

CONF.UNIV.DR. RADU BORIGA

CONF.UNIV.DR. ANA CRISTINA DĂSCĂLESCU

PROF.UNIV.DR. HORIA IOAN GEORGESCU

LIMBAJE FORMALE, AUTOMATE ȘI CALCULABILITATE

UNIVERSITATEA TITU MAIORESCU DIN BUCUREȘTI

FACULTATEA DE INFORMATICĂ

2014

PREZENTAREA MODULULUI

Modulul "*Limbaje formale, automate și calculabilitate*" are drept scop prezentarea unor aspecte teoretice și practice fundamentale din domeniul limbajelor formale și automatelor, accentul fiind pus pe modul de utilizarea al acestora în modelarea unor sisteme informatiche. De asemenea, în cadrul acestui modul se demonstrează faptul că nu există algoritmi de rezolvare pentru orice problemă.

Modulul începe cu o unitate de învățare în care se prezintă noțiuni fundamentale despre gramatici și clasificarea Chomsky a acestora, precum și numeroase exemple. În următorul modul sunt prezentate aspecte referitoare la gramaticile independente de context: forma normală Chomsky, arbori de derivare și metode de simplificare a gramaticilor independente de context. În al treilea modul sunt prezentate gramaticile regulate și automatele finite, precum și echivalența dintre acestea. În al patrulea modul sunt prezentate automatele pushdown și echivalența lor cu gramaticile independente de context. În ultimul modul sunt prezentate proprietăți de închidere ale limbajelor independente de context și ale celor regulate.

Prin însăși natura sa, acest modul este strâns legat de modulele "*Logică matematică și computațională*" și "*Fundamentele algebrice ale informaticii*".

CUPRINS

MODULUL I - Gramatici. Clasificarea Chomsky a gramaticilor	4
1.1. Notații și definiții	4
1.2. Clasificarea Chomsky a gramaticilor	5
1.3. Exerciții propuse	7
MODULUL II - Gramatici independente de context.....	9
2.1. Derivări în gramatici independente de context	9
2.2. Simplificarea gramaticilor independente de context.....	10
2.3. Forma normală Chomsky a unei gramatici independente de context	17
2.4. Arbori de derivare	24
2.5. Algoritmul Cocke-YOUNGER-Kasami	28
2.6. Exerciții propuse	30
MODULUL III - Gramatici regulate. Automate finite deterministe și nedeterministe	32
3.1. Gramatici regulate.....	32
3.2. Automate finite deterministe.....	33
3.3. Automate finite nedeterministe	37
3.4. Echivalența dintre automatele finite deterministe și automatele finite nedeterministe.....	39
3.5. Echivalența dintre gramaticile regulate și automatele finite nedeterministe.....	42
3.6. Exerciții propuse	43
MODULUL IV - Automate pushdown.....	45
4.1. Automate pushdown	45
4.2. Construcția unui automat pushdown echivalent cu o gramatică independentă de context	48
4.3. Construcția unei gramatici independente de context echivalentă cu un automat pushdown	49
MODULUL V - Proprietăți ale limbajelor independente de context și ale limbajelor regulate..	51
5.1. Lema Bar-Hillel. Lema de pompare	51
5.2. Operații cu limbaje.....	52
BIBLIOGRAFIE.....	55

MODULUL I

Gramatici. Clasificarea Chomsky a gramaticilor

1.1. Notații și definiții

Fie V un *alfabet*, adică o mulțime finită și nevidă de simboluri. În continuare vom folosi următoarele notații:

- V^k = mulțimea cuvintelor de lungime k formate cu simboluri din V ;
- V^* = mulțimea cuvintelor de orice lungime peste alfabetul V ;
- $|w|$ = lungimea unui cuvânt $w \in V^*$;
- λ = cuvântul vid (are lungimea $|\lambda| = 0$).

Definiția 1.1: Se numește *limbaj peste un alfabet* V o mulțime $L \subset V^*$.

Exemplul 1.1: Fie $V = \{a, b\} \Rightarrow V^* = \{\lambda, a, b, ab, ba, bb, aba, bba, \dots, baba, \dots\}$. Putem defini un limbaj $L = \{a, ba, aba, baba\}$.

Definiția 1.2: Se numește *gramatică* un cvadruplu $G = (N, T, S, P)$ în care:

- N se numește *alfabetul simbolurilor neterminale*;
- T se numește *alfabetul simbolurilor terminale*;
- S se numește *simbolul inițial al gramaticii* ($S \in N$);
- P este *mulțimea producțiilor*, adică reguli de substituție de forma $\alpha \rightarrow \beta$, unde $\alpha \in (N \cup T)^* N (N \cup T)^*$, iar $\beta \in (N \cup T)^*$.

Definiția 1.3: Spunem că din α derivă β și notăm acest lucru prin $\alpha \Rightarrow \beta$ dacă $\alpha = \alpha_1 \alpha_2 \alpha_3$, $\beta = \alpha_1 \alpha'_2 \alpha_3$ și $\alpha_2 \rightarrow \alpha'_2 \in P$.

În continuare, vom nota *închiderea reflexivă și tranzitivă a relației* \Rightarrow prin $\stackrel{*}{\Rightarrow}$. Deci $\alpha \stackrel{*}{\Rightarrow} \beta$ fie dacă $\alpha = \beta$, fie dacă există $\alpha_1, \alpha_2, \dots, \alpha_k$ astfel încât $\alpha \Rightarrow \alpha_1 \Rightarrow \alpha_2 \dots \Rightarrow \alpha_k \Rightarrow \beta$.

Definiția 1.4: Se numește *limbaj generat de o gramatică* G mulțimea:

$$L(G) = \left\{ w \in T^* \mid S \stackrel{*}{\Rightarrow} w \right\}$$

Exemplul 1.2: Fie gramatica $G = (N, T, S, P)$, unde $N = \{S\}$, $T = \{a\}$ și $P = \{S \rightarrow a, S \rightarrow aS\}$. Se poate observa foarte ușor că limbajul generat este $LG = a, aa, aaa, \dots = an | n \geq 1$.

Exemplul 1.3: Fie gramatica $G = (N, T, S, P)$, unde $N = \{S\}$, $T = \{a, b\}$, iar mulțimea producțiilor P este următoarea:

$$\begin{array}{ll}
 S \rightarrow \lambda & (1) \\
 S \rightarrow a & (2) \\
 S \rightarrow b & (3) \\
 S \rightarrow aSa & (4) \\
 S \rightarrow bSb & (5)
 \end{array}$$

Cuvântul $w = aababaa \in L(G)$ deoarece $S \xrightarrow[4]{*} aSa \xrightarrow[4]{*} aaSaa \xrightarrow[5]{*} aabSbaa \xrightarrow[2]{*} aababaa = w$. Se poate observa că limbajul generat de gramatica G este format din toate palindromurile formate din literele a și b .

Exemplul 1.4: Fie gramatica $G = (N, T, S, P)$, unde $N = \{S\}$, $T = \{(,)\}$, iar mulțimea producțiilor P este următoarea:

$$\begin{array}{ll}
 S \rightarrow () & (1) \\
 S \rightarrow (S) & (2) \\
 S \rightarrow SS & (3)
 \end{array}$$

Fie cuvântul $w = (())(())(())()$. Cuvântul $w \in L(G)$, deoarece se poate obține printr-o derivare din simbolul inițial S astfel:

$$\begin{aligned}
 S &\xrightarrow[3]{*} SS \xrightarrow[3]{*} SSS \xrightarrow[1]{*} SS() \xrightarrow[2]{*} (S)S() \xrightarrow[3]{*} (SS)S() \xrightarrow[1]{*} ((S))S() \xrightarrow[1]{*} (())S() \xrightarrow[2]{*} (()) \\
 &\xrightarrow[2]{*} (())(S)() \xrightarrow[2]{*} (())((S))() \xrightarrow[1]{*} (())(())(())() \Rightarrow w \in L(G)
 \end{aligned}$$

Se observă că limbajul generat de gramatica G este format din toate sirurile de paranteze rotunde care se închid corect.

Definiția 1.5: Două gramatici G_1 și G_2 sunt *echivalente* dacă $L(G_1) = L(G_2)$.

1.2. Clasificarea Chomsky a gramaticilor

În funcție de forma producțiilor, Noam Chomsky a ierarhizat gramaticile astfel:

Tip	Denumire	Forma producțiilor
0	gramatică generală	oarecare
1	gramatică dependentă de context	$\alpha \rightarrow \beta$ cu $ \alpha \leq \beta $
2	gramatică independentă de context	$A \rightarrow \alpha$ cu $A \in N$ și $\alpha \in (N \cup T)^*$
3	gramatică regulată	$A \rightarrow aB$ sau $A \rightarrow a$ cu $A, B \in N$ și $a \in T$

Observația 1.1: Fie \mathcal{G}_i familia gramaticilor de tip i ($i = \overline{0,3}$). Se observă că $\mathcal{G}_3 \subset \mathcal{G}_2 \subset \mathcal{G}_1 \subset \mathcal{G}_0$.

Definiția 1.6: Un limbaj L este de tip i ($i = \overline{0,3}$) dacă există o gramatică G de tipul i pentru care $L(G) = L$.

Observația 1.2: Fie \mathcal{L}_i familia limbajelor de tip i ($i = \overline{0,3}$). Se observă că $\mathcal{L}_3 \subset \mathcal{L}_2 \subset \mathcal{L}_1 \subset \mathcal{L}_0$.

Exemplul 1.5: Fie gramatica $G = (N, T, S, P)$, unde $N = \{S, A, B\}, T = \{a, b\}$. În tabelul de mai jos este indicat tipul gramaticii G în funcție de mulțimea producțiilor P :

P			
$A \rightarrow a$ $B \rightarrow b$ $S \rightarrow aA$ $aaA \rightarrow bB$ $bbB \rightarrow S$	$A \rightarrow a$ $B \rightarrow b$ $S \rightarrow aA$ $aA \rightarrow bB$ $bB \rightarrow SB$	$A \rightarrow a$ $B \rightarrow b$ $S \rightarrow aA$ $A \rightarrow bB$ $B \rightarrow bB$	$A \rightarrow a$ $B \rightarrow b$ $S \rightarrow aA$ $S \rightarrow bB$ $A \rightarrow aB$
gramatică generală (tip 0)	gramatică dependentă de context (tip 1)	gramatică independentă de context (tip 2)	gramatică regulată (tip 3)

Propoziția 1.1: Fie $G = (N, T, S, P)$ o gramatică de tipul i ($i = \overline{1,3}$). Atunci există o gramatică G' echivalentă cu G și de același tip, cu proprietatea că simbolul inițial S nu apare în membrul drept al producțiilor.

Demonstrație:

Fie S' un simbol nou, adică $S' \notin N \cup T$. Construim gramatica $G' = (N \cup \{S'\}, T, S', P')$, unde $P' = P \cup \{S' \rightarrow \alpha | S \rightarrow \alpha\}$. Se observă ușor că $L(G) = L(G')$.

Observația 1.3: Din forma producțiilor pentru gramaticile de tipul 1,2 sau 3 rezultă că λ nu aparține limbajului generat de ele. Dacă dorim ca și cuvântul vid λ să aparțină limbajului generat de o gramatică, admitem în mod excepțional producția $S \rightarrow \lambda$, care nu poate avea alte repercusiuni, conform propoziției 1 precedente.

Teorema 1.1: Pentru gramaticile de tipurile 1,2 și 3 este posibil să verificăm apartenența unui cuvânt la limbajul generat de ele.

Demonstrație:

Fie $G = (N, T, S, P)$ o gramatică de tipul 1,2 sau 3 și fie $w \in T^*$. Dorim să verificăm dacă $w \in L(G)$.

Fie $n = |w|$ și folosim metoda șirului crescător de mulțimi:

$$\begin{cases} T_0 &= \{S\} \\ T_{k+1} &= T_k \cup \left\{ \alpha \in (N \cup T)^* \mid \exists \beta \in T_k \text{ cu } \beta \xrightarrow{*} \alpha \text{ și } |\alpha| \leq n \right\} \end{cases}$$

Deoarece $T_0 \subset T_1 \subset \dots \subset T_k \subset T_{k+1} \subset \dots \subset F$, unde F este mulțimea finită a cuvintelor de lungime cel mult n formate din simboluri terminale și neterminale, rezultă că sirul se stabilizează, respectiv $\exists k_0 \in \mathbb{N}$ astfel încât $T_{k_0} = T_{k_0+1} = T_{k_0+2} = \dots$. Evident, $w \in L(G) \Leftrightarrow w \in T_{k_0}$.

Observația 1.4: În practică sunt utilizate mai ales gramaticile independente de context și cele regulate.

1.3. Exerciții propuse

1. Ce relație ierarhică există între gramaticile de tipurile 0,1,2 și 3?
2. Pentru ce tipuri de gramatici nu se poate verifica apartenența unui cuvânt la limbajul generat de ele?
3. Fie gramatica $G = (N, T, S, P)$, unde $N = \{S\}, T = \{a, b\}$, iar mulțimea producțiilor este $P = \{S \rightarrow a, S \rightarrow b, S \rightarrow aSa, S \rightarrow bSb\}$. De ce tip este gramatica G ? Arătați că $w = ababa \in L(G)$.
4. Fie gramatica $G = (N, T, S, P)$, unde $N = \{S\}, T = \{((),)\}$, iar mulțimea producțiilor P este următoarea:

$$S \rightarrow () \quad (1)$$

$$S \rightarrow (S) \quad (2)$$

$$S \rightarrow SS \quad (3)$$

Stabiliți tipul gramaticii și arătați că $w_1 = (())() \in L(G)$, iar $w_2 = ()(()) \notin L(G)$.

5. Fie gramatica $G = (N, T, S, P)$, unde $N = \{S\}, T = \{a, b\}$, iar mulțimea producțiilor P este următoarea:

$$S \rightarrow aa \quad (1)$$

$$S \rightarrow bb \quad (2)$$

$$S \rightarrow aSa \quad (3)$$

$$S \rightarrow bSb \quad (4)$$

Arătați că $w_1 = aabbabbabaa \in L(G)$ și $w_2 = bbabb \notin L(G)$. Stabiliți tipul gramaticii G și descrieți limbajul generat de ea.

6. Fie gramatica $G = (N, T, S, P)$, unde $N = \{S, A, B\}, T = \{a, b\}$, iar mulțimea producțiilor P este indicată în tabelul de mai jos. Completați pe ultima linie a tabelului tipul fiecărei gramatici.

P			
$A \rightarrow a$	$A \rightarrow a$	$A \rightarrow a$	$A \rightarrow a$
$A \rightarrow b$	$B \rightarrow b$	$B \rightarrow b$	$B \rightarrow b$
$S \rightarrow aA$	$S \rightarrow aA$	$S \rightarrow aAa$	$S \rightarrow aA$
$A \rightarrow bB$	$A \rightarrow aB$	$A \rightarrow bBb$	$aA \rightarrow bbbB$
$B \rightarrow A$	$B \rightarrow bA$	$B \rightarrow A$	$bbB \rightarrow A$

MODULUL II

Gramatici independente de context

2.1. Derivări în gramatici independente de context

Definiția 2.1: O gramatică $G = (N, T, S, P)$ se numește *gramatică independentă de context* dacă orice producție a sa este de forma $A \rightarrow \alpha$ cu $A \in N, \alpha \in (NUT)^*$.

Definiția 2.2: Spunem ca o derivare este *derivare stângă* (notată prin \Rightarrow^*) dacă la fiecare pas se înlocuiește neterminálul cel mai din stânga conform unei producții a gramaticii.

Propoziția 2.1: Într-o gramatică independentă de context G orice cuvânt $w \in L(G)$ se poate obține printr-o derivare stângă din simbolul inițial.

Demonstrație:

Demonstrăm prin inducție după numărul k de pași din derivare că:

$$\forall A \in N \text{ și } \forall w \in T^* \text{ avem } A \xrightarrow{*} w \Leftrightarrow A \xrightarrow{S} w$$

Este evident că dacă $A \xrightarrow{S} w$, atunci $A \xrightarrow{*} w$.

Demonstrăm că $A \xrightarrow{*} w \Rightarrow A \xrightarrow{S} w$ astfel:

- Pentru $k = 1$ este evident.
- Presupunem că relația este adeverată pentru $\forall A \in N$ și orice derivare de lungime cel mult k și considerăm o derivare de lungime $k + 1$, în care punem în evidență primul pas:

$$A \xrightarrow{*} \underset{k}{\alpha \xrightarrow{S} w}$$

Atunci $\alpha = A_1 A_2 \dots A_n$ cu $A_1, A_2, \dots, A_n \in N \cup T$, iar $w = w_1 w_2 \dots w_n$ cu $A_i = w_i$ dacă $A_i \in T$ sau $A_i \xrightarrow{*} w_i$ dacă $A_i \in N$. Conform ipotezei de inducție, există derivările stângi $A_i \xrightarrow{S} w_i$, deci vom obține derivarea stângă $A \xrightarrow{S} w$ căutată astfel:

1. aplic $A \rightarrow A_1 A_2 \dots A_n$;
2. aplic pe rând derivările stângi $A_i \xrightarrow{S} w_i$ pentru $i = \overline{1, n}$.

Exemplul 2.1: Fie gramatica independentă de context $G = (N, T, S, P)$, unde $N = \{S, A\}, T = \{a, b\}$, iar mulțimea producțiilor P este următoarea:

$$S \rightarrow aAS \quad (1)$$

$$A \rightarrow SbA \quad (2)$$

$$A \rightarrow SS \quad (3)$$

$$S \rightarrow a \quad (4)$$

$$A \rightarrow ba \quad (5)$$

Cuvântul $w = abaaabbaa \in L(G)$ deoarece:

$$S \xrightarrow[1]{*} aAS \xrightarrow[1]{*} aAaAS \xrightarrow[2]{*} aAaSbAS \xrightarrow[4]{*} aAaSbAa \xrightarrow[5]{*} abaaSbAa \xrightarrow[4]{*} abaaabAa \xrightarrow[5]{*} abaaabbaa = w$$

Arătăm acum că același cuvânt $w = abaaabbaa$ se poate obține din simbolul inițial S printr-o derivare stângă:

$$S \xrightarrow[1]{*} aAS \xrightarrow[5]{*} abaS \xrightarrow[1]{*} abaaAS \xrightarrow[2]{*} abaaSbAS \xrightarrow[4]{*} abaaabAS \xrightarrow[5]{*} abaaabbaS \xrightarrow[4]{*} abaaabbaa = w$$

2.2. Simplificarea gramaticilor independente de context

Prin *simplificarea unei gramatici independente* de context $G = (N, T, S, P)$ se înțelege aducerea producțiilor sale la o formă mai simplă și/sau eliminarea unor simboluri și producții inutile.

Definiția 2.3: Se numește λ -producție o producție de forma $A \rightarrow \lambda$, cu $A \in N$.

Observația 2.1: Dacă $\lambda \in L(G)$, atunci există cel puțin o λ -producție în G .

Observația 2.2: Dacă G are o λ -producție atunci nu neapărat rezultă că $\lambda \in L(G)$.

Exemplul 2.2: Fie $G = (N, T, S, P)$, unde $N = \{S\}$, $T = \{a\}$ și mulțimea producțiilor P este următoarea:

$$S \rightarrow aA \quad (1)$$

$$A \rightarrow aA \quad (2)$$

$$A \rightarrow \lambda \quad (3)$$

Se observă foarte ușor că $L(G) = \{a^n | n \geq 1\}$, deci $\lambda \notin L(G)$!

Teorema 2.1: Pentru orice gramatică independentă de context $G = (N, T, S, P)$ care conține λ -producții, există o gramatică independentă de context $G' = (N, T, S, P')$ echivalentă cu ea și care nu conține λ -producții.

Propoziția 2.2: Eliminarea λ -producțiilor dintr-o gramatică independentă de context se poate realiza folosind următorul algoritm:

Pasul 1: Definim o mulțime $M = \{A \in N | A \xrightarrow{*} \lambda\}$.

Pasul 2: Definim $P' = P \setminus \{A \rightarrow \lambda | A \in N\}$.

Pasul 3: Pentru fiecare producție din P' ce conține în membrul drept un neterminal $A \in M$ adăugăm în P' o nouă producție în care înlocuim neterminalul A cu λ .

Exemplul 2.3: Pentru a elimina λ -producțiile din gramatica independentă de context G din exemplul 2.2, aplicăm algoritmul descris în propoziția 2.2 astfel:

Pasul 1: $M = \{A\}$

Pasul 2: $P' = \{S \rightarrow aA, A \rightarrow aA\}$

Pasul 3: $P' = \{S \rightarrow aA, A \rightarrow aA\} \cup \{S \rightarrow a, A \rightarrow a\} = \{S \rightarrow aA, A \rightarrow aA, S \rightarrow a, A \rightarrow a\}$

Exemplu 2.4: Se consideră gramatica independentă de context $G = (N, T, S, P)$, $N = \{S, X, Y, Z, W\}$, $T = \{a, b\}$ și mulțimea producțiilor P este următoarea:

$$\begin{array}{ll} S \rightarrow X|XY|Z & (1) \\ X \rightarrow Z|\lambda & (2) \\ Y \rightarrow Wa|a & (3) \\ Z \rightarrow WX|AZ|Zb & (4) \\ W \rightarrow XYZ|bXa|\lambda & (5) \end{array}$$

Pentru a elimina λ -producțiile din gramatica G , aplicăm algoritmul descris în propoziția 2.2 astfel:

Pasul 1: $M = \{A \in N \mid A \xrightarrow{*} \lambda\} = \{X, W, S, Z\}$ deoarece $S \xrightarrow[1]{*} X \xrightarrow[2]{*} \lambda$ și $Z \xrightarrow[4]{*} W \xrightarrow[2]{*} X \xrightarrow[5]{*} W \xrightarrow{*} \lambda$.

Pasul 2: Determinăm mulțimea P' inițială:

$$\begin{array}{ll} S \rightarrow X|XY|Z & (1) \\ X \rightarrow Z & (2) \\ Y \rightarrow Wa|a & (3) \\ Z \rightarrow WX|AZ|Zb & (4) \\ W \rightarrow XYZ|bXa & (5) \end{array}$$

Pasul 3: Determinăm mulțimea P' finală, determinând mai întâi producțiile echivalente cu λ -producțiile existente în P :

Producție inițială	Producții echivalente
$S \rightarrow X$	$S \xrightarrow{*} \lambda$
$S \rightarrow XY$	$S \rightarrow Y$
$S \rightarrow Z$	$S \xrightarrow{*} \lambda$
$Y \rightarrow Wa$	$Y \xrightarrow{*} a$
$Z \rightarrow WX$	$Z \rightarrow W$ $Z \rightarrow X$ $Z \xrightarrow{*} \lambda$
$Z \rightarrow aZ$	$Z \rightarrow a$

Producție inițială	Producții echivalente
$Z \rightarrow Zb$	$Z \rightarrow b$
$W \rightarrow XYZ$	$W \rightarrow YZ$ $W \rightarrow XY$ $W \rightarrow Y$
$W \rightarrow bXa$	$W \rightarrow ba$

Astfel, obținem că mulțimea P' este următoarea:

$$S \rightarrow X|XY|Z|Y \quad (1)$$

$$X \rightarrow Z \quad (2)$$

$$Y \rightarrow Wa|a \quad (3)$$

$$Z \rightarrow WX|aZ|Zb|W|X|a|b \quad (4)$$

$$W \rightarrow XYZ|bXa|YZ|XY|Y|ba \quad (5)$$

Definiția 2.4: Se numește *redenumire* o derivare de forma $A \xrightarrow{*} B$, unde $A, B \in N$.

Teorema 2.2: Fie G o gramatică independentă de context. Atunci există o gramatică independentă de context G' echivalentă cu G în care nu mai apar redenumiri.

Demonstrație:

Fie $G = (N, T, S, P)$ o gramatică independentă de context.

Fie $P_1 = \{A \rightarrow B | A, B \in N \text{ și } A \rightarrow B \in P\}$ și $P_2 = P \setminus P_1$.

Fie $G' = (N, T, S, P')$ cu $P' = \left\{ A \rightarrow \alpha \mid \exists B \in N \text{ cu } A \xrightarrow{*} B \text{ și } B \rightarrow \alpha \in P_2 \right\}$.

Demonstrăm în continuare că $L(G) = L(G')$:

- $L(G) \supset L(G')$: Este suficient să observăm că dacă $\xrightarrow[G]{*} \beta$ rezultă și că $\xrightarrow[G]{*} \beta$.
- $L(G) \subset L(G')$: Fie $S \xrightarrow[G]{*} \alpha_1 \xrightarrow[G]{*} \alpha_2 \xrightarrow[G]{*} \dots \xrightarrow[G]{*} w$ o derivare stângă din G și fie k cel mai mare indice pentru care $S \rightarrow \alpha_1, \alpha_1 \rightarrow \alpha_2, \dots, \alpha_{k-1} \rightarrow \alpha_k \in P_1$. Atunci $S \rightarrow \alpha_{k+1} \in P'$ și deci $S \xrightarrow[G']{*} \alpha_{k+1}$.

Se repetă acest raționament, folosind faptul că dacă se aplică succesiv mai multe redenumiri, ele au loc pe aceeași poziție, deoarece derivarea este stângă.

Propoziția 2.3: Eliminarea redenumirilor dintr-o gramatică independentă de context se poate realiza folosind următorul algoritm:

Pasul 1: Se determină toate redenumirile din gramatica G .

Pasul 2: Pentru fiecare redenumire de forma $A \xrightarrow{*} B$ considerăm toate producțiile de forma $B \rightarrow \alpha$ și pentru fiecare dintre aceste producții adăugăm în P' câte o nouă producție $A \rightarrow \alpha$.

Pasul 3: Adăugăm în P' toate producțiile din P care nu sunt redenumiri

Exemplul 2.5: Fie gramatica independentă de context $G = (N, T, S, P)$, unde $N = \{S, A, B\}$, $T = \{a, b\}$ și mulțimea producțiilor P este următoarea:

$$\begin{array}{ll} S \rightarrow A & (1) \\ A \rightarrow aSB & (2) \\ A \rightarrow B & (3) \\ B \rightarrow bSA & (4) \\ A \rightarrow a & (5) \\ B \rightarrow b & (6) \end{array}$$

Determinăm toate redenumirile din P , precum și producțiile noi necesare, astfel:

Redenumiri	Producții noi
$S \rightarrow A$	$S \rightarrow aSB$ $S \rightarrow a$
$A \rightarrow B$	$A \rightarrow bSA$ $A \rightarrow b$
$S \xrightarrow{*} B$	$S \rightarrow b$ $S \rightarrow bSA$

Gramatica independentă de context echivalentă cu G și fără redenumiri este $G' = (N, T, S, P')$, unde mulțimea producțiilor P' este următoarea:

$$\begin{array}{ll} A \rightarrow aSB & (1) \\ B \rightarrow bSA & (2) \\ A \rightarrow a & (3) \\ B \rightarrow b & (4) \\ S \rightarrow aSB & (5) \\ S \rightarrow a & (6) \\ A \rightarrow bSA & (7) \\ A \rightarrow b & (8) \\ S \rightarrow b & (9) \\ S \rightarrow bSA & (10) \end{array} \left. \begin{array}{l} \text{producțiile din } P \text{ care nu sunt redenumiri} \\ \text{producții noi} \end{array} \right\}$$

Exemplul 2.6: Se consideră gramatica independentă de context $G = (N, T, S, P)$, unde $N = \{S, X, Y, Z\}$, $T = \{a, b\}$ și mulțimea producțiilor P este următoarea:

$$S \rightarrow X|Y|bb \quad (1)$$

$$X \rightarrow Z|aXY \quad (2)$$

$$Y \rightarrow Xa|a \quad (3)$$

$$Z \rightarrow XY|S|Zb \quad (4)$$

Pentru a construi gramatica independentă de context $G' = (N, T, S, P')$ echivalentă cu G și fără redenumiri, mai întâi determinăm toate redenumirile din P , precum și producțiile noi necesare, astfel:

Producție inițială	Producție nouă
$S \rightarrow X$	$S \rightarrow Z aXY$
$S \rightarrow Y$	$S \rightarrow Xa a$
$X \rightarrow Z$	$X \rightarrow YX Zb \emptyset$
$Z \rightarrow S$	$Z \rightarrow X Y bb$
$\overset{*}{\overrightarrow{S \Rightarrow X \Rightarrow Z}}$ $\overset{*}{\overrightarrow{S \Rightarrow Z}}$	$S \rightarrow XY S Zb$
$\overset{*}{\overrightarrow{X \Rightarrow Z \Rightarrow S}}$ $\overset{*}{\overrightarrow{X \Rightarrow S}}$	$X \rightarrow X Y bb$
$\overset{*}{\overrightarrow{X \Rightarrow Z \Rightarrow S \Rightarrow Y}}$ $\overset{*}{\overrightarrow{X \Rightarrow Y}}$	$X \rightarrow Xa a$
$\overset{*}{\overrightarrow{Z \Rightarrow S \Rightarrow X}}$ $\overset{*}{\overrightarrow{Z \Rightarrow Y}}$	$Z \rightarrow Z aXY$
$\overset{*}{\overrightarrow{Z \Rightarrow Y}}$	$Z \rightarrow Xa a$

Obținem că mulțimea producțiilor P' este următoarea:

$$S \rightarrow bb|aXY|Xa|a|XY|Zb \quad (1)$$

$$X \rightarrow aXY|XY|Zb|bb|Xa|a \quad (2)$$

$$Y \rightarrow Xa|a \quad (3)$$

$$Z \rightarrow YX|Zb|bb|aXY|Xa|a \quad (4)$$

Definiția 2.6: Fie $G = (N, T, S, P)$ o gramatică independentă de context și $X \in N$. Spunem că X este *neterminal productiv* dacă există cel puțin o derivare de forma $X \xrightarrow{*} w$, unde $w \in T^*$. În caz contrar spunem că X este *neterminal neproductiv*.

Observația 2.3: Este evident faptul că numai neterminale productive sunt utile în generarea cuvintelor din $L(G)$, deci producțiile care conțin în membrul stâng sau în cel drept neterminale neproductive pot fi eliminate din P .

Propoziția 2.4: Neterminalele productive ale unei gramatici independente de context $G = (N, T, S, P)$ se pot obține prin metoda sirului crescător de mulțimi, astfel:

$$\begin{cases} M_0 = \{X \in N \mid \exists X \rightarrow \alpha \in P \text{ cu } \alpha \in T^*\} \\ M_{n+1} = M_n \cup \{X \in N \mid \exists X \rightarrow \alpha \in P \text{ cu } \alpha \in (T \cup M_n)^*\} \end{cases}$$

Se observă că $M_0 \subset M_1 \subset \dots \subset M_n \subset M_{n+1} \subset N$. Deoarece mulțimea N este finită, rezultă că există un indice $k \geq 0$ pentru care sirul de mulțimi se stabilizează, respectiv $M_k = M_{k+1} = M_{k+2} = \dots$, deci M_k este deci mulțimea neterminalelor productive. Putem elimina acum producțiile care conțin neterminale neproductive, adică neterminalele din mulțimea $N \setminus M_k$.

Exemplul 2.7: Se consideră gramatica independentă de context $G = (N, T, S, P)$, unde $N = \{S, A, B, C, D\}$, $T = \{a, b\}$ și mulțimea producțiilor P este următoarea:

$$\begin{array}{ll} S \rightarrow AB|aBC & (1) \\ A \rightarrow BA|a & (2) \\ B \rightarrow b|AC & (3) \\ C \rightarrow AC|CB & (4) \\ D \rightarrow AD|a & (5) \end{array}$$

În continuare, aplicăm propoziția 2.4 pentru a determina neterminalele productive ale gramaticii G :

$$\begin{aligned} M_0 &= \{X \in N \mid \exists X \rightarrow \alpha \in P \text{ cu } \alpha \in T^*\} = \{A, B, D\} \\ M_1 &= M_0 \cup \{X \in N \mid \exists X \rightarrow \alpha \in P \text{ cu } \alpha \in (T \cup M_0)^*\} = \\ &= \{A, B, D\} \cup \{X \in N \mid \exists X \rightarrow \alpha \in P \text{ cu } \alpha \in \{A, B, D, a, b\}^*\} = \\ &= \{A, B, D\} \cup \{S, A, B, D\} = \{S, A, B, D\} \\ M_2 &= M_1 \cup \{X \in N \mid \exists X \rightarrow \alpha \in P \text{ cu } \alpha \in (T \cup M_1)^*\} = \\ &= \{S, A, B, D\} \cup \{X \in N \mid \exists X \rightarrow \alpha \in P \text{ cu } \alpha \in \{S, A, B, D, a, b\}^*\} = \\ &= \{S, A, B, D\} \cup \{S, A, B, D\} = \{S, A, B, D\} \end{aligned}$$

Se observă că în acest moment sirul de mulțimi s-a stabilizat deoarece $M_2 = M_1$, deci neterminalele productive sunt cele din $M_2 = \{S, A, B, D\}$. Rezultă că singurul neterminal neproductiv este C . Pentru a simplifica gramatica G , îl eliminăm pe C din N și eliminăm din P producțiile în care acesta apare, obținând astfel o gramatică independentă de context $G' = (N', T, S, P')$ echivalentă cu G și care nu conține neterminale neproductive, unde $N' = \{S, A, B, D\}$, iar și mulțimea producțiilor P' este următoarea:

$$S \rightarrow AB \quad (1)$$

$$A \rightarrow BA|a \quad (2)$$

$$B \rightarrow b|AC \quad (3)$$

$$D \rightarrow AD|a \quad (4)$$

Definiția 2.7: Fie $G = (N, T, S, P)$ o gramatică independentă de context și $X \in N \cup T$. Spunem că X este *simbol accesibil* dacă există cel puțin o derivare de forma $S \xrightarrow{*} \alpha X \beta$, unde $\alpha, \beta \in (N \cup T)^*$. În caz contrar spunem că X este *simbol inaccesibil*.

Observația 2.4: Este evident faptul că numai simbolurile accesibile sunt utile în generarea cuvintelor din $L(G)$, deci producțiile care conțin în membrul stâng sau în cel drept simboluri inaccesibile pot fi eliminate din P .

