

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:

- Fie $V = \{a, b\} \Rightarrow V^* = \{\lambda, a, b, ab, ba, bb, aba, bba, \dots, baba, \dots\}$. Putem defini un limbaj $L = \{a, ba, aba, baba\}$.
- V = alfabetul englez, iar L = cuvintele corecte din limba engleză;
- $V = \{\text{litere, } +, -, *, /, (,)\}$, iar L = mulțimea expresiilor aritmetice corecte formate cu aceste caractere;
- V = alfabetul ASCII, iar L = mulțimea programelor corecte din Java (un program poate fi privit ca un cuvânt din V^*).

Fie V un alfabet, iar L un limbaj peste V , reprezentând în general cuvintele „corecte” formate cu litere din V . Cum putem defini limbajul?

- Dacă L este finit, putem forma o listă cu cuvintele sale;
- Dacă L este infinit, dar cuvintele sale au o anumită formă, ca de exemplu $\{a^n b^n c^n | n > 0\}$
Dacă nu suntem în aceste situații, calea este de a preciza reguli de formare a cuvintelor sale, deci de a lucra cu **gramatici!**

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 initial al gramaticii* ($S \in N$);

- P este multimea producțiilor, adică reguli de substituție de forma $\alpha \rightarrow \beta$, unde $\alpha \in (N \cup T)^*$, $\beta \in (N \cup T)^*$.

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 \Rightarrow^* . Deci $\alpha \Rightarrow^* \beta$ 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:

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

Gramaticile ne interesează din punctul de vedere al limbajului generat!

Exemplul 1.2: Fie gramatica $G = (N, T, S, P)$, unde

$$N = \{S\}, \quad T = \{a\} \quad \text{și} \quad P = \{S \rightarrow a, \quad S \rightarrow aS\}.$$

Se poate observa foarte ușor că limbajul generat este $L(G) = \{a, aa, aaa, \dots\} = \{a^n \mid n \geq 1\}$.

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

$$S \rightarrow \lambda$$

$$S \rightarrow a$$

$$S \rightarrow b$$

$$S \rightarrow aSa$$

$$S \rightarrow bSb$$

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 palindroamele formate din literele a și b .

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

$$S \rightarrow ()$$

$$S \rightarrow (S)$$

$$S \rightarrow SS$$

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

$$S \xrightarrow[(3)]{*} SS \xrightarrow[(1)]{*} (S) \xrightarrow[(2)]{*} ((S)) \xrightarrow[(1)]{*} ((()) . \text{ Rezultă că } w \in L(G)$$

Exemplul 1.5: Fie gramatica $G = (N, T, S, P)$, unde $N = \{S, B, C\}$, $T = \{a, b, c\}$, iar multimea producțiilor P este următoarea:

- (1) $S \rightarrow aSBC$
- (2) $S \rightarrow aBC$
- (3) $CB \rightarrow BC$
- (4) $aB \rightarrow ab$
- (5) $bB \rightarrow bb$
- (6) $bC \rightarrow bc$
- (7) $cC \rightarrow cc$

Fie cuvântul $w = aabbcc = a^2b^2c^2$. Cuvântul $w \in L(G)$, deoarece se poate obține printr-o derivare din simbolul inițial S astfel:

$$S \xrightarrow[(1)]{*} aSBC \xrightarrow[(2)]{*} aaBCBC \xrightarrow[(4)]{*} aabCBC \xrightarrow[(3)]{*} aabBCC \xrightarrow[(5)]{*} aabbCC \xrightarrow[(6)]{*} aabbC \xrightarrow[(7)]{*} aabbcc$$

. Rezultă că $w \in L(G)$.

Arătăm în continuare că $L(G) = \{a^n b^n c^n \mid n > 0\}$

Plecăm bineînțeles, de la S .

Până când S ”dispare”, cuvântul curent are forma $a^n S \alpha$, unde în α există:

- n de B și b

- n de C și c
După dispariția lui S :
- toate C -urile sunt deplasate la dreapta după b -uri, conform (3)
- toate B -urile care urmează lui a sau b trec în b , conform (4) și (5)
- toate C -urile care urmează lui b sau c trec în b , conform (6) și (7)

Definiția 1.6: 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 $ cu α conținând cel puțin un neterminal
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 ($i = 0, 3$). 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 ($i = 0, 3$) 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 ($i = 0, 3$). 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 :

$A \rightarrow a$	$A \rightarrow a$	$A \rightarrow a$
$B \rightarrow b$	$B \rightarrow b$	$B \rightarrow b$
$S \rightarrow aA$	$S \rightarrow aA$	$S \rightarrow aA$
$aA \rightarrow bB$	$A \rightarrow bB$	$S \rightarrow bB$
$bB \rightarrow SB$	$B \rightarrow bB$	$A \rightarrow aB$
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 ($i = 1, 2, 3$). 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 $S' \notin N \bigcup T$. Construim gramatica $G' = \left(N \bigcup \{S'\}, T, S', P' \right)$, unde $P' = P \bigcup \{S' \rightarrow \alpha \mid S \rightarrow \alpha\}$. 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.

Metoda șirului crescător de mulțimi

Fie A o mulțime finită și fie p o proprietate definită pe mulțimea submulțimilor lui A .

Fie X o submulțime a lui A cu $p(X)=1$. Definim următorul șir crescător de mulțimi:

$$X_0 = X$$

$$X_{k+1} = X_k \cup \{x \in A \mid p(X_k \cup \{x\}) = 1\}, \forall k > 0$$

Evident, $X_0 \subset X_1 \subset \dots \subset X_k \subset X_{k+1} \subset \dots \subset A$. Cum A este finită, şirul de submulțimi se stabilizează!

Propoziție. Dacă $X_{k+1} = X_k$, atunci $X_{k+i} = X_k, \forall i \in \mathbb{N}$.

Facem demonstrația prin inducție după i .

Pentru $i=1$ rezultatul este evident.

Presupunem $X_{k+i} = X_k$ și demonstrăm că $X_{k+i+1} = X_k$:

X_{k+i+1} = cf. definiției şirului de mulțimi

$$= X_{k+i} \cup \{x \in A \mid p(X_{k+i} \cup \{x\}) = 1\} = \text{cf. ipotezei de inducție}$$

$$= X_k \cup \{x \in A \mid p(X_k \cup \{x\}) = 1\} = \text{cf. definiției şirului de mulțimi}$$

$$= X_{k+1} = \text{cf. ipotezei}$$

$$= X_k.$$

Consecință. Ne oprim cu construcția şirului crescător de mulțimi la primul k cu $X_k = X_{k+1}$.

Problema apartenenței

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

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 şirului 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, $w \in L(G) \iff w \in T_{k_0}$.

Simboluri “nefolositoare”

Gramaticile independent de context, deci și gramaticile regulate, pot fi aduse la o formă mai simplă prin eliminarea a două tipuri de simboluri: observabile și accesibile.

- **Neterminale observabile**

Fie $G = (N, T, S, P)$ o gramatică independentă de context și $X \in N$. Spunem că X este **neterminal observabil** dacă există cel puțin o derivare de forma $X \xrightarrow{*} w$, unde $w \in T^*$.

Este evident faptul că numai neterminalele observabile sunt utile în generarea cuvintelor din $L(G)$, deci producțiile care conțin în membrul stâng sau în cel drept neterminale neobservabile din P .

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

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

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 neobservabile, adică neterminalele din mulțimea $N \setminus M_k$.

Exemplu. 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:

$$\begin{aligned} S &\rightarrow AB \mid aBC \\ A &\rightarrow BA \mid a \\ B &\rightarrow b \mid AC \\ C &\rightarrow AC \mid CB \\ D &\rightarrow AD \mid a \end{aligned}$$

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

Se observă că în acest moment sirul de mulțimi s-a stabilizat deoarece $M_2 = M_1$, deci neterminalele observabile sunt cele din $M_2 = \{S, A, B, D\}$. Rezultă că singurul neterminal neobservabil este C . Putem simplifica gramatica G , eliminând pe C din N și eliminând din P producțiile în care acesta apare.

Obținem astfel o gramatică independentă de context $G' = (N', T, S, P')$ echivalentă cu G și fără neterminale neobservabile: $N' = \{S, A, B, D\}$, iar P' :

$$S \rightarrow AB$$

$$A \rightarrow BA | a$$

$$B \rightarrow b | AC$$

$$D \rightarrow AD | a$$

Teoremă. Fie $G = (N, T, S, P)$ o gramatică independentă de context. Există algoritm pentru a verifica dacă $L(G)$ este vid sau nu.

Suficient să verificăm, conform celor de mai sus, dacă simbolul inițial S este observabil: $L(G) \neq \emptyset \iff S$ este observabil.

- **Simboluri inaccesibile**

Fie $G = (N, T, S, P)$ o gramatică independentă de context și $X \in N \cup T$. Spunem că $X \in N \cup T$ este un 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.

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 .

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

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

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 $(N \cup T) \setminus M_k$.

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

$$S \rightarrow AB$$

$$A \rightarrow BA \mid a$$

$$B \rightarrow b$$

$$D \rightarrow AD \mid a$$

$$M_0 = \{S\}$$

$$M_1 = M_0 \cup \{A, B\} = \{S, A, B\}$$

$$M_2 = M_1 \cup \{a,b\} = \{S,A,B,a,b\}$$

$$M_3 = M_2$$

Se observă că în acest moment sirul de multimi s-a stabilizat deoarece $M_3 = M_2$, deci simbolurile accesibile sunt cele din $M_3 = \{S, A, B, a, b\}$. Rezultă că simbolurile inaccesibile sunt D și c . Eliminăm pe D din N și pe c din T . Eliminăm din P producțiile în care acestea apar, obținând astfel o gramatică independentă de context $G' = (N', T, S, P')$ echivalentă cu G și care nu conține neterminale inaccesibile, unde $N' = \{S, A, B\}$, $T' = \{a, b\}$ iar și mulțimea producțiilor P' este următoarea:

$$S \rightarrow AB$$

$$A \rightarrow BA | a$$

$$B \rightarrow b | AC$$

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:

$$S \rightarrow aS$$

$$S \rightarrow bA$$

$$A \rightarrow cA$$

$$A \rightarrow c$$

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

$$\begin{array}{ccccccc} * & * & * \\ S \xrightarrow{(1)} aS \xrightarrow{(2)} abA \xrightarrow{(4)} abc = w_1 \in L(G) \end{array}$$

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

$$\begin{array}{cccccccccc} * & * & * & * & * & * & * & * & * \\ S \xrightarrow{(1)} aS \xrightarrow{(1)} aaS \xrightarrow{(1)} aaaS \xrightarrow{(2)} aaabA \xrightarrow{(3)} aaabcA \xrightarrow{(3)} aaabccA \xrightarrow{(3)} aaabcccA \xrightarrow{(4)} aaabcccc \end{array}$$

Deci $w_2 \in L(G)$.

