Limbaje formale

Mircea Drăgan

May 25, 2010

Cuprins

1 Limbaje formale **3**
- 1.1 Limbaje, gramatici, proprietăţi generale . . . 3
- 1.2 Ierarhia Chomsky . . . 10
- 1.3 Proprietăţi de închidere a familiilor Chomsky . . . 14
- 1.4 Problema compilării . . . 19
- 1.5 Probleme propuse . . . 22

2 Limbaje regulate **25**
- 2.1 Automate finite şi limbaje regulate . . . 25
- 2.2 Proprietăţi speciale ale limbajelor regulate . . . 31
- 2.3 Lema de pompare pentru limbaje regulate . . . 34
- 2.4 Expresii regulate . . . 35
- 2.5 Sisteme tranziţionale . . . 37
- 2.6 Analiza lexicală . . . 39
- 2.7 Probleme propuse . . . 50

3 Limbaje independente de context **53**
- 3.1 Arbori de derivare . . . 53
- 3.2 Decidabilitate şi ambiguitate în familia $\mathcal{L}_2$. . . . 56
- 3.3 Forme normale pentru gramatici de tipul 2 . . . 59
- 3.4 Automate push-down (APD) . . . 65
- 3.5 Automate push–down deterministe . . . 73
- 3.6 Lema Bar–Hillel . . . 76
- 3.7 Închiderea familiei $\mathcal{L}_2$ la substituţii . . . 80
- 3.8 Caracterizarea limbajelor independente de context . . . 84

4 Limbaje dependente de context **91**
- 4.1 Gramatici monotone . . . 91
- 4.2 Automate liniar mărginite . . . 95

5 Limbaje de tipul zero **101**
- 5.1 Forma normală . . . 101
- 5.2 Maşina Turing . . . 102

Capitolul 1

Limbaje formale

1.1 Limbaje, gramatici, proprietăţi generale

Noţiunea generală de limbaj. Vom considera o mulţime finită şi nevidă V, numită *alfabet* sau *vocabular*. Elementele mulţimii le vom numi *simboluri* (sau litere, caractere, variabile).

Definiţie 1.1 *Un cuvânt peste un alfabet V este un şir $p = a_1 a_2 \dots a_n$, $a_i \in V, i = \overline{1,n}$.*

Numărul n, deci numărul simbolurilor cuvântului p, se numeşte *lungimea cuvântului* şi va fi notat cu $|p|$ sau $l(p)$. Vom considera şi *cuvântul vid* λ sau e, care nu conţine nici un simbol; evident $|\lambda| = 0$. Un rol deosebit îl are simbolul *blanc* notat cu $\flat$(bnu coincide cu λ) precum şi simbolul marcaj notat # pentru delimitarea cuvintelor. Noţiunea de "cuvânt" este fundamentală în teoria limbajelor formale sau în alte domenii ale informaticii; termeni sinonimi utilizaţi în literatură sunt propoziţie, frază sau şir. Să observăm că nu există o similitudine foarte bună între noţiunile de "alfabet", "cuvânt", etc. din teoria limbajelor formale şi noţiunile corespunzătoare din lingvistică.

Mulţimea tuturor cuvintelor peste un alfabet V o notăm cu V^+. Această mulţime împreună cu cuvântul vid va fi notată cu V^*.

În general, vom utiliza litere mari de la sfârşitul alfabetului pentru notarea diverselor alfabete, U, V, W, etc.; litere de la începutul alfabetului (mari sau mici) pentru notarea simbolurilor, $A, B, C, \dots, a, b, c, \dots, i, j, \dots$ (uneori cifre 0,1,2,...); pentru notarea cuvintelor vom utiliza litere mici de la sfârşitul alfabetului, $p, q, r, s, t, u, v, w, x, y, z$, (această convenţie de notare nu va fi absolută).

Fie $p = a_1 \dots a_n$, $q = b_1 \dots b_m$. Definim pe mulţimea V^* operaţia de concatenare sau juxtapunere prin $pq = a_1 \dots a_n b_1 \dots b_m$.

Se poate verifica uşor că această operaţie este asociativă. Prin urmare, mulţimea V^+ înzestrată cu această operaţie este un semigrup (*semigrupul liber peste V*). Mulţimea V^* cu aceiaşi operaţie este un semigrup cu unitate, deci un monoid, unitatea fiind cuvântul vid (*monoidul liber* peste V).

Fie din nou $p, q \in V^*$. Vom spune că q este un subcuvât sau un *infix* (eventual propriu) al lui p dacă $p = uqv,\ u, v \in V^*$ (eventual $u, v \in V^+$); q este *prefix* (propriu) al lui p dacă $p = qv, v \in V^*(v \in V^+)$; q este *sufix* (propriu) al lui p dacă $p = uq, u \in V^*(u \in V^+)$.

Definiţie 1.2 *Un* **limbaj** L *peste un alfabet* V *este o parte a monoidului liber* V^**, deci* $L \subseteq V^*$*.*

Să observăm că V^* (sau V^+) este întotdeauna o mulţime infinită (evident numărabilă); în această accepţiune generală, un limbaj poate să fie o mulţime finită sau infinită, uneori chiar vidă.

Exemplu. Fie $V = \{0, 1\}$. Avem

$$V^+ = \{0, 1, 00, 01, 10, 000, \ldots\},$$

$$V^* = \{\lambda, 0, 1, 00, 01, 10, 000, \ldots\}.$$

Limbaje peste alfabetul V sunt de exemplu mulţimile

$$L_1 = \{\lambda, 00, 11\},$$

$$L_2 = \{1, 11, 111, \ldots\} = \{1^n | n \geq 1\}.$$

Observaţie. Notaţia a^n, unde a este un simbol al unui alfabet, înseamnă cuvântul constituit din n simboluri a, adică $\underbrace{aa\ldots a}$. În particular $a^0 = \lambda$.

Operaţii cu limbaje. Limbajele fiind mulţimi se pot efectua cu limbaje operaţiile obişnuite cu mulţimi: reuniune, intersecţie, diferenţă, complementariere (faţă de V^*). Există şi operaţii specifice limbajelor:

- *Produsul* (concatenarea) a două limbaje definit prin

$$L_1 L_2 = \{pq | p \in L_1, q \in L_2\}.$$

  Dacă $L_1 = L_2 = L$ vom nota $LL = L^2$. Prin recurenţă, se defineşte L^n astfel

$$L^0 = \{\lambda\}, L^k = L^{k-1}L, k \geq 1.$$

- *Închiderea (Kleene)* a unui limbaj L este

$$L^* = \bigcup_{k=0}^{\infty} L^k.$$

  În general, o operaţie de n-aritate oarecare (cel mai adesea binară sau unară) pe mulţimea V^* defineşte o operaţie corespunzătoare pe mulţimea limbajelor. Astfel, dacă

$$\alpha : V^* \longrightarrow \mathcal{P}(V^*) \text{ şi } \beta : V^* \times V^* \longrightarrow \mathcal{P}(V^*)$$

sunt două operaţii pe V^* (unară şi respectiv binară) şi L_1, L_2 sunt două limbaje peste V, putem defini limbajele $\alpha(L_1)$ respectiv $\beta(L_1, L_2)$ prin

$$\alpha(L_1) = \bigcup_{x \in L_1} \alpha(x), \quad \beta(L_1, L_2) = \bigcup_{x \in L_1, y \in L_2} \beta(x, y).$$

Exemple:

1. Limbajul $Sub(L)$. Fie V^* şi $Sub(x)$ mulţimea tuturor subcuvintelor lui x (evident Sub este o operaţie unară pe V^*). Dacă L este un limbaj peste V, putem defini limbajul

$$Sub(L) = \bigcup_{x \in L} Sub(x).$$

adică limbajul constituit din toate subcuvintele tututor cuvintelor lui L. Semnificaţii analoage vor avea şi limbajele $Pref(L)$ şi $Suf(L)$.

2. Limbajul $Mi(L)$. Fie $x = a_1 \dots a_n$ un cuvânt peste alfabetul V. Cuvântul $Mi(x) = a_n \dots a_1$ se numeşte răsturnatul lui x (Mi este prescurtarea cuvântului englez *mirror*). Se mai notează $Mi(x) = \tilde{x}$. Avem atunci şi răsturnatul unui limbaj

$$Mi(L) = \bigcup_{x \in L} Mi(x).$$

3. Operaţia de *substituţie*. Fie U şi V două alfabete şi fie aplicaţia $s : V \longrightarrow \mathcal{P}(U^*)$. Extindem (prelungim) această aplicaţie la V^* prin

$$s(\lambda) = \{\lambda\}, s(xy) = s(x)s(y), x, y \in V^*.$$

O astfel de prelungire se numeşte *canonică*; ea păstrează operaţia de concatenare, în sensul că dacă $p = xy$, atunci $s(p) = s(x)s(y)$ este concatenarea limbajelor $s(x), s(y)$. Operaţia de substituţie a limbajelor este dată de

$$s(L) = \bigcup_{x \in L} s(x).$$

Să observăm că această operaţie transformă un limbaj peste un alfabet V într-un limbaj peste un alfabet U şi că păstrează operaţia de concatenare. Dacă $card\ s(a) < \infty, \forall a \in V$, vom spune că substituţia este *finită*, iar dacă $card\ s(a) = 1, \forall a \in V$ vom spune că s este un *homomorfism*.

Gramatici generative de tip Chomsky. Un limbaj peste un alfabet poate să fie o mulţime finită sau infinită. Dacă este o mulţime finită, el poate fi definit

prin scrierea efectivă a cuvintelor limbajului. În cazul în care este o mulţime infinită, el poate fi definit în anumite cazuri punând în evidenţă structura cuvintelor lui. De exemplu

$$L_2 = \{01, 0011, 000111, \ldots\} = \{0^n 1^n | n \geq 1\}.$$

Există două procedee mai generale pentru definirea limbajelor:

1. Procedee *generative*, care permit generarea tuturor cuvintelor limbajului. Există mai multe tipuri de mecanisme de generare a limbajelor, între care gramaticile Chomsky, sisteme Lindenmayer,etc.

2. Procedee *analitice*, care determina dacă un cuvânt dat aparţine sau nu limbajului. Sunt aşa-numitele *automate, automate finite, automate push-down*, etc.

Un rol deosebit în teoria limbajelor formale îl au gramaticile Chomsky. Fie V_N şi V_T două alfabete disjuncte, numite respectiv alfabetul simbolurilor neterminale (V_N) şi alfabetul simbolurilor terminale (V_T). Notăm $V_G = V_N \cup V_T$ (alfabetul general) şi fie $P \subseteq V_G^* V_N V_G^* \times V_G^*$. Mulţimea P va fi deci formată din perechi de forma (u, v), unde $u = u'Au''$, $u', u'' \in V_G^*$, $A \in V_N$ iar $v \in V_G^*$, deci u şi v sunt cuvinte peste V_G, cu observaţia că u trebuie să conţină cel puţin un simbol neterminal. Vom spune că o astfel de pereche este o **regulă** (producţie, regulă de generare, regulă de rescriere) şi o vom nota $u \to v$ (vom spune: u se transformă în v). Apartenenţa unei reguli la P o vom nota în mod obişnuit $(u \to v) \in P$, sau mai simplu, $u \to v \in P$ (nu va exista confuzia cu faptul că v este un element al lui P).

Definiţie 1.3 *O gramatică (Chomsky) G este un sistem $G = (V_N, V_T, S, P)$, unde V_N este alfabetul simbolurilor neterminale, V_T este alfabetul simbolurilor terminale, $S \in V_N$ şi se numeşte simbolul de start al gramaticii, iar P este mulţimea de reguli.*

Observaţie. Simbolurile alfabetului V_N le vom nota în general cu litere mari $A, B, C, \ldots, X, Y, Z$ (mai puţin U, V, W) iar cele ale alfabetului V_T cu litere mici de la început $a, b, c, \ldots$ sau cu cifre $0, 1, 2, \ldots$.

Fie G o gramatică şi $p, q \in V_G^*$. Vom spune că p *se transformă direct în* q şi vom scrie $p \underset{G}{\Rightarrow} q$ (sau $p \Rightarrow q$ dacă nu există nici o confuzie) dacă există $r, s, u, v \in V_G^*$ astfel încît $p = rus, q = rvs$ iar $u \to v \in P$. Vom spune ca p' *se transformă în* p'' (fără specificaţia direct) dacă există $p_1, p_2, \ldots, p_n$, $n \geq 1$ astfel încît

$$p' = p_1 \underset{G}{\Rightarrow} p_2 \underset{G}{\Rightarrow} \ldots \underset{G}{\Rightarrow} p_n = p''.$$

Vom scrie $p' \overset{+}{\underset{G}{\Rightarrow}} p''$ (sau $p' \overset{+}{\Rightarrow} p''$ când nu există nici o confuzie) dacă $n > 1$ şi $p' \overset{*}{\underset{G}{\Rightarrow}} p''$ (sau $p' \overset{*}{\underset{G}{\Rightarrow}} p''$) dacă $n \geq 1$. Şirul de mai sus va fi numit **derivare** iar numărul de derivări directe din şir îl vom numi **lungimea** derivării; se mai spune că p' derivă în p''.

Să observăm că transformările astfel definite $\Rightarrow$, $\overset{+}{\Rightarrow}$, $\overset{*}{\Rightarrow}$ sunt relaţii pe V_G^*. Este clar că relaţia $\overset{+}{\Rightarrow}$ este închiderea tranzitivă a relaţiei $\Rightarrow$, iar relaţia $\overset{*}{\Rightarrow}$ este închiderea tranzitivă şi reflexivă a relaţiei de transformare directă .

Definiţie 1.4 . *Limbajul generat de gramatica G este prin definiţie limbajul*

$$L(G) = \{p |\ p \in V_T^*, S \overset{*}{\underset{G}{\Rightarrow}} p\}.$$

Observaţie. Dacă $p \in V_G^*$ şi $S \overset{*}{\underset{G}{\Rightarrow}} p$ se spune că p este o *formă propoziţională* în gramatica G.