Propoziția 2.5: Simbolurile accesibile ale unei gramatici independente de context $G = (N, T, S, P)$ se pot obține prin metoda șirului crescător de mulțimi, astfel:

$$\begin{cases} M_0 = \{S\} \\ M_{n+1} = M_n \cup \{X \in N \cup T \mid \exists Y \in M_n \cap N \text{ astfel încât } Y \rightarrow \alpha X \beta \in P \text{ cu } \alpha, \beta \in (N \cup T)^*\} \end{cases}$$

Se observă că $M_0 \subset M_1 \subset \dots \subset M_n \subset M_{n+1} \subset N$. Deoarece mulțimea N este finită, rezultă că există un indice $k \geq 0$ pentru care șirul de mulțimi se stabilizează, respectiv $M_k = M_{k+1} = M_{k+2} = \dots$, deci M_k este deci mulțimea simbolurilor accesibile. Putem elimina acum producțiile care conțin simboluri inaccesibile, adică simbolurile din mulțimea $(N \cup T) \setminus M_k$.

Exemplul 2.8: Se consideră gramatica independentă de context $G = (N, T, S, P)$, unde $N = \{S, A, B, D\}$, $T = \{a, b\}$ și mulțimea producțiilor P este următoarea:

$$S \rightarrow AB \quad (1)$$

$$A \rightarrow BA|a \quad (2)$$

$$B \rightarrow b \quad (3)$$

$$D \rightarrow AD|a \quad (4)$$

Aplicăm propoziția 2.5 pentru a determina simbolurile accesibile ale gramaticii G :

$$\begin{aligned}
M_0 &= \{S\} \\
M_1 &= M_0 \cup \{X \in N \cup T \mid \exists Y \in M_0 \cap N \text{ a.î. } Y \rightarrow \alpha X \beta \in P \text{ cu } \alpha, \beta \in (N \cup T)^*\} = \\
&= \{S\} \cup \{X \in N \cup T \mid \exists Y \in \{S\} \text{ a.î. } Y \rightarrow \alpha X \beta \in P \text{ cu } \alpha, \beta \in (N \cup T)^*\} = \\
&= \{S\} \cup \{A, B\} = \{S, A, B\} \\
M_2 &= M_1 \cup \{X \in N \cup T \mid \exists Y \in M_1 \cap N \text{ a.î. } Y \rightarrow \alpha X \beta \in P \text{ cu } \alpha, \beta \in (N \cup T)^*\} = \\
&= \{S, A, B\} \cup \{X \in N \cup T \mid \exists Y \in \{S, A, B\} \text{ a.î. } Y \rightarrow \alpha X \beta \in P \text{ cu } \alpha, \beta \in (N \cup T)^*\} = \\
&= \{S, A, B\} \cup \{A, B, a, b\} = \{S, A, B, a, b\} \\
M_3 &= M_2 \cup \{X \in N \cup T \mid \exists Y \in M_2 \cap N \text{ a.î. } Y \rightarrow \alpha X \beta \in P \text{ cu } \alpha, \beta \in (N \cup T)^*\} = \\
&= \{S, A, B, a, b\} \cup \{X \in N \cup T \mid \exists Y \in \{S, A, B\} \text{ a.î. } Y \rightarrow \alpha X \beta \in P \text{ cu } \alpha, \beta \in (N \cup T)^*\} = \\
&= \{S, A, B, a, b\} \cup \{A, B, a, b\} = \{S, A, B, a, b\}
\end{aligned}$$

Se observă că în acest moment şirul de mulțimi s-a stabilizat deoarece $M_3 = M_2$, deci simbolurile accesibile sunt cele din $M_3 = \{S, A, B, a, b\}$. Rezultă că singurul simbol inaccesibil este D . Pentru a simplifica gramatica G , îl eliminăm pe D din N și eliminăm din P producțiile în care acesta apare, obținând astfel o gramatică independentă de context $G' = (N', T, S, P')$ echivalentă cu G și care nu conține simboluri inaccesibile, unde $N' = \{S, A, B\}$, iar și mulțimea producțiilor P' este următoarea:

$$\begin{aligned}
S &\rightarrow AB & (1) \\
A &\rightarrow BA | a & (2) \\
B &\rightarrow b | AC & (3)
\end{aligned}$$

2.3. Forma normală Chomsky a unei gramatici independente de context

Definiția 2.8: O gramatică independentă de context $G = (N, T, S, P)$ este în *forma normală Chomsky* dacă orice producție a sa este fie de forma $A \rightarrow a$, fie de forma $A \rightarrow BC$, unde $A, B, C \in N$ și $a \in T$.

Propoziția 2.6: Fie $G = (N, T, S, P)$ o gramatică independentă de context. Atunci există o gramatică independentă de context G' echivalentă cu G ale cărei producții sunt fie de forma $A \rightarrow a$, fie de forma $A \rightarrow B_1 B_2 \dots B_m$, unde $a \in T$, $B_i \in N$ pentru $\forall i \in \{1, 2, \dots, m\}$ și $m \geq 2$.

Demonstrație:

Bazându-ne pe propoziția anterioară, putem presupune că în G nu există redenumiri, adică producții de forma $A \rightarrow B$.

Fie T' o dublură a lui T . Evident, rezultă că există o bijecție $T \xrightarrow{f} T'$, unde $f(a) = \tilde{a}$.

Fie $G' = (N \cup T', T, S, P')$, unde pentru construcția lui P' se consideră pe rând producțiile $A \rightarrow \alpha \in P$, astfel:

- dacă $|\alpha| = 1$, atunci $\alpha \in T$ și producția este trecută în P' ;
- dacă $|\alpha| > 1$, atunci în P' se înscrie producția $A \rightarrow \tilde{\alpha}$, unde $\tilde{\alpha}$ se obține din α înlocuind terminalele cu corespondentul lor din T' .

În final se adaugă la P' producțiile $\{\tilde{\alpha} \rightarrow a | a \in T'\}$.

Teorema 2.3: Fie G o gramatică independentă de context. Atunci există o gramatică independentă de context G' în formă normală Chomsky echivalentă cu G .

Demonstrație:

Bazându-ne pe propoziția anterioară, putem presupune că toate producțiile gramaticii G sunt fie de forma $A \rightarrow a$, fie de forma $A \rightarrow B_1B_2 \dots B_m$, unde $a \in T$, $B_i \in N$ pentru $\forall i \in \{1, 2, \dots, m\}$ și $m \geq 2$.

Fie o producție $A \rightarrow B_1B_2 \dots B_m$ cu $m \geq 2$ și $A, B_1, B_2, \dots, B_m \in N$. Această producție trebuie înlocuită cu producții de forma $X \rightarrow YZ$, evident cu păstrarea limbajului generat de gramatică. Se observă că producția respectivă poate fi înlocuită cu producțiile

$$\begin{cases} A \rightarrow B_1D_1 \\ D_1 \rightarrow B_2D_2 \\ \dots \\ D_{m-2} \rightarrow B_{m-1}B_m \end{cases}$$

unde D_1, D_2, \dots, D_{m-2} sunt simboluri noi care vor fi adăugate la N , obținându-se astfel N' .

Exemplul 2.9: Fie gramatica independentă de context $G = (N, T, S, P)$, unde $N = \{S, A, B\}$, $T = \{a, b\}$, iar mulțimea producțiilor este P :

$$S \rightarrow aB|bA|A \quad (1)$$

$$A \rightarrow bAa|aS|B|a \quad (2)$$

$$B \rightarrow aBb|bS|b \quad (3)$$

Pentru a aduce gramatica G la forma normală Chomsky, vom elimina mai întâi redenumirile:

Redenumire	Producții echivalente
$S \rightarrow A$	$S \rightarrow bAa aS a$
$A \rightarrow B$	$A \rightarrow aBb bS b$
$S \xrightarrow{*} B$	$S \rightarrow aBb bS b$

Astfel, vom obține gramatica $G_1 = (N, T, S, P_1)$, fără redenumiri și echivalentă cu G , ale cărei producții sunt date de mulțimea P_1 :

$$S \rightarrow aB|bA|bAa|aS|a|aBb|bS|b \quad (1)$$

$$A \rightarrow bAa|aS|a|aBb|bS|b \quad (2)$$

$$B \rightarrow aBb|bS|b \quad (3)$$

În continuare, aplicăm propoziția 2.6 asupra gramaticii G_1 , obținând astfel o gramatică echivalentă $G_2 = (N_2, T, S, P_2)$, unde $N_2 = \{S, A, B, \tilde{A}, \tilde{B}\}$ și mulțimea producțiilor P_2 este următoarea:

$$\tilde{A} \rightarrow a \quad (1)$$

$$\tilde{B} \rightarrow b \quad (2)$$

$$S \rightarrow \tilde{A}B|\tilde{B}A|\tilde{B}A\tilde{A}|\tilde{A}S|a|\tilde{A}B\tilde{B}|\tilde{B}S|b \quad (3)$$

$$A \rightarrow \tilde{B}AA|\tilde{A}S|a|\tilde{A}B\tilde{B}|\tilde{B}S|b \quad (4)$$

$$B \rightarrow \tilde{A}B\tilde{B}|\tilde{B}S|b \quad (5)$$

În acest moment putem aplica asupra gramaticii G_2 teorema 2.3, obținând astfel o nouă gramatică echivalentă $G_3 = (N_3, T, S, P_3)$ și care este adusă la forma normală Chomsky, în care $N_3 = \{S, A, B, \tilde{A}, \tilde{B}, D_1, D_2, D_3, D_4, D_5\}$, iar mulțimea producțiilor P_3 este următoarea:

$$\tilde{A} \rightarrow a \quad (1)$$

$$\tilde{B} \rightarrow b \quad (2)$$

$$S \rightarrow \tilde{A}B|\tilde{B}A|\tilde{B}D_1|\tilde{A}S|a|\tilde{A}D_2|\tilde{B}S|b \quad (3)$$

$$D_1 \rightarrow A\tilde{A} \quad (4)$$

$$D_2 \rightarrow B\tilde{B} \quad (5)$$

$$A \rightarrow \tilde{B}D_3|\tilde{A}S|a|\tilde{A}D_4|\tilde{B}S|b \quad (6)$$

$$D_3 \rightarrow A\tilde{A} \quad (7)$$

$$D_4 \rightarrow B\tilde{B} \quad (8)$$

$$B \rightarrow \tilde{A}D_5|\tilde{B}S|b \quad (9)$$

$$D_5 \rightarrow B\tilde{B} \quad (10)$$

Exemplul 2.10: Fie gramatica independentă de context $G = (N, T, S, P)$, unde $N = \{S, A, B\}$, $T = \{a, b\}$, iar mulțimea producțiilor este P :

$$S \rightarrow aABA \quad (1)$$

$$A \rightarrow AaA|AB|a \quad (2)$$

$$B \rightarrow BbB|BAb|b \quad (3)$$

Deoarece gramatica G nu conține redenumiri, putem să aplicăm propoziția 2.6 asupra sa, obținând astfel o gramatică echivalentă $G_1 = (N_1, T, S, P_1)$, unde $N_1 = \{S, A, B, \tilde{A}, \tilde{B}\}$ și mulțimea producțiilor P_1 este următoarea:

$$S \rightarrow \tilde{A}ABA \quad (1)$$

$$A \rightarrow A\tilde{A}|AB|a \quad (2)$$

$$B \rightarrow B\tilde{B}B|BA\tilde{B}|b \quad (3)$$

$$\tilde{A} \rightarrow a \quad (4)$$

$$\tilde{B} \rightarrow b \quad (5)$$

În acest moment putem aplica asupra gramaticii G_1 teorema 2.3, obținând astfel o nouă gramatică echivalentă $G_2 = (N_2, T, S, P_2)$ și care este adusă la forma normală Chomsky, în care $N_2 = \{S, A, B, \tilde{A}, \tilde{B}, X_1, X_2, X_3, X_4, X_5\}$, iar mulțimea producțiilor P_2 este următoarea:

$$S \rightarrow \tilde{A}X_1 \quad (1)$$

$$X_1 \rightarrow AX_2 \quad (2)$$

$$X_2 \rightarrow BA \quad (3)$$

$$A \rightarrow AX_3 \quad (4)$$

$$X_3 \rightarrow \tilde{A}A \quad (5)$$

$$A \rightarrow AB \quad (6)$$

$$A \rightarrow a \quad (7)$$

$$B \rightarrow BX_4 \quad (8)$$

$$X_4 \rightarrow \tilde{B}B \quad (9)$$

$$B \rightarrow BX_5 \quad (10)$$

$$X_5 \rightarrow A\tilde{B} \quad (11)$$

$$B \rightarrow b \quad (12)$$

$$\tilde{A} \rightarrow a \quad (13)$$

$$\tilde{B} \rightarrow b \quad (14)$$

Exemplul 2.11: Fie gramatica independentă de context $G = (N, T, S, P)$, unde $N = \{S, A, B, C, D, E\}$, $T = \{a, b\}$, iar mulțimea producțiilor este P :

$$S \rightarrow aB|AC \quad (1)$$

$$A \rightarrow ASC|BC|aD|a \quad (2)$$

$$B \rightarrow bS|b \quad (3)$$

$$C \rightarrow BA|\lambda \quad (4)$$

$$D \rightarrow abC \quad (5)$$

$$E \rightarrow aB \quad (6)$$

În continuare, vom simplifica gramatica G prin eliminarea λ -producțiilor, a simbolurilor neproductive și a celor inaccesibile, iar apoi o vom aduce la forma normală Chomsky.

a) *Eliminarea λ -producțiilor din gramatica G :*

Se realizează aplicând propoziția 2.2. Determinăm mai întâi mulțimea $M = \{A \in N \mid A *_{\lambda \in P} = \mathcal{C}\}$.

Producții inițiale	Producții noi
$S \rightarrow AC$	$S \rightarrow A$
$A \rightarrow ASC$	$A \rightarrow AS$
$A \rightarrow BC$	$A \rightarrow B$
$D \rightarrow abC$	$D \rightarrow ab$

Gramatica G este echivalentă cu gramatica independentă de context $G_1 = (N, T, S, P_1)$ și care nu conține λ -producții, în care mulțimea producțiilor P_1 este următoarea:

$$S \rightarrow aB|AC|A \quad (1)$$

$$A \rightarrow ASC|BC|aD|AS|B|a \quad (2)$$

$$B \rightarrow bS|b \quad (3)$$

$$C \rightarrow BA \quad (4)$$

$$D \rightarrow abC|ab \quad (5)$$

$$E \rightarrow aB \quad (6)$$

b) *Eliminarea simbolurilor neproductive din gramatica G_1 :*

În continuare, aplicăm propoziția 2.4 pentru a determina neterminalele productive ale gramaticii G_1 :

$$M_0 = \{X \in N \mid \exists X \rightarrow \alpha \in P \text{ cu } \alpha \in T^*\} = \{A, B, D\}$$

$$\begin{aligned} M_1 &= M_0 \cup \{X \in N \mid \exists X \rightarrow \alpha \in P \text{ cu } \alpha \in (T \cup M_0)^*\} = \\ &= \{A, B, D\} \cup \{X \in N \mid \exists X \rightarrow \alpha \in P \text{ cu } \alpha \in \{A, B, D, a, b\}^*\} = \\ &= \{A, B, D\} \cup \{S, A, B, C, D, E\} = \{S, A, B, C, D, E\} \end{aligned}$$

Deoarece $M_1 = N \Rightarrow$ algoritmul se termină, deoarece gramatica G_1 nu are simboluri neproductive.

c) *Eliminarea simbolurilor inaccesibile din gramatica G_1 :*

Aplicăm propoziția 2.5 pentru a determina simbolurile accesibile ale gramaticii G_1 :

$$\begin{aligned}
M_0 &= \{S\} \\
M_1 &= M_0 \cup \{X \in N \cup T \mid \exists Y \in M_0 \cap N \text{ a. i. } Y \rightarrow \alpha X \beta \in P\} = \\
&= \{S\} \cup \{X \in N \cup T \mid \exists Y \in \{S\} \text{ a. i. } Y \rightarrow \alpha X \beta \in P\} = \\
&= \{S\} \cup \{A, B\} = \{S, A, B, C, a\} \\
M_2 &= M_1 \cup \{X \in N \cup T \mid \exists Y \in M_1 \cap N \text{ a. i. } Y \rightarrow \alpha X \beta \in P\} = \\
&= \{S, A, B, C, a\} \cup \{X \in N \cup T \mid \exists Y \in \{S, A, B, C\} \text{ a. i. } Y \rightarrow \alpha X \beta \in P\} = \\
&= \{S, A, B, C, a\} \cup \{a, A, B, C, S, D, b\} = \{S, A, B, C, D, a, b\} \\
M_3 &= M_2 \cup \{X \in N \cup T \mid \exists Y \in M_2 \cap N \text{ a. i. } Y \rightarrow \alpha X \beta \in P\} = \\
&= \{S, A, B, C, D, a, b\} \cup \{X \in N \cup T \mid \exists Y \in \{S, A, B, C, D\} \text{ a. i. } Y \rightarrow \alpha X \beta \in P\} = \\
&= \{S, A, B, C, D, a, b\}
\end{aligned}$$

Se observă că în acest moment sirul de multimi s-a stabilizat deoarece $M_3 = M_2$, deci simbolurile accesibile sunt cele din $M_3 = \{S, A, B, C, D, a, b\}$. Rezultă că singurul simbol inaccesibil este E . Pentru a simplifica gramatica G_1 , îl eliminăm pe E din N_1 și eliminăm din P_1 producțiile în care acesta apare, obținând astfel o gramatică independentă de context $G_2 = (N_2, T, S, P_2)$ echivalentă cu G_1 și care nu conține simboluri inaccesibile, unde $N_2 = \{S, A, B, C, D\}$, iar și multimea producțiilor P_2 este următoarea:

$$S \rightarrow aB|AC|A \quad (1)$$

$$A \rightarrow ASC|BC|aD|AS|B|a \quad (2)$$

$$B \rightarrow bS|b \quad (3)$$

$$C \rightarrow BA \quad (4)$$

$$D \rightarrow abC|ab \quad (5)$$

d) *Eliminarea redenumirilor din gramatica G_2 :*

Pentru a construi gramatica independentă de context $G_3 = (N_2, T, S, P_3)$ echivalentă cu G_2 și fără redenumiri, mai întâi determinăm toate redenumirile din P_2 , precum și producțiile noi necesare, astfel:

Redenumiri	Producții noi
$S \rightarrow A$	$S \rightarrow aAS$ $S \rightarrow BC$ $S \rightarrow ASC$ $S \rightarrow aD$
$A \rightarrow B$	$A \rightarrow b$ $A \rightarrow bS$
$S \xrightarrow{*} B$	$S \rightarrow b$ $S \rightarrow bS$

Astfel, obținem că mulțimea producțiilor P_3 este următoarea:

$$S \rightarrow aB|AC|ASC|BC|aD|AS|bS|a|b \quad (1)$$

$$A \rightarrow ASC|BC|aD|AS|bS|a|b \quad (2)$$

$$B \rightarrow bS|b \quad (3)$$

$$C \rightarrow BA \quad (4)$$

$$D \rightarrow abC|ab \quad (5)$$

e) Aducerea gramaticii G_3 la forma normală Chomsky:

Aplicăm propoziția 2.6 asupra gramaticii G_3 , obținând astfel o gramatică echivalentă $G_4 = (N_4, T, S, P_4)$, unde $N_4 = \{S, A, B, C, D, \tilde{A}, \tilde{B}\}$ și mulțimea producțiilor P_4 este următoarea:

$$\tilde{A} \rightarrow a \quad (1)$$

$$\tilde{B} \rightarrow b \quad (2)$$

$$S \rightarrow \tilde{A}B|AC|ASC|BC|\tilde{A}D|AS|\tilde{B}S|a|b \quad (3)$$

$$A \rightarrow ASC|BC|\tilde{A}D|AS|\tilde{B}S|a|b \quad (4)$$

$$B \rightarrow \tilde{B}S|b \quad (5)$$

$$C \rightarrow BA \quad (6)$$

$$D \rightarrow \tilde{A}\tilde{B}C|\tilde{A}\tilde{B} \quad (7)$$

În acest moment putem aplica asupra gramaticii G_4 teorema 2.3, obținând astfel o nouă gramatică echivalentă $G_5 = (N_5, T, S, P_5)$ și care este adusă la forma normală Chomsky, unde $N_5 = \{S, A, B, C, D, \tilde{A}, \tilde{B}, X_1, X_2\}$, iar mulțimea producțiilor P_5 este următoarea:

$$\tilde{A} \rightarrow a \quad (1)$$

$$\tilde{B} \rightarrow b \quad (2)$$

$$S \rightarrow \tilde{A}B|AC|AX_1|BC|\tilde{A}D|AS|\tilde{B}S|a|b \quad (3)$$

$$X_1 \rightarrow SC \quad (4)$$

$$A \rightarrow AX_1|BC|\tilde{A}D|AS|\tilde{B}S|a|b \quad (5)$$

$$B \rightarrow \tilde{B}S|b \quad (6)$$

$$C \rightarrow BA \quad (7)$$

$$D \rightarrow \tilde{A}X_2|\tilde{A}\tilde{B} \quad (8)$$

$$X_2 \rightarrow \tilde{B}C \quad (9)$$

2.4. Arbori de derivare

Arborii de derivare reprezintă o metodă vizuală de descriere a unei derivări într-o gramatică independentă de context.

Definiția 2.9: Fie $G = (N, T, S, P)$ o gramatică independentă de context. Un *arbore de derivare* în gramatica G este un arbore situat pe niveluri cu următoarele proprietăți:

1. rădăcina este etichetată cu S ;
2. vârfurile interne sunt etichetate cu neterminale;
3. frunzele sunt etichetate cu terminale;
4. dacă vârfurile v_1, \dots, v_k (etichetate cu A_1, \dots, A_k) sunt în ordine de la stânga la dreapta descendenții direcți ai vârfului v (etichetat cu A), atunci $A \rightarrow A_1 \dots A_k \in P$.

Exemplul 2.12: Fie gramatica independentă de context $G = (N, T, S, P)$, unde $N = \{S, A\}$, $T = \{a, b\}$, iar mulțimea producțiilor este următoarea:

$$S \rightarrow aAS \quad (1)$$

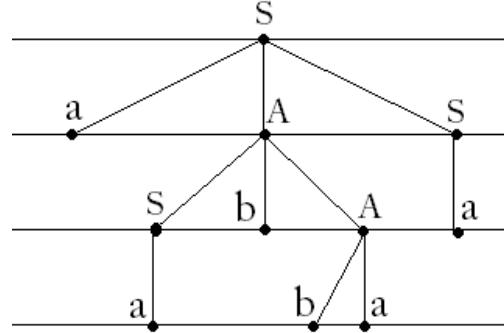
$$S \rightarrow a \quad (2)$$

$$A \rightarrow SbA \quad (3)$$

$$A \rightarrow SS \quad (4)$$

$$A \rightarrow ba \quad (5)$$

Un arbore de derivare în gramatica G este următorul:



Se observă ușor că acest arbore corespunde următoarei derivări:

$$S \xrightarrow{*} aAS \xrightarrow{*} aSbAS \xrightarrow{*} aSbAA \xrightarrow{*} aabAA \xrightarrow{*} aabbAA = a^2b^2a^2$$

Pentru două vârfuri terminale v_1 și v_2 dintr-un arbore de derivare punem în evidență drumurile care le leagă de rădăcină și vârful v din care cele două drumuri se despart. Spunem că v_1 este la stânga lui v_2 dacă drumul de la v la v_1 se află la stânga celui de la v la v_2 .

Definiția 2.10: Frontiera unui arbore de derivare este cuvântul format din etichetele vârfurilor terminale, în ordine de la stânga la dreapta.

Exemplul 2.13: Frontiera arborelui de derivare anterior este $a^2b^2a^2$.

Observația 2.5: Fie $G = (N, T, S, P)$ o gramatică independentă de context și fie $A \in N$. Vom nota cu $G_A = (N, T, A, P)$ gramatica independentă de context obținută din gramatica G prin înlocuirea simbolului inițial S cu A .

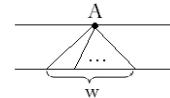
Teorema 2.4: Fie $G = (N, T, S, P)$ o gramatică independentă de context și fie $w \in T^*$. Atunci $w \in L(G)$ dacă și numai dacă în G există un arbore de derivare cu frontiera w .

Demonstrație:

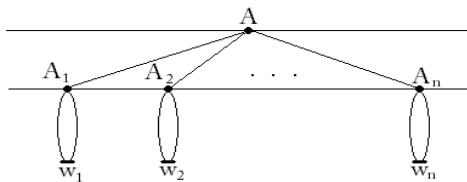
Pentru orice $A \in N$ arătăm că $A \xrightarrow{*} w$ dacă și numai dacă în G_A există un arbore de derivare cu frontiera w .

" \Rightarrow " Inducție după lungimea k a derivării:

- Pentru $k = 1 : A \rightarrow w \in P$ și arborele de derivare este următorul:



- Presupunem adevărat pentru orice $A \in N$ și orice derivare de lungime k . Fie derivarea $A \xrightarrow{*} A_1 A_2 \dots A_n \xrightarrow{k \text{ pași}} w$. Atunci $w = w_1 w_2 \dots w_k$ cu $A_i \xrightarrow{\leq k \text{ pași}} w_i$. Conform ipotezei de inducție, vor exista în G_{A_i} arbori de derivare cu frontierele W_i , iar arborele căutat va fi:



" \Leftarrow " Inducție după $k =$ numărul vârfurilor neterminale.

- Pentru $k = 1 :$ și $A \rightarrow A_1 \dots A_n \in P$

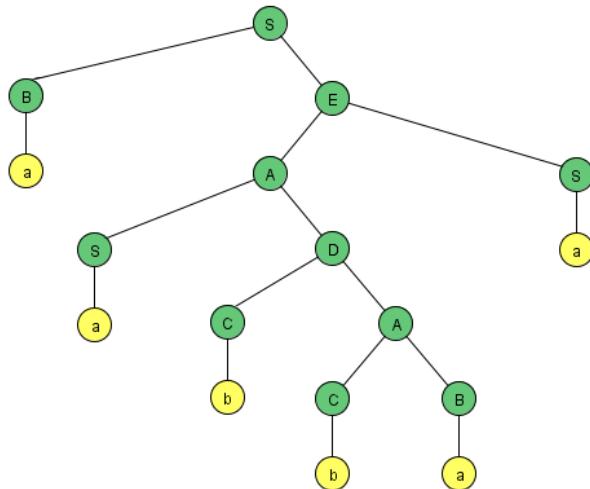
- Presupunem afirmația adevărată pentru $\forall A \in N$ și orice arbore G_A care are cel mult k vârfuri neterminale. Fie în G_A un arbore de derivare cu $k+1$ vârfuri neterminale. Fie A_1, A_2, \dots, A_n descendenții direcți ai lui $A \Rightarrow A \rightarrow A_1 \dots A_n \in P$. Pentru $\forall i \in \overline{1, n}$ considerăm arborele de rădăcină A_i și fie w_i frontiera sa. Cum fiecare dintre acești arbori are cel mult k vârfuri neterminale, rezultă că $A_i \xrightarrow{*} w_i$ (dacă $A_i \in T$, atunci $A_i = w_i$). Obținem acum că $A \Rightarrow A_1 \dots A_n \xrightarrow{*} w_1 \dots w_n = w$.

Observația 2.6: Dacă o gramatică independentă de context G este în formă normală Chomsky, atunci orice arbore de derivare este arbore binar.

Exemplul 2.14: Gramatica independentă de context G din exemplul 2.10 este echivalentă cu gramatica independentă de context în formă normală Chomsky $G' = (N', T, S, P')$, unde $N' = \{S, A, B, C, D, E\}$, $T = \{a, b\}$, iar mulțimea producțiilor P' este următoarea:

- | | |
|--------------------|-----|
| $S \rightarrow BE$ | (1) |
| $E \rightarrow AS$ | (2) |
| $S \rightarrow a$ | (3) |
| $A \rightarrow SD$ | (4) |
| $D \rightarrow CA$ | (5) |
| $A \rightarrow SS$ | (6) |
| $A \rightarrow CB$ | (7) |
| $C \rightarrow b$ | (8) |
| $B \rightarrow a$ | (9) |

Evident, $w = aabbaa \in L(G') = L(G)$, iar un arbore de derivare binar în gramatica G' corespunzător cuvântului w este următorul:



Definiția 2.10: O gramatică independentă de context G se numește *gramatica ambiguă*, dacă un cuvânt $w \in L(G)$ se poate obține prin cel puțin două derivări stângi din S distincte.

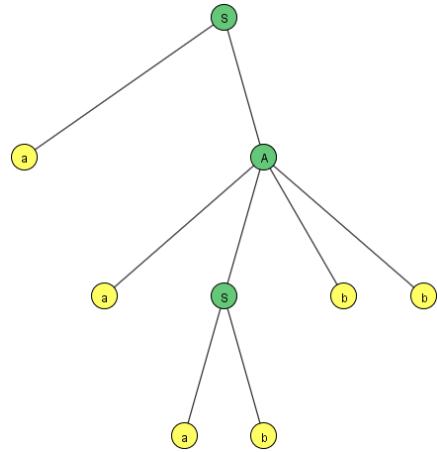
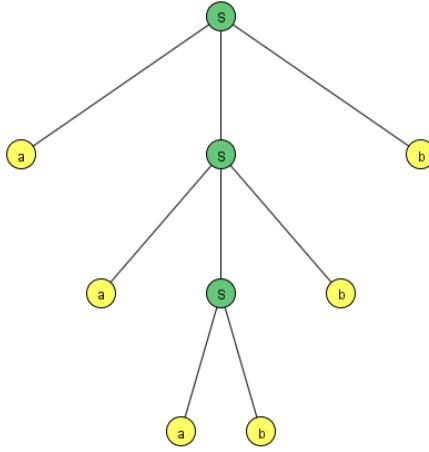
Exemplul 2.15: Considerăm următoarea gramatică independentă de context $G = (N, T, S, P)$, unde $N = \{S, A\}$, $T = \{a, b\}$, iar mulțimea producțiilor este următoarea:

- | | |
|----------------------|-----|
| $S \rightarrow ab$ | (1) |
| $S \rightarrow aSb$ | (2) |
| $S \rightarrow aA$ | (3) |
| $A \rightarrow aSbb$ | (4) |

Gramatica G este ambiguă, deoarece cuvântul $w = aaabbba \in T^*$ poate fi obținut prin două derivări stângi distincte din simbolul inițial S , după cum se poate cu ușurință observa din figurile de mai jos:

$$S \xrightarrow[\substack{(2) \\ (2)}]{} aSb \xrightarrow[\substack{(2) \\ (2)}]{} aaSbb \xrightarrow[\substack{(1) \\ (1)}]{} aaabbb$$

$$S \xrightarrow[\substack{(3) \\ (4)}]{} aA \xrightarrow[\substack{(4) \\ (4)}]{} aaSbb \xrightarrow[\substack{(1) \\ (1)}]{} aaabbb$$



Exemplul 2.16: Considerăm următoarea gramatică independentă de context $G = (N, T, S, P)$, unde $N = \{S\}$, $T = \{+, *\}, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, iar mulțimea producțiilor este următoarea:

$$S \rightarrow 0|1|2|3|4|5|6|7|8|9 \quad (1)$$

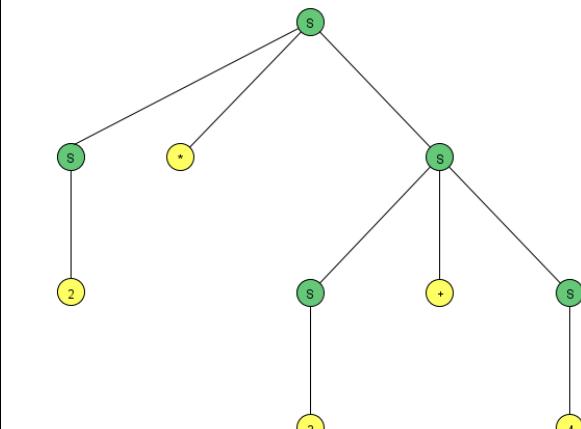
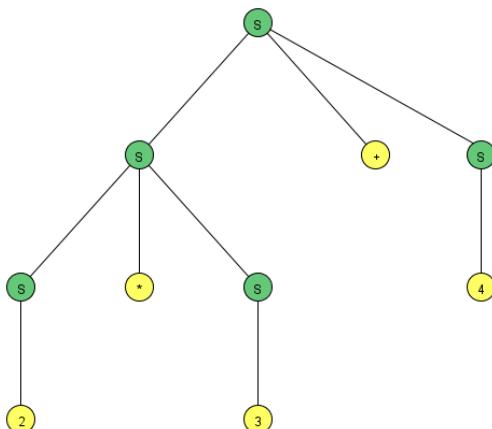
$$S \rightarrow S + S \quad (2)$$

$$S \rightarrow S * S \quad (3)$$

Gramatica G este ambiguă, deoarece cuvântul $w = 2 * 3 + 4 \in T^*$ poate fi obținut prin două derivări stângi distincte din simbolul inițial S , după cum se poate cu ușurință observa din figurile de mai jos:

$$\begin{aligned} S &\xrightarrow[\substack{(1) \\ (1)}]{} S + S \xrightarrow[\substack{(2) \\ (2)}]{} S + S * S \xrightarrow[\substack{(3) \\ (3)}]{} 2 * S + S \xrightarrow[\substack{(3) \\ (3)}]{} \\ &\qquad\qquad\qquad \Rightarrow 2 * 3 + S \xrightarrow[\substack{(3) \\ (3)}]{} 2 * 3 + 4 \end{aligned}$$

$$\begin{aligned} S &\xrightarrow[\substack{(2) \\ (2)}]{} S * S \xrightarrow[\substack{(3) \\ (3)}]{} 2 * S \xrightarrow[\substack{(1) \\ (1)}]{} 2 * S + S \xrightarrow[\substack{(3) \\ (3)}]{} \\ &\qquad\qquad\qquad \Rightarrow 2 * 3 + S \xrightarrow[\substack{(3) \\ (3)}]{} 2 * 3 + 4 \end{aligned}$$



Practic, ambiguitatea gramaticii G ne indică faptul că expresia $2 * 3 + 4$ poate fi evaluată în două moduri:

$$2 * 3 + 4 = \begin{cases} (2 * 3) + 4 = 6 + 4 = 10 \\ 2 * (3 + 4) = 2 * 7 = 14 \end{cases}$$

Eliminarea acestei ambiguități din procesul de evaluare a unei expresii aritmetice se poate realiza fie prin utilizarea parantezelor, fie prin asocierea unor priorități operatorilor aritmetici.