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

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:

$$S \rightarrow aA$$

$$A \rightarrow aA \mid aB$$

$$B \rightarrow bC$$

$$C \rightarrow cB \mid c$$

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

$$\begin{array}{ccccccc} * & * & * & * \\ S \xrightarrow{(1)} aA \xrightarrow{(2)} aaB \xrightarrow{(3)} aabC \xrightarrow{(4)} aabc = w \in L(G) \end{array}$$

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

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:

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

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

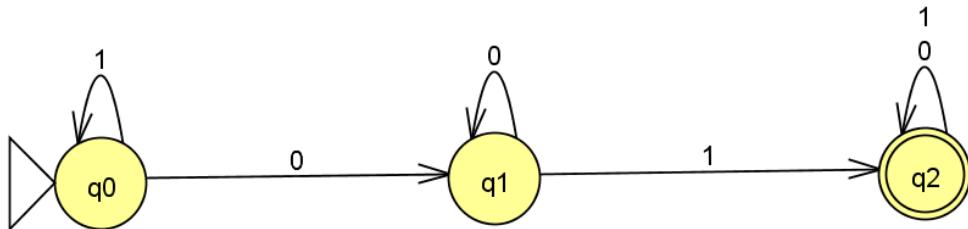
Definiția 3.3: Limbajul acceptat de un automat finit deterministic $A = (\Sigma, Q, \delta, q_0, F)$ este mulțimea $\mathcal{T}(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \in F\}$

Observație: dacă la gramatici ne interesează *limbajul generat*, la automate ne interesează *limbajul acceptat*!

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:

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

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

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

$$\delta(q_0, 110110) = \delta(q_0, 10110) = \delta(q_0, 0110) = \delta(q_1, 110) = \delta(q_2, 10) = \delta(q_2, 0) = q_2.$$

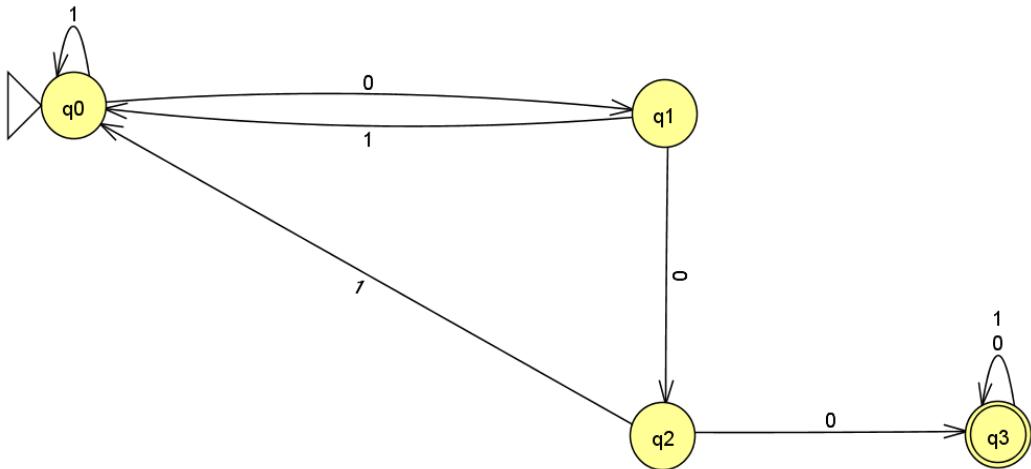
Dar $q_2 \in F \Rightarrow w_1 \in \mathcal{T}(A)$

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

$$\delta(q_0, 1100) = \delta(q_0, 100) = \delta(q_0, 00) = \delta(q_1, 0) = q_1.$$

Dar $q_1 \notin F \Rightarrow w_2 \notin \mathcal{T}(A)$

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:

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

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

Folosind definiția 3.3 arătăm că $w_1 = 11000 \in \mathcal{T}(A)$:

$$\delta(q_0, 11000) = \delta(q_0, 100) = \delta(q_0, 00) = \delta(q_1, 0) = \delta(q_2, 0) = q_3 .$$

Dar $q_3 \in F \Rightarrow w_1 \in \mathcal{T}(A)$

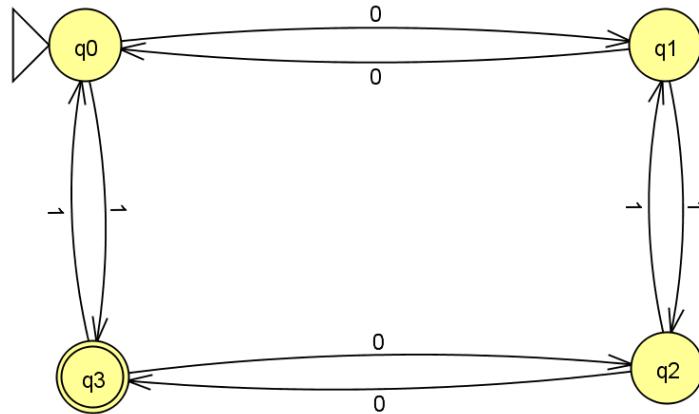
Folosind definiția 3.3 arătăm că $w_2 = 1001001 \notin \mathcal{T}(A)$:

$$\begin{aligned} \delta(q_0, 1001001) &= \delta(q_0, 001001) = \delta(q_1, 01001) = \delta(q_2, 1001) = \delta(q_0, 001) = \\ &= \delta(q_1, 01) = \delta(q_2, 1) = q_3 \notin F \end{aligned}$$

Exemplul 3.6: Automatul finit determinist $A = (\Sigma, Q, \delta, q_0, F)$ acceptă toate şirurile 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 : şirul conține un număr par de 0 și un număr par de 1;
- q_1 : şirul conține un număr impar de 0 și un număr par de 1;
- q_2 : şirul conține un număr impar de 0 și un număr impar de 1;
- q_3 : şirul 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:

	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

$$\Sigma = \{0,1\}$$

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

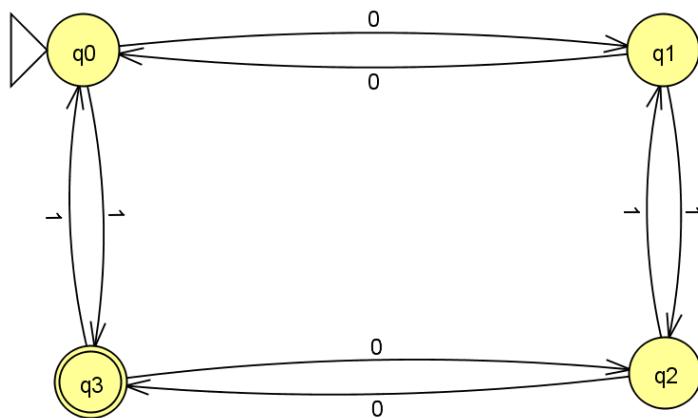
Folosind definiția 3.3 arătăm că $w_1 = 1001010 \in \mathcal{T}(A)$:

$$\begin{aligned}\delta(q_0, 1001010) &= \delta(q_3, 001010) = \delta(q_2, 01010) = \delta(q_3, 1010) = \delta(q_0, 010) = \\ &= \delta(q_1, 10) = \delta(q_2, 0) = q_3 \in F\end{aligned}$$

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

$$\begin{aligned}\delta(q_0, 1011001) &= \delta(q_3, 011001) = \delta(q_2, 11001) = \delta(q_1, 1001) = \delta(q_2, 001) = \\ &= \delta(q_3, 01) = \delta(q_2, 1) = q_0 \notin F\end{aligned}$$

Să demonstrăm că $\mathcal{T}(A) =$ mulțimea sirurilor binare cu un număr par de 0 și impar de 1.



Arătăm prin inducție după $n = |w|$ că din q_0 se poate ajunge în q_0, q_1, q_2, q_3 numai prin:

Stare	Nr. de 0	Nr. de 1
q_0	par	par
q_1	impar	par
q_2	impar	impar
q_3	par	impar

Pentru $n=1$ și $n=2$: evident.

Prin inducție, trecem de la n la $n+1$.

Considerăm o stare oarecare și analizăm pe rând stările anterioare.

În starea q_0 putem ajunge:

- din q_1 dacă w se termină cu 0; atunci ajungem la (par,par);
- sau din q_3 dacă w se termină cu 1; atunci ajungem la (par,par);

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.

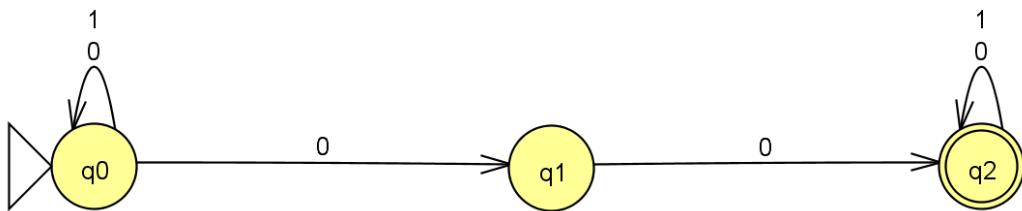
Practic, funcția de tranziție este definită prin $\delta: Q \times \Sigma \rightarrow \mathcal{P}(Q)$. Extinderea funcției de tranziție $\bar{\delta}: Q \times \Sigma^* \rightarrow P(Q)$ se va face astfel:

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

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

Deci un cuvânt $w \in \Sigma^*$ este acceptat de automat dacă (plecând din starea inițială) prin el putem ajunge într-o stare finală.

