- iar funcţia de tranziţie f este definită prin următorul graf de tranziţie:

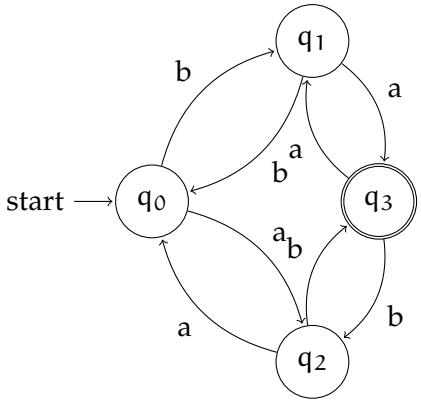


Figura 15: Exemplu AFD

3.1.2 *Automate finite nedeterministe*

Definitie 29 *Un **automat finit nedeterminist** este un cvintet* $AFN = (\Sigma, Q, F, q_0, f)$*, unde* Σ, Q, F, q_0 *sunt definite în acelaşi mod ca şi pentru un automat finit determinist, iar funcţia de tranziţie* f *este definită astfel:*

$f : Q \times \Sigma \to 2^Q - \emptyset$.

În cazul unui automat finit nedeterminist, pentru o aceeaşi stare şi pentru un acelaşi simbol de la intrare, pot exista mai multe tranziţii posibile, ceea ce înseamnă că automatul poate trece în mai multe stări. Evident, automatul nu are niciun criteriu suplimentar în baza căruia să facă alegerea.

Exemplu:

Fie automatul finit $M = (\Sigma, Q, F, q_0, f)$, unde:

- ★ $\Sigma = \{0, 1\}$
- ★ $Q = \{q_0, q_1, q_2, q_3\}$
- ★ $F = \{q_3\}$
- ★ iar funcţia de tranziţie f este definită prin următorul graf de tranziţie:

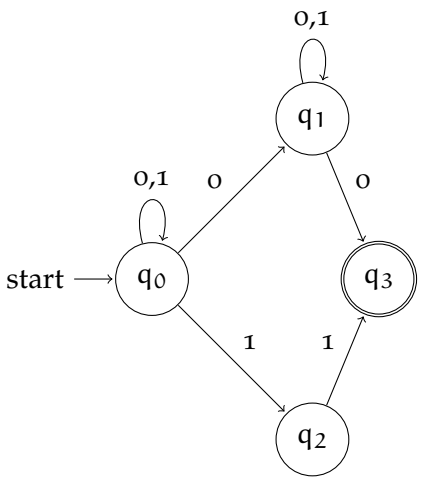


Figura 16: Exemplu AFN

Se observă că dacă automatul finit se află în starea q_0, iar la intrare urmează simbolul 0, atunci el poate fie să rămână în starea q_0, fie să treacă în starea q_1. Analog şi pentru starea q_1. Prin urmare, acest automat este un automat finit nedeterminist.

Automatele finite deterministe şi automatele finite nedeterministe sunt echivalente (au aceeaşi expresivitate). Automatele finite nedeterministe au fost introduse pentru ca problema construirii unui automat finit care să definească un limbaj dat să fie mai uşor de abordat, prin eliminarea restricţiei privind faptul că tranziţiile care pleacă din aceeaşi stare trebuie să se facă în baza unor simboluri distincte.

Din punct de vedere al implementării analizei realizate de către un automat finit nedeterminist, problema se poate reformula în felul următor. Pentru o stare curentă dată şi pentru următorul simbol citit de la intrare, schimbarea stării curente a automatului poate da mai multe rezultate. Adică algoritmul care descrie funcţionarea unui automat finit nedeterminist este şi el nedeterminist. Un astfel de algoritm se poate implementa numai prin backtracking sau explorând în paralel toate soluţiile posibile. O astfel de implementare este însă costisitoare şi ineficientă. De aceea implementarea funcţionării unui automat finit nedeterminist nu se face ca atare. Se procedează în felul următor: se deduce automatul finit determinist echivalent cu automatul finit nedeterminist dat, şi apoi se face implementarea pentru cel determinist.

Teorema 1 *Pentru orice automat finit nedeterminist* $M = (\Sigma, Q, F, q_0, f)$*, există un automat finit determinist* M' *echivalent, adică* $L(M') = L(M)$.

Fie $M = (\Sigma, Q, F, q_0, f)$ un automat finit nedeterminist. Atunci automatul finit determinist echivalent M' este definit astfel $M' = (\Sigma, Q', F', q_0', f')$, unde:

	AFN	AFD
Alfabetul	Σ	Σ
M. stărilor	$Q = \{q_0, q_1, q_2, \dots q_n\}$	$Q' = \{\emptyset, \{q_0\}, \{q_1\}, \{q_2\}, \dots, \{q_n\}, \{q_0, q_1\}, \{q_0, q_2\}, \dots, \{q_0, q_n\}, \dots, \{q_{n-1}, q_n\}, \dots, \{q_0, q_1, \dots q_n\}\}$
Starea iniţială	q_0	$\{q_0\}$
M. stărilor finale	$F \subset Q$	$F' = \{s \in Q' \mid$ *s conţine cel puţin o stare finală a AFN* $\}$
Tranziţii	f	$f'(\{q_{i_1}, q_{i_2}, \dots, q_{i_k}\}, a) = f(q_{i_1}, a) \cup f(q_{i_2}, a) \cup \dots \cup f(q_{i_k}, a)$

După generarea elementelor automatului finit determinist pornind de la cele ale automatului finit nedeterminist, se elimina stările şi tranziţiile inutile. Stările inutile sunt stările la care automatul nu poate ajunge pornind din starea iniţială şi făcând una sau mai multe tranziţii. Tranziţiile inutile sunt tranziţiile care pleacă dintr-o stare inutilă.

Cele două automate au acelaşi alfabet Σ. Fiind echivalente (adică definind acelaşi limbaj), este normal ca alfabetul să fie şi el acelaşi.

Mulţimea stărilor automatului finit determinist echivalent Q' este formată din combinările stărilor automatului finit nedeterminist luate, pe rând, câte o, câte una, câte două, câte trei, ş.a.m.d., până la cardinalul mulţimii stărilor automatului finit nedeterminist.

Mulţimea stărilor finale ale automatului finit determinist echivalent F' este formată din toate elementele mulţimii stărilor automatului finit determinist echivalent, în denumirea cărora apare cel puţin o stare finală a automatului nedeterminist.

$F' = \{\{q_0, q_1, \ldots, q_k\} \in Q' | \exists\, 0 \leqslant i \leqslant k \text{ a.i. } q_i \in F\}$.

Valoarea funcţiei de tranziţie a automatului finit determinist echivalent pentru o stare $q_{i_1}, q_{i_2}, \ldots, q_{i_k}$ şi un simbol $a \in \Sigma$ este egală cu reuniunea valorilor funcţiei de tranziţie a automatului nedeterminist pentru fiecare dintre componentele stării şi pentru simbolul a.

Exemplu:

În continuare vom exemplifica aplicarea teoremei deducerii automatului finit determinist echivalent, pentru automatul finit nedeterminist dat prin graful de tranziţie din figura următoare:

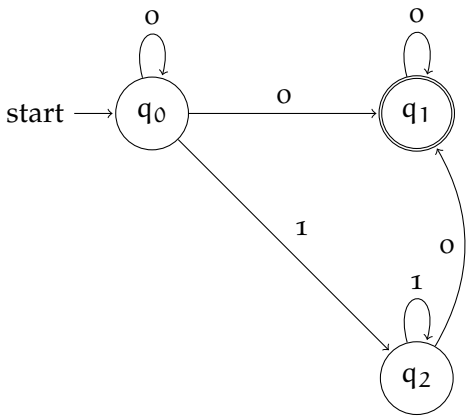


Figura 17: Un automat finit nedeterminist

Aplicând teorema deducerii automatului finit nedeterminist, se poate afirma că pentru automatul finit nedeterminist $M = (\Sigma, Q, F, q_0, f)$ dat prin graful de tranziţie anterior, există un automat finit determinist $M' = (\Sigma, Q', F', q_0', f')$ echivalent, adică care recunoaşte acelaşi limbaj. Elementele acestui automat finit determinist se deduc pornind de la elementele corespunzătoare ale automatului finit nedeterminist de la care se pleacă, aşa cum arată teorema. Mai jos este dat tabelul funcţiei de tranziţie f' a automatului finit determinist echivalent.

f'	0	1
$\varnothing$	$\varnothing$	$\varnothing$
$\{q_0\}$	$\{q_0 q_1\}$	$\{q_2\}$
$*\{q_1\}$	$\{q_1\}$	$\varnothing$
$\{q_2\}$	$\{q_1\}$	$\{q_2\}$
$*\{q_0 q_1\}$	$\{q_0 q_1\}$	$\{q_2\}$
$\{q_0 q_2\}$	$\{q_0 q_1\}$	$\{q_2\}$
$*\{q_1 q_2\}$	$\{q_1\}$	$\{q_2\}$
$*\{q_0 q_1 q_2\}$	$\{q_0 q_1\}$	$\{q_2\}$

Figura 18: Funcţia de tranziţie a automatului finit determinist echivalent

După deducerea elementelor automatului finit determinist echivalent, urmează etapa eliminării stărilor şi tranziţiilor inutile. Pentru a fi uşor de observat care sunt stările şi tranziţiile inutile, se va desena graful de tranziţie al automatului finit determinist echivalent, pe baza tabelului funcţiei de tranziţie.

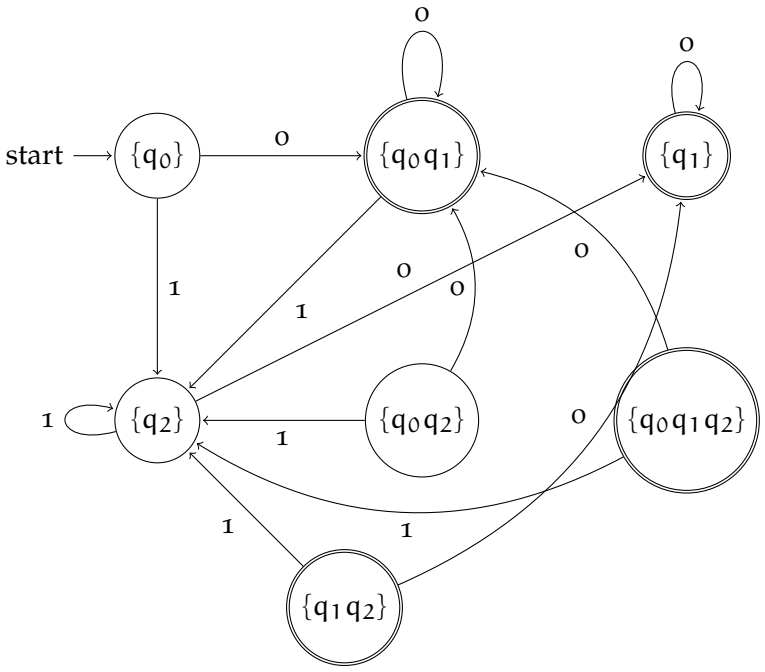


Figura 19: Graful de tranziţie (complet) al automatului finit determinist echivalent

Se observă că stările $\{q_0q_2\}$,$\{q_1q_2\}$ şi $\{q_0q_1q_2\}$ au numai tranziţii care pleacă din ele, însă nicio tranziţie care ajunge în ele. Prin urmare, nu există nicio succesiune de tranziţii care să ajungă din starea iniţială a automatului finit în aceste stări. Ca urmare, ele sunt stări inutile, iar tranziţiile care pleacă din ele sunt tranziţii inutile; ele pot fi eliminate din automat fără a afecta limbajul recunoscut de către acesta.

După eliminarea stărilor şi tranziţiilor inutile, se obţine automatul finit determinist echivalent final.

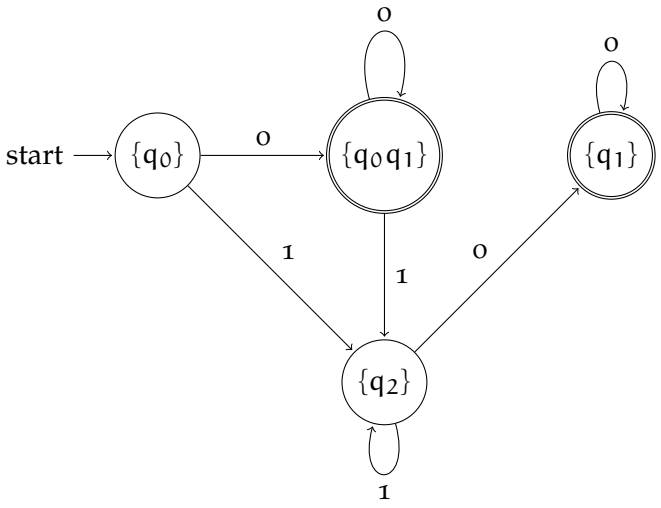


Figura 20: Graful de tranziţie (redus) al automatului finit determinist echivalent

În practică, se poate sări peste etapa desenării grafului de tranziţie complet şi a eliminării ulterioare a stărilor şi tranziţiilor inutile, construind direct numai partea utilă a acestuia. În acest sens se va proceda în felul următor. Se desenează starea iniţială şi tranziţiile care pleacă din aceasta (ceea ce va implica desenarea stărilor în care se poate ajunge plecând din starea iniţială). Apoi, pentru fiecare dintre noile stări desenate, se va proceda în acelaşi mod: se vor

desena toate tranziţiile care pleacă din fiecare dintre noile stări (ceea ce va implica probabil desenarea unor noi stări). Procesul se repetă pentru fiecare nouă stare desenată la pasul anterior, până în momentul în care nu mai este desenată nicio nouă stare. Graful obţinut este graful de tranziţie redus, iar stările şi tranziţiile care apar în tabelul funcţiei de tranziţie, dar care nu au fost desenate, sunt stări şi tranziţii inutile.

În continuare, vom exemplifica acest proces pentru graful anterior. Se pleacă de la tabelul funcţiei de tranziţie f' şi se începe prin desenarea stării iniţiale.

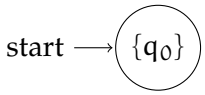


Figura 21: Construirea directă a grafului de tranziţie minimal al automatului finit determinist echivalent

Se desenează apoi cele două tranziţii care pleacă din starea iniţială, ceea ce va implica desenarea stărilor $\{q_2\}$ şi $\{q_0q_1\}$.

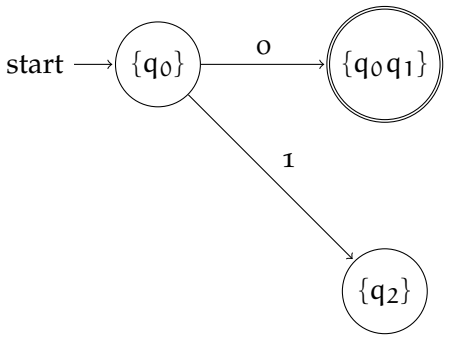


Figura 22: Construirea directă a grafului de tranziţie minimal al automatului finit determinist echivalent

Deoarece stările $\{q_2\}$ şi $\{q_0q_1\}$ sunt stări noi, care au apărut în urma ultimei operaţii de desenare, pentru fiecare dintre ele se va proceda la fel ca şi pentru starea iniţială. Prin urmare, se vor desena toate tranziţiile care pleacă din starea $\{q_0q_1\}$. În urma acestui pas, nu a fost desenată nicio stare nouă.

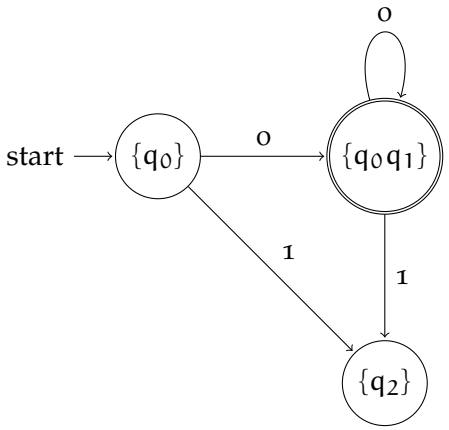


Figura 23: Construirea directă a grafului de tranziţie minimal al automatului finit determinist echivalent

Apoi, se vor desena toate tranziţiile care pleacă din starea $\{q_2\}$. În urma acestui pas, a fost desenată noua stare $\{q_1\}$.

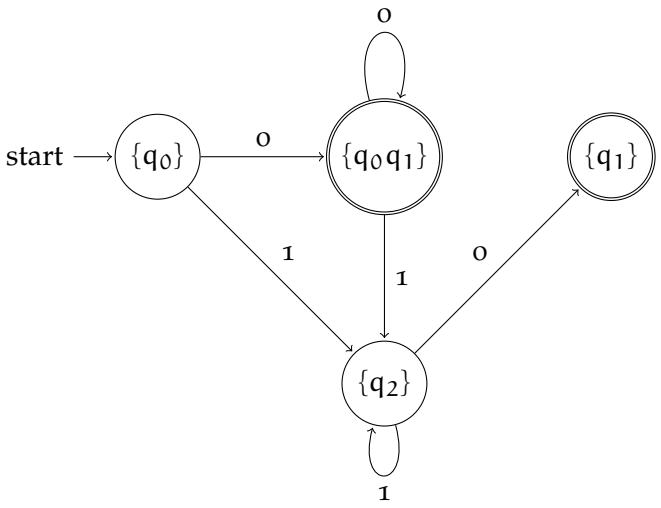


Figura 24: Construirea directă a grafului de tranziţie minimal al automatului finit determinist echivalent

În urma paşilor anterior, a apărut o singură stare nouă, şi anume $\{q_1\}$. Prin urmare, se continuă cu desenarea tuturor tranziţiilor care pleacă din această stare.

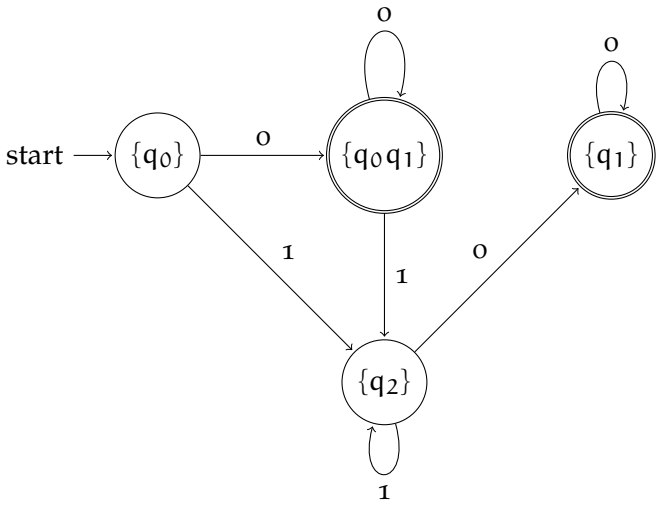


Figura 25: Construirea directă a grafului de tranziţie minimal al automatului finit determinist echivalent

În urma acestui pas nu a fost desenată nicio stare nouă. Aceasta înseamnă că procesul s-a încheiat, iar graful de tranziţie obţinut este cel final. Stările $\{q_0q_2\}$,$\{q_1q_2\}$ şi $\{q_0q_1q_2\}$ apar în tabelul funcţiei de tranziţie, dar nu şi în cadrul acestui graf. Prin urmare aceste stări, precum şi tranziţiile care pleacă din ele, sunt inutile.

3.1.3 *Tranziţii* ϵ. *Automate finite* ϵ

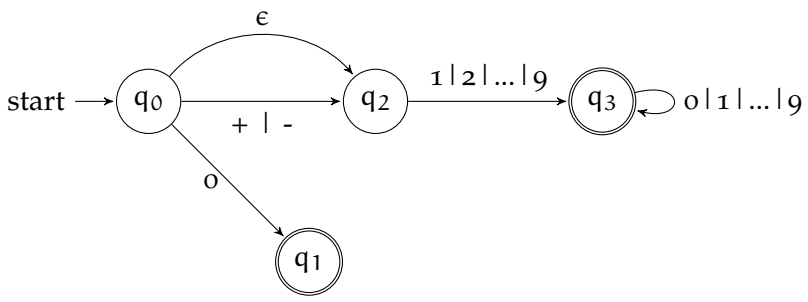
O tranziţie ϵ este o tranziţie dintr-o stare în alta, fără a "consuma" niciun simbol din şirul de intrare. Aceste tranziţii sunt practic tranziţii spontane, care se fac fără a privi la simbolul

care urmează în şirul de intrare. Un automat finit care are cel puţin o tranziţie ϵ se numeşte automat finit ϵ.

Definitie 30 *Un **automat finit** ϵ este un cvintet $AF - \epsilon = (\Sigma \cup \{\epsilon\}, Q, F, q_0, f)$, unde Σ, Q, F, q_0 sunt definite în acelaşi mod ca şi pentru un automat finit determinist, iar funcţia de tranziţie f este definită astfel:*

$f : Q \times \Sigma \cup \{\epsilon\} \to 2^Q - \emptyset$.

Exemplu:



Definitie 31 *Fie $AF - \epsilon = (\Sigma \cup \{\epsilon\}, Q, F, q_0, f)$ un automat finit ϵ. **Închiderea unei stări (closure)** $q \in Q$, notată cu $CL(q)$, este mulţimea tuturor stărilor la care se poate ajunge pornind din starea q şi făcând numai tranziţii ϵ:*

$CL(q) = \{q' \in Q \mid (x, q) \vdash^+ (x, q')\}$.

Pentru a specifica faptul că închiderea unei stări q se calculează pentru un anumit automat finit AF, vom folosi notaţia $CL(q)|_{AF}$.

Automatele finite nedeterministe şi automatele finite ϵ sunt echivalente (au aceeaşi expresivitate). Tranziţiilor ϵ au fost introduse pentru ca problema construirii unui automat finit care să definească un limbaj dat să fie mai uşor de abordat, prin eliminarea restricţiei privind faptul că orice tranziţie se face în baza unui simbol.

Ca şi în cazul automatelor finite nedeterministe, implementarea funcţionării unui automat finit ϵ nu se face ca atare. În schimb, se procedează în felul următor: se deduce automatul finit nedeterminist echivalent cu automatul finit ϵ dat. Apoi se deduce automat finit determinist echivalent automatului finit nedeterminist obţinut. Şi, în cele din urmă, se face implementarea pentru cel determinist. Deducerea automatului finit nedeterminist echivalent unui automat finit ϵ se face prin eliminarea tranziţiile ϵ şi înlocuirea lor cu tranziţii noi, obţinute prin combinarea fiecărei tranziţii ϵ cu fiecare dintre tranziţiile care pleacă din starea ţintă a tranziţiei ϵ respective.

Teorema 2 *Pentru orice automat finit ϵ $M = (\Sigma \cup \{\epsilon\}, Q, F, q_0, f)$, există un automat finit nedeterminist M' echivalent, adică $L(M') = L(M)$.*

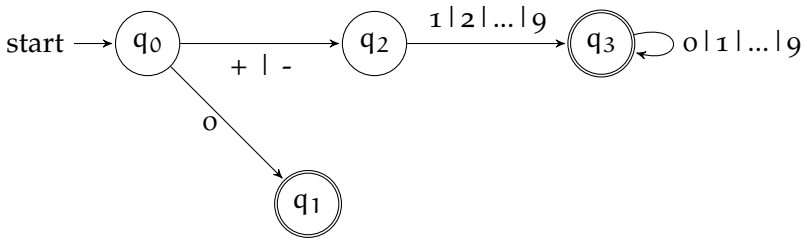
Fie $M = (\Sigma \cup \{\epsilon\}, Q, F, q_0, f)$ un automat finit ϵ. Atunci automatul finit nedeterminist echivalent M' este definit astfel $M' = (\Sigma, Q', F', q_0, f')$, unde:

	AFϵ	AFD
Alfabetul	$\Sigma \cup \{\epsilon\}$	Σ
M. stărilor	$Q = \{q_0, q_1, q_2, \dots q_n\}$	$Q' = \{q_0, q_1, q_2, \dots q_n\}$
Starea inițială	q_0	q_0
M. stărilor finale	$F \subset Q$	$F' = F \cup \{q' \in Q' \mid \exists\ s \in CL(q')|_{AF\epsilon}$ a.i. $s \in F\}$
Tranziții	f	$f'(q, a) = f(q, a)$ $f'(q, b) = f(f(\dots f(q, \epsilon) \dots, \epsilon), b)$ $\forall\ q \in Q'$ și $a, b \in \Sigma$

După generarea elementelor automatului finit nedeterminist pornind de la cele ale automatului finit ϵ, se elimina stările și tranzițiile inutile.

Vom exemplifica deducerea automatului finit nedeterminist echivalent pentru automatul finit ϵ din exemplul anterior:

★ Primul pas constă în rescrierea automatului finit fără tranzițiile ϵ.



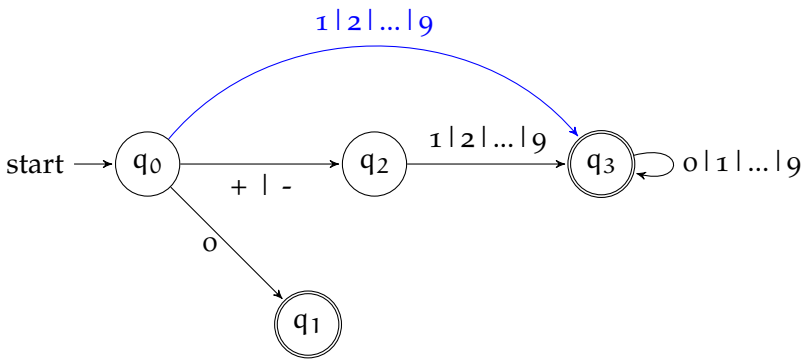
★ Al doilea pas constă în înlocuirea fiecărei tranziții ϵ conform regulii enunțate mai sus.

În acest caz, automatul finit ϵ are o singură tranziție ϵ, și anume cea din starea q_0 în starea q_2. Pentru a putea stabili tranziția sau tranzițiile cu care această tranziție ϵ va fi înlocuită, trebuie să se aibă în vedere toate tranzițiile care pleacă din starea în care ajunge tranziția ϵ, și anume starea q_2. Din starea q_2 pleacă o singură tranziție, cea către starea q_3, în baza simbolurilor $1 \mid 2 \mid \cdots \mid 9$.