2.5. Algoritmul Cocke-Younger-Kasami (1965)

Algoritmul Cocke-Younger-Kasami (CYK) folosește metoda programării dinamice pentru a decide dacă un cuvânt aparține sau nu limbajului generat de o gramatică independentă de context în formă normală Chomsky.

Algoritmul se bazează pe descompunerea cuvântului dat în subșiruri de lungimi din ce în ce mai mari și găsirea simbolurilor neterminale din care poate fi obținut subșirul respectiv. Complexitatea algoritmului este $O(n^3)$.

Fie un cuvânt $w = a_1 a_2 \dots a_n \in T^*$. Notăm prin $w_{i,j} = a_i a_{i+1} \dots a_{i+j-1}$, respectiv subșirul din cuvântul w care începe pe poziția i ($1 \leq i \leq n$) și are lungimea j ($1 \leq j \leq n+1-i$). Definim o mulțime $A_{i,j}$ formată din neterminalele gramaticii $G = (N, T, S, P)$ din care poate fi derivat cuvântul $w_{i,j}$, deci $A_{i,j} = \{X \in N \mid X \xrightarrow{*} w_{i,j}\}$.

Detaliind, definim mulțimile $A_{i,j}$ astfel:

$$\begin{aligned} A_{i,1} &= \{X \in N \mid X \rightarrow a_i \in P\}, \forall i \in \{1, \dots, n\} \\ A_{i,j} &= \bigcup_{\substack{2 \leq j \leq n \\ 1 \leq i \leq n+1-j \\ 1 \leq k \leq j-1}} \{X \in N \mid \exists Y \in A_{i,k}, \exists Z \in A_{i+k, j-k} \text{ a. î. } X \rightarrow YZ \in P\} \end{aligned}$$

Evident, mulțimile $A_{i,j}$ formează o matrice triunghiular superioară, iar cuvântul $w \in L(G) \Leftrightarrow S \xrightarrow{*} w \Leftrightarrow S \in A_{1,n}$.

Exemplul 2.17: Fie gramatica independentă de context $G = (N, T, S, P)$, unde $N = \{S, D\}$, $T = \{a, b\}$ și mulțimea producțiilor P este următoarea:

$$S \rightarrow aSb | aDb \quad (1)$$

$$D \rightarrow aD | a \quad (2)$$

Gramatica independentă de context echivalentă cu G și aflată în formă normală Chomsky este $G' = (N', T, S, P')$, unde $N = \{S, A, B, C, D\}$, $T = \{a, b\}$ și mulțimea producțiilor P este următoarea:

$$S \rightarrow AB|AC \quad (1)$$

$$A \rightarrow a \quad (2)$$

$$B \rightarrow SB|b \quad (3)$$

$$C \rightarrow DB \quad (4)$$

$$D \rightarrow AD|a \quad (5)$$

Aplicăm algoritmul CYK pentru a verifica dacă $w_1 = aaaabb \in L(G)$, deci $n = |w| = 6$.

i j	1	2	3	4	5	6
1	{ D, A }	{ D, A }	{ D, A }	{ D, A }	{ B }	{ B }
2	{ D }	{ D }	{ D }	{ S, C }	\emptyset	
3	{ D }	{ D }	{ S, C }	{ B }		
4	{ D }	{ S, C }	{ S, B, C }			
5	{ S, C }	{ S, B, C }				
6	{ S, B, C }					

Deoarece $S \in A_{1,6} \Rightarrow S \xrightarrow{*} w_1 \Rightarrow w_1 = aaaabb \in L(G)$.

Aplicăm algoritmul CYK pentru a verifica dacă $w_2 = aabaabbba \in L(G)$, deci $n = 9$.

i j	1	2	3	4	5	6	7	8	9
1	{ D, A }	{ D, A }	{ B }	{ D, A }	{ D, A }	{ B }	{ B }	{ B }	{ D, A }
2	{ D }	{ S, C }	\emptyset	{ D }	{ S, C }	\emptyset	\emptyset	\emptyset	
3	{ S, C }	\emptyset	\emptyset	{ S, C }	{ B }	\emptyset	\emptyset		
4	\emptyset	\emptyset	\emptyset	{ S, B, C }	\emptyset	\emptyset			
5	\emptyset	\emptyset	\emptyset	{ B }	\emptyset				
6	\emptyset	{ B }	\emptyset	\emptyset					
7	{ S, B, C }	{ B }	\emptyset						
8	{ S, B, C }	\emptyset							
9	\emptyset								

Deoarece $S \notin A_{1,9} \Rightarrow w_2 = aabaabbba \notin L(G)$.

2.6. Exerciții propuse

1. Fie gramatica independentă de context $G = (N, T, S, P)$, unde $N = \{S, X, Y\}$, $T = \{a, b\}$ și mulțimea producțiilor P este următoarea:

$$S \rightarrow X \quad (1)$$

$$X \rightarrow XY|aYX|a|\lambda \quad (2)$$

$$Y \rightarrow X|XYb|b \quad (3)$$

- a) Desenați un arbore de derivare în G pentru un cuvânt de lungime cel puțin 5, indicând și cuvântul corespunzător arborelui respectiv.
 - b) Simplificați gramatica G .
 - c) Aduceți gramatica G la forma normală Chomsky.
2. Fie gramatica independentă de context $G = (N, T, S, P)$, unde $N = \{S, X, Y\}$, $T = \{a, b\}$ și mulțimea producțiilor P este următoarea:
- $$S \rightarrow X|Y|XY \quad (1)$$
- $$X \rightarrow aXYX|bXY|Y|a \quad (2)$$
- $$Y \rightarrow XYXY|aXYb|b|\lambda \quad (3)$$
- a) Desenați un arbore de derivare în G pentru un cuvânt de lungime cel puțin 7, indicând și cuvântul corespunzător arborelui respectiv.
 - b) Simplificați gramatica G .
 - c) Aduceți gramatica G la forma normală Chomsky.
3. Fie gramatica independentă de context $G = (N, T, S, P)$, unde $N = \{S, A, B, C\}$, $T = \{a, b, c\}$ și mulțimea producțiilor P este următoarea:
- $$S \rightarrow A|B|aA|bB \quad (1)$$
- $$A \rightarrow aAa|bAb|B|\lambda \quad (2)$$
- $$B \rightarrow bBb|aBa|A|\lambda \quad (3)$$
- $$C \rightarrow cCc|aABb|\lambda \quad (4)$$
- a) Desenați un arbore de derivare în G pentru un cuvânt de lungime cel puțin 7, indicând și cuvântul corespunzător arborelui respectiv.
 - b) Simplificați gramatica G .
 - c) Aduceți gramatica G la forma normală Chomsky.
4. Fie gramatica independentă de context $G = (N, T, S, P)$, unde $N = \{S, A, B, C\}$, $T = \{a, b\}$ și mulțimea producțiilor P este următoarea:

$$S \rightarrow AB \quad (1)$$

$$A \rightarrow BC|a \quad (2)$$

$$B \rightarrow AC|b \quad (3)$$

$$C \rightarrow a|b \quad (4)$$

- a) Folosind algoritmul CYK arătați că $w_1 = abaababb \in L(G)$ și $w_2 = babaab \in L(G)$.
b) Folosind algoritmul CYK arătați că $w_3 = baaaa \notin L(G)$ și $w_4 = aaaabaaa \notin L(G)$.

5. Fie gramatica independentă de context $G = (N, T, S, P)$, unde $N = \{S, A, B, C\}$, $T = \{a, b\}$ și mulțimea producțiilor P este următoarea:

$$S \rightarrow AB|BC \quad (1)$$

$$A \rightarrow BA|a \quad (2)$$

$$B \rightarrow CC|b \quad (3)$$

$$C \rightarrow AB|a \quad (4)$$

- a) Folosind algoritmul CYK arătați că $w_1 = baaba \in L(G)$ și $w_2 = babaab \in L(G)$.
b) Folosind algoritmul CYK arătați că $w_3 = abaaba \notin L(G)$ și $w_4 = abababab \notin L(G)$.

6. Fie gramatica $G = (N, T, S, P)$, unde $N = \{S\}$, $T = \{(), \}\},$ iar mulțimea producțiilor P este următoarea:

$$S \rightarrow () \quad (1)$$

$$S \rightarrow (S) \quad (2)$$

$$S \rightarrow SS \quad (3)$$

- a) Aduceți gramatica G la forma normală Chomsky.
b) Folosind algoritmul CYK arătați că $w_1 = (())(()()) \in L(G)$, iar $w_2 = ())(() \notin L(G)$.

MODULUL III

Gramatici regulate. Automate finite deterministe și nedeterministe

3.1. Gramatici regulate

Definiția 3.1: O gramatică $G = (N, T, S, P)$ este *gramatică regulată* dacă orice producție a sa este fie de forma $A \rightarrow aB$, fie de forma $A \rightarrow a$ cu $A, B \in N$ și $a \in T$.

Exemplul 3.1: Fie gramatica regulată $G = (N, T, S, P)$, unde $N = \{S\}$, $T = \{a\}$, iar mulțimea producțiilor este $P = \{S \rightarrow a|aS\}$. Se observă foarte ușor că limbajul generat de gramatica G este $L(G) = \{a^n | n \geq 1\}$.

Exemplul 3.2: Fie gramatica regulată $G = (N, T, S, P)$, unde $N = \{S, A\}$, $T = \{a, b, c\}$, iar mulțimea producțiilor P este următoarea:

$$\begin{array}{ll} S \rightarrow aS & (1) \\ S \rightarrow bA & (2) \\ A \rightarrow cA & (3) \\ A \rightarrow c & (4) \end{array}$$

Cuvântul $w_1 = abc \in L(G)$ deoarece:

$$S \xrightarrow[(1)]{} aS \xrightarrow[(2)]{} abA \xrightarrow[(4)]{} abc = w_1 \in L(G)$$

Cuvântul $w_2 = aaabcccc \in L(G)$ deoarece:

$$S \xrightarrow[(1)]{} aS \xrightarrow[(1)]{} aaS \xrightarrow[(1)]{} aaaS \xrightarrow[(2)]{} aaabA \xrightarrow[(3)]{} aaabcA \xrightarrow[(3)]{} aaabccA \xrightarrow[(3)]{} aaabcccA \xrightarrow[(4)]{} aaabcccc = w_2 \in L(G)$$

Se poate observa că limbajul generat de gramatica G este $L(G) = \{a^nbc^m | n \geq 1, m \geq 2\}$.

Exemplul 3.3: Fie gramatica regulată $G = (N, T, S, P)$, unde $N = \{S, A\}$, $T = \{a, b, c\}$, iar mulțimea producțiilor P este următoarea:

$$\begin{array}{ll} S \rightarrow aA & (1) \\ A \rightarrow aA|aB & (2) \\ B \rightarrow bC & (3) \\ C \rightarrow cB|c & (4) \end{array}$$

Cuvântul $w = aabc \in L(G)$ deoarece:

$$S \xrightarrow[(1)]{} aA \xrightarrow[(2)]{} aaB \xrightarrow[(3)]{} aabC \xrightarrow[(4)]{} aabc = w \in L(G)$$

Se poate observa că limbajul generat de gramatica G este $L(G) = \{a^n(bc)^m | n \geq 2, m \geq 1\}$.

3.2. Automate finite deterministe

Definiția 3.2: Se numește *automat finit deterministic* un cvintuplu $A = (\Sigma, Q, \delta, s_0, F)$, unde:

- Σ se numește *alfabetul de intrare*;
- Q se numește *mulțimea stărilor*;
- $\delta: Q \times \Sigma \rightarrow Q$ se numește *funcția de tranziție*;
- $q_0 \in Q$ reprezintă *starea inițială*;
- $F \subseteq Q$ se numește *mulțimea stărilor finale*.

Observația 3.1: Extindem funcția de tranziție $\delta: Q \times \Sigma \rightarrow Q$ la o funcție $\bar{\delta}: Q \times \Sigma^* \rightarrow Q$ care să poată fi aplicată și unui întreg cuvânt, ci nu numai unui singur simbol, astfel:

$$\begin{cases} \bar{\delta}(q, \lambda) = q, \forall q \in Q \\ \bar{\delta}(q, wa) = \bar{\delta}(\bar{\delta}(q, w), a), \forall q \in Q, w \in \Sigma^*, a \in \Sigma \end{cases}$$

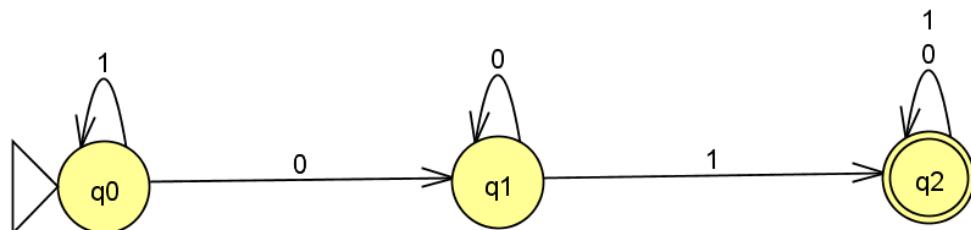
Observația 3.2: Pentru a nu complica inutil notațiile, în continuare vom nota $\bar{\delta}$ tot prin δ .

Definiția 3.3: Limbajul acceptat de un automat finit deterministic $A = (\Sigma, Q, \delta, q_0, F)$ este mulțimea $\mathcal{T}(A) = \{w \in \Sigma^* | \delta(q_0, w) \in F\}$

Automatele finit deterministe pot fi reprezentate prin două metode:

- 1) *analitic*: se vor preciza mulțimile Σ și Q , iar funcția de tranziție δ va fi dată sub forma unui tabel în care liniile reprezintă stările automatului, coloanele reprezintă simbolurile din alfabetul de intrare. Starea inițială va fi marcată cu \rightarrow , iar stările finale vor fi marcate cu $*$.
- 2) *grafic*: sub forma unui graf orientat în care un nod al grafului reprezintă o stare a automatului, un arc dintre două stări s_1 și s_2 este etichetat cu simbolul a care realizează tranziția respectivă $\delta(s_1, a) = s_2$. Starea inițială se marchează cu o săgeată, iar o stare finală se reprezintă printr-o încercuire dublă.

Exemplul 3.4: Automatul finit deterministic $A = (\Sigma, Q, \delta, q_0, F)$ acceptă toate sirurile binare care conțin subșirul 01, respectiv $w_1 = 110110 \in \mathcal{T}(A)$, iar $w_2 = 1100 \notin \mathcal{T}(A)$.



Analitic, automatul $A = (\Sigma, Q, \delta, q_0, F)$ va fi reprezentat astfel:

- $\Sigma = \{0,1\}$
- $Q = \{q_0, q_1, q_2\}$
- funcția de tranziție δ :

δ	0	1
\rightarrow	q_0	q_1
	q_1	q_2
*	q_2	q_2

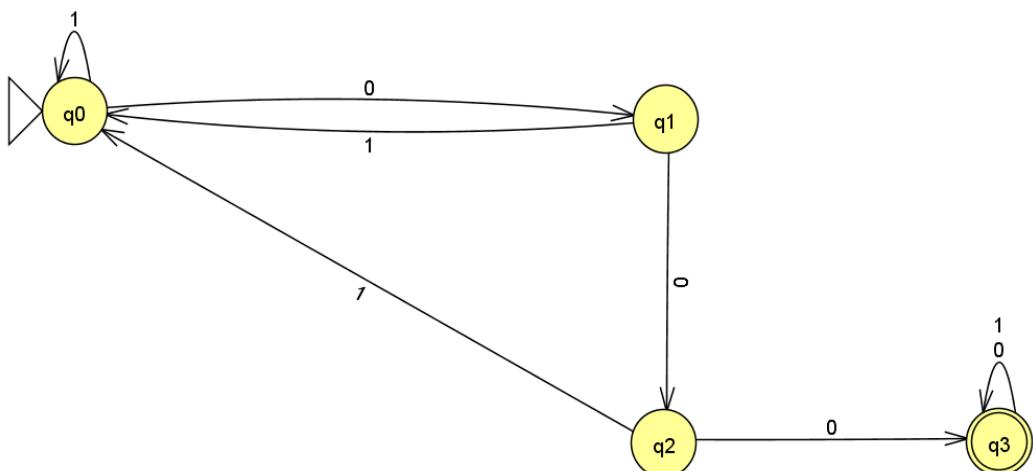
Folosind definiția 3.3 arătăm că $w_1 = 110110 \in \mathcal{T}(A)$:

$$\begin{aligned}
 \delta(q_0, 1) &= q_0 \\
 \delta(q_0, 11) &= \delta(\delta(q_0, 1), 1) = \delta(q_0, 1) = q_0 \\
 \delta(q_0, 110) &= \delta(\delta(q_0, 11), 0) = \delta(q_0, 0) = q_1 \\
 \delta(q_0, 1101) &= \delta(\delta(q_0, 110), 1) = \delta(q_1, 1) = q_2 \\
 \delta(q_0, 11011) &= \delta(\delta(q_0, 1101), 1) = \delta(q_2, 1) = q_2 \\
 \delta(q_0, 110110) &= \delta(\delta(q_0, 11011), 0) = \delta(q_2, 0) = q_2 \in F \Rightarrow w_1 \in \mathcal{T}(A)
 \end{aligned}$$

Folosind definiția 3.3 arătăm că $w_2 = 1100 \notin \mathcal{T}(A)$:

$$\begin{aligned}
 \delta(q_0, 1) &= q_0 \\
 \delta(q_0, 11) &= \delta(\delta(q_0, 1), 1) = \delta(q_0, 1) = q_0 \\
 \delta(q_0, 110) &= \delta(\delta(q_0, 11), 0) = \delta(q_0, 0) = q_1 \\
 \delta(q_0, 1100) &= \delta(\delta(q_0, 110), 0) = \delta(q_1, 0) = q_1 \notin F \Rightarrow w_2 \notin \mathcal{T}(A)
 \end{aligned}$$

Exemplul 3.5: Automatul finit determinist $A = (\Sigma, Q, \delta, q_0, F)$ acceptă toate sirurile binare care conțin subșirul 000, respectiv $w_1 = 1000110 \in \mathcal{T}(A)$, iar $w_2 = 1001001 \notin \mathcal{T}(A)$.



Analitic, automatul $A = (\Sigma, Q, \delta, q_0, F)$ va fi reprezentat astfel:

- $\Sigma = \{0,1\}$
- $Q = \{q_0, q_1, q_2, q_3\}$
- funcția de tranziție δ :

δ	0	1
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_3	q_0
*	q_3	q_3

Folosind definiția 3.3 arătăm că $w_1 = 100011 \in \mathcal{T}(A)$:

$$\begin{aligned}
 \delta(q_0, 1) &= q_0 \\
 \delta(q_0, 10) &= \delta(\delta(q_0, 1), 0) = \delta(q_0, 0) = q_1 \\
 \delta(q_0, 100) &= \delta(\delta(q_0, 10), 0) = \delta(q_1, 0) = q_2 \\
 \delta(q_0, 1000) &= \delta(\delta(q_0, 100), 0) = \delta(q_2, 0) = q_3 \\
 \delta(q_0, 10001) &= \delta(\delta(q_0, 1000), 1) = \delta(q_3, 1) = q_3 \\
 \delta(q_0, 100011) &= \delta(\delta(q_0, 10001), 1) = \delta(q_3, 1) = q_3 \in F \Rightarrow w_1 \in \mathcal{T}(A)
 \end{aligned}$$

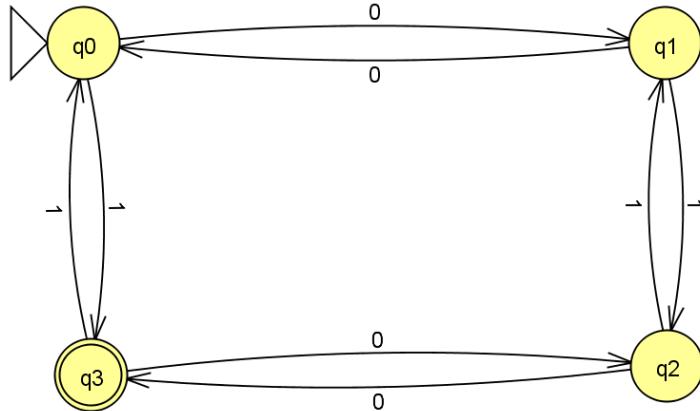
Folosind definiția 3.3 arătăm că $w_2 = 1001001 \notin \mathcal{T}(A)$:

$$\begin{aligned}
 \delta(q_0, 1) &= q_0 \\
 \delta(q_0, 10) &= \delta(\delta(q_0, 1), 0) = \delta(q_0, 0) = q_1 \\
 \delta(q_0, 100) &= \delta(\delta(q_0, 10), 0) = \delta(q_1, 0) = q_2 \\
 \delta(q_0, 1001) &= \delta(\delta(q_0, 100), 1) = \delta(q_2, 1) = q_0 \\
 \delta(q_0, 10010) &= \delta(\delta(q_0, 1001), 0) = \delta(q_0, 0) = q_1 \\
 \delta(q_0, 100100) &= \delta(\delta(q_0, 10010), 0) = \delta(q_1, 0) = q_2 \\
 \delta(q_0, 1001001) &= \delta(\delta(q_0, 100100), 1) = \delta(q_2, 1) = q_0 \notin F \Rightarrow w_2 \notin \mathcal{T}(A)
 \end{aligned}$$

Exemplul 3.6: Automatul finit determinist $A = (\Sigma, Q, \delta, q_0, F)$ acceptă toate sirurile binare care au un număr par de 0 și un număr impar de 1, respectiv $w_1 = 1001010 \in \mathcal{T}(A)$, iar $w_2 = 1011001 \notin \mathcal{T}(A)$.

Vom considera $Q = \{q_0, q_1, q_2, q_3\}$, stările având următoarele semnificații:

- q_0 : sirul conține un număr par de 0 și un număr par de 1;
- q_1 : sirul conține un număr impar de 0 și un număr par de 1;
- q_2 : sirul conține un număr impar de 0 și un număr impar de 1;
- q_3 : sirul conține un număr par de 0 și un număr impar de 1.



Analitic, automatul $A = (\Sigma, Q, \delta, q_0, F)$ va fi reprezentat astfel:

- $\Sigma = \{0,1\}$
- $Q = \{q_0, q_1, q_2, q_3\}$
- funcția de tranziție δ :

δ	0	1
$\rightarrow q_0$	q_1	q_3
q_1	q_0	q_2
q_2	q_3	q_1
$* q_3$	q_2	q_0

Folosind definiția 3.3 arătăm că $w_1 = 1001010 \in \mathcal{T}(A)$:

$$\begin{aligned}
 \delta(q_0, 1) &= q_3 \\
 \delta(q_0, 10) &= \delta(\delta(q_0, 1), 0) = \delta(q_3, 0) = q_2 \\
 \delta(q_0, 100) &= \delta(\delta(q_0, 10), 0) = \delta(q_2, 0) = q_3 \\
 \delta(q_0, 1001) &= \delta(\delta(q_0, 100), 1) = \delta(q_3, 1) = q_0 \\
 \delta(q_0, 10010) &= \delta(\delta(q_0, 1001), 0) = \delta(q_0, 0) = q_1 \\
 \delta(q_0, 100101) &= \delta(\delta(q_0, 10010), 1) = \delta(q_1, 1) = q_2 \\
 \delta(q_0, 1001010) &= \delta(\delta(q_0, 100101), 0) = \delta(q_2, 0) = q_3 \in F \Rightarrow w_1 \in \mathcal{T}(A)
 \end{aligned}$$

Folosind definiția 3.3 arătăm că $w_2 = 1011001 \notin \mathcal{T}(A)$:

$$\begin{aligned}
 \delta(q_0, 1) &= q_3 \\
 \delta(q_0, 10) &= \delta(\delta(q_0, 1), 0) = \delta(q_3, 0) = q_2 \\
 \delta(q_0, 101) &= \delta(\delta(q_0, 10), 1) = \delta(q_2, 1) = q_1 \\
 \delta(q_0, 1011) &= \delta(\delta(q_0, 101), 1) = \delta(q_1, 1) = q_2 \\
 \delta(q_0, 10110) &= \delta(\delta(q_0, 1011), 0) = \delta(q_2, 0) = q_3 \\
 \delta(q_0, 101100) &= \delta(\delta(q_0, 10110), 0) = \delta(q_3, 0) = q_2 \\
 \delta(q_0, 1011001) &= \delta(\delta(q_0, 101100), 1) = \delta(q_2, 1) = q_1 \notin F \Rightarrow w_2 \notin \mathcal{T}(A)
 \end{aligned}$$

3.3. Automate finite nedeterministe

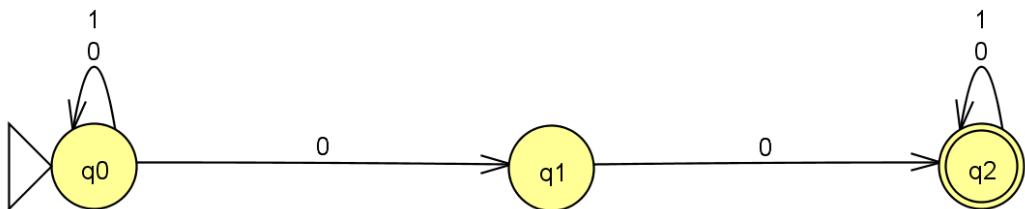
Diferența dintre automatele finite deterministe și cele nedeterministe constă în faptul că dintr-o anumită stare un automat finit nedeterminist poate să treacă în mai multe stări, în timp ce un automat finit deterministic poate să treacă doar într-o singură stare.

Practic, funcția de tranziție este definită prin $\delta: Q \times \Sigma \rightarrow \mathcal{P}(Q)$. Extinderea funcției de tranziție $\bar{\delta}: Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$ se va face astfel:

$$\begin{cases} \bar{\delta}(q, \lambda) = \{q\}, \forall q \in Q \\ \bar{\delta}(s, wa) = \bigcup_{y=\delta(w,a)} \delta(y, a), \forall q \in Q, w \in \Sigma^*, a \in \Sigma \end{cases}$$

Definiția 3.4: Limbajul acceptat de un automat finit nedeterminist $A = (\Sigma, Q, \delta, q_0, F)$ este mulțimea $\mathcal{T}(A) = \{w \in \Sigma^* | \delta(q_0, w) \cap F \neq \emptyset\}$.

Exemplul 3.7: Automatul finit nedeterminist $A = (\Sigma, Q, \delta, q_0, F)$ acceptă toate sirurile binare care conțin subșirul 00, respectiv $w_1 = 1101001 \in \mathcal{T}(A)$, iar $w_2 = 10101 \notin \mathcal{T}(A)$.



Analitic, automatul $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ va fi reprezentat astfel:

- $\Sigma = \{0,1\}$
- $Q = \{q_0, q_1, q_2\}$
- funcția de tranziție δ :

δ	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\{q_2\}$	\emptyset
$*$ q_2	$\{q_2\}$	$\{q_2\}$

Folosind definiția 3.4 arătăm că $w_1 = 1101001 \in \mathcal{T}(A)$:

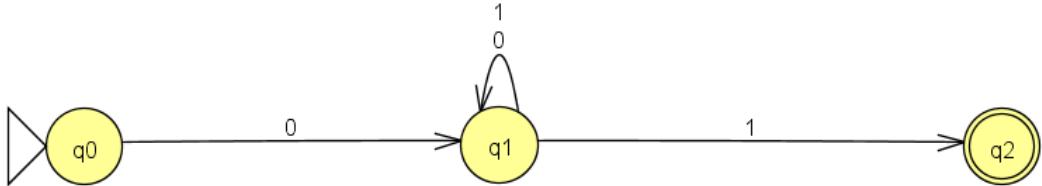
$$\begin{aligned} \delta(\{q_0\}, 1) &= \{q_0\} \\ \delta(\{q_0\}, 11) &= \delta(\delta(\{q_0\}, 1), 1) = \delta(\{q_0\}, 1) = \{q_0\} \\ \delta(\{q_0\}, 110) &= \delta(\delta(\{q_0\}, 11), 0) = \delta(\{q_0\}, 0) = \{q_0, q_1\} \end{aligned}$$

$$\begin{aligned}
\delta(\{q_0\}, 1101) &= \delta(\delta(\{q_0\}, 100), 1) = \delta(\{q_0, q_1\}, 1) = \delta(\{q_0\}, 1) \cup \delta(\{q_1\}, 1) = \\
&= \{q_0\} \cup \emptyset = \{q_0\} \\
\delta(\{q_0\}, 11010) &= \delta(\delta(\{q_0\}, 1101), 0) = \delta(\{q_0\}, 0) = \{q_0, q_1\} \\
\delta(\{q_0\}, 110100) &= \delta(\delta(\{q_0\}, 11010), 0) = \delta(\{q_0, q_1\}, 0) = \delta(\{q_0\}, 0) \cup \delta(\{q_1\}, 0) = \\
&= \{q_0, q_1\} \cup \{q_2\} = \{q_0, q_1, q_2\} \\
\delta(\{q_0\}, 1101001) &= \delta(\delta(\{q_0\}, 110100), 1) = \delta(\{q_0, q_1, q_2\}, 1) = \\
&= \delta(\{q_0\}, 1) \cup \delta(\{q_1\}, 1) \cup \delta(\{q_2\}, 1) = \{q_0\} \cup \emptyset \cup \{q_2\} = \\
&= \{q_0, q_2\} \cap F = \{q_2\} \neq \emptyset \Rightarrow w_1 \in \mathcal{T}(A)
\end{aligned}$$

Folosind definiția 3.4 arătăm că $w_2 = 10101 \notin \mathcal{T}(A)$:

$$\begin{aligned}
\delta(\{q_0\}, 1) &= \{q_0\} \\
\delta(\{q_0\}, 10) &= \delta(\delta(\{q_0\}, 1), 0) = \delta(\{q_0\}, 0) = \{q_0, q_1\} \\
\delta(\{q_0\}, 101) &= \delta(\delta(\{q_0\}, 10), 1) = \delta(\{q_0, q_1\}, 1) = \delta(\{q_0\}, 1) \cup \delta(\{q_1\}, 1) = \\
&= \{q_0\} \cup \emptyset = \{q_0\} \\
\delta(\{q_0\}, 1010) &= \delta(\delta(\{q_0\}, 101), 0) = \delta(\{q_0\}, 0) = \{q_0, q_1\} \\
\delta(\{q_0\}, 10101) &= \delta(\delta(\{q_0\}, 1010), 1) = \delta(\{q_0, q_1\}, 1) = \delta(\{q_0\}, 1) \cup \delta(\{q_1\}, 1) = \\
&= \{q_0\} \cup \emptyset = \{q_0\} \cap F = \emptyset \Rightarrow w_2 \notin \mathcal{T}(A)
\end{aligned}$$

Exemplul 3.8: Automatul finit nedeterminist $A = (\Sigma, Q, \delta, q_0, F)$ acceptă toate sirurile binare care încep cu 0 și se termină cu 1, respectiv $w_1 = 01101 \in \mathcal{T}(A)$, iar $w_2 = 01010 \notin \mathcal{T}(A)$.



Analitic, automatul $A = (\Sigma, Q, \delta, q_0, F)$ va fi reprezentat astfel:

- $\Sigma = \{0,1\}$
- $Q = \{q_0, q_1, q_2\}$
- funcția de tranziție δ :

δ	0	1
→ q_0	{ q_1 }	\emptyset
→ q_1	{ q_1 }	{ q_1, q_2 }
* q_2	\emptyset	\emptyset

Folosind definiția 3.4 arătăm că $w_1 = 01101 \in \mathcal{T}(A)$:

$$\delta(\{q_0\}, 0) = \{q_1\}$$

$$\begin{aligned}
\delta(\{q_0\}, 01) &= \delta(\delta(\{q_0\}, 0), 1) = \delta(\{q_1\}, 0) = \{q_1\} \\
\delta(\{q_0\}, 011) &= \delta(\delta(\{q_0\}, 01), 1) = \delta(\{q_1\}, 1) = \{q_1, q_2\} \\
\delta(\{q_0\}, 0110) &= \delta(\delta(\{q_0\}, 011), 0) = \delta(\{q_1, q_2\}, 0) = \delta(\{q_1\}, 0) \cup \delta(\{q_2\}, 0) = \\
&= \{q_1\} \cup \emptyset = \{q_1\} \\
\delta(\{q_0\}, 01101) &= \delta(\delta(\{q_0\}, 0110), 1) = \delta(\{q_1\}, 1) = \{q_1, q_2\} \cap F = \\
&= \{q_2\} \neq \emptyset \Rightarrow w_1 \in \mathcal{T}(A)
\end{aligned}$$

Folosind definiția 3.4 arătăm că $w_2 = 01010 \notin \mathcal{T}(A)$:

$$\begin{aligned}
\delta(\{q_0\}, 0) &= \{q_1\} \\
\delta(\{q_0\}, 01) &= \delta(\delta(\{q_0\}, 0), 1) = \delta(\{q_1\}, 0) = \{q_1\} \\
\delta(\{q_0\}, 010) &= \delta(\delta(\{q_0\}, 01), 0) = \delta(\{q_1\}, 0) = \{q_1\} \\
\delta(\{q_0\}, 0101) &= \delta(\delta(\{q_0\}, 010), 1) = \delta(\{q_1\}, 1) = \{q_1, q_2\} \\
\delta(\{q_0\}, 01010) &= \delta(\delta(\{q_0\}, 0101), 0) = \delta(\{q_1, q_2\}, 0) = \delta(\{q_1\}, 0) \cup \delta(\{q_2\}, 0) = \\
&= \{q_1\} \cup \emptyset = \{q_1\} \cap F = \emptyset \Rightarrow w_2 \notin \mathcal{T}(A)
\end{aligned}$$

3.4. Echivalența dintre automatele finite deterministe și automatele finite nedeterministe

Definiția 3.5: Două automate finite A_1 și A_2 sunt echivalente dacă ele acceptă același limbaj, respectiv $\mathcal{T}(A_1) = \mathcal{T}(A_2)$.