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:

- $\Sigma = \{0,1\}$
- $Q = \{q_0, q_1, q_2\}$

- $F = \{q_2\}$
- funcția de tranziție δ :

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

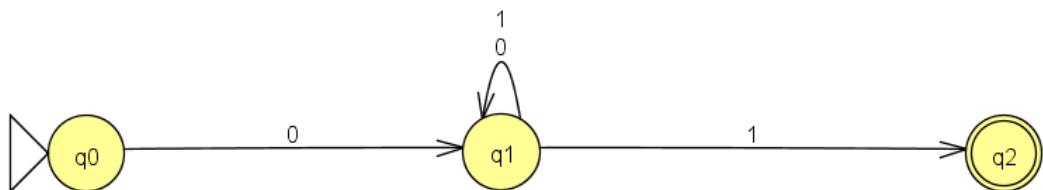
Folosind definiția 3.4 arătăm că $w_1 = 1101001 \in \mathcal{T}(A)$:

$$\begin{aligned} \delta(q_0, 1101001) &= \delta(\{q_0\}, 101001) = \delta(\{q_0\}, 01001) = \delta(\{q_0, q_1\}, 1001) = \delta(\{q_0\}, 001) = \\ &= \delta(\{q_0, q_1\}, 01) = \delta(\{q_0, q_1, q_2\}, 1) = \{q_0, q_2\} \\ \text{Dar } \{q_0, q_2\} \cap F &= \{q_2\} \neq \emptyset \Rightarrow w_1 \in \mathcal{T}(A) \end{aligned}$$

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

$$\begin{aligned} \delta(q_0, 10101) &= \delta(\{q_0\}, 0101) = \delta(\{q_0, q_1\}, 101) = \delta(\{q_0\}, 01) = \delta(\{q_0, q_1\}, 1) = \{q_0\} \\ \text{Dar } \{q_0\} \cap F &= \emptyset \Rightarrow w_2 \notin \mathcal{T}(A). \end{aligned}$$

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:

- $\Sigma = \{0, 1\}$
- $Q = \{q_0, q_1, q_2\}$

- $F = \{q_2\}$
- funcția de tranziție δ :

δ	0	1
q₀	{q ₁ }	∅
q₁	{q ₁ }	{q ₁ , q ₂ }
*	∅	∅

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

$$\delta(q_0, 01101) = \delta(\{q_1\}, 1101) = \delta(\{q_1, q_2\}, 101) = \delta(\{q_1, q_2\}, 01) = \delta(\{q_1\}, 1) = \{q_1, q_2\}$$

$$\text{Dar } \{q_1, q_2\} \cap F = \{q_2\} \neq \emptyset \Rightarrow w_1 \in \mathcal{T}(A)$$

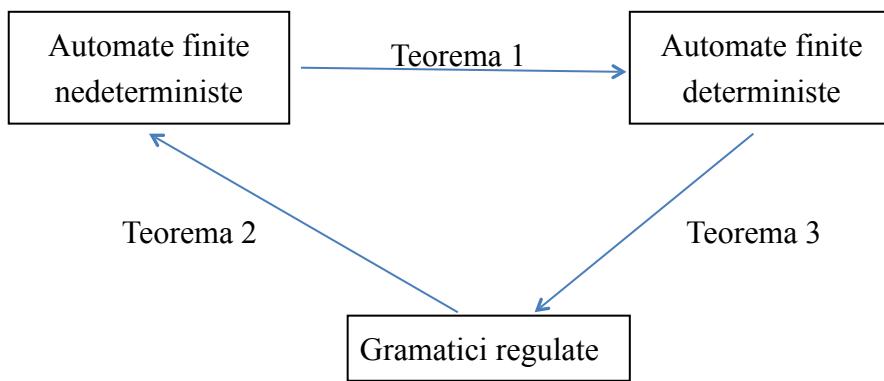
Folosind definiția 3.4 arătăm că $w_2 = 01010 \notin \mathcal{T}(A)$:

$$\delta(q_0, 01010) = \delta(\{q_1\}, 1010) = \delta(\{q_1, q_2\}, 010) = \delta(\{q_1\}, 10) = \delta(\{q_1, q_2\}, 0) = \{q_1\}$$

$$\text{Dar } \{q_1\} \cap F = \emptyset \Rightarrow w_2 \notin \mathcal{T}(A)$$

Echivalența dintre gramaticile regulate, automatele finite deterministe și automatele finite nedeterministe

Cele ce urmează pot fi sintetizate prin:



Teorema 1. Orice limbaj acceptat de un automat finit nedeterminist A_N este acceptat și de un automat finit determinist A_D echivalent.

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

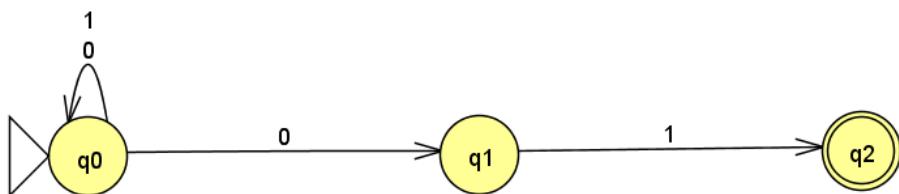
- $\sum_D = \sum_N$
- $Q_D = \mathcal{P}(Q_N)$
- funcția δ_D se va calcula pentru fiecare $q \in Q_D = \mathcal{P}(Q_N)$ și pentru fiecare $a \in \sum_N$:

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

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

Exemplu. Considerăm un automat finit nedeterminist $A_N = \left(\sum_N, Q_N, \delta_N, q_0^N, F_N \right)$ care acceptă sirurile binare care se termină cu 01 și determinăm automatul finit determinist $A_D = \left(\sum_D, Q_D, \delta_D, q_0^D, F_D \right)$ echivalent.

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



Analitic, automatul A_N va fi reprezentat astfel:

- $\sum_N = \{0, 1\}$
- $Q_N = \{q_0^N, q_1, q_2\}$
- $F_N = \{q_2\}$
- 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 $A_D = \left(\sum_D, Q_D, \delta_D, q_0^D, F_D \right)$ echivalent va fi construit

astfel:

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

	δ_D	0	1	
→	\emptyset	\emptyset	\emptyset	A
	$\{q_0^N\}$	$\{q_0^N, q_1\}$	$\{q_0^N\}$	B
	$\{q_1\}$	\emptyset	$\{q_2^N\}$	C
*	$\{q_2\}$	\emptyset	\emptyset	D
	$\{q_0^N, q_1\}$	$\{q_0^N, q_1\}$	$\{q_0^N, q_2\}$	E
*	$\{q_0^N, q_2\}$	$\{q_0^N, q_1\}$	$\{q_0^N\}$	F
*	$\{q_1, q_2\}$	\emptyset	$\{q_2\}$	G
*	$\{q_0^N, q_1, q_2\}$	$\{q_0^N, q_1\}$	$\{q_0^N, q_2\}$	H

Redenumind convenabil stările automatului A_D , putem redefini automatul astfel:

- $\sum_D = \{0,1\}$
- $Q_D = \{A, B, C, D, E, F, G, H\}$
- $F_D = \{D, F, G, H\}$

- funcția de tranziție δ_D :

δ_D	0	1
A	A	A
B	E	B
C	A	D
D	A	A
E	E	F
F	E	B
G	A	D
H	E	F

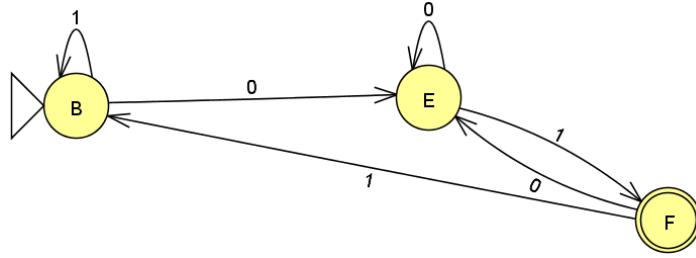
Observație. Prin aplicarea acestei metode, automatul finit determinist echivalent obținut conține și stări inaccesibile. Simplificarea sa constă în determinarea stărilor accesibile din starea inițială, care este întotdeauna accesibilă, și eliminarea stărilor inaccesibile.

Se observă ușor că în automatul A_D sunt accesibile doar stările B , F și E , 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
B	E	B
E	E	F
F	E	B

Reprezentarea grafică a automatului A_D este următoarea:



Teorema 2. Pentru orice gramatică regulată G se poate construi un automat finit nedeterminist A care să accepte limbajul generat de gramatica G , respectiv $L(G) = \mathcal{T}(A)$.