$f(q_0, \epsilon) = q_2$ și $f(q_2, 1 \mid 2 \mid \cdots \mid 9) = q_3 \Leftrightarrow f(f(q_0, \epsilon), 1 \mid 2 \mid \cdots \mid 9) = q_3$

Deci tranziția ϵ din starea q_0 în starea q_2 va fi înlocuită cu o tranziție din starea q_0 în starea în care ajunge tranziția care pleacă din q_2, și anume în starea q_3, în baza acelorași simboluri ca și cele consumate de tranziția din starea q_2 în starea q_3 ($1 \mid 2 \mid \cdots \mid 9$).

$f'(q_0, 1 \mid 2 \mid \cdots \mid 9) = q_3$



În eliminarea tranziţiilor ϵ poate să apară situaţia în care din starea în care se trece prin tranziţia ϵ nu pleacă nicio tranziţie. În acest caz, tranziţia ϵ respectivă nu este înlocuită cu nicio altă tranziţie.

★ Al treilea pas constă în determinarea stărilor automatului finit nedeterminist echivalent care vor deveni stări finale. Toate stările automatului finit ϵ ale căror închideri conţin o stare finală, devin stări finale în automatul finit nedeterminist echivalent.

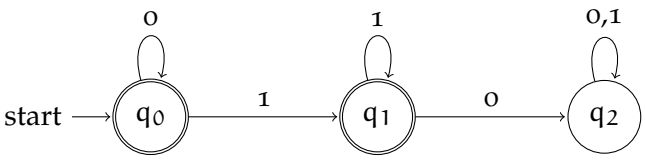
Se începe prin calcularea închiderii stărilor automatului finit ϵ.

$CL(q_0) = \{q_2\}$, iar $CL(q_1) = CL(q_2) = CL(q_3) = \emptyset$

Deoarece niciuna dintre închideri nu conţine o stare finală a automatului finit ϵ, în automatul finit nedeterminist echivalent stările nu se modifică.

3.1.4 *Automate finite versus gramatici regulate*

Atunci când s-a vorbit despre clasificarea gramaticilor, a fost definit tipul gramaticilor regulate. Automatele finite, indiferent de tip, permit definirea numai a unor limbaje regulate. Este foarte simplu de observat acest lucru, încercând să construim gramatica formală care defineşte acelaşi limbaj ca şi un automat finit dat. De exemplu, pentru automatul finit determinist următor:



mulţimea regulilor de producţie a gramaticii care defineşte acelaşi limbaj, are elementele:

★ $S \to 0S \mid 1P \mid \epsilon$

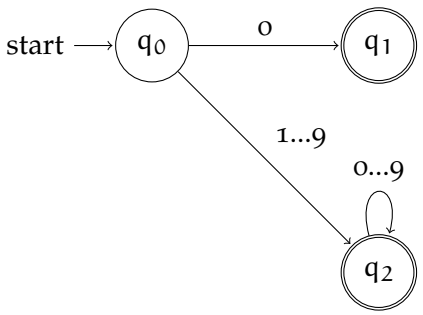
★ $P \to 1P \mid 0Q \mid \epsilon$

★ $Q \to 0Q \mid 1Q$

Prin urmare, automatele finite şi gramaticile regulate sunt echivalente.

3.1.5 *Exerciţii*

1. Să se construiască un automat finit care să definească limbajul numerelor naturale. Apoi, să se precizeze ce fel de automat finit a fost construit şi să se scrie succesiunea de relaţii de mişcare pentru analiza propoziţiilor 1310 şi 001.



Se observă că automatul finit construit este un automat finit determinist.

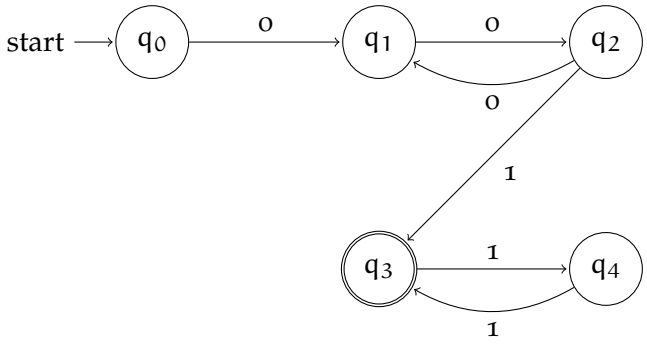
$(1310, q_0) \vdash (310, q_2) \vdash (10, q_2) \vdash (0, q_2) \vdash (\epsilon, q_2) \Leftrightarrow 1310 \in L(AF)$

Configuraţia (ϵ, q_2) este o configuraţie finală, deoarece automatul nu mai poate executa nicio tranziţie. Deoarece automatul a consumat toate simbolurile propoziţiei şi în plus a rămas în starea q_2, care este o stare finală, înseamnă că propoziţia a fost acceptată şi deci aparţine limbajului definit de către automat.

$(001, q_0) \vdash (01, q_1) \Leftrightarrow 001 \notin L(AF)$

Configuraţia $(01, q_1)$ este o configuraţie finală, deoarece automatul nu mai poate executa nicio tranziţie. Deoarece automatul nu a consumat toate simbolurile propoziţiei, înseamnă că propoziţia nu a fost acceptată şi deci nu aparţine limbajului definit de către automat.

2. Să se construiască un automatul finit care defineşte limbajul format din mulţimea propoziţiilor alcătuite pe baza simbolurilor 0 şi 1 şi formate dintr-un număr par de 0, urmaţi de un număr impar de 1. Apoi, să se precizeze ce fel de automat finit a fost construit şi să se scrie succesiunea de relaţii de mişcare pentru analiza propoziţiilor 001 şi 0011.



Se observă că automatul finit construit este un automat finit determinist.

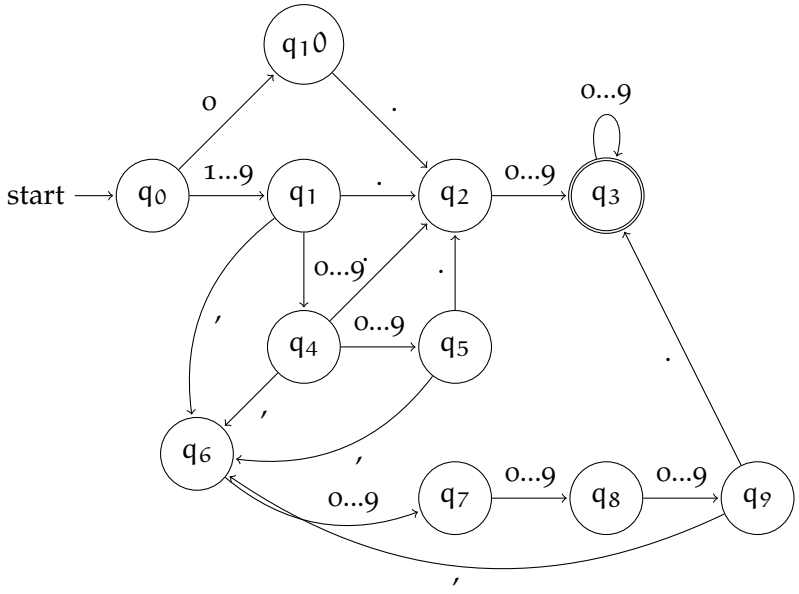
$(001, q_0) \vdash (01, q_1) \vdash (1, q_2) \vdash (\epsilon, q_3) \Leftrightarrow 001 \in L(AF)$

Configuraţia (ϵ, q_3) este o configuraţie finală, deoarece automatul nu mai poate executa nicio tranziţie. Deoarece automatul a consumat toate simbolurile propoziţiei şi în plus a rămas în starea q_3, care este o stare finală, înseamnă că propoziţia a fost acceptată şi deci aparţine limbajului definit de către automat.

$(0011, q_0) \vdash (011, q_1) \vdash (11, q_2) \vdash (1, q_3) \vdash (\epsilon, q_4) \Leftrightarrow 0011 \notin L(AF)$

Configuraţia (ϵ, q_4) este o configuraţie finală, deoarece automatul nu mai poate executa nicio tranziţie. Cu toate că automatul a consumat toate simbolurile propoziţiei, deoarece starea q_4 nu este o stare finală, înseamnă că propoziţia nu a fost acceptată şi deci nu aparţine limbajului definit de către automat.

3. Sa se construiască un automat finit care să definească limbajul numerelor reale scrise folosind separatorul de mii (virgula). Apoi, să se precizeze ce fel de automat finit a fost construit şi să se scrie succesiunea de relaţii de mişcare pentru analiza propoziţiilor 1,925.12 şi 15,0.1.



Se observă că automatul finit construit este un automat finit determinist.

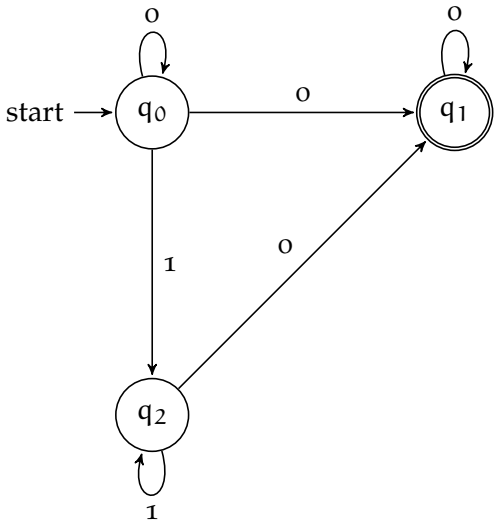
$(1, 925.12, q_0) \vdash (, 925.12, q_1) \vdash (925.12, q_6) \vdash (25.12, q_7) \vdash (5.12, q_8) \vdash (.12, q_9) \vdash (12, q_3) \vdash (2, q_3) \vdash (\epsilon, q_3) \Leftrightarrow 1,925.12 \in L(AF)$

Configuraţia (ϵ, q_3) este o configuraţie finală, deoarece automatul nu mai poate executa nicio tranziţie. Deoarece automatul a consumat toate simbolurile propoziţiei şi în plus a rămas în starea q_3, care este o stare finală, înseamnă că propoziţia a fost acceptată şi deci aparţine limbajului definit de către automat.

$(15, 0.1, q_0) \vdash (5, 0.1, q_1) \vdash (, 0.1, q_4) \vdash (0.1, q_6) \vdash (.1, q_7) \Leftrightarrow 15, 0.1 \notin L(AF)$

Configuraţia $(.1, q_7)$ este o configuraţie finală, deoarece automatul nu mai poate executa nicio tranziţie. Deoarece automatul nu a consumat toate simbolurile propoziţiei, înseamnă că aceasta nu a fost acceptată şi deci nu aparţine limbajului definit de către automat.

4. Să se construiască un automat finit cu exact trei stări, care să accepte limbajul alcătuit din propoziţii de forma $0*1*0*0$. Apoi, să se precizeze ce fel de automat finit a fost construit şi să se scrie succesiunea de relaţii de mişcare pentru analiza propoziţiilor 0010 şi 0111.



Se observă că automatul finit construit este un automat finit nedeterminist, deoarece din starea q_0 pleacă două tranziţii pentru acelaşi simbol 0.

$$(0010, q_0) = \begin{cases} \vdash (010, q_0) \begin{cases} \vdash (10, q_0) \vdash (0, q_2) \vdash (\epsilon, q_1) \\ \vdash (10, q_1) \end{cases} \\ \vdash (010, q_1) \vdash (10, q_1) \end{cases}$$

$\Leftrightarrow 0010 \in L(AF)$

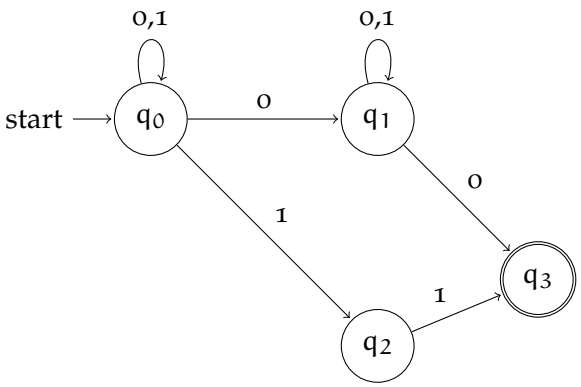
$$(0111, q_0) = \begin{cases} \vdash (111, q_0) \vdash (11, q_2) \vdash (1, q_2) \vdash (\epsilon, q_2) \\ \vdash (111, q_1) \end{cases}$$

$\Leftrightarrow 0111 \notin L(AF)$

Deoarece automatul este un automat finit nedeterminist, analiza propoziţiilor nu mai este liniară, ci arborescentă. De fiecare dată când pentru perechea formată din starea curentă şi simbolul de intrare există mai multe tranziţii posibile, va apărea o împărţire în mai multe direcţii posibile de continuare. Se va obţine astfel un arbore. Dacă în arbore există cel puţin o ramură pe care propoziţia a fost acceptată, atunci ea aparţine limbajului definit de către automat. Dacă în arbore nu există nicio ramură pe care propoziţia să fi fost acceptată, atunci ea nu aparţine limbajului definit de către automat.

Pentru exemplele de mai sus, propoziţia 0010 a fost acceptată pe ramura care se termină cu (ϵ, q_1) şi deci aparţine limbajului definit de către automat. În schimb, propoziţia 0111 nu a fost acceptată pe nicio ramură şi deci nu aparţine limbajului definit de către automat.

5. Să se deducă automatul finit determinist echivalent pentru următorul automat finit nedeterminist:



Elementele automatului finit determinist echivalent se deduc aplicând teorema cu acelaşi nume. Toate aceste elemente pot fi observate în tabelul de definiţie a funcţiei de tranziţie a acestui automat, dat mai jos.

f′	0	1
∅	∅	∅
$\{q_0\}$	$\{q_0q_1\}$	$\{q_0q_2\}$
$\{q_1\}$	$\{q_1q_3\}$	$\{q_1\}$
$\{q_2\}$	∅	$\{q_3\}$
*$\{q_3\}$	∅	∅
$\{q_0q_1\}$	$\{q_0q_1q_3\}$	$\{q_0q_1q_2\}$
$\{q_0q_2\}$	$\{q_0q_1\}$	$\{q_0q_2q_3\}$
*$\{q_0q_3\}$	$\{q_0q_1\}$	$\{q_0q_2\}$
$\{q_1q_2\}$	$\{q_1q_3\}$	$\{q_1q_3\}$
*$\{q_1q_3\}$	$\{q_1q_3\}$	$\{q_1\}$
*$\{q_2q_3\}$	∅	$\{q_3\}$
$\{q_0q_1q_2\}$	$\{q_0q_1q_3\}$	$\{q_0q_1q_2q_3\}$
*$\{q_0q_1q_3\}$	$\{q_0q_1q_3\}$	$\{q_0q_1q_2\}$
*$\{q_0q_2q_3\}$	$\{q_0q_1\}$	$\{q_0q_2q_3\}$
*$\{q_1q_2q_3\}$	$\{q_1q_3\}$	$\{q_1q_3\}$
*$\{q_0q_1q_2q_3\}$	$\{q_0q_1q_3\}$	$\{q_0q_1q_2q_3\}$

Graful de tranziţie redus al automatului finit determinist echivalent este următorul:

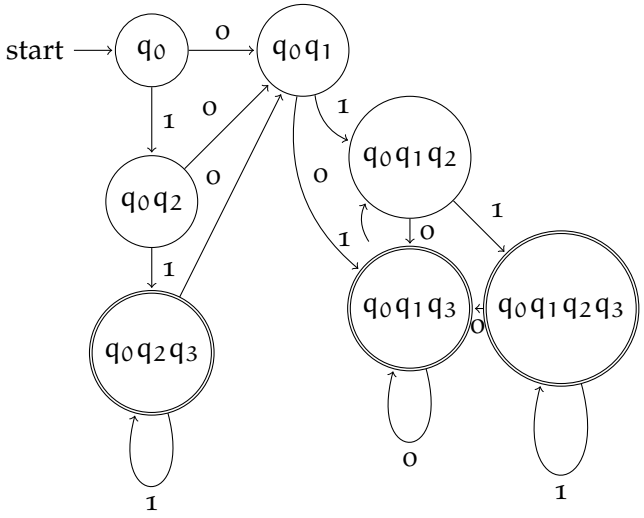
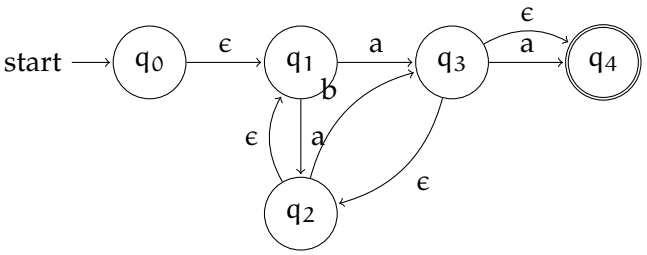
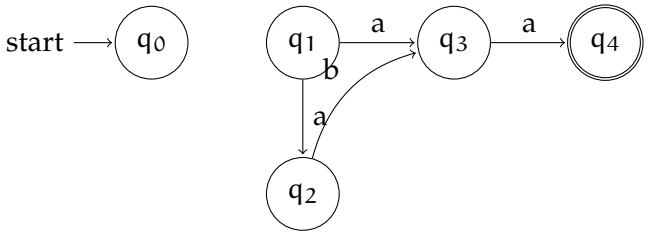


Figura 26: Automatul finit determinist

6. Să se deducă automatul finit nedeterminist echivalent pentru următorul automat finit ϵ:



Primul pas al construcţiei automatului finit nedeterminist echivalent este desenarea grafului de tranziţie a automatului fără tranziţiile ϵ.



În cadrul celui de al doilea pas, fiecare tranziţie ϵ eliminată este înlocuită cu una sau mai multe tranziţii, aşa cum s-a arătat în cadrul secţiunii corespunzătoare.

a) Înlocuirea pentru **tranziţia** ϵ **de la** q_0 **la** q_1, se va face având în vedere cele două tranziţii care pleacă din q_1, una către q_3, iar cealaltă către q_2.

   ★ $f(q_0, \epsilon) = q_1$ şi $f(q_1, a) = q_3 \Leftrightarrow f(f(q_0, \epsilon), a) = q_3$

   Eliminând tranziţia ϵ, putem scrie că:

   $f'(q_0, a) = q_3$

   ★ $f(q_0, \epsilon) = q_1$ şi $f(q_1, a) = q_2 \Leftrightarrow f(f(q_0, \epsilon), a) = q_2$

   Eliminând tranziţia ϵ, putem scrie că:

   $f'(q_0, a) = q_2$

   ★ Prin urmare, tranziţia $f(q_0, \epsilon) = q_1$ a automatului finit ϵ va fi înlocuită în automatul finit nedeterminist cu tranziţiile $f'(q_0, a) = q_2$ şi $f'(q_0, a) = q_3$.

b) Înlocuirea pentru **tranziţia** ϵ **de la** q_2 **la** q_1, se va face având în vedere aceleaşi două tranziţii care pleacă din q_1, care au fost considerate anterior.

   ★ $f(q_2, \epsilon) = q_1$ şi $f(q_1, a) = q_3 \Leftrightarrow f(f(q_2, \epsilon), a) = q_3$

   Eliminând tranziţia ϵ, putem scrie că:

   $f'(q_2, a) = q_3$

   ★ $f(q_2, \epsilon) = q_1$ şi $f(q_1, a) = q_2 \Leftrightarrow f(f(q_2, \epsilon), a) = q_2$

   Eliminând tranziţia ϵ, putem scrie că:

   $f'(q_2, a) = q_2$

   ★ Prin urmare, tranziţia $f(q_2, \epsilon) = q_1$ a automatului finit ϵ va fi înlocuită în automatul finit nedeterminist cu tranziţiile $f'(q_2, a) = q_2$ şi $f'(q_2, a) = q_3$.

c) În mod asemănător se va proceda şi pentru **tranziţia** ϵ **de la** q_3 **la** q_2.

   ★ $f(q_3, \epsilon) = q_2$ şi $f(q_2, b) = q_3 \Leftrightarrow f(f(q_3, \epsilon), b) = q_3$

   Eliminând tranziţia ϵ, putem scrie că:

   $f'(q_3, b) = q_3$

   ★ $f(q_3, \epsilon) = q_2$ şi $f(q_2, \epsilon) = q_1$ şi $f(q_1, a) = q_2 \Leftrightarrow f(f(f(q_3, \epsilon), \epsilon), a) = q_2$

   Eliminând cele două tranziţii ϵ, putem scrie că:

   $f'(q_3, a) = q_2$

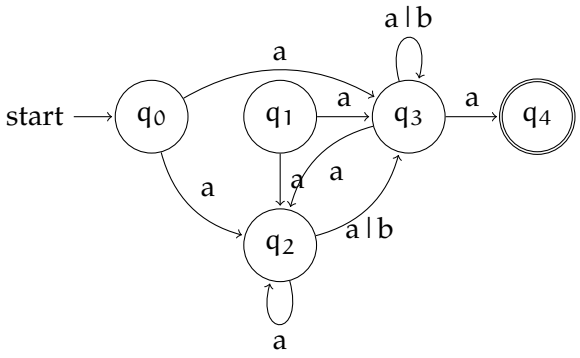
★ $f(q_3, \epsilon) = q_2$ şi $f(q_2, \epsilon) = q_1$ şi $f(q_1, a) = q_3 \Leftrightarrow f(f(f(q_3, \epsilon), \epsilon), a) = q_3$

Eliminând cele două tranziţii ϵ, putem scrie că:

$f'(q_3, a) = q_3$

★ Prin urmare, tranziţia $f(q_3, \epsilon) = q_2$ a automatului finit ϵ va fi înlocuită în automatul finit nedeterminist cu tranziţiile $f'(q_3, a) = q_2$, $f'(q_3, a) = q_3$ şi $f'(q_3, b) = q_3$.

d) Având în vedere că din starea q_4 nu pleacă nicio tranziţie, **tranziţia ϵ de la q_3 la q_4** va fi eliminată fără a fi înlocuită cu o altă tranziţie.



Ultimul pas al construcţiei, constă în stabilirea stărilor automatului finit nedeterminist care devin stări finale. Acest lucru se face pe baza calculului închiderii fiecărei stări a automatului finit ϵ. Stările automatului finit ϵ pentru care închiderea conţine o stare finală în cadrul acestui automat, vor deveni stări finale în automatul finit nedeterminist echivalent.

$CL(q_0) = \{q_1\}$

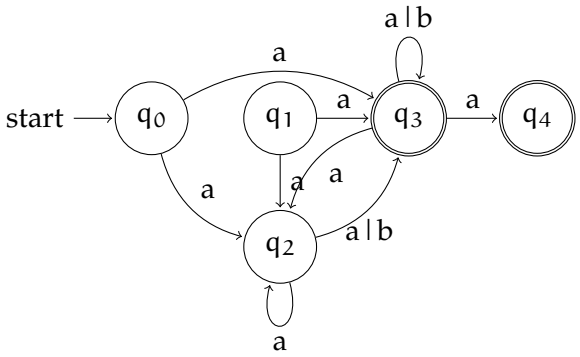
$CL(q_1) = \emptyset$

$CL(q_2) = \{q_1\}$

$CL(q_3) = \{q_1, q_2, q_4\}$

$CL(q_4) = \emptyset$

Deoarece închiderea stării q_3 conţine o stare finală a automatului finit ϵ (adică pe q_4), starea q_3 devine stare finală a automatului finit nedeterminist echivalent.



3.1.6 *Exerciţii propuse*

1. Să se dezvolte într-un limbaj de programare oarecare o aplicaţie care să automatizeze procesul de construire a automatului finit determinist echivalent pentru un automat finit nedeterminist dat. Pentru automatul finit nedeterminist se vor da, prin intermediul unui fişier de intrare, alfabetul, mulţimea stărilor, starea iniţială, mulţimea stărilor finale şi tabelul de definiţie a funcţiei de tranziţie. Aplicaţia va desena graful de tranziţie al automatului finit determinist echivalent.

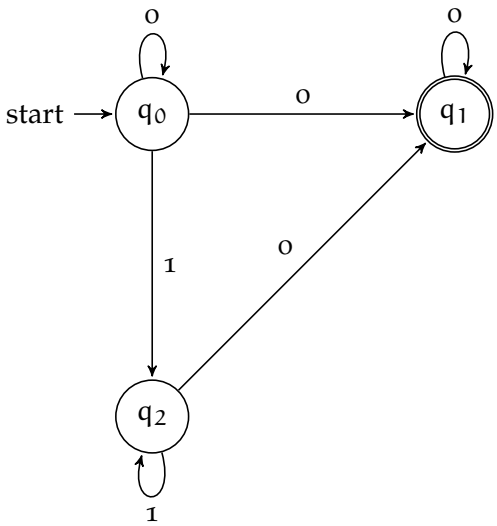
2. Să se dezvolte într-un limbaj de programare oarecare o aplicaţie care să automatizeze procesul de construire a automatului finit nedeterminist echivalent pentru un automat finit ϵ dat. Pentru automatul finit ϵ se vor da, prin intermediul unui fişier de intrare, alfabetul, mulţimea stărilor, starea iniţială, mulţimea stărilor finale şi tabelul de definiţie a funcţiei de tranziţie. Aplicaţia va desena graful de tranziţie al automatului finit nedeterminist echivalent.

3. Să se dezvolte într-un limbaj de programare oarecare o aplicaţie care să construiască gramatica formală echivalentă pentru un automat finit determinist dat. Pentru automatul finit determinist se vor furniza, prin intermediul unui fişier de intrare, alfabetul, mulţimea stărilor, starea iniţială, mulţimea stărilor finale şi tabelul de definiţie a funcţiei de tranziţie. Aplicaţia va furniza elementele de definiţie a gramaticii formale echivalente.