Exemple:

1. Fie $G = (V_N, V_T, S, P)$, unde $V_N = \{A\}$, $V_T = \{0, 1\}$, $S = A$ (evident) şi $P = \{A \to 0A1, A \to 01\}$. O derivare în această gramatică este, de exemplu
$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000111 = 0^3 1^3.$$
Este evident că $L(G) = \{0^n 1^n | n \geq 1\}$.
Observaţie. În cazul în care mai multe reguli au aceeaşi parte stângă, le vom scrie compact astfel $u \to v_1|v_2|\ldots|v_n$, simbolul | având sensul de sau; în cazul nostru, $A \to 0A1|01$.

2. $G = (V_N, V_T, S, P)$, unde
$V_N = \{$<propoziţie>, <subiect>, <atribut>, <predicat>, <complement>, <substantiv>, <adjectiv>, <verb>, <articol>$\}$,
$V_T = \{$o, orice, matrice, funcţie, derivabilă, continuă, este $\}$,
$S =$<propoziţie>,
$P = \{$<propoziţie>→<subiect><atribut><predicat><complement>,
<subiect>→<articol><substantiv>,
<atribut>→<adjectiv>,
<predicat>→<verb>,
<complement>→<adjectiv>,
<articol>→o|orice,
<substantiv>→matrice|funcţie,
<adjectiv>→derivabilă|continuă,
<verb>→este.
Observaţie. În acest exemplu, "<propoziţie>", "<subiect>", etc., reprezintă fiecare câte un simbol neterminal; de asemenea, "o", "orice", "matrice",

etc., reprezintă simboluri terminale.
Se poate uşor observa că această gramatică generează propoziţii simple de forma subiect-atribut-predicat-complement care exprimă judecăţi asupra conceptelor de "matrice" şi "funcţie". De exemplu, se poate forma propoziţia: "orice funcţie derivabilă este continuă", care este din punct de vedere semantic corectă, precum şi propoziţia "orice funcţie continuă este derivabilă", care, după cum se ştie, este falsă. Evident, se pot genera şi numeroase propoziţii care nu au sens. Ceea ce ne interesează în acest moment este aspectul formal, deci din punct de vedere sintactic toate aceste propoziţii sunt corecte; semantic, unele propoziţii pot să fie incorecte sau chiar să nu aibe sens.
Să mai observăm că o gramatică Chomsky este în măsură să constituie un model matematic pentu sintaxa unei limbi, fără să intereseze aspectele semantice. Este ceea ce a încercat să întreprindă Naom Chomsky pentru limba engleză în lucrările sale din anii 1957.

3. $G = (V_N, V_T, A_0, P)$, unde
V_N = {<program>, <instrucţie>, <atribuire>, <if>, <expresie>, <termen>, <factor>, <variabilă>, <index>},
V_T = {begin, end, if, then, stop, t, i, +, *, (,), =, ,, ; },
$A0$ =<program>
P = {<program>→begin <linie> end
<linie>→<linie>;<instrucţie> | <instrucţie>
<instrucţie>→<atribuire> | <if> | stop
<atribuire>→<variabilă>=<expresie>
<if>→if(<expresie>) then <atribuire>
<expresie>→<expresie> + <termen> | <termen>
<termen>→<termen>*<factor> | <factor>
<factor>→(<expresie>)| <variabilă>
<variabilă>→t(<index>)|i
<index>→<index>, <expresie> | <expresie>
Gramatica din acest exemplu defineşte un limbaj de programare simplu cu trei tipuri de instrucţii: atribuiri, if-then, stop. Expresiile aritmetice au numai operatorii + şi * iar variabilele pot fi simple sau indexate (tablouri). Menţionăm că definirea în acest mod a unui limbaj, inclusiv utilizarea croşetelor pentru desemnarea simbolurilor neterminale, poartă adesea denumirea de notaţie Backus Naur; în acest mod s-a definit limbajul ALGOL.

Tipuri de gramatici. După forma regulilor de generare, gramaticile Chomsky se împart în mai multe tipuri; clasificarea obişnuită este următoarea:

- *Gramatici de tipul 0*; sunt gramatici fără restricţii asupra regulilor;

- *Gramatici de tipul 1 (dependente de context)*; sunt gramatici care au reguli de forma

$$uAv \to upv,\ u,p,v \in V_G^*,\ p \neq \lambda,\ A \in V_N$$

sau $A \to \lambda$ şi în acest caz $A \in V_N$ nu apare în dreapta vreunei reguli.
Observaţie. Evident, regulile de forma a doua au sens numai dacă A este simbolul de start.

- *Gramaticile de tipul 2 (independente de context)*; sunt gramatici care au reguli de forma

$$A \to p,\ A \in V_N,\ p \in V_G^*.$$

- *Gramaticile de tipul 3 (regulate)*; sunt gramatici care au reguli de forma

$$\left\{ \begin{array}{l} A \to Bp \\ C \to q \end{array} \right. \ sau \left\{ \begin{array}{l} A \to pB \\ C \to q \end{array} \right.$$

cu $A, B, C \in V_N$ şi $p, q \in V_T^*$.

Vom nota cu $L_j,\ j = 0,1,2,3$ familiile de limbaje generate de gramaticile de tipurile $j = 0,1,2,3$; vom avea astfel limbaje de tipul 0, limbaje de tipul 1 (sau *dependente de context*) , limbaje de tipul 2 (sau *independente de context*) şi limbaje de tipul 3 (sau *regulate*). Să observăm că este importantă structura cuvintelor unui limbaj şi nu modul în care sunt notate simbolurile terminale. De exemplu , limbajele

$$L_2' = \{0^n 1^n | n \geq 1\},\ L_2'' = \{a^n b^n | n \geq 1\}$$

sunt în mod practic identice. Putem utiliza o unică notaţie pentru alfabetul simbolurilor terminale , de exemplu , $V_T = \{i_1, \ldots, i_n\}$. Clasificarea de mai sus este fundamentală în teoria limbajelor formale, ea a fost introdusă de Naom Chomsky în 1958 şi prin tradiţie noile clase de limbaje sunt raportate la această clasificare. O altă clasificare este următoarea

- *Gramatici de tipul 0*; fără restricţii;

- *Gramatici monotone*:

$$u \to v,\ |u| \leq |v|,\ u, v \in V_G^*;$$

- *Gramatici dependente de context (de tipul 1)*:

$$uAv \to upv,\ u,p,v \in V_G^*,\ p \neq \lambda,\ A \in V_N;$$

- *Gramatici independente de context (de tipul 2)*:

$$A \to p,\ A \in V_N,\ p \in V_G^*;$$

- *Gramatici liniare*:

$$\begin{cases} A \to uBv \\ C \to p \end{cases} \quad A, B, C \in V_N,\ u, v, p \in V_T^*;$$

- *Gramatici stâng (drept) liniare*:

$$\begin{cases} A \to uB\ (A \to Bv) \\ C \to p \end{cases} \quad A, B, C \in V_N,\ u, v, p \in V_T^*;$$

- *Gramatici regulate (de tipul 3)*; gramatici stâng liniare sau gramatici drept liniare.

Gramaticile monotone ca şi cele dependente de context nu pot avea reguli cu partea dreaptă vidă. Se introduce următoarea convenţie de completare: *într-o gramatică monotonă sau dependentă de context se admite o regula de forma $A \to \lambda$ cu condiţia ca A să nu apară în partea dreaptă a vreunei reguli.* După cum vom vedea , existenţa sau inexistenţa regulilor de forma $A \to \lambda$, reguli numite de ştergere, poate modifica esenţial puterea generativă a unei gramatici. O gramatică în care nu avem astfel de reguli o vom numi uneori λ-liberă; de asemenea, un limbaj care nu conţine cuvântul vid, îl vom numi λ-liber. Să mai observăm că existenţa regulilor de ştergere într-o gramatică nu implică în mod necesar existenţa cuvântului vid în limbajul generat.

Două gramatici care generează acelaşi limbaj se numesc **echivalente**.

Gramaticile monotone şi gramaticile dependente de context sunt echivalente; de asemenea, gramaticile drept şi stâng liniare sunt echivalente, justificându-se astfel clasa gramaticilor regulate.

1.2 Ierarhia Chomsky

Lemele care urmează vor avea o utilizare frecventă în cele ce urmează. **Lema de localizare a grama**

Lema 1.1 *Fie G o gramatica independentă de context şi fie derivarea*

$$X_1 \dots X_m \overset{*}{\Rightarrow} p, X_j \in V_G, j = \overline{1, m},\ p \in V_G^*.$$

Atunci există $p_1, p_2, \dots, p_m \in V_G^*$ astfel încât $p = p_1 \dots p_m$ şi $X_j \overset{*}{\Rightarrow} p_j,\ j = \overline{1, m}$.

Demonstraţie. Procedăm prin inducţie asupra lungimii derivării l.
Dacă $l = 0$ atunci $p = X_1 \dots X_m$ şi luăm $p_j = X_j$.
Presupunem că proprietatea este adevărată pentru derivări de lungime l şi fie o derivare de lungime $l+1, X_1 \dots X_m \overset{*}{\Rightarrow} p$. Punem în evidenţă ultima derivare directă $X_1 \dots X_m \overset{*}{\Rightarrow} q \Rightarrow p$. Conform ipotezei inductive, $q = q_1 \dots q_m$ şi $X_j \overset{*}{\Rightarrow} q_j,\ j = \overline{1,m}$.

Fie apoi $A \to u$ regula care se aplică în derivarea directă $q \Rightarrow p$ şi să presupunem că A intră în subcuvântul q_k, deci $q_k = q_k' A q_k''$. Vom lua

$$p_j = \begin{cases} q_j & , j \neq k \\ q_k' u q_k'' & , j = k \end{cases}$$

Este evident că $X_j \overset{*}{\Rightarrow} p_j,\ j \neq k$, iar pentru $j = k$ avem

$$X_k \overset{*}{\Rightarrow} q_k = q_k' A q_k'' \Rightarrow q_k' u q_k'' = p_k. \square$$

Vom pune în evidenţă în continuare o proprietate asupra structurii regulilor gramaticilor Chomsky. Partea dreaptă a unei reguli, pentru toate tipurile de gramatici, este un cuvânt format din terminale sau neterminale. Este convenabil de multe ori ca partea dreaptă a regulilor să conţină un singur tip de simboluri, terminale sau neterminale. Acest lucru este posibil fără modificarea tipului gramaticii.

Lema 1.2 *Fie $G = (V_N, V_T, S, P)$ o gramatică de tipul 2. Există o gramatică G' echivalentă, de acelaşi tip, cu proprietatea că dacă o regulă are în partea dreaptă un terminal, atunci ea este de forma $A \to i, A \in V_N, i \in V_T$.*

Demonstraţie. Luăm gramatica G' de forma $G' = (V_N', V_T', S, P')$ unde V_N' şi P' se construiesc astfel: $V_N \subseteq V_N'$ şi includem în P' toate regulile din P care convin. Fie acum o regulă din P care nu convine

$$u \to v_1 i_1 v_2 i_2 \dots i_n v_{n+1},\ i_k \in V_T,\ v_k \in V_N^*.$$

Vom introduce în P' următoarele reguli:

$$u \to v_1 X_{i_1} v_2 X_{i_2} \dots X_{i_n} v_{n+1},\ \ X_{i_k} \to i_k,\ k = \overline{1,n},$$

unde X_{i_k} sunt neterminale noi pe care le adăugăm la V_N'. Este evident că G' păstrează tipul lui G şi că $L(G') = L(G). \square$
Ierarhia Chomsky. Este evident că $\mathcal{L}_3 \subseteq \mathcal{L}_2$ şi că $\mathcal{L}_1 \subseteq \mathcal{L}_0$, deoarece orice regulă a unei gramatici de tipul 3 respectă prescripţiile unei gramatici de tipul 2; analog pentru familiile $\mathcal{L}_1$ şi $\mathcal{L}_2$. Aparent, o regulă de forma $A \to p$ (de tipul 2) este un caz particular a unei reguli de forma $uAv \to upv$ (de tipul 1), pentru $u = v = \lambda$; totuşi, realitatea nu este aceasta, deoarece la tipul 2 de gramatici

sunt permise reguli de ştergere, pe când la tipul 1 se impune condiţia $p \neq \lambda$. Vom arăta că avem $\mathcal{L}_2 \subseteq \mathcal{L}_1$.

Şirul de incluziuni

$$\mathcal{L}_3 \subseteq \mathcal{L}_2 \subseteq \mathcal{L}_1 \subseteq \mathcal{L}_0$$

poartă denumirea de ierarhia Chomsky (vom arăta pe parcursul acestui curs că incluziunile sunt stricte). Această ierarhie caracterizează puterea generativă a celor patru tipuri de gramatici, această putere fiind crescătoare de la 3 la 0. Orice alte mecanisme generative se raportează la această ierarhie fundamentală. Vom demonstra mai întâi următoarea lemă.

Lema 1.3 *(*Lema eliminării regulilor de ştergere*). Orice limbaj independent de context λ-liber poate fi generat de o gramatică de tipul 2 fără reguli de ştergere.*

Demonstraţie. Fie $G = (V_N, V_T, S, P)$ o gramatică independentă de context şi $L(G)$ limbajul generat. Prin ipoteză $\lambda \notin L(G)$. Definim prin recurenţă şirul de mulţimi $\{U_k\}_{k \in N}$ astfel:

$$U_1 = \{X | X \in V_N, X \to \lambda \in P\}$$
$$U_{k+1} = \{X | X \in V_N, X \to p \in P, p \in U_k^*\}.$$

Să observăm că $U_1 \subseteq U_2 \subseteq \ldots \subseteq U_k \ldots$. Într-adevăr, $U_1 \subseteq U_2$ deoarece dacă $X \in U_1$ rezultă că $X \to \lambda \in P$ şi cum $\lambda \in U_1^*$ rezultă că $X \in U_2$. Dacă acum $U_{k-1} \subseteq U_k$, atunci evident $U_{k-1}^* \subseteq U_k^*$ şi dacă $X \in U_k$, deci $X \to p$, $p \in U_{k-1}^* \subseteq U_k^*$, ceea ce înseamnă că $X \in U_{k+1}$, adică $U_k \subseteq U_{k+1}$.