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:
 - $\delta(B, a) = \{C \in N \mid B \rightarrow aC \in P\} \cup \{X \mid B \rightarrow a \in P\}$
 - $\delta(X, a) = \emptyset, \forall a \in \Sigma$

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

$$S \rightarrow aA \mid bB \mid b$$

$$A \rightarrow aA \mid bB \mid b$$

$$B \rightarrow cB \mid c$$

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

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

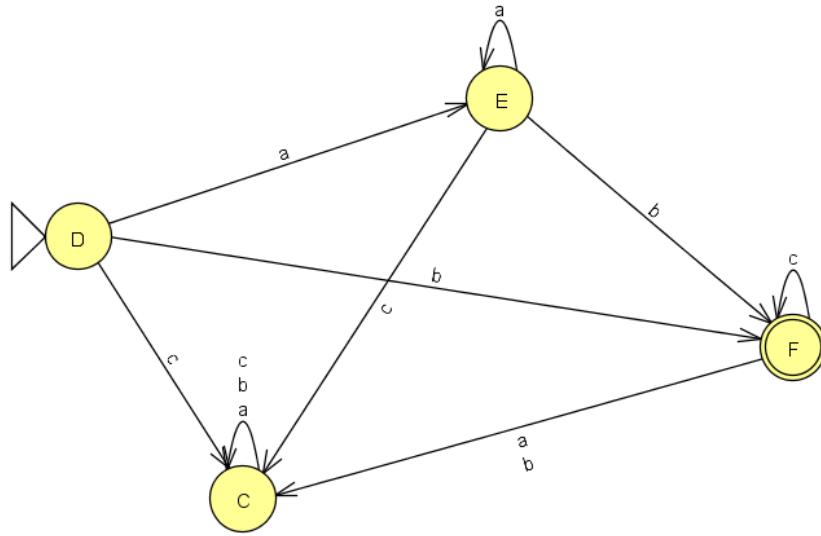
	δ_N	a	b	c
\rightarrow	S	{A}	{B, X}	\emptyset
	A	{A}	{B, X}	\emptyset
*	B	\emptyset	\emptyset	{B, X}
*	X	\emptyset	\emptyset	\emptyset

Automatul finit nedeterminist A_N poate fi reprezentat grafic astfel:

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

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

Teorema 3. Fie $A = (\Sigma, Q, \delta, q_0, F)$ un automat finit determinist. Atunci există G gramatică regulată cu $L(G) = T(A)$.



Construim $G = (N, T, S, P)$ astfel:

$$\begin{aligned} N &= Q & T &= \Sigma & S &= q_0 \\ P &= \{ A \rightarrow aB \mid \delta(A, a) = B \} \cup \{ A \rightarrow a \mid \delta(A, a) \in F \} \end{aligned}$$

Exemplu. Fie $A = (\Sigma, Q, \delta, q_0, F)$ a.f.d. și G gramatica regulată corespunzătoare:

	δ	0	1	P
\rightarrow	q_0	q_1	q_0	$q_0 \rightarrow 0q_1, \quad q_0 \rightarrow 1q_0$
	q_1	q_1	q_2	$q_1 \rightarrow 0q_1, \quad q_1 \rightarrow 1q_2, \quad q_1 \rightarrow 1$
*	q_2	q_2	q_2	$q_2 \rightarrow 0q_2, \quad q_2 \rightarrow 1q_2, \quad q_2 \rightarrow 0, \quad q_2 \rightarrow 1$

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 automatul 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 determinist $A = (\Sigma, K, \delta, s_0, F)$, 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.

Lema stelei

Fie L limbaj regulat \Rightarrow există k număr natural astfel încât pentru orice $w \in L$ cu $|w| \geq k$ există descompunerea:

$$w = xyz \text{ cu}$$

$$0 < |y| \leq k \quad (*)$$

$xy^iz \in L$ pentru orice $i = 0, 1, 2, \dots$

Fie $A = (\Sigma, Q, \delta, q_0, F)$ a.f.d. cu $T(A) = L$.

Aleg $k = |Q|$.

Fie $w \in L$ cu $|w| \geq k$. Deci de la q_0 la $\delta(q_0, w)$, automatul trece prin cel puțin $k+1$ stări. Aleg cele mai apropiate stări care se repetă; fie ele q' :

$$q_0 \rightarrow \dots \rightarrow q' \rightarrow \dots \rightarrow q' \rightarrow \dots \rightarrow \delta(q_0, w) \in F$$

și fie x, y și z cu $\delta(q_0, x) = q', \delta(q', y) = q', \delta(q', z) = \delta(q_0, w)$.

Evident, $|y| \leq k$. Se observă că:

- Pentru $i=0 : xz \in L$;
- Pentru $i \geq 2 : xy^iz \in L$.

Propoziție. Există un algoritm pentru a verifica dacă $L(G)$ este infinit sau nu, unde G este o gramatică regulată.

Arăt că $L(G)$ este infinit \iff există $w \in L(G)$ cu $k \leq |w| < 2k$.

“ \Leftarrow “ : Conform ipotezei, există $w \in L(G)$ cu $|w| \geq k$.

Conform lemei stelei, există descompunerea:

[

$w = xyz$ cu:

$$0 < |y| \leq k \quad (*)$$

$xy^iz \in L(G)$ pentru orice $i = 0, 1, \dots$

Dar $|xy^iz| < |xy^iz| < |xy^iz| < \dots$

“ \Rightarrow ”: Cum $L(G)$ este infinit, există $w \in L(G)$ cu $|w| \geq k$.

Dacă $|w| < 2k$, OK!

Altfel, consider $w' = xz \in L(G)$. Cum lungimea scade cu cel mult k , rezultă $|w'| \geq k$.

Reiau raționamentul pentru w' . La un moment dat vom avea $|w| < 2k$.

Deci este suficient să generez toate cuvintele $(N \cup T)^*$ de lungime $< 2k$ care derivă din S și să verific dacă printre ele există unul din T^* de lungime $\geq k$.

Exercițiu. Aplicând lema stelei, să se arate că limbajul $L = \{a^n b^n \mid n \geq 1\}$ nu este limbaj regulat.

Fie $2n > k$ din lema stelei.

Studiem cazurile când în descompunerea xyz a lui $a^n b^n$:

- y este format numai din a -uri: xy^2z nu este din L (prea mulți de a)
- y este format numai din b -uri; xy^2z nu este din L (prea mulți de b)
- y este format atât din a -uri, cât și din b -uri (în xy^2z apare b înaintea lui a).

GRAMATICI INDEPENDENTE DE CONTEXT

Derivări în gramatici independente de context

Definiție: 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 $A \in N$, $\alpha \in (N \bigcup T)^*$.

Definiție: Spunem ca o derivare este *derivare stângă* (notată prin \xrightarrow{S}) dacă la fiecare pas se înlocuiește, conform unei producții a gramaticii, neterminul cel mai din stânga.

Propoziție: 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:

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

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

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

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

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 $A_i \xrightarrow{\leq k} w_i$ 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 $i = 1, n$.

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

$$S \rightarrow aAS$$

$$A \rightarrow SbA$$

$$A \rightarrow SS$$

$$S \rightarrow a$$

$$A \rightarrow ba$$

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

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

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

$$S \xrightarrow[\text{(1)}]{s} aAS \xrightarrow[\text{(5)}]{s} abaS \xrightarrow[\text{(1)}]{s} abaaAS \xrightarrow[\text{(2)}]{s} abaaSbAS \xrightarrow[\text{(4)}]{s} abaaabAS \xrightarrow[\text{(5)}]{s} abaaabbaS \xrightarrow[\text{(4)}]{s} abaaabbbaa = w$$

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.

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

Observații:

1. Dacă $\lambda \in L(G)$, atunci există cel puțin o λ -producție în G .
2. Dacă G are o λ -producție atunci nu neapărat rezultă că $\lambda \in L(G)$.

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

$$S \rightarrow aA$$

$$A \rightarrow aA$$

$$A \rightarrow \lambda$$

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

Teoremă: 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.

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

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

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

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 .

Exemplu: Pentru a elimina λ -producțiile din gramatica independentă de context G din exemplu considerat, aplicăm algoritmul descris mai sus 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: 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:

$$S \rightarrow X \mid XY \mid Z$$

$$X \rightarrow Z \mid \lambda$$

$$Y \rightarrow Wa \mid a$$

$$Z \rightarrow WX \mid AZ \mid Zb$$

$$W \rightarrow XYZ \mid bXa \mid \lambda$$

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

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

$$Z \xrightarrow[4]{*} WX \xrightarrow[2]{*} W \xrightarrow[5]{*} \lambda.$$

Pasul 2: Determinăm mulțimea P' inițială:

$$S \rightarrow X \mid XY \mid Z$$

$$\begin{aligned}
X &\rightarrow Z \\
Y &\rightarrow Wa \mid a \\
Z &\rightarrow WX \mid AZ \mid Zb \\
W &\rightarrow XYZ \mid bXa
\end{aligned}$$

Pasul 3: Determinăm mulțimea P' finală, determinând mai întâi producțiile care să „compenseze” λ-producțiile existente în P și eliminate:

Producție inițială	Producții echivalente
$S \rightarrow X$	$S \rightarrow \lambda$
$S \rightarrow XY$	$S \rightarrow Y$
$S \rightarrow Z$	$S \rightarrow \lambda$
$Y \rightarrow Wa$	$Y \rightarrow a$
	$Z \rightarrow W$
$Z \rightarrow WX$	$Z \rightarrow X$ $Z \rightarrow \lambda$
$Z \rightarrow aZ$	$Z \rightarrow a$
$Z \rightarrow Zb$	$Z \rightarrow b$
	$W \rightarrow YZ$
$W \rightarrow XYZ$	$W \rightarrow XY$ $W \rightarrow Y$
$W \rightarrow bXa$	$W \rightarrow ba$

Obținem astfel că mulțimea P' este următoarea:

$$\begin{aligned}
S &\rightarrow X \mid XY \mid Z \mid Y \\
&\quad X \rightarrow Z \\
&\quad Y \rightarrow Wa \mid a
\end{aligned}$$

$$\begin{aligned} Z &\rightarrow WX \mid aZ \mid Zb \mid W \mid X \mid a \mid b \\ W &\rightarrow XYZ \mid bXa \mid YZ \mid XY \mid Y \mid ba \end{aligned}$$

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

Teoremă: 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 $P_1 = \{A \rightarrow B \mid A, B \in N \text{ și } A \rightarrow B \in P\}$ 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 $\alpha \xrightarrow[G']{*} \beta$ rezultă și că $\alpha \xrightarrow[G]{*} \beta$.
- $L(G) \subset L(G')$: 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.