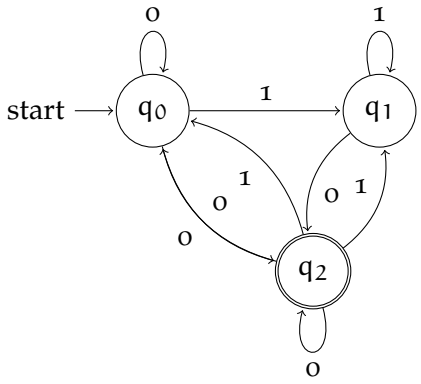
   Observaţii:

   - ★ Alfabetul automatului finit determinist va constitui mulţimea terminalelor gramaticii.
   - ★ Mulţimea stărilor automatului va constitui mulţimea neterminalelor gramaticii.
   - ★ Starea iniţială va constitui simbolul de start.
   - ★ Regulile de producţie se vor construi pe baza tranziţiilor automatului.

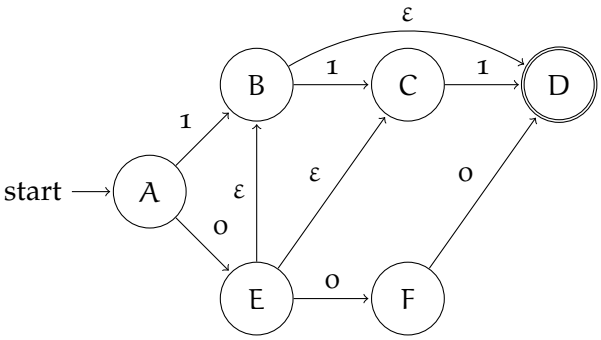
4. Să se deducă automatul finit determinist echivalent pentru următorul automat finit nedeterminist:



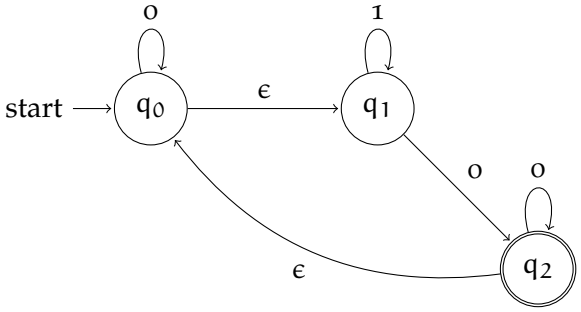
5. Să se deducă automatul finit determinist echivalent pentru următorul automat finit nedeterminist:



6. Să se deducă automatul finit nedeterminist echivalent pentru următorul automat finit ϵ:



7. Să se deducă automatul finit nedeterminist echivalent pentru următorul automat finit ϵ:



3.2 EXPRESII REGULATE

Asemeni automatelor finite, expresiile regulate constituie un metalimbaj, adică un limbaj care permite definirea unor alte limbaje. Expresiile regulate au aceeaşi expresivitate ca şi automatele finite şi ca şi gramaticile regulate, adică permit numai definirea limbajelor regulate.

Definitie 32 *O expresie regulata* R *este definită recursiv, astfel:*

- ★ *Şirul vid* ϵ *este o expresie regulată, iar limbajul definit de această expresie regulată este limbajul care nu conţine nicio propoziţie.*

$$L(\epsilon) = \{\}$$

- ★ *Orice simbol singular* c *este o expresie regulată, iar limbajul definit de această expresie regulată este limbajul format dintr-o singură propoziţie, şi anume propoziţia* c.

$$L(c) = \{c\}$$

- ★ ***Concatenarea:*** *Dacă* R_1 *şi* R_2 *sunt două expresii regulate, atunci* $R1R2$ *(se citeşte* R_1 *concatenat cu* R_2*) este o expresie regulată, iar limbajul definit de această expresie regulată este definit astfel:*

$$L(R_1R_2) = \{s_1s_2 | s_1 \in R1 \wedge s_2 \in R_2\}$$

- ★ ***Reuniunea:*** *Dacă* R_1 *şi* R_2 *sunt două expresii regulate, atunci* $R_1|R_2$ *(se citeşte* R_1 *sau* R_2*) este o expresie regulată, iar limbajul definit de această expresie regulată este definit astfel:*

$$L(R_1|R_2) = L(R_1) \cup L(R_2).$$

- ★ ***Închiderea:*** *Dacă* R *este o expresie regulată, atunci* R^* *este o expresie regulată, iar limbajul definit de această expresie regulată este definit astfel:*

$$L(R^*) = L(\epsilon) \cup L(R) \cup L(RR) \cup L(RRR) \cup \ldots$$

  Cu alte cuvinte, limbajul definit de expresia regulată R^* *este limbajul format din mulţimea propoziţiilor obţinute prin concatenarea a zero sau mai multe propoziţii, fiecare din* $L(R)$.

- ★ ***Închiderea nereflexivă:*** *Dacă* R *este o expresie regulată, atunci* R^+ *este o expresie regulată, iar limbajul definit de această expresie regulată este definit astfel:*

$$L(R^+) = L(R) \cup L(RR) \cup L(RRR) \cup \ldots$$

  Cu alte cuvinte, limbajul definit de expresia regulată R^+ *este limbajul format din mulţimea propoziţiilor obţinute prin concatenarea a una sau mai multe propoziţii, fiecare din* $L(R)$.

  R^+ *este o scriere prescurtată pentru* RR^*.

- ★ ***Opţional:*** *Dacă* R *este o expresie regulată, atunci* $R?$ *este o expresie regulată, iar limbajul definit de această expresie regulată este definit astfel:*

$$L(R?) = L(R)|L(\epsilon)$$

  $R?$ *este o scriere prescurtată pentru* $R|\epsilon$.

În general, motoarele de procesare a expresiilor regulate implementate de către aplicaţiile care permit utilizarea lor, permit şi folosirea următoarelor notaţii:

- ★ $[c_1c_2\ldots]$ este o prescurtare pentru $c_1|c_2|\ldots$.

★ $[a_1 - a_n]$ este o prescurtare pentru $a_1|\ldots|a_n$.

★ $[\hat{}c_1c_2\ldots]$ este o prescurtare pentru $d_1|d_2|\ldots$, unde $d_1, d_2, \ldots$ sunt toate acele simboluri care nu apar printre $c_1, c_2, \ldots$.

★ . este o prescurtare pentru toate şirurile de caractere care sunt formate dintr-un singur caracter, iar acesta nu este sfârşitul de linie.

În continuare vor fi prezentate câteva exemple de utilizare a expresiilor regulate pentru a căuta o anumită mulţime de şiruri de caractere într-un text dat.

★ Potrivirea unui şir de caractere
  - **Text**: Anna Jones and a friend went to lunch
  - **Regex**: went
  - **Matches**: Anna Jones and a friend **went** to lunch

★ Potrivirea unui singur caracter oarecare, cu excepţia sfârşitului de linie
  - **Text**: abc def ant cow
  - **Regex**: a..
  - **Matches**: **abc** def **ant** cow

★ Potrivirea unui singur caracter dintr-o mulţime dată
  - **Text:** abc def ant cow
  - **Regex:** [da]..
  - **Matches:** **abc def ant** cow

  - **Text**: abc pen nda uml
  - **Regex**: .[a-d].
  - **Matches**: **abc** pen **nda** uml

★ Potrivirea unui spaţiu alb
  - **Text**: abc anaconda ant
  - **Regex**: a[ ]
  - **Matches**: abc anaconda ant

★ Potrivirea oricărui caracter cu excepţia celor din mulţimea specificată
  - **Text:** abc def ant cow
  - **Regex:** [^da]..
  - **Matches:** abc def ant cow

★ Specificarea numărului de potriviri - de zero sau de mai multe ori
  - **Text**: Anna Jones and a friend owned an anaconda
  - **Regex**: a[a-z]*
  - **Matches**: **Anna** Jones **and a** friend owned **an anaconda**

★ Specificarea numărului de potriviri - o data sau de mai multe ori

- **Text**: Anna Jones and a friend owned an anaconda
- **Regex**: a[a-z]+
- **Matches**: **Anna** Jones **and** a friend owned **an anaconda**

★ Specificarea numărului de potriviri - de zero ori sau o data

- **Text:** Anna Jones and a friend owned an anaconda
- **Regex:** an?
- **Matches: An**n**a** Jones **an**d **a** friend owned **an a**cond**a**

★ Specificarea exactă a numărului de potriviri

- **Text:** Anna Jones and Anne owned an anaconda
- **Regex:** an{2}
- **Matches: Ann**a Jones and **Ann**e owned an anaconda

- **Text:** Anna and Anne lunched with an anaconda annnnnex
- **Regex:** an{2, 3}
- **Matches: Ann**a and **Ann**e lunched with an anaconda **annn**nnex

★ Potrivirea începutului textului în care se face căutarea

- **Text:** an anaconda ate Anna Jones
- **Regex:** ∧a
- **Matches: a**n anaconda ate Anna Jones

★ Potrivirea sfârşitului textului în care se face căutarea

- **Text:** an anaconda ate Anna Jones
- **Regex:** [a-zA-Z]+$
- **Matches:** an anaconda ate Anna **Jones**

3.2.1 *Convenţii de aplicare a expresiilor regulate*

3.2.1.1 *Precedenţa*

În scrierea expresiilor regulate, precum şi în implementarea motoarelor de expresii regulate, trebuie să se ţină seama de următoarea precedenţă: operatorii închidere, închidere nereflexivă şi opţional au prioritate faţă de concatenare, iar concatenarea are prioritate faţă de reuniune. Pentru o scriere clară şi fără ambiguităţi se pot utiliza parantezele rotunde.

De exemplu, expresia regulată:

$$ab?cd*ef|g$$

este echivalenta cu:

$$(a(b?)c(d*)ef)|g$$

şi reprezintă limbajul format din propoziţiile: *acef*, *g*, *abcef*, *acdddef*, *abcdddef*, *abcddddef*, ş.a.m.d.

3.2.1.2 *Prioritatea*

În aplicarea expresiilor regulate asupra unui text dat, poate să apară situaţia în care acelaşi şir de caractere din textul în care se face căutarea, să se potrivească mai multor expresii regulate dintre cele folosite la căutare. De aceea, în general, prioritatea expresiilor regulate folosite la căutare este dată de ordinea în care ele sunt utilizate.

De exemplu, fie expresiile regulate:

(ab)*

şi

[a-c]*

Şirul de caractere "ab" se potriveşte ambelor expresii. Însă, având în vedere ordinea în care au fost date cele două expresii regulate, se va considera că "ab" se potriveşte lui *(ab)**.

Având în vedere această convenţie, dacă în textul în care se face căutarea folosind o expresie regulată de tipul $R_1|R_2$, există un şir de caractere care se potriveşte atât expresiei regulate R_1, cât şi expresiei regulate R_2, atunci se va considera că se potriveşte lui R_1.

De asemenea, în aplicarea expresiilor regulate asupra unui text dat, poate să apară şi situaţia în care, în textul în care se face căutarea, să existe şiruri de caractere diferite, care să se potrivească aceleiaşi expresii regulate. În această situaţie, se foloseşte convenţia **leftmost longest**: dintre toate şirurile de caractere din textul în care se face căutarea, care se potrivesc aceleiaşi expresii regulate, se va alegea cel mai lung dintre ele, căutat de la stânga spre dreapta.

De exemplu, fie expresia regulată:

ab | abcd

Şi fie textul:
"abcd".

Expresiei regulate se potrivesc următoarele două şiruri de caractere: "ab" sau "abcd". Având în vedere convenţia **leftmost longest**, aplicarea expresiei regulate va duce la potrivirea celui de-al doilea şir de caractere, care este mai lung.

3.2.2 *De la expresii regulate la automate finite*

Având în vedere că expresiile regulate şi automatele finite sunt formalisme echivalente, ambele permiţând definirea limbajelor regulate, înseamnă că pentru orice expresie regulată dată se poate construi un automat finit echivalent, adică un automat finit care să definească exact acelaşi limbaj. Pentru fiecare dintre expresiile regulate de bază, din definiţia recursivă a expresiilor regulate, există un automat finit ϵ echivalent. Pornind de la aceste echivalenţe, se poate construi automatul finit echivalent oricărei expresii regulate date, indiferent de complexitatea sa.

În continuare, vor fi prezentate automatele finite ϵ echivalente fiecărei expresii regulate de bază. În acest scop se vor folosi următoarele convenţii:

- ★ O elipsă reprezintă un automat, care recunoaşte expresia regulată R, sau aceeaşi maşină, dar cu toate stările componente transformate în stări nefinale.
- ★ O elipsă este o colecţie de stări şi tranziţii.
- ★ Cele două stări marcate reprezintă starea iniţială şi respectiv starea finală, sau starea iniţială şi starea finală, transformată într-o stare nefinală.

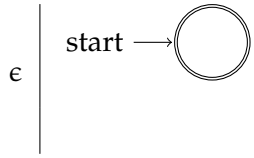


Figura 27: Automatul finit ϵ echivalent expresiei regulate ϵ

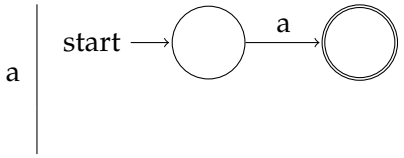


Figura 28: Automatul finit ϵ echivalent expresiei regulate a

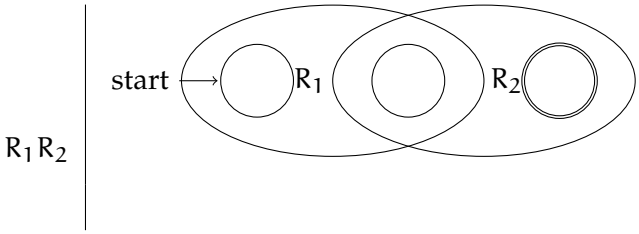


Figura 29: Automatul finit ϵ echivalent expresiei regulate R_1R_2

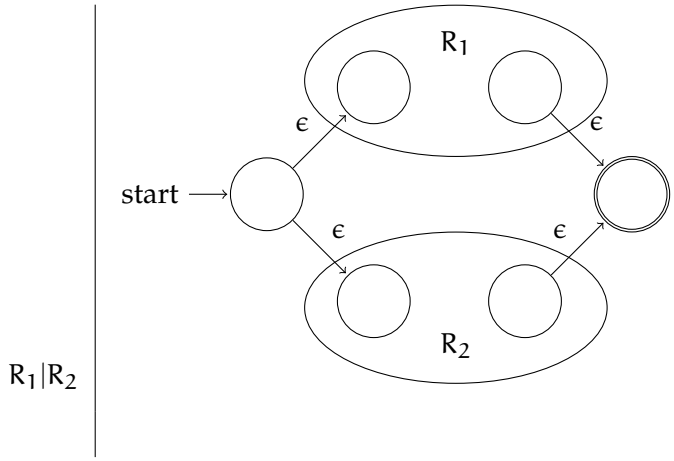


Figura 30: Automatul finit ϵ echivalent expresiei regulate $R_1|R_2$

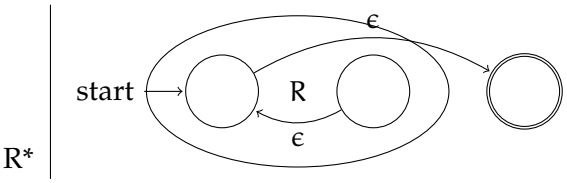


Figura 31: Automatul finit ϵ echivalent expresiei regulate R∗

3.3 DEMONSTRAREA/INFIRMAREA REGULARITĂŢII UNUI LIMBAJ

Metoda uzuală folosită pentru a demonstra că un limbaj este regular constă în a construi un automat cu stări finite sau o expresie regulară care să accepte limbajul respectiv. Cum se poate însp demonstra că un limbaj nu este regular? În continuare vor fi oferite trei astfel de metode/criterii: principiul cuiburilor de porumbei, lema pompării și teorema MyHill-Nerode à la Brzozowski.

3.3.1 *Principiul cuiburilor de porumbei*

Principiul cuiburilor de porumbei afirmă faptul că dacă *n* porumbei zboară spre *m* cuiburi şi dacă *n>m*, atunci cel puţin un cuib va avea doi sau mai mulţi porumbei.

Acest principiu poate fi aplicat şi automatelor finite. Deoarece acestea au un număr finit de stări şi deoarece, în general, există un număr infinit de propoziţii care sunt acceptate de către automat, atunci trebuie să existe cel puţin o stare la care automatul, în analiză, se întoarce de două sau de mai multe ori. În cadrul acestei analogii, stările automatului sunt cuiburile, iar simbolurile propoziţiilor de la intrare sunt porumbeii.

Fie limbajul format din orice succesiune de 0 şi 1 care începe cu un 1, continuă apoi cu unul sau mai mulţi de zero, iar la sfârşit are un 1. Mulţimea propoziţiilor acestui limbaj este infinită, cuprinzând succesiunile de simboluri: 101, 1001, 10001, $10\ldots01$. Cu toate acestea, mulţimea stărilor automatului este finită. Prin urmare, conform principiului cuiburilor de porumbei, trebuie să există o stare q_m astfel încât, în analiza a două succesiuni de simboluri 10^p şi 10^q, cu $p \neq q$, automatul va ajunge în aceeaşi stare q_m.

Principiul cuburilor de porumbei poate fi folosit pentru a demonstra că un limbaj dat nu este regulat. Vom exemplifica acest lucru pentru următorul limbaj:

$L = \{a^n b^n | n \geqslant 1\}$

Dacă ar exista un automat finit care să definească acest limbaj, atunci el ar trebui ca, pe măsură ce analizează şirul de la intrare, să ţină minte numărul de simboluri de a pe care le-a analizat. Valorile pe care le poate lua numărul de simboluri de a sunt infinite şi deci automatul ar trebui să poată să gestioneze un număr infinit de posibilităţi. Acest lucru însă nu se poate face cu un număr finit de stări. Deci limbajul dat este un limbaj neregulat.

În continuare se va demonstra acest lucru folosind principiul cuiburilor de porumbei. Demonstraţia va fi o demonstraţie prin reducere la absurd.

DEMONSTRAȚIE: Presupunem că limbajul L este un limbaj regulat. Atunci, înseamnă că există un automat finit $AF = (\{a, b\}, Q, F, q_0, f)$ care acceptă acest limbaj. Considerăm $f^*(q_0, a^i)$ pentru i egal cu $1, 2, \ldots$. Deoarece i, numărul de a, poate lua un număr infinit de valori, şi deoarece mulţimea de stări din automat este finită, atunci, conform principiului cuiburilor de porumbei, există o stare q_k astfel încât:

$f^*(q_0, a^m) = q_k$ şi $f^*(q_0, a^n) = q_k$, cu $m \neq n$.

Dar, deoarece automatul accepta şirurile de forma $a^n b^n$, atunci este obligatoriu să fie adevărat că:

$f^*(q_k, b^n) = q_f \in F$.

Pe baza relaţiile scrise mai sus, se poate deduce că:

$f^*(q_0, a^m b^n) = f^*(f^*(q_0, a^m), b^n) = f^*(q_k, b^n) = q_f \in F$

Dar acest lucru înseamnă că automatul accepta şi şirurile de forma $a^m b^n$, cu $m \neq n$, ceea ce contrazice presupunerea iniţială cum că el accepta limbajul L. Deci presupunerea iniţială este falsă şi deci L este un limbaj neregulat. □

3.3.2 *Lema pompării pentru limbajele regulare*

Teorema 1 ***[Lema pompării]****Dacă L este un limbaj regulat, atunci există un număr natural* $N \geqslant 1$*, astfel încât* $\forall$ *propoziţie* $w \in L$*, unde* $|w| \geqslant N$*,* $\exists$ *x, y, z astfel încât w=xyz, şi :*

- ★ $|xy| \leqslant N$
- ★ $y \neq \epsilon$
- ★ $\forall\, q \geqslant o$, $xy^q z \in L$.

Cu $|x|$ s-a notat lungimea propoziţiei x, iar cu x^q s-a notat concatenarea propoziţiei x cu ea însăşi de un număr de ori egal cu q.

Cu alte cuvinte, lema pompării enunţă faptul că dacă L este un limbaj regulat, atunci există un număr natural N care este cel puţin egal cu 1, astfel încât, pentru orice propoziţie w din limbajul L, care are lungimea cel puţin egală cu N, există o formă de scriere prin concatenarea a trei subşiruri x, y, z, cu proprietăţile că lungimea subşirului xy este cel mult egală cu N, iar lungimea subşirului y este cel puţin egală cu 1. În aceste condiţii, subşirul y poate fi pompat ori de câte ori între x şi z ($xyy \ldots yz$) sau chiar poate fi eliminat (xz), iar propoziţiile obţinute astfel aparţin şi ele limbajului L.

DEMONSTRAŢIE: Dacă L este un limbaj regulat, atunci înseamnă că există un automat finit determinist care îl defineşte. Presupunem că automatul are un număr $n+1$ de stări şi că acestea sunt etichetate cu $q_0, q_1, \ldots, q_n$.

Fie atunci o propoziţie $w \in L$, astfel încât $|w| \geqslant m = n+1$ şi fie $q_0, q_i, q_j, \ldots, q_f$ succesiunea de stări prin care automatul trece în analiza propoziţiei w (evident q_0 este starea iniţială, iar q_f este o stare finală). Numărul de stări din cadrul acestei succesiuni de stări este egal cu lungimea şirului w plus 1, deoarece pentru a accepta un număr de simboluri egale cu lungimea lui w, este nevoie de acelaşi număr de tranziţii, iar numărul de stări necesare pentru a forma aceste tranziţii este mai mare cu 1. Acest lucru implică faptul că cel puţin o stare din succesiunea $q_0, q_i, q_j, \ldots, q_f$ se repetă. Prin urmare, succesiune de stări poate fi rescrisă $q_0, q_i, q_j, \ldots, q_r, \ldots, q_r, \ldots, q_f$.

Această scriere a succesiuni de stări prin care trece automatul în analiza şi acceptarea propoziţiei w, indică că aceasta poate fi scrisă ca o concatenare a trei subşiruri x, y, z, astfel încât:

- ★ şirul x este prima parte a propoziţiei w, pentru analiza căreia automatul trece prin succesiunea $q_0, q_i, q_j, \ldots, q_r$, şi se poate scrie:

  $(q_0, x) \vdash^* (q_r, \epsilon)$ sau $f^*(q_0, x) = q_r$
- ★ şirul y este partea de mijloc a propoziţiei w, pentru analiza căreia automatul trece prin succesiunea $q_r, \ldots, q_r$, şi se poate scrie:

  $(q_r, y) \vdash^+ (q_r, \epsilon)$ sau $f^*(q_r, y) = q_r$
- ★ iar şirul z este ultima parte a propoziţiei w, pentru analiza căreia automatul trece prin succesiunea $q_r, \ldots, q_f$, şi se poate scrie:

  $(q_r, z) \vdash^+ (q_f, \epsilon)$ sau $f^*(q_r, z) = q_f$.

De aici rezultă că:

- ★ $|xy| \leqslant n+1 = m$ deoarece am presupus ca singura stare care se repetă în analiza propoziţiei w este q_r şi deci, în rest, stările sunt distincte. Prin urmare, pentru şirurile x, y se pot analiza un număr maxim de simboluri egal cu numărul maxim de stări care pot fi implicate în analiză, adică $n+1$;

★ $|y| \geqslant 1$, deoarece s-a arătat că starea q_r trebuie să se repete, şi deci între cele două apariţii ale acestei stări va fi acceptat cel puţin un simbol (în cazul în care a doua apariţie a lui q_r urmează imediat după prima).

Totodată, având în vedere relaţiile de mai sus, se poate scrie că:

1. $(q_0, xz) \vdash^* (q_r, z) \vdash^+ (q_f, \epsilon)$, ceea ce înseamnă că $xz \in L$;
2. $(q_0, xy^2z) \vdash^* (q_r, y^2z) \vdash^+ (q_r, yz) \vdash^+ (q_r, z) \vdash^+ (q_f, \epsilon)$, ceea ce înseamnă că $xy^2z \in L$;
3. în acelaşi mod se poate arăta că $xy^3z \in L$ şi $xy^4z \in L$, şi, în general, $xy^qz \in L$, pentru orice număr natural q.

Cu aceasta, demonstraţia teoremei s-a încheiat. □

Lema pompării este folosită pentru a demonstra că un limbaj dat nu este un limbaj regulat. Această demonstraţie se face totdeauna prin reducere la absurd.

În continuare se va exemplifica modul în care se utilizează această teoremă pentru a arăta că un limbaj dat nu este regulat și anume se va arăta că limbajul:

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

nu este un limbaj regular.

DEMONSTRAȚIE: Presupunem că limbajul este unul regular. Atunci, înseamnă că i se poate aplica lema pompării și anume: există un număr natural $N \geqslant 1$ astfel încât pentru orice propoziție $w \in L$ care are lungimea cel puțin egală cu N se îndeplinesc afirmațiile lemei.

Fie $w = a^N\ b^N$. Atunci, indiferent de modul în care propoziția w este scrisă ca și o concatenare de trei șiruri xyz, se îndeplinesc afirmațiile teoremei. Având în vedere că, în conformitate cu lema pompării, $|xy| \leqslant N$ și având în vedere felul în care l-am ales pe w, înseamnă că y poate lua următoarele forme:

$y=a^i$, $1 \leqslant |y| \leqslant N$.

Însă, pentru că $|xy| \leqslant N$:

$x = a^j$, cu condiția că $i + j \leqslant N$.

Cu aceste observații, rezultă că șirul w are forma:

$w = a^j\ a^i\ a^{N-(i+j)}\ b^N$.

În aceste condiții, lema afirmă că pentru orice număr natural $q \geqslant 0$, $xy^qz \in L$.

Pentru că demonstrația pe care o facem este una prin reducere la absurd, căutăm o valoare a lui q astfel încât:

$w' = a^j\ a^{iq}\ a^{N-(i+j)}\ b^N \notin L$,

unde w' este şirul pompat.

Se observă că pentru $q = 0$:

$w' = a^j\ a^{N-(i+j)}\ b^N = a^{N-i}\ b^N \notin L$.

Am ajuns astfel la o contradicție, prin urmare, limbajul L nu este un limbaj regular. □

Lema pompării nu poate fi folosită pentru a demonstra că un limbaj este regulat, deoarece reciproca ei nu este adevărată: există limbaje care pot fi pompate, dar care, totuşi, nu sunt regulate. În continuare se va exemplifica acest lucru pe baza limbajului:

L = { uvwxy : u,y ∈ {0,1,2,3}* ; v, w, x ∈ {0,1,2,3} ∧ (v = w ∨ v=x ∨ x=w) } ∪ { w : w ∈ { 0,1,2,3 }* ∧ exact 1/7 simboluri din w să fie 3 }.