Cum toate aceste mulţimi sunt incluse în V_N şi V_N este finită, rezultă că există o mulţime finală U_f astfel astfel încât

$$U_1 \subseteq U_2 \subseteq \cdots \subseteq U_f = U_{f+1} = \cdots.$$

Are loc de asemenea şi implicaţia $X \in U_f \Leftrightarrow X \overset{*}{\Rightarrow} \lambda$.

Vom ilusta această implicaţie cu un exemplu. Să presupunem că $U_f = U_3$ şi fie $X \in U_3$. Atunci în mod necesar trebuie să avem $X \to p \in P$, $p \in U_2^*$; de exemplu $p = X_1X_2$ şi $X_1, X_2 \in U_2$. În mod analog

$$\begin{cases} X_1 \to Y_1Y_2Y_3 \\ X_2 \to Z_1Z_2 \end{cases}, \; Y_1, Y_2, Y_3, Z_1, Z_2 \in U_1,$$

prin urmare $Y_1 \to \lambda$, $Y_2 \to \lambda$, $Y_3 \to \lambda$, $Z_1 \to \lambda$, $Z_2 \to \lambda$. Putem scrie derivarea

$$X \Rightarrow X_1X_2 \overset{*}{\Rightarrow} Y_1Y_2Y_3Z_1Z_2 \overset{*}{\Rightarrow} \lambda.$$

Definim acum următoarea gramatică independentă de context fără reguli de stergere $G' = (V_N, V_T, S, P')$ unde V_N, V_T, S sunt ca în gramatica dată, iar P' se construieşte pornind de la P astfel. Fie $X \to p \in P, p \neq \lambda$. Includem atunci în

P' această regulă precum şi toate regulile de forma $X \to p_j$, unde p_j se obţine din p lăsând la o parte, în toate modurile posibile, simbolurile din U_f. De exemplu dacă $X \to ABC \in P$ şi $A, B \in U_f$, vom induce în P' regulile

$$X \to ABC,\ X \to BC,\ X \to AC,\ X \to C.$$

Să observăm că în acest fel mulţimea P a fost pe de o parte micşorată (au fost excluse regulile de ştergere), iar pe de altă parte îmbogăţită cu eventualele noi reguli. Să mai obsevăm că G' este independentă de context şi că nu conţine reguli de ştergere.

Vom arăta că $L(G) = L(G')$.

Mai întâi, să arătăm că $L(G) \subseteq L(G')$.

Fie $p \in L(G)$, deci $S \underset{G}{\overset{*}{\Rightarrow}} p$; vom arăta că $S \underset{G'}{\overset{*}{\Rightarrow}} p$.

Vom arăta o implicaţie ceva mai generală, $X \underset{G}{\overset{*}{\Rightarrow}} p$ implică $X \underset{G'}{\overset{*}{\Rightarrow}} p$, pentru orice $X \in V_N$ (relaţia cerută se obţine pentru $X = S$). Procedăm prin inducţie asupra lungimii derivării l. Dacă $l = 1$ avem

$$X \underset{G}{\Rightarrow} p,\ X \to p \in P,\ p \neq \lambda \ \Rightarrow\ X \to p \in P'$$

şi deci $X \underset{G'}{\overset{*}{\Rightarrow}} p$.

Presupunem că afirmaţia este adevărată pentru $l = n$ şi luăm o derivare cu lungimea $l = n + 1$. Punem în evidenţă prima derivare directă

$$X \underset{G}{\Rightarrow} X_1 \dots X_m \underset{G}{\overset{*}{\Rightarrow}} p$$

Conform lemei de localizare avem $p = p_1 \dots p_m$ şi $X_j \underset{G}{\overset{*}{\Rightarrow}} p_j,\ j = \overline{1, m}$. Unele din cuvintele p_j pot să fie vide; pentru precizarea ideilor să presupunem că $p_2, p_3, p_5 = \lambda$. Atunci pentru derivările $X_j \underset{G}{\overset{*}{\Rightarrow}} p_j,\ j \neq 2, 3, 5$, care au lungimea de cel mult n, conform ipotezei inductive avem $X_j \underset{G'}{\overset{*}{\Rightarrow}} p_j,\ \ j = 2, 3, 5$.

Pe de altă parte , pentru $j = 2, 3, 5$ avem $X_2 \underset{G}{\overset{*}{\Rightarrow}} \lambda,\ X_3 \underset{G}{\overset{*}{\Rightarrow}} \lambda,\ X_5 \underset{G}{\overset{*}{\Rightarrow}} \lambda$, deci $X_2, X_3, X_5 \in U_f$. Rezultă că

$$X \to X_1 X_4 X_6 \dots X_m \in P'$$

aşa încât putem scrie

$$X \underset{G'}{\Rightarrow} X_1 X_4 X_6 \dots X_m \underset{G'}{\overset{*}{\Rightarrow}} p_1 p_4 p_6 \dots p_m = p.$$

Deci $X \underset{G'}{\overset{*}{\Rightarrow}} p$ şi luând $X = S$ obţinem $p \in L(G')$. Prin urmare $L(G) \subseteq L(G')$.

Să arătăm acum incluziunea inversă , $L(G'') \subseteq L(G)$.

Fie $p \in L(G')$, $S \underset{G'}{\overset{*}{\Rightarrow}} p$. Punem în evidenţă o derivare directă oarecare

$$S \underset{G'}{\overset{*}{\Rightarrow}} u \underset{G'}{\Rightarrow} v \underset{G'}{\overset{*}{\Rightarrow}} p.$$

Dacă în derivarea directă $u \underset{G'}{\Rightarrow} v$ se aplică o regulă care există şi în G, atunci evident pasul respectiv poate fi făcut şi în G. Să presupunem că se aplică o regulă nou introdusă de exemplu $X \to BC$, deci pasul respectiv va avea forma

$$u = u'Xu'' \underset{G'}{\Rightarrow} u'BCu'' = v$$

dar regula $X \to BC$ din P' a provenit dintr-o regulă din P, lăsând la o parte simboluri din U_f, în cazul nostru din $X \to ABC$, lăsându-l la o parte pe $A \in U_f$. Deoarece $A \underset{G}{\overset{*}{\Rightarrow}} \lambda$, avem

$$u = u'Xu'' \underset{G}{\Rightarrow} u'ABCu'' \underset{G}{\overset{*}{\Rightarrow}} u'ABCu'' = v.$$

Prin urmare orice pas al derivării considerate se poate obţine şi în gramatica G, deci $A_0 \underset{G}{\overset{*}{\Rightarrow}} p$ şi $p \in L(G)$, adică $L(G') \subseteq L(G)$.

Să considerăm acum o gramatica G de tipul 2 şi să presupunem că $\lambda \in L(G)$. Construim gramatica G' ca şi mai sus; orice cuvânt $p \neq \lambda$ din $L(G)$ se poate obţine în G' şi invers , deci $L(G') = L(G) \setminus \{\lambda\}$. Considerăm atunci o gramatică $G'' = (V_N \cup \{S''\}, V_T, S'', P' \cup \{S'' \to \lambda, S'' \to S\})$. Evident $L(G'') = L(G)$. Toate regulile lui G'' respectă tipul 1 (cu $u = v = \lambda$) şi conţine o singură regulă de ştergere $S'' \to \lambda$ iar S'' nu apare în partea dreaptă a vreunei reguli. Deci G'' este de tipul 1 şi orice limbaj independent de context este inclus în $\mathcal{L}_1$, adică $\mathcal{L}_2 \subseteq \mathcal{L}_1$.□

1.3 Proprietăţi de închidere a familiilor Chomsky

Fiind dată o operaţie binară notată cu "$\bullet$" pe o familie de limbaje L, vom spuna că familia L este închisă la operaţia "$\bullet$" dacă $L_1, L_2 \in L$ implică $L_1 \bullet L_2 \in L$. Definiţia noţiunii de închidere pentru operaţii unare sau cu aritate oarecare este analoagă.

Vom numi familiile din clasificarea Chomsky, mai scurt, familiile Chomsky. Se poate demonstra că familiile de limbaje din ierarhia Chomsky sunt închise la

operaţiile regulate, adica reuniune, produs şi închidere Kleene. O demonstraţie detaliată poate fi găsită în Dragan[1998] sau Jucan[1999]. In cele ce urmează vom prezenta doar demonstraţiile pentru cazul familiilor $\mathcal{L}_j, j = 2, 3$.

Închiderea familiilor Chomsky la reuniune.

Teorema 1.1 *Familiile de limbaje $L_j, j = 2, 3$ sunt închise faţă de reuniune.*

Demonstraţie. Fie $G_k = (V_{N_k}, V_{T_k}, S_k, P_k), k = 1, 2$ două gramatici de acelaşi tip $j, j = 0, 1, 2, 3$. Putem presupune că $V_{N_1} \cap V_{N_2} = \emptyset$. Trebuie să arătăm că limbajul $L(G_1) \cup L(G_2)$ este de acelaşi tip j. În acest scop vom construi o gramatică de tipul j care să genereze limbajul $L(G_1) \cup L(G_2)$. Considerăm următoarea gramatică:

$$G = (V_{N_1} \cup V_{N_2} \cup \{S\}, V_{T_1} \cup V_{T_2}, P_1 \cup P_2 \cup \{S \to S_1 | S_2\}).$$

Este evident că G are acelaş tip j ca şi cele două gramatici date.

Vom arăta că $L(G) = L(G_1) \cup L(G_2)$.

Fie $p \in L(G)$, adică $S \overset{*}{\underset{G}{\Rightarrow}} p$. În prima derivare directă trebuie cu necesitate să se utilizeze una din regulile $S \to S_1$ sau $S \to S_2$. Prin urmare avem una din următoarele două variante:

$$S \underset{G}{\Rightarrow} S_1 \overset{*}{\underset{G}{\Rightarrow}} p, \text{ deci } S_1 \overset{*}{\underset{G_1}{\Rightarrow}} p, p \in L(G_1) \subseteq L(G_1) \cup L(G_2)$$
$$S \underset{G}{\Rightarrow} S_2 \overset{*}{\underset{G}{\Rightarrow}} p, \text{ deci } S_2 \overset{*}{\underset{G_2}{\Rightarrow}} p, p \in L(G_2) \subseteq L(G_1) \cup L(G_2).$$

Deci $p \in L(G_1) \cup L(G_2)$, adică $L(G) \subseteq L(G_1) \cup L(G_2)$.

Observaţie. În implicaţia "$S_1 \overset{*}{\Rightarrow} p$ în gramatica G_1 rezultă $S_1 \overset{*}{\underset{G}{\Rightarrow}} p$" se utilizează faptul că neterminalele celor două gramatici sunt disjuncte. Astfel, derivarea $S_1 \Rightarrow p$ se va scrie detaliat astfel

$$G: \ S_1 \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_n = p.$$

Întrucât $S_1 \in V_{N_1}$ şi $S_1 \notin V_{N_2}$ în derivarea directă $S_1 \Rightarrow u_1$ se utilizează cu necesitate o regulă din P_1; prin urmare $u_1 \in V_{G_1}^*$, deci toate neterminalele din u_1 sunt din V_{N_1}. Mai departe, în derivarea directă $u_1 \Rightarrow u_2$ se utilizează cu necesitate o regulă din P_1, etc. În consecinţă, în derivarea $S_1 \overset{*}{\underset{G}{\Rightarrow}} p$ toate simbolurile sunt din V_{G_1} şi se utilizează numai reguli din P_1, de unde rezultă implicaţia.

Invers, fie $p \in L(G_1) \cup L(G_2)$. Dacă, de exemplu, $p \in L(G_1)$, avem $S_1 \overset{*}{\underset{G_1}{\Rightarrow}} p$, rezultă $S_1 \overset{*}{\underset{G}{\Rightarrow}} p$, deci putem construi derivarea $S \underset{G}{\Rightarrow} S_1 \overset{*}{\underset{G}{\Rightarrow}} p$.

Prin urmare, $p \in L(G)$ şi $L(G_1) \cup L(G_2) \subseteq L(G)$.□

Exemplu. Fie limbajele

$$L_1 = \{1, 11, 111, \ldots\}, \quad L_2 = \{01, 0011, 000111, \ldots\}.$$

Se poate verifica că aceste limbaje sunt generate de gramaticile

$$G_1 : \ A \to 1A|1,$$
$$G_2 : \ B \to 0B1|01.$$

Se vede că $L_1 \in \mathcal{L}_3 \subseteq \mathcal{L}_2$, $L_2 \in \mathcal{L}_2$ deci ambele limbaje sunt de tipul 2. Problema închiderii familiei $\mathcal{L}_2$ la reuniune constă în: *este limbajul*

$$L_1 \cup L_2 = \{1, 11, \ldots, 01, 0011, \ldots\}$$

de asemenea de tipul 2 (adică există o gramatică de tipul 2 care să îl genereze) ? Răspunsul este afirmativ; gramatica de tipul 2 care generează limbajul $L_1 \cup L_2$ are regulile

$$S \to A|B, \ A \to 1A|1, \ B \to 0B1|01.$$

Închiderea familiilor Chomsky la produs.

Teorema 1.2 *Familiile de limbaje $\mathcal{L}_j, j = 2, 3$ sunt închise faţă de produs.*

Demonstraţie. Fie $G_k = (V_{N_k}, V_{T_k}, S_k, P_k), k = 1, 2$ două gramatici de acelaşi tip $j, j = 2, 3$. Putem presupune că $V_{N_1} \cap V_{N_2} = \emptyset$. Vom construi o gramatică de tipul j care să genereze limbajul $L(G_1)L(G_2)$.