Propoziție: 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

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

$$S \rightarrow A$$

$$A \rightarrow aSB$$

$$A \rightarrow B$$

$$B \rightarrow bSA$$

$$A \rightarrow a$$

$$B \rightarrow b$$

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:

$$\begin{array}{ll}
 A \rightarrow aSB & \} \text{producțiile din } P \text{ care nu sunt redenumiri} \\
 B \rightarrow bSA & \\
 A \rightarrow a & \\
 B \rightarrow b & \\
 S \rightarrow aSB & \} \text{producții noi} \\
 S \rightarrow a & \\
 A \rightarrow bSA &
 \end{array}$$

$$A \rightarrow b$$

$$S \rightarrow b$$

$$S \rightarrow bSA$$

Exemplu: 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:

$$S \rightarrow X \mid Y \mid bb$$

$$X \rightarrow Z \mid aXY$$

$$Y \rightarrow Xa \mid a$$

$$Z \rightarrow XY \mid S \mid Zb$$

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
$S \rightarrow X$	$S \rightarrow Z \mid aXY$
$S \rightarrow Y$	$S \rightarrow Xa \mid a$
$X \rightarrow Z$	$X \rightarrow YX \mid Zb \mid b$
$Z \rightarrow S$	$Z \rightarrow X \mid Y \mid bb$
$\begin{array}{c} \xrightarrow{*} \\ S \end{array} \xrightarrow{*} \begin{array}{c} \xrightarrow{*} \\ X \end{array} \xrightarrow{*} \begin{array}{c} \xrightarrow{*} \\ Z \end{array}$ $\begin{array}{c} * \\ S \end{array} \Rightarrow Z$	$S \rightarrow XY \mid S \mid Zb$
$\begin{array}{c} \xrightarrow{*} \\ X \end{array} \xrightarrow{*} \begin{array}{c} \xrightarrow{*} \\ Z \end{array} \xrightarrow{*} \begin{array}{c} \xrightarrow{*} \\ S \end{array}$ $\begin{array}{c} * \\ X \end{array} \Rightarrow S$	$X \rightarrow X \mid Y \mid bb$
$\begin{array}{c} \xrightarrow{*} \\ X \end{array} \xrightarrow{*} \begin{array}{c} \xrightarrow{*} \\ Z \end{array} \xrightarrow{*} \begin{array}{c} \xrightarrow{*} \\ S \end{array} \xrightarrow{*} \begin{array}{c} \xrightarrow{*} \\ Y \end{array}$ $\begin{array}{c} * \\ X \end{array} \Rightarrow Y$	$X \rightarrow Xa \mid a$

$Z \xrightarrow{*} S \xrightarrow{*} X$	$Z \rightarrow Z \mid aXY$
$Z \xrightarrow{*} X$	
$Z \xrightarrow{*} Y$	$Z \rightarrow Xa \mid a$

Obținem că mulțimea producțiilor P' este următoarea:

$$\begin{aligned}
 S &\rightarrow bb \mid aXY \mid Xa \mid a \mid XY \mid Zb \\
 X &\rightarrow aXY \mid XY \mid Zb \mid bb \mid Xa \mid a \\
 Y &\rightarrow Xa \mid a \\
 Z &\rightarrow YX \mid Zb \mid bb \mid aXY \mid Xa \mid a
 \end{aligned}$$

Definiție: Fie $G = (N, T, S, P)$ o gramatică independentă de context și $X \in N$.

Spunem că X este *neterminal observabil* dacă există cel puțin o derivare de forma $X \xrightarrow{*} w$, unde $w \in T^*$. În caz contrar spunem că X este *neterminal neobservabil*.

Observație: Este evident faptul că numai neterminalele observabile sunt utile în generarea cuvintelor din $L(G)$, deci producțiile care conțin în membrul stâng sau în cel drept neterminalele neobservabile pot fi eliminate din P .

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

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

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 observabile. Putem elimina acum producțiile care conțin neterminale neobservabile, adică neterminalele din mulțimea $N \setminus M_k$.

Exemplu: 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:

$$S \rightarrow AB \mid aBC$$

$$A \rightarrow BA \mid a$$

$$B \rightarrow b \mid AC$$

$$C \rightarrow AC \mid CB$$

$$D \rightarrow AD \mid a$$

În continuare, aplicăm propoziția de mai sus pentru a determina neterminalele observabile ale gramaticii G :

$$M_0 = \{A, B, D\}$$

$$M_1 = M_0 \cup \{S, A, B, D\} = \{S, A, B, D\}$$

$$M_2 = M_1 \cup \{S, A, B, D\} = \{S, A, B, D\}$$

Se observă că în acest moment sirul de mulțimi s-a stabilizat deoarece $M_2 = M_1$, deci neterminalele observabile sunt cele din $M_2 = \{S, A, B, D\}$. Rezultă că singurul neterminal neobservabil 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 neobservabile, unde $N' = \{S, A, B, D\}$, iar mulțimea producțiilor P' este următoarea:

$$S \rightarrow AB$$

$$A \rightarrow BA \mid a$$

$$B \rightarrow b$$

$$D \rightarrow AD \mid a$$

Definiție: 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*.

Observație: 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: Simbolurile accesibile ale unei grămatice independente de context $G = (N, T, S, P)$ se pot obține prin metoda sirului crescător de mulțimi, astfel:

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

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 simbolurilor accesibile. Putem elimina acum producțiile care conțin simboluri inaccesibile, adică simbolurile din mulțimea $(N \cup T) \setminus M_k$.

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

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow BA \mid a \\ B &\rightarrow b \\ D &\rightarrow AD \mid a \end{aligned}$$

Aplicăm propoziția de mai sus pentru a determina simbolurile accesibile ale grămaticii G :

$$\begin{aligned} M_0 &= \{S\} \\ M_1 &= M_0 \cup \{A, B\} = \{S, A, B\} \\ M_2 &= M_1 \cup \{A, B, a, b\} = \{S, A, B, a, b\} \\ M_3 &= M_2 \cup \{A, B, a, b\} = \{S, A, B, a, b\} \end{aligned}$$

Se observă că în acest moment sirul 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 grămatica 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 mulțimea producțiilor P' este următoarea:

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow BA \mid a \\ B &\rightarrow b \end{aligned}$$

Forma normală Chomsky a unei gramatici independente de context

Definiție: 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 $a \in T$.

Propoziție: 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 $A \rightarrow B_1B_2\dots B_m$, unde $a \in T$, $B_i \in N$ pentru $\forall i \in \{1, 2, \dots, m\}$ și $m \geq 2$.

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 corespondențul lor din T' .

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

Teoremă: 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 $m > 2$ 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

$$\left\{ \begin{array}{l} A \rightarrow B_1D_1 \\ D_1 \rightarrow B_2D_2 \\ \dots \\ D_{m-2} \rightarrow B_{m-1}B_m \end{array} \right.$$

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

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

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

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

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

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

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

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 :

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

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

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

În continuare, aplicăm propoziția anterioară 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:

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

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

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

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

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

În acest moment putem aplica asupra gramaticii G_2 teorema de mai sus, 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:

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

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

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

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

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

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

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

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

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

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

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

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

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

$$B \rightarrow BbB \mid BA b \mid b \quad (3)$$

Deoarece gramatica G nu conține redenumiri, putem să aplicăm propoziția anterioară 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:

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

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

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

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

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

În acest moment putem aplica asupra gramaticii G_1 teorema de mai sus, 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:

$$\begin{aligned} S &\rightarrow \tilde{A}X_1 & (1) \\ X_1 &\rightarrow AX_2 & (2) \\ X_2 &\rightarrow BA & (3) \\ A &\rightarrow AX_3 & (4) \\ X_3 &\rightarrow \tilde{A}A & (5) \\ A &\rightarrow AB & (6) \\ A &\rightarrow a & (7) \\ B &\rightarrow BX_4 & (8) \\ X_4 &\rightarrow \tilde{B}B & (9) \\ B &\rightarrow BX_5 & (10) \\ X_5 &\rightarrow A\tilde{B} & (11) \\ B &\rightarrow b & (12) \\ \tilde{A} &\rightarrow a & (13) \\ \tilde{B} &\rightarrow b & (14) \end{aligned}$$

Exemplu: 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 :

$$\begin{aligned} S &\rightarrow aB \mid AC & (1) \\ A &\rightarrow ASC \mid BC \mid aD \mid a & (2) \\ B &\rightarrow bS \mid b & (3) \\ C &\rightarrow BA \mid \lambda & (4) \\ D &\rightarrow abC & (5) \\ E &\rightarrow aB & (6) \end{aligned}$$

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

a) Eliminarea λ -producțiilor din gramatica G :

Determinăm mai întâi mulțimea $M = \left\{ A \in N \mid A \xrightarrow{*} \lambda \in P \right\} = \{C\}$.

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:

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

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

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

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

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

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

b) Eliminarea simbolurilor neobservabile din gramatica G_1 :

În continuare, vom determina neterminalele neobservabile ale gramaticii G_1 :

$$M_0 = \{A, B, D\}$$

$$M_1 = \{A, B, D\} \cup \{S, A, B, C, D, E\} = \{S, A, B, C, D, E\}$$

Dar $M_1 = N \Rightarrow$ algoritmul se termină, deoarece gramatica G_1 nu are neterminale neobservabile.

c) Eliminarea simbolurilor inaccesibile din gramatica G_1 :

Aplicăm algoritmul descris pentru a determina simbolurile accesibile ale gramaticii G_1 :

$$M_0 = \{S\}$$

$$M_1 = \{S\} \cup \{A, B, C, a\} = \{S, A, B, C, a\}$$

$$M_2 = \{S, A, B, C, a\} \cup \{D, b\} = \{S, A, B, C, D, a, b\}$$

$$M_3 = M_2$$

Se observă că în acest moment sirul de mulțimi s-a stabilizat, deci simbolurile accesibile sunt cele din $M_2 = \{S, A, B, C, D, a, b\}$. 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 mulțimea producțiilor P_2 este următoarea:

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

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

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

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

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

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
-------------------	----------------------

$S \rightarrow A$	$S \rightarrow aSC$ $S \rightarrow BC$ $S \rightarrow AS$ $S \rightarrow aD$
$A \rightarrow B$	$A \rightarrow b$ $A \rightarrow bS$
$S \xrightarrow{*} B$	$S \rightarrow b$ $S \rightarrow bS$

Astfel, obținem că mulțimea producțiilor P_3 este următoarea:

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

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

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

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

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

e) Aducerea gramaticii G_3 la forma normală Chomsky:

Obținem mai întâi, plecând de la G_3 , 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:

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

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

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

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

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

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

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

În acest moment putem aplica asupra gramaticii G_4 înlocuirea producțiilor cu mai mult de două neterminale în membrul drept cu producții echivalente cu exact două neterminale în membrul drept. Obținem 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:

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

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

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

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

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

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

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

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

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

Arborei de derivare

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

Definiție: 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 $v_1, \boxed{?}, v_k$ (etichetate cu $A_1, \boxed{?}, A_k$) sunt în ordine de la stânga la dreapta descendenți direcți ai vârfului v (etichetat cu A), atunci $A \rightarrow A_1 \boxed{?} A_k \in P$.