Observația 3.3: Evident, orice automat finit determinist poate fi considerat un caz particular de automat finit nedeterminist.

Teorema 3.1: Orice limbaj acceptat de un automat finit nedeterminist A_N este acceptat și de un automat finit determinist A_D echivalent.

Demonstrație:

Vom prezenta doar modalitatea de construcție a automatului finit determinist $A_D = (\Sigma_D, Q_D, \delta_D, q_0^D, F_D)$ echivalent cu un automat finit nedeterminist $A_N = (\Sigma_N, Q_N, \delta_N, q_0^N, F_N)$ dat. Astfel, vom considera:

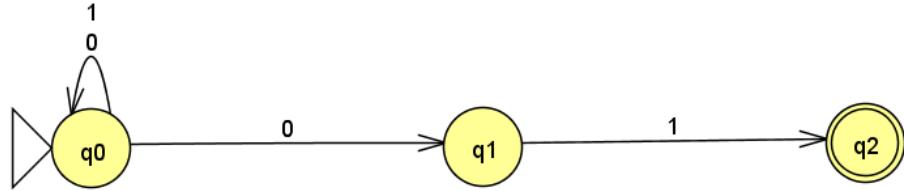
- $\Sigma_D = \Sigma_N$
- $Q_D = \mathcal{P}(Q_N)$
- funcția δ_D se va calcula pentru fiecare $q \in Q_D = \mathcal{P}(Q_N)$ și pentru fiecare $a \in \Sigma_N$:

$$\delta_D(q, a) = \bigcup_{y \in q} \delta_N(y, a)$$

- $q_0^D = \{q_0^N\}$
- $F_D = \{q \in \mathcal{P}(Q_N) \mid q \cap F_N \neq \emptyset\}$

Exemplul 3.9: Considerăm un automat finit nedeterminist $A_N = (\Sigma_N, Q_N, \delta_N, q_0^N, F_N)$ care acceptă sirurile binare care se termină cu 01 și determinăm automatul finit determinist $A_D = (\Sigma_D, Q_D, \delta_D, q_0^D, F_D)$ echivalent.

Automatul finit nedeterminist $A_N = (\Sigma_N, Q_N, \delta_N, q_0^N, F_N)$ poate fi reprezentat grafic astfel:



Analitic, automatul A_N va fi reprezentat astfel:

- $\Sigma_N = \{0,1\}$
- $Q_N = \{q_0^N, q_1, q_2\}$
- $F_N = \{q_2\}$
- funcția de tranziție δ_N :

δ	0	1
q_0^N	$\{q_0^N, q_1\}$	$\{q_0^N\}$
q_1	\emptyset	$\{q_2\}$
q_2	\emptyset	\emptyset

Automatul finit determinist $A_D = (\Sigma_D, Q_D, \delta_D, q_0^D, F_D)$ echivalent va fi construit astfel:

- $\Sigma_D = \Sigma_N = \{0,1\}$
- $Q_D = \mathcal{P}(Q_N) = \mathcal{P}(\{s_0^N, s_1, s_2\})$
- $F_D = \{\{s_2\}, \{s_0^N, s_2\}, \{s_1, s_2\}, \{s_0^N, s_1, s_2\}\}$
- funcția de tranziție δ_D :

δ_D	0	1
\emptyset	\emptyset	\emptyset
$\{q_0^N\}$	$\{q_0^N, q_1\}$	$\{q_0^N\}$
$\{q_1\}$	\emptyset	$\{q_0^N\}$
$\{q_2\}$	\emptyset	\emptyset
$\{q_0^N, q_1\}$	$\{q_0^N, q_1\}$	$\{q_0^N, q_2\}$
$\{q_0^N, q_2\}$	$\{q_0^N, q_1\}$	$\{q_0^N\}$
$\{q_1, q_2\}$	\emptyset	$\{q_2\}$
$\{q_0^N, q_1, q_2\}$	$\{q_0^N, q_1\}$	$\{q_0^N, q_2\}$

A
B
C
D
E
F
G
H

Redenumind convenabil stările automatului A_D , putem redefini automatul astfel:

- $\Sigma_D = \{0,1\}$
- $Q_D = \{A, B, C, D, E, F, G, H\}$
- $F_D = \{D, F, G, H\}$
- funcția de tranziție δ_D :

δ_D	0	1
$\rightarrow A$	A	A
$\rightarrow B$	E	B
$\rightarrow C$	A	D
$\ast D$	A	A
$\ast E$	E	F
$\ast F$	E	B
$\ast G$	A	D
$\ast H$	E	F

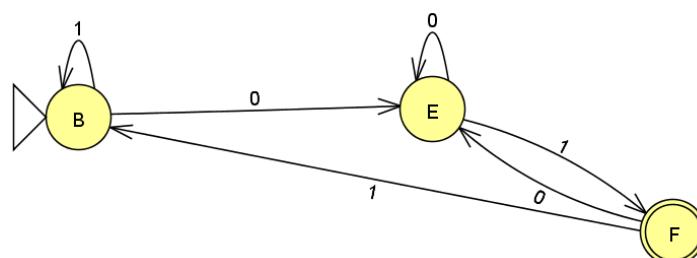
Observația 3.4: Prin aplicarea acestei metode, automatul finit determinist echivalent obținut conține și stări inaccesibile. Simplificarea sa constă în determinarea stărilor accesibile din starea inițială, care este întotdeauna accesibilă, și eliminarea stărilor inaccesibile.

Se observă ușor că automatul A_D nu are decat stările B, F și E accesibile, deci restul stărilor pot fi eliminate. Se obține astfel forma finală a automatului A_D :

- $\Sigma_D = \{0,1\}$
- $Q_D = \{B, E, F\}$
- $F_D = \{F\}$
- funcția de tranziție δ_D :

δ_D	0	1
$\rightarrow B$	E	B
$\rightarrow E$	E	F
$\ast F$	E	B

Reprezentarea grafică a automatului A_D este următoarea:



3.5. Echivalența dintre gramaticile regulate și automatele finite nedeterministe

Teorema 3.2: Pentru orice gramatică regulată G se poate construi un automat finit nedeterminist A care să accepte limbajul generat de gramatica G , respectiv $L(G) = \mathcal{T}(A)$.

Demonstrație:

Fie $G = (N, T, S, P)$ o gramatică regulată și X un simbol nou, respectiv $X \notin N \cup T$.

Construim automatul finit nedeterminist echivalent $A = (\Sigma, Q, \delta, q_0, F)$ astfel:

- $\Sigma = T$
- $Q = N \cup \{X\}$
- $q_0 = S$
- $F = X$
- funcția de tranziție δ se definește pentru $\forall B \in N, a \in \Sigma = T$ astfel:
 - $\delta(B, a) = \{C \in N \mid B \rightarrow aC \in P\} \cup \{X \mid B \rightarrow a \in P\}$
 - $\delta(X, a) = \emptyset, \forall a \in \Sigma$

Exemplul 3.10: Se consideră gramatica regulată $G = (N, T, S, P)$, unde $N = \{S, A, B\}, T = \{a, b, c\}$, iar mulțimea producțiilor P este următoarea:

$$S \rightarrow aA|bB|b \quad (1)$$

$$A \rightarrow aA|bB|b \quad (2)$$

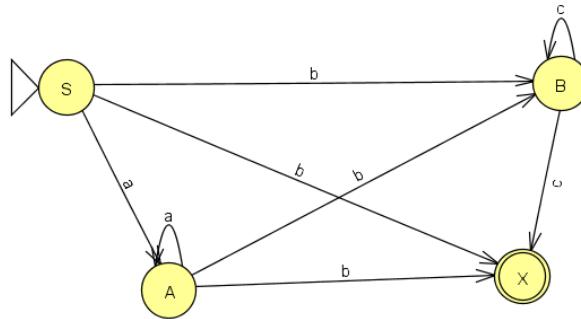
$$B \rightarrow cB|c \quad (3)$$

Automatul finit nedeterminist $A_N = (\Sigma_N, Q_N, \delta_N, q_0^N, F_N)$ echivalent cu gramatica regulată G , astfel:

- $\Sigma_N = \{a, b, c\}$
- $Q_N = \{S, A, B, X\}$
- $q_0^N = S$
- $F_N = X$
- funcția de tranziție δ_N :

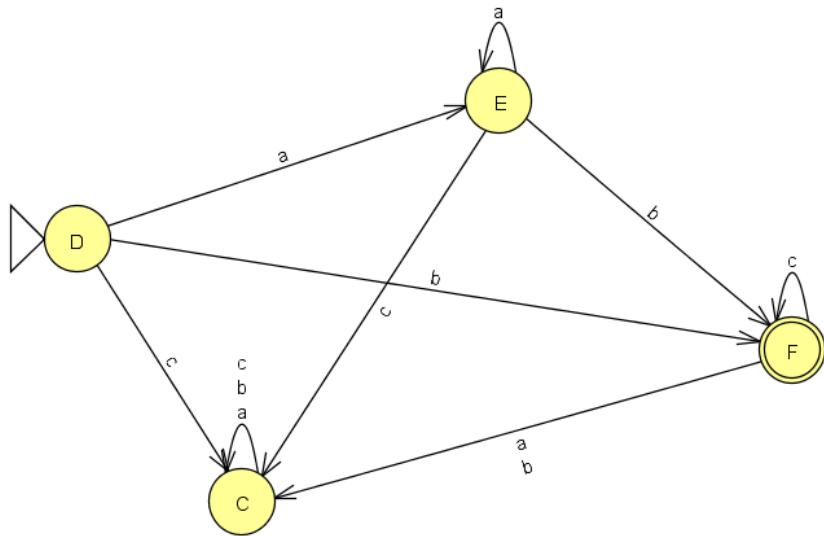
δ_N	a	b	c
S	$\{A\}$	$\{B, X\}$	\emptyset
A	$\{A\}$	$\{B, X\}$	\emptyset
B	\emptyset	\emptyset	$\{B, X\}$
X	\emptyset	\emptyset	\emptyset

Automatul finit nedeterminist A_N poate fi reprezentat grafic astfel:



Se poate observa că limbajul acceptat de automatul finit nedeterminist A_N este $\mathcal{T}(A) = \{a^m b c^n | m, n \geq 0\}$.

Un automat finit determinist echivalent cu automatul finit nedeterminist A_N este următorul:



3.6. Exerciții propuse

1. Se consideră automatul finit determinist $A = (\Sigma, Q, \delta, q_0, F)$, unde $\Sigma = \{0, 1\}$, $Q = \{q_0, q_1, q_2, q_3\}$, $F = \{q_0, q_1, q_2\}$, iar funcția de tranziție δ este dată în următorul tabel:

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

- a) Arătați că $w_1 = 01001011 \in T(A)$, dar $w_2 = 10100010 \notin T(A)$.
- b) Reprezentați graficul automatului A .
- c) Descrieți, pe scurt, limbajul acceptat de către automatul A .
- d) Scrieți o gramatică regulată G echivalentă cu A .

2. Se consideră automatul finit deterministic $A = (\Sigma, K, \delta, s_0, F)$, unde $\Sigma = \{0,1\}$, $K = \{s_0, s_1, s_2, s_3\}$, $F = \{s_3\}$, iar funcția de tranziție δ este dată în tabel următor:

δ	0	1
s_0	s_0	s_1
s_1	s_0	s_2
s_2	s_0	s_3
s_3	s_0	s_3

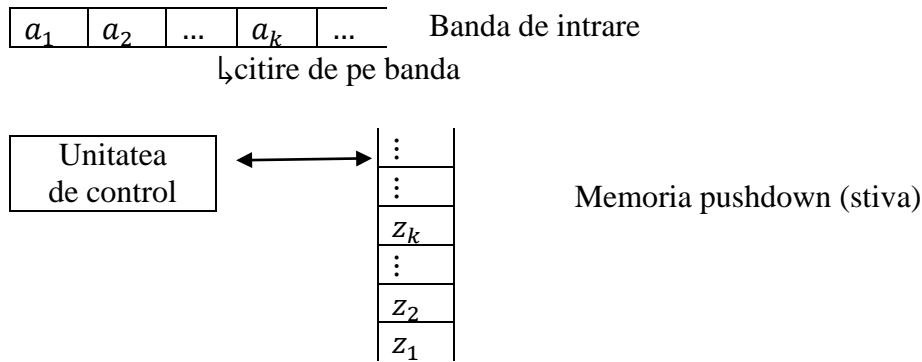
- a) Arătați că $w_1 = 0010111 \in T(A)$ și $w_2 = 101110 \notin T(A)$.
 - b) Reprezentați grafic automatul A .
 - c) Descrieți limbajul acceptat de către automatul A .
 - d) Scrieți o gramatică regulată G echivalentă cu automatul A dat.
3. Determinați automatele finit deterministe echivalente cu automatele finit nedeterministe din exemplele 3.7 și 3.8.
4. Determinați automatele finit deterministe echivalente cu gramaticile regulate din exemplele 3.1, 3.2 și 3.3.

MODULUL IV

Automate pushdown

4.1. Automate pushdown

Un automat pushdown este, practic, un automat finit nedeterminist prevăzut cu o memorie organizată sub formă de stivă.



Definiția 4.1: Un *automat pushdown* P este un sistem $P = (\Sigma, K, \Gamma, \delta, s_0, Z_0, F)$, unde:

- Σ se numește *alfabetul de intrare*;
 - K se numește *mulțimea stărilor*;
 - Γ se numește *alfabetul memoriei pushdown*
 - δ reprezintă *funcția de tranziție* a automatului și este definită astfel:

$$\delta: K \times (\Sigma \cup \{\lambda\}) \times \Gamma \rightarrow P(K \times \Gamma^*)$$

$$\delta(s, a, X) = \{(t, \gamma) | t \in K, \gamma \in \Gamma^*\}$$

$$\text{configurația} \\ \text{curentă}$$
- În funcție de forma șirului γ , se disting următoarele trei cazuri:
- 1) dacă $\gamma = \lambda$, atunci se efectuează o operație pop asupra memoriei pushdown;
 - 2) dacă $\gamma = Y \in \Gamma$, atunci se înlocuiește simbolul X din vârful stivei cu simbolul Y , fără a se efectua nici o operație pop sau push;
 - 3) dacă $\gamma = YZ \in \Gamma^*$, atunci simbolul X se înlocuiește cu simbolul Z , iar asupra simbolului Y se efectuează o operație push.
- $s_0 \in K$ reprezintă *starea inițială a automatului*;
 - Z_0 se numește *simbolul inițial al memoriei pushdown*;
 - F se numește *mulțimea stărilor finale ale automatului*.

Definiția 4.2: Spunem că automatul pushdown P trece din configurația $(s, aw, X\alpha)$ în configurația $(t, w, \gamma\alpha)$ și notăm acest lucru prin \vdash dacă perechea $(t, \gamma) \in \delta(s, aw, X)$, unde $s, t \in K, a \in \Sigma, w \in \Sigma^*, X \in \Gamma, \alpha, \gamma \in \Gamma^*$.

Definiția 4.3: Notăm prin \vdash^* închiderea tranzitivă și reflexivă a relației ce definește mișcarea unui automat.

Definiția 4.4: Un automat pushdown P poate să accepte un cuvânt $w \in \Sigma^*$ în două moduri:

- 1) Spunem că *automatul P acceptă cuvântul w cu stare finală* dacă există $s \in F$ și $\alpha \in \Gamma^*$ astfel încât $(s_0, w, Z_0) \vdash^* (s, \lambda, \alpha)$.
- 2) Spunem că *automatul P acceptă cuvântul w cu memorie pushdown vidă* dacă există $s \in K$ astfel încât $(s_0, w, Z_0) \vdash^* (s, \lambda, \lambda)$.

Observația 4.1: În cazul automatelor pushdown care acceptă cu memorie vidă se consideră implicit că $F = \emptyset$.

Definiția 4.5: Limbajul acceptat de un automat pushdown este format din multimea tuturor cuvintelor acceptate de el, indiferent de mod.

Exemplul 4.1: Considerăm automatul pushdown $P = (\Sigma, K, \Gamma, \delta, s_0, Z_0, F)$ care să accepte limbajul $L = \{a^n b^n | n \geq 0\}$ cu stare finală, definit astfel:

$$\begin{aligned}\Sigma &= \{a, b\} \\ K &= \{s_0, s_1, s_2\} \\ \Gamma &= \{a, Z_0\} \\ F &= \{s_0\} \\ \delta: & \\ \delta(s_0, a, Z_0) &= \{(s_1, aZ_0)\} \\ \delta(s_1, a, a) &= \{(s_1, aa)\} \\ \delta(s_1, b, a) &= \{(s_2, \lambda)\} \\ \delta(s_2, b, a) &= \{(s_2, \lambda)\} \\ \delta(s_2, \lambda, Z_0) &= \{(s_0, \lambda)\}\end{aligned}$$

Cuvântul $w_1 = aaabbb \in L(P)$ deoarece:

$$\begin{aligned}(s_0, aaabbb, Z_0) &\vdash^* (s_1, aabb, aZ_0) \vdash^* (s_1, abbb, aaZ_0) \vdash^* (s_1, bbb, aaaZ_0) \vdash^* \\ (s_2, bb, aaZ_0) &\vdash^* (s_2, b, aZ_0) \vdash^* (s_2, \lambda, Z_0) \vdash^* (s_0, \lambda, \lambda) \Rightarrow s_0 \in F\end{aligned}$$

Observația 4.2: Automatul P acceptă cuvântul w_1 și cu memorie pushdown vidă, dar ar fi trebuit să avem $F = \emptyset$.

Cuvântul $w_2 = aaabb \notin L(P)$ deoarece:

$$\begin{aligned}(s_0, aaabb, Z_0) &\vdash^* (s_1, aabb, aZ_0) \vdash^* (s_1, abb, aaZ_0) \vdash^* (s_1, bb, aaaZ_0) \vdash^* \\ (s_2, b, aaZ_0) &\vdash^* (s_2, \lambda, aZ_0) \Rightarrow s_0 \notin F\end{aligned}$$

Observația 4.3: Cuvântul w_1 nu ar fi fost acceptat nici cu memorie pushdown vidă, deoarece am terminat de citit cuvantul, dar nu am golit memoria pushdown.

Exemplul 4.2: Un automat pushdown care acceptă limbajul $L = \{ww^R | w \in \{a, b\}^*\}$.

$$w = c_1 c_2 \dots c_n \Rightarrow w^R = c_n c_{n-1} \dots c_1$$

Observația 4.4: Se poate demonstra, folosind lema de pompăre, că limbajul L nu este regulat
 \Rightarrow nu există niciun automat finit care să-l accepte.

Observația 4.5: Limbajul L este un limbaj independent de context, deoarece este generat de următoarea gramatică independentă de context $G = (N, T, S, P)$:

$$N = \{S\}$$

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

$$P: S \rightarrow \lambda$$

$$S \rightarrow aSa$$

$$S \rightarrow bSb$$

Automatul pushdown $P = (\Sigma, K, \Gamma, \delta, s_0, Z_0, F)$ este definit astfel:

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

$$K = \{s_0, s_1, s_2\}$$

$$\Gamma = \{a, b, Z_0\}$$

$$F = \{s_2\}$$

δ :

$$\delta(s_0, a, Z_0) = \{(s_1, aZ_0)\}$$

$$\delta(s_0, b, Z_0) = \{(s_1, bZ_0)\}$$

$$\delta(s_0, a, a) = \{(s_0, aa)\}$$

$$\delta(s_0, b, a) = \{(s_0, ba)\}$$

$$\delta(s_0, a, b) = \{(s_0, ab)\}$$

$$\delta(s_0, b, b) = \{(s_0, bb)\}$$

$$\delta(s_0, \lambda, a) = \{(s_1, a)\}$$

$$\delta(s_0, \lambda, b) = \{(s_1, b)\}$$

$$\delta(s_0, \lambda, \lambda) = \{(s_1, \lambda)\}$$

$$\delta(s_1, a, a) = \{(s_1, \lambda)\}$$

$$\delta(s_1, b, b) = \{(s_1, \lambda)\}$$

$$\delta(s_1, \lambda, Z_0) = \{(s_2, \lambda)\}$$

Fie $w_1 = abba$. Atunci:

$$(s_0, abba, z_0) \xrightarrow{*} (s_0, bba, aZ_0) \xrightarrow{*} (s_0, ba, baZ_0) \xrightarrow{*} (s_1, ba, baZ_0) \xrightarrow{*}$$

$$(s_1, a, aZ_0) \xrightarrow{*} (s_1, \lambda, Z_0) \xrightarrow{*} (s_2, \lambda, Z_0) \Rightarrow s_2 \in F \Rightarrow w_1 \text{ este acceptat}$$

4.2. Construcția unui automat pushdown echivalent cu o gramatică independentă de context

Fie $G = (N, T, S, P)$ o gramatică independentă de context. Construim automatul pushdown echivalent $A = (\Sigma, K, \Gamma, \delta, s_0, Z_0, F)$ astfel:

- $\Sigma = T$
- $K = \{s_0\}$
- $\Gamma = N \cup T \cup \{Z_0\}$
- $Z_0 = S$
- $F \neq \emptyset$
- Funcția de tranziție δ se construiește astfel:
 - $\delta(s_0, \lambda, A) = \{(s_0, \alpha) | (A \rightarrow \alpha) \in P\}$ pentru $\forall A \in N$
 - $\delta(s_0, a, a) = \{(s_0, \lambda)\}$ pentru $\forall a \in T$

Exemplul 4.3: Considerăm următoarea gramatică independentă de context G care generează expresii aritmetice simple în variabila a :

$$G = (N, T, S, P)$$

$$N = \{S, E, T, F\}$$

$$T = \{a, +, *, +(), ()\}$$

$$P: S \rightarrow E \quad (1)$$

$$E \rightarrow E + T \quad (2)$$

$$E \rightarrow T \quad (3)$$

$$T \rightarrow T * F \quad (4)$$

$$T \rightarrow F \quad (5)$$

$$F \rightarrow (E) \quad (6)$$

$$F \rightarrow a \quad (7)$$

Arătăm că $w = a * (a + a * a) \in L(G)$:

$$\begin{aligned} S &\xrightarrow[\substack{(1) \\ (2) \\ (3) \\ (4) \\ (5) \\ (6) \\ (7)}]{*} E \xrightarrow[\substack{(1) \\ (2) \\ (3) \\ (4) \\ (5) \\ (6) \\ (7)}]{*} T \xrightarrow[\substack{(1) \\ (2) \\ (3) \\ (4) \\ (5) \\ (6) \\ (7)}]{*} T * F \xrightarrow[\substack{(1) \\ (2) \\ (3) \\ (4) \\ (5) \\ (6) \\ (7)}]{*} F * F \xrightarrow[\substack{(1) \\ (2) \\ (3) \\ (4) \\ (5) \\ (6) \\ (7)}]{*} a * F \xrightarrow[\substack{(1) \\ (2) \\ (3) \\ (4) \\ (5) \\ (6) \\ (7)}]{*} a * (E) \xrightarrow[\substack{(1) \\ (2) \\ (3) \\ (4) \\ (5) \\ (6) \\ (7)}]{*} a * (E + T) \xrightarrow[\substack{(1) \\ (2) \\ (3) \\ (4) \\ (5) \\ (6) \\ (7)}]{*} a * (T + T) \\ &\xrightarrow[\substack{(1) \\ (2) \\ (3) \\ (4) \\ (5) \\ (6) \\ (7)}]{*} a * (F + T) \xrightarrow[\substack{(1) \\ (2) \\ (3) \\ (4) \\ (5) \\ (6) \\ (7)}]{*} a * (a + T) \xrightarrow[\substack{(1) \\ (2) \\ (3) \\ (4) \\ (5) \\ (6) \\ (7)}]{*} a * (a + T * F) \xrightarrow[\substack{(1) \\ (2) \\ (3) \\ (4) \\ (5) \\ (6) \\ (7)}]{*} a * (a + F * F) \\ &\xrightarrow[\substack{(1) \\ (2) \\ (3) \\ (4) \\ (5) \\ (6) \\ (7)}]{*} a * (a + a * F) \xrightarrow[\substack{(1) \\ (2) \\ (3) \\ (4) \\ (5) \\ (6) \\ (7)}]{*} a * (a + a * a) \end{aligned}$$

Construim automatul pushdown echivalent:

$$A = (\Sigma, K, \Gamma, \delta, s_0, Z_0, F)$$

$$\Sigma = \{a, +, *, +(), ()\}$$

$$K = \{s_0\}$$

$$\Gamma = \{S, E, T, F, a, +, *, +(), ()\}$$

$$Z_0 = S$$

δ :

$$\begin{aligned}\delta(s_0, \lambda, S) &= \{(s_0, E)\} \\ \delta(s_0, \lambda, E) &= \{(s_0, E + T), (s_0, T)\} \\ \delta(s_0, \lambda, T) &= \{(s_0, T * F), (s_0, F)\} \\ \delta(s_0, \lambda, F) &= \{(s_0, (E)), (s_0, a)\}\end{aligned}$$

Fie $w = a * (a + a * a)$. Atunci:

$$\begin{aligned}\delta(s_0, a * (a + a * a), S) &\xrightarrow{*} \delta(s_0, * (a + a * a), E) \xrightarrow{*} \delta(s_0, a * (a + a * a), T) \\ &\xrightarrow{*} \delta(s_0, a * (a + a * a), T * F) \xrightarrow{*} \delta(s_0, a * (a + a * a), F * F) \xrightarrow{*} \delta(s_0, a \\ &\quad * (a + a * a), a * F) \\ &\xrightarrow{*} \delta(s_0, * (a + a * a), * F) \xrightarrow{*} \delta(s_0, (a + a * a), F) \xrightarrow{*} (s_0, (a + a * a), (E)) \\ &\xrightarrow{*} \delta(s_0, a + a * a, E) \xrightarrow{*} \delta(s_0, a + a * a, E + T) \xrightarrow{*} \delta(s_0, a + a * a, T + T) \\ &\xrightarrow{*} \delta(s_0, a + a * a, F + T) \xrightarrow{*} \delta(s_0, a + a * a, a + T) \xrightarrow{*} \delta(s_0, +a * a, +T) \\ &\xrightarrow{*} \delta(s_0, a * a, T) \xrightarrow{*} \delta(s_0, a * a, T * F) \xrightarrow{*} \delta(s_0, a * a, F * F) \xrightarrow{*} \delta(s_0, a * a, a \\ &\quad * F) \\ &\xrightarrow{*} \delta(s_0, * a, * F) \xrightarrow{*} \delta(s_0, a, F) \xrightarrow{*} \delta(s_0, a, a) \xrightarrow{*} \delta(s_0, ,)) \xrightarrow{*} \delta(s_0, \lambda, \lambda)\end{aligned}$$

4.3. Construcția unei grămatici independente de context echivalentă cu un automat pushdown

Fie $A = (\Sigma, K, \Gamma, \delta, s_0, Z_0, \emptyset)$ un automat pushdown care acceptă cu memorie vidă. Construim o grămatică independentă de context $G = (N, T, S, P)$ echivalentă cu el astfel:

- $N = \{S\} \cup \{[sXt] | s, t \in K, X \in \Gamma\}$
- $\Sigma = T$
- mulțimea producțiilor P se construiește astfel:
 - $\forall s \in K, S \rightarrow [s_0 Z_0 s]$
 - dacă $\delta(s, a, X) = (t, y_1 y_2 \dots y_k)$ atunci adăugăm producțiile: $[sXr_k] \rightarrow a[tY_1r_1][r_1Y_2r_2] \dots [r_{k-1}Y_kr_k], \forall (r_1, r_2, \dots, r_k) \in K$
 - dacă avem $k = 0$, atunci introducem și producția $[sXt] \rightarrow a$

Exemplul 4.4: Fie limbajul $L = \{01^n 0 | n \geq 1\}$ care este acceptat de automatul pushdown $A = (\Sigma, K, \Gamma, \delta, s_0, Z_0, \emptyset)$, definit astfel:

- $\Sigma = \{a, b\}$
- $K = \{s_0, s_1, s_2\}$
- $\Gamma = \{X, Z_0\}$

- δ :

$$\begin{aligned}\delta(s_0, 0, Z_0) &= \{(s_1, XZ_0)\} \\ \delta(s_1, 1, X) &= \{(s_1, X)\} \\ \delta(s_1, 0, X) &= \{(s_2, \lambda)\} \\ \delta(s_2, \lambda, Z_0) &= \{(s_2, \lambda)\}\end{aligned}$$

Construim gramatica independentă de context $G = (N, T, S, P)$ echivalentă cu A astfel:

- $N = \{S\} \cup \{[sXt] | s, t \in K, X \in \Gamma\}$
- $T = \{a, b\}$
- mulțimea producțiilor P este următoarea:

$$\begin{aligned}S &\rightarrow [s_0Z_0s_0] \\ S &\rightarrow [s_0Z_0s_1] \\ S &\rightarrow [s_0Z_0s_2] \\ [s_1Xs_2] &\rightarrow 0 \\ [s_2Z_0s_2] &\rightarrow \lambda \\ [s_1Xs_0] &\rightarrow 1[s_1Xs_0] \\ [s_1Xs_1] &\rightarrow 1[s_1Xs_1] \\ [s_1Xs_2] &\rightarrow 1[s_1Xs_2] \\ [s_0Z_0s_0] &\rightarrow 0[s_1Xs_0][s_0Z_0s_0] \\ [s_0Z_0s_1] &\rightarrow 0[s_1Xs_1][s_1Z_0s_1] \\ [s_0Z_0s_2] &\rightarrow 0[s_1Xs_2][s_2Z_0s_2] \\ [s_0Z_0s_1] &\rightarrow 0[s_1Xs_0][s_0Z_0s_1] \\ [s_0Z_0s_2] &\rightarrow 0[s_1Xs_0][s_0Z_0s_2] \\ [s_0Z_0s_0] &\rightarrow 0[s_1Xs_1][s_1Z_0s_0] \\ [s_0Z_0s_2] &\rightarrow 0[s_1Xs_1][s_1Z_0s_2] \\ [s_0Z_0s_0] &\rightarrow 0[s_1Xs_2][s_2Z_0s_0] \\ [s_0Z_0s_1] &\rightarrow 0[s_1Xs_2][s_2Z_0s_1]\end{aligned}$$

Gramatica $G = (N, T, S, P)$ se poate simplifica, astfel obținându-se:

- $N = \{S, [s_0Z_0s_2], [s_1Xs_2], [s_2Z_0s_2], [s_1Xs_2]\}$
- $T = \{a, b\}$
- mulțimea producțiilor P este următoarea:

$$\begin{aligned}S &\rightarrow [s_0Z_0s_2] \\ [s_1Xs_2] &\rightarrow 0 \\ [s_2Z_0s_2] &\rightarrow \lambda \\ [s_1Xs_2] &\rightarrow 1[s_1Xs_2] \\ [s_0Z_0s_2] &\rightarrow 0[s_1Xs_2][s_2Z_0s_2]\end{aligned}$$

MODULUL V

Proprietăți ale limbajelor independente de context și ale limbajelor regulate

5.1. Lema Bar-Hillel. Lema de pompare

Lema Bar-Hillel este utilizată pentru a demonstra că un limbaj nu este limbaj independent de context.

Lema 5.1 (Bar-Hillel): Fie L un limbaj independent de context. Atunci $\exists n \in \mathbb{N}^*$ astfel încât orice cuvânt $w \in L$ cu $|w| \geq n$ poate fi scris sub forma $w = xuyvz$, unde:

- 1) $|uyv| \leq n$
- 2) $uv \neq \lambda$ ($|uv| \neq 0$)
- 3) $\forall k \in \mathbb{N} \Rightarrow xu^kyv^kz \in L$

Exemplul 5.1: Arătăm că limbajul $L = \{a^n | n \text{ este număr prim}\}$ nu este un limbaj independent de context.

Presupunem că L este un limbaj independent de context $\xrightarrow{\text{lema Bar-Hillel}} \exists n \in \mathbb{N}^*$ astfel încât oricare ar fi $w \in L$ cu $|w| \geq n$ poate fi scris sub forma $w = xuyvz$, unde:

- 1) $|uyv| \leq n$
- 2) $|uv| \neq 0$
- 3) $\forall k \in \mathbb{N} \Rightarrow xu^kyv^kz \in L$

Fie $w \in L \Rightarrow w = a^p = xu^kyv^kz \Rightarrow x = a^{k_1}, u = a^{k_2}, y = a^{k_3}, n = a^{k_4}, z = a^{k_5}$ cu $k_1 + k_2 + k_3 + k_4 + k_5 = p$.

Din condiția 3) $\Rightarrow xu^iyv^iz \in L \Rightarrow a^{k_1+k_2+k_3+k_4+k_5} \in L, \forall i \in \mathbb{N}$

Fie $i = p + 1 \Rightarrow \begin{cases} p \geq 2 \\ i \geq 3 \end{cases} \Rightarrow k_1 + (p+1)k_2 + k_3 + (p+1)k_4 + k_5$ este număr prim
 $\Rightarrow \underbrace{k_1 + k_2 + k_3 + k_4 + k_5}_p + p(k_2 + k_4)$ este număr prim $\Rightarrow p(1 + k_2 + k_4)$ este număr

prim $\Rightarrow 1 + k_2 + k_4 = 1 \Rightarrow k_2 + k_4 = 0$

Din condiția 2) $\Rightarrow |uv| \neq 0 \Rightarrow |a^{k_2} * a^{k_4}| \neq 0 \Rightarrow |a^{k_2+k_4}| \neq 0 \Rightarrow k_2 + k_4 \neq 0$

$\begin{cases} k_2 + k_4 = 0 \\ k_2 + k_4 \neq 0 \end{cases} \Rightarrow$ contradicție cu presupunerea făcută \Rightarrow limbajul L nu este un limbaj independent de context.