Familia $\mathcal{L}_2$. Luăm gramatica G astfel

$$G = (V_{N_1} \cup V_{N_2} \cup \{S\}, V_{T_1} \cup V_{T_2}, P_1 \cup P_2 \cup \{S \to S_1S_2\}).$$

Fie $p \in L(G), S \overset{*}{\underset{G}{\Rightarrow}} p$. Această derivare va avea forma

$$S \underset{G}{\Rightarrow} S_1S_2 \overset{*}{\underset{G}{\Rightarrow}} p.$$

Vom aplica lema de localizare de la limbajele independente de context: dacă $S_1S_2 \overset{*}{\Rightarrow} p$ atunci $p = p_1p_2$ şi $S_1 \overset{*}{\Rightarrow} p_1$, $S_2 \overset{*}{\Rightarrow} p_2$.

Mai departe, avem că $p_1 \in L(G_1), p_2 \in L(G_2)$ şi deci $p \in L(G_1)L(G_2)$.

Invers, fie $p \in L(G_1)L(G_2)$. Atunci $p = p_1p_2$ şi $S_1 \overset{*}{\underset{G_1}{\Rightarrow}} p_1$, $S_2 \overset{*}{\underset{G_2}{\Rightarrow}} p_2$. Aceste derivări sunt valabile şi în gramatica G. Prin urmare putem scrie:

$$G : \ S \underset{G}{\Rightarrow} S_1S_2 \overset{*}{\underset{G}{\Rightarrow}} p_1p_2 = p,$$

adică $p \in L(G)$ şi $L(G_1)L(G_2) \subseteq L(G)$.□

Să mai observăm că construcţia de mai sus nu se poate aplica la familia $\mathcal{L}_3$, deoarece regula nou introdusă $S \to S_1S_2$ nu este de tipul 3.

Familia $\mathcal{L}_3$. Luăm gramatica G de forma

$$G = (V_{N_1} \cup V_{N_2}, V_{T_1} \cup V_{T_2}, S_1, P_1' \cup P_2),$$

unde P_1' se obţine din P_1 prin înlocuirea regulilor de forma $A \to p$ cu $A \to pS_2$. Evident, G astfel construită este de tipul 3.

Vom numi regulile de forma $A \to pB$ de categoria I iar cele de forma $C \to q$ de categoria a II-a. Evident că într-o derivare care transformă simbolul de start într-un cuvânt al limbajului generat de gramatică, în toate derivările directe se aplică reguli de categoria I iar în ultima o regulă de categoria II. Să mai observăm că P_1' are reguli numai de categoria I.

Fie $p \in L(G)$, deci $S_1 \overset{*}{\underset{G}{\Rightarrow}} p$. În această derivare, primele reguli care se aplică sunt din P_1, apoi o regulă nou introdusă (cu necesitate, căci altfel nu putem elimina neterminalele) iar ultimele reguli sunt din P_2. Prin urmare, derivarea va avea forma

$$S_1 \overset{*}{\underset{G}{\Rightarrow}} qA \underset{G}{\Rightarrow} qrS_2 = p_1S_2 \overset{*}{\underset{G}{\Rightarrow}} p_1p_2 = p.$$

Este clar că $S_1 \overset{*}{\underset{G_1}{\Rightarrow}} qr = p_1$ (la ultima derivare s-a reutilizat regula de categoria II din P_1 din care a provenit regula nouă) şi $S_2 \overset{*}{\underset{G_2}{\Rightarrow}} p_2$, adică $p_1 \in L(G_1), p_2 \in L(G_2)$. Prin urmare $p = p_1p_2 \in L(G_1)L(G_2)$ şi deci $L(G) \subseteq L(G_1)L(G_2)$.

Fie acum $p \in L(G_1)L(G_2)$. Rezultă că $p = p_1p_2,\ p_1 \in L(G_1), p_2 \in L(G_2)$ şi deci $S_1 \overset{*}{\underset{G_1}{\Rightarrow}} p_1, S_2 \overset{*}{\underset{G_2}{\Rightarrow}} p_2$. În derivarea $S_1 \overset{*}{\underset{G_1}{\Rightarrow}} p_1$ ultima regulă utilizată este de categoria a II-a, să presupunem $A \to r$, deci această derivare va avea forma

$$S_1 \overset{*}{\underset{G_1}{\Rightarrow}} qA \Rightarrow qr = p_1.$$

Ţinând cont că în G avem regula $A \to rS_2$, putem scrie în G derivarea

$$S_1 \overset{*}{\underset{G}{\Rightarrow}} qA \underset{G}{\Rightarrow} qrS_2 = p_1S_2 \overset{*}{\underset{G}{\Rightarrow}} p_1p_2 = p,$$

adică $p \in L(G)$ şi $L(G_1)L(G_2) \subseteq L(G)$.□

Închiderea familiilor Chomsky la operaţia închidere Kleene.

Teorema 1.3 *Familiile $\mathcal{L}_j, j = 2, 3$ sunt închise faţă de operaţia de închidere Kleene.*

Demonstraţie.

Familia $\mathcal{L}_2$. Construim gramatica G^* astfel

$$G^* = (V_N \cup \{S^*\}, V_T, S^*, P \cup P' \cup \{S^* \to S^*S|\lambda\})$$

Fie $p \in L(G^*)$, deci $S \overset{*}{\Rightarrow} p$. Această derivare va avea forma (în G^*)

$$(B) \quad S^* \overset{*}{\Rightarrow} S^*S \ldots S \overset{*}{\Rightarrow} p.$$

Conform lemei de localizare, $p = p_1 \ldots p_n,\ S \overset{*}{\Rightarrow} p_j, j = \overline{1,n}$ şi deci $S \overset{*}{\Rightarrow} p_j$, ceea ce înseamnă că $p \in L(G)^n \subseteq L(G)^*$, adică $L(G)^* \subseteq L(G^*)$.

Invers, fie $p \in L(G)^*$; atunci $p = p_1 \ldots p_n$, $S \overset{*}{\Rightarrow} p_j, j = \overline{1,n}$. Putem scrie atunci derivarea (B) şi $p \in L(G^*)$, adică $L(G)^* \subseteq L(G^*)$.□

Familia $\mathcal{L}_3$. Construim gramatica G^* astfel

$$G^* = (V_N \cup \{S^*\}, V_T, S^*, P \cup P' \cup \{S^* \to S|\lambda\}),$$

unde P' se obţine din regulile de categoria II din P şi anume, dacă $A \to p \in P$ atunci $A \to pS \in P'$.

Fie $p \in L(G^*)$, adică $S^* \overset{*}{\Rightarrow} p$. Punând în evidenţă prima derivare directă, avem

$$G^* :\ S^* \Rightarrow S \overset{*}{\Rightarrow} p.$$

Dacă în derivarea $S \overset{*}{\Rightarrow} p$ se aplică numai reguli din P, atunci $S \underset{G}{\overset{*}{\Rightarrow}} p, p \in L(G) \subseteq L(G^*)$.

Să presupunem că se aplică şi reguli din P'; derivarea va avea forma

$$(C) \quad S^* \Rightarrow S \overset{*}{\Rightarrow} p_1S \overset{*}{\Rightarrow} p_1p_2S \overset{*}{\Rightarrow} \ldots \overset{*}{\Rightarrow} p_1 \ldots p_n.$$

Evident, $S \overset{*}{\Rightarrow} p_j$, deci $p_j \in L(G)$, adică $p \in L(G)^*$, ceea ce înseamnă că $L(G^*) \subseteq L(G)^*$.

Invers, fie $p \in L(G)^*$; atunci $p = p_1 \ldots p_n,\ S \overset{*}{\Rightarrow} p_j, j = \overline{1,n}$, sau $S \overset{*}{\Rightarrow} p_j$ şi putem scrie derivarea (C). Aceasta înseamnă că $p \in L(G^*)$ sau $L(G)^* \subseteq L(G^*)$.□

Recapitulare asupra construcţiilor gramaticilor:

Reuniune

$$j = 2, 3 :$$
$$G = (V_{N_1} \cup V_{N_2} \cup \{S\}, V_{T_1} \cup V_{T_2}, P_1 \cup P_2 \cup \{S \to S_1|S_2\}).$$

Produs

$$j = 2 :$$
$$G = (V_{N_1} \cup V_{N_2} \cup \{S\}, V_{T_1} \cup V_{T_2}, P_1 \cup P_2 \cup \{S \to S_1S_2\}).$$
$$j = 3 :$$
$$G = (V_{N_1} \cup V_{N_2}, V_{T_1} \cup V_{T_2}, S_1, P_1' \cup P_2),$$

unde P_1' se obţine din P_1 prin înlocuirea regulilor de forma $A \to p$ cu $A \to pS_2$.

Închidere Kleene

$$j = 2 :$$
$$G^* = (V_N \cup \{S^*\}, V_T, S^*, P \cup \{S^* \to S^*S|\lambda\})$$
$$j = 3 :$$
$$G^* = (V_N \cup \{S^*\}, V_T, S^*, P \cup P' \cup \{S^* \to S|\lambda\}).$$

1.4 Problema compilării

Un **limbaj de programare** este un limbaj care are drept scop descrierea unor procese de prelucrare a anumitor date şi a structurii acestora (în unele cazuri descrierea structurii datelor este preponderentă), prelucrare care se realizează în general cu ajutorul unui sistem de calcul.

Există în prezent un număr mare de limbaje de programare de *nivel înalt* sau evoluate care se caracterizează printr-o anumită naturaleţe, în sensul că descrierea procesului de prelucrare cu ajutorul limbajului este apropiată de descrierea naturală a procesului respectiv precum şi prin independenţa unor astfel de limbaje faţă de sistemul de calcul. Dintre limbajele de acest tip cu o anumită răspândire în momentul de faţă menţionăm limbajele PASCAL, FORTRAN, C, JAVA, etc.

O altă clasă importantă de limbaje, sunt *limbajele de asamblare*, sau de nivel inferior, ale căror caracteristici depind de sistemul de calcul considerat. In general, fiecare sistem de calcul (sau tip de sistem de calcul), are propriul său limbaj de asamblare; de exemplu, limbajul de asamblare ale sistemelor de calcul echipate cu procesoare de tip Intel Z-80 este denumit în mod curent ASSEMBLER. Instrucţiile unui limbaj de asamblare corespund cu operaţiile simple ale sistemului de calcul iar stocarea datelor în memorie este realizată direct de utilizator la nivelul locaţiilor elementare ale memoriei. Există de asemenea anumite *pseudo-instrucţii* sau *directive* referitoare la alocarea memoriei, generarea datelor, segmentarea programelor, etc., precum şi macroinstrucţii care permit generarea unor secvenţe tipice de program sau accesul la bibliotecile de subprograme.

In afară de limbajele evoluate şi de limbajele de asamblare, există numeroase *limbaje specializate*, numite uneori şi de *comandă*, care se referă la o anumită clasă de aplicaţii. Menţionăm de exemplu limbajul pentru prelucrarea listelor LISP, limbajele utilizate în cadrul softwarelui matematic (Mathematica, Maple, MATCAD, etc.) şi multe altele. În general însă astfel de limbaje nu sunt considerate limbaje de programare propriuzise.

Un program redactat într-un limbaj de programare poartă denumirea de **program sursă**. Fiecare sistem de calcul, în funcţie de particularităţile sale, posedă un anumit limbaj propriu, numit *cod maşină*, acesta fiind singurul limbaj înţeles de procesorul sistemului. Un astfel de limbaj depinde de structura instrucţiilor procesorului, de setul de instrucţii, de posibilităţile de adresare, etc. Un program redactat în limbajul cod maşină al sistemului de calcul îl numim **program**

obiect.

Procesul de transformare al unui program sursă în program obiect se numeşte **compilare** sau *translatare*, uneori chiar *traducere*. De obicei termenul de compilare este utilizat numai în cazul limbajelor evoluate, în cazul limbajelor de asamblare fiind utilizat termenul de *asamblare*.

Compilarea (asamblarea) este efectuată de un program al sistemului numit **compilator** (*asamblor*). De multe ori compilatoarele nu produc direct program obiect, ci un text intermediar apropiat de programul obiect, care în urma unor prelucrări ulterioare devine program obiect. De exemplu, compilatoarele sistemelor de operare DOS produc un text numit obiect (fişiere cu extensia obj) care în urma unui proces numit editare de legături şi a încărcării în memorie devine program obiect propriuzis, numit program executabil (fişiere cu extensia exe). Există mai multe raţiuni pentru o astfel de tratare, între care posibilitatea cuplării mai multor module de program realizate separat sau provenite din limbaje sursă diferite, posibilitatea creării unor biblioteci de programe în formatul intermediar şi utilizarea lor în alte programe, etc.

Un program sursă poate de asemenea să fie executat de către sistemul de calcul direct, fără transformarea lui prealabilă în program obiect. În acest caz, programul sursă este prelucrat de un program al sistemului numit *interpretor*; acesta încarcă succesiv instrucţiile programului sursă, le analizează din punct de vedere sintactic şi semantic şi după caz, le execută sau efectuează anumite operaţii auxiliare.

Procesul de compilare este un proces relativ complex şi comportă operaţii care au un anumit caracter de autonomie. Din aceste motive procesul de compilare este de obicei descompus în mai multe subprocese sau faze, fiecare fază fiind o operaţie coerentă, cu caracteristici bine definite. În principiu aceste faze sunt parcurse secvenţial (pot să existe şi anumite reveniri) iar programul sursă este transformat succesiv în formate intermediare. Se poate considera că fiecare fază primeşte de la faza precedentă un fişier cu programul prelucrat într-un mod corespunzător fazei respective, îl prelucrează şi furnizează un fişier de ieşire, iarăşi într-un format bine precizat, fişier utilizat în faza următoare. Există cinci faze de compilare principale: analiza lexicală, analiza sintactică, generarea formatului intermediar, generarea codului, optimizarea codului şi două faze auxiliare, tratarea erorilor şi tratarea tabelelor (vezi figura 1.1).