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

$$S \rightarrow aAS$$

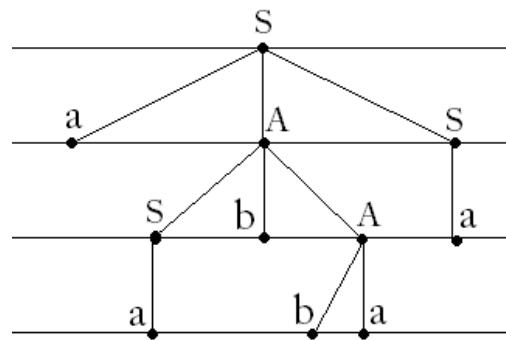
$$S \rightarrow a$$

$$A \rightarrow SbA$$

$$A \rightarrow SS$$

$$A \rightarrow ba$$

Un arbore de derivare în gramatica G este următorul:



Se observă ușor că acest arbore corespunde următoarei derivări:

$$S \xrightarrow{*} aAS \xrightarrow{*} aSbAS \xrightarrow{*} aSbAAa \xrightarrow{*} aabAAa \xrightarrow{*} aabbaaa = a^2b^2a^2$$

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 .

Definiție: Frontiera unui arbore de derivare este cuvântul format din etichetele vâfurilor terminale, în ordine de la stânga la dreapta. Altfel spus, frontieră unui arbore este mulțimea etichetelor vâfurilor terminale, considerate în preordine.

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

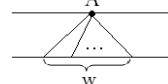
Observație: Fie $G = (N, T, S, P)$ o gramatică independentă de context și fie $A \in N$. Vom nota cu $G_A = (N, T, A, P)$ gramatica independentă de context obținută din gramatica G prin înlocuirea simbolului inițial S cu A .

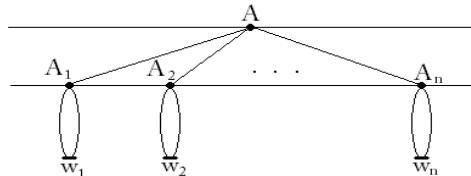
Teoremă: Fie $G = (N, T, S, P)$ o gramatică independentă de context și fie $w \in T^*$. Atunci $w \in L(G)$ dacă și numai dacă în G există un arbore de derivare cu frontiera w .

Demonstrație:

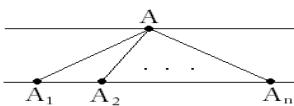
Pentru orice $A \in N$ arătăm că $A \xrightarrow{*} w$ dacă și numai dacă în G_A există un arbore de derivare cu frontiera w .

" \Rightarrow " Inducție după lungimea k a derivării:

- Pentru $k = 1: A \xrightarrow{} w \in P$ și arborele de derivare este următorul:
 
- Presupunem adevărat pentru orice $A \in N$ și orice derivare de lungime k . Fie derivarea $A \xrightarrow{k \text{ pași}} A_1A_2\dots A_n \xrightarrow{*} w$. Atunci $w = w_1w_2\dots w_k$ 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 $k = 1: A \xrightarrow{} A_1 \quad A_1 \xrightarrow{*} w_1 \in P$
- 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 $\xrightarrow{*} A \xrightarrow{} A_1 \dots A_n \in P$. 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 $A \xrightarrow{*} A_1 \dots A_n \xrightarrow{*} w_1 \dots w_n = w$.

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

Exemplu: Gramatica independentă de context G de mai sus este echivalentă cu gramatica independentă 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:

$$S \rightarrow BE$$

$$E \rightarrow AS$$

$$S \rightarrow a$$

$$A \rightarrow SD$$

$$D \rightarrow CA$$

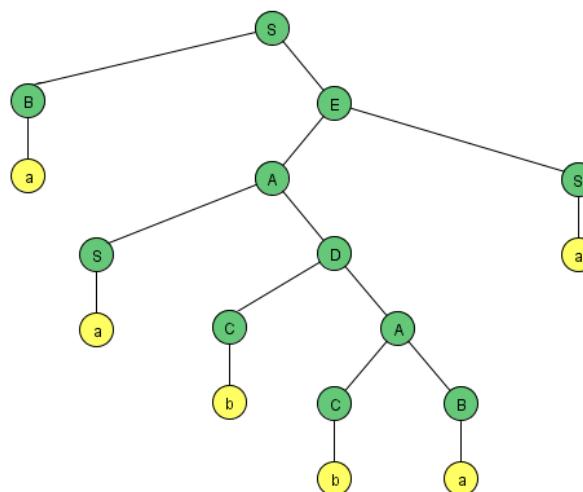
$$A \rightarrow SS$$

$$A \rightarrow CB$$

$$C \rightarrow b$$

$$B \rightarrow a$$

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



Definiție: 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 distințe.

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

$$S \rightarrow ab$$

$$S \rightarrow aSb$$

$$S \rightarrow aA$$

$$A \rightarrow aSbb$$

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

$S \xrightarrow[2]{*} aSb \xrightarrow[2]{*} aaSbb \xrightarrow[1]{*} aaabbb$	$S \xrightarrow[3]{*} aA \xrightarrow[4]{*} aaSbb \xrightarrow[1]{*} aaabbb$
---	--

Exemplu: 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:

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

$$S \rightarrow S + S$$

$$S \rightarrow S * S$$

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

$\underset{(1)}{\underbrace{S}} \underset{(2)}{\underbrace{+ S}} \underset{(3)}{\underbrace{+ S * S}} \underset{(3)}{\underbrace{\Rightarrow 2 * S + S}} \underset{(3)}{\underbrace{\Rightarrow}}$	$\underset{(2)}{\underbrace{S}} \underset{(3)}{\underbrace{* S}} \underset{(1)}{\underbrace{2 * S}} \underset{(3)}{\underbrace{\Rightarrow 2 * S + S}} \underset{(3)}{\underbrace{\Rightarrow}}$
$\underset{(3)}{\underbrace{\Rightarrow 2 * 3 + S}} \underset{(3)}{\underbrace{\Rightarrow 2 * 3 + 4}}$	$\underset{(3)}{\underbrace{\Rightarrow 2 * 3 + S}} \underset{(3)}{\underbrace{\Rightarrow 2 * 3 + 4}}$

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.

Algoritmul Cocke-Younger-Kasami

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 $A_{i,j} = \left\{ X \in N \mid X \stackrel{*}{\Rightarrow} w_{i,j} \right\}$.

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

$$A_{i,1} = \left\{ X \in N \mid X \rightarrow a_i \in P \right\}, \quad \forall i \in \{1, \dots, n\}$$

$$A_{i,j} = \bigcup_{\substack{2 \leq j \leq n \\ 1 \leq i \leq n+1-j \\ 1 \leq k \leq j-1}} \left\{ X \in N \mid \exists Y \in A_{i,k}, \exists Z \in A_{i+k, j-k} \text{ a. } \hat{t}. \quad X \rightarrow^{\hat{t}} YZ \in P \right\}$$

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

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

$$S \rightarrow aSb \mid aDb$$

$$D \rightarrow aD \mid a$$

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:

$$S \rightarrow AB \mid AC$$

$$A \rightarrow a$$

$$B \rightarrow SB \mid b$$

$$C \rightarrow DB$$

$$D \rightarrow AD \mid a$$

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	$\{S, B, C\}$						

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

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

Exerciții propuse

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

$$S \rightarrow X$$

$$X \rightarrow XY \mid aYX \mid a \mid \lambda$$

$$Y \rightarrow X \mid XYb \mid b$$

- 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:

$$S \rightarrow X \mid Y \mid XY$$

$$X \rightarrow aXYX \mid bXY \mid Y \mid a$$

$$Y \rightarrow XYXY \mid aXYb \mid b \mid \lambda$$

- 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:

$$S \rightarrow A \mid B \mid aA \mid bB$$

$$A \rightarrow aAa \mid bAb \mid B \mid \lambda$$

$$B \rightarrow bBb \mid aBa \mid A \mid \lambda$$

$$C \rightarrow cCc \mid aABb \mid \lambda$$

- 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:

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow BC \mid a \\ B &\rightarrow AC \mid b \\ C &\rightarrow a \mid b \end{aligned}$$

- 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:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

- 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:

$$\begin{aligned} S &\rightarrow () \\ S &\rightarrow (S) \\ S &\rightarrow SS \end{aligned}$$

- a) Aduceți gramatica G la forma normală Chomsky.

b) Folosind algoritmul CYK arătați că $w_1 = ((())()) \in L(G)$, iar $w_2 = ()() \notin L(G)$.

Lema de pompare

1. Limbajul de programare S . Funcții calculabile

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, optional, 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 nulă, 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:

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

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.*

Fie P un program scris în limbajul S . Fie V mulțimea tuturor variabilelor (de intrare, de lucru și de ieșire) ce apar în P . Vom da în continuare o definiție riguroasă a *semanticii* programului P .