Într-un mod asemănător, lema de pompare este utilizată pentru a demonstra că un limbaj nu este limbaj regulat.

Lema 5.2 (lema de pompare): Fie L un limbaj regulat. Atunci există $n \in \mathbb{N}$ astfel încât oricare ar fi $w \in L$ cu $|w| \geq n$ poate fi scris sub forma $w = xyz$, unde:

- 1) $|xy| \leq n$

- 2) $|y| \neq 0$
- 3) $xy^k z \in L \forall k \in \mathbb{N}$

Exemplul 5.2: Arătăm că limbajul $L = \{a^m b^m | m \geq 1\}$ nu este un limbaj regulat.

Presupunem că L este limbaj regulat $\xrightarrow{\text{Lema de pompare}} \exists n \in \mathbb{N}$ astfel încât $w \in L$ cu $|w| \geq n$ poate fi scris sub forma $w = xyz$, unde:

- 1) $|xy| \leq n$
- 2) $|y| \neq 0$
- 3) $xy^k z \in L \forall k \in \mathbb{N}$

Fie $w = a^m b^m \in L \Rightarrow w = \underbrace{aaa \dots aaa}_{m \text{ ori}} \underbrace{bbb \dots bbb}_{m \text{ ori}}$

Dar $\begin{cases} |xy| \leq n \Rightarrow xy \text{ este format doar din } a \\ |y| \neq 0 \end{cases} \Rightarrow y \text{ conține cel puțin un } a$

Pentru $k = 0 \Rightarrow xz \in L$, ceea ce reprezintă o contradicție deoarece x are cel mult $m - 1$ de a , iar y are exact m de b , deci presupunerea făcută este falsă.

5.2. Operații cu limbaje

Fie $\begin{cases} \mathcal{L}_1 = \text{familia limbajelor dependente de context} \\ \mathcal{L}_2 = \text{familia limbajelor independente de context} \\ \mathcal{L}_3 = \text{familia limbajelor regulate} \end{cases}$

Teorema 5.1: Familiile \mathcal{L}_i , $i = (1, 2, 3)$ sunt închise la operația de reuniune.

Demonstrație:

Fie $L', L'' \in \mathcal{L}_i$ și $G' = (V'_N, V'_T, S', P')$ cu $L(G') = L'$
 $G'' = (V''_N, V''_T, S'', P'')$ cu $L(G'') = L''$

Putem presupune că $V'_N \cap V''_N = \emptyset$. Fie $S \notin V'_N \cup V''_N \cup V'_T \cup V''_T$ un simbol nou.

- Pentru $i = 0, 1, 2$ consider $G = (V_N, V_T, S, P)$ cu:

$$\begin{cases} V_N = V'_N \cup V''_N \cup \{S\}; V_T = V'_T \cup V''_T \\ P = P' \cup P'' \cup \{S \rightarrow S', S \rightarrow S''\} \end{cases}$$

- Pentru $i = 3$, în construcția de mai sus:

$$\begin{cases} S \rightarrow S' \text{ se înlocuiește cu } \{S \rightarrow \alpha | S' \rightarrow \alpha \in P'\} \\ S \rightarrow S'' \text{ se înlocuiește cu } \{S \rightarrow \alpha | S'' \rightarrow \alpha \in P''\} \end{cases}$$

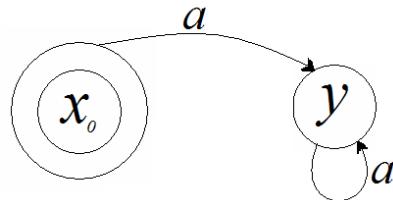
Teorema 5.2: Familiile L_i , $i = (1, 2, 3)$ conțin limbajele finite.

Demonstrație:

Cum $L_3 \subset L_2 \subset L_1$, este suficient să arătăm că L_3 conține limbajele finite.

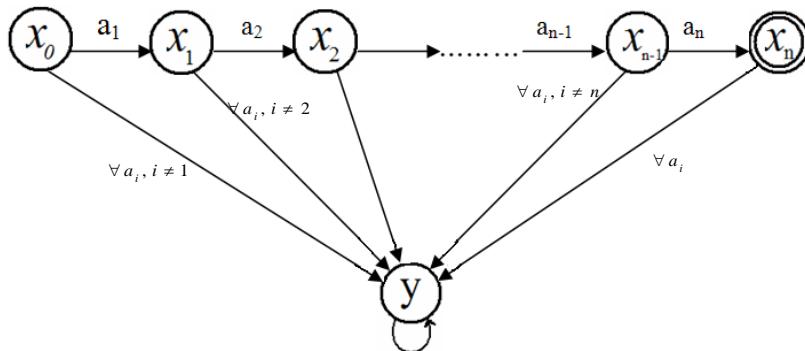
Fie L limbaj finit.

- $L = \emptyset$. Este acceptată de automatul $A = (\Sigma, X, \delta, x_0, F)$.
- $L = \{\lambda\}$. Acceptat de automatul următor:



- $L = \{w\}$ cu $w = a_1 a_2 \cdots a_n$, $n \geq 1$

Consider $A = (\{x_0, x_1, \dots, x_n, y\}, \{a_1, \dots, a_n\}, \delta, x_0, \{x_n\})$



- $L = \{w_1, w_2, \dots, w_n\}$ aplic teorema 5.1.

Teorema 5.3: Familiile L_i , $i = (1, 2, 3)$ sunt închise la concatenare.

Demonstrație:

Fie $G^+ = (V_N^+, V_T^+, S^+, P^+)$ și $G^+ = (V_N^+, V_T^+, S^+, P^+)$ de tip I, cu $L(G^+) = L^+$, $L(G^+) = L^+$.

- Pentru $i = 1, 2$: Fie S simbol nou. Considerăm gramatica:

$$G = (V_N^+ \cup V_N^+ \cup \{S\}, V_T^+ \cup V_T^+, S, P^+ \cup P^+ \cup \{S \rightarrow S^+ S^+\})$$

Evident $L(G) = L^+ L^+$

- Pentru $i = 3$: Consider gramatica:

$$G = (V_N^+ \cup V_N^+, V_T^+ \cup V_T^+, S^+, P) \text{ cu}$$

$$P = \{A \rightarrow aB \mid A \rightarrow aB \in P^+\} \cup \{A \rightarrow aS^+ \mid A \rightarrow a \in P^+\} \cup P^+$$

adică atunci când urmează să fie aplicată o producție de tipul $A \rightarrow a$ din P' (care încheie generarea unui cuvânt din L'), aplic $A \rightarrow aS''$ pentru a deschide calea generării în continuare a unui cuvânt din L'' .

Teorema 5.4: Familia L_3 este închisă la complementară.

Fie $L \in L_3$ și $A = (\Sigma, X, \delta, x_0, F)$ cu $T(A) = L$. Atunci $A' = (X, \Sigma, \delta, x_0, X \setminus F)$ generează $T(A') = \Sigma^* \setminus L = \bar{L}$.

Corolarul 5.1: Familia L_3 este închisă la intersecție.

Demonstrație:

Este suficient să folosim legile lui de Morgan: $L' \cap L'' = (\bar{L}' \cup \bar{L}'')^*$ unde prin \bar{L} am notat complementara lui L .

Definiția 5.1: Fie L limbaj. Definim $L^* = \{w_1 \cdots w_n \mid n \geq 0, w_i \in L\} =$ mulțimea concatenărilor de oricâte cuvinte din L .

Teorema 5.5: Familia L_3 este închisă față de operația $*$.

Demonstrație:

Fie $L \in L_3$ și $G = (V_N, V_T, S, P)$ cu $L(G) = L$.

Considerăm $G' = (V_N, V_T, S, P')$ unde $P' = P \cup \{A \rightarrow aS \mid A \rightarrow a \in P\}$

O caracterizare a familiei de limbaje L_3 este dată de următoarea teoremă, datorată lui Kleene:

Teorema 5.6 (Kleene): Familia L_3 este cea mai mică clasă de limbaje ce conține limbajele finite și este închisă la operațiile de reuniune, concatenare și $*$.

BIBLIOGRAFIE

1. Livovschi L., Georgescu H., Țăndăreanu N., Popovici C.P. – *Bazele informaticii*, Ed. Didactică și Pedagogică, București, 1981
2. Creangă I., Reischer C., Simovici D. – *Introducere în algebra informatică*, Ed. Junimea, 1974
3. Jalobeanu C., Marinescu D. – *Bazele teoriei calculului*, Ed. Albastră, 2007
4. Orman G. – *Limbaje formale și acceptori*, Ed. Albastră, 2008
5. Hopcroft J., Motwani R., Ullman J. – *Introduction to Automata Theory, Languages and Computation*, Addison Wesley, 2000
6. Davis M.D., Sigal R., Weyuker E.J. – *Computability, Complexity and Languages*, Academic Press (Morgan Kaufmann), 1994
7. Papadimitriou C.H. – *Computational Complexity*, Addison-Wesley, 1994

UNITATEA DE ÎNVĂȚARE NR. 1

LIMBAJUL DE PROGRAMARE S. NOȚIUNEA DE FUNCȚIE CALCULABILĂ. MACROINSTRUCȚIUNI ÎN LIMBAJUL S

Cuprins

1.1.	Obiectivele unității de învățare nr. 1	3
1.2.	Indicații metodice pentru unitatea de învățare nr. 1	3
1.3.	Limbajul de programare S	3
1.4.	Noțiunea de funcție calculabilă	5
1.5.	Macroinstructiuni în limbajul S	6
1.6.	Tema de autoinstruire nr. 1	11
1.7.	Testul de autoevaluare nr. 1	11
1.8.	Comentarii și răspunsuri la testul nr. 1 de autoevaluare	12
1.9.	Lucrare de verificare pentru studenți	12
1.10.	Bibliografie	13

1.1. Obiectivele unității de învățare nr. 1

După ce veți parcurge această unitate de învățare, veți reuși să:

- cunoașteți instrucțiunile limbajului S, precum și noțiunea de macroinstructiune;
- înțelegeți noțiunea de funcție calculabilă.

1.2. Indicații metodice pentru unitatea de învățare nr. 1

Materialul trebuie parcurs în ordinea sa firească, prezentată în continuare. Se recomandă conspectarea și notarea ideilor principale, precum și consultarea bibliografiei pentru detalii și informații suplimentare.

Timpul minim pe care trebuie să-l acordați acestei unități de învățare este de **8 ore**.

1.3. Limbajul de programare S

Un program P scris **în limbajul s** este o secvență de instrucțiuni (un număr finit de instrucțiuni scrise într-o anumită ordine), program în care intervin:

- numere naturale
 - variabile de intrare, notate de obicei prin x_1, x_2, \dots
 - variabila de ieșire y
 - variabilele intermediare (de lucru), notate de obicei prin z_1, z_2, \dots
- cu mențiunea că valorile luate de variabile pot fi *orice* numere naturale.

Deoarece lucrăm numai cu numere naturale, vom spune uneori pur și simplu număr în loc de număr natural.

Înainte de a prezenta instrucțiunile limbajului s , precizăm că ele pot fi precedate, opțional, de etichete. Acestea sunt reprezentate prin litere mari (eventual indexate cu numere naturale), cuprinse între paranteze și plasate la începutul instrucțiunii. Se admite ca mai multe instrucțiuni să aibă aceeași etichetă; rațiunea acestui fapt va fi prezentată în capitolul 4.

Instrucțiunile admise și semnificația lor sunt următoarele:

$v \leftarrow v + 1$	Valoarea curentă a variabilei v crește cu o unitate.
$v \leftarrow v - 1$	Valoarea curentă a variabilei v scade cu o unitate (dacă era pozitivă), respectiv rămâne 0 (dacă era 0).
if $v \neq 0$ goto L	Dacă valoarea curentă a variabilei v este nenulă, atunci se face transfer necondiționat la prima instrucțiune din program etichetată cu L ; dacă nici o instrucțiune nu este etichetată cu L , atunci programul se termină. Dacă valoarea curentă a lui v este 0, atunci se trece la instrucțiunea următoare.
$v \leftarrow v$	Este instrucțiunea de efect nul, a cărei utilitate va apărea ulterior.

Exemplul 1. Programul:

```
[A] x ← x - 1
    y ← y + 1
    if x ≠ 0 goto A
```

se termină pentru orice valoare inițială α a lui x . La ieșire vom avea:

$$y = \begin{cases} 1, & \text{dacă } \alpha = 0 \\ \alpha, & \text{dacă } \alpha > 0 \end{cases}$$

Exemplul 2. Programul:

```
[A] x ← x + 1
    if x ≠ 0 goto A
```

nu se termină niciodată, indiferent de valoarea inițială a lui x .

Observație. Faptul că programele în limbajul s produc o singură valoare de ieșire nu este o restricție. Într-adevăr, dacă dorim să obținem mai multe valori de ieșire, vom scrie câte un program pentru fiecare și vom executa succesiv aceste programe. Menționăm că, în acest studiu, suntem interesați numai de existența algoritmilor, nu și de eficiența lor.

1.4. Notiunea de funcție calculabilă

Fie P un program scris în limbajul s . Fie V mulțimea tuturor variabilelor (de intrare, de lucru și de ieșire) ce apar în P .

Definim *starea* programului la un moment oarecare a executării sale ca fiind o funcție $s : V \rightarrow \mathbf{N}$, unde \mathbf{N} este mulțimea numerelor naturale. Deci o stare este dată de valorile curente ale variabilelor din program.

O *configurație* a programului P de lungime n (având n instrucțiuni) este o pereche (i, s) cu $i \in \{1, 2, \dots, n, n+1\}$ și s stare.

O *configurație inițială* are forma $(1, s_0)$, unde:

$$s_0(u) = \begin{cases} r_i, & \text{dacă } u = x_i \text{ și } r_i \text{ este valoarea inițială a lui } x_i \\ 0, & \text{dacă } u \text{ nu este o variabilă de intrare} \end{cases}$$

O *configurație terminală* are forma $(n+1, s)$. Vom asimila transferul la o etichetă inexistentă cu transferul la instrucțiunea cu numărul $n+1$.

Fie (i, s) o configurație neterminată. *Succesoarea ei* (j, t) este definită astfel:

- 1) Dacă instrucțiunea i este $v \leftarrow v + 1$, atunci $j = i + 1$, iar:

$$t(u) = \begin{cases} u, & \text{dacă } u \neq v \\ u + 1, & \text{dacă } u = v \end{cases}$$

- 2) Dacă instrucțiunea i este $v \leftarrow v - 1$, atunci $j = i + 1$, iar:

$$t(u) = \begin{cases} s(v) - 1, & \text{dacă } u = v \text{ și } s(v) > 0 \\ s(v), & \text{în caz contrar} \end{cases}$$

- 3) Dacă instrucțiunea i este $v \leftarrow v$, atunci $j = i + 1$ și $t = s$.

- 4) Dacă instrucțiunea i este **if** $v \neq 0$ **goto** L , atunci $t = s$, iar:

$$j = \begin{cases} i + 1, & \text{dacă } s(v) = 0 \\ n + 1, & \text{dacă } s(v) \neq 0 \text{ și nu există vreo instrucțiune etichetată cu } L \\ k, & \text{dacă } s(v) \neq 0 \text{ și instrucțiunea } k \text{ este prima etichetată cu } L \end{cases}$$

Se observă că succesoarea oricărei configurații neterminale este unic determinată și nu depinde de eventuala etichetare a sa.

Un calcul al programului P este o secvență de configurații consecutive c_0, c_1, \dots, c_k cu c_0 configurație inițială și c_k configurație finală (terminală).

Fie P un program în care variabilele de intrare sunt x_1, \dots, x_m . Atunci *funcția (parțială) calculată de programul P* este $\psi_P^{(m)} : \mathbb{N}^m \rightarrow \mathbb{N}$ definită astfel:

- 1) $\psi_P^{(m)}(x_1, \dots, x_m) = \bar{y}$ dacă există un calcul c_0, c_1, \dots, c_k al lui P cu $c_0 = (1, s_0)$ și $s_k(y) = \bar{y}$;
- 2) $\psi_P^{(m)}(x_1, \dots, x_m) = \uparrow$ (nedefinit) dacă există un sir infinit de configurații c_0, c_1, \dots al lui P cu $c_0 = (1, s_0)$ (deci dacă programul P nu se termină).

Observație. Fie P un program în care apar variabilele de intrare x_1, \dots, x_m . Pentru orice n natural, îi putem asocia o funcție $\psi_P^{(n)}$ astfel:

- dacă $n \leq m$, atunci $\psi_P^{(n)}(x_1, \dots, x_n) = \psi_P^{(m)}(x_1, \dots, x_m)$, adică x_{n+1}, \dots, x_m sunt ignorate;
- dacă $n > m$, atunci $\psi_P^{(n)}(x_1, \dots, x_n) = \psi_P^{(m)}(x_1, \dots, x_m, 0, \dots, 0)$.

Fie $f : \mathbb{N}^m \rightarrow \mathbb{N}$. Funcția f se numește *parțial calculabilă* dacă există un program P în limbajul s cu $\psi_P^{(m)} = f$. Dacă $\text{Dom } f = \mathbb{N}^m$, atunci f se numește *calculabilă*.

Exemplul 1 de mai sus arată că funcția $f : \mathbb{N} \rightarrow \mathbb{N}$ dată de:

$$f(x) = \begin{cases} 1, & \text{dacă } x = 0 \\ x, & \text{dacă } x > 0 \end{cases}$$

este calculabilă.

1.5. Macroinstructiuni în limbajul S

Macroinstructiunile sunt abrevieri pentru secvențe de instructiuni în limbajul s . Menirea lor este de a scrie programe cât mai inteligibile. Prezența unei macroinstructiuni într-un program trebuie interpretată ca prezența în acel punct al programului a unei *dezvoltări* a sale.

Chiar dacă în același program apare în diferite locuri o aceeași macroinstructiune, aceasta nu înseamnă că va fi inserată aceeași dezvoltare a sa. Mai mult, chiar impunem ca aparițiile unei aceleași macroinstructiuni pe poziții diferite din program să presupună dezvoltări diferite ale macroinstructiunii (variabile de lucru și etichete distințe). De asemenea, trebuie respectată regula ca valoarea variabilelor care apar în macroinstructiune, cu excepția celor care apar în membrul stâng al unei atribuirii, să nu fie modificate ca efect al executării macroinstructiunii.

1) Macroinstructiunea goto L are următoarea dezvoltare posibilă:

```
[A] z ← z + 1
    if z ≠ 0 goto A
```

2) Macroinstructiunea v ← 0 are dezvoltarea:

```
[A] v ← v - 1
    if v ≠ 0 goto A
```

unde eticheta A nu mai apare nicăieri în programul ce conține această macroinstructiune.

Să remarcăm faptul că variabilele locale dezvoltării unei macroinstructiuni sunt presupuse a avea inițial valoarea 0, ca orice variabilă de lucru. Totuși, în cadrul dezvoltării trebuie în general să le atribuim mai întâi valoarea 0 (cu excepția cazului în care suntem siguri că după executarea macroinstructiunii valoarea variabilelor locale va fi egală cu 0), deoarece macroinstructiunea poate apărea în program în cadrul unui ciclu.

3) Macroinstructiunea $v \leftarrow k$ are dezvoltarea:

```
v ← 0
v ← v + 1
...
v ← v + 1
```

unde instrucțiunea $v \leftarrow v + 1$ apare de exact k ori.

Exemplul 3. Funcția $f : \mathbf{N} \rightarrow \mathbf{N}$ dată de $f(x) = x$ este calculabilă. Vom demonstra acest fapt scriind un program P cu $\psi_P^{(1)} = f$, program ce va asigura păstrarea valorii inițiale a lui x :

```
[A] if  $x \neq 0$  goto B
    goto C
[B] x ← x - 1
    y ← y + 1
    z ← z + 1
    goto A
[C] if  $z \neq 0$  goto D
    goto E
[D] z ← z - 1
    x ← x + 1
    goto C
```

unde primele 6 instrucțiuni copiază valoarea lui x în variabilele y și z , iar următoarele recopiază valoarea lui z în x .

Putem acum introduce macroinstructiunea $v \leftarrow u$ cu dezvoltarea:

```
v ← 0
[A] if  $u \neq 0$  goto B
    goto C
[B] u ← u - 1
    v ← v + 1
    z ← z + 1
    goto A
[C] if  $z \neq 0$  goto D
    goto E
[D] z ← z - 1
    u ← u + 1
```

goto *c*
[E] *v* \leftarrow *v*

Se impun următoarele trei remarci:

1. nu am inițializat pe *z* cu 0, deoarece este clar că după orice executare a macroinstructiunii, valoarea sa va fi 0;
2. apare necesitatea instructiunii *v* \leftarrow *v*;
3. ca de obicei, variabila *z* nu mai apare nicăieri în program (este o variabilă de lucru "nouă"), iar etichetele *A, B, C, D, E* nu mai apar nicăieri în program (sunt etichete "noi").

Având convingerea că cititorul s-a obișnuit deja cu aceste reguli, nu le vom mai repeta în continuare, presupunându-le subînțelese.

Exemplul 4. Funcția $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ definită prin $f(x_1, x_2) = x_1 + x_2$ este calculabilă. Un program care calculează această funcție este următorul:

y \leftarrow *x*₁
z \leftarrow *x*₂
[B] **if** *z* $\neq 0$ **goto** *A*
goto *E*
[A] *z* \leftarrow *z* - 1
y \leftarrow *y* + 1
goto *B*

iar macroinstructiunea *z* \leftarrow *z*₁ + *z*₂ are următoarea dezvoltare posibilă:

u \leftarrow *z*₁
v \leftarrow *z*₂
[B] **if** *v* $\neq 0$ **goto** *A*
goto *E*
[A] *v* \leftarrow *v* - 1
u \leftarrow *u* + 1
goto *B*
[E] *z* \leftarrow *u*

Exemplul 5. Funcția $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ definită prin $f(x_1, x_2) = x_1 \times x_2$ este calculabilă, iar macroinstructiunea *z* \leftarrow *z*₁ × *z*₂ are următoarea dezvoltare posibilă:

u \leftarrow 0
v \leftarrow *z*₂
[B] **if** *v* $\neq 0$ **goto** *A*
goto *E*
[A] *v* \leftarrow *v* - 1
u \leftarrow *u* + *z*₁
goto *B*
[E] *z* \leftarrow *u*

Exemplul 6. Fie $f : \mathbf{N} \times \mathbf{N} \rightarrow \mathbf{N}$ funcția parțială definită prin:

$$f(x_1, x_2) = \begin{cases} x_1 - x_2, & \text{dacă } x_1 \geq x_2 \\ \uparrow, & \text{dacă } x_1 < x_2 \end{cases}$$

Un program care calculează această funcție este următorul:

```

y ← x1
z ← x2
[C] if z ≠ 0 goto A
      goto E
[A] if y ≠ 0 goto B
      goto A {y = 0 și z ≠ 0}
[B] y ← y - 1
      z ← z - 1
      goto C
  
```

Macroinstrucțiunea asociată este $\underline{z \leftarrow z_1 - z_2}$ și are următoarea dezvoltare posibilă:

```

u1 ← z1
u2 ← z2
[C] if u2 ≠ 0 goto A
      goto E
[A] if u1 ≠ 0 goto B
      goto A
[B] u1 ← u1 - 1
      u2 ← u2 - 1
      goto C
[E] z ← u1
  
```

Exemplul 7. Fie $f : \mathbf{N}^m \rightarrow \mathbf{N}$ o funcție (parțial) calculabilă, calculată de un program P . Vom introduce macroinstrucțiunea $\underline{z \leftarrow f(z₁, ..., z_m)}$.

Pentru a obține o dezvoltarea a acestei macroinstrucțiuni, să presupunem că în programul P variabilele de intrare sunt x_1, \dots, x_m , variabilele de lucru sunt z_1, \dots, z_k , iar etichetele sunt A_1, \dots, A_p , unde (cu eventuala excepție a lui A_p), orice etichetă ce apare într-o instrucțiune **if** este eticheta a cel puțin uneia dintre instrucțiunile din P . Atunci o dezvoltarea a macroinstrucțiunii $\underline{z \leftarrow f(z_1, \dots, z_m)}$ este următoarea:

```

— y ← 0
— x1 ← x1
...
— xm ← xm
— z1 ← 0
  
```

\dots
 $\overline{z}_k \leftarrow 0$
 \overline{P}
 I

unde:

- I este una dintre instrucțiunile

$$\begin{cases} z \leftarrow \overline{y}, \text{ dacă eticheta } A_p \text{ este definită} \\ [A_p] \quad z \leftarrow y, \text{ altfel} \end{cases}$$
- $\overline{y}, \overline{x}_1, \dots, \overline{x}_m, \overline{z}_1, \dots, \overline{z}_k$ sunt variabile de lucru noi;
- $\overline{A}_1, \dots, \overline{A}_p$ sunt etichete noi;
- \overline{P} se obține din P înlocuind pe $y, x_1, \dots, x_m, z_1, \dots, z_k, A_1, \dots, A_p$ cu $\overline{y}, \overline{x}_1, \dots, \overline{x}_m, \overline{z}_1, \dots, \overline{z}_k, \overline{A}_1, \dots, \overline{A}_p$.

Se observă imediat că dacă pentru anumite valori inițiale r_1, \dots, r_m ale lui x_1, \dots, x_m programul P nu se termină, atunci nici executarea macroinstrucțiunii $\overline{z \leftarrow f(z_1, \dots, z_m)}$ nu se termină pentru z_1, \dots, z_m având, înainte de executarea macroinstrucțiunii, valorile r_1, \dots, r_m .

Dacă o macroinstrucțiune dintr-un program nu se termină, atunci nici programul care o folosește nu se termină.

Numim *predicat* o funcție $f : \mathbb{N}^m \rightarrow \{0,1\}$, parțial sau total definită, unde 0 corespunde falsului, iar 1 corespunde adevărului.

Exemplul 8. Fie $P : \mathbb{N}^m \rightarrow \{0,1\}$ un predicat calculabil. Atunci putem introduce macroinstrucțiunea:

if $P(x_1, \dots, x_m)$ **goto** L

a cărei dezvoltare este:

$z \leftarrow P(x_1, \dots, x_m)$
if $z \neq 0$ **goto** L

Un caz particular îl constituie predicatul:

$$P(x) = \begin{cases} 1, & \text{dacă } x = 0 \\ 0, & \text{dacă } x \neq 0 \end{cases}$$

care este calculabil conform programului:

if $x \neq 0$ **goto** E
 $y \leftarrow y + 1$

ceea ce permite introducerea următoarei macroinstrucțiuni:

if $v = 0$ **goto** L .

1.6. Tema de autoinstruire nr. 1

Consultați bibliografia pentru:

1. a cunoaște alte exemple de funcții calculabile;
2. a cunoaște alte exemple de macroinstructiuni în limbajul S.

1.7. Testul de autoevaluare nr. 1

Răspunsurile la test se vor da în spațiul liber aflat în continuarea enunțurilor!

1. Fie următorul program P scris în limbajul S:

```
[A] x1 ← x1 − 1  
y ← y + 1  
if x1 ≠ 0 goto A  
[B] x2 ← x2 − 1  
y ← y + 1  
if x2 ≠ 0 goto B
```

Indicați valoarea pe care o va avea variabila de ieșire y la sfârșitul executării programului, în fiecare din următoarele cazuri:

- a) x₁ = 7, x₂ = 3 b) x₁ = 0, x₂ = 0 c) x₁ = 0, x₂ = 5 d) x₁ = 4, x₂ = 0

2. Determinați funcția calculată de următorul program scris în limbajul S:

```
[A] x1 ← x1 − 1  
x2 ← x2 − 1  
if x2 ≠ 0 goto A  
[B] x1 ← x1 − 1  
y ← y + 1  
if x1 ≠ 0 goto B
```

3. Demonstrați că funcția $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ definită prin $f(x_1, x_2) = x_1 \times x_2$ este calculabilă, scriind programul corespunzător în limbajul S.

Răspunsurile la acest test se găsesc pe pagina următoare!

1.8. Comentarii și răspunsuri la testul nr. 1 de autoevaluare

1. a) 10 b) 2 c) 6 d) 5

2. $\Psi_p^{(2)}(x_1, x_2) = \begin{cases} x_1 - x_2, & \text{dacă } x_1 > x_2 \\ 1, & \text{dacă } x_1 \leq x_2 \end{cases}$

3.

$z \leftarrow x_2$

[B] **if** $z \neq 0$ **goto** A

goto E

[A] $z \leftarrow z - 1$

$y \leftarrow y + x_1$

goto B

1.9. Lucrare de verificare pentru studenți

1. Fie următorul program P scris în limbajul S:

$z \leftarrow x_2$

[B] **if** $z \neq 0$

goto A

goto E

[A] $z \leftarrow z - 1$

$y \leftarrow y + x_1$

goto B

Indicați valoarea pe care o va avea variabila de ieșire y la sfârșitul executării programului, în fiecare din următoarele cazuri:

- a) $x_1 = 0, x_2 = 7$ b) $x_1 = 3, x_2 = 8$ c) $x_1 = 12, x_2 = 2$

2. Determinați funcția calculată de următorul program scris în limbajul S:

[A] $x_1 \leftarrow x_1 - 1$

$y \leftarrow y + 1$

if $x_1 \neq 0$ *goto* A

[B] $x_2 \leftarrow x_2 - 1$

$y \leftarrow y + 1$

if $x_2 \neq 0$ *goto* B

3. Considerând definite macroinstructiunile *goto L*, $v \leftarrow u$ și $v \leftarrow v + u$, demonstrați faptul că funcția $f: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$, $f(x_1, x_2) = (x_1 + 1)^{x_2}$ este calculabilă, scriind un program P în limbajul S astfel încât $\Psi_p^{(2)} = f$.

Rezolvările, dar și dificultățile întâmpinate, vor fi expediate prin email tutorelui.

1.10. Bibliografie

1. C. Calude – "*Complexitatea calculului – aspecte calitative*", Editura Științifică și Enciclopedică, 1962.
2. M.D. Davis, R. Sigal, E.J. Weyuker – "*Computability, Complexity and Languages*", Academic Press (Morgan Kaufmann), 1994.
3. N.D. Jones – "*Computability and Complexity*", MIT Press, 1997.
4. Christos H. Papadimitriou – "*Computational Complexity*", Addison-Wesley, 1994.
5. C.P. Popovici, S. Rudeanu, H. Georgescu – "*Bazele informaticii*", Vol.II, Tipografia Universității București, 1991.

UNITATEA DE ÎNVĂȚARE NR. 2

PROPRIETĂȚI DE ÎNCHIDERE ALE CLASEI FUNCȚIILOR CALCULABILE. EXEMPLE DE FUNCȚII CALCULABILE STABILITE ÎN BAZA PROPRIETĂȚILOR DE ÎNCHIDERE

Cuprins

2.1.	Obiectivele unității de învățare nr. 2	14
2.2.	Indicații metodice pentru unitatea de învățare nr. 2	14
2.3.	Proprietăți de închidere ale clasei funcțiilor calculabile. Funcții primitiv recursive.....	14
2.4.	Exemple de funcții calculabile stabilite în baza proprietăților de închidere	16
2.5.	Tema de autoinstruire nr. 2	20
2.6.	Testul de autoevaluare nr. 2	20
2.7.	Comentarii și răspunsuri la testul nr. 2 de autoevaluare.....	21
2.8.	Lucrare de verificare pentru studenți	21
2.9.	Bibliografie	21

2.1. Obiectivele unității de învățare nr. 2

După ce veți parcurge această unitate de învățare, veți reuși să:

- înțelegeți noțiunea de funcție primitivă și să cunoașteți mai multe proprietăți ale sale;
- demonstrați că o funcție este primitiv recursivă și, implicit, calculabilă.

2.2. Indicații metodice pentru unitatea de învățare nr. 2

Materialul trebuie parcurs în ordinea sa firească, prezentată în continuare. Se recomandă conspectarea și notarea ideilor principale, precum și consultarea bibliografiei pentru detalii și informații suplimentare.

Timpul minim pe care trebuie să-l acordați acestei unități de învățare este de **8 ore**.

2.3. Proprietăți de închidere ale clasei funcțiilor calculabile. Funcții primitiv recursive

Fie g_1, \dots, g_k funcții de aritate n , iar h o funcție de aritate k . Putem atunci defini funcția f de aritate n prin:

$$f(x_1, \dots, x_n) = h(g_1(x_1, \dots, x_n), \dots, g_k(x_1, \dots, x_n))$$

Propoziția 1. Dacă g_1, \dots, g_k, h sunt (parțial) calculabile, atunci și funcția f este (parțial) calculabilă.

Demonstrație:

Funcția f este calculată de următorul program:

```

 $z_1 \leftarrow g_1(x_1, \dots, x_n)$ 
...
 $z_k \leftarrow g_k(x_1, \dots, x_n)$ 
 $y \leftarrow h(z_1, \dots, z_k)$ 

```

Observație: Propoziția de mai sus arată că atât clasa funcțiilor parțial calculabile, cât și clasa funcțiilor calculabile, sunt închise la operația de compunere.

Fie n un număr natural și funcțiile $f : \mathbf{N}^n \rightarrow \mathbf{N}$, $g : \mathbf{N}^{n+2} \rightarrow \mathbf{N}$. Vom defini funcția $h : \mathbf{N}^{n+1} \rightarrow \mathbf{N}$ prin:

$$(1) \quad \begin{cases} h(x_1, \dots, x_n, 0) = f(x_1, \dots, x_n) \\ h(x_1, \dots, x_n, t+1) = g(x_1, \dots, x_n, t, h(x_1, \dots, x_n, t)) \end{cases}$$

Propoziția 2. Dacă f și g sunt calculabile, atunci h este calculabilă.

Demonstrație:

Într-adevăr, funcția h este calculată de programul:

```

 $y \leftarrow f(x_1, \dots, x_n)$ 
[A] if  $x_{n+1} = 0$  goto E
 $y \leftarrow g(t, y, x_1, \dots, x_n)$ 
 $t \leftarrow t + 1$ 
 $x_{n+1} \leftarrow x_{n+1} - 1$ 
goto A

```

Astfel, am demonstrat închiderea clasei funcțiilor calculabile la recursii de tipul (1).