Cu alte cuvinte, limbajul L este reprezentat de mulțimea tuturor propozițiile construite pe baza alfabetului {0,1,2,3} care au un subșir de trei simboluri, în care un același simbol apare de două ori, precum și toate propozițiile formate pe baza aceluiași alfabet în care exact 1/7 dintre simboluri sunt simbolul 3.

Acest limbaj nu este un limbaj regular (spre exemplu $(013)^{3m}(012)^{i} \in L$ dacă și numai dacă $i = 4m$), dar poate fi pompat pentru N având valoarea 5. Fie deci o propoziție w care are lungimea cel puțin egală cu 5. Deoarece alfabetul limbajului are numai 4 simboluri, înseamnă că cel puțin două dintre primele 5 simboluri sunt identice, iar aceste două simboluri identice sunt separate de cel mult trei simboluri.

★ Dacă cele două simboluri identice nu sunt separate de niciun simbol sau sunt separate de un singur simbol, atunci se poate pompa oricare dintre celelalte două simboluri din propoziție, fără a afecta subșirul care conține simbolurile duplicate.

Variantele posibile pot fi schematizate astfel:

aabcd, **a**b**a**cd, b**aa**cd, b**a**c**a**d, bc**aa**d, bc**a**d**a**, bcd**aa**, etc.

Spre exmplu, pentru prima variantă, se poate pompa simbolul c sau simbolul d, obținându-se:

aabc^{q}d sau **aa**bcd^{q}.

Se observă că subșirul care conține simbolurile identice (aab) nu este afectat și deci propozițiile obținute aparțin limbajului L (având un subșir de trei simboluri dintre care două sunt identice).

★ Dacă cele două simboluri identice sunt separate de două sau de trei simboluri, atunci pot fi pompate două dintre simbolurile care le separă. Pomparea va duce la obținerea unui subșir de trei simboluri, dintre care două sunt identice.

Variantele posibile pot fi schematizate astfel:

abc**a**d, **a**bcd**a**, b**a**cd**a**, etc.

Spre exemplu, pentru prima variantă, se poate pompa grupul bc, obținându-se:

a(bc)^{q}ad.

Este evident că orice repetare a grupului bc va duce la crearea unui subșir de trei simboluri dintre care două sunt identice.

Exemplul de mai sus ne arată că lema pompării ne dă o condiţie necesară, dar nu și suficientă, pentru a arata că un limbaj este regular.

3.3.3 *Teorema Myhill-Nerode à la Brzozowski*

Janusz Antoni Brzozowski a definit în anul 1964 operația de derivare a unui limbaj, definind operația de derivare a unei expresii regulare. Derivata unui limbaj L în raport cu un simbol c este un nou limbaj obținut prin aplicarea următoarelor două operații:

1. Se reține toate propozițiile limbajului L care încep cu simbolul c.
2. Se elimină simbolul c din poate aceste propoziții.

Formal, derivata unui limbaj L în raport cu un simbol c se definește astfel:
$\frac{d}{dc}L = \{ v \mid cv \in L \}$.
Oferim mai jos câteva exemple de derivate pentru niște expresi regulare.
$\frac{d}{da}$ ab*a = b*a
$\frac{d}{db}$ ab*a =$\emptyset$
$\frac{d}{db}$ b*a = b*a

Utilizând operația de derivare, este foarte ușor de observat și de demonstrat că derivata unui limbaj regular este tot un limbaj regular. Orice limbaj regular poate fi definit prin intermediul unui automat cu stări finite. Dar, operația de derivare, prin care se obține limbajul derivat, corespunde schimbării stării inițiale a automatului finit care definește limbajul supus operației de derivare. Prin urmare și derivata unui limbaj este definită printr-un automat cu stări finite, ceea ce înseamnă că este un limbaj regular.

Utilizând un raționament similar, se poate observa și demonstra cum că un limbaj regular are un număr finit de derivate distincte. Având în vedere că operația de derivare a unui limbaj înseamnă de fapt schimbarea stării inițiale a automatului cu stări finite care definește limbajul supus derivării și având în vedere că numărul de stări ale unui automat este finit, rezultă că numărul de derivate posibile este maxim egal cu numărul de stări ale automatului și deci este un număr finit.

Reciproca afirmației anterioare este și ea adevărată: dacă un limbaj are un număr finit de derivate distincte, atunci el este un limbaj regular. Această afirmație este numită Teorema Myhill-Nerode à la Brzozowski.

În continuare se va exemplifica modul în care această teoremă poate fi utilizată pentru a arăta că un limbaj dat nu este regular.

Fie limbajul:

$L = \{ a^i b^i \mid \forall i \in \mathbb{N} \}$.

Să se arate că L nu este un limbaj regular.

DEMONSTRAȚIE: Pentru a arăta că limbajul nu este regular, se va arăta că numărul de derivate distincte ale limbajului este infinit. De exemplu, se va deriva limbajul dat în raport cu a^n. Se va obține:

$\frac{d}{da^n}\ a^{i+n}\ b^{i+n} = a^i\ b^{i+n}, \forall i, n \in \mathbb{N}$.

Depinzând de n, este evident că numărul de derivate distincte obținute este infinit. Prin urmare, limbajul nu este unul regular. □

Aplicând aceeași logică și în cazul limbajului:

$L = \{ a^i b \mid \forall i \in \mathbb{N} \}$.

vom obține că:

$\frac{d}{da^n}\ a^{i+n}b = a^i b$

Numărul de derivate este tot infinit, însă ele nu sunt distincte. Prin urmare limbajul este unul regular.

Vom exemplifica utilizarea teoremei și pentru limbajul L folosit în secțiunea anterioară pentru a arăta că reciproca lemei pompării nu este adevărată.

Fie limbajul:

$L = \{ uvwxy : u,y \in \{0,1,2,3\}^* ; v, w, x \in \{0,1,2,3\} \wedge (v = w \vee v=x \vee x=w) \} \cup \{ w \in \{0,1,2,3\}^* \mid (n_{\{0,1,2\}}(w) = 6 * n_{\{3\}}(w) \}$.

Să se demonstreze că acest limbaj nu este un limbaj regular.

DEMONSTRAȚIE: Notând cu C a doua submulțime a propozițiilor din limbajul L, vom calcula derivata:

$\frac{d}{d(0123)^i}C = \{w \in \{0123\}^* \mid n_{\{0,1,2\}}((0123)^i w) = 6 * n_{\{3\}}((0123)^i w) \} =$

$\{w \in \{0123\}^* \mid n_{\{0,1,2\}}((0123)^i) + n_{\{0,1,2\}}(w) = 6 * (n_{\{3\}}((0123)^i) + n_{\{3\}}(w))\} =$

$\{w \in \{0123\}^* \mid 3 * i + n_{\{0,1,2\}}(w) = 6 * i + 6 * n_{\{3\}}(w) \} =$

$\{w \in \{0123\}^* \mid n_{\{0,1,2\}}(w) = 3 * i + 6 * n_{\{3\}}(w) \}$.

Aceste limbaje sunt distincte, deoarece, spre exemplu, pentru fiecare valoare a lui i derivata $\frac{d}{d(0123)^i}C$ conține șirul $(012)^i$, însă niciuna dintre celelate derivate nu conține acest șir. Prin urmare, L are un număr infinit de derivate distincte și deci nu este un limbaj regular. □

3.3.4 *Exerciţii*

1. Să se definească expresiile regulate care să permită recunoaşterea următorilor atomi lexicali, pentru limbajul C# şi să se verifice folosind http://regexr.com/:
   - ★ variabilele:

     ```
     ([A-Za-z][A-Za-z_0-9]*)|([_][A-Za-z_0-9]+)
     ```

   - ★ comentariile de o singură linie:

     ```
     (\/\/).*
     ```

   - ★ comentariile de mai multe linii:

     ```
     \/\*([^*]|[\r\n]|(\*+([^*/]|[\r\n])))*\*+\/
     ```

   - ★ constantele de tip numere întregi:

     ```
     [+-]?[0-9]+
     ```

   - ★ constantele de tip numere reale:

     ```
     [+-]?[0-9]+[.][0-9]+
     ```

   - ★ constantele de tip caracter:

     ```
     '(.|\\n|\\t)'
     ```

   - ★ constantele de tip şir de caractere:

     ```
     \"((\\\")|[^\"])*\"
     ```

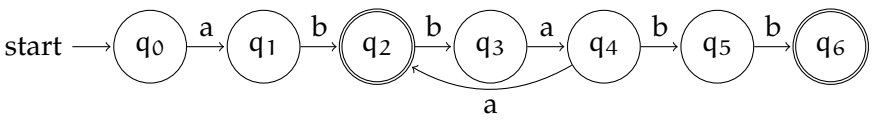
2. Să se construiască expresia regulată şi automatul finit determinist echivalente cu gramatica regulată:
   - ★ $\Sigma = \{a, b\}$
   - ★ $V_N = \{S, A, B\}$
   - ★ P={ $S \to abA$, $A \to baB$, $B \to aA \mid bb$ }

   Propoziţiile din limbajul generat de această gramatică, vor avea forma următoarele forme (de la cea mai simplă, până la cea mai complexă):
   - ★ ab**ba**bb
   - ★ ab**baaba**bb
   - ★ ab**baabaaba**bb
   - ★ ab**baaba...aba**bb

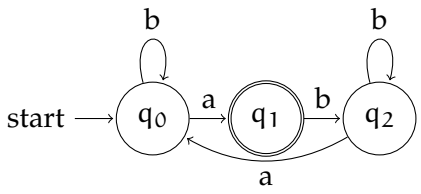
   Rezultă deci că expresia regulată echivalentă este:

   ```
   abba[aba]*bb
   ```

   Automatul finit echivalent este:



3. Să se construiască expresia regulată şi gramatica echivalente cu automatul finit determinist:



Propoziţiile din limbajul generat de această gramatică, vor avea forma următoarele forme (de la cea mai simplă, până la cea mai complexă):

- ★ a
- ★ **b** a
- ★ **bb** a
- ★ **b**...**b** a
- ★ **b**...**b** a b b...b a **b**...**b** a
- ★ **b**...**b** a b b...b a **b**...**b** a b b...b a **b**...**b** a
- ★ ...

Rezultă deci că expresia regulată echivalentă este:

`b* a (b + a b* a)*`

Gramatica regulată echivalentă este:

- ★ $\Sigma = \{a, b\}$
- ★ $V_N = \{S, A, B\}$
- ★ P={ $S \to bS \mid aA \mid a$, $A \to bB$, $B \to bB \mid aS$ }

4. Să se genereze automatul ϵ care verifică expresia regulată: o?[1-3]+ | (01)+ .

   o? se înlocuieşte cu o | ϵ.

   Atunci expresia regulată devine: ((o | ϵ)[1-3]+) | (01)+

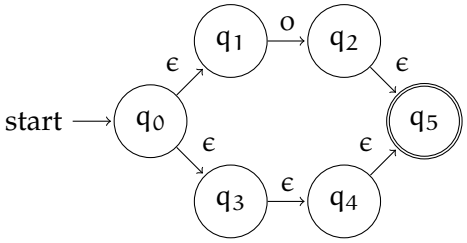


Figura 32: Automatul pentru expresia: o | ϵ

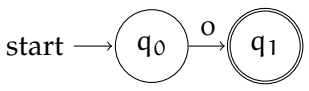


Figura 33: Automatul pentru o

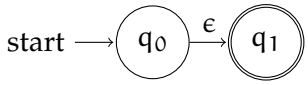


Figura 34: Automatul pentru ϵ

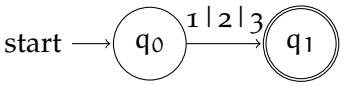


Figura 35: Automatul pentru [1-3]

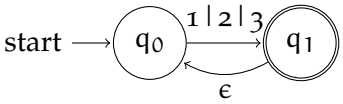


Figura 36: Automatul pentru [1-3]+

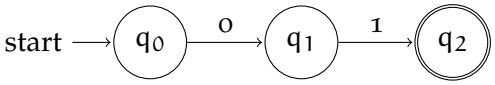


Figura 37: Automatul pentru (01)

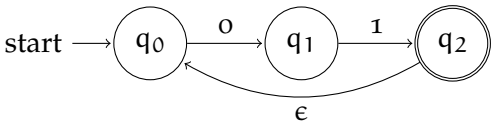


Figura 38: Automatul pentru (01)+

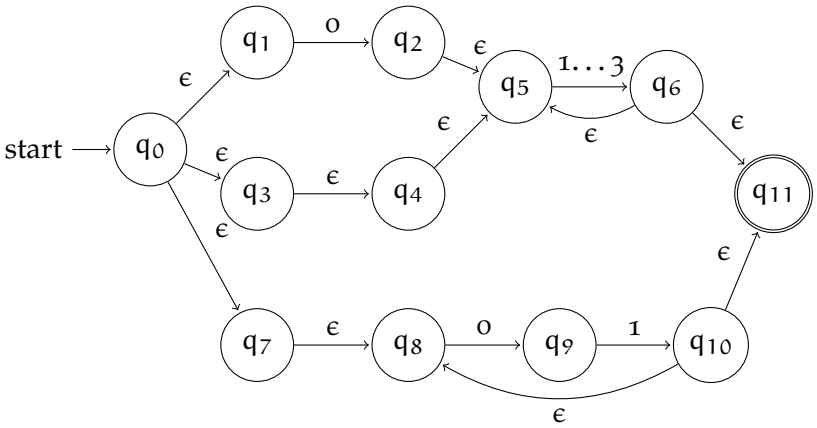


Figura 39: Automatul epsilon pentru 0?[1-3]+ | (01)+

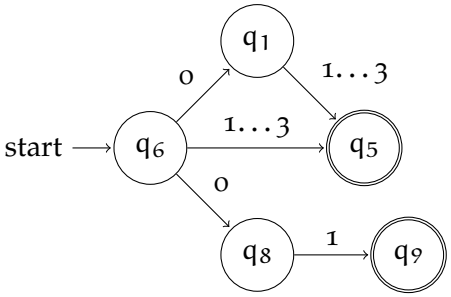


Figura 40: Automatul pentru o?[1-3]+|(01)+ fără stări ϵ

5. Fie expresia regulată: $0(00)^*(10)+$. Construiţi automatul finit epsilon care acceptă limbajul definit de această expresie regulată.

Pas 1: 0

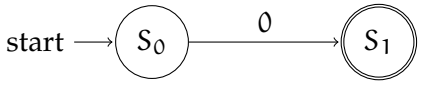


Figura 41: 0

Pas 2: $(00)^*$

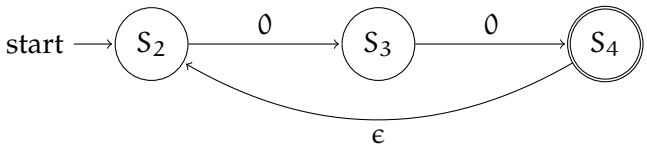


Figura 42: $(00)^*$

Pas 3: Contopim S1 cu S2 si obtinem:

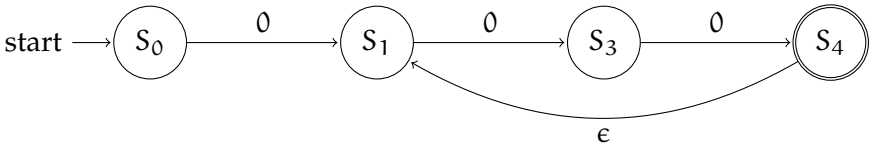


Figura 43: $0(00)^*$

Pas 4: $(10)^*$

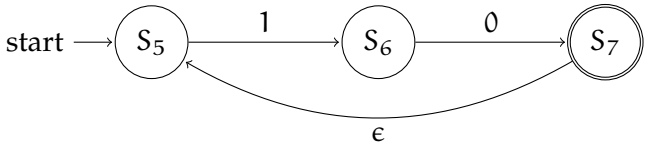


Figura 44: $(10)^*$

Pas 5: Contopim starile q_1 cu q_5.

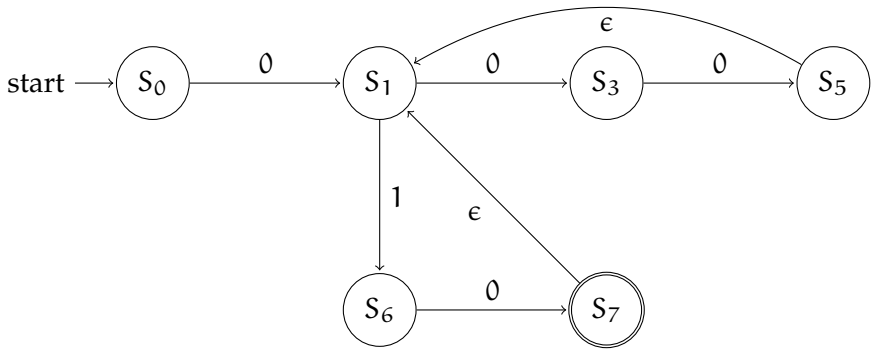


Figura 45: $0(00)^*(10)+$

Pas 6: $CL(S_0) = \emptyset$

$CL(S_1) = \emptyset$

$CL(S_3) = \emptyset$

$CL(S_5) = \{S_1\}$

$CL(S_6) = \emptyset$

$CL(S_7) = \{S_1\}$

Dupa ce eliminam tranzitiile ϵ si starile inutile vom obtine:

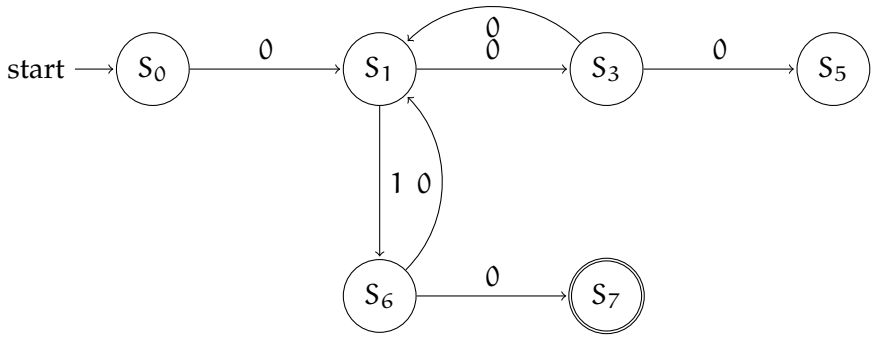


Figura 46: $0(00)^*(10)+$

6. Fie expresia regulată: $(10|[0-2]^*|(01)?)^+$. Construiţi automatul finit epsilon care acceptă limbajul definit de această expresie regulată.

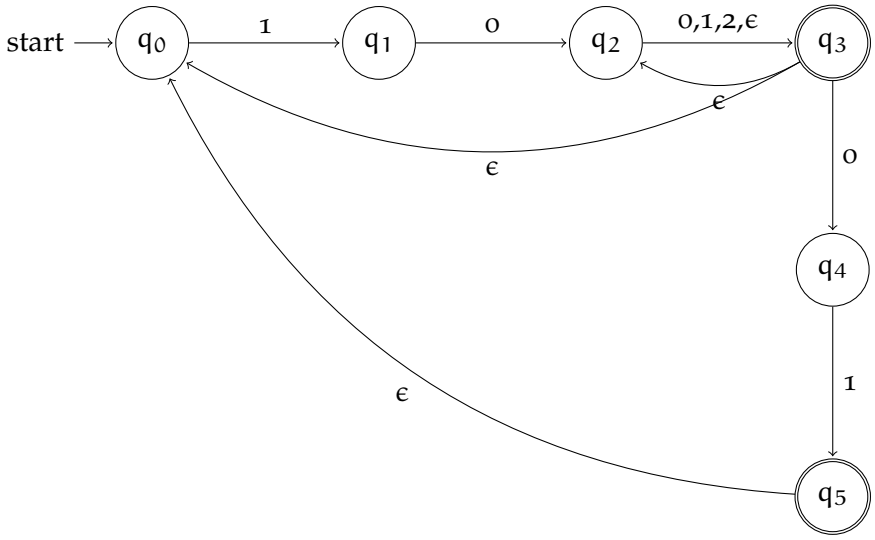
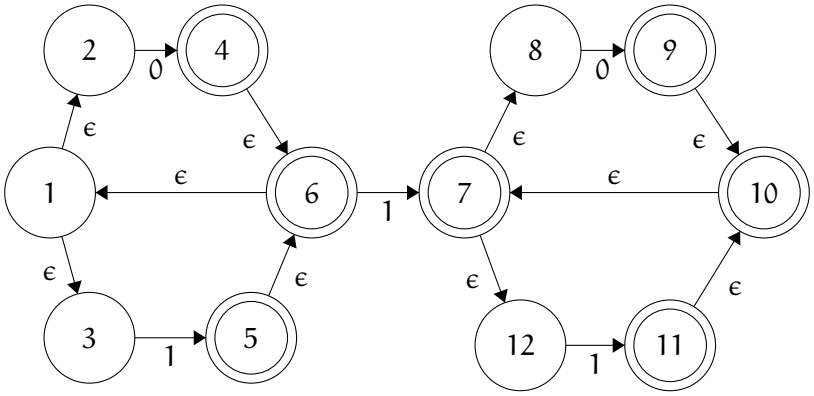


Figura 47: Automat finit ϵ

7. Fie expresia regulată: (0 ∪ 1)*1(0 ∪ 1)*. Construiţi automatul finit epsilon care acceptă limbajul definit de această expresie regulată.



8. Fie expresia regulată: $0 * (1|000*) * 0*$. Construiţi automatul finit epsilon care acceptă limbajul definit de această expresie regulată.

   Se fac automate finite pentru bucati din expresia regulata : 0* , 1 , 000* . In urma operatiilor de concatenare , reuniune intre automatele mentionate anterior se obtine automatul finit epsilon sub forma finala schitat mai jos.

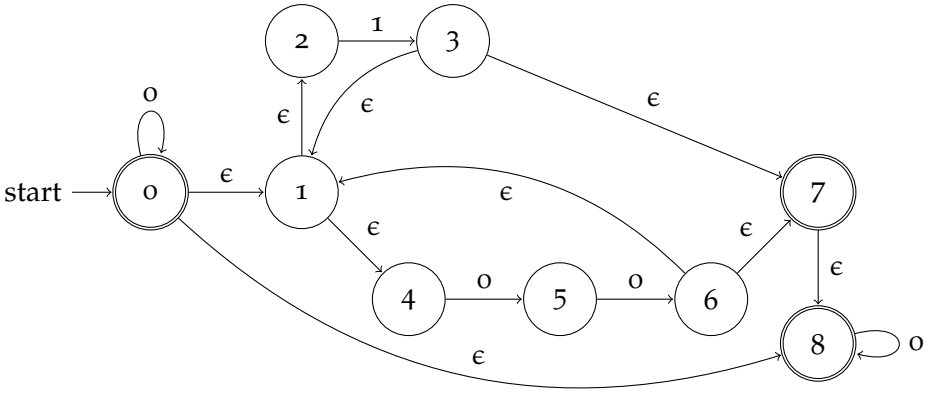
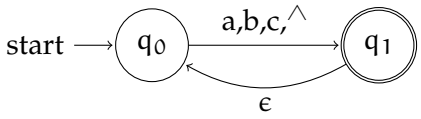


Figura 48: Automat finit epsilon ex.6

9. Fie expresia regulată: $[a-c^{\wedge}]+ \mid (abc)? \mid \epsilon$. Construiţi automatul finit epsilon care acceptă limbajul definit de această expresie regulată.