Definim *starea* programului la un moment oarecare a executării sale ca fiind o funcție $s:V \rightarrow \mathbb{N}$, unde \mathbb{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:

$$s_0(u) = \begin{cases} r_i, & \text{dacă } u = x_i \\ 0, & \text{dacă } u \end{cases} \quad \text{ui } x_i$$

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:

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

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:

$$j = \begin{cases} i + 1, & \text{dacă } s(v) = \\ n + 1, & \text{dacă } s(v) \neq \\ k, & \text{dacă } s(v) \neq \end{cases} \quad \begin{array}{ll} \text{ne etichetat} & L \\ & \\ \text{etichetat} & L \end{array}$$

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

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

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

este calculabilă.

Exemplul 2 arată că funcția unară cu domeniul vid este parțial calculabilă.

2. Macroinstructiuni. Exemple de funcții calculabile

Macroinstructiunile sunt abrevieri pentru sevențe 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 distincte). 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 L
```

2) Macroinstructiunea $v \leftarrow 0$ 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:

$v \leftarrow 0$

$v \leftarrow v + 1$

...

$v \leftarrow v + 1$

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

Exemplul 3. Funcția $f : \mathbf{N} \rightarrow \mathbf{N}$ dată de $f(x) = x$ este calculabilă.

Pentru demonstrație, vom scrie următorul 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] $x \leftarrow x - 1$

$y \leftarrow y + 1$

$z \leftarrow z + 1$

goto A

[C] if $z \neq 0$ **goto** D

goto E

[D] $z \leftarrow z - 1$

$x \leftarrow x + 1$

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:

```

 $v \leftarrow 0$ 

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

care "transcrie" funcția din exemplul 4. Se impun următoarele trei remarci:

- nu am inițializat pe z cu 0, deoarece este clar că după orice executare a macroinstructiunii, valoarea sa va fi 0;
- apare necesitatea instrucțiunii $v \leftarrow v$;
- 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.

Un program care calculează această funcție este următorul:

$z \leftarrow x_2$

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

goto E

[A] $z \leftarrow z - 1$

$y \leftarrow y + x_1$

goto B

iar macroinstructiunea $z \leftarrow z_1 \times z_2$ 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 $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ 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:

$y \leftarrow x_1$

$z \leftarrow x_2$

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

goto E

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

goto A $\{y = 0 \text{ și } z \neq 0\}$

[B] $y \leftarrow y - 1$

$z \leftarrow z - 1$

goto C

Macroinstrucțiunea asociată este $Z \leftarrow Z_1 - Z_2$ și are următoarea dezvoltare posibilă:

$u_1 \leftarrow z_1$

$u_2 \leftarrow z_2$

$[C]$ **if** $u_2 \neq 0$ **goto** A

goto E

$[A]$ **if** $u_1 \neq 0$ **goto** B

goto A

$[B]$ $u_1 \leftarrow u_1 - 1$

$u_2 \leftarrow u_2 - 1$

goto C

$[E]$ $z \leftarrow u_1$

Codificarea programelor

1. Reprezentarea perechilor și n -uplelor ca un singur număr

- *Reprezentarea unei perechi de numere naturale ca un singur număr natural*

Orice pereche (x,y) de numere naturale poate fi reprezentată ca un număr natural prin bijecția $\langle x,y \rangle = 2^x(2y+1)-1$.

De exemplu $\langle 2,3 \rangle = 2^2(2 \cdot 3 + 1) - 1 = 4 \cdot 7 - 1 = 27$.

Funcția astfel definită este bijectivă deoarece pentru orice $z \in \mathbb{N}$, există și sunt unice numerele naturale x,y cu $\langle x,y \rangle = z$ astfel:

- x este cea mai mare putere a lui 2 ce divide pe $z+1$;
- y poate fi determinat de relația $2y+1 = (z+1)/2^x$ (membrul drept al acestei relații fiind impar).

De exemplu $27 = 4 \cdot 7 - 1 = 2^2(2 \cdot 3 + 1) - 1 = \langle 2,3 \rangle$.

- *Reprezentarea unui n -uplu de numere naturale ca un singur număr natural*

Pentru reprezentarea unui n -uplu $[a_1, \dots, a_n]$ 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 $[2,0,1] = 2^2 \cdot 3^0 \cdot 5^1 = 20$.

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:

- pentru orice x nenul, există n, a_1, \dots, a_n cu $[a_1, \dots, a_n] = x$ (deci funcția f este surjectivă);
- din $[a_1, \dots, a_n] = [b_1, \dots, b_n]$ rezultă $a_i = b_i$ pentru $\forall i=1, \dots, n$.

De exemplu: $20 = 2^2 \cdot 3^0 \cdot 5^1 = [2,0,1]$

Mai observăm că $[x_1, \dots, x_n] = [x_1, \dots, x_n, 0]$. Prin urmare funcția ce atașează oricărui n -uplu numărul său Gödel nu este injectivă.

De exemplu: $1 = [0] = [0,0] = [0,0,0, \dots]$, ceea ce ne permite să definim **numărul Gödel** atașat secvenței vide (cu $n=0$) ca fiind 1.

2. Numărul atașat unei instrucțiuni

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:

$y, x_1, z_1, x_2, z_2, \dots$

și le atașăm pozițiile lor în acest sir prin funcția $\#$ astfel:

$\#(y) = 1, \#(x_i) = 2i, \#(z_i) = 2i+1$ pentru $\forall i=1, \dots$

Știm că fiecare instrucțiune poate avea atașată o etichetă. Vom numerota etichetele folosite în program cu E_1, E_2, \dots

Putem acum extinde 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;
- unică variabilă v ce intervene în ea.

Corespunzător, pentru fiecare instrucțiune definim trei numere naturale a, b, c astfel:

$$a = \begin{cases} 0, & \text{dac } I \\ \#(L), & \text{dac } I \end{cases} \quad L$$

$$b = \begin{cases} 0, & \text{dac } I \\ 1, & \text{dac } I \\ 2, & \text{dac } I \\ \#(L) + 2, & \text{dac } I \end{cases} \quad \begin{array}{l} v \leftarrow v \\ v \leftarrow v + \\ v \leftarrow v - \\ \text{if } v \neq \text{ goto } L \end{array}$$

$$c = \#(v) - 1$$

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:

$$[E_1] \quad y \leftarrow y + 1$$

Atunci $a=1, b=1, c=1$ și deci $\langle b, c \rangle = 5$ și $\#(I) = \langle 1, 5 \rangle = 21$.

Pentru instrucțiunea :

$$x_2 \leftarrow x_2 - 1$$

avem $a=0, b=2, c=3$, deci $\langle b, c \rangle = 4 \cdot 7 - 1 = 27$, iar $\langle a, \langle b, c \rangle \rangle = \langle 0, 27 \rangle = 54$.

Restricția funcției $\#$ la mulțimea instrucțiunilor, funcție având codomeniul \mathbb{N} , este bijectivă, deoarece funcția $\langle \cdot, \cdot \rangle$ este bijectivă.

De exemplu $54 = \langle 0, 27 \rangle = \langle 0, \langle 2, 3 \rangle \rangle$, deci $a=0, b=2, c=3$.

Să mai observăm că unica instrucțiune I cu $\#(I)=0$ este instrucțiunea neetichetată $y \leftarrow y$.

3. Numărul atașat unui program

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 evident calculabilă.

Ținând 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:

"*Niciun 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; acest fapt se putea deduce și în alte moduri, dar cel de mai sus permite chiar "numerotarea programelor".

Exemplul 2. Căutăm programul P al cărui număr atașat este $\#(P) = 199$.

Cum $199 + 1 = 200 = 2^3 \cdot 3^0 \cdot 5^2$, rezultă că programul P este format din 3 instrucțiuni, ale căror numere atașate sunt în ordine 3, 0 și 2.

$\#(I_1) = 3 = \langle a, \langle b, c \rangle \rangle \Rightarrow a=2$ și $\langle b, c \rangle = 0 \Rightarrow b=c=0$, deci I_1 este: $[E_2] \quad y \leftarrow y$.

$\#(I_2) = 0$, deci I_2 este instrucțiunea neetichetată: $y \leftarrow y$.

$\#(I_3) = 2 = \langle a, \langle b, c \rangle \rangle \Rightarrow a=0$ și $\langle b, c \rangle = 1 \Rightarrow b=1$ și $c=0$, deci I_3 este: $y \leftarrow y+1$.

Rezultă că programul P căutat este următorul:

$[E_2]$ $y \leftarrow y$
 $y \leftarrow y$
 $y \leftarrow y+1$

Să mai facem următoarele observații:

- programul vid are numărul atașat egal cu: $1-1 = 0$;
- corespondențele de mai sus sunt cele care au făcut necesare admiterea etichetării unor instrucțiuni diferite din același program cu aceeași etichetă.

4. 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 predicalul binar definit astfel:

$$\text{HALT}(x_1, x_2) = 1 \iff \text{programul } P \text{ cu } \#(P) = x_2 \text{ se termină pentru valoarea } x_1.$$

Teorema 1. Predicalul HALT nu este calculabil.

Demonstrație. Să presupunem prin absurd că predicalul HALT ar fi calculabil. Atunci putem considera următorul program (notat prin P_0) ce constă din unica instrucțiune:

[A] **if** $\text{HALT}(x, x)$ **goto** A

Atunci funcția de un argument calculată de P_0 este $p(x)$:

- nedefinită dacă $\text{HALT}(x, x) = 1$ sau
- 0 dacă $\text{HALT}(x, x) = 0$

Fie $y_0 = \#(P_0)$. Atunci, conform definiției lui HALT :

$\text{HALT}(x, y_0) = 1 \iff p(x)$ este definită $\iff \text{HALT}(x, x) = 0$ (programul cu numărul x se termină pentru valoarea x).

Punând $x = y_0$, obținem:

$$\text{HALT}(y_0, y_0) = 1 \iff \text{HALT}(y_0, y_0) = 0$$

ajungându-se astfel la o contradicție.

Cum predicalul 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*.

2. Limbajul de programare S . Funcții calculabile

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, optional, 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]    $x \leftarrow x - 1$ 

 $y \leftarrow y + 1$ 

if  $x \neq 0$  goto  $A$ 

```

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

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

Exemplul 2. Programul:

```

[A]    $x \leftarrow x + 1$ 

if  $x \neq 0$  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*.