Analiza lexicală are ca obiectiv principal determinarea unităţilor lexicale ale unui program, furnizarea codurilor acestora şi detectarea eventualelor erori lexicale. Pe lângă aceste operaţii de bază, la analiza lexicală se mai pot efectua anumite operaţii auxiliare precum: eliminarea blank-urilor (dacă limbajul permite utilizarea fără restricţii a acestora), ignorarea comentariilor, diferite conversiuni ale unor date (care se pot efectua la această fază), completarea tabelelor compilatorului.

Analiza sintactică determină unităţile sintactice ale programului (secvenţe de text pentru care se poate genera format intermediar) şi verifică programul

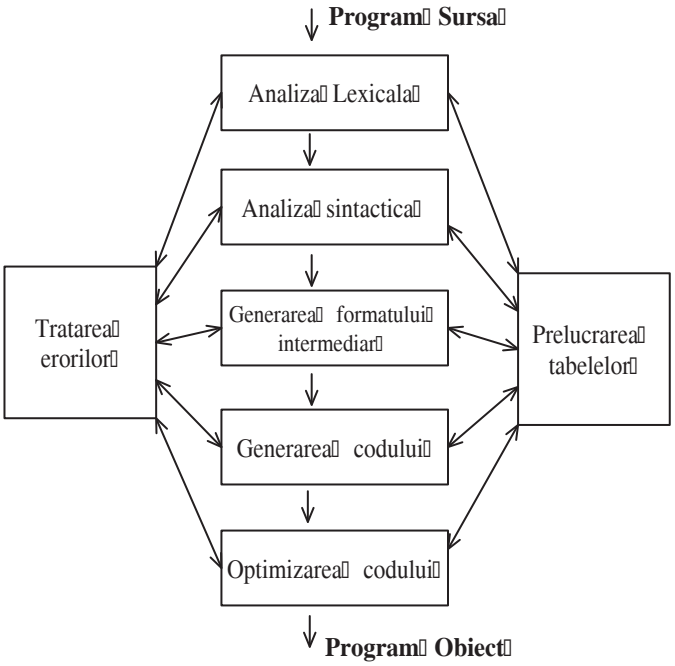

Figura 1.1: Fazele compilării

din punct de vedere sintactic. Este faza centrală a unui compilator, deseori toate celelalte faze sunt rutine apelate de analizorul sintactic pe măsură ce este posibilă efectuara unei părţi din faza respectivă. Tot la analiza sintactică se definitivează tabelele de informaţii şi se realizează prelucrarea erorilor sintactice.

Faza de *generare a formatului intermediar* se referă la transformarea programului într-o formă numită intermediară pornind de la care se poate, printr-o procedură relativ simplă, să se obţină programul obiect. Structura acestei faze depinde de formatul intermediar ales de către proiectant şi de modalitatea de implementare; uzual se folosesc *cvadruple, triplete* sau *şirurile poloneze*.

Generarea codului este o fază in care se realizează codul obiect corespunzător programului. Practic în această fază se obţine programul în limbaj de asamblare corespunzător programului sursă redactat într-un limbaj evoluat. În mod obişnuit generatorul de cod este constituit din rutinele generatoare de cod corespunzătoare diverselor unităţi ale formatului intermediar.

În faza de *optimizare* a codului se realizează o anumită îmbunătăţire a codului obiect astfel încât programul rezultat să fie cât mai performant (în privinţa consumului de timp şi memorie). Cele mai tipice exemple sunt eliminarea încărcărilor şi a memorărilor redundante sau optimizarea ciclurilor.

Fazele de *Prelucrare a tabelelor* şi de *Tratare a erorilor* vor fi atinse numai parţial în aceast curs. Facem însă menţiunea că prelucrarea tabelelor poate să aibă o influenţă importantă asupra performanţelor unui compilator, în mod special din punctul de vedere al timpului de execuţie (compilare) şi că tratarea erorilor are o anumită implicaţie în eliminarea erorilor sintactice.

1.5 Probleme propuse

1. Găsiţi limbajul generat de gramatica $G = (V_N, V_T, S, P)$, precizând tipul gramaticii (cf. clasificării Chomsky):

   (a) $V_N = \{S\}$; $V_T = \{0, 1, 2\}$;
   $P = \{S \to 0S0|1S1|2S2|0\}$.

   (b) $V_N = \{S\}$; $V_T = \{a, b\}$;
   $P = \{S \to aSb|ab\}$.

   (c) $V_N = \{S, A, B\}$; $V_T = \{a, b, c\}$;
   $P = \{S \to abc|aAbc,\ Ab \to bA,\ Ac \to Bbcc,\ bB \to Bb,\ aB \to aaA|aa\}$.

   (d) $V_N = \{S, A, C\}$; $V_T = \{+, -, 0, 1, \ldots, 9\}$;
   $P = \{S \to A| + A| - A,\ A \to AC|C,\ C \to 0|1|\ldots|9\}$.

   (e) $V_N = \{S, A, B\}$; $V_T = \{0, 1\}$;
   $P = \{S \to 0S|1A,\ A \to 0B|1S|0,\ B \to 0A|1B|1\}$.

   (f) $V_N = \{A\}$, $V_T = \{a, b\}$, $S = A$, $P = \{A \to aA|b\}$;

(g) $V_N = \{x_0\}$, $V_T = \{A, B, \ldots, Z\}$, $S = x_0$, $P = \{x_0 \to x_1 D$, $x_1 \to x_2 N, x_2 \to E\}$;

(h) $V_N = \{A\}$, $V_T = \{0, 1, 2\}$, $S = A$, $P = \{A \to 0A0|1A1|2A2|\lambda\}$;

(i) $V_N = \{S, A\}$, $V_T = \{0, 1, \ldots, 9, .\}$, $P = \{S \to A.A, A \to 0A|1A| \ldots |9A|0|1| \ldots |9\}$;

(j) $V_N = \{S\}$, $V_T = \{PCR, PDAR, UDMR\}$, $P = \{S \to PCR|PDAR|UDMR\}$;

(k) $V_N = \{A, B, C\}$, $V_T = \{0, 1\}$, $S = A$, $P = \{A \to 0A|1B|1$, $B \to 0C|1A, C \to 0B|1C|0\}$;

(l) $V_N = \{S, A, B, C\}$, $V_T = \{0, 1, \ldots, 9, +, -\}$, $P = \{S \to +A| - A|A, A \to 0A|1A| \ldots |9A|0|1| \ldots |9\}$;

(m) $V_N = \{S\}$, $V_T = \{(,)\}$, $P = \{S \to S(S)S|\lambda\}$;

(n) $V_N = \{E, T, F\}$, $V_T = \{(,), i, +, *\}$, $S = E$, $P = \{E \to E + T|T, T \to T * F|F, F \to (E)|i\}$;

(o) $V_N = \{S, A, B\}$, $V_T = \{a, b, c\}$, $P = \{S \to abc|aAbc$, $Ab \to bA, Ac \to Bbcc, bB \to Bb, aB \to aaA|aa\}$;

(p) $V_N = \{S, A, B, C, D, E\}, V_T = \{a\}$, $P = \{S \to ACaB, Ca \to aaC, CB \to DB|E, aD \to Da, AD \to AC, aE \to Ea, AE \to \lambda\}$;

(q) $V_N = \{S, A, B, C, D, E\}$, $V_T = \{a, b\}$, $P = \{S \to ABC$, $AB \to aAD|bAE, DC \to BaC, EC \to BbC, Da \to aD, Db \to bD, Ea \to aE, Eb \to bE, AB \to \lambda, C \to \lambda, aB \to Ba, bB \to Bb\}$;

2. Precizaţi care dintre gramaticile precedente sunt echivalente.

3. Găsiţi gramatici pentru generarea următoarelor limbaje:

(a) $L = \{\lambda\}$;

(b) $L = \emptyset$;

(c) $L = \{0^n | n \in N\}$;

(d) $L = \{a, ab, aab, ba, baa\}$.

(e) $L = \{a^n c b^n | n \geq 1\}$.

(f) $L = \{w \in \{a, b, c\}^* | w$ conţine cc şi se termină cu $a\}$.

(g) $L = \{BEGIN|END|IF|WHILE|UNTIL\}$.

(h) $L = \{w \in \{0, 1\}^* |$ reprezentarea binară pe 8 biţi a unui întreg $\}$.

(i) $L = \{a^n b^{n+3} a^{n+1} | n \geq 0\}$.

(j) $L = \{w \in \{a, b, c\}^* | w$ începe cu a şi are maxim 4 litere $\}$.

(k) $L = \{a^i b^j c^k | i, j, k > 0\}$.

(l) $L = \{w \in \{0,1\}^* | w$ multiplu de 8 $\}$.

(m) $L = \{$ constante reale în scrierea obişnuită cu punct zecimal $p_i.p_z\}$.

(n) $L = \{a^i b^j a^i b^j\}$;

(o) $L = \{awbbw' | w, w' \in \{0,1\}^*\}$;

(p) $L = \{w \in \{0,1\}^* | w$ conţine maxim 2 de 0 $\}$;

(q) $L = \{wa\tilde{w} | w \in \{0,1\}^*\}$;

(r) $L = \{w | w$ octet ce reprezintă un număr par $\}$;

(s) $L = \{A, B, C, \ldots, Z\}$;

4. Construiţi o gramatică ce conţine reguli de ştergere, dar generează un limbaj λ-liber.

5. Folosind teorema să se construiască o gramatică independentă de context, fără reguli de ştergere care generează limbajul de la punctul precedent.

Capitolul 2

Limbaje regulate

2.1 Automate finite şi limbaje regulate

Automate finite. Automatele finite sunt mecanisme pentru recunoaşterea limbajelor de tipul 3 (regulate). Un automat finit (AF) se compune dintr-o *bandă de intrare* şi un *dispozitiv de comandă*.

Pe banda de intrare sunt înregistrate simboluri ale unui alfabet de intrare, constituind pe bandă un cuvânt p. La fiecare pas de funcţionare banda se deplasează cu o poziţie spre stânga.

Dispozitivul de comandă posedă un *dispozitiv de citire* de pe bandă; dispozitivul se află permanent într-o anumită stare internă, element al unei mulţimi finite de stări. Schema unui automat finit este redată în figura 2.1.

Automatul finit funcţionează în paşi discreţi. Un pas de funcţionare constă din: dispozitivul de comandă citeşte de pe banda de intrare simbolul aflat în dreptul dispozitivului de citire; în funcţie de starea internă şi de simbolul citit, automatul trece într-o nouă stare şi mută banda cu o poziţie spre stânga. Automatul îşi încetează funcţionarea după ce s-a citit ultimul simbol înregistrat pe

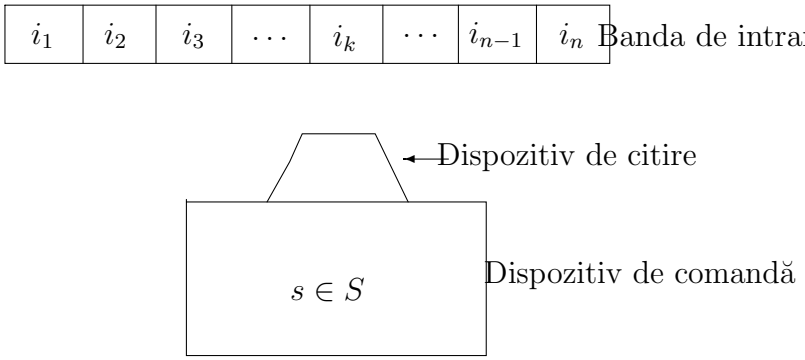

Figura 2.1: Reprezentarea schematică a unui automat finit

bandă; în acest moment el se va afla într-o anumită stare, care, după cum vom vedea, va juca un rol important în recunoaşterea cuvintelor.

Din punct de vedere matematic, un automat finit este un sistem

$$AF = (\Sigma, I, f, s_0, \Sigma_f),$$

unde

Σ este mulţimea de stări;

I este alfabetul de intrare;

$f : \Sigma \times I \longrightarrow \mathcal{P}(\Sigma)$ este funcţia de evoluţie;

$s_0 \in \Sigma$ este starea iniţială;

$\Sigma_f \subseteq \Sigma$ este mulţimea de stări finale.

Dacă pentru orice $(s, i) \in \Sigma \times I$, avem $|f(s, i)| \leq 1$ automatul se numeşte *determinist*; în caz contrar se numeşte *nedeterminist*.

Observaţii:

(1) $\mathcal{P}(\Sigma)$ este mulţimea părţilor lui Σ; deoarece $\emptyset \in \mathcal{P}(\Sigma)$ este posibil ca pentru un anumit $s \in S$ şi $i \in I$ să avem $f(s, i) = \emptyset$.

(2) În cazul unui automat determinist vom scrie $f(s, i) = s'$ (în loc de $f(s, i) \in \{s'\}$.

(3) Definiţia dată este specifică teoriei limbajelor formale. O altă definiţie (mai generală) , întâlnită în teoria automatelor este următoarea: un automat finit este un sistem $AF = (\Sigma, I, O, f, g, s_0, F)$ unde, Σ, I, f, s_0, F au semnificaţia de mai sus, O este *alfabetul de ieşire* iar $g : \Sigma \times I \longrightarrow \mathcal{P}(O)$ este *funcţie de ieşire*. Funcţionalitatea unui astfel de automat finit este analoagă cu cea descrisă mai sus, cu deosebirea că la fiecare pas automatul furnizează o ieşire $o \in g(s, i)$.

Funcţionarea unui automat finit se poate bloca în situaţia în care el se află în starea s, citeşte de pe bandă simbolul i şi $f(s, i) = \emptyset$; evident că în acest caz funcţionarea în continuare nu mai este posibilă.