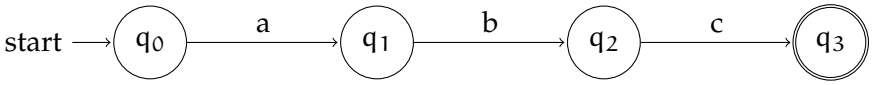
   ★ $[a-c^{\wedge}]+ \equiv (a \mid b \mid c \mid {}^{\wedge})+$

   Automat care accepta $[a-c^{\wedge}]+$:

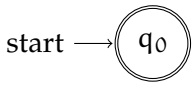


   ★ $(abc)? \equiv abc \mid \epsilon$

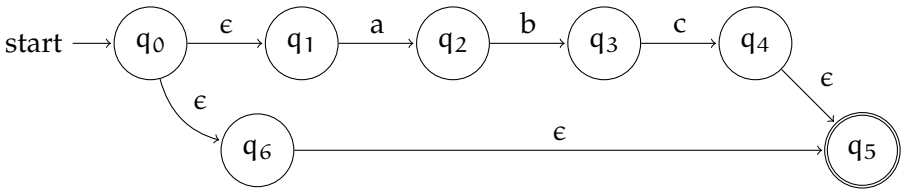
   Automat care accepta abc:



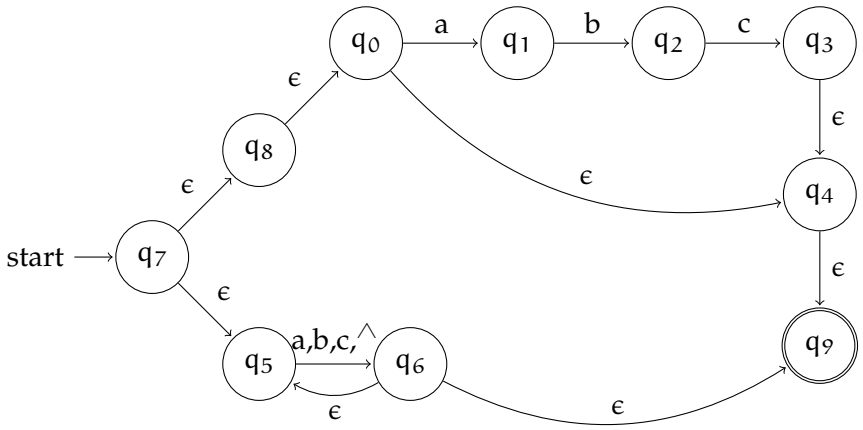
   Automat care accepta ϵ:



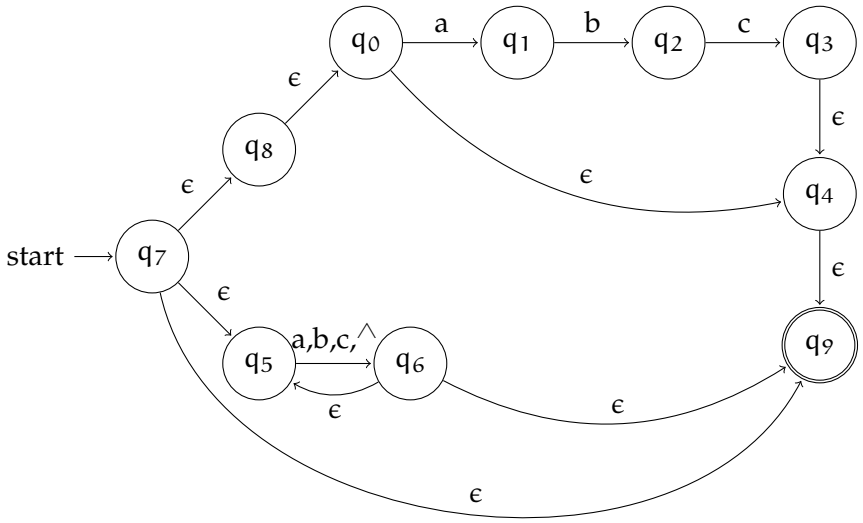
   Automat care accepta $abc \mid \epsilon$



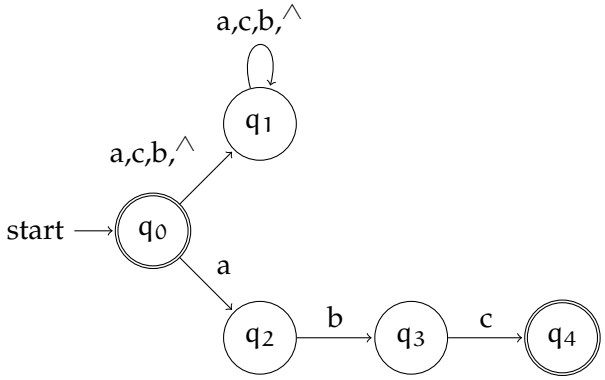
   ★ Concatenarea $[a-c^{\wedge}]+ \mid (abc)?$



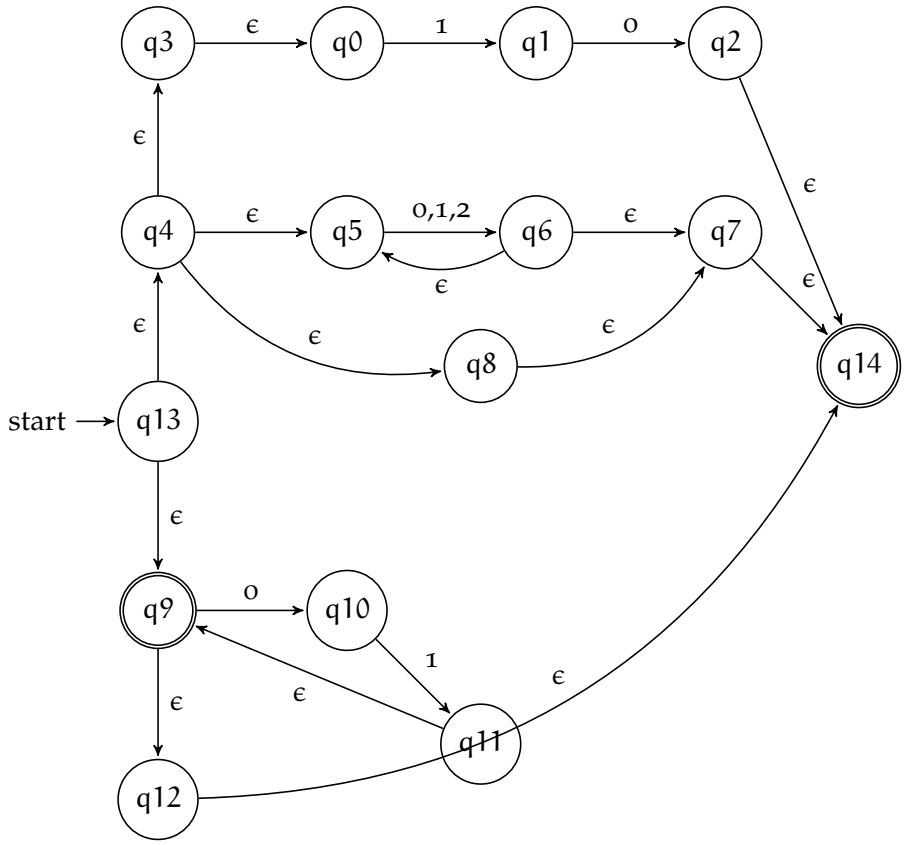
★ Concatenarea $[a-c^{\wedge}]+ \mid (abc)? \mid \epsilon$



Eliminand tranzitiile ϵ vom obtine automatul determinist cautat.



10. Fie expresia regulată: $(10|[0-2]*|(01)?)+$. Construiţi automatul finit epsilon care acceptă limbajul definit de această expresie regulată.



11. Să se demonstreze folosind teorema pompării că limbajul

    $L = \{ a^i b^j c^k , i+j=k \}$

    nu este un limbaj regulat.

    $w=a^N b^N c^{2N}$

    $1 \leqslant |y| \leqslant N, y \neq \epsilon$

    $y = a^i, 1 \leqslant y \leqslant N$

    $|xy| \leqslant N, x=a^j$

    $i+j \leqslant N$

    $w=a^j a^i (a^{N-(i+j)} b^N c^{2N})$

    pentru $q=2 \Rightarrow w'=a^{(N+i)} b^N c^{2N}$, $\forall$ i, $w' \notin L$.

    $\forall$ i, $N+i+N \neq 2N$, $1 \leqslant i \leqslant N$, $\Rightarrow$ limbajul este regulat.

12. Să se demonstreze folosind teorema pompării că limbajul

    Fie $L=\{ ab^n c^n \}$

    nu este un limbaj regulat.

    $N = ab^{n-1} c^N$

    $y = ab^i$, $1 \leqslant i \leqslant N$, $0 \leqslant j \leqslant N\text{-}1$.

    Cazul 1

    $x = \epsilon \Rightarrow y=ab^i$, $z = b^{(N-1-i)} c^{(N-1)}$

    $w' = (ab^i)^q (b^{(N-1-i)} c^{(N-1)})$

    pentru $q=0 : w'_0=b^{(N-1-i)} c^{(N-1)} \notin L$.

Cazul 2

$x = ab^j \Rightarrow y = b^k$, $z = b^{(N-1-j-k)}c^{(N-1)}$

$w' = ab^j(b^k)^q\ b^{(N-1-j-k)}c^{N-1}$

pentru q=0 : $w'_0 = ab^ib^{(N-1-j-k)}c^{(N-1)} \notin L$.

3.3.5 *Exerciţii propuse*

Aplicaţie facultativă

★ Implementarea într-un limbaj de programare oarecare a unei aplicaţii care să construiască AFD care acceptă acelaşi limbaj ca şi o expresie regulată dată.

★ Intrare: Expresia regulată.

★ Iesire:

- Elementele de definiţie ale AF-ϵ echivalent: alfabetul, mulţimea stărilor, starea iniţială, mulţimea stărilor finale, graful de tranziţie.
- Codul care implementează funcţionarea AFD echivalent pentru AF-ϵ de mai sus.

3.4 EXERCIŢII RECAPITULATIVE

1. Fie limbajul: *Toate succesiunile de caractere formate pe baza caracterelor a şi b care încep cu a, se termină cu b şi conţin subşirul abc.*
   a) Construiţi o gramatică care să definească acest limbaj.
   b) Având în vedere ierarhia Chomsky, precizaţi cărora dintre tipuri corespunde gramatica definită şi explicaţi de ce.
   c) Alegeţi o propoziţie care aparţine acestui limbaj şi o propoziţie care nu aparţine acestui limbaj. Scrieţi succesiunea de derivări directe care pot fi folosite pentru a genera propoziţiile pornind de la simbolul de start. Explicaţi.
   d) Construiţi arborii de derivare (arborele ascendent şi arborele descendent) pentru propoziţia de la punctul 3. Explicaţi.

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

   $V_N = \{N, Y, B\}$

   $\Sigma = \{a, b, c\}$

   $S = N$

   $P = \{$

   $(1,2,3,4)\ N \to aYBYb|aBYb|aYBb|aBb$

   $(5,6,7,8,9,10)\ Y \to aY|bY|cY|a|b|c$

   $(11)\ B \to abc$

   $\}$

   b) Gramatica este o gramatică de tipul al 2-lea (gramatică independentă de context), deoarece toate regulile de producţie din mulţimea P a gramaticii, au forma $A \to \alpha$, unde $A \in V_N$, iar $\alpha \in V^*$.

   c)
   - $aabaabcb \in L(G)$

     Derivare stânga: $N \Rightarrow^3 aYBb \Rightarrow^5 aaYBb \Rightarrow^6 aabYBb \Rightarrow^8 aabaBb \Rightarrow^{11} aabaabcb \Leftrightarrow aabaabcb \in L(G)$

     Derivare dreapta: $N \Rightarrow^3 aYBb \Rightarrow^{11} aYabcb \Rightarrow^5 aaYabcb \Rightarrow^6 aabYabcb \Rightarrow^8 aabaabcb \Leftrightarrow aabaabcb \in L(G)$
   - $ab \notin L(G)$

     $N \Rightarrow^4 aBb \Leftrightarrow ab \notin L(G)$

     Nu există nicio succesiune de derivări care să ducă la obţinerea propoziţiei ab.

   d) Arborele descendent este reprezentat în Figura 49. Arborele ascendent este reprezentat în Figura 50.

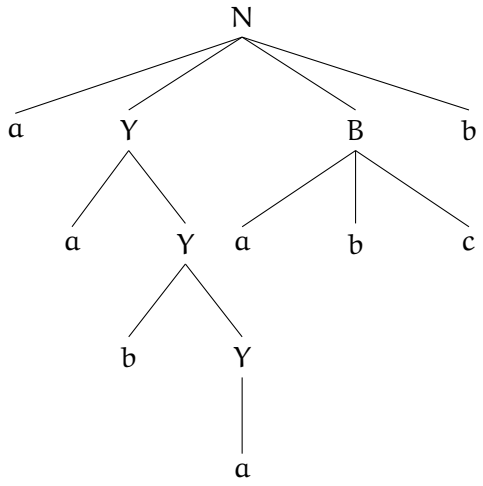


Figura 49: Arborele sintactic descendent

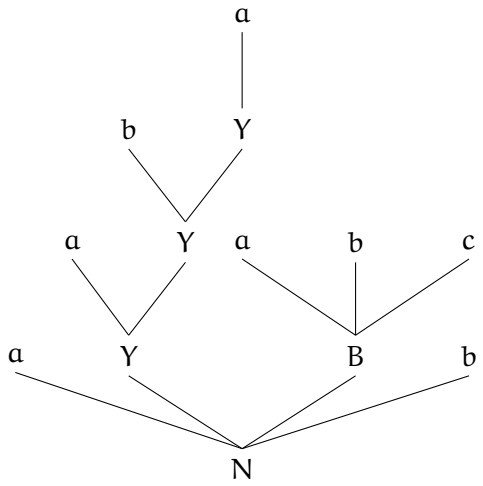


Figura 50: Arborele sintactic ascendent

2. Fie limbajul: *Toate succesiunile de caractere formate pe baza caracterelor a şi b care conţin subşirul ab sau subşirul ba*. Aceleaşi cerinţe ca şi la problema anterioară.

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

   $V_N = \{N, Y, Z\}$

   $\Sigma = \{a, b\}$

   $S = N$

   $P = \{$

   $(1, 2, 3)\ N \to aN|bN|Y$

   $(4, 5)\ Y \to abZ|baZ$

   $(6, 7, 8, 9)\ Z \to aZ|bZ|a|b$

   $\}$

   b) Gramatica este o gramatică de tipul al 3-lea (gramatică regulată), deoarece toate regulile de producţie din mulţimea P a gramaticii, au forma $A \to xB$ sau $A \to x$, unde $A, B \in V_N$, iar $x \in \Sigma^*$.

c) ★ $aabab \in L(G)$:

Derivare stanga:

$N \Rightarrow^1 aN \Rightarrow^1 aaN \Rightarrow^3 aaY \Rightarrow^5 aababZ \Rightarrow^9 aabab \Leftrightarrow aabab \in L(G)$

Derivare dreapta: Deoarece gramatica este o gramatică regulată, derivarea dreapta coincide cu derivarea stânga, adică au acelaşi număr de paşi (pentru a ajunge de la simbolul de start la propoziţie), iar formele propoziţionale intermediare sunt identice, pentru fiecare pas în parte.

aaa

Derivare stanga:

$N \Rightarrow aN \Rightarrow aaN \Rightarrow aaaN \Leftrightarrow aaa \notin L(G)$

Derivare dreapta: Aceeaşi observaţie ca şi mai sus.

d) Pentru propoziţia $aabab$, arborele descendent este ilustrat in Figura **??**, iar arborele ascendent este ilustrat in Figura **??**.

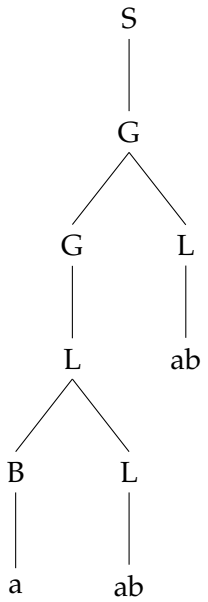


Figura 51: Arborele sintactic descendent

3. Fie limbajul: *Toate succesiunile de caractere formate pe baza caracterelor u şi v care conţin subşirurile uu şi vv, în această ordine*. Aceleaşi cerinţe ca şi la problema anterioară.

a) Gramatica generatoare pentru acest limbaj este:

★ $\Sigma = \{u, v\}$

★ S simbol de start

★ $P = \{S \to CuuCvvC$
$C \to uL_v|vL_v|vL_u|\epsilon$
$L_v \to v|vL_u|vL_v|\epsilon$
$L_u \to u|uL_v|\epsilon\}$

★ $V_N = \{S, C, L_u, L_v\}$

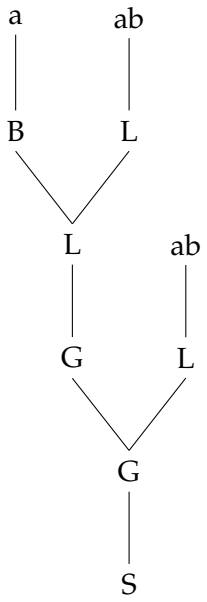


Figura 52: Arborele sintactic ascendent

b) Avand in vedere ierarhia Chomsky putem spune ca gramatica este una de tip 2, gramatica independenta de context.Dupa cum spune definitia aceste gramatici sunt definite de niste reguli de productie ce au in stanga un neterminal iar in dreapta o serie de terminale si neterminale.

c) ★ **uvvuvuuvuvv**
S $\rightarrow$ CuuCvv**C** $\rightarrow$ CuuCvvϵ $\rightarrow$ Cuu**C**vv $\rightarrow$ CuuvL$_u$vv $\rightarrow$ **C**uuvuvv $\rightarrow$ uL$_v$uuvuvv $\rightarrow$ uvL$_v$uuvuvv $\rightarrow$ uvvL$_u$uuvuvv $\rightarrow$ uvvuL$_v$uuvuvv $\rightarrow$ uvvuvuuvuvv
Avand in vedere ca am obtinut secventa ceruta urmand regulile de productie aceasta apartine limbajului.

★ **uvuvvuuv**
S
Avand in vedere ca nu putem forma secventa ceruta urmand regulile de productie aceasta este respinsa de limbaj.

a) Reprezentarea derivarilor la stanga si la dreapta sub forma de arbore ascendent respectiv descendent este:

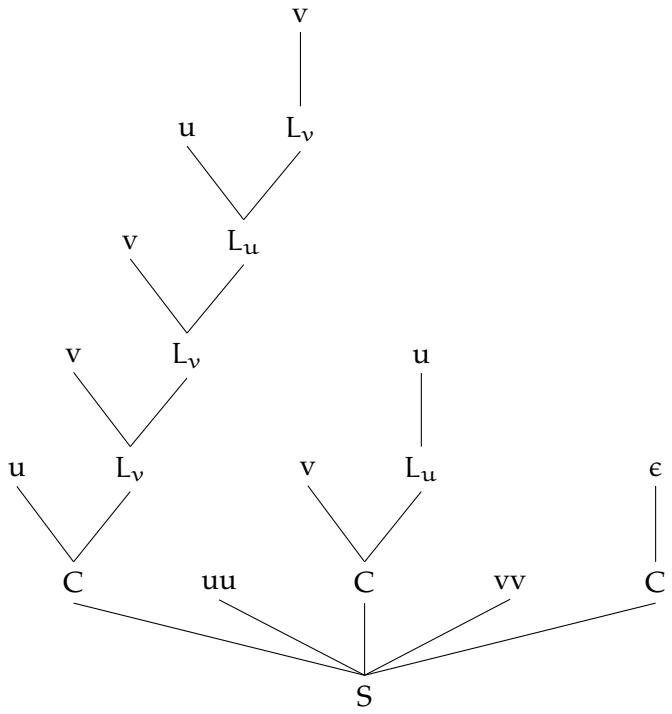


Figura 53: Arbore ascendent pentru derivare la stanga ex.1.4

Derivare la stanga inseamna ca mereu se schimba , conform unei reguli de productie, neterminalul cel mai din stanga.
Derivare la dreapta inseamna ca mereu se schimba tot conform unei reguli de productie insa neterminalul aflat cel mai in dreapta.
Arbore ascendent inseamna ca se pleaca de la secventa data si se fac operatiile inverse de la regulile de productie, nu urmam sensul sagetii ci facem invers.
Arbore descendent inseamna ca se pleaca de la simbolul de start si se urmaresc regulile de productie urmand ca pe nodurile frunza,citite de la stanga la dreapta, sa se construiasca secventa ceruta.

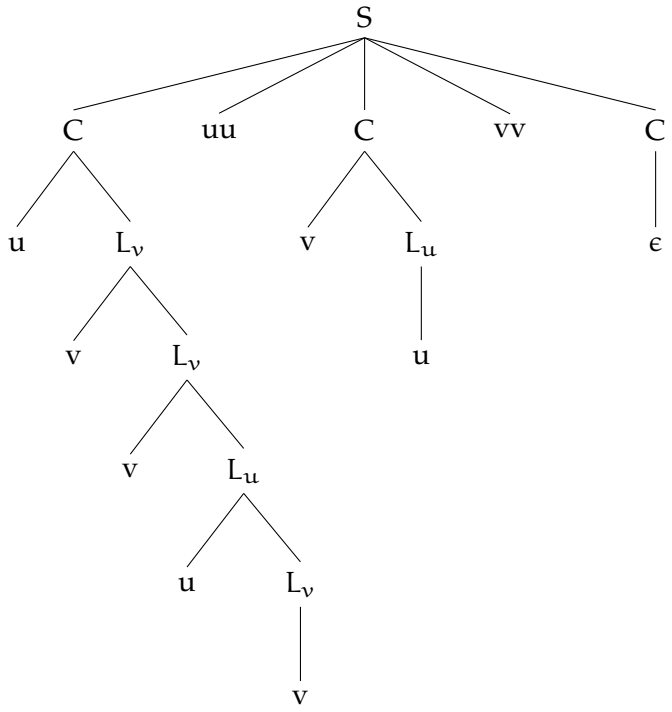


Figura 54: Arbore descendent pentru derivare la dreapta ex.1.4

4. Fie limbajul: *Toate succesiunile de caractere formate pe baza caracterelor u şi v care conţin unul sau mai multe* ***subşiruri*** *de tipul uuv.* Aceleaşi cerinţe ca şi la problema anterioară.

   a) Gramatica:

   - ★ $\Sigma = \{u, v\}$;
   - ★ $V_N = \{S, G, Lc, C\}$;
   - ★ $S \to LcGLc \mid GLc \mid LcG \mid G$;
   - ★ $G \to Guuv \mid uuv$;
   - ★ $Lc \to CLc \mid C$;
   - ★ $C \to u \mid v$;

   b) Gramatica definita este o gramatica de tipul 3 deoarece este o gramatica regulata in care se aplica succesiv derivari stanga sau dreapta.Si este o gramatica finita deoarece este o reprzentare finita.

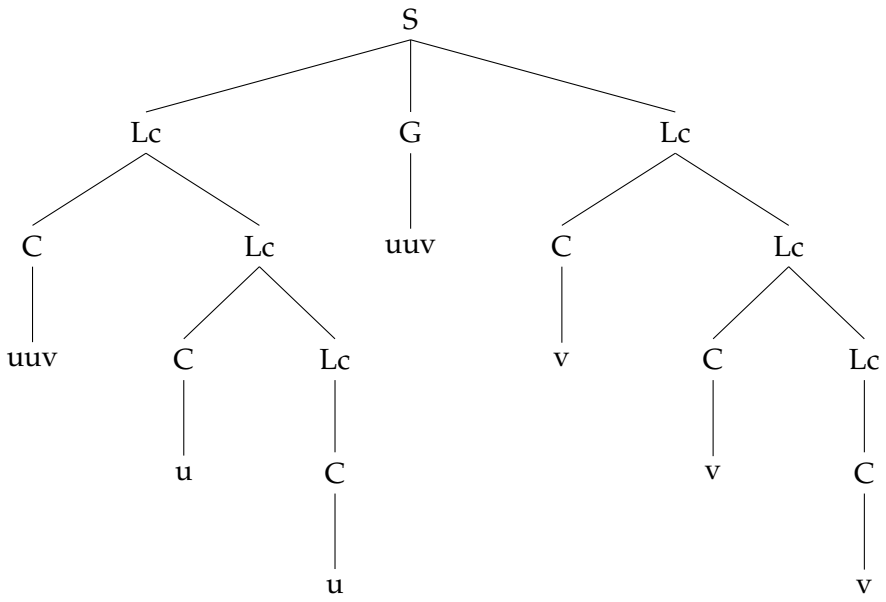
   c) Următorul şir de derivări, care reprezintă generarea propoziţiei *uuuuuvvvv*, este format din derivări stânga.

   $S \Rightarrow LcGLc \Rightarrow CLcGLc \Rightarrow uLcGLc \Rightarrow uCLcGLc \Rightarrow uuLcGLc \Rightarrow uuCGLc \Rightarrow uuuGLc \Rightarrow uuuuuvLc \Rightarrow uuuuuvCLc \Rightarrow uuuuuvvLc \Rightarrow uuuuuvvCLc \Rightarrow uuuuuvvvLc \Rightarrow uuuuuvvvC \Rightarrow uuuuuvvvv$

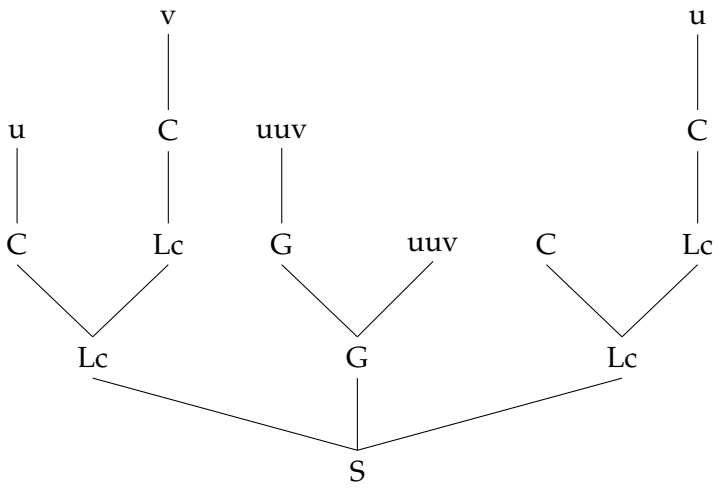
   Următorul şir de derivări, care reprezintă generarea propoziţiei *uvuuvuuvvu*, este format din derivări dreapta.

   $S \Rightarrow LcGLc \Rightarrow LcGCLc \Rightarrow LcGCC \Rightarrow LcGuuvCC \Rightarrow CLcGuuvCC \Rightarrow CCGuuvCC \Rightarrow CCGuuvCu \Rightarrow CCGuuvvu \Rightarrow CCuuvuuvvu \Rightarrow Cvuuvuuvvu \Rightarrow uvuuvuuvvu$

   d) **Arborele descendent** pentru derivare stanga→pornim de la nodul de start si obtinem propozitia.



Arborele ascendent pentru derivare dreapta→pornim de la frunze si obtinem nodul de start.