Pentru cazul $n = 0$, recursia (1) se scrie:

$$(1') \quad \begin{cases} h(0) = k \in \mathbf{N} \\ h(t+1) = g(t, h(t)) \end{cases}$$

iar pentru $n = 1$, ea se scrie

$$(1'') \quad \begin{cases} h(x, 0) = f(x) \\ h(x, t+1) = g(x, t, h(x, t)) \end{cases}$$

unde în (1'') s-a ținut cont că funcția de aritate 0 se identifică cu o constantă.

Definiție. Vom numi *funcții inițiale* următoarele trei funcții, evident calculabile:

1. *funcția successor* $s : \mathbf{N} \rightarrow \mathbf{N}$ dată de $s(x) = x + 1$;

2. *funcția nulă* $n : \mathbb{N} \rightarrow \mathbb{N}$ dată de $n(x) = 0$;
3. *funcțiile proiecții* $u_i^n : \mathbb{N}^n \rightarrow \mathbb{N}$ ($i = 1, 2, \dots, n$) dată de $u_i^n(x_1, \dots, x_n) = x_i$.

Definiție. O funcție se numește *primitiv recursivă* dacă se obține din funcțiile inițiale aplicând de un număr finit de ori operația de compunere și recursia (1).

Evident, orice funcție primitiv recursivă este calculabilă. Menționăm însă că există funcții calculabile care nu sunt primitiv recursive (de exemplu, funcția lui Ackermann). În continuare vom lucra numai cu funcții calculabile.

Proprietățile de închidere prezentate mai sus ne permit să prezentăm și alte exemple de funcții calculabile. Fiecare dintre ele permite introducerea unei macroinstrucțiuni, conform exemplelor din unitatea de învățare nr.1. Noile exemple ne vor fi utile în continuare și, în plus, ne vor mări încrederea în limbajul *s*.

2.4. Exemple de funcții calculabile stabilite în baza proprietăților de închidere

Exemplul 1. Funcția $h : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ dată de $h(x_1, x_2) = x_1^{x_2}$ este calculabilă.

Pentru demonstrație, plecăm de la relațiile $x^0 = 1$; $x^{y+1} = x^y \times x$ (am considerat, prin convenție, că $0^0 = 1$), obținând:

$$\begin{cases} h(x_1, 0) = 1 \\ h(x_1, t + 1) = h(x_1, t) \cdot x_1 = g(x_1, t, h(x_1, t)), t > 0 \end{cases}$$

unde $g(x_1, x_2, x_3) = x_2 \cdot x_3$.

Exemplul 2. Funcția $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ dată de

$$f(x_1, x_2) = x_1 \div x_2 = \begin{cases} x_1 - x_2, & \text{dacă } x_1 \geq x_2 \\ 0, & \text{dacă } x_1 < x_2 \end{cases}$$

este calculabilă (demonstrația este propusă ca exercițiu).

Exemplul 3. Funcția $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ dată de $f(x_1, x_2) = |x_1 - x_2|$ este calculabilă, deoarece $|x_1 - x_2| = (x_1 \div x_2) + (x_2 \div x_1)$.

Exemplul 4. Funcția $\alpha : \mathbb{N} \rightarrow \mathbb{N}$ dată de

$$\alpha(x) = \begin{cases} 1, & \text{dacă } x = 0 \\ 0, & \text{dacă } x \neq 0 \end{cases}$$

este calculabilă, așa cum s-a arăta în exemplul 8 din unitatea de învățare nr.1.

Exemplul 5. Predicatul $f : \mathbf{N} \times \mathbf{N} \rightarrow \{0,1\}$ (predicatul " \leq ") dat de

$$f(x_1, x_2) = \begin{cases} 1, & \text{dacă } x_1 \leq x_2 \\ 0, & \text{altele} \end{cases}$$

este calculabil, deoarece $f(x_1, x_2) = \alpha(x_1 \dot{-} x_2)$; într-adevăr, $x_1 \leq x_2 \Leftrightarrow x_1 \dot{-} x_2 = 0$.

Propoziția 3. Dacă P și Q sunt predicate calculabile, atunci $\neg P$, $P \vee Q$, $P \wedge Q$ sunt predicate calculabile.

Demonstrație: Este suficient să observăm că:

$$\neg P = \alpha^\circ P; \quad P \wedge Q = P \cdot Q; \quad P \vee Q = \neg((\neg P) \wedge (\neg Q))$$

Propoziția 4. Dacă g, h sunt funcții n -are (parțial) calculabile, iar P un predicat n -ar calculabil, atunci funcția n -ară:

$$f(x_1, \dots, x_n) = \begin{cases} g(x_1, \dots, x_n), & \text{dacă } P(x_1, \dots, x_n) = 1 \\ h(x_1, \dots, x_n), & \text{dacă } P(x_1, \dots, x_n) = 0 \end{cases}$$

este (parțial) calculabilă (observăm analogia cu construcția **if-then-else**).

Demonstrație: Într-adevăr, $f = g \cdot P + h \cdot (\alpha^\circ P)$.

Propoziția 5. Fie f o funcție calculabilă de aritate $n+1$. Atunci funcțiile g și h definite de:

$$g(x_1, \dots, x_n, x_{n+1}) = \sum_{t=0}^{x_{n+1}} f(x_1, \dots, x_n, t)$$

$$h(x_1, \dots, x_n, x_{n+1}) = \prod_{t=0}^{x_{n+1}} f(x_1, \dots, x_n, t)$$

sunt calculabile.

Demonstrație: Este suficient să observăm că:

$$\begin{cases} g(x_1, \dots, x_n, 0) = f(x_1, \dots, x_n, 0) \\ g(x_1, \dots, x_n, t+1) = g(x_1, \dots, x_n, t) + f(x_1, \dots, x_n, t+1) \end{cases}$$

$$\begin{cases} h(x_1, \dots, x_n, 0) = f(x_1, \dots, x_n, 0) \\ h(x_1, \dots, x_n, t+1) = h(x_1, \dots, x_n, t) \cdot f(x_1, \dots, x_n, t+1) \end{cases}$$

și să aplicăm propoziția 2.

Propoziția 6. Fie P un predicat calculabil de aritate $n+1$. Atunci sunt calculabile și următoarele predicate de aritate $n+1$ cu definiție evidentă:

$$(\forall t)_{\leq x_{n+1}} P(x_1, \dots, x_n, t)$$

$$(\exists t)_{\leq x_{n+1}} P(x_1, \dots, x_n, t)$$

$$(\forall t)_{< x_{n+1}} P(x_1, \dots, x_n, t)$$

$$(\exists t)_{< x_{n+1}} P(x_1, \dots, x_n, t)$$

Demonstrație: Demonstrația se bazează pe următoarele echivalențe:

$$(\forall t)_{\leq x_{n+1}} P(x_1, \dots, x_n, t) \Leftrightarrow \prod_{t=0}^{x_{n+1}} P(x_1, \dots, x_n, t) = 1$$

$$(\exists t)_{\leq x_{n+1}} P(x_1, \dots, x_n, t) \Leftrightarrow \sum_{t=0}^{x_{n+1}} P(x_1, \dots, x_n, t) \neq 0$$

$$(\forall t)_{< x_{n+1}} P(x_1, \dots, x_n, t) \Leftrightarrow (\forall t)_{\leq x_{n+1}} [t = x_{n+1} \vee P(x_1, \dots, x_n, t)]$$

$$(\exists t)_{< x_{n+1}} P(x_1, \dots, x_n, t) \Leftrightarrow (\exists t)_{\leq x_{n+1}} [t \neq x_{n+1} \wedge P(x_1, \dots, x_n, t)]$$

Exemplul 6. Predicatul binar următor (notat prin " $|$ "):

$$P(x_1, x_2) = 1 \Leftrightarrow x_1 | x_2$$

este calculabil (demonstrația este propusă ca exercițiu).

Exemplul 7. Predicatul unar *Prim*, definit prin:

$$\text{Prim}(x) = 1 \Leftrightarrow (x > 1) \wedge (\forall t)_{\leq x} [(t = 1) \vee \neg(t | x)].$$

este calculabil.

Propoziția 7. Fie P un predicat calculabil de aritate $n+1$. Atunci următoarea funcție de aritate $n+1$ este calculabilă:

$$\min_{t \leq x_{n+1}} P(x_1, \dots, x_n, t)$$

cu precizarea că dacă nu există nici un astfel de t , valoarea obținută va fi 0.

Demonstrație: Într-adevăr, funcția de mai sus este calculată de programul:

```
[A] if P(x1, ..., xn, t) goto B
    t ← t + 1
    if t ≤ xn+1 goto A
    y ← 0
    goto E
[B] y ← t
```

Rațiunea pentru care am ales prin lipsă valoarea 0 pentru minim va apărea în paragraful următor.

Exemplul 8. Funcția binară f ("partea întreagă din raport") dată de:

$$f(x_1, x_2) = \lfloor x_1 / x_2 \rfloor$$

este calculabilă.

Într-adevăr,

$$\lfloor x_1 / x_2 \rfloor = \min_{t \leq x_1} [(t+1)x_2 > x_1]$$

Să observăm că am acceptat că $\lfloor x / 0 \rfloor = 0$

Exemplul 9. Funcția binară R ce calculează restul împărțirii a două numere naturale este calculabilă.

Într-adevăr, $R(x_1, x_2) = x_1 - (\lfloor x_1 / x_2 \rfloor \cdot x_2)$. Să observăm că am acceptat că $R(x, 0) = x$.

Exemplul 10. Funcția unară definită prin:

$$f(n) = \begin{cases} \text{al } n - \text{lea număr prim, dacă } n > 0 \\ 0, \text{ dacă } n = 0 \end{cases}$$

este calculabilă.

Vom nota $p_n = f(n)$, deci $p_0 = 0, p_1 = 2, p_2 = 3, p_3 = 5, p_4 = 7, p_5 = 11, \dots$.

Într-adevăr, pentru $n > 0$ avem:

$$p_{n+1} = \min_{t \leq p_n!+1} [Prim(t) \wedge (t > p_n)]$$

Această egalitate se bazează pe inegalitatea $p_{n+1} \leq p_n! + 1$, care este adevărată deoarece fie $p_n! + 1$ este prim, fie el are un divizor prim mai mic decât el și care nu poate fi nici unul dintre p_1, \dots, p_n .

Definim acum succesiv funcțiile h și k , evident calculabile, date de:

$$h(y, z) = \min_{t \leq z} [Prim(t) \wedge (t > y)]$$

$$k(x) = h(x, x! + 1)$$

Rezultă:

$$\begin{cases} p_0 = 0 \\ p_{n+1} = k(p_n), \forall n \geq 0 \end{cases}$$

și conform propoziției 2, funcția este calculabilă.

2.5. Tema de autoinstruire nr. 2

Consultați bibliografia pentru:

1. a cunoaște exemple de funcții primitiv recursive;
2. a cunoaște funcția lui Ackermann și proprietățile sale.

2.6. Testul de autoevaluare nr. 2

Răspunsurile la test se vor da în spațiul liber aflat în continuarea enunțurilor!

1. Care este legătura dintre funcțiile primitiv recursive și cele calculabile?
2. Demonstrați că funcția $h: \mathbb{N} \rightarrow \mathbb{N}$, $h(n) = n!$ este calculabilă.
3. Demonstrați că predicatul $f: \mathbb{N} \times \mathbb{N} \rightarrow \{0,1\}$ (predicatul "=") dat de relația:
$$f(x_1, x_2) = \begin{cases} 1, & \text{dacă } x_1 = x_2 \\ 0, & \text{altfel} \end{cases}$$
este calculabil.

Răspunsurile la acest test se găsesc pe pagina următoare!

2.7. Comentarii și răspunsuri la testul nr. 2 de autoevaluare

1. Orice funcție primitiv recursivă este calculabilă, dar reciproca nu este neapărat adevărată.

2.

$$\begin{cases} h(0) = 1 \\ h(t+1) = h(t) \cdot (t+1) = g(t, h(t)), t > 0 \end{cases}, \text{ unde } g(x_1, x_2) = (x_1 + 1)x_2$$

3. $f(x_1, x_2) = \alpha(|x_1 - x_2|)$, unde α este predicatul prezentat în exemplul 4.

2.8. Lucrare de verificare pentru studenți

1. Demonstrați că funcția $f: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ dată de

$$f(x_1, x_2) = x_1 \dot{-} x_2 = \begin{cases} x_1 - x_2, & \text{dacă } x_1 \geq x_2 \\ 0, & \text{dacă } x_1 < x_2 \end{cases}$$

este calculabilă.

2. Demonstrați că predicatul binar notat prin "||" (divide) și definit prin $P(x_1, x_2) = 1 \iff x_1 | x_2$ este calculabil.

3. Demonstrați că funcția $f: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$, $f(x_1, x_2) = x_1 x_2 + 1$ este primitiv recursivă.

Rezolvările, dar și dificultățile întâmpinate, vor fi expediate prin email tutorelui.

2.9. Bibliografie

1. C. Calude – "Complexitatea calculului – aspecte calitative", Editura Științifică și Enciclopedică, 1962.
2. M.D. Davis, R. Sigal, E.J. Weyuker – "Computability, Complexity and Languages", Academic Press (Morgan Kaufmann), 1994.
3. N.D. Jones – "Computability and Complexity", MIT Press, 1997.
4. Christos H. Papadimitriou – "Computational Complexity", Addison-Wesley, 1994.
5. C.P. Popovici, S. Rudeanu, H. Georgescu – "Bazele informaticii", Vol.II, Tipografia Universității București, 1991.

UNITATEA DE ÎNVĂȚARE NR. 3

CODIFICAREA PROGRAMELOR

Cuprins

3.1.	Obiectivele unității de învățare nr. 3	22
3.2.	Indicații metodice pentru unitatea de învățare nr. 3	22
3.3.	Reprezentarea perechilor și n-uplelor ca un singur număr.....	22
3.4.	Numărul atașat unui program.....	24
3.5.	Tema de autoinstruire nr. 3	26
3.6.	Testul de autoevaluare nr. 3	26
3.7.	Comentarii și răspunsuri la testul nr. 3 de autoevaluare.....	27
3.8.	Lucrare de verificare pentru studenți	27
3.9.	Bibliografie	28

3.1. Obiectivele unității de învățare nr. 3

După ce veți parcurge această unitate de învățare, veți reuși să:

- înțelegeți noțiunea de funcție primitivă și să cunoașteți mai multe proprietăți ale sale;
- demonstrați că o funcție este primitiv recursivă și, implicit, calculabilă.

3.2. Indicații metodice pentru unitatea de învățare nr. 3

Materialul trebuie parcurs în ordinea sa firească, prezentată în continuare. Se recomandă conspectarea și notarea ideilor principale, precum și consultarea bibliografiei pentru detalii și informații suplimentare.

Timpul minim pe care trebuie să-l acordați acestei unități de învățare este de **8 ore**.

3.3. Reprezentarea perechilor și n-uplelor ca un singur număr

Pentru orice două numere naturale $x, y \in \mathbb{N}$, definim $\langle x, y \rangle = 2^x(2y + 1) - 1$. Funcția $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ dată de $f(x, y) = \langle x, y \rangle$ este bijectivă.

Într-adevăr pentru orice $z \in \mathbb{N}$, există și sunt unice numerele x, y cu $\langle x, y \rangle = z$, astfel:

- 1) x este cea mai mare putere a lui 2 ce divide pe $z + 1$;
- 2) y poate fi determinat de relația $2y + 1 = \frac{z+1}{2^x}$ (membrul drept al acestei relații fiind impar).

Notând prin ℓ și r funcțiile prin care se obțin x și y din z (adică $\langle \ell(z), r(z) \rangle = z$), observăm că ele sunt calculabile deoarece:

$$\begin{cases} \ell(z) = \min_{x \leq z} [\neg(2^{x+1} \mid (z+1))] \\ r(z) = \left\lfloor \left(\left\lfloor (z+1)/2^x \right\rfloor - 1 \right) / 2 \right\rfloor \end{cases}$$

Pentru reprezentarea n -uplului $(a_1, \dots, a_n) \in \mathbb{N}^n$ definim **numărul Gödel** atașat astfel:

$$[a_1, \dots, a_n] = \prod_{i=1}^n p_i^{a_i}$$

unde $p_1 = 2, p_2 = 3, p_3 = 5, \dots$ este sirul numerelor prime.

De exemplu, numărul Gödel $[2, 0, 1] = 2^2 \cdot 5^1 = 20 = [2, 0, 1, 0, \dots, 0]$.

Pentru orice $n \geq 1$, funcția n -ară f dată de $f(x_1, \dots, x_n) = [x_1, \dots, x_n]$ este evident calculabilă. Din teorema fundamentală a aritmeticii deducem că:

- 1) pentru orice x nenul, există n, a_1, \dots, a_n cu $[a_1, \dots, a_n] = x$ (deci funcția f este surjectivă);
- 2) din $[a_1, \dots, a_n] = [b_1, \dots, b_n]$ rezultă $a_i = b_i, \forall i = 1, \dots, n$.

Mai observăm că $[a_1, \dots, a_n] = [a_1, \dots, a_n, 0]$. Prin urmare funcția ce atașează oricărui program numărul său Gödel nu este injectivă.

De exemplu: $1 = [0] = [0, 0] = [0, 0, 0] = \dots [0, 0, \dots, 0]$, ceea ce ne permite să definim numărul Gödel atașat secvenței vide (cu $n = 0$) ca fiind 1.

Pentru orice $i \geq 1$, considerăm funcția $(\)_i : \mathbb{N} \rightarrow \mathbb{N}$ care pentru fiecare număr $x = [a_1, a_2, \dots]$ calculează $(x)_i = a_i$. Aceste funcții sunt calculabile deoarece:

$$(x)_i = \min_{t \leq x} (\neg(p_i^{t+1} \mid x))$$

Să observăm că $(0)_i = (1)_i = 0, \forall i$.

Mai introducem funcția $Lt : \mathbb{N} \rightarrow \mathbb{N}$ care atașează fiecărui număr natural x numărul de ordine al celui mai mare număr prim care îl divide pe x . În exemplul de mai sus pentru $x = 20 = [2, 0, 1, 0, \dots, 0]$ avem $Lt(x) = 3$.

Funcția Lt este calculabilă deoarece:

$$Lt(x) = \min_{i \leq x} \left[((x)_i \neq 0) \wedge (\forall j)_{j \leq x} ((j \leq i) \vee ((x)_j = 0)) \right]$$

(se caută cel mai mic i cu $x_i \neq 0$ și $(x)_j = 0, \forall j > i$).

Să observăm că:

- 1) $Lt(0) = Lt(1) = 0$;
- 2) $Lt([a_1, \dots, a_n]) = n \Leftrightarrow a_n \neq 0$ (unde $n \geq 1$) ;
- 3) pentru orice $z \in \mathbb{N} \setminus \{0, 1\}$, există și sunt unice n, a_1, \dots, a_n cu $a_n \neq 0$ și $[a_1, \dots, a_n] = z$ (pentru $z = 1$ se obține $n = 0$, adică secvența vidă).

3.4. Numărul atașat unui program

Începem prin a așeza variabilele de intrare (x_1, x_2, \dots) , variabila de ieșire y și variabilele de lucru (z_1, z_2, \dots) în următoarea ordine: $y, x_1, z_1, x_2, z_2, \dots$. Variabilelor le atașăm pozițiile lor în acest sir prin funcția # astfel:

$$\#(y) = 1; \#(x_i) = 2i; \#(z_i) = 2i + 1, \forall i \geq 1.$$

Fiecare instrucțiune poate avea atașată o etichetă. Vom numerota etichetele folosite în program cu E_1, E_2, \dots . Extindem funcția # la etichete astfel :

$$\#(E_i) = i.$$

În continuare observăm că o instrucțiune I în limbajul s este bine determinată de: etichetarea ei, tipul instrucțiunii și unica variabilă v ce intervine în ea.

Corespunzător, pentru fiecare instrucțiune definim trei numere naturale a, b, c astfel:

$$a = \begin{cases} 0, & \text{dacă } I \text{ nu este etichetată} \\ \#(L), & \text{dacă } I \text{ este etichetată cu } L \end{cases}$$

$$b = \begin{cases} 0, & \text{dacă } I \text{ este de tipul } v \leftarrow v \\ 1, & \text{dacă } I \text{ este de tipul } v \leftarrow v + 1 \\ 2, & \text{dacă } I \text{ este de tipul } v \leftarrow v - 1 \\ \#(L) + 2, & \text{dacă } I \text{ este de tipul } \mathbf{if } v \neq 0 \mathbf{ goto } L \end{cases}$$

$$c = \#(v) - 1$$

unde v este unica variabilă ce apare în instrucțiunea I .

Putem extinde acum funcția # la instrucțiuni astfel:

$$\#(I) = \langle a, \langle b, c \rangle \rangle$$

unde prin $\#(I)$ am notat numărul atașat instrucțiunii I .

Exemplul 1. Fie I următoarea instrucțiune:

$$[E_1] \quad x_1 \leftarrow x_1 + 1$$

Atunci $a = 1, b = 1, c = 1$ și deci $\#(I) = \langle 1, \langle 1, 1 \rangle \rangle = \langle 1, 5 \rangle = 21$.

Restricția funcției $\#$ la mulțimea instrucțiunilor este bijectivă, deoarece funcția \langle , \rangle este bijectivă.

Să mai observăm că unica instrucțiune I cu $\#(I) = 0$ este instrucțiunea neetichetată $y \leftarrow y$.

Fie acum un program P constând, în ordine, din instrucțiunile I_1, \dots, I_n . Extindem atunci funcția $\#$ la programe astfel:

$$\#(P) = [\#(I_1), \dots, \#(I_n)] - 1$$

unde prin $\#(P)$ am notat numărul atașat programului P . Funcția $\#$ definită pe mulțimea programelor este calculabilă.

Înținând cont de observațiile făcute anterior asupra numărului Gödel atașat unei secvențe, rezultă că restricția funcției $\#$ la programe devine bijectivă dacă ultima instrucțiune a programelor este diferită de instrucțiunea neetichetată $y \leftarrow y$. Cum efectul acesteia este nul, vom impune (fără a scădea din generalitate) regula următoare: *nici un program nu se poate termina cu instrucțiunea neetichetată $y \leftarrow y$* . În acest mod fiecare număr natural poate fi privit ca un (unic) program în limbajul s . O consecință imediată este că mulțimea programelor în limbajul s este numărabilă.

Exemplul 2. Căutăm programul P al cărui număr atașat este $\#(P) = 199$.

Cum $199 + 1 = 200 = 2^3 \cdot 5^2 = [3, 0, 2]$, rezultă că programul P este format din 3 instrucțiuni, ale căror numere atașate sunt în ordine 3, 0 și 2.

$$\#(I_1) = 3 = \langle a_1, \langle b_1, c_1 \rangle \rangle \Rightarrow a_1 = 2 \text{ și } \langle b_1, c_1 \rangle = 0 \Rightarrow b_1 = c_1 = 0, \text{ deci } I_1 \text{ este } [E_2] \quad y \leftarrow y + 1.$$

$$\#(I_2) = 0, \text{ deci } I_2 \text{ este } y \leftarrow y.$$

$$\#(I_3) = 2 = \langle a_3, \langle b_3, c_3 \rangle \rangle \Rightarrow a_3 = 0 \text{ și } \langle b_3, c_3 \rangle = 1 \Rightarrow b_3 = 1 \text{ și } c_3 = 0, \text{ deci } I_3 \text{ este } y \leftarrow y + 1.$$

Rezultă că programul P căutat este următorul:

$$\begin{aligned} [E_2] \quad & y \leftarrow y \\ & y \leftarrow y \\ & y \leftarrow y + 1 \end{aligned}$$

Să mai facem următoarele observații:

- 1) programul vid are numărul atașat egal cu 0;
- 2) corespondențele de mai sus sunt cele care au făcut necesare admiterea etichetării unor instrucțiuni diferite din același program cu aceeași etichetă.

3.5. Tema de autoinstruire nr. 3

Consultați bibliografia pentru:

1. a cunoaște mai multe proprietăți ale numărului Gödel;
2. a cunoaște mai multe exemple de codificarea a unui program scris în limbajul S.

3.6. Testul de autoevaluare nr. 3

Răspunsurile la test se vor da în spațiul liber aflat în continuarea enunțurilor!

1. Cu cât sunt egale expresiile $\langle x, 0 \rangle$ și $\langle 0, y \rangle$?
2. Determinați numărul atașat următorului program:

```
[E3] x1 ← x1 - 1  
      x2 ← x2 - 1  
      y ← y + 1  
      if x2 ≠ 0 goto E3
```

3. Determinați programul cu numărul atașat 799999.

Răspunsurile la acest test se găsesc pe pagina următoare!

3.7. Comentarii și răspunsuri la testul nr. 3 de autoevaluare

1. $\langle x, 0 \rangle = 2^x - 1$ și $\langle 0, y \rangle = 2y$

2.

$$\left. \begin{array}{l} \#(I_1) = \langle 3, \langle 2, 1 \rangle \rangle = \langle 3, 11 \rangle = 183 \\ \#(I_2) = \langle 0, \langle 2, 3 \rangle \rangle = \langle 0, 27 \rangle = 54 \\ \#(I_3) = \langle 0, \langle 1, 0 \rangle \rangle = \langle 0, 1 \rangle = 2 \\ \#(I_4) = \langle 0, \langle 5, 3 \rangle \rangle = \langle 0, 223 \rangle = 446 \end{array} \right\} \Rightarrow \#(P) = 2^{183} \cdot 3^{54} \cdot 5^2 \cdot 7^{446} - 1$$

3. $\#(P) = [\#(I_1), \#(I_2), \dots, \#(I_n)] - 1 \Rightarrow [\#(I_1), \dots, \#(I_n)] = 800000 = 2^8 \cdot 5^5 = [8, 0, 5] \Rightarrow$

$$\left. \begin{array}{l} \#(I_1) = 8 \Rightarrow \langle a_1, \langle b_1, c_1 \rangle \rangle = 8 \Rightarrow \begin{cases} a_1 = 0 \\ \langle b_1, c_1 \rangle = 4 \Rightarrow \begin{cases} b_1 = 0 \\ c_1 = 2 \end{cases} \end{cases} \Rightarrow I_1 \text{ este } z_1 \leftarrow z_1 \\ \#(I_2) = 0 \Rightarrow I_2 \text{ este } y \leftarrow y \\ \#(I_3) = 5 \Rightarrow \langle a_3, \langle b_3, c_3 \rangle \rangle = 5 \Rightarrow \begin{cases} a_3 = 1 \\ \langle b_3, c_3 \rangle = 1 \Rightarrow \begin{cases} b_3 = 1 \\ c_3 = 0 \end{cases} \end{cases} \Rightarrow I_3 \text{ este } [E_1] y \leftarrow y + 1 \end{array} \right\}$$

Deci programul cu numărul atașat 799999 este următorul:

```

 $z_1 \leftarrow z_1$ 
 $y \leftarrow y$ 
 $[E_1] \quad y \leftarrow y + 1$ 

```

3.8. Lucrare de verificare pentru studenți

1. Demonstrați că $\#(I) = 0 \Leftrightarrow$ instrucțunea I este $y \leftarrow y$.

2. Determinați numărul atașat următorului program, precum și funcția calculată de el:

```

 $[E_1]$  if  $x_1 \neq 0$  goto  $E_2$ 
 $z_1 \leftarrow z_1 + 1$ 
if  $z_1 \neq 0$  goto  $E_3$ 
 $[E_2]$   $y \leftarrow y$ 
 $x_1 \leftarrow x_1 - 1$ 
 $z_2 \leftarrow z_2 + 1$ 
if  $z_2 \neq 0$  goto  $E_1$ 

```

3. Determinați programul având numărul atașat $\#(P) = 2^{187} \cdot 3^{18} \cdot 5^{318} \cdot 7^{11} \cdot 11^{22} \cdot 13^{34} \cdot 17^{142} - 1$.

Rezolvările, dar și dificultățile întâmpinate, vor fi expediate prin email tutorelui.

3.9. Bibliografie

1. C. Calude – "*Complexitatea calculului – aspecte calitative*", Editura Științifică și Enciclopedică, 1962.
2. M.D. Davis, R. Sigal, E.J. Weyuker – "*Computability, Complexity and Languages*", Academic Press (Morgan Kaufmann), 1994.
3. N.D. Jones – "*Computability and Complexity*", MIT Press, 1997.
4. Christos H. Papadimitriou – "*Computational Complexity*", Addison-Wesley, 1994.
5. C.P. Popovici, S. Rudeanu, H. Georgescu – "*Bazele informaticii*", Vol.II, Tipografia Universității București, 1991.

UNITATEA DE ÎNVĂȚARE NR. 4

TEZA LUI CHURCH. MULTIMI CALCULABILE

Cuprins

4.1.	Obiectivele unității de învățare nr. 4	29
4.2.	Indicații metodice pentru unitatea de învățare nr. 4	29
4.3.	Teza lui Church	29
4.4.	Programul universal	31
4.5.	Mulțimi calculabile	33
4.6.	Teorema lui Rice	36
4.7.	Tema de autoinstruire nr. 4	38
4.8.	Testul de autoevaluare nr. 4	38
4.9.	Comentarii și răspunsuri la testul nr. 4 de autoevaluare	39
4.10.	Lucrare de verificare pentru studenți	39
4.11.	Bibliografie	39

4.1. Obiectivele unității de învățare nr. 4

După ce veți parurge această unitate de învățare, veți reuși să:

- cunoașteți teza lui Church;
- cunoașteți exemple de probleme decidabile și, respectiv, de probleme nedecidabile.

4.2. Indicații metodice pentru unitatea de învățare nr. 4

Materialul trebuie parcurs în ordinea sa firească, prezentată în continuare. Se recomandă conspectarea și notarea ideilor principale, precum și consultarea bibliografiei pentru detalii și informații suplimentare.

Timpul minim pe care trebuie să-l acordați acestei unități de învățare este de **8 ore**.

4.3. Teza lui Church

Teza lui Church (1936) se exprimă astfel: *Date fiind numerele naturale x_1, \dots, x_n , numărul y poate fi "calculat" pe baza lor dacă și numai dacă există un program în limbajul s având la intrare valorile x_1, \dots, x_n și la ieșire valoarea y .*

Altfel spus, înțelegem prin *algoritm* ce calculează valoarea y plecând de la valorile x_1, \dots, x_n un program în limbajul s ce realizează acest lucru.

Evident, se pune problema dacă definiția cuvântului "algoritm" dată mai sus nu este prea restrictivă. Legat de aceasta, se impun următoarele precizări:

- 1) noțiunea de algoritm nu poate fi definită decât pe baza unui limbaj de programare particular sau a unei "mașini matematice ideale"; de aceea teza lui Church nu poate fi demonstrată ca o teoremă din matematică;
- 2) toate încercările de a defini noțiunea de algoritm, încercări dintre care unele vor fi prezentate în continuare, au condus la definiții ce s-au dovedit echivalente cu cea din enunțul tezei lui Church.

Aceste considerații ne determină să acceptăm definiția algoritmului aşa cum apare ea în teza lui Church.

Suntem acum în măsură să prezentăm o primă *problemă nedecidabilă*, adică o problemă pentru care nu există un program în limbajul s care să o rezolve. Este vorba de *problema opririi* (terminării) programelor.

Fie HALT predicatul binar definit astfel:

$$\text{HALT}(x_1, x_2) = 1 \Leftrightarrow \text{programul } P \text{ cu } \#(P) = x_2 \text{ se termină pentru valoarea de intrare } x_1.$$

Teorema 1. Predicatul HALT nu este calculabil.

Demonstrație. Să presupunem prin absurd că predicatul HALT ar fi calculabil. Atunci putem considera următorul program (notat prin P) ce constă din unica instrucție:

[A] if $\text{HALT}(x, x)$ goto A

Atunci funcția de un argument calculată de P este:

$$\psi_P^{(1)}(x) = \begin{cases} \uparrow, & \text{dacă } \text{HALT}(x, x) = 1 \\ 0, & \text{dacă } \text{HALT}(x, x) = 0 \end{cases}$$

Fie $y_0 = \#(P)$. Atunci avem că $\text{HALT}(x, y_0) = 1 \Leftrightarrow \psi_P^{(1)}(x) \downarrow$ (este definită) $\Leftrightarrow \text{HALT}(x, y_0) = 0$ (programul cu numărul y_0 se termină pentru valoarea x).

Punând $x = y_0$, obținem $\text{HALT}(y_0, y_0) = 1 \Leftrightarrow \text{HALT}(y_0, y_0) = 0$, ajungându-se astfel la o contradicție.

Cum predicatul HALT nu este calculabil, conform tezei lui Church ajungem la următorul rezultat:

Corolar. *Problema opririi programelor este nedecidabilă*, în sensul că nu există un algoritm care, pentru orice program scris în limbajul s și orice valori de intrare, să decidă dacă programul se termină pentru acele date de intrare.

4.4. Programul universal

Pentru fiecare $m > 0$, definim următoarea funcție de $m + 1$ argumente:

$$\Phi^{(m)}(x_1, \dots, x_n, x_{m+1}) = \psi_P^{(m)}(x_1, \dots, x_m),$$

unde P este programul cu $\#(P) = x_{m+1}$.

Teorema 2 (Teorema de universalitate). Pentru orice $m > 0$, funcția $\Phi^{(m)}$ este parțial calculabilă.

Demonstrație:

Demonstrația va consta în a scrie, pentru fiecare $m > 0$, un program U_m (numit *program universal*) care determină din x_{m+1} programul P cu $\#(P) = x_{m+1}$ și lucrează conform lui pentru valorile de intrare x_1, \dots, x_m . Presupunând U_m construit, vom avea:

$$\psi_{U_m}^{(m+1)}(x_1, \dots, x_m, x_{m+1}) = \Phi^{(m)}(x_1, \dots, x_{m+1})$$

ceea ce arată că funcția $\Phi^{(m)}$ este calculabilă.

Pentru fiecare $m > 0$, programul U_m corespunzător folosește printre altele următoarele variabile de lucru:

- k este numărul instrucțiunii ce urmează a fi executată, cu precizarea că valoarea lui k va deveni 0 în urma executării unei instrucțiuni de transfer în care apare o etichetă inexistentă;
- s este numărul natural asociat stării curente a programului: $s = [y, x_1, z_1, x_2, z_2, \dots]$.