Prin diagrama de stări a unui automat finit înţelegem un graf orientat care are nodurile etichetate cu stările $s \in \Sigma$ iar arcele se construiesc astfel: nodurile s, s' se unesc cu un arc orientat de la s la s' dacă există $i \in I$ astfel încât $s' \in f(s, i)$; arcul respectiv va fi notat cu i.

Exemplu $AF = (\Sigma, I, f, s_0, \Sigma_f)$ unde $\Sigma = \{s_0, s_1, s_2\}$, $I = \{i_1, i_2\}$, $\Sigma_f = \{s_2\}$, iar f este dată de tabelul

	s_0	s_1	s_2
i_1	$\{s_1\}$	$\{s_2\}$	$\{s_0\}$
i_2	$\emptyset$	$\{s_0, s_1\}$	$\{s_0, s_1\}$

Diagrama de stări corespunzătoare este prezentată în figura 2.2.

Funcţia de evoluţie Funcţia f se prelungeşte de la $\Sigma \times I$ la $\mathcal{P}(\Sigma) \times I^*$ deci $f : \mathcal{P}(\Sigma) \times I^* \longrightarrow \mathcal{P}(\Sigma)$, astfel:

(a) $f(s, \lambda) = \{s\}, \forall s \in \Sigma$,

(b) $f(\emptyset, i) = \emptyset, \forall i \in I$,

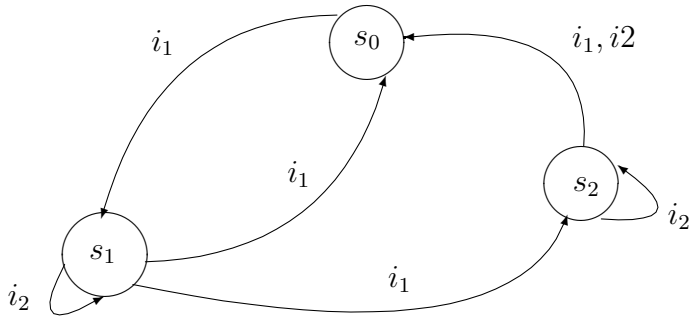

Figura 2.2: Diagrama de stări

(c) $f(Z,i) = \bigcup_{s\in Z} f(s,i), Z \in \mathcal{P}(\Sigma), Z \neq \emptyset$,
(d) $f(Z,pi) = f(f(Z,p),i)$.

Să observăm că relaţiile de mai sus constituie o definiţie prin recurenţă, corectă; fiind dat f, putem defini mai întâi toate valorile $f(Z,i)$ (un număr finit, deoarece I este o mulţime finită), apoi $f(Z,p)$ pentru $|p| = 2$, în continuare $f(Z,p)$ pentru $|p| = 3$, etc.

Proprietăţile funcţiei f:

1. Dacă Z_k este o familie de părţi a lui Σ, avem

$$f(\bigcup_k Z_k, i) = \bigcup_k f(Z_k, i)$$

Demonstraţie. Utilizând (c) putem scrie

$$\bigcup_k f(Z_k,i) = \bigcup_k (\bigcup_{s\in Z_k} f(s,i)) = \bigcup_{s\in \cup Z_k} f(s,i) = f(\bigcup_k Z_k, i).\square$$

2. $f(Z,p) = \bigcup_{s\in Z}(f(s,p)),\ p \in I^*$.

Demonstraţie. Prin inducţie asupra lungimii lui p.

Pentru $|p| = 1$ avem $f(Z,i) = \bigcup_{s\in Z} f(s,i)$, adică (c).

Presupunem că relaţia este adevărată pentru $|p| = m$ şi considerăm un p astfel încât $|p| = m+1$, deci $p = p'i, |p'| = m$. Putem scrie

$$\begin{aligned} f(Z,p) = f(Z,p'i) = f(f(Z,p'),i) = f(\textstyle\bigcup_{s\in Z} f(s,p'),i) = \\ = \textstyle\bigcup_{s\in Z} f(f(s,p'),i) = \bigcup_{s\in Z} f(s,p'i) = \bigcup_{s\in Z} f(s,p). \end{aligned} \quad \square$$

3. $f(s,pq) = f(f(s,p),q),\ p,q \in I^*$.

Demonstraţie. Inducţie asupra lungimii lui q.

Dacă $|q| = 1$, atunci $q = i$ şi relaţia se reduce la (d).

Presupunem că proprietatea este adevărată pentru r şi considerăm $|q| = r+1$. Deci $q = q'i$. Avem

$$f(s,pq) = f(s,pq'i) = f(f(s,pq'),i) = f(f(f(s,p),q'),i) = \\ = f(f(s,p),q'i) = f(f(s,p),q). \quad \square$$

Limbaje regulate.

Definiţie 2.1 *Limbajul recunoscut de automatul finit* $AF = (\Sigma, I, f, s_0, \Sigma_f)$ *este*

$$L(AF) = \{p | p \in I^*, f(s_0,p) \cap \Sigma_f \neq \emptyset\}$$

Deci $p \in L(AF)$ dacă automatul aflându-se în starea iniţială s_0, după $|p|$ paşi de funcţionare poate să ajungă într-o stare finală.

În cazul unui automat finit determinat limbajul recunoscut poate fi definit în modul următor. Pentru fiecare $s \in \Sigma$ definim funcţia $f_s : I^* \longrightarrow \mathcal{P}(\Sigma)$ prin $f_s(p) = f(s,p)$.

Atunci

$$f(s_0,p) \cap \Sigma_f \neq \emptyset \Leftrightarrow f(s_0,p) = f_{s_0}(p) \in \Sigma_f,$$

deci

$$L(AF) = \{p | f(s_0,p) \cap \Sigma_f \neq \emptyset\} = f_{s_0}^{-1}(\Sigma_f)$$

Limbajele recunoscute de automate finite le vom numi *limbaje regulate*; familia acestor limbaje o vom nota cu $\mathcal{R}$. Evident, familia limbajelor recunoscute de automate finite deterministe, $\mathcal{R}_d$, este o parte a lui $\mathcal{R}$, $\mathcal{R}_d \subseteq \mathcal{R}$. Vom arăta că cele două familii coincid.

Teorema 2.1 $\mathcal{R}_d = \mathcal{R}$.

Demonstraţie. Fie $AF = (\Sigma, I, f, s_0, \Sigma_f)$ un automat finit (în general nedeterminist). Construim următorul automat finit determinist $AF' = (\Sigma', I, f', \{s_0\}, \Sigma'_f)$ unde $\Sigma' = \mathcal{P}(\Sigma)$, $f' = f$ (prelungirea la $\mathcal{P}(\Sigma) \times I^*$), $\Sigma'_f = \{Z |\ Z \in \mathcal{P}(\Sigma), Z \cap \Sigma_f \neq \emptyset\}$.

Evident, automatul AF' este determinist.

Fie $p \in L(AF)$. Atunci $f(s_0,p) \cap \Sigma_f \neq \emptyset$ şi $f(s_0,p) \in \Sigma'_f$. Pe de altă parte, conform cu proprietatea 2 a funcţiei de evoluţie, avem

$$f'(\{s_0\},p) = f(s_0,p)$$

şi deci $f(s_0,p) \in \Sigma'_f$, adică $p \in L(AF')$ şi $L(AF) \subseteq L(AF')$.

Pe o cale analoagă se arată că $L(AF') \subseteq L(AF)$.$\square$

Observaţie. Faptul că un cuvânt este recunoscut de un automat finit se poate verifica prin calcul direct sau pe diagrama de stări.

Exemplu. Considerăm automatul din exemplul anterior şi fie $p = i_1 i_2 i_1$. Prin calcul direct:

$$f(s_0, i_1 i_2 i_1) = f(f(f(s_0,i_1),i_2),i_1) = f(f(\{s_1\},i_2),i_1) = \\ = f(\{s_0,s_1\},i_1) = f(\{s_0\},i_1) \cup f(\{s_1\},i_1) = \{s_1\} \cup \{s_2\}.$$

Astfel că $f(s_0, i_1i_2i_1) \cap \Sigma_f = \{s_2\} \neq \emptyset$ şi $p \in L(AF)$.

Pe diagrama de stări există traiectoriile:

$$s_0 \xrightarrow{i_1} s_1 \xrightarrow{i_2} s_0 \xrightarrow{i_1} s_1;$$
$$s_0 \xrightarrow{i_1} s_1 \xrightarrow{i_2} s_1 \xrightarrow{i_1} s_2;$$

A doua traiectorie ne duce într-o stare finală, deci $p \in L(AF)$.

Limbaje de tipul trei şi limbaje regulate. Vom arăta în cele ce urmează că familia limbajelor de tipul 3 coincide cu familia limbajelor regulate. În prealabil vom pune în evidenţă o formă specială a limbajelor de tipul 3, pe care convenim să o numim formă normală.

Definiţie 2.2 *Vom spune că o gramatică de tipul 3 este în forma normală dacă are reguli de generare de forma*

$$\begin{cases} A \to iB, \\ C \to j, \end{cases} \quad unde\ A, B, C \in V_N,\ i, j \in V_T$$

sau regula de completare $S \to \lambda$ şi în acest caz S nu apare în dreapta vreunei reguli.

Lema 2.1 *Orice gramatică de tipul 3 admite o formă normală.*

Demonstraţie. Dacă $G = (V_N, V_T, S, P)$ este gramatica dată, eliminăm în primul rând regulile de ştergere (teorema de la ierarhia Chomsky) apoi construim gramatica $G' = (V'_N, V_T, S, P')$, unde V'_N şi P' se definesc astfel: introducem în V'_N toate simbolurile din V_N iar în P' toate regulile din P care convin; fie acum în P o regulă de forma

$$A \to pB,\ p = i_1 \dots i_n$$

Vom introduce în P' regulile:

$$\begin{aligned} &A \to i_1 Z_1, \\ &Z_1 \to i_2 Z_2, \\ &\dots, \\ &Z_{n-1} \to i_n B, \end{aligned}$$

iar simbolurile $Z_1, \dots Z_{n-1}$ le includem in V'_N.

În cazul unei reguli de forma $A \to p$ cu $|p| > 1$ procedăm analog, exceptând ultima regulă nou introdusă care va avea forma $Z_{n-1} \to i_n$. Să mai facem observaţia că simbolurile $Z_1, \dots, Z_{n-1}$ le luăm distincte pentru fiecare caz.

Se poate arăta uşor că $L(G) = L(G')$.□

Teorema 2.2 *Familia limbajelor de tipul 3 coincide cu familia limbajelor regulate.*

Demonstraţie. Partea I: $E \in \mathcal{L}_3 \Rightarrow E \in \mathcal{R}$.

Fie E un limbaj de tipul 3 şi $G = (V_N, V_T, S, P)$ gramatica care îl generează; putem presupune că G este în formă normală. Construim automatul finit $AF = (\Sigma, I, f, s_0, \Sigma_f)$ unde $\Sigma = V_N \cup \{X\}$ (X simbol nou), $I = V_T, s_0 = S$ şi

$$\Sigma_f = \begin{cases} \{X, S\} & , pentru\ \Sigma \to \lambda \in P \\ \{X\} & , pentru\ \Sigma \to \lambda \notin P \end{cases}$$

Funcţia de evoluţie este definită de:

dacă $A \to iB \in P$ luăm $B \in f(A, i)$,
dacă $C \to j \in P$ luăm $X \in f(C, j)$,
în rest $\emptyset$

Observaţie. Automatul astfel definit este în general nedeterminist. De exemplu, dacă $A \to 0B|0$ atunci $f(A, 0) = \{B, X\}$.

Fie $p \in L(G)$, $p = i_1 \dots i_n$, deci $S \stackrel{*}{\Rightarrow} p$. Detaliat, această derivare va avea forma

$$(1)\quad S \Rightarrow i_1A_1 \Rightarrow i_1i_2A_2 \Rightarrow i_1i_2 \dots i_{n-1}A_{n-1} \Rightarrow i_1i_2 \dots i_n.$$

S-au aplicat regulile:

$$(2)\quad \begin{array}{l} S \to i_1A_1, \\ A_1 \to i_2A_2, \\ \dots \\ A_{n-1} \to i_n. \end{array}$$

În automat avem corespunzător:

$$(3):\quad \begin{array}{l} A_1 \in f(S, i_1), \\ A_2 \in f(A_1, i_2), \\ \dots \\ X \in f(A_{n-1}, i_n) \end{array}$$

Putem scrie traiectoria

$$(4)\quad S \xrightarrow{i_1} A_1 \xrightarrow{i_2} A_2 \xrightarrow{i_3} \dots \xrightarrow{i_n} X \in \Sigma_f$$

Deci $p \in L(AF)$.

Dacă $p = \lambda$, atunci $S \Rightarrow \lambda$ şi $S \to \lambda \in P$. Dar atunci $\lambda \in L(AF)$, căci automatul este în starea S şi rămîne în această stare după "citirea" lui λ; cum însă în acest caz $S \in \Sigma_f$ rezultă că şi în acest caz $p \in L(AF)$. În consecinţă $L(G) \subseteq L(AF)$.

Fie acum $p = i_1 \dots i_n \in L(AF)$; atunci avem traiectoria (4), relaţiile (3), regulile de generare (2) şi putem scrie derivarea (1), adică $p \in L(G)$ şi $L(AF) \subseteq$

$L(G)$.□
Partea II $E \in \mathcal{R} \Rightarrow E \in \mathcal{L}_3$.

Vom indica numai modul de construcţie a gramaticii. Fie $AF = (\Sigma, I, f, s_0, \Sigma_f)$ automatul finit care recunoaşte limbajul E, pe care îl presupunem determinist. Construim gramatica $G = (\Sigma, I, s_0, P)$ unde mulţimea P este definită astfel

$$f(A, i) = B \text{ generează regula } A \to iB \in P,$$
$$\text{în plus dacă } B \in \Sigma_f \text{ se generează şi regula } A \to i \in P.$$

Putem arăta că $L(G) = L(AF)$.□

2.2 Proprietăţi speciale ale limbajelor regulate

Caracterizarea limbajelor regulate. Limbajele regulate, fiind părţi din I^*, se pot caracteriza algebric, independent de mecanismele de generare (gramaticile de tipul 3) sau de cele de recunoaştere (automatele finite).