5. Fie limbajul: *Toate succesiunile de caractere formate pe baza caracterelor o şi 1 care încep cu cel puţin un 1 după care urmează doi de o şi apoi o succesiune oarecare de o şi 1.* Aceleaşi cerinţe ca şi la problema anterioară.

   a) Alfabetul limbajului (multimea terminalelor): $\Sigma = \{0, 1\}$
   Multimea neterminalelor: $V_N = \{N, L_A, L_B, A, B, Z\}$
   Simbolul de start: $S = N$
   Multimea regulilor de producie:

      i. $N \to AL_AZL_B$

      ii. $L_A \to AL_A \mid \epsilon$

      iii. $A \to 1$

      iv. $L_B \to BL_B \mid \epsilon$

v. $B \to 0 \mid 1$

vi. $Z \to 00$

Verificare: subpunctul 3

b) Gramatica este de tipul 2, deoarece regulile sale sunt de tipul $x \to r$ cu x neterminal si r un sir de terminali si neterminali ($x \in V_N, r \in V^*$). Aceasta este o gramatica libera, o gramatica independenta de context.

c) ★ $11100010 \in$ limbajului

$$N \overset{(a)}{\Longrightarrow} AL_AZL_B \overset{(c),(b.1)}{\underset{(f),(d.1)}{\Longrightarrow}} 1AL_A00BL_B \overset{(e)}{\underset{(c)}{\Longrightarrow}} 11L_A000L_B \overset{(b.1)}{\underset{(d.1)}{\Longrightarrow}} 11AL_A000BL_B \overset{(c)}{\underset{(e),(d.1)}{\Longrightarrow}}$$
$$111L_A0001BL_B \overset{(b.2)}{\underset{(e)}{\Longrightarrow}} 11100010L_B \overset{(d.2)}{\Longrightarrow} 11100010 \quad \checkmark$$

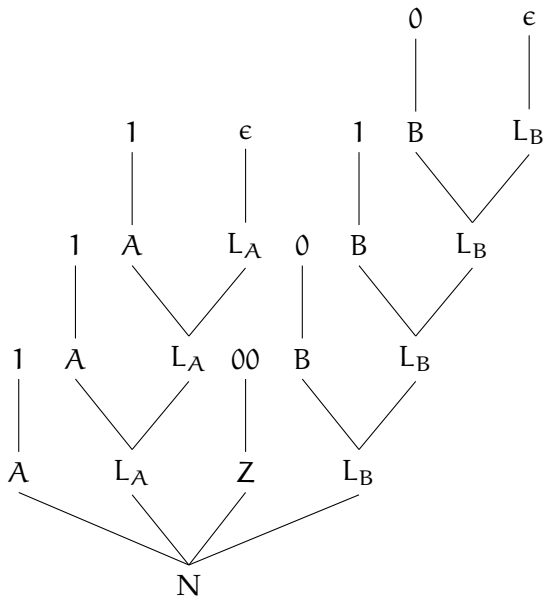
★ $100 \in$ limbajului

$$N \overset{(a)}{\Longrightarrow} AL_AZL_B \overset{(f)}{\underset{(c)}{\Longrightarrow}} 1L_A00L_B \overset{(b.2)}{\underset{(d.2)}{\Longrightarrow}} 100 \quad \checkmark$$

★ 001

$$N \overset{(a)}{\Longrightarrow} AL_AZL_B \overset{(c)}{\underset{(f)}{\Longrightarrow}} 1L_A00L_B \quad \checkmark \text{ nu se poate genera } 001 \notin \text{ limbajului}$$

d) Arborele ascendent pentru derivarea stânga (11100010)

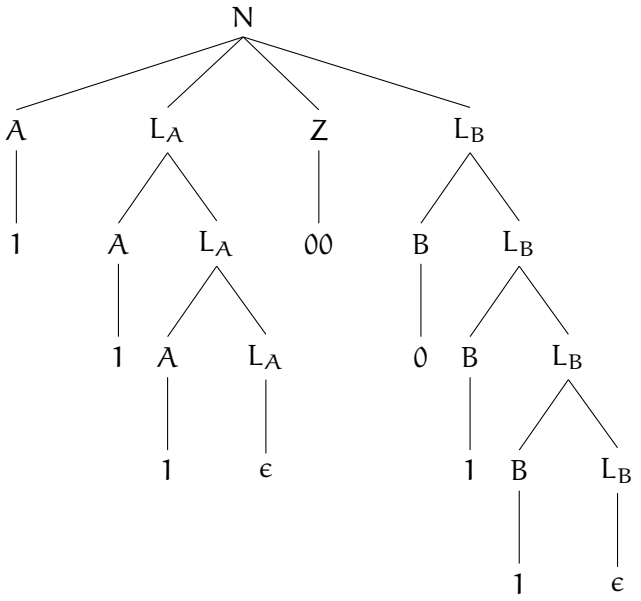


Arborele ascendent se formeaza pornind de la frunze(propozitie) catre radacina şi identificând regulile de productie aplicate. Astfel se ajunge la simbolul de start.

Arborele descendent pentru derivarea dreapta 11100010:

$$N \overset{(a)}{\Longrightarrow} AL_AZL_B \overset{(b.1)}{\underset{(d.1)}{\Longrightarrow}} AAL_AZBL_B \overset{(b.1)}{\underset{(d.1)}{\Longrightarrow}} AAAL_AZBBL_B \overset{(b)}{\underset{(d.1)}{\Longrightarrow}} AAAZBBBL_B \overset{(d.2)}{\Longrightarrow}$$
$$AAAZBBB \overset{(c),(f)}{\underset{(e)}{\Longrightarrow}} 11100010$$

Arborele descendent se formeaza pornind de la radacina şi aplicand succesiv regulile de productie, frunze continand fiecare un element al limbajului. Astfel, putem citi de la stanga la dreapta frunzele arborelui pentru a reface propozitia de la care am plecat.



6. Fie limbajul: *Toate succesiunile de caractere formate pe baza caracterelor a şi b care se termină cu a urmat de oricâţi de b şi apoi de un a.* Aceleaşi cerinţe ca şi la problema anterioară.

   a) $\Sigma = \{a, b\}$
   N $\rightarrow$ AaBa
   A $\rightarrow$ aA | bA | a | b
   B$\rightarrow$ bB | b

   b) Gramatica definita este independenta de context deoarece este reprezentata ca reguli de forma A $\rightarrow \gamma$ cu A neterminal si γ un sir de terminali si neterminali. Aceste limbaje sunt exact toate limbajele care poti fi recunoscute de un automat cu stiva nedeterminist. Limbajele independente de context constituie baza teoretica a majoritatii limbajelor de programare.

   c) Apartine : ababa N $\rightarrow$ AaBa $\rightarrow$ aAaBa $\rightarrow$aAaBa $\rightarrow$ abaBa $\rightarrow$ ababa

      Nu apartine : ababaa

      N $\rightarrow$ AaBa $\rightarrow$ aAaBa $\rightarrow$aAaBa $\rightarrow$ abaBa $\rightarrow$ ababa $\Rightarrow$ ababaa nu apartine acestui limbaj

   d) Construiţi arborii de derivare (arborele ascendent pentru derivarea stânga şi arborele descendent pentru derivarea dreapta) pentru propoziţiile de la punctul 3. Explicaţi.

7. Fie limbajul: *Toate succesiunile de simboluri formate pe baza simbolurilor a, b şi c care începe cu cel puţin un grup bc, conţine maxim un grup cba şi se încheie cu cel puţin doi de a.* Aceleaşi cerinţe ca şi la problema anterioară.

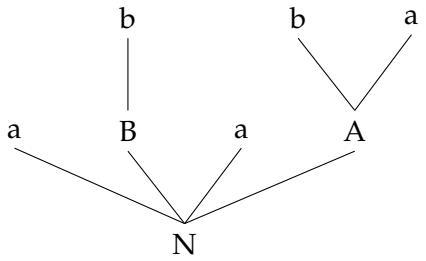


Figura 55: Arborele sintactic ascendent pentru propoziţia *ababa*

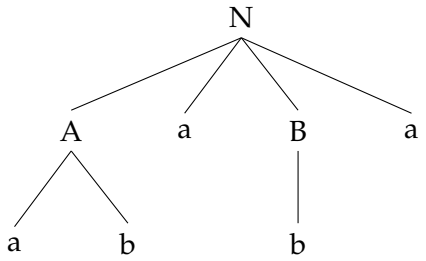


Figura 56: Arborele sintactic descendent pentru propoziţia *ababa*

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

$\Sigma = \{a, b, c\}$

$V_N = \{A, B, C, X\}$

$N \to AXBXC$

$A \to bc|bcA$

$B \to bca|\epsilon$

$C \to aaC|aa$

$X \to aX|bX|CY$

$Y \to aY|bM|cY|\epsilon$

$M \to aL|bM|cM|\epsilon$

$L \to bL|cL|\epsilon$

b) Gramatica este de tip 2, deoarece este independenta de context si poate fi recunoscuta de un automat cu stiva.

c)
- bcaa
  $N \Rightarrow AXBXC \Rightarrow bcXBXC \Rightarrow bc\epsilon BXC \Rightarrow bcBXC \Rightarrow bc\epsilon XC \Rightarrow bcXC \Rightarrow bc\epsilon C \Rightarrow bcC \Rightarrow bcaa \Rightarrow$ propozitia $\in$ limbajului
- bcca
  $N \Rightarrow AXBXC \Rightarrow bcXBXC \Rightarrow bccBXC \Rightarrow bcc\epsilon XC \Rightarrow bccaC \Rightarrow$ nu se pot face derivari pentru C astfel incat sa se potriveasca $\Rightarrow$ propozotia $\notin$ limbajului.

d) Vom deriva pentru propozitia bc cba aa
Derivare stanga:
$N \Rightarrow AXBXC \Rightarrow bcXBXC \Rightarrow bc\epsilon BXCbcBXC \Rightarrow bccbaXC \Rightarrow bccba\epsilon C \Rightarrow bccbaC \Rightarrow bccbaC \Rightarrow$ bc cba aa

Derivare dreapta:
$N \Rightarrow ABC \Rightarrow ABaa \Rightarrow Acbaaa \Rightarrow bccbaC \Rightarrow$ bc cba a

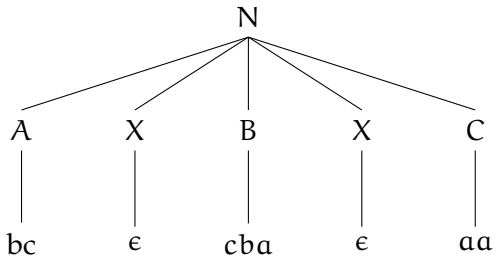


Figura 57: Arbore sintactic descendent derivare stanga

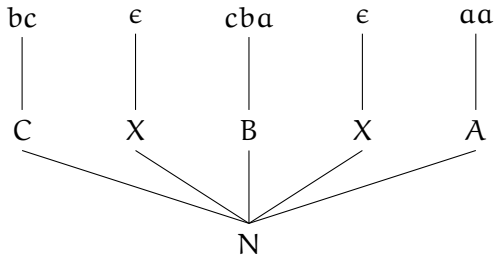


Figura 58: Arbore sintactic descendent derivare stanga

8. Aceleaşi cerinţe ca şi problema anterioară, pentru limbajul: *Toate succesiunile de simboluri formate pe baza simbolurilor a, b şi c care conţin cel puţin doi de b şi cel mult un c.*

   a) Gramatica:

   $\sum = \{a, b, c\}$

   Simbol de start:S

   Multimea de neterminale: $V_N = \{G, L, A, B, C, D\}$

   Reguli de productie:

   $S \Rightarrow G$

   $G \Rightarrow LBCL|LCBL|CLBL$

   $L \Rightarrow A|B|D|L|\epsilon$

   $B \Rightarrow Dbb|bbD$

   $D \Rightarrow bD|\epsilon$

   $A \Rightarrow aA|a$

   $C \Rightarrow c|\epsilon$

   b) Conform ierarhiei Chomsky, gramatica definita la punctul 1 este independenta de context, fiind o gramatica formala sau generativa care descrie structural un limbaj peste $\sum$, iar fiecare regula de productie $\alpha \longrightarrow \beta$ verifica $\alpha \in V - \sum$, pentru $\beta \in V^*$. Altfel spus, este de forma $A \longrightarrow u, A \in V_N$.

   c) Se considera exemplele:

   caabbbaa

   Prin derivare la stanga:

   $S \longrightarrow G \longrightarrow CLBL \longrightarrow cLBL \longrightarrow cABL \longrightarrow caABL \longrightarrow caaBL \longrightarrow caaDbbL \longrightarrow caabDbbL \longrightarrow caab\epsilon bbL \longrightarrow caabbbL \longrightarrow caabbbA \longrightarrow caabbbaA \longrightarrow caabbbaa$

   Prin derivare la dreapta:

$S \longrightarrow G \longrightarrow CLBL \longrightarrow CLBA \longrightarrow CLBaA \longrightarrow CLBaa \longrightarrow CLbbDaa \longrightarrow CLbbbDaa \longrightarrow CLbbb\epsilon aa \longrightarrow CLbbbaa \longrightarrow CAbbbaa \longrightarrow CaAbbbaa \longrightarrow Caabbbaa \longrightarrow caabbbaa$

aabcc

Derivare la stanga:

$S \longrightarrow G \longrightarrow LBCL \longrightarrow ABCL \longrightarrow aABCL \longrightarrow aaBCL \longrightarrow aaDbbCL$?? Propozitia nu apartine limbajului!

Derivare la dreapta:

$S \longrightarrow G \longrightarrow LBCL \longrightarrow LBC\epsilon \longrightarrow LBC \longrightarrow LBc \longrightarrow LDbbc$?? Propozitia nu apartine limbajului!

d) Pentru exemplul "caabbbaa" arborele descendent este ilustrat in Figura 1.3, iar cel ascendent in Figura 1.4

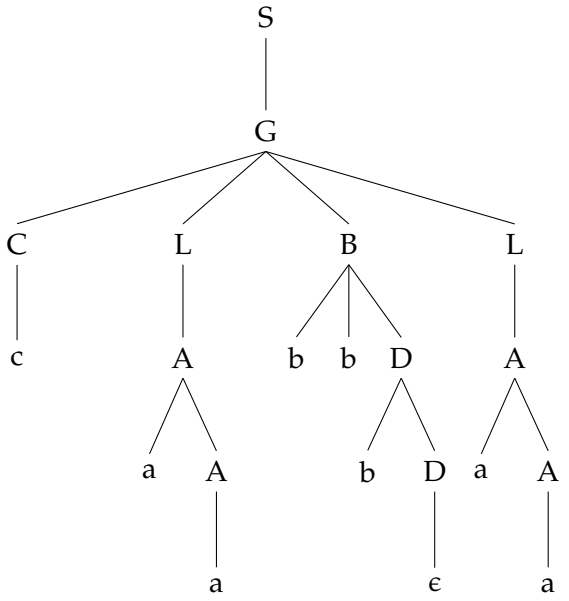


Figura 59: Arborele descendent

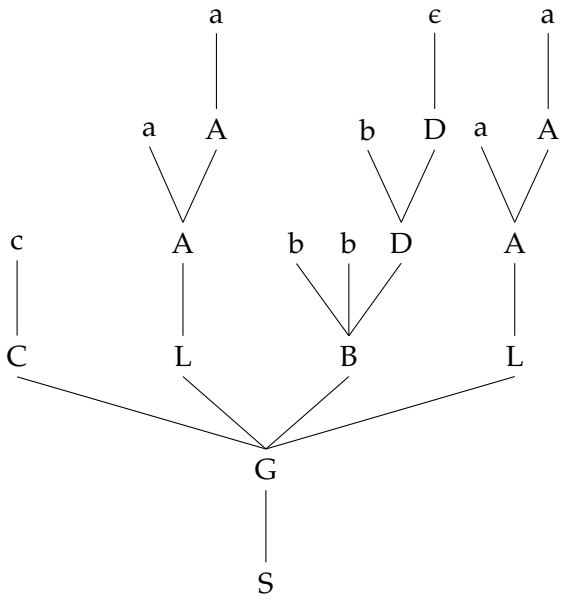


Figura 60: Arborele ascendent

9. Aceleaşi cerinţe ca şi la problema anterioară, pentru limbajul: *Toate succesiunile de simboluri formate pe baza simbolurilor a, b şi c care încep cu abc, conţin cel puţin un grup bb şi se termină cu a sau cu c.*

   a) ★ Σ = {a,b,c};
      ★ V_N = {S, M,N,B,F,Lc};
      ★ S $\rightarrow$ MNF;
      ★ N $\rightarrow$ LcBLc | LcB | BLc | B ;
      ★ Lc $\rightarrow$ FLc | F;
      ★ M $\rightarrow$ abc;
      ★ B $\rightarrow$ bb;
      ★ F $\rightarrow$ a | c;

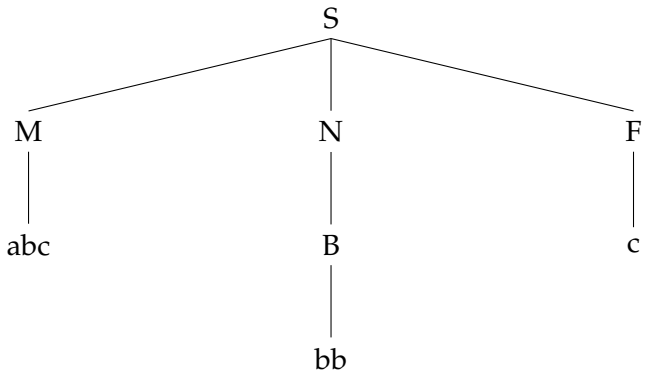
   b) Gramatica definita este o gramatica de tipul 3 deoarece este o gramatica regulata in care se aplica succesiv derivari stanga sau dreapta.Si este o gramatica finita deoarece este o reprzentare finita.

   c) abcbbc Derivare stanga: S$\rightarrow$ MNF $\rightarrow$ abcNF $\rightarrow$ abcBF $\rightarrow$ abcbbF $\rightarrow$ abcbbc (am consumat toate simbolurile => propozitia apartine limbajului)

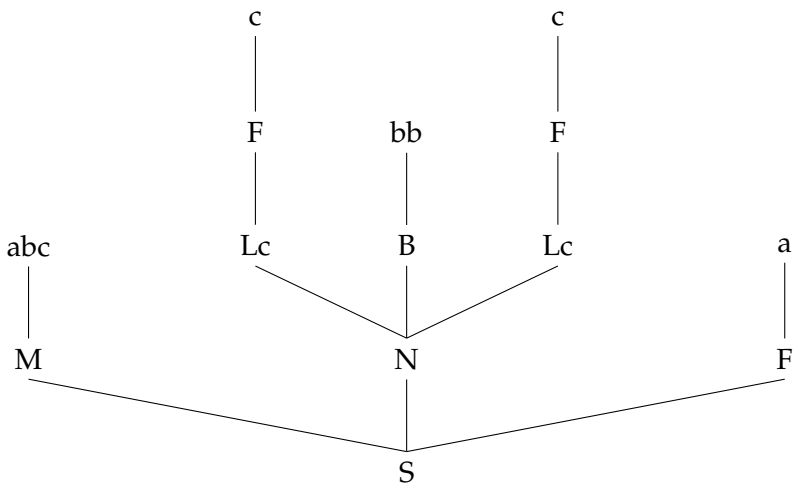
      Exemplul12: (abccbbca)

      Derivare dreapta: S$\rightarrow$ MNF $\rightarrow$ MLcBLcF $\rightarrow$ M $\rightarrow$ MLcBFF $\rightarrow$ MFBFF $\rightarrow$ abccbbca (am consumat toate simbolurile => propozitia apartine limbajului)

   d) **Arborele descendent** pentru derivare stanga$\rightarrow$pornim de la nodul de start si obtinem propozitia.



Arborele ascendent pentru derivare dreapta→pornim de la frunze si obtinem nodul de start.



10. Aceleaşi cerinţe ca şi la problema anterioară, pentru limbajul: *Toate succesiunile de simboluri formate pe baza simbolurilor a, b şi c care au lungimea cel puţin egală cu 3, iar al treilea simbol este a.*

    a) Gramatica generatoare pentru acest limbaj este:

    - ★ $\Sigma = \{a, b, c\}$
    - ★ S simbol de start
    - ★ $P = \{S \to C_o C_o aL$
      $C_o \to a|b|c$
      $L \to CL|C$
      $C \to a|b|c|\epsilon\}$
    - ★ $V_N = \{S, C_o, L, C\}$

    b) Avand in vedere ierarhia Chomsky putem spune ca gramatica este una de tip 2, gramatica independenta de context.Dupa cum spune definitia aceste gramatici sunt definite de niste reguli de productie ce au in stanga un neterminal iar in dreapta o serie de terminale si neterminale.

c) ★ **abaabc**
S → $C_o C_o$aL → aC_oaL → aba**L** → aba**C**L → abaa**L** → abaa**C**L → abaab**L** → abaab**C** → abaabc
Avand in vedere ca am obtinut secventa ceruta urmand regulile de productie aceasta apartine limbajului.

★ **ba**
S → C_oC)o¤L
Avand in vedere ca nu putem trece din C_o in ϵ secventa nu poate fi creata, cu alte cuvinte nu este acceptata de limbaj.

d) Reprezentarea derivarilor la stanga si la dreapta sub forma de arbore ascendent respectiv descendent este:
Explicatiile de la exercitiul anterior sunt valabile si pentru acesta.

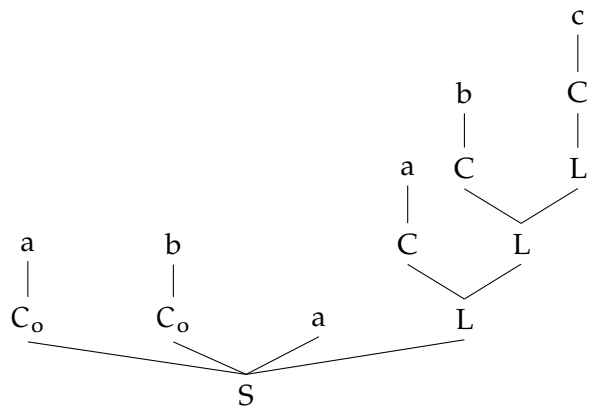


Figura 61: Arbore ascendent pentru derivare la stanga ex.2.4

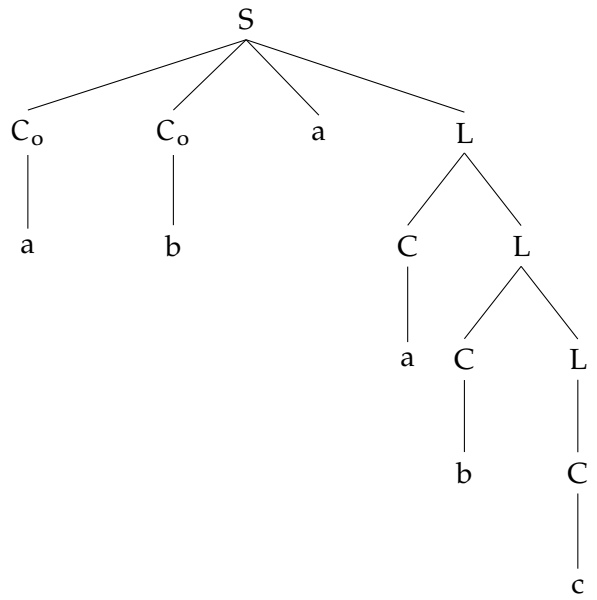


Figura 62: Arbore descendent pentru derivare la dreapta ex.2.4

11. Aceleaşi cerinţe ca şi la problema anterioară, pentru limbajul: *Toate succesiunile de simboluri formate pe baza simbolurilor a, b şi c care încep cu a sau cu b, conţin subşirurile ac şi bc şi se termină cu un c sau cu doi de c.*

a) $\Sigma = \{a, b, c\}$
$N \rightarrow ABC$
$A \rightarrow a \mid b$
$B \rightarrow D\,E\,D\,F\,D\,B \mid D\,E\,D\,F\,D \mid D\,F\,D\,E\,D\,B \mid D\,F\,D\,E\,D$
$C \rightarrow c \mid cc$
$D \rightarrow aD \mid bD \mid cD \mid a \mid b \mid c \mid \epsilon$
$E \rightarrow acE \mid ac$
$F \rightarrow bcF \mid bc$

b) Gramatica definita este independenta de context deoarece este reprezentata ca reguli de forma $A \rightarrow \gamma$ cu A neterminal si γ un sir de terminali si neterminali. Aceste limbaje sunt exact toate limbajele care poti fi recunoscute de un automat cu stiva nedeterminist. Limbajele independente de context constituie baza teoretica a majoritatii limbajelor de programare.

c) Apartine : abcacc
$N \rightarrow ABC \rightarrow aBC \rightarrow aDFDEDC \rightarrow aFDEDC \rightarrow abcDEDC \rightarrow abcEDC \rightarrow abcacDC \rightarrow abcacC \rightarrow abcacc \Rightarrow$ apartine acestui limbaj

Nu apartine : abcac

$N \rightarrow ABC \rightarrow aBC \rightarrow aDFDEDC \rightarrow aFDEDC \rightarrow abcDEDC \rightarrow abcEDC \rightarrow abcacDC \rightarrow abcacC \Rightarrow$ nu apartine acestui limbaj

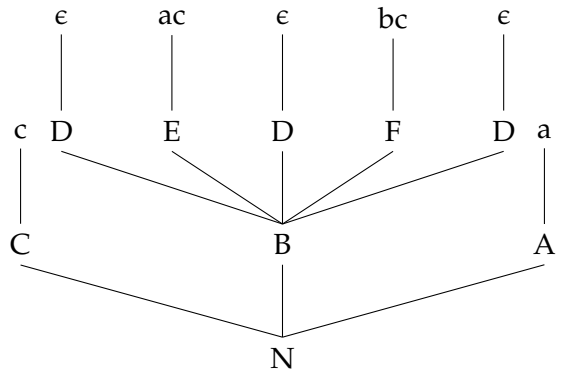


Figura 63: Arborele sintactic ascendent pentru propoziţia *abcacc*

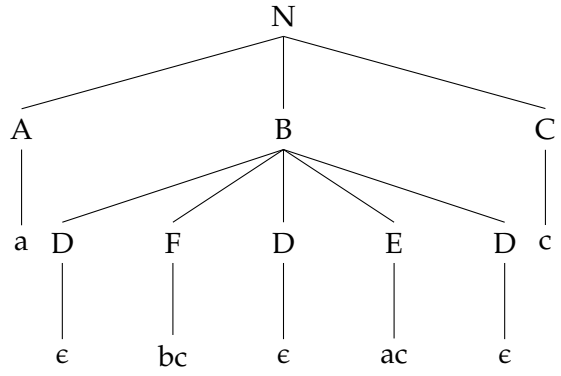
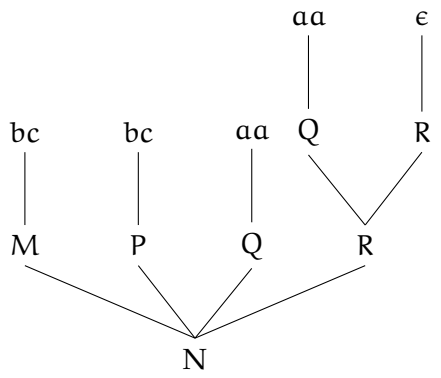


Figura 64: Arborele sintactic descendent pentru propoziţia *abcacc*



d)

12. Aceleaşi cerinţe ca şi la problema anterioară, pentru limbajul: *Toate succesiunile de simboluri formate pe baza simbolurilor a, b şi c care încep cu bc, conţin subşirul ac sau subşirul bc şi se termină cu cel puţin un grup aa.*

a) Alfabetul limbajului (multimea terminalelor): $\Sigma = \{a, b, c\}$
Multimea neterminalelor: $V_N = \{N, M, P, Q, R, S\}$
Simbolul de start: $S = N$
Multimea regulilor de producie:

i. $N \to MPQR$

ii. $M \to bc$

iii. $P \to ac \mid bc$

iv. $R \to QR \mid \epsilon$

v. $Q \to aa$

Verificare: subpunctul 3

b) Gramatica este de tipul 2, deoarece regulile sale sunt de tipul $x \to r$ cu x neterminal si r un sir de terminali si neterminali ($x \in V_N, r \in V^*$). Aceasta este o gramatica libera, o gramatica independenta de context.

c) ★ bcbcaaaa $\in$ limbajului
$N \overset{(a)}{\Longrightarrow} MPQR \underset{(e),(d)}{\overset{(c),(b)}{\Longrightarrow}} bcbcaaQR \underset{(d.2)}{\overset{(h)}{\Longrightarrow}} bcbcaaaa$ ✓

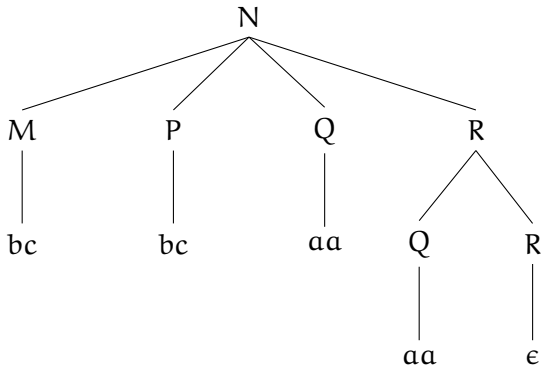
★ bcacaa $\in$ limbajului
$N \overset{(a)}{\Longrightarrow} MPQR \underset{(e),(d.2)}{\overset{(b),(c)}{\Longrightarrow}} bcacaa$ ✓

★ bcaaaabc $\notin$ limbajului
$N \overset{(a)}{\Longrightarrow} MPQR \underset{(e)}{\overset{(b}{\Longrightarrow}} bcPaa$ ✓ nu se poate genera 001 $\notin$ limbajului

d) Arborele ascendent pentru derivarea stânga (11100010)

Arborele ascendent se formeaza pornind de la frunze(propozitie) catre radacina şi identificând regulile de productie aplicate. Astfel se ajunge la simbolul de start.