Putem scrie acum programul U_m , însotit în partea dreaptă de comentarii asupra unei instrucțiuni sau a unui grup de instrucțiuni:

$z \leftarrow x_{m+1} + 1$	$z \leftarrow [\#(I_1), \#(I_2), \dots] = \#(P) + 1$
$s \leftarrow \prod_{i=1}^m p_{2i}^{x_i}$	starea inițială este $z = [0, x_1, 0, x_2, 0, \dots]$
$k \leftarrow 1$	se începe cu prima instrucțiune
[R] if $k = Lt(z) + 1 \vee k = 0$ goto E	programul se termină dacă se trece de ultima instrucțiune din program sau se face transfer la instrucțiune inexistentă
$d \leftarrow r((z)_k)$	se decodifică instrucțiunea k : $z_k = \langle a, d \rangle$ cu $d = \langle b, c \rangle$
$b \leftarrow l(d)$ $c \leftarrow r(d)$	a , ce identifică eticheta instrucțiunii I_k , nu intervine în executarea instrucțiunii

$q \leftarrow p_{c+1}$	$q =$ numărul prim corespunzător variabilei v din instrucțiunea curentă, deci $\#(v) =$ numărul de ordine al lui q în sirul numerelor prime
if $b \neq 0$ goto A	
$k \leftarrow k + 1$ goto R	instrucțiunea k din P este $v \leftarrow v$
[A] if $b \neq 1$ goto B	
$s \leftarrow s \cdot q$ $k \leftarrow k + 1$ goto R	instrucțiunea k din P este $v \leftarrow v + 1$
[B] if $b \neq 2$ goto D	
if $\neg(q s)$ goto C $s \leftarrow \lfloor s / q \rfloor$	instrucțiunea k din P este $v \leftarrow v - 1$ și se ține cont că $v \neq 0 \Leftrightarrow q s$
[C] $k \leftarrow k + 1$ goto R	
[D] if $q s$ goto F $k \leftarrow k + 1$ goto R	instrucțiunea k din P este if $v \neq 0$ goto L cu $\#(L) = b - 2$ și se ține cont că $v \neq 0 \Leftrightarrow q s$ și că dacă $v \neq 0$ se trece la prima instrucțiune I_i etichetată cu L ; mai reamintim că pentru multimea vidă, minimul este 0.
[F] $k \leftarrow \min_{i \leq Lt(z)} [l((z)_i) = b - 2]$ goto R	
[E] $y \leftarrow (s)_1$	

Cu aceasta teorema de universalitate este demonstrată.

Să observăm că teorema ne asigură că a lucra cu numărul asociat unui program în loc de a lucra cu acel program ne păstrează în sfera decidibilității.

În cele ce urmează vom folosi și notația:

$$\Phi_y^{(m)}(x_1, \dots, x_m) = \Phi^{(m)}(x_1, \dots, x_m, y) \text{ și deci } \Phi_y^{(m)} = \psi_p^{(m)}, \text{ unde } \#(P) = y.$$

Reamintim că s-a demonstrat anterior că problema opririi programelor este nedecidabilă. În continuare arătăm că *problema opririi după cel mult t pași (cu t fixat) este decidabilă*.

Considerăm predicatul de aritate $m + 2$ următor:

$$STEP^{(m)}(x_1, \dots, x_m, x_{m+1}, x_{m+2}) = 1 \Leftrightarrow \text{programul } P \text{ cu } \#(P) = x_{m+1} \text{ se oprește după cel mult } x_{m+2} \text{ pași pentru intrările } x_1, \dots, x_m.$$

Teorema 3 (Teorema contorului). Predicatul $STEP^{(m)}$ este calculabil.

Demonstrație. Pentru demonstrație, vom modifica programul universal U_m astfel încât să se execute cel mult t pași (instrucțiuni) din programul P cu $\#(P) = x_{m+1}$.

Dacă P se oprește după cel mult t pași, valoarea de ieșire y va fi 1, iar în caz contrar y va lua valoarea 0. În acest scop se va folosi un contor v care va număra câte instrucțiuni din P au fost executate.

Modificările în programul U_m sunt următoarele:

- 1) instrucțiunea etichetată cu R se înlocuiește cu următoarele:

```
[R] if (k = Lt(z) + 1) ∨ (k = 0) goto E
    if v ≥ xm+2 goto G
    v ← v + 1
```

- 2) instrucțiunea etichetată cu E devine:

```
[E] y ← 1
```

Instrucțiunea etichetată cu G nu va exista în program, deci în acest caz valoarea de ieșire va fi 0.

4.5. Mulțimi calculabile

Fie $B \subset \mathbb{N}$ o submulțime a mulțimii numerelor naturale \mathbb{N} . Submulțimii B îi atașăm un predicat $\chi_B : \mathbb{N} \rightarrow \{0,1\}$, numit *funcția caracteristică a submulțimii B* , astfel:

$$\chi_B(x) = \begin{cases} 1, & \text{dacă } x \in B \\ 0, & \text{dacă } x \notin B \end{cases}$$

Spunem că (sub)mulțimea B este *calculabilă (recursivă)* dacă predicatul χ_B este calculabil. Altfel spus, B este calculabilă dacă și numai dacă există un algoritm care stabilește pentru orice număr natural x dacă $x \in B$ sau $x \notin B$.

Propoziția 1. Dacă B, C sunt mulțimi calculabile, atunci și $B \cap C$, $B \cup C$, \overline{B} (unde $\overline{B} = \mathbb{N} \setminus B$) sunt mulțimi calculabile.

Este suficient să observăm că:

$$\chi_{B \cup C} = \chi_B \vee \chi_C, \chi_{B \cap C} = \chi_B \cdot \chi_C, \chi_{\overline{B}} = \alpha^\circ \chi_B,$$

unde funcția α a fost descrisă în unitatea de învățare nr.2.

Propoziția 2. Există mulțimi care nu sunt calculabile.

Fie $K = \{x \in \mathbb{N} \mid \Phi_x^{(1)}(x) \downarrow\}$. Atunci $\chi_K(x) = 1 \Leftrightarrow \Phi_x^{(1)}(x) \downarrow \Leftrightarrow HALT(x, x) = 1$. Presupunem că predicatul χ_K este calculabil, rezultă că putem considera următorul program P :

[A] **if** $\chi_K(x)$ **goto** A

Evident, avem

$$\psi_P^{(1)}(x) = \begin{cases} \uparrow, & \text{dacă } \chi_K(x) = 1 \\ 0, & \text{dacă } \chi_K(x) = 0 \end{cases}$$

Fie $y_0 = \#(P)$. Atunci $\chi_K(y_0) = 1 \Leftrightarrow (\text{din definiția funcției } \chi_K) \Leftrightarrow \Phi_{y_0}^{(1)}(y_0) \downarrow \Leftrightarrow \psi_P^{(1)}(y_0) \downarrow \Leftrightarrow (\text{conform modului în care acționează } \psi_P^{(1)}) \Leftrightarrow \chi_K(y_0) = 0$ și am ajuns la o contradicție. Rezultă că χ_K nu este calculabilă, deci mulțimea K nu este calculabilă.

Teorema 1. (Teorema s-m-n). Pentru orice $m, n > 0$, există o funcție calculabilă S_m^n de $n + 1$ argumente cu proprietatea:

$$\Phi^{(m+n)}(x_1, \dots, x_m, u_1, \dots, u_n, y) = \Phi^{(m)}(x_1, \dots, x_m, S_m^n(u_1, \dots, u_n, y))$$

pentru orice valori ale argumentelor funcției $\Phi^{(m+n)}$.

Semnificația teoremei este următoarea: dacă fixăm pe u_1, \dots, u_n, y , atunci $\Phi^{(m+n)}$ se transformă într-o funcție, tot calculabilă, de m argumente, deci există un program P care o calculează.

Teorema afirmă nu numai că $q = \#(P)$ există, dar și că poate fi obținut într-un mod calculabil din u_1, \dots, u_n, y conform unei funcții S_m^n calculabile în argumentele u_1, \dots, u_n, y .

Demonstrăm teorema prin inducție după n .

Pentru $n = 1$, căutăm o funcție S_1^1 cu proprietatea că:

$$\Phi^{(m+1)}(x_1, \dots, x_m, u, y) = \Phi^{(m)}(x_1, \dots, x_m, S_m^1(u, y)),$$

adică pentru orice u, y fixați căutăm numărul $S_m^1(u, y)$ al programului P' care pentru orice intrări x_1, \dots, x_m calculează același lucru ca și programul P cu $\#(P) = y$ aplicat intrărilor x_1, \dots, x_m, u .

Evident, programul P' va consta în instrucțiunile prin care x_{m+1} primește valoarea lui u , urmate de instrucțiunile programului P :

$$P' \left\{ \begin{array}{l} x_{m+1} \leftarrow x_{m+1} + 1 \\ \cdots \\ x_{m+1} \leftarrow x_{m+1} + 1 \\ \cdots \\ P \end{array} \right.$$

(unde instrucțiunea $x_{m+1} \leftarrow x_{m+1} + 1$ se repetă de u ori).

Cum $\#(x_{m+1}) = 2m + 2$, atunci numărul atașat fiecăreia dintre primele u instrucțiuni este $\langle 0, \langle 1, 2m + 1 \rangle \rangle = \langle 0, 8m + 5 \rangle = 16m + 10$.

Fie k numărul de instrucțiuni ale programului P și fie în ordine $\alpha_1, \dots, \alpha_k$ numerele atașate acestor instrucțiuni prin funcția $\#$ (deci $y = p_1^{\alpha_1} \dots p_k^{\alpha_k} - 1$ și $k = Lt(y + 1)$), rezultă că

$$S_m^1(u, y) = \#(P') = \left[\left(\prod_{i=1}^u p_i \right)^{16m+10} \cdot \prod_{j=1}^k p_{u+j}^{\alpha_j} \right] - 1$$

și deci S_m^1 este o funcție calculabilă.

Presupunem acum că rezultatul este adevărat pentru un $n \geq 1$ dat. Atunci:

$$\begin{aligned} \Phi^{(m+n+1)}(x_1, \dots, x_m, u_1, \dots, u_n, u_{n+1}, y) &= (\text{aplic teorema pentru } n=1) \\ &= \Phi^{(m+n)}(x_1, \dots, x_m, u_1, \dots, u_n, S_{m+n}^1(u_{n+1}, y)) = (\text{conform ipotezei de inducție}) \\ &= \Phi^{(m)}(x_1, \dots, x_m, S_m^n(u_1, \dots, u_n, S_{m+n}^1(u_{n+1}, y))). \end{aligned}$$

Este suficient acum să luăm:

$$S_m^{n+1}(u_1, \dots, u_{n+1}, y) = S_m^n(u_1, \dots, u_n, S_{m+n}^1(u_{n+1}, y))$$

deoarece S_m^{n+1} este calculabilă conform proprietății de închidere la compunere a clasei funcțiilor calculabile.

Teorema 2 (de recursie). Fie g o funcție parțială calculabilă de $m+1$ variabile. Există atunci un număr e cu proprietatea:

$$\Phi_e^{(m)}(x_1, \dots, x_m) = g(e, x_1, \dots, x_m)$$

(adică există un număr natural e astfel încât programul cu numărul e calculează funcția $g(x_1, \dots, x_m) = g(e, x_1, \dots, x_m)$).

Considerăm funcția calculabilă \bar{g} de $m+1$ argumente dată de:

$$\bar{g}(u, x_1, \dots, x_m) = g(S_m^1(u, u), x_1, \dots, x_m).$$

Cum \bar{g} este calculabilă, există $z_0 \in \mathbb{N}$ cu proprietatea:

$$\begin{aligned}\bar{g}(u, x_1, \dots, x_m) &= \Phi^{(m+1)}(x_1, \dots, x_m, u, z_0) = (\text{conform teoremei precedente}) \\ &= \Phi^{(m)}(x_1, \dots, x_m, S_m^1(u, z_0)).\end{aligned}$$

Luând $u = z_0$ și notând $e = S_m^1(z_0, z_0)$, obținem: $\bar{g}(z_0, x_1, \dots, x_m) = \Phi^{(m)}(x_1, \dots, x_m, e)$, adică $g(e, x_1, \dots, x_m) = \Phi_e^{(m)}(x_1, \dots, x_m)$.

Fie acum Γ o subclasă a clasei funcțiilor parțiale calculabile de o variabilă. Îi atașăm mulțimea de numere naturale:

$$R_\Gamma = \{t \in \mathbb{N} \mid \Phi_t^{(1)} \in \Gamma\}.$$

Conform definiției mulțimilor calculabile, R_Γ este calculabilă dacă și numai dacă există un program care pentru fiecare $t \in \mathbb{N}$ determină dacă funcția $\Phi_t^{(1)}$ aparține sau nu lui Γ .

4.6. Teorema lui Rice

Prezentăm în continuare teorema lui Rice, fundamentală în domeniul calculabilității, precum și 3 corolare ale sale.

Teorema 3 (Rice). Fie Γ o subclasă de funcții parțiale calculabile de o variabilă și fie f, g funcții parțiale calculabile cu $f \in \Gamma, g \notin \Gamma$. Atunci R_Γ nu este calculabilă.

Presupunem prin absurd că R_Γ este calculabilă. Considerăm predicatul P_Γ și funcția h date de:

$$P_\Gamma(t) = \begin{cases} 1, & \text{dacă } t \in R_\Gamma \\ 0, & \text{altfel} \end{cases}$$

$$h(t, x) = \begin{cases} g(x), & \text{dacă } t \in R_\Gamma \\ f(x), & \text{altfel} \end{cases}$$

Se observă că $h(t, x) = g(x) \cdot P_\Gamma(t) + f(x) \cdot \alpha(P_\Gamma(t))$, ceea ce arată că h este parțială calculabilă. Atunci conform teoremei de recursie, există numărul $e \in \mathbb{N}$ cu proprietatea:

$$\Phi_e^{(1)}(x) = h(e, x) = \begin{cases} g(x), & \text{dacă } e \in R_\Gamma \\ f(x), & \text{altfel} \end{cases}$$

pentru toți $x \in \mathbb{N}$, adică

$$\Phi_e^{(1)} = \begin{cases} g, & \text{dacă } e \in R_\Gamma \\ f, & \text{altfel} \end{cases}$$

Observăm acum că:

$$\begin{aligned} e \in R_\Gamma &\Rightarrow \Phi_e^{(1)} = g \Rightarrow \Phi_e^{(1)} \notin \Gamma \Rightarrow e \notin R_\Gamma \\ e \notin R_\Gamma &\Rightarrow \Phi_e^{(1)} = f \Rightarrow \Phi_e^{(1)} \in \Gamma \Rightarrow e \in R_\Gamma \end{aligned}$$

ajungând astfel la o contradicție.

Corolarul 1. Nu există un algoritm care să determine pentru orice program P din limbajul S dacă $\psi_p^{(1)}$ este sau nu calculabilă.

Vom lua drept Γ clasa funcțiilor calculabile, iar drept f, g funcțiile de o variabilă date de $f(x) = x$, $g(x) = 1 - x$. Evident, $f \in \Gamma$, dar $g \notin \Gamma$ (deoarece domeniul lui g este $\{0,1\}$). Putem aplica acum teorema lui Rice și obținem că R_Γ nu este calculabilă și ajungem tocmai la afirmația din enunțul corolarului.

Corolarul 1 spune că problema "să se determine pentru un program oarecare dacă el se termină pentru orice valoare de intrare" este nedecidabilă.

Observație. Rezultatul este diferit de cel din Teorema 1 (predicatul $HALT$ nu este calculabil) deoarece aici problema are ca date de intrare programe, pe când în Teorema 1 problema avea ca date de intrare programe însotite de valori de intrare.

Corolarul 2. Fie f o funcție (parțială) calculabilă de o variabilă. Atunci nu există un algoritm pentru următoarea problemă: să se determine pentru orice program P dacă $\psi_p^{(1)} = f$.

Este suficient să luăm $\Gamma = \{f\}$. Evident $g \notin \Gamma$, unde $g(x) = f(x) + 1$. Conform teoremei lui Rice, R_Γ nu este calculabilă, deci nu există un algoritm pentru a determina pentru orice program P dacă funcția $\psi_p^{(1)} = f$.

Facem următoarea observație legată de acest corolar: Să considerăm o funcție calculabilă simplă, de exemplu cea dată de $f(x) = x^2$. Presupunem că cerem mai multor persoane să scrie programe care să calculeze această funcție. Corolarul de mai sus ne spune că *nu există un algoritm care să verifice că aceste programe produc rezultatul dorit. Deci, cel puțin din punctul de vedere al corectării programelor, profesorul nu poate fi înlocuit de calculator*.

Corolarul 3. Nu există un algoritm pentru următoarea problemă: fiind date $u, v \in \mathbb{N}$, să se determine dacă $\Phi_u^{(1)} = \Phi_v^{(1)}$ (deci problema echivalenței a două programe este nedecidabilă).

Dacă ar exista un algoritm pentru rezolvarea problemei precedente, atunci fixând pe v și notând $\Phi_v^{(1)} = f$, ar exista un algoritm pentru problema: să se determine pentru orice u natural dacă $\Phi_u^{(1)} = f$; contradicție, conform corolarului precedent (reamintim că $\Phi_u^{(1)} = \psi_p^{(1)}$, unde $\#(P) = u$).

4.7. Tema de autoinstruire nr. 4

Consultați bibliografia pentru:

1. a cunoaște mai multe definiții echivalente ale noțiunii de algoritm;
2. a cunoaște mai multe exemple de probleme nedecidabile.

4.8. Testul de autoevaluare nr. 4

Răspunsurile la test se vor da în spațiul liber aflat în continuarea enunțurilor!

1. Explicați diferența dintre predicatul HALT și predicatul $\text{STEP}^{(m)}$.
2. Fie $n = 2^{183} \cdot 3^{54} \cdot 5^2 \cdot 7^{446} - 1$. Calculați $\Phi_n^{(2)}(7,3)$.
3. Demonstrați că pentru orice $x \in \mathbb{N}$ există $e \in \mathbb{N}$ astfel încât $\Phi_e(x) = e$.

Răspunsurile la acest test se găsesc pe pagina următoare!

4.9. Comentarii și răspunsuri la testul nr. 4 de autoevaluare

1. În predicatul $HALT$ se testează dacă un program se termină pentru un set de valori ale datelor de intrare, în timp ce în predicatul $STEP^{(m)}$ se verifică dacă acest lucru se întâmplă într-un număr finit de pași dat.
2. Programul P cu $\#(P) = n = 2^{183} \cdot 3^{54} \cdot 5^2 \cdot 7^{446} - 1$ este următorul (vezi exercițiul 2 din testul de autoevaluare din unitatea de învățare nr.3):

```
[E3] x1 ← x1 − 1  
x2 ← x2 − 1  
y ← y + 1  
if x2 ≠ 0 goto E3
```

Astfel obținem că $\Phi_n^{(2)}(7,3) = 3$.

3. Fie funcția calculabilă $g(z,x) = z$. Aplicând teorema de recursie rezultă că există $e \in \mathbb{N}$ astfel încât $\Phi_e(x) = g(e,x) = e$.

4.10. Lucrare de verificare pentru studenți

1. Demonstrați că $\#(I) = 0 \Leftrightarrow$ instrucțiunea I este $y \leftarrow y$.
2. Fie $n = 2^{100} \cdot 5^{73} \cdot 7^{511} \cdot 15^{321} - 1$. Calculați $\Phi_n^{(3)}(10,9,8)$.
3. Fie $f(z)$ o funcție calculabilă. Demonstrați că pentru orice $x \in \mathbb{N}$ există $e \in \mathbb{N}$ astfel încât $\Phi_{f(e)}(x) = \Phi_e(x)$.

Rezolvările, dar și dificultățile întâmpinate, vor fi expediate prin email tutorelui.

4.11. Bibliografie

1. C. Calude – "Complexitatea calculului – aspecte calitative", Editura Științifică și Enciclopedică, 1962.
2. M.D. Davis, R. Sigal, E.J. Weyuker – "Computability, Complexity and Languages", Academic Press (Morgan Kaufmann), 1994.
3. N.D. Jones – "Computability and Complexity", MIT Press, 1997.
4. Christos H. Papadimitriou – "Computational Complexity", Addison-Wesley, 1994.
5. C.P. Popovici, S. Rudeanu, H. Georgescu – "Bazele informaticii", Vol.II, Tipografia Universității București, 1991.

UNITATEA DE ÎNVĂȚARE NR.5

LIMBAJUL S_n . LIMBAJUL POST-TURING. MAȘINI TURING

Cuprins

5.1.	Obiectivele unității de învățare nr. 5	40
5.2.	Indicații metodice pentru unitatea de învățare nr. 5	40
5.3.	Limbajul S_n	40
5.4.	Limbajul Post-Turing	43
5.5.	Mașini Turing	48
5.6.	Tema de autoinstruire nr. 5	52
5.7.	Testul de autoevaluare nr. 5	52
5.8.	Comentarii și răspunsuri la testul nr. 5 de autoevaluare	54
5.9.	Lucrare de verificare pentru studenți	54
5.10.	Bibliografie	55

5.1. Obiectivele unității de învățare nr. 5

După ce veți parcurge această unitate de învățare, veți reuși să:

- cunoașteți limbajele S_n și Post-Turing;
- cunoașteți noțiunea de mașină Turing.

5.2. Indicații metodice pentru unitatea de învățare nr. 5

Materialul trebuie parcurs în ordinea sa firească, prezentată în continuare. Se recomandă conspectarea și notarea ideilor principale, precum și consultarea bibliografiei pentru detalii și informații suplimentare.

Timpul minim pe care trebuie să-l acordați acestei unități de învățare este de **8 ore**.

5.3. Limbajul S_n

Pentru orice număr natural $n > 0$, introducem limbajul S_n , care lucrează cu variabile ce sunt cuvinte într-un alfabet A cu $|A| = n$. Instrucțiunile acestui limbaj sunt următoarele:

$v \leftarrow sv$	se adaugă la stânga cuvântului v litera s
$v \leftarrow v^-$	se șterge ultima literă a lui v ; $\lambda^- = \lambda$
if v ends s goto L	dacă v se termină cu litera s , se face transfer la prima instrucțiune din program cu eticheta L ; în caz contrar, se trece la executarea instrucțiunii următoare

Evident în instrucțiunile de mai sus trebuie să avem $v \in A^*, s \in A$. La fel ca și în cazul limbajului S , variabilele diferite de variabilele de intrare se consideră inițializate cu λ (cuvântul vid, corespunzător lui 0), se folosesc macroinstrucțiuni etc. Diferența față de s constă numai în aceea că variabilele sunt gândite ca având valori în A^* , iar forma instrucțiunilor diferă, fiind acum adecvată lucrului cu cuvinte. Instrucțiunile din s_n sunt naturale pentru cuvinte și nenaturale pentru numere, aşa cum instrucțiunile din s sunt naturale pentru numere, dar nu și pentru cuvinte.

Definiția 1. Spunem că o funcție $f : (A^*)^m \rightarrow A^*$ este (*partial*) *calculabilă* în s_n dacă este calculată de un program în limbajul s_n .

Observație. Vom întâlni în continuare adeseori secvența:

```
if v ends s1 goto B1
. . .
if v ends sn goto Bn
```

pe care o vom scrie prescurtat: **if** v **ends** s_i **goto** B_i ($1 \leq i \leq n$) și o vom numi **filtru**.

Prezentăm câteva macroinstrucțiuni în limbajul s_n , împreună cu dezvoltările corespunzătoare (se consideră $A = \{s_1, \dots, s_n\}$):

if $v \neq \lambda$ goto L	if v ends s_i goto L ($1 \leq i \leq n$)
$v \leftarrow \lambda$	[A] $v \leftarrow v^-$ if $v \neq \lambda$ goto A
goto L	$z \leftarrow s_1 z$ if $z \neq \lambda$ goto L
$v' \leftarrow v$	$z \leftarrow \lambda$ $v' \leftarrow \lambda$ [A] if v ends s_i goto B_i ($1 \leq i \leq n$) goto C
	[B _i] $\begin{cases} v \leftarrow v^- \\ v' \leftarrow s_i v' \\ z \leftarrow s_i z \\ \text{goto } A \end{cases} \quad i = 1, \dots, n$
	[C] if v ends s_i goto D_i ($1 \leq i \leq n$) goto E
	[D _i] $\begin{cases} z \leftarrow z^- \\ v \leftarrow s_i v \\ \text{goto } C \end{cases} \quad i = 1, \dots, n$

Teoremă. Fie $f : \mathbb{N}^m \rightarrow \mathbb{N}$ (parțial) calculabilă $\Rightarrow \forall n$ și $A = \{s_1, \dots, s_n\}$, funcția $f_n : (A^*)^m \rightarrow A^*$ este (parțial) calculabilă în s_n .

Fie P programul din S care calculează funcția f . Arătăm cum se construiește un program Q în S_n care simulează pas cu pas programul P , în sensul că orice modificare a valorii variabilei v din α în β în S corespunde modificării valorii variabilei v din $\varphi_n^{-1}(\alpha)$ în $\varphi_n^{-1}(\beta)$ în S_n .

Este suficient să înlocuim fiecare instrucțiune din s cu o macroinstrucțiune în s_n care să o simuleze:

- instrucțiunea **if** $v \neq 0$ **goto** L este simulață de **if** $v \neq \lambda$ **goto** L ;
 - instrucțiunile $v \leftarrow v + 1$ și $v \leftarrow v - 1$ vor fi simulate respectiv de macroinstrucțiunile $v \leftarrow v \oplus 1$ și $v \leftarrow v ! 1$ din S_n , unde (presupunând că în A avem $s_1 < s_2 < \dots < s_n$), $v \oplus 1$ va fi "succesorul" lui v , iar $v ! 1$ va fi "predecesorul" lui v , definite analog ca pentru numere.

Exemplu. Dacă $S = \{a, b, c\}$ cu $a < b < c$, atunci: $baac \oplus 1 = bbaa, bcaa ! \cdot 1 = bbcc, \lambda ! \cdot 1 = \lambda$.

Rămâne de arătat că funcțiile $f_1, f_2 : A^* \rightarrow A^*$ definite prin $f_1(x) = x \oplus 1$ și $f_2(x) = x ! 1$ sunt calculabile în S_n .

Propoziția 1. f_1 este calculabilă în s_n .

Este suficient să producem un program care calculează această funcție:

$[B]$	if x ends s_i goto A_i ($1 \leq i \leq n$)
	$y \leftarrow s_1 y$
	goto E
$[A_i]$	$\begin{cases} x \leftarrow x^- \\ y \leftarrow s_{i+1} y & 1 \leq i < n \\ \textbf{goto } C \end{cases}$
$[A_n]$	$x \leftarrow x^-$
	$y \leftarrow s_1 y$
	goto B
$[C]$	if x ends s_i goto D_i ($1 \leq i \leq n$)
	goto E
$[D_i]$	$\begin{cases} x \leftarrow x^- \\ y \leftarrow s_i y & 1 \leq i \leq n \\ \textbf{goto } C \end{cases}$

5.4. Limbajul Post-Turing

Fie $A = \{s_1, \dots, s_n\}$ un alfabet format din n litere. Pe baza lui, vom introduce un nou limbaj: *limbajul Post-Turing* T_n .

Se folosește o bandă infinită, atât la dreapta cât și la stânga, formată din celule ce pot conține fie o literă din A , fie un blanc (notat prin b sau s_0). Mai există un cap de citire/scriere ce poate explora la fiecare moment de timp o singură celulă. Întotdeauna banda va conține numai un număr finit de celule în care apar litere din A .

Instrucțiunile limbajului Post-Turing T_n sunt următoarele:

print s	în celula din dreptul capului de citire/scriere se scrie litera s
if s goto L	dacă celula din dreptul capului de citire/scriere conține litera s , atunci se face transfer la prima instrucțiune din program cu eticheta L ; în caz contrar, se trece la instrucțiunea următoare
right	capul de citire/scriere este deplasat cu o poziție la dreapta
left	capul de citire/scriere este deplasat cu o poziție la stânga

Prin *configurația* benzii de intrare la un moment de timp oarecare înțelegem conținutul ei, împreună cu poziția capului de citire/scriere; dacă va apărea numai o porțiune din bandă, se va considera că în celulele din dreapta și stânga sa apar numai blancuri. Pentru a indica poziția capului de citire/scriere, vom figura îngroșat (**bold**) simbolul din celula în dreptul căruia se află.

Configurația initială pentru un program în limbajul T_n ce prelucrează datele de intrare $x_1, \dots, x_m \in A^*$ este următoarea:

	b	x_1	b	x_2	\dots	b	x_m	b	
--	----------	-------	-----	-------	---------	-----	-------	-----	--

unde, aşa cum am convenit, poziția capului de citire/scriere este în stânga celulei ce conține x_1 .

Dacă de exemplu $x_2 = \lambda$, atunci după x_1 vor apărea două blancuri.

Să observăm că nu putem face distincție între această configurație și cea cu $m + 1$ intrări cu $x_{m+1} = \lambda$.

Exemplul 1. Fie $A = \{s_1, s_2, s_3\}$ și $x \in A^*$. Următoarea schimbare de configurație:

	b	x	b		\mapsto		b	x	s_1	s_1	b	
--	----------	-----	-----	--	-----------	--	----------	-----	-------	-------	-----	--

este realizată de exemplu de programul următor, scris în limbajul T_3 :

```
[A] right
if  $s_i$  goto A (1 ≤ i ≤ 3)
```

```

print  $s_1$ 
right
print  $s_1$ 
[A] left
if  $s_i$  goto  $B$  (1 ≤  $i$  ≤ 3)

```

Exemplul 2. Fie $A = \{s_1\}$ și $x \in A^*$. Următorul program realizează schimbarea de configurație:



Vom folosi în afară de s_1 și $b = s_0$ un simbol auxiliar $m \notin A$ (numit *marcăj*):

```

[A] right
if  $b$  goto  $E$ 
print  $m$ 
[B] right
if  $s_1$  goto  $B$ 
[C] right
if  $s_1$  goto  $C$ 
print  $s_1$ 
[D] left
if  $s_1$  goto  $D$ 
if  $b$  goto  $D$ 
print  $s_1$ 
if  $s_1$  goto  $A$ 

```

program ce necesită următoarele explicații:

- 1) primele 3 instrucțiuni înlocuiesc primul s_1 cu m ;
- 2) următoarele 4 instrucțiuni determină deplasarea la dreapta peste toți s_1 , apoi peste b , apoi peste toți s_1 ;
- 3) cele 4 instrucțiuni ce încep cu cea etichetată cu D determină deplasarea la stânga peste toți s_1 și b , până la m ;
- 4) penultima instrucțiune înlocuiește pe m cu s_1 ;
- 5) ultima instrucțiune face transfer la prima, dar acum capul de citire/scriere este cu o poziție mai la dreapta.

Pentru $x = s_1 s_1$, se trece succesiv prin următoarele configurații:

$$\begin{array}{ccccccccc}
 \mathbf{b} s_1 s_1 b & \xrightarrow{1} & b \mathbf{m} s_1 b & \xrightarrow{2} & b m s_1 b \mathbf{b} & \xrightarrow{3} & b m s_1 b s_1 b & \xrightarrow{4} \\
 & & & & & & & \\
 b \mathbf{m} s_1 b s_1 b & \xrightarrow{5} & b s_1 s_1 b s_1 b & \xrightarrow{1} & b s_1 \mathbf{m} b s_1 b & \xrightarrow{2} & b s_1 m b s_1 \mathbf{b} & \xrightarrow{3}
 \end{array}$$

$$bs_1mbs_1s_1b \xrightarrow{4} bs_1\mathbf{m}bs_1s_1b \xrightarrow{5} bs_1s_1bs_1s_1b \xrightarrow{1} bs_1s_1\mathbf{b}$$

Exemplul de mai sus arată necesitatea (sau cel puțin utilitatea) folosirii unor simboluri auxiliare (marcaje), pe lângă simbolurile din $A \cup \{s_0\}$.

Definiție. Fie funcția parțială $f : (A^*)^m \rightarrow A^*$. Spunem că *un program P în limbajul T_n calculează funcția f* dacă:

- 1) plecându-se din configurația inițială cu valorile de intrare x_1, \dots, x_m , programul se oprește
 $\Leftrightarrow f(x_1, \dots, x_m) \downarrow$;
- 2) la oprire, prin stergerea de pe bandă a simbolurilor ce nu sunt în A (adică $s_0 = b$ sau marcaje), rămâne $f(x_1, \dots, x_m)$.

Un program P în limbajul T_n calculează strict pe f dacă, în plus:

- 3) singurele simboluri prezente în instrucțiuni și pe bandă sunt s_0, s_1, \dots, s_n ;
- 4) când P se oprește, configurația benzii este:

.....	b	$f(x_1, \dots, x_m)$	b
-------	----------	----------------------	-----	-------

Prezentăm în continuare câteva macroinstrucțiuni ale limbajului T_n , împreună cu dezvoltările lor:

- 1) **goto L :**
if s_i **goto L** ($0 \leq i \leq n$)
- 2) **right to next blank :**

```
[A]      right
        if b goto E
        goto A
```

- 3) **left to next blank :**

```
[A]      left
        if b goto E
        goto A
```

- 4) **move block right**, care realizează schimbarea de configurație:



unde blocul nu conține b :

```

[C]   left
      if  $s_i$  goto  $A_i$  ( $0 \leq i \leq n$ )
      {
        right
        print  $s_i$             $1 \leq i \leq n$ 
        left
      } goto C

[ $A_i$ ] right
print b
left

```

5) **erase a block :**

```

[A]   right
      if  $b$  goto E
      print  $b$ 
      goto A

```

În continuare vom demonstra că pentru orice funcție parțială $f : \mathbb{N}^m \rightarrow \mathbb{N}$ și orice n , sunt valabile implicațiile din figura 1, ceea ce arată că cele 3 moduri diferite în care s-a introdus noțiunea de calculabilitate sunt echivalente.