Teorema 2.3 *Fie $E \subset I^*$ un limbaj. Următoarele afirmaţii sunt echivalente.*
(a) $E \in \mathcal{R}$;
(b) E este o reuniune de clase de echivalenţe a unei congruenţe de rang finit;
(c) Următoarea congruenţă

$$\mu = \{(p, q) | \chi_E(r_1 p r_2) = \chi_E(r_1 q r_2), \forall r_1, r_2 \in I^*\},$$

unde χ_E este funcţia caracteristică a lui E, este de rang finit.

Demonstraţie: Vom arăta următoarele implicaţii: $(a) \Rightarrow (b), (b) \Rightarrow (c), (c) \Rightarrow (a)$.
$(a) \Rightarrow (b)$.

Fie $AF = (\Sigma, I, f, s_0, \Sigma_f)$ automatul finit care recunoaşte limbajul E. Definim pe I^* relaţia

$$\xi = \{(p, q) | f(s, p) = f(s, q), \forall s \in \Sigma\}.$$

Se poate vedea cu uşurinţă că ξ este o relaţie de echivalenţă (reflexivă , simetrică , tranzitivă). În plus , dacă $r \in I^*$ şi $(p, q) \in \xi$, atunci $(pr, qr) \in \xi$ şi $(rp, rq) \in \xi$. De exemplu, prima apartenenţă se deduce astfel

$$f(s, pr) = f(f(s, p), r) = f(f(s, q), r) = f(s, qr), \ etc.$$

Prin urmare ξ este o relaţie de congruenţă.

Să arătăm că această congruenţă este de rang finit, adică mulţimea cât I^*/ξ este finită.

Fie $\alpha : \Sigma \longrightarrow \Sigma$ o aplicaţie oarecare şi fie mulţimea

$$I^*(\alpha) = \{p | p \in I^*, f(s, p) = \alpha(s), \forall s \in \Sigma\}.$$

Să observăm că dacă α este funcţia identică atunci $\lambda \in I^*(\alpha)$. Deci nu toate $I^*(\alpha)$ sunt vide; în acest caz $I^*(\alpha)$ este o clasă de echivalenţă. Într-adevăr, fie $p \in I^*(\alpha) \subseteq I^*$ fixat şi fie C_p clasa de echivalenţă a lui p. Arătăm că $C_p = I^*(\alpha)$. Dacă $q \in I^*(\alpha)$, atunci $f(s,q) = \alpha(s), \forall s \in \Sigma$, ceea ce înseamnă că $f(s,q) = f(s,p), \forall s \in \Sigma$, şi deci $(p,q) \in \xi$ adică $q \in C_p$ şi $I^*(\alpha) \subseteq C_p$. Invers dacă $q \in C_p$ atunci $f(s,q) = f(s,p) = \alpha(s), \forall s \in \Sigma$ şi $q \in I^*(\alpha)$, adică $C_p \subseteq I^*(\alpha)$. Aceasta înseamnă că $I^*(\alpha) = C_p$, adică $I^*(\alpha)$ este o clasă de echivalenţă .

Între mulţimea cât I^*/ξ şi mulţimea funcţiilor definite pe Σ cu valori în Σ putem stabili următoarea corespondenţă biunivocă: *unei funcţii* $\alpha : \Sigma \longrightarrow \Sigma$ *îi corespunde clasa de echivalenţă* $I^*(\alpha)$. Invers, fiind dată o clasă de echivalenţă C, luăm $p \in C$ (oarecare) şi ataşăm lui C funcţia $\alpha(s) = f(s,p) \forall s \in \Sigma$. Ţinând cont că dacă $q \in C$ atunci $f(s,p) = f(s,q), \forall s \in \Sigma$, rezultă că funcţia α nu depinde de elementul p ales.

Dar mulţimea funcţiilor definite pe Σ cu valori în Σ este finită, deci I^*/ξ este finită, adică congruenţa ξ este de rang finit.

Fie acum $p \in L(AF)$ şi q astfel ca $(p,q) \in \xi$. Avem

$$f(s_0, q) = f(s_0, p) \in \Sigma_f;$$

adică $q \in L(AF)$. Aceasta înseamnă că odată cu elementul p, $L(AF)$ conţine clasa de echivalenţă a lui p. De aici rezultă că $L(AF)$ este constituit dintr-un anumit număr de clase de echivalenţă a lui ξ.□

(b)⇒(c)

Fie ξ o congruenţă de rang finit şi E o reuniune de clase de echivalenţă. Fie apoi $(p,q) \in \xi$; aceasta înseamnă că $r_1pr_2 \in E \Leftrightarrow r_1qr_2 \in E$, deci

$$\chi_E(r_1pr_2) = \chi_E(r_1qr_2), \forall r_1, r_2 \in I^*.$$

Prin urmare, $(p,q) \in \mu$. Orice clasă de echivalenţă din I^*/ξ este inclusă într-o clasă de echivalenţă din I^*/μ, aşa încât $card(I^*/\mu) < card(I^*/\xi)$, adică congruenţa μ este de rang finit.□

(c)⇒ (a)

Presupunem că μ este o congruenţă de rang finit; considerăm automatul finit $AF = (I^*/\mu, I, f, C_\lambda, \Sigma_f)$, unde funcţia de evoluţie f şi mulţimea de stări finale sunt

$$f(C_p, i) = C_{pi},\ \Sigma_f = \{C_p | p \in E\}.$$

Vom arăta că $E = L(AF)$. În primul rând să observăm că $f(C_p, q) = C_{pq}$ (se poate arăta prin inducţie asupra lui $|q|$). Avem

$$p \in E \Leftrightarrow C_p \in \Sigma_f \Leftrightarrow f(C_\lambda, p) = C_{\lambda p} = C_p \in \Sigma_f \Leftrightarrow p \in L(AF)$$

adică $E = L(AF)$ şi E este un limbaj regulat.□

Corolar 2.4 *Familia R este închisă la operaţia de răsturnare.*

Demonstraţie. Fie $E \in \mathcal{R}$ şi $\widetilde{E}$ răsturnatul lui E. Conform teoremei de caracterizare, $card(I^*/\mu^E) < \infty$. Avem

$$(p,q) \in \mu^E \Leftrightarrow \chi_E(r_1pr_2) = \chi_E(r_1qr_2) \Leftrightarrow$$

şi

$$\chi_{\widetilde{E}}(\widetilde{r_1}\widetilde{pr_2}) = \chi_{\widetilde{E}}(\widetilde{r_1}\widetilde{qr_2}) \Leftrightarrow (\widetilde{p},\widetilde{q}) \in \mu^{\widetilde{E}}$$

Cu alte cuvinte dacă C este o clasă de echivalenţă a lui μ^E atunci $\widetilde{C}$ (răsturnatul lui C) este o clasă de echivalenţă a lui $\mu^{\widetilde{E}}$. Aceasta înseamnă că $card(I^*/\mu^{\widetilde{E}}) = card(I^*/\mu^E) < \infty$ şi conform aceleiaşi teoreme de caracterizare $\widetilde{E} \in \mathcal{R}$.□

Observaţie. Am văzut că limbajele de tipul 3 pot fi definite de gramatici cu reguli de două categorii: drept liniare sau stâng liniare. Este evident că limbajele stâng liniare sunt răsturnatele limbajelor drept liniare. Cum familia limbajelor regulate (drept liniare) este închisă la operaţia de răsturnare, rezultă că cele două familii de limbaje coincid.

Închiderea familiei $\mathcal{L}_3$ la operaţiile $Pref$ şi complementariere.

Operaţiile $Pref$ şi complementariere se definesc în modul următor

$$Pref(E) = \{p | \exists r \in I^*, pr \in E\}, \;\; C(E) = I^* \setminus E.$$

Teorema 2.5 *Familia $\mathcal{L}_3$ este închisă la operaţiile $Pref$ şi complementariere.*

Demonstraţie. Fie $AF = (\Sigma, I, f, s_0, \Sigma_f)$ automatul finit care recunoaşte limbajul E. Putem presupune că AF este determinist.

Limbajul $Pref(E)$. Construim automatul finit $AF' = (\Sigma, I, f, s_0, \Sigma'_f)$ unde

$$\Sigma'_f = \{s \in \Sigma | s = f(s_0, q), \; q \in Pref(E)\}.$$

Este evident că $Pref(E) \subseteq L(AF')$, căci dacă $q \in Pref(E)$ atunci $s = f(s_0, q) \in \Sigma'_f$ conform definiţiei lui Σ'_f.

Să arătăm acum că $L(AF') \subseteq Pref(E)$. Fie $r \in L(AF')$, atunci $f(s_0, q) \in \Sigma'_f$, deci există $q \in Pref(E)$ astfel încât $f(s_0, r) = f(s_0, q)$. Cum q este prefixul unui cuvânt din E, există $w \in I^*$ astfel încât $qw \in E$, adică $f(s_0, q) \in \Sigma_f$. Dar

$$f(s_0, rw) = f(f(s_0, r), w) = f(f(s_0, q), w) = f(s_0, qw) \in \Sigma_f,$$

deci $rw \in E$ şi $r \in Pref(E)$. Aceasta înseamnă că $L(AF') \subseteq Pref(E)$ şi prin urmare $Pref(E) = L(AF')$, adică $Pref(E)$ este limbaj regulat.□

Limbajul $C(E)$. Avem

$$C(E) = \{p \in I^* | p \notin E\} = \{p \in I^* | f(s_0, p) \notin \Sigma_f\} = \{p \in I^*, f(s_0, p) \in C(\Sigma_f)\}.$$

Prin urmare $C(E) = L(AF_c)$ unde $AF_c = (\Sigma, I, f, s_0, C(\Sigma_f)\}$, adică $C(E)$ este un limbaj regulat.□

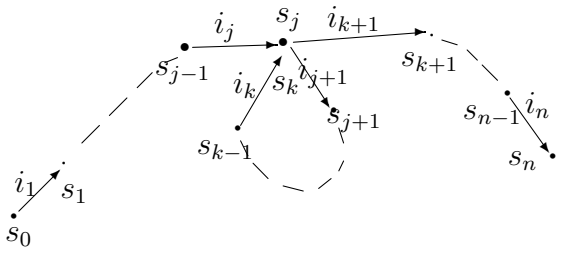

Figura 2.3: Traiectoria automatului finit

2.3 Lema de pompare pentru limbaje regulate

Sub această denumire (sau lema *uvw*) este cunoscută o proprietate a limbajelor regulate (ca şi a altor familii de limbaje) care ne permit să descompunem cuvintele suficient de lungi ale limbajului în forma *uvw* şi să multiplicăm subcuvântul v de un număr arbitrar de ori, obţinând cuvinte care aparţin de asemenea limbajului. Cu alte cuvinte, putem să "pompăm" în cuvântul dat o anumită parte a sa. Astfel de leme se utilizează deseori pentru a rezolva probleme de neapartenenţă, adică pentru a arăta că un anumit limbaj nu aparţine unei familii date de limbaje.

Lema 2.2 *Fie E un limbaj regulat şi $AF = (\Sigma, I, f, s_0, \Sigma_f)$ automatul finit care îl recunoaşte. Dacă $p \in E$ şi $|p| \geq card(\Sigma)$ atunci p se descompune în forma $p = uvw,\ v \neq \lambda$ şi $uv^m w \in E,\ \forall m \in N$.*

Demonstraţie. Fie $p = i_1 \dots i_n,\ n \geq card(\Sigma)$; fie $s_0, s_1, \dots, s_n$ stările parcurse de automat la citirea cuvântului p. Atunci, $s_j = f(s_{j-1}, i_j),\ j = \overline{1,n},\ s_n \in \Sigma_f$. Există în mod necesar două stări egale, $s_j = s_k,\ j < k$. Traiectoria va avea o buclă (vezi figura 2.3).

Descompunem cuvântul p în forma $p = uvw$ unde $u = i_1 \dots i_j,\ v = i_{j+1} \dots i_k,\ w = i_{k+1} \dots i_n$. Este clar că $v \neq \lambda$, căci $j < k$. Pe de altă parte, putem parcurge traiectoria făcând de mai multe ori bucla, adică

$$f(s_0, uv^m w) = s_n \in \Sigma_f.$$

Prin urmare $uv^m w \in E$.□

Consecinţă 2.3 *Incluziunea $\mathcal{L}_3 \subseteq \mathcal{L}_2$ este strictă.*

În adevăr, fie limbajul $L_2 = \{0^n 1^n | n \geq 1\}$. Ştim că acest limbaj este de tipul 2 şi că poate fi generat de gramatica $G = (\{A\}, \{0,1\}, A, \{A \to 0A1|01\})$. Să arătăm că L_2 nu este de tipul 3.