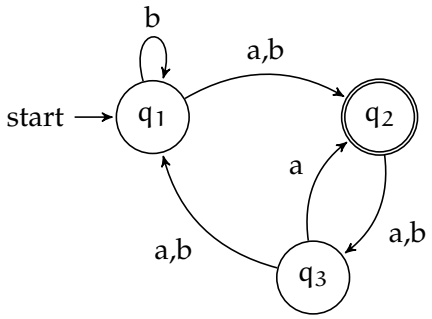
Arborele descendent pentru derivarea dreapta bcbcaaaa:



$$N \overset{(a)}{\Longrightarrow} MPQR \overset{(d.1)}{\Longrightarrow} MPQQR \overset{(d.2)}{\Longrightarrow} MPQQ \overset{\substack{(b),(c)\\(e)}}{\Longrightarrow} bcbcaaaa$$

Arborele descendent se formeaza pornind de la radacina şi aplicand succesiv regulile de productie, frunze continand fiecare un element al limbajului. Astfel, putem citi de la stanga la dreapta frunzele arborelui pentru a reface propozitia de la care am plecat.

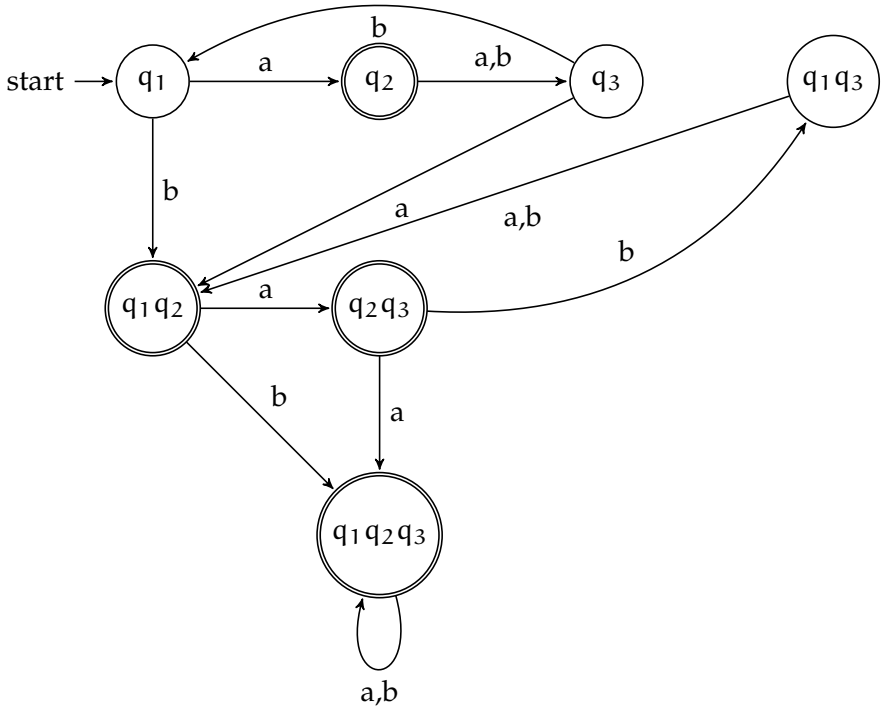
13. Fie automatul finit nedeterminist reprezentat prin graful de tranziţie:



f	a	b
$\varnothing$	-	-
q_1	$\{q_2\}$	$\{q_1 q_2\}$
$*q_2$	$\{q_3\}$	$\{q_3\}$
q_3	$\{q_1 q_2\}$	$\{q_1\}$
$*q_1 q_2$	$\{q_2 q_3\}$	$\{q_1 q_2 q_3\}$
$q_1 q_3$	$\{q_1 q_2\}$	$\{q_1 q_2\}$
$*q_2 q_3$	$\{q_1 q_2 q_3\}$	$\{q_1 q_3\}$
$*q_1 q_2 q_3$	$\{q_1 q_2 q_3\}$	$\{q_1 q_2 q_3\}$

1.Automatul finit determinist care accepta limbajul de mai sus.

2.**Exemplul 1**
(aabba,$\{q_1\}$)$\rightarrow$ (abba,$\{q_2\}$) $\rightarrow$ (bba,$\{q_3\}$) $\rightarrow$ (ba,$\{q_1\}$) $\rightarrow$ (a,$\{q_1 q_2\}$) $\rightarrow$ (ϵ, $\{q_2 q_3\}$)



⇒ propozitia este acceptata deoarece am consumat toate simbolurile si am ajuns intr-stare finala.
Exemplul2
(aabbab,{q_1})→ (abbab,{q_2}) → (bbab,{q_3}) → (bab,{q_1}) → (ab,{q_1q_2}) → (b, {q_2q_3}) → (ϵ,{q_1q_3})

⇒ propozitia nu este acceptata pentru ca, desi am consumat toate simbolurile starea q_1q_3 nu este stare finala.

14. Fie automatul finit nedeterminist reprezentat prin graful de tranziţie:

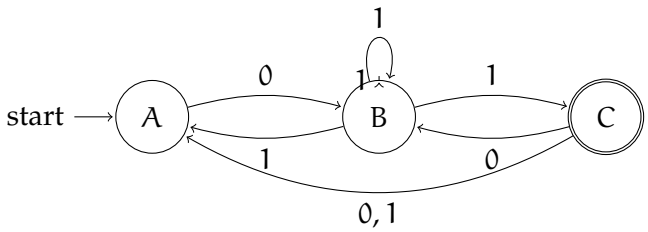


Figura 65: Graful de tranziţie

1. Deduceţi automatul finit determinist care acceptă acelaşi limbaj.

f′	0	1
A	{B}	-
*B	-	{A, B, C}
*C	{A, B}	{A}
*AB	{B}	{A, B, C}
AC	{A, B}	{A}
*BC	{A, B}	{A, B, C}
*ABC	{A, B}	{A, B, C}

Figura 66: Tabela de tranziţie

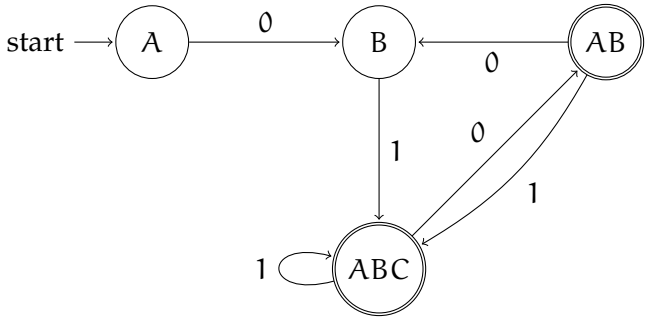


Figura 67: Graful de tranziţie

2. Alegeţi o propoziţie care aparţine acestui limbaj şi una care nu aparţine limbajului. Scrieţi pentru fiecare dintre ele succesiunea de relaţii de mişcare pentru automatul de la 1. Explicaţi.
   - 011
     $(011, A) \vdash (11, B) \vdash (1, ABC) \vdash (\epsilon, ABC) \Rightarrow$ propozitia $\in$ limbajului.
   - 0100
     $(0100, A) \vdash (100, B) \vdash (00, ABC) \vdash (0, AB) \vdash (\epsilon, B) \Rightarrow$ propozitia nu s-a terminat intr-o stare terminala $\Rightarrow$ propozitia nu a fost acceptata.

15. Fie automatul finit nedeterminist reprezentat prin graful de tranziţie din Figura 1.5.
    1. Deduceţi automatul finit determinist care acceptă acelaşi limbaj.
    2. Alegeţi o propoziţie care aparţine acestui limbaj şi una care nu aparţine limbajului. Scrieţi pentru fiecare dintre ele succesiunea de relaţii de mişcare pentru automatul de la 1. Explicaţi.

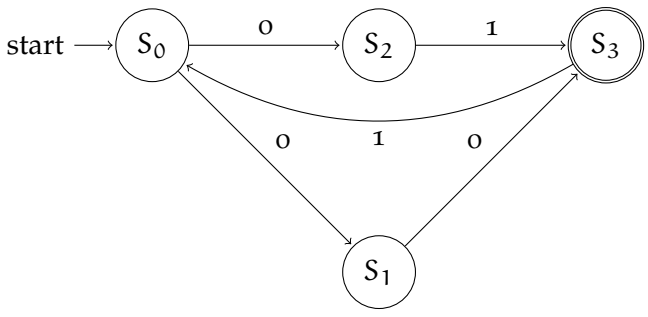


Figura 68: Automat finit nedeterminist

Rezolvare:
1.Pentru deducerea automatului finit determinist care accepta limbajul se construieste tabela de tranzitii ilustrata la pagina 7.

f'	0	1
$\{S_0\}$	$\{S_1, S_2\}$	$\emptyset$
$\{S_1\}$	$\{S_3\}$	$\emptyset$
$\{S_2\}$	$\emptyset$	$\{S_3\}$
$^*\{S_3\}$	$\emptyset$	$\{S_0\}$
$\{S_0, S_1\}$	$\{S_1, S_2, S_3\}$	$\emptyset$
$\{S_0, S_2\}$	$\{S_1, S_2\}$	$\{S_3\}$
$^*\{S_0, S_3\}$	$\{S_1, S_2\}$	$\{S_0\}$
$\{S_1, S_2\}$	$\{S_3\}$	$\{S_3\}$
$^*\{S_1, S_3\}$	$\{S_3\}$	$\{S_0\}$
$^*\{S_2, S_3\}$	$\emptyset$	$\{S_0, S_3\}$
$\{S_0, S_1, S_2\}$	$\{S_1, S_2, S_3\}$	$\{S_3\}$
$^*\{S_0, S_1, S_3\}$	$\{S_1, S_2, S_3\}$	$\{S_0\}$
$^*\{S_0, S_2, S_3\}$	$\{S_1, S_2\}$	$\{S_0, S_3\}$
$^*\{S_1, S_2, S_3\}$	$\{S_3\}$	$\{S_0, S_3\}$
$^*\{S_0, S_1, S_2, S_3\}$	$\{S_1, S_2, S_3\}$	$\{S_0, S_3\}$

Automatul finit rezultat pe baza tabelei este ilustrat in Figura 1.6.
2.Se considera propozitiile"01100" si "011" .In continuare se vor scrie succesiunile relatiilor de miscare corespunzatoare propozitiilor pentru automatul finit determinist de la punctul 1.
$(01100, \{S_0\}) \vdash (1100, \{S_1, S_2\}) \vdash (100, \{S_3\}) \vdash (00, \{S_0\}) \vdash (0, \{S_1, S_2\} \vdash (\epsilon, \{S_3\}) \Rightarrow$ propozitia a ajuns intr-una din starile finale ale automatului, fiind acceptata de catre acesta.
$(011, \{S_0\} \vdash (11, \{S_1, S_2\}) \vdash (1, \{S_3\}) \vdash (\epsilon, \{S_0\})$?? $\{S_0\}$ nu este stare finala $\Rightarrow$ propozitia nu a fost acceptata de automat!

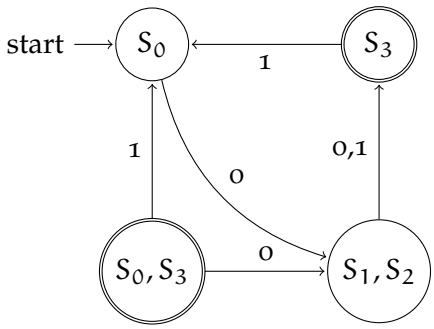


Figura 69: Automat finit determinist construit pe baza tabelei de mai sus

16. Fie automatul finit nedeterminist reprezentat prin graful de tranziţie:

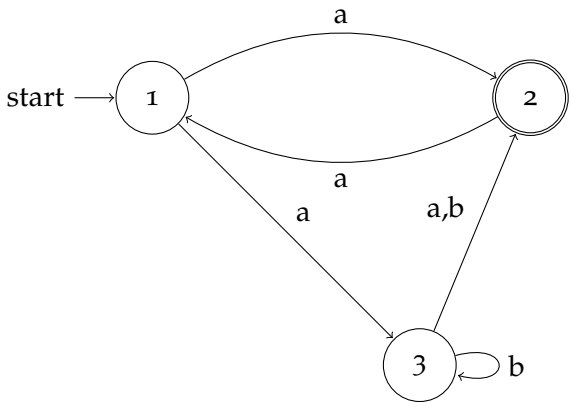


Figura 70: AFN ex.3

a) Deduceţi automatul finit determinist care acceptă acelaşi limbaj.

b) Alegeţi o propoziţie care aparţine acestui limbaj şi una care nu aparţine limbajului. Scrieţi pentru fiecare dintre ele succesiunea de relaţii de mişcare pentru automatul de la 1. Explicaţi.

a) ★ Automatul finit determinist il deducem folosind tabela de tranzitii schitata mai jos.

★ Incepem cu starea 1 si cream automatul in adancime.

★ Astfel se vor elimina starile inutile si vom ajunge la AFD-ul cerut.

f′	a	b
{1}	{23}	-
{2}	{1}	-
{3}	{2}	{23}
{12}	{123}	-
{13}	{23}	{23}
{23}	{12}	{23}
{123}	{123}	{23}

Figura 71: Tabela de tranziţie pentru ex.3

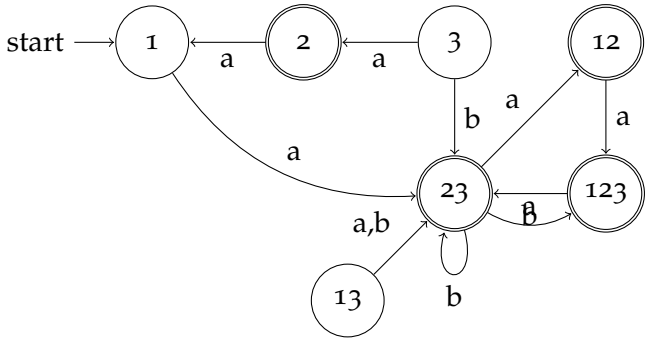


Figura 72: AFN ex.3

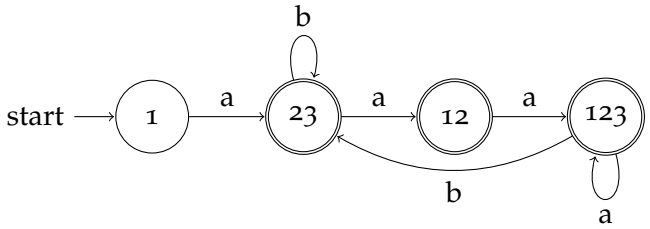
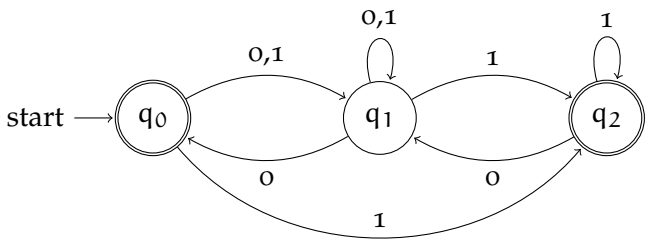


Figura 73: AFD ex.3

b) ★ **abaaa**
(abaaa,1) |– (baaa,23) |– (aaa,23) |– (aa,12) |– (a,123) |– (ϵ,123)
Am consumat toate simbolurile de la intrare si am ajuns intr-o stare finala, cu alte cuvinte secventa a fost acceptata de limbaj.

★ **abbab**
(abbab,1) |– (bbab,23) |– (bab,23) |– (ab,23) |– (b,12)
Am ajuns intr-o stare finala insa nu au fost consumate toate simbolurile de la intrare, cu alte cuvinte secventa nu a fost acceptata de limbaj.

17. Fie automatul finit nedeterminist reprezentat prin graful de tranziţie:



a) Deduceţi automatul finit determinist care acceptă acelaşi limbaj.

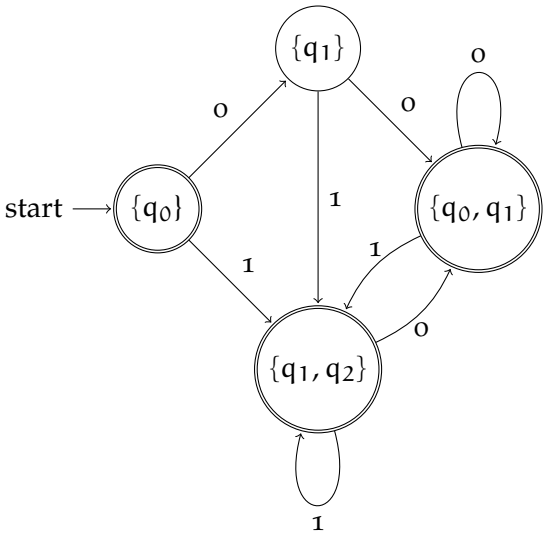
b) Alegeţi o propoziţie care aparţine acestui limbaj şi una care nu aparţine limbajului. Scrieţi pentru fiecare dintre ele succesiunea de relaţii de mişcare pentru automatul de la 1. Explicaţi.

a) $AFN = \{Q, F, q_0, f, \Sigma\}$
$\Sigma = \{0, 1\}$ - alfabetul limbajului

$Q = \{q_0, q_1, q_2\}$ - multimea starilor
$F = \{q_0, q_2\}$ - multimea starilor finale
q_0 - stare initiala
f - functia de tranzitie

$AFD = \{Q', F', \{q_0\}, f', \Sigma\}$

f'	0	1
* $\{q_0\}$	$\{q_1\}$	$\{q_1, q_2\}$
$\{q_1\}$	$\{q_0, q_1\}$	$\{q_1, q_2\}$
* $\{q_2\}$	$\{q_1\}$	$\{q_2\}$
* $\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_1, q_2\}$
* $\{q_0, q_2\}$	$\{q_1\}$	$\{q_1, q_2\}$
* $\{q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_1, q_2\}$
*$\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_1, q_2\}$



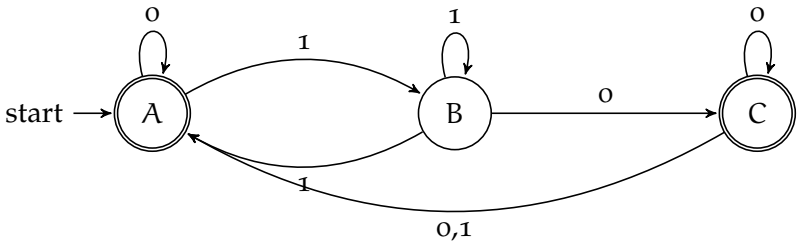
b) Fie propozitia "001101" care apartine limbajului.
Vom porni de la stare initiala q_0 care ne va duce in starea q_1 cu "01101" s.a.m.d. Vom ajunge in starea $\{q_1, q_2\}$ cu "1" care ne va duce in aceeasi stare care este si stare finala. Astfel se sfarseste succesiunea.

$(001101, \{q_0\}) \to (01101, \{q_1\}) \to (1101, \{q_0, q_1\}) \to (101, \{q_0, q_1\}) \to (01, \{q_0, q_1\}) \to (1, \{q_1, q_2\}) \to (\epsilon, \{q_1\})$

Fie propozitia "0" care nu apartine limbajului.

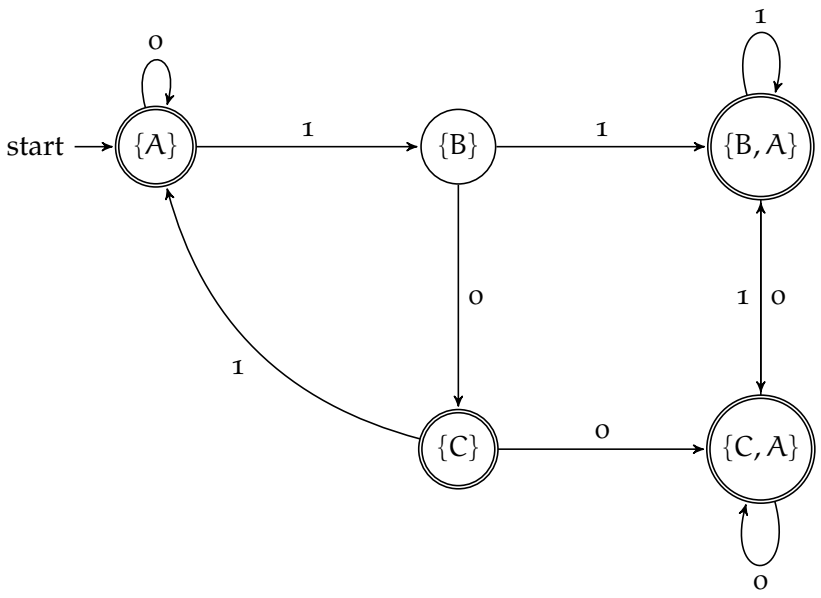
$(0, \{q_0\}) \to (\epsilon, \{q_1\})$, dar starea q_1 nu este stare finala (automat finit determinist corect, propozitia nu apartine limbajului).

18. Fie automatul finit nedeterminist reprezentat prin graful de tranziţie:



1. Deduceţi automatul finit determinist care acceptă acelaşi limbaj.

f′	0	1
{A}	{A}	{B}
{B}	{C}	{B, A}
{C}	{C, A}	{A}
{A, B}	{A, C}	{B, A}
{A, C}	{A, C}	{A, B}
{B, C}	{A, C}	{A, B}
{A, B, C}	{A, C}	{A, B}



2. Alegeţi o propoziţie care aparţine acestui limbaj şi una care nu aparţine limbajului. Scrieţi pentru fiecare dintre ele succesiunea de relaţii de mişcare pentru automatul de

la 1. Explicaţi.

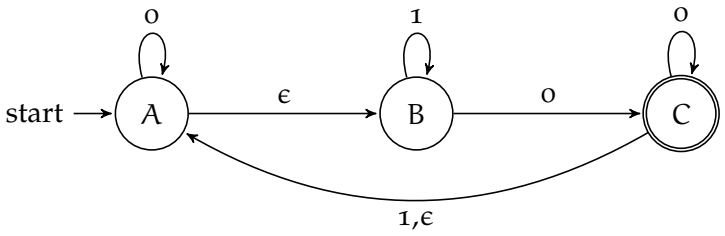
Propozitie ce apartine limbajului: 001001

(001001,{A}) → (01001,{A}) → (1001,{A}) → (001,{B}) → (01,{C}) → (1,{A,C}) → (ϵ ,{A,B}) : S-au consumat toate cuvintele si ne aflam in starea $\{A, B\}$ care este stare finala ⇒ propozitia 001001 apartine limbajului!

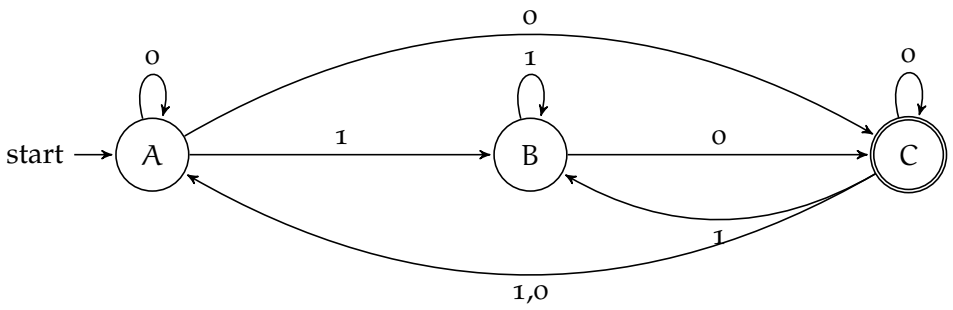
Propozitie ce nu apartine limbajului: 101001

(101001,{A}) → (01001,{B}) → (1001,{C}) → (001,{A}) → (01,{A}) → (1,{A}) → (ϵ ,{B}) : S-au consumat toate cuvintele si ne aflam in starea $\{B\}$ care nu este stare finala ⇒ propozitia 101001 nu apartine limbajului!

19. Fie automatul finit epsilon reprezentat prin graful de tranziţie:



Eliminati tranzitiile epsilon.