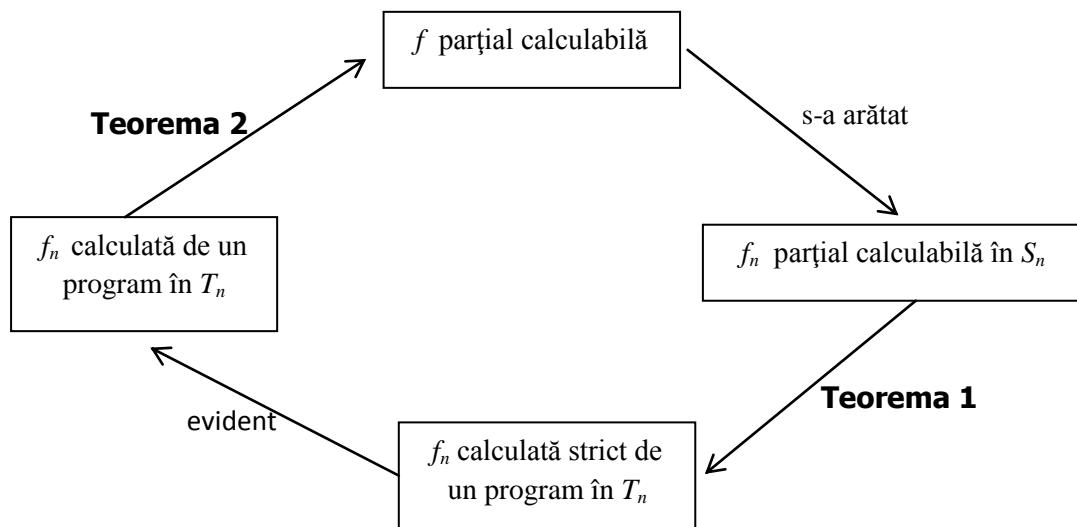


Figura 1. Echivalența între definițiile noțiunii de calculabilitate

Teorema 1. f_n este parțial calculabilă în $S_n \Rightarrow f_n$ este calculată strict de un program în T_n .

Fie P un program în S_n ce calculează funcția f_n . Fie z_1, \dots, z_k variabilele de lucru utilizate. În P apar atunci variabilele $x_1, \dots, x_m, z_1, \dots, z_k, y$ pe care le notăm în această ordine cu v_1, \dots, v_l ; deci $l = m + k + 1$.

La începutul fiecărui pas, banda va avea următoarea configurație:

b $x_1 b \dots b x_m b z_1 b \dots b z_k b y b$

în care apar valorile curente ale variabilelor de mai sus. Observăm că acest lucru este convenabil, întrucât configurația inițială:

b $x_1 b x_2 b \dots b x_m b$

corespunde inițializării cu λ a variabilelor z_1, \dots, z_k, y .

Convenim că dacă o macroinstructiune este urmată de un număr natural scris între paranteze, atunci se consideră că macroinstructiunea apare de acel număr de ori.

Instructiunile limbajului s_n sunt simulate în T_n astfel:

$v_j \leftarrow s_i v_j :$ <hr/> $v_j \leftarrow v_j^- :$ <hr/> $\text{if } v_j \text{ ends } s_i \text{ goto } L$	right to next blank (l) move block right ($l - j + 1$) right print s_i left to next blank (j) right to next blank (j) left if b goto A move block right (j) right goto E $[A]$ left to next blank ($j - 1$) right to next blank (j) left if s_i goto A right left to next blank (j) goto E $[A]$ left to next blank (j) goto L
--	---

Programul Q se obține din P înlocuind fiecare instructiune a lui P cu secvența corespunzătoare în limbajul T_n , iar la sfârșit adăugând macroinstructiunea:

erase a block ($l - 1$)

pentru a obține configurația finală **b** $y b$. Deci Q calculează strict pe f_n .

Teorema 2. Dacă f_n este o funcție parțială calculată de un program în T_n , atunci f este parțial calculabilă.

Demonstrația acestei teoreme o propunem ca temă de studiu individual.

5.5. Maşını Turing

La fel ca în cazul programelor Post-Turing, vom folosi o bandă infinită atât la dreapta cât și la stânga, formată din celule ce conțin simboluri dintr-un alfabet T precum și simbolul b . La fiecare moment de timp pe bandă se vor afla doar un număr finit de simboluri din T . Alfabetul T conține o submulțime A . Ne va interesa în continuare cum din cuvintele $x_1, \dots, x_m \in A^*$ se poate obține un cuvânt $y \in A^*$; simbolurile din $T \setminus A$ se numesc tot *marcaje*. Capul de citire/scriere poate explora de asemenea o unică celulă. Deosebirea va consta în faptul că trecerea de la o configurație la alta nu se va mai realiza pe baza unor instrucțiuni, ci pe baza unor aşa numite *stări* ale mașinii Turing.

Definiția 1. O mașină Turing este un quadruplu $M = (K, T, q_1, Q)$ unde:

- K este alfabetul stărilor;
 - T este alfabetul mașinii Turing;
 - $q_1 \in K$ este *starea inițială*;
 - \mathcal{Q} este o mulțime finită de quadruple pe baza cărora se trece de la o configurație la alta.

O configurație are forma: $\frac{\alpha s \beta}{q}$

unde $\alpha, \beta \in (T \cup \{b\})^*$, $s \in T \cup \{b\}$ este simbolul din dreptul capului de citire/scriere, iar q este starea curentă.

Prezentăm în continuare forma quadrupelor din mulțimea \mathcal{Q} , împreună cu schimbările de configurație corespunzătoare:

- 1) (q_i, s_j, s_k, q_l) $\alpha s_j \beta \mapsto q_i$ $\alpha s_k \beta \mapsto q_l$
- 2) (q_i, s_j, R, q_l) $\alpha s_j s_k \beta \mapsto q_i$ $\alpha s_j s_k \beta \mapsto q_l$
- 3) (q_i, s_j, L, q_l) $\alpha s_k s_j \beta \mapsto q_i$ $\alpha s_k s_j \beta \mapsto q_l$

unde q_i este starea din care se pleacă, s_j este simbolul explorat de capul de citire/scriere, iar q_l este starea în care se ajunge; cele 3 forme de quadruple determină respectiv înlocuirea simbolului s_j explorat prin s_k , deplasarea la dreapta și deplasarea la stânga a capului de citire/scriere. Vom nota în continuare $s_0 = b$.

Configurația inițială a benzii este următoarea:

$b \ x_1 b x_2 b \dots b x_m$

q_1

unde $x_1, x_2, \dots, x_m \in A^*$.

Schimbările de configurație se fac conform quadruprelor din κ . Mașina Turing se oprește dacă se ajunge într-o configurație de forma

$\alpha \ s_j \beta$

q_i

și nici un quadruplu din \mathcal{Q} nu începe cu (q_i, s_j) .

Definiția 2. Spunem că M calculează funcția f parțială de aritate m pe A^* dacă plecând de la configurația inițială, M se oprește pentru $(x_1, \dots, x_m) \in (A^*)^m$ dacă și numai dacă $f(x_1, \dots, x_m)$ este definită și, în plus, la oprire pe bandă se obține, stergându-se toate simbolurile ce nu sunt în A , tocmai $f(x_1, \dots, x_m)$.

Definiția 3. Spunem că M calculează strict pe f dacă în plus pe bandă apar tot timpul simboluri din $A \cup \{s_0\}$, iar la oprire se ajunge în configurația:

$s_0 y$

q_i

cu $y = f(x_1, \dots, x_m)$ și $q_i \in K$ arbitrar.

Exemplu. Fie $M = ((q_1, q_2, q_3), \{1\}, q_1, Q)$ unde:

$$\mathcal{Q} = \{(q_1, s_0, R, q_2), (q_2, 1, R, q_2), (q_2, s_0, 1, q_3), (q_3, 1, R, q_3), (q_3, s_0, 1, q_1)\}$$

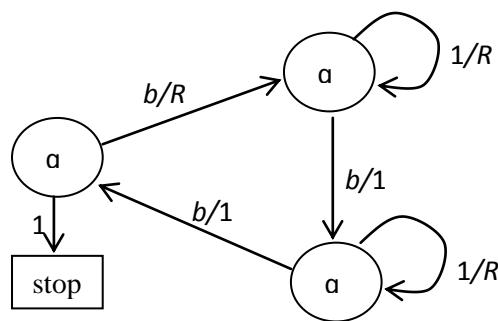
Cum $s_0 = b$, avem de exemplu:

$$\begin{array}{ccccccccccccc} b & 1 & 1 & 1 & \mapsto & 1 & 1 & 1 & 1 \\ q_1 & & q_2 & & & q_2 & & q_2 & & q_2 & & q_3 & & q_3 & & q_3 & & q_1 & & q_1 \end{array}$$

Observație. Această mașină Turing calculează funcția dată de $f(x) = x + 2$.

Alte reprezentări pentru o mașină Turing sunt cele sub formă de tabel și sub formă de graf, pe care ne mărginim a le prezenta pentru exemplul considerat.

	q_1	q_2	q_3
b	R, q_2	$1, q_3$	$1, q_1$
1	R, q_2	R, q_3	



Teorema 1. Orice funcție parțială calculată de un program Post-Turing este calculată și de o mașină Turing cu același alfabet.

Fie $f : (A^*)^m \rightarrow A^*$ calculată de un program P în T_n unde $A = \{s_1, \dots, s_n\}$ (putem presupune că f este calculată strict de P).

Fie I_1, \dots, I_p instrucțiunile lui P .

Este suficient să construim o mașină Turing M ce simulează pe P .

Fie $M = (\{q_1, \dots, q_{p+1}\}, \{s_1, \dots, s_n\}, q_1, Q)$, unde Q va fi construită astfel încât vom ajunge în starea i exact atunci când trebuie executată instrucțiunea I_i . În acest scop, pentru fiecare instrucțiune I_i vom introduce unul sau mai multe quadruple, după cum urmează:

- dacă I_i este **print** s_k , introducem $(q_i, s_j, s_k, q_{i+1}), \forall j = 0, 1, \dots, n$
- dacă I_i este **right** introducem $(q_i, s_j, R, q_{i+1}), \forall j = 0, 1, \dots, n$
- dacă I_i este **left**, introducem $(q_i, s_j, L, q_{i+1}), \forall j = 0, 1, \dots, n$
- dacă I_i este **if** s_k **goto** L introducem

$$\begin{cases} (q_i, s_k, s_k, q_r) \\ (q_i, s_j, s_j, q_{i+1}), \forall j \in \{0, 1, \dots, n\}, j \neq k \end{cases}$$

unde r este fie numărul primei instrucțiuni din P etichetate cu L (dacă o astfel de instrucțiune există), fie $p+1$ (în caz contrar).

O variantă de mașină Turing o constituie *mașina Turing quintuplă*, în care Q este formată din quintuple având una dintre următoarele două forme (se precizează și schimbările de configurație respective):

$$1) \quad (q_i, s_j, s_k, R, q_l) \quad \alpha s_j s_m \beta \mapsto \alpha s_k s_m \beta$$

$$q_i \qquad \qquad \qquad q_l$$

$$2) \quad (q_i, s_j, s_k, L, q_l) \quad \alpha s_m s_j \beta \mapsto \alpha s_m s_k \beta$$

$$q_i \qquad \qquad \qquad q_l$$

Și mașina Turing quintuplă va fi considerată *deterministă*, deci nu există două quintuple ce încep cu aceeași pereche (q_i, s_j) .

Teorema 2. Dacă funcția parțială f este calculată de o mașină Turing, atunci ea este calculată și de o mașină Turing quintuplă cu același alfabet.

Fie $M = (\{q_1, \dots, q_r\}, \{s_1, \dots, s_n\}, q_1, Q)$ mașina Turing ce calculează pe f . Construim atunci mașina Turing quintuplă $\bar{M} = (\{q_1, \dots, q_r, \dots, q_{2r}\}, \{s_1, \dots, s_n\}, q_1, \bar{Q})$, unde \bar{Q} este construită astfel:

- pentru fiecare (q_i, s_j, R, q_l) introducem (q_i, s_j, s_j, R, q_l)
- pentru fiecare (q_i, s_j, L, q_l) introducem (q_i, s_j, s_j, L, q_l)
- pentru fiecare (q_i, s_j, s_k, q_l) introducem (q_i, s_j, s_k, R, q_l)

- în plus introducem quintuplele: $\{q_{l+r}, s_j, s_j, L, q_l\} | l = 1, \dots, r, j = 0, \dots, n\}$.

Teorema 3. Dacă funcția parțială f este calculată de o mașină Turing quintuplă, ea este calculată și de un program Post-Turing.

Fie $M = (\{q_1, \dots, q_r\}, \{s_1, \dots, s_n\}, q_1, Q)$ o mașină Turing quintuplă ce calculează funcția f .

Pentru fiecare $i = 1, \dots, r, j = 0, \dots, n$ asociem :

$$q_i \mapsto \text{eticheta } A_i ; (q_i, s_j) \mapsto \text{eticheta } B_{ij}.$$

În programul Post-Turing pe care îl construim, eticheta A_i apare în fața filtrului:

$[A_i] \quad \mathbf{if} \ s_j \ \mathbf{goto} \ B_{ij} \quad (0 \leq j \leq n).$

Pentru fiecare (q_i, s_j, s_k, R, q_l) introducem blocul:

```
[Bij] print sk
        right
        goto Al
```

iar pentru fiecare (q_i, s_j, s_k, L, q_l) introducem blocul:

```
[Bij] print sk
        left
        goto Al
```

Pentru toate perechile (q_i, s_j) care nu sunt la începutul vreunui quadruplu din Q , introducem instrucțiunea:

$[B_{ij}] \quad \mathbf{goto} \ E.$

Programul Post-Turing căutat va fi cel obținut prin concatenarea tuturor blocurilor introduse mai sus, cu singura restricție că programul începe cu filtrul etichetat cu A_1 .

Corolar. O funcție parțială f este calculată de un program Post-Turing dacă și numai dacă f este calculată de o mașină Turing.

Mai mult, am obținut că pentru orice funcție parțială $f : N^m \rightarrow N$, f este parțial calculabilă $\Leftrightarrow f$ este calculată de o mașină Turing. Am obținut astfel o nouă caracterizare a funcțiilor parțial calculabile și anume cu ajutorul mașinilor Turing. Am obținut de asemenea un argument în plus în sprijinul tezei lui Church.

Fie $M = (K, A, q_1, Q)$. Un cuvânt $u \in A^*$ este *acceptat de* M dacă plecându-se din configurația inițială

b_u

q₁

mașina se oprește. În mod corespunzător vom defini *limbajul acceptat de mașina Turing M* ca fiind:

$$L(M) = \{u \in A^* \mid u \text{ acceptat de } M\}.$$

Fie $A = \{s_1, \dots, s_n\}$ un alfabet cu n litere. Reamintim că fiecare $u \in A^*$ poate fi privit ca un număr natural, notat $\varphi_n(u)$. Atunci pentru orice $B \subset A^*$ introducem în mod natural următoarea definiție: B este calculabilă dacă există un algoritm care decide pentru orice $u \in A^*$ dacă $u \in B$ sau $u \notin B$ (altfel spus, dacă $\varphi_n(B) \subset \mathbb{N}$ este sau nu calculabil).

Teorema 4. Există M mașină Turing peste A cu $L(M)$ necalculabilă.

Știm că mulțimea $K = \{x \in N \mid \varphi_x^{(1)}(x) \downarrow\}$ nu este calculabilă. Funcția f definită prin $f(x) = \varphi_x^{(1)}(x) = \varphi^{(1)}(x, x)$ este parțial calculabilă. Atunci funcția $f_n : A^* \rightarrow A^*$ este calculată de o mașină Turing M peste alfabetul A . Deci $f_n(x) \downarrow \Leftrightarrow M$ se oprește pentru intrarea x ; ca urmare:

$$L(M) = \{x \in A^* \mid f_n(x) \downarrow\} = \varphi_n^{-1}(\{x \in N \mid f(x) \downarrow\}) = \varphi_n^{-1}(K).$$

Rezultă că limbajul $L(M)$ nu este calculabil.

5.6. Tema de autoinstruire nr. 5

Consultați bibliografia pentru:

1. a cunoaște mai multe exemple de programe scrise în limbajul S_n ;
2. a cunoaște mai multe exemple de programe scrise în limbajul Post-Turing;
3. a cunoaște mai multe exemple de mașini Turing.

5.7. Testul de autoevaluare nr. 5

Răspunsurile la test se vor da în spațiul liber aflat în continuarea enunțurilor!

1. Scrieți dezvoltarea macroinstrucției **erase a block** din limbajul T_n .

2. Demonstrați că funcția $f_2: A^* \rightarrow A^*$, $f_2(x) = x \ominus 1$ este calculabilă în S_n .
3. Descrieți o mașină Turing care să înlocuiască într-un sir binar fiecare 0 cu 1 și fiecare 1 cu 0.

Răspunsurile la acest test se găsesc pe pagina următoare!

5.8. Comentarii și răspunsuri la testul nr. 5 de autoevaluare

1. Dezvoltarea macroinstrucției erase a block în limbajul T_n este următoarea:

```
[A]      right
          if b goto E
          print b
          goto A
```

2. Este suficient să scriem un program care calculează funcția $f_2: A^* \rightarrow A^*$, $f_2(x) = x \ominus 1$:

```
[B] if x ends si goto Ai (1 ≤ i ≤ n)
    goto E
    [Ai] { x ← x̄
              y ← si-1y      1 < i ≤ n
              goto C1
[A1] x ← x̄
        if x ≠ λ goto C2
        goto E
[C2] y ← sny
        goto B
[C1] if x ends si goto Di (1 ≤ i ≤ n)
        [Di] { x ← x̄
                  y ← siy      1 ≤ i ≤ n
                  goto C1
```

3. O mașină Turing quintuplă care înlocuiește într-un sir binar fiecare 0 cu 1 și fiecare 1 cu 0 este următoarea: $M = (\{q_1\}, \{0,1\}, q_1, Q)$ unde $Q = \{(q_2, 0, 1, R, q_2), (q_2, 1, 0, R, q_2)\}$.

5.9. Lucrare de verificare pentru studenți

1. Scrieți un program în limbajul S_n și un program în limbajul Post-Turing care să înlocuiască într-un sir binar fiecare 0 cu 1 și fiecare 1 cu 0.
2. Descrieți o mașină Turing care să realizeze incrementarea cu o unitatea a unui număr binar dat.
3. Descrieți o mașină Turing care să decidă dacă un sir binar este palindrom sau nu.

Rezolvările, dar și dificultățile întâmpinate, vor fi expediate prin email tutorelui.

5.10. Bibliografie

1. C. Calude – "*Complexitatea calculului – aspecte calitative*", Editura Științifică și Enciclopedică, 1962.
2. M.D. Davis, R. Sigal, E.J. Weyuker – "*Computability, Complexity and Languages*", Academic Press (Morgan Kaufmann), 1994.
3. N.D. Jones – "*Computability and Complexity*", MIT Press, 1997.
4. Christos H. Papadimitriou – "*Computational Complexity*", Addison-Wesley, 1994.
5. C.P. Popovici, S. Rudeanu, H. Georgescu – "*Bazele informaticii*", Vol.II, Tipografia Universității București, 1991.

UNITATEA DE ÎNVĂȚARE NR. 6

CLASE DE COMPLEXITATE

Cuprins

6.1.	Obiectivele unității de învățare nr. 6	56
6.2.	Indicații metodice pentru unitatea de învățare nr. 6	56
6.3.	NP-completitudine	56
6.4.	Problema clicii maximale.....	59
6.5.	Tema de autoinstruire nr. 6	61
6.6.	Testul de autoevaluare nr. 6	62
6.7.	Comentarii și răspunsuri la testul nr. 6 de autoevaluare.....	63
6.8.	Lucrare de verificare pentru studenți	63
6.9.	Bibliografie	63

6.1. Obiectivele unității de învățare nr. 6

După ce veți parcurge această unitate de învățare, veți reuși să:

- cunoașteți noțiunea de algoritm nedeterminist;
- cunoașteți principalele clase de complexitate.

6.2. Indicații metodice pentru unitatea de învățare nr. 6

Materialul trebuie parcurs în ordinea sa firească, prezentată în continuare. Se recomandă conspectarea și notarea ideilor principale, precum și consultarea bibliografiei pentru detalii și informații suplimentare.

Timpul minim pe care trebuie să-l acordați acestei unități de învățare este de **8 ore**.

6.3. NP-completitudine

Știm că doar algoritmii polinomiali sunt eficienți, deci urmărим totdeauna să elaborăm astfel de algoritmi. Este evident însă că nu există algoritmi polinomiali pentru orice problemă, ca de exemplu în cazul în care numărul datelor de ieșire nu este polinomial: determinarea tuturor submulțimilor unei mulțimi finite, generarea permutărilor etc. De aceea căutăm să delimităm clasa problemelor pentru care încercăm să elaborăm algoritmi polinomiali. În acest scop introducem noțiunea de algoritm nedeterminist.

Algoritmii nedeterminiști sunt algoritmi secvențiali care, spre deosebire de cei obișnuiți, admit și următoarele instrucțiuni suplimentare:

- 1) `success` și `failure`, care arată că algoritmul se termină cu succes, respectiv cu eșec. Ele înlocuiesc instrucțiunea `stop`. Forma lor arată că vom studia doar probleme de decizie, pentru care rezultatul poate fi doar afirmativ sau negativ (după cum vom vedea, foarte multe probleme pot fi aduse la aceasta formă).
- 2) `choice (A)`, unde A este o mulțime finită; este o funcție care întoarce ca rezultat un element oarecare al lui A .

Se consideră că cele trei instrucțiuni nou introduse necesită un *temp constant*.

Mașina abstractă care execută un astfel de algoritm, când întâlnește o instrucțiune `choice` lucrează astfel:

- dacă există o valoare din A care conduce la o instrucțiune `success`, va fi aleasă o altfel de valoare;
- în caz contrar, va fi aleasă o valoare oarecare.

Cu alte cuvinte, un algoritm nedeterminist se termină cu eșec dacă nu există o modalitate de a efectua alegeri care să conducă la o instrucțiune `success`.

Funcționarea mașinii abstracte se asemănă calculului paralel. De câte ori este întâlnită o instrucțiune `choice`, intră în acțiune atâtea procesoare câte elemente are mulțimea A . Algoritmul se termină cu succes dacă unul dintre procesoarele active ajunge la o instrucțiune `success`. Timpul de executare va fi, în cazul unei terminări cu succes, timpul necesar ajungerii la respectiva instrucțiune `success`.

Mai precizăm că ne interesează doar terminările cu succes.

Exemplul 1. Se cere să se verifice dacă un număr x aparține sau nu unei mulțimi $A = \{a_1, \dots, a_n\}$.

Știm că timpul de executare pentru algoritmul determinist pentru această problemă este de ordinul $O(n)$, adică liniar.

Putem scrie următorul algoritm nedeterminist:

```
i ← choice({1, 2, ..., n})
if a_i=x then write i; success
else failure
```

care rezolvă problema în *temp constant*.

Exemplul 2. Se cere să se ordoneze crescător vectorul $a = (a_1, \dots, a_n)$.

Știm că cel mai bun algoritm determinist pentru această problemă necesită un timp de ordinul $O(n \log_2 n)$.

Ideea algoritmului nedeterminist este următoarea: copiem elementele vectorului a într-un vector auxiliar b într-o ordine oarecare, după care verificăm dacă elementele lui b apar în ordine crescătoare:

```

for i=1,n
    bi ← ∞
for i=1,n
    j ← choice({1,2,...,n})
    if bj=∞ then bj ← ai
        else failure
for i=1,n-1
    if bi>bi+1 then failure
write(b);
success

```

Timpul de executare al acestui algoritm este $O(n)$, deci liniar. Se observă că:

- am obținut un timp de executare mai bun;
- problema sortării a fost tratată ca o problemă de decizie.

Exemplul 3. Problema validării

Fie $F(x_1, \dots, x_n)$ o expresie booleană în forma normală conjunctivă (FNC): $F = C_1 \wedge \dots \wedge C_k$, unde C_1, \dots, C_k sunt disjuncții de variabile de forma x_j sau $\overline{x_j}$ ($\overline{x_j}$ este negația lui x_j). Se cere să se determine dacă există $a_1, \dots, a_n \in \{0,1\}$ cu $F(a_1, \dots, a_n) = 1$.

Problema va fi referită în continuare sub numele *VALID*.

Un exemplu de instanță a problemei este următorul: $F(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$.

Un algoritm nedeterminist simplu care rezolvă problema este următorul:

```

for i=1,n
    xi ← choice({0,1})
    if F(x1, ..., xn)=1 then success
        else failure

```

Timpul este proporțional cu lungimea formulei, deci *liniar*.

Observație. Nu se cunoaște un algoritm determinist polinomial pentru problema validării !

Introducem *clasele de probleme (de decizie)* următoare:

- **P** - clasa problemelor pentru care există algoritmi determiniști polinomiali;
- **NP** - clasa problemelor pentru care există algoritmi nedeterminiști polinomiali.

Vom studia problema existenței algoritmilor (determiniști) polinomiali doar pentru problemele din **NP**.

Evident **P** ⊂ **NP**. Este însă inclusiunea strictă sau **P** = **NP** ?
 Precizăm de la început că această problemă este deschisă!

În 1976, Samuel Cook a obținut un rezultat care a simplificat problema și care a părut promițător în rezolvarea ei:

Teoremă. $P = NP \Leftrightarrow VALID \in P$.

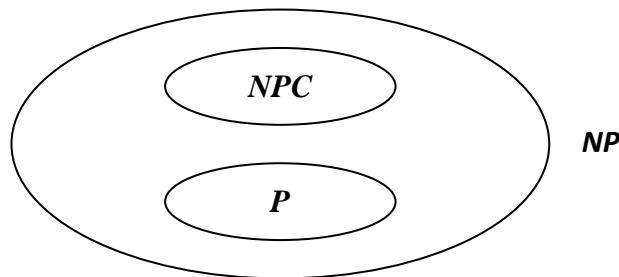
Teorema spune că dacă reușim să găsim un algoritm determinist polinomial pentru $VALID$, atunci va exista un algoritm determinist polinomial pentru *orice* problemă din NP .

Întrucât nu s-a reușit să se obțină un algoritm polinomial pentru $VALID$, s-a încercat să se rezolve probleme "echivalente" cu $VALID$. Mai precis s-a definit clasa NPC .

Clasa de probleme NPC este definită astfel:

- 1) $VALID \in NPC$
- 2) Pentru o problemă P , $P \in NPC$ dacă:
 - 2.1) se cunoaște pentru P un algoritm nedeterminist polinomial;
 - 2.2) $VALID$ se poate reduce la P în timp determinant polinomial.

Problemele din NPC se numesc *NP-complete*.



Observație. Este suficient să arătăm pentru o singură problemă NP -completă că admite un algoritm polinomial, pentru a obține egalitatea $P = NP$. Lista problemelor NP -complete a depășit 1000, dar pentru nici una dintre ele nu s-a reușit să se obțină un algoritm determinant polinomial. Drept urmare, aşa cum am menționat anterior, problema egalității $P = NP$ rămâne o problema deschisă.

Prezentăm în continuare una dintre multele probleme NP -complete, respectiv *problema clicii maximale*.

6.4. Problema clicii maximale

Fie $G = (X, M)$ un graf neorientat. Multimea $Y \subset X$ se numește *clică* dacă pentru $\forall i, j \in Y$ avem $(i, j) \in M$. Se cere să se determine ordinul unei clici maximale.

Problema de decizie corespunzătoare este următoarea: *Pentru un k dat, există în G o clică de ordin k ?*

Pentru această problemă vom scrie procedura `clica(G, k)`, care furnizează răspunsul în timp nedeterminist polinomial.

Presupunând cunoscut acest lucru, algoritmul pentru problema clicii maximale va fi:

```
for k=n,1,-1
    clica(G,k)
```

care este tot polinomial.

```
procedure clica(G,k)
    for i=1,n
        ales(i) ← 0
    for i=1,k
        j ← choice({1,...,n})
        if ales(j)=1 then failure
            else ales(j) ← 1; bi ← j
    for i=1,k
        for j=1,k
            if i≠j & (bi,bj) ∉ M
                then failure
    write(k); success
end;
```

Timpul este evident polinomial. Prin urmare $CLICA(k) \in NP$, deci și $CLICA \in NP$.

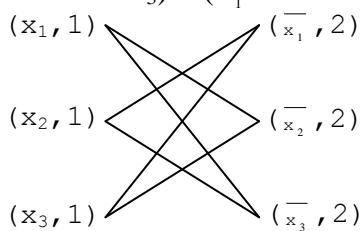
Arătăm în continuare că $VALID$ se reduce la $CLICA$ în timp determinist polinomial.

Fie $F(x_1, \dots, x_n)$ o expresie booleană în forma normală conjunctivă (FNC): $F = C_1 \wedge \dots \wedge C_k$, unde C_1, \dots, C_k sunt disjuncții de variabile de forma x_j sau \overline{x}_j , numiți *literali*.

Atașăm lui F graful $G = (X, M)$ astfel:

- $X = \{(\alpha, i) \mid \alpha$ este un literal din $C_i\}$
- $M = \{[(\alpha, i), (\beta, j)] \mid \alpha \neq \bar{\beta}\}$, adică α și β pot fi satisfăcuți concomitent.

De exemplu, pentru $F(x_1, x_2, x_3) = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee \overline{x}_3)$. Graful este:



Construcția grafului necesită un timp determinist polinomial.

Mai trebuie demonstrat că în G există o clică de ordin $k \Leftrightarrow F$ este validabilă.

Începem cu demonstrarea necesității.

Fie $S = \{(\alpha_i, i) \mid i = 1, \dots, k\}$ o clică de ordin k și fie $S_1 = \{\alpha_i \mid i = 1, \dots, k\}$.

Conform construcției lui M , nu este posibil ca α_i și $\overline{\alpha_i}$ să apară simultan în S_1 .

Alegem fiecare a_i astfel:

$$a_i = \begin{cases} 1, & \text{dacă } x_i \in S_1, \text{ adică } \alpha_i = x_i \\ 0, & \text{dacă } \overline{x_i} \in S_1 \end{cases}$$

arbitrar în caz contrar.

Pentru această alegere, $F(a_1, \dots, a_n) = 1$, fiecare conjuncție având valoarea 1.

Pentru exemplul considerat: $S = \{(x_1, 1), (\overline{x_3}, 2)\}$; $S_1 = \{x_1, \overline{x_3}\}$; $a_1 = 1$; a_2 arbitrar; $a_3 = 0$.

Continuăm cu demonstrarea suficienței.

Conform ipotezei, există a_1, \dots, a_n cu $C_i(a_1, \dots, a_n) = 1, \forall i = 1, \dots, k$.

Pentru fiecare $i = 1, \dots, k$, există un literal α_i din C_i care are valoarea 1.

Fie $S = \{(\alpha_i, i) \mid i = 1, \dots, k\}$, rezultă că S este o clică. Într-adevăr, pentru $(\alpha_i, i), (\alpha_j, j) \in S$ diferite rezultă $i \neq j$ și $\alpha_i \neq \alpha_j$, deoarece pentru $x = (a_1, \dots, a_n)$ avem $\alpha_i = \alpha_j = 1$.

Încheiem prin a prezenta enunțul altor probleme **NP**-complete.

- 1) Fie $G = (X, M)$ un graf. $Y \subset X$ se numește *k-acoperire* dacă $|Y| = k$ și dacă pentru orice $(i, j) \in M$ avem $i \in Y$ sau $j \in Y$. Există o *k*-acoperire?
- 2) Problema ciclului hamiltonian.
- 3) Problema colorării hărților.
- 4) Fie A o mulțime și fie $s: A \rightarrow \mathbf{Z}^+$. Căutăm $B \subset A$ cu proprietatea că suma elementelor lui B să fie egală cu suma elementelor lui $A \setminus B$.

6.5. Tema de autoinstruire nr. 6

Consultați bibliografia pentru:

1. a cunoaște mai multe exemple de algoritmi nedeterminiști;
2. probleme **NP**-complete.

6.6. Testul de autoevaluare nr. 6

Răspunsurile la test se vor da în spațiul liber aflat în continuarea enunțurilor!

1. Ce sunt algoritmii nedeterminiști?
 2. Ce reprezintă clasa **P**? Dar clasa **NP**?
 3. Dați trei exemple de probleme **NP**-complete din teoria grafurilor!

Răspunsurile la acest test se găsesc pe pagina următoare!

6.7. Comentarii și răspunsuri la testul nr. 6 de autoevaluare

1. Algoritmii nedeterminiști sunt algoritmi secvențiali care, spre deosebire de cei obișnuiți, admit și instrucțiunile suplimentare *success*, *failure* și *choice*.
2. Clasa **P** este formată din problemele pentru care există algoritmi de rezolvare determiniști polinomiali, în timp ce clasa **NP** este formată din problemele pentru care există algoritmi de rezolvare nedeterminiști polinomiali.
3. Problema ciclului hamiltonian, problema clicii maximale, problema k-acoperirii.

6.8. Lucrare de verificare pentru studenți

Scrieți câte un program nedeterminist pentru cele trei probleme de mai jos:

1. Testarea apartenenței unui număr la o mulțime (vezi exemplul 1).
2. Sortarea unui vector (vezi exemplul 2).
3. Problema validării (vezi exemplul 2).

Rezolvările, dar și dificultățile întâmpinate, vor fi expediate prin email tutorelui.

6.9. Bibliografie

1. C. Calude – "Complexitatea calculului – aspecte calitative", Editura Științifică și Enciclopedică, 1962.
2. M.D. Davis, R. Sigal, E.J. Weyuker – "Computability, Complexity and Languages", Academic Press (Morgan Kaufmann), 1994.
3. H. Georgescu – "Tehnici de programare", Editura Universității din București, 2005.
4. N.D. Jones – "Computability and Complexity", MIT Press, 1997.
5. Christos H. Papadimitriou – "Computational Complexity", Addison-Wesley, 1994.
6. C.P. Popovici, S. Rudeanu, H. Georgescu – "Bazele informaticii", Vol.II, Tipografia Universității București, 1991.