Să presupunem că L_2 este de tipul 3 şi fie $AF = (\Sigma, I, f, s_0, \Sigma_f)$ automatul finit care îl recunoaşte. Cum L_2 conţine cuvinte oricât de lungi, fie $p \in L_2$

astfel încât $p \geq card(\Sigma)$. Conform lemei de pompare, p se descompune în forma $p = uvw$, $v \neq \lambda$ şi $uv^m w \in E$. Putem avea una din situaţiile:

$$(1)\ p = \underbrace{0\ldots0}_{u}\underbrace{0\ldots0}_{v}\underbrace{0\ldots01\ldots1}_{w},$$
$$(2)\ p = \underbrace{0\ldots01\ldots1}_{u}\underbrace{1\ldots1}_{v}\underbrace{1\ldots1}_{w},$$
$$(3)\ p = \underbrace{0\ldots0}_{u}\underbrace{0\ldots1}_{v}\underbrace{1\ldots1}_{w}.$$

Primele două cazuri nu pot avea loc deoarece multiplicându-l pe v, numărul de simboluri 0 şi 1 nu s-ar păstra egal. În al treilea caz, luând de exemplu, $m = 2$ obţinem

$$p_2 = 0\ldots00\ldots10\ldots11\ldots1 \in L_2$$

ceea ce din nou nu este posibil, întrucât se contrazice structura cuvintelor lui L_2. Prin urmare L_2 nu este de tipul 3.□

Observaţie. Este interesant de observat că limbajele simple de forma lui L_2 sunt semnificative pentru clasele din clasificarea Chomsky. Astfel

$$L_1 = \{a^n | n \geq 1\},\ L_1 \in \mathcal{L}_3;$$
$$L_2 = \{a^n b^n | n \geq 1\},\ L_2 \in \mathcal{L}_2,\ L_2 \notin \mathcal{L}_3;$$
$$L_3 = \{a^n b^n c^n | n \geq 1\},\ L_3 \in \mathcal{L}_1(?),\ L_3 \notin \mathcal{L}_2;$$

Ne-am putea aştepta ca limbajul L_3, un exemplu analog lui L_2, să fie de tip 1, adică $L_3 \in \mathcal{L}_1,\ L_3 \notin \mathcal{L}_2$. În adevăr, se poate arăta că $L_3 \notin \mathcal{L}_2$, dar după cunoştinţa autorului, aparenţa $L_3 \in \mathcal{L}_1$ este o problemă deschisă.

Consecinţă 2.4 *Fie E un limbaj regulat şi $AF = (\Sigma, I, f, s_0, \Sigma_f)$ automatul finit care îl recunoaşte. Atunci E este infinit dacă şi numai dacă există $p \in E$ astfel încât $|p| \geq card(\Sigma)$.*

Dacă limbajul este infinit, este clar că există $p \in E$ cu $|p| \geq card(\Sigma)$. Invers, dacă există $p \in E$ cu $|p| \geq card(\Sigma)$ atunci $p = uvw,\ v \neq \lambda$ şi $uv^m w \in E, \forall m \in N$, deci limbajul este infinit.□

2.4 Expresii regulate

Expresii regulate şi limbaje reprezentate Fie V un alfabet. Expresiile regulate sunt cuvinte peste alfabetul $V \cup \{\bullet, *, |\}$, unde simbolurile suplimentare introduse le vom considera operatori:"$|$"-sau, "$\bullet$"-produs, "$*$"-închidere. Expresiile regulate se definesc astfel:

(1) λ este o expresie regulată;

(2) pentru orice $a \in V$, cuvântul a este o expresie regulată;

(3) dacă R şi S sunt expresii regulate, atunci $R|S$ $R \bullet S$ şi R^* sunt expresii regulate.

Pentru a pune în evidenţă ordinea de aplicare a operatorilor vom utiliza paranteze; de exemplu $(R|S) \bullet P$. Vom considera că operatorul $*$ are ponderea cea mai mare, apoi operatorul $\bullet$ şi $|$ ponderea cea mai mică. Deci prin $R|S^*$ vom înţelege $R|(S^*)$.

Observaţie. Expresiile regulate se pot defini cu ajutorul gramaticii $G = (\{E, T, F\}, V \cup \{|, \bullet, *, (,)\}, E, P)$ unde

$$P = \begin{cases} E \to E|T \mid E \bullet T \mid T, \\ T \to T^* \mid F \\ F \to (E) \mid a \mid \lambda. \end{cases}$$

Unei expresii regulate îi putem asocia un anumit limbaj peste V; vom spune că expresia regulată reprezintă (desemnează, notează) acel limbaj. Modul în care asociem un limbaj unei expresii regulate este

(1) λ reprezintă limbajul $\{\lambda\}$,

(2) a reprezintă limbajul $\{a\}$,

(3) dacă R şi S sunt expresii regulate şi reprezintă respectiv limbajele L_R şi L_S atunci

(i) $R|S$ reprezintă limbajul $L_R \cup L_S$;

(ii) $R \bullet S$ reprezintă limbajul $L_R L_S$;

(iii) R^* reprezintă limbajul $(L_R)^*$.

Fie R, S, P trei expresii regulate şi L_R, L_S, L_P limbajele reprezentate. Avem :

$$L_{(R|S)|P} = (L_R \cup L_S) \cup L_P = L_R \cup (L_S \cup L_P) = L_{R|(S|P)},$$

întrucât operaţia de reuniune este asociativă. Vom scrie $(R|S)|P = R|(S|P)$. În mod analog se pot obţine şi alte proprietăţi. De exemplu:

$S|R = S|R$,
$R|R = R$,
$(R \bullet S) \bullet P = R \bullet (S \bullet P)$,
$R \bullet (S|P) = (R \bullet S)|(R \bullet P)$, etc.

În cazul în care nu există pericol de confuzie, vom nota cu L (fără indice) limbajul reprezentat de o anumită expresie regulată.

Exemple.

1. $R = a^*$; $L = \bigcup_{j=0}^{\infty} \{a^j\} = \{\lambda, a, a^2, \ldots\}$.
2. $R = aa^*$; $L = a \bullet \{\lambda, a, a^2, \ldots\} = \{a, a^2, a^3 \ldots\}$.
3. $R = (a|b)^*$; $L = (L_a \cup L_b)^* = (\{a\} \cup \{b\})^* = \{a, b\}^*$;

$\{a,b\}^* = \{\lambda\} \cup \{a,b\}^1 \cup \{a,b\}^2 \cup \ldots = \{\lambda, a, b, aa, ab, ba, bb, \ldots\}$, adică $(a|b)^*$ reprezintă mulţimea tuturor cuvintelor peste alfabetul $\{a,b\}$.

4. $R = a|ba^*$; $L = \{a\} \cup \{b\} \bullet \{\lambda, a, a^2, \ldots\} = \{a, b, ba, ba^2, \ldots\}$.

Limbajele reprezentate de expresii regulate constituie o anumită familie de limbaje; o vom nota cu $\mathcal{L}_{lr}$. Apare următoarea problemă: *care este poziţia acestei familii în ierarhia Chomsky*? Vom arăta că $\mathcal{L}_{lr}$ coincide cu familia limbajelor regulate.

2.5 Sisteme tranziţionale

Definiţie 2.3 *Un sistem tranziţional este un sistem de forma*

$ST = (\Sigma, I, f, \Sigma_0, \Sigma_f, \delta)$

unde:

Σ *este o mulţime (finită) de stări;*
I *este alfabetul de intrare;*
$f : \Sigma \times I \longrightarrow \mathcal{P}(\Sigma)$ *este funcţia de tranziţie;*
$\Sigma_0 \subseteq \Sigma$ *este mulţimea de stări iniţiale;*
$\Sigma_f \subseteq \Sigma$ *este mulţimea de stări finale;*
$\delta \subset \Sigma \times \Sigma$ *este relaţia de tranziţie .*

Exemplu. $\Sigma = \{s_0, s_1, s_2\}$, $I = \{0, 1\}$, $\Sigma_0 = \{s_0, s_1\}$, $\Sigma_f = \{s_2\}$ iar funcţia şi relaţia de tranziţie sunt date de:

f	s_0	s_1	s_2
0	$\{s_1\}$	$\{s_2\}$	$\{s_0\}$
1	λ	$\{s_0, s_1\}$	$\{s_0, s_2\}$

$\delta = \{(s_0, s_1), (s_2, s_1)\}$.

Ca şi în cazul unui automat finit putem construi diagrama de stări completată cu relaţia δ (arcele punctate). În cazul exemplului nostru diagrama de stări este prezenta în figura 2.4

Observaţie: δ fiind o relaţie, are sens δ^* (închiderea tranzitivă şi reflexivă).

Fie $i \in I \cup \{\lambda\}$; vom spune că sistemul tranziţional evoluează direct din starea s' în starea s'' dacă:

(1) $i = \lambda$ şi $(s', s'') \in \delta^*$. Pe diagrama de stări vom avea o traiectorie punctată de la starea s' la starea s'';

$$s' \longrightarrow O \longrightarrow O \longrightarrow \ldots \longrightarrow O \longrightarrow s''.$$

(2) $i \neq \lambda$ şi există $s_1, s_2 \in \Sigma$ astfel încât $(s', s_1) \in \delta$, $s_2 \in f(s_1, i)$ şi $(s_2, s'') \in \delta^*$. Pe diagrama de stări, putem ajunge din s' în s'' pe o traiectorie punctată, apoi un pas pe un arc plin şi din nou pe o traiectorie punctată.

$$s' \longrightarrow O \longrightarrow \ldots \longrightarrow s_1 \xrightarrow{i} s_2 \longrightarrow O \longrightarrow \ldots \longrightarrow s''.$$

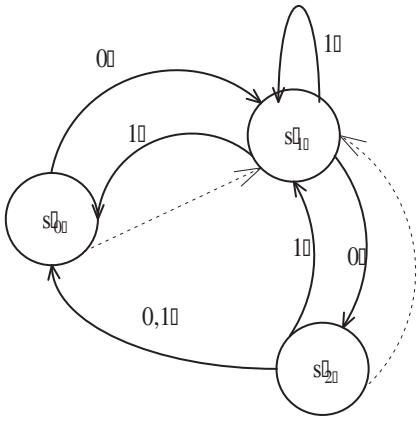

Figura 2.4: Diagrama de stări a sistemului tranziţional

Vom scrie $s' \overset{i}{\vdash} s''$.

Fie acum $p = i_1 \dots i_n$. Vom spune că sistemul evoluează din starea s' în starea s'' dacă există $s_0, s_1, \dots, s_n$ astfel încât

$$s' = s_0 \overset{i_1}{\vdash} s_1 \overset{i_2}{\vdash} \dots \overset{i_n}{\vdash} s_n = s''.$$

Vom scrie $s' \overset{p}{\vdash} s''$.

Definiţie 2.4 *Limbajul recunoscut de un sistem tranziţional ST este*

$$L(ST) = \{p | p \in I^*,\ \exists s_0 \in \Sigma_0,\ s_0 \overset{p}{\vdash} s,\ s \in \Sigma_f\}.$$

Vom nota cu $\mathcal{L}_{ST}$ familia limbajelor recunoscute de sisteme tranziţionale. Este evident că orice automat finit este un sistem tranziţional particular în care $card(\Sigma_0) = 1$ iar $\delta = \emptyset$ (nu există arce punctate). Prin urmare $\mathcal{R} \subseteq \mathcal{L}_{ST}$.

Teorema 2.6 $\mathcal{R} = \mathcal{L}_{ST}$.

Demonstraţie. Evident, trebuie să arătăm incluziunea $\mathcal{L}_{ST} \subseteq \mathcal{R}$. Fie $ST = (\Sigma, I, f, \Sigma_0, \Sigma_f, \delta)$ un sistem tranziţional. Construim automatul finit $AF = (\mathcal{P}(\Sigma), I, f', \Sigma_0, \Sigma'_f)$ unde

$$f'(Z, i) = \{s | \exists s' \in Z,\ s' \overset{i}{\vdash} s\},$$
$$\Sigma'_f = \{Z | Z \cap \Sigma_f \neq \emptyset\}.$$

Fie $p = i_1 \dots i_n \in L(ST)$ şi fie următoarea evoluţie a sistemului tranziţional

$$s_0 \overset{i_1}{\vdash} s_1 \overset{i_2}{\vdash} \dots \overset{i_n}{\vdash} s_n \in \Sigma_f.$$

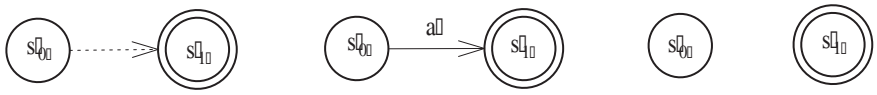

Figura 2.5: Sistemele tranziţionale ce recunosc limbajele λ, a,$\emptyset$.

Putem construi o traiectorie a lui AF de forma

$$\Sigma_0 \xrightarrow{i_1} Z_1 \xrightarrow{i_2} \ldots \xrightarrow{i_n} Z_n,$$

unde $Z_1 = f'(\Sigma_0, i_1)$, $Z_k = f'(Z_{k-1}, i_k)$, $k = 2, \ldots, n$. Să observăm că $s_0 \in \Sigma_0$ şi că dacă $s_{k-1} \in Z_{k-1}$, atunci conform definiţiei funcţiei f', avem $s_k \in f'(Z_{k-1}, i_k) = Z_k$. Asftel, $s_k \in Z_k, k = 1, \ldots, n$; pentru $k = n$ avem $s_n \in Z_n$ şi cum $s_n \in \Sigma_f$ rezultă că $Z_n \cap \Sigma_f \neq \emptyset$, adică $Z_n \in \Sigma'_f$. Deci automatul ajunge într-o stare finală, $p \in L(AF)$ şi $L(ST) \subseteq L(AF)$.

Incluziunea inversă se arată în mod analog.□

Construcţia sistemelor tranziţionale pentru expresii regulate. Fiind dată o expresie regulată putem întotdeauna construi un sistem tranziţional care recunoaşte limbajul reprezentat de expresia respectivă.

Construcţia se face cu ajutorul diagramelor de stări.

Sistemele tranziţionale (diagramele de stări) corespunzătoare expresiilor regulate λ, a şi $\emptyset$ sunt prezentate în figura 2.5.

Dacă R şi S sunt expresii regulate şi notăm cu ST_R şi ST_S sistemele tranziţionale corespunzătoare, atunci sistemele tranziţionale pentru $R|S$, $R \bullet S$ şi R^* sunt redate în figura 2.6.

În acest mod putem construi succesiv (recurent) un sistem tranziţional corespunzător unei expresii regulate.

Exemplu. Sistemul tranziţional corespunzător expresiei $R = a|b \bullet a^*$ este redat în figura 2.7.

Consecinţă Dându-se