Fie P un program scris în limbajul S . Fie V mulțimea tuturor variabilelor (de intrare, de lucru și de ieșire) ce apar în P . Vom da în continuare o definiție riguroasă a *semanticii* programului P .

Definim *starea* programului la un moment oarecare a executării sale ca fiind o funcție $s: V \rightarrow \mathbb{N}$, unde \mathbb{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 initială* are forma $(1, s_0)$, unde:

$$s_0(u) = \begin{cases} r_i, & \text{dacă } u = x_i \\ 0, & \text{dacă } u \end{cases} \quad \text{ui } x_i$$

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:

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

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

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

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

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

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

$$j = \begin{cases} i+1, & \text{dacă } s_v = \\ n+1, & \text{dacă } s_v \neq \\ k, & \text{dacă } s_v \neq \end{cases}$$

ne etichetat L
etichetat L

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

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

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

este calculabilă.

Exemplul 2 arată că funcția unară cu domeniul vid este parțial calculabilă.

2. Macroinstructiuni. Exemple de funcții calculabile

Macroinstructiunile sunt abrevieri pentru secvențe de instrucțiuni 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 distincte). De asemenea trebuie respectată regula ca valoarea variabilelor care apar în macroinstructiune, cu excepția celor care apar în membrul stâng al unei atribuiri, 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 L
```

2) Macroinstructiunea $v \leftarrow 0$ 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:

 $v \leftarrow 0$
 $v \leftarrow v + 1$

...

 $v \leftarrow v + 1$

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

Exemplul 3. Funcția $f : \mathbf{N} \rightarrow \mathbf{N}$ dată de $f(x) = x$ este calculabilă.

Pentru demonstrație, vom scrie următorul 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] $x \leftarrow x - 1$

$y \leftarrow y + 1$

$z \leftarrow z + 1$

goto A

[C] if $z \neq 0$ **goto** D

goto E

[D] $z \leftarrow z - 1$

$x \leftarrow x + 1$

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:

$v \leftarrow 0$

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

```

goto C
[B]   u ← u - 1
      v ← v + 1
      z ← z + 1
goto A
[C]   if z ≠ 0 goto D
goto E
[D]   z ← z - 1
      u ← u + 1
goto C
[E]   v ← v

```

care "transcrie" funcția din exemplul 4. Se impun următoarele trei remarci:

- nu am inițializat pe z cu 0, deoarece este clar că după orice executare a macroinstructiunii, valoarea sa va fi 0;
- apare necesitatea instrucțiunii $v \leftarrow v$;
- 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.

Un program care calculează această funcție este următorul:

```

 $z \leftarrow x_2$ 

```

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

goto E

$[A]$ $z \leftarrow z - 1$

$y \leftarrow y + x_1$

goto B

iar macroinstructiunea $z \leftarrow z_1 \times z_2$ 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 $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ 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:

$y \leftarrow x_1$

$z \leftarrow x_2$

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

goto E

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

goto A $\{y = 0 \text{ și } z \neq 0\}$

[B] $y \leftarrow y - 1$

$z \leftarrow z - 1$

goto C

Macroinstructiunea asociata este $z \leftarrow z_1 - z_2$ și are următoarea dezvoltare posibilă:

$u_1 \leftarrow z_1$

$u_2 \leftarrow z_2$

[C] **if** $u_2 \neq 0$ **goto** A

goto E

[A] **if** $u_1 \neq 0$ **goto** B

goto A

[B] $u_1 \leftarrow u_1 - 1$

$u_2 \leftarrow u_2 - 1$

```
  goto C  
[E]    z ← u1
```

Calculabilitate

Dorim să arătăm că există probleme **nedecidabile**, adică probleme pentru care nu pot fi elaborați algoritmi.

Pentru aceasta trebuie precizat *ce este un algoritm*.

Un algoritm poate fi specificat:

- fie printr-un *program într-un limbaj de programare*
- fie printr-o *mașină matematică*.

Pe de altă parte, abordările se pot referi:

- fie la lucrul cu numere naturale
- fie la lucrul cu siruri de caractere peste un alfabet dat.

Studiul diferitelor abordări a arătat că ele sunt echivalente!

Plan de lucru:

- 1) prezentăm un limbaj de programare, numit S pentru lucrul cu numere și arătăm că folosindu-l putem efectua “toate” calculele aritmetice uzuale.
- 2) Arătăm că există o bijecție între programele din S și mulțimea numerelor naturale. Aceasta va permite ca un program (din S) să aibă drept argument și un program din S .

- 3) Prezentăm o problemă nedecidabilă și anume *problema opririi programelor*: “Nu există algoritm (program în limbajul S) care pentru orice (program P în S , date de intrare) să decidă dacă programul P se termină sau nu”.
- 4) Prezentăm alte abordări ale noțiunii de algoritm: *limbajul pe cuvinte S_n* , *mașinile Post-Turing* și *Turing* etc., demonstrând echivalența lor.

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.

Înainte de a prezenta instrucțiunile limbajului S , precizăm că ele pot fi precedate, optional, 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 ulterior.

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:

$$^{[A]} \quad x \leftarrow x - 1$$

$$y \leftarrow y + 1$$

$$\text{if } x \neq 0 \text{ goto } A$$

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

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

Exemplul 2. Programul:

^[A] $x \leftarrow x + 1$

if $x \neq 0$ **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.*

Expresii aritmetice. Sume

Ne vom ocupa de expresiile aritmetice în care intervin operatorii binari + și -, dar și cuprinderea între paranteze rotunde.

Pentru simplificare, vom presupune că variabilele care apar în expresii sunt litere.

Scopul propus constă în evaluarea expresiilor (presupunând că variabilele au valori stabilite oarecare).

Stim că:

- o *expresie* este o "sumă" de termeni (între care intervin doar operatorii + și -);
- un *termen* este fie o variabilă, fie o expresie între paranteze

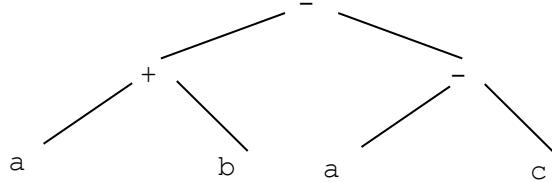
Gramatica independentă de context corespunzătoare este:

$E \rightarrow T \mid T+E \mid T-E$

$T \rightarrow \text{literă} \mid (E)$

Primul pas în rezolvarea problemelor enunțate constă în construcția arborelui binar atașat expresiei.

Exemplu. Expresiei aritmetice $((a+b)-(a-c))$ îi corespunde următorul arbore:



Construcția arborelui atașat urmează întocmai definiția expresiilor aritmetice: va fi scrisă câte o metodă pentru expresie și termen.

Este clar că pentru evaluarea expresiei, acest arbore trebuie parcurs în postordine; evident, trebuie cunoscute valorile variabilelor ce intervin în expresie.

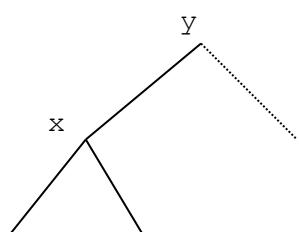
Fiecare vârf (obiect de tipul clasei varf) are câmpurile st, dr și et desemnând respectiv descendental stâng, cel drept și eticheta sa.

Se presupune că variabilele sunt litere mici de la începutul alfabetului și până la o literă last citită de la intrare. Deci numai pentru literele de la 'a' până la 'last' vor fi citite valorile lor.

Expresia este citită în sirul de caractere ea. Mai este folosită o variabilă ch care joacă rolul unui "spion" către caracterul următor din expresie.

ea	.	.	ch	.	.	@
poz						

Vom folosi asociativitatea la stânga a operatorilor. De exemplu dacă un termen începe cu o "sumă" de termeni pentru care am construit arborele de rădăcină x și urmează un termen pentru care am construit obiectul y, x va deveni descendental stâng al lui y, care va fi rădăcina noului arbore corespunzător termenului:




```
import java.util.*;  
  
class Sume {  
  
    public static void main(String[] s) {  
  
        varf Ob = new varf();  
  
        varf.rad = Ob.expresie();  
  
        System.out.println("Val. expresiei: " +  
                           Ob.valoare(varf.rad));  
  
    }  
  
}  
  
class varf {  
  
    static varf rad; static char ch,last;  
  
    static double[] val; char et; varf st,dr;  
  
    static String ea; static int poz;  
  
    static Scanner sc = new Scanner(System.in);  
  
    varf() {  
  
        System.out.print("Expresia : ");  
  
        ea = sc.next(); ch = ea.charAt(poz);  
  
        System.out.print("Ultima litera: ");  
  
        last = sc.next().charAt(0);  
  
        val = new double[last-'a'+1];  
  
        for (char c='a'; c<=last; c++) {  
  
            System.out.print(c + "= "); val[c-'a'] = sc.nextDouble();  
        }  
    }  
}
```

```

varf(char e) { et = e; }

varf expresie() {
    varf x,y;
    x = termen();
    while ( (ch=='+') || (ch=='-') ) {
        y = new varf(ch); y.st = x;
        ch = ea.charAt(++poz);
        y.dr = termen(); x=y;
    }
    return x;
}

varf termen() {
    varf x;
    if ( ch != '(' ) {
        x=new varf(ch); ch = ea.charAt(++poz);
    }
    else {
        ch = ea.charAt(++poz); x = expresie(); ch = ea.charAt(++poz);
    }
    return x;
}

double valoare(varf x) {
    if (x.st == null) return val[x.et - 'a'];
    else if (x.et == '+') return valoare(x.st) + valoare( x.dr);
    else if (x.et == '-') return valoare(x.st) - valoare( x.dr);
}

```

$E \rightarrow T \mid T+E \mid T-E$

$T \rightarrow \text{literă} \mid (E)$

```
else return 9999;  
}  
}
```