20. Fie automatul finit ϵ reprezentat prin graful de tranziţie:

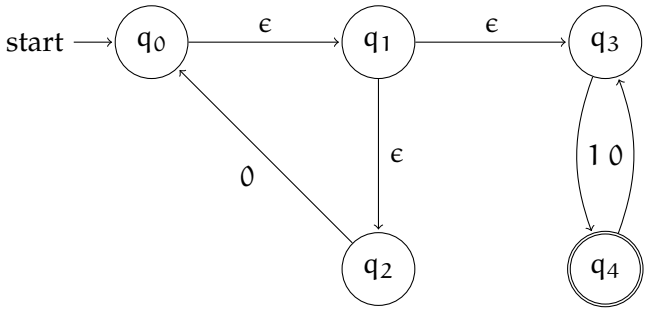


Figura 74: Graful de tranziţie

Transcriem graful fara tranzitiile ϵ;

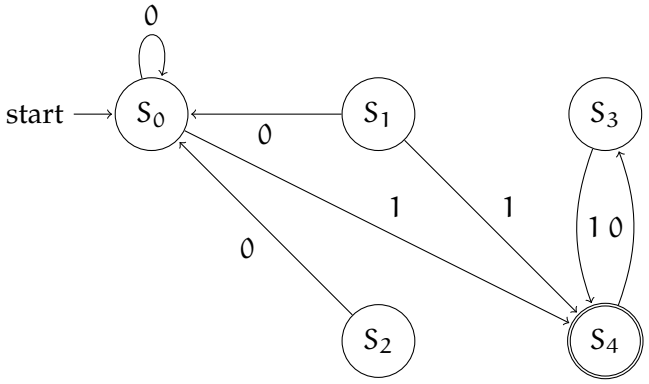


Figura 75: Graful de tranziţie

$CL(S_0) = S_1, S_2, S_3$

$CL(S_1) = S_2, S_3$

$CL(S_2) = \emptyset$

$CL(S_3) = \emptyset$

$CL(S_4) = \emptyset$

Se elimina starile inutile. Obtinem:

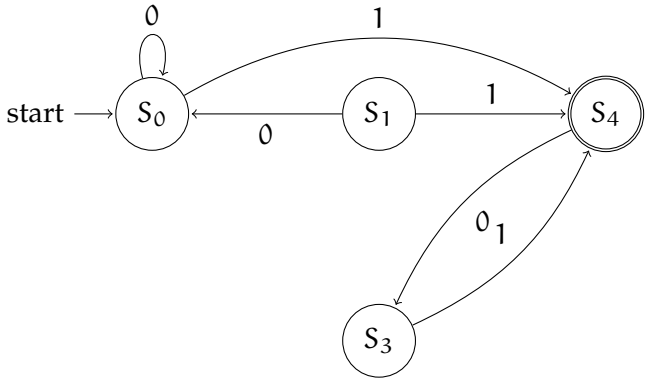


Figura 76: Graful de tranziţie

21. Se considera automatul finit epsilon reprezentat prin graful de tranziţie din Figura 1.7. Se cere eliminarea tranzitiilor ϵ.

    In automatul finit considerat exista o singura tranzitie ϵ, si anume de la starea q_2 la starea q_3.

    Prin urmare $CL(q_2) = \{q_3\}$

    Iar $CL(q)$ reprezinta multimea starilor la care se poate ajunge pornind din starea q si facand doar tranzitii ϵ.

    Astfel, cum din q_3 se poate ajunge in q_3 cu $0 \Rightarrow$ din q_2 se poate ajunge in q_3 cu 0. De asemenea, din q_3 se poate ajunge si in starea initiala cu 0 sau $1 \Rightarrow$ si din q_2 se poate ajunge in q_1 cu 0 sau 1.

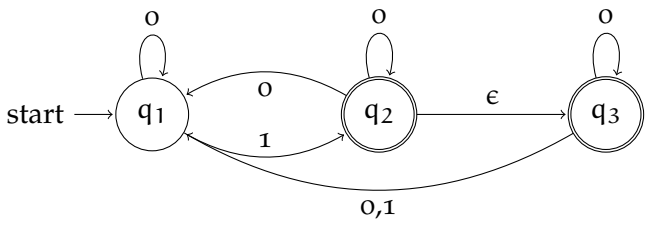


Figura 77: Automat finit ϵ

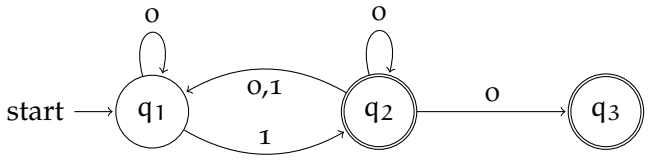


Figura 78: Automat finit rezultat in urma eliminarii ϵ

22. Fie automatul finit epsilon reprezentat prin graful de tranziţie de mai jos.Eliminati tranzitiile ϵ .

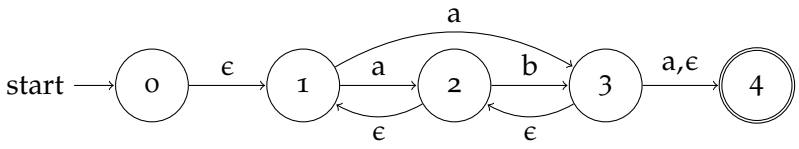


Figura 79: Automat epsilon ex.4

Eliminarea tranzitiilor ϵ inseamna inlocuirea lor cu combinatiile de dupa ele.

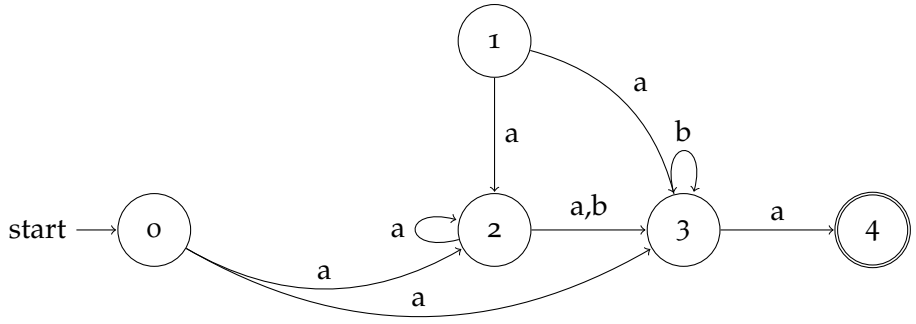


Figura 80: Automat finit fara tranzitii ϵ ex.4

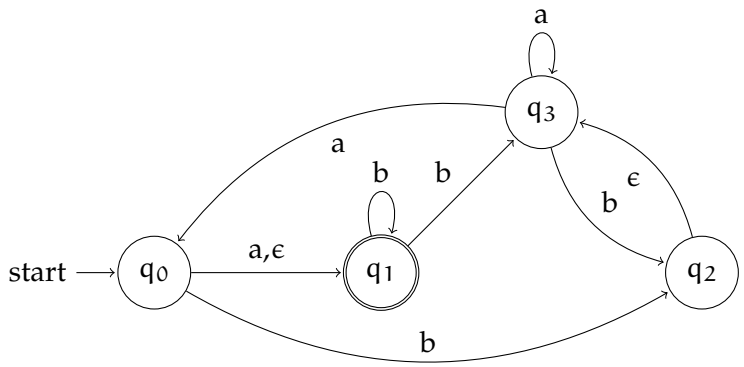
23. Fie automatul finit epsilon reprezentat prin graful de tranziţie:

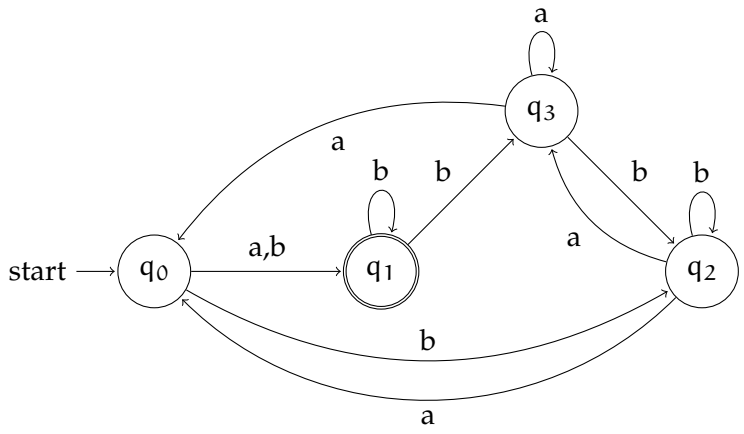
Eliminaţi tranziţiile ϵ.

Eliminam tranzitiile ϵ, dupa care restabilim starile finale.

Calculam multimile closure pentru a stabili ce stari vor fi transformate in stari finale.

$C_L\{q_0\} = \varnothing$
$C_L\{q_1\} = \varnothing$
$C_L\{q_2\} = \{q_3\}$
$C_L\{q_3\} = \varnothing$





Deoarece starea finala initiala q_1 nu se regaseste in multimile closure, nu exista stari finale noi.

24. Fie automatul finit epsilon reprezentat prin graful de tranziţie:

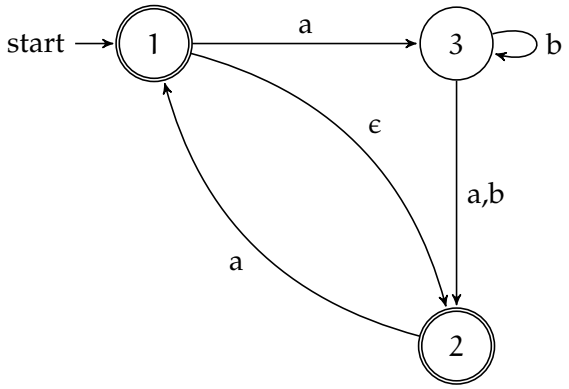
    Eliminaţi tranziţiile ϵ.

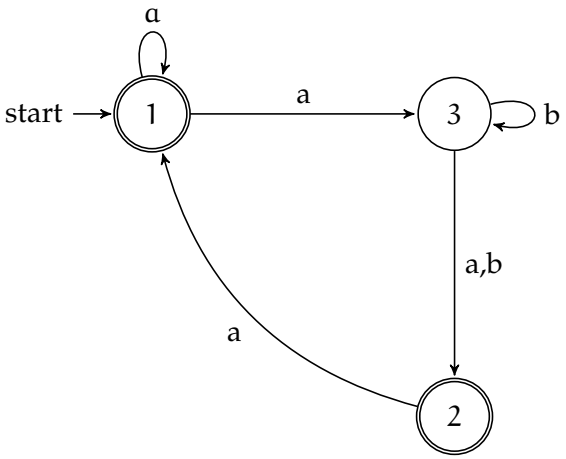
25. Fie gramatica generatoare ale cărei elemente de definiţie se deduc din regulile:

    $S \to 0AB|0B$

    $A \to 00A|\epsilon$

    $B \to 10B|10$





★ 1. Construiţi automatul finit determinist care acceptă limbajul definit de această gramatică.

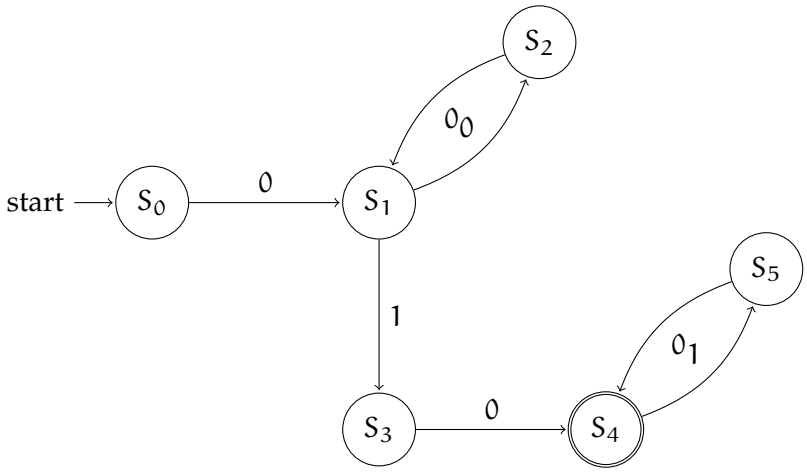


Figura 81: Graful de tranziţie

★ Definiţi expresia regulată care recunoaşte limbajul definit de această gramatică.
$0(00)^*(10)+$

26. Fie gramatica generatoare ale cărei elemente de definiţie se deduc din regulile:

$S \longrightarrow AS|\epsilon$

$A \longrightarrow aBb$

$B \longrightarrow baB|ba$

1. Construiţi automatul finit determinist care acceptă limbajul definit de această gramatică.

2. Definiţi expresia regulată care recunoaşte limbajul definit de această gramatică.

Automatul finit determinist care accepta limbajul definit de aceasta gramatica se va construi pe baza unui automat finit nedeterminist, ilustrat in Figura 1.9, in urma realizarii tabelei de tranzitii de la pagina 9. Prin urmare, automatul cerut este reprezentat in Figura 1.10.

Expresia regulata este $[a(ba)^+b]^+|\epsilon$

f'	a	b
$*\{q_0\}$	$\{q_1\}$	$\emptyset$
$\{q_1\}$	$\emptyset$	$\{q_2\}$
$\{q_2\}$	$\{q_3\}$	$\emptyset$
$*\{q_3\}$	$\emptyset$	$\{q_0, q_2\}$
$*\{q_0, q_1\}$	$\{q_1\}$	$\{q_2\}$
$*\{q_0, q_2\}$	$\{q_1, q_3\}$	$\emptyset$
$*\{q_0, q_3\}$	$\{q_1\}$	$\{q_0, q_2\}$
$\{q_1, q_2\}$	$\{q_3\}$	$\{q_2\}$
$*\{q_1, q_3\}$	$\emptyset$	$\{q_0, q_2\}$
$*\{q_2, q_3\}$	$\{q_3\}$	$\{q_0, q_2\}$
$*\{q_0, q_1, q_2\}$	$\{q_1, q_3\}$	$\{q_2\}$
$*\{q_0, q_1, q_3\}$	$\{q_1\}$	$\{q_0, q_2\}$
$*\{q_0, q_2, q_3\}$	$\{q_1, q_3\}$	$\{q_0, q_2\}$
$*\{q_1, q_2, q_3\}$	$\{q_3\}$	$\{q_0, q_2\}$
$*\{q_0, q_1, q_2, q_3\}$	$\{q_1, q_3\}$	$\{q_0, q_2\}$

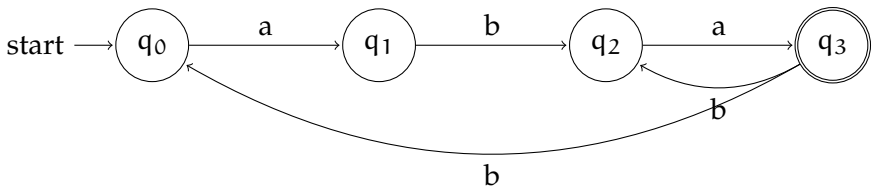


Figura 82: Automat finit nedeterminist

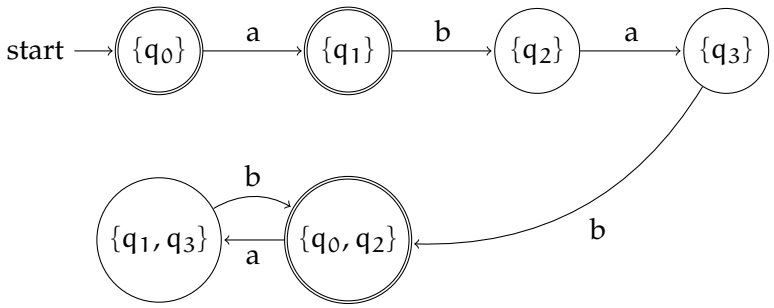


Figura 83: Automat determinist

27. Fie gramatica generatoare:

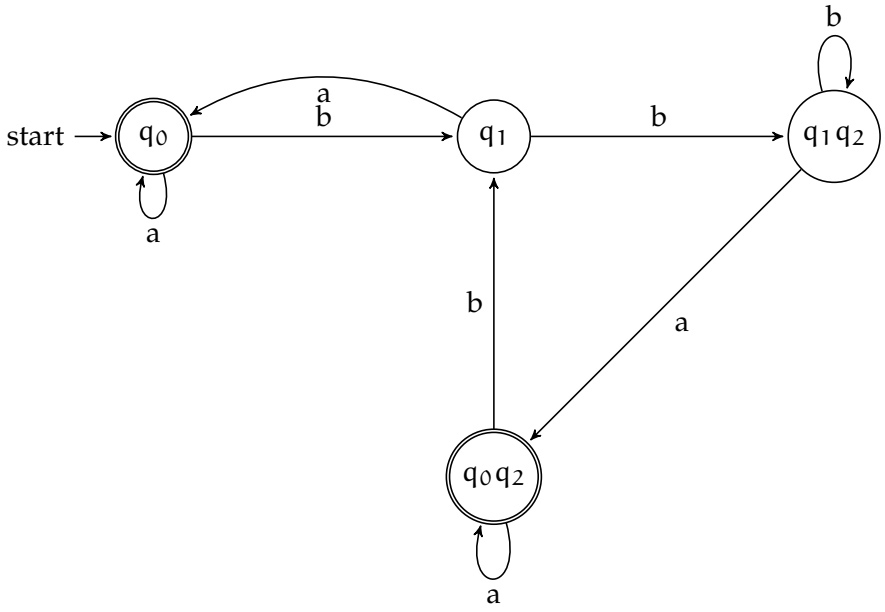
S →A | ABa | AbA

A →Aa | epsilon

B→ Bb | BC

C→ CB | CA | bB

1. Construiţi automatul finit determinist care acceptă limbajul definit de această gramatică.



2. Definiţi expresia regulată care recunoaşte limbajul definit de această gramatică.

$a*(b*a+)*$

28. Fie automatul finit nedeterminist schitat mai jos.
    a) Construiţi o gramatică generatoare care să accepte acelaşi limbaj.
    b) Definiţi expresia regulată care recunoaşte limbajul definit de această gramatică.

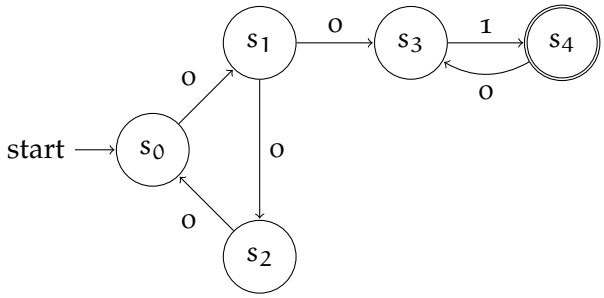


Figura 84: AFN ex.5

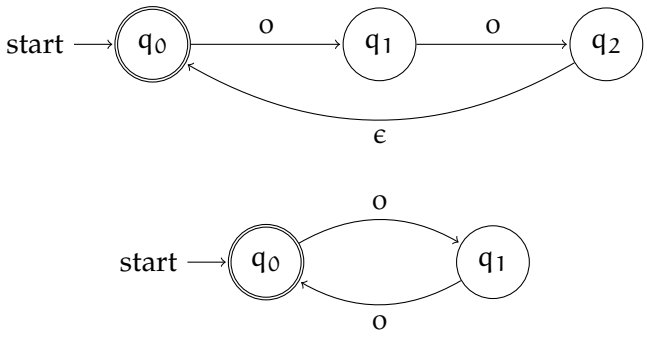
a) Gramatica generatoare care accepta limbajul este definita de:
   - ★ $\Sigma = \{0, 1\}$
   - ★ S simbol de start
   - ★ $P = \{S \to 0SecF$
     $Sec \to \epsilon|000Sec|000$
     $F \to 01|01F\}$
   - ★ $V_N = \{S, Sec, F\}$

b) Expresia regulata ce corespunde automatului este : $(0|0(000)+)(01)+$

29. Fie gramatica generatoare ale cărei elemente de definiţie se deduc din regulile:

$S \to ASA \mid A \mid \epsilon$

$A \to oo \mid \epsilon$



a) Construiţi automatul finit determinist care acceptă limbajul definit de această gramatică.

b) Definiţi expresia regulată care recunoaşte limbajul definit de această gramatică.

a) Expresia regulata: (00)* (nr par de 0)

Eliminam tranzitiile ϵ:

b) i. $S \to \epsilon$

ii. $S \to A$
$A \to 00 \mid \epsilon$

iii. $S \to ASA \xrightarrow{S \to ASA} AASAA \xrightarrow[S, A_2, A_4 \to \epsilon]{A_1, A_3 \to 00} 00 \ldots 00$ (nr par de 0)

Expresia regulata: (00)* (grupul 00 apare ori de cate ori)

30. Fie gramatica generatoare ale carei elemente de definitie se deduc din regulile :

$S -> aaS|aA$

$A -> abS|b$

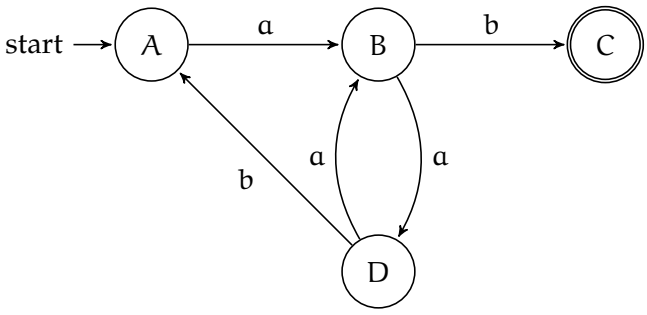
1. Construiţi automatul finit determinist care acceptă limbajul definit de această gramatică.

2. Definiţi expresia regulată care recunoaşte limbajul definit de această gramatică.

Expresia regulata ce recunoaste limbajul definit de aceasta gramatica este : a((aa) | (aba))*b .

31. Scrieţi expresia regulată care permite recunoaşterea propoziţiilor din limbajul cu definiţia:

★ „Toate succesiunile de simboluri formate pe baza simbolurilor a, b şi c în care, dacă apare un grup de trei de a, urmează obligatoriu un c, iar dacă nu apare un astfel



de grup, atunci nu va apărea deloc nici c ."
$([abc]^*(aaac) + [abc]^*)$

★ „Toate succesiunile de simboluri formate pe baza simbolurilor a şi b în care numărul de b este impar."
\(ba)*b((a*b){2}*a*\b

32. Scrieţi expresia regulată care permite recunoaşterea propoziţiilor din limbajul cu definiţia:
1. "Toate succesiunile de simboluri formate pe baza simbolurilor a, b şi c, care conţin un număr par de grupuri ab, dacă conţin un număr par de grupuri bc, sau un număr impar de grupuri bc şi niciun grup ab."
2. "Toate succesiunile de simboluri formate pe baza simbolurilor 0, 1 şi 2 în care 2 este urmat imediat de 1."

Rezolvare:
1. Varianta 1:
$\big((a^*c^*|b^*a^*|b^*a^+c^*)(abab)^+(bcbc)^+(a^*c^+b^*|a^*c^*|b^*a^+c^*|c^*b^*a^*)\big)|\big((a^*c^*|b^*|b^*a^+c^+)\big(bc(bcbc)^*\big)(b^*a^+c^*|a^*c^*|a^*c^+b^*|c^*b^*a^*)\big)$
Varianta 2:
$\big((b^*|c^*|a^*c^+|c^*b^*)\big((bcbc)^+a^*c^*(abab)^+(bcbc)^+b^*a^+c^*(abab)^+\big)(a^*|b^*|a^+c^+b^*|b^*a^*|b^*a^+c^*\big)|\big((a^*c^*|b^*|b^*a^+c^+)\big((bc(bcbc)^*\big)(b^*a^+c^*|a^*c^*|a^*c^+b^*|c^*b^*a^*)\big)$
2.
$0^*1^*0^*(21)^+0^*1^*0^*$

33. Scrieţi expresia regulată care permite recunoaşterea propoziţiilor din limbajul cu definiţia:

1. „Toate succesiunile de simboluri formate pe baza simbolurilor a şi b care conţin atât subşirul aa cât şi subşirul bb, sau nu conţin mai mulţi de a succesiv sau mai mulţi de b succesiv."

$((ab)*(aa)(bb)(ab)*)|((ba)*(bb)(aa)(ba)*)$
-toate succesiunile decaractere care contin a,b si contin subsirul aa si bb,dar nu contin mai multi de a succesiv sau de b succesiv;

2. „Toate succesiunile de simboluri formate pe baza simbolurilor 0, 1 şi 2 care conţin un număr par de 2."

$(0^*1^*(0^*1^*2(1^*0^*1^*0^*1^*)^*20^*1^*)^*0^*1^*)^*$

34. Scrieţi expresia regulată care permite recunoaşterea propoziţiilor din limbajul cu definiţia:

a) „Toate succesiunile de simboluri formate pe baza simbolurilor a şi b care conţin subşirul aa, dar nu şi subşirul bb."

b) "Toate succesiunile de simboluri formate pe baza simbolurilor 0 şi 1 care constă din 1 şi 0 alternativ."

a) $[a-b]*aa[\hat{}(b+)]$

b) $(01)^* \mid (10)^*$

35. Scrieţi expresia regulată care permite recunoaşterea propoziţiilor din limbajul cu definiţia:

    1. „Toate succesiunile de simboluri formate pe baza simbolurilor a şi b care conţin subşirul aba sau subşirul bab sau ambele. ”

    $[a-b]*((aba)+(bab)*|(aba)*(bab)+|(aba)+(bab)+)+[a-b]*.$

    2. „Toate succesiunile de simboluri formate pe baza simbolurilor a, b şi c în care numărul de b este 4*i + 1, cu i >= 0.”

    ([ac]* b [ac]*) ([ac]* b [ac]*){4})*.

36. Scrieţi expresia regulată care permite recunoaşterea propoziţiilor din limbajul cu definiţia:

    a) **"Toate succesiunile de simboluri formate pe baza simbolurilor a, b şi c care încep cu c şi se termină cu o succesiune de doi sau mai mulţi de b."**

    b) **"Toate succesiunile de simboluri formate pe baza simbolurilor 0, 1 şi 2 în care apare un număr impar de 1."**

    a) **c[abc]*bb+**

    b) **[02]*1[02]*((110)|(112)|(101)|(121)|(011)|(211))*[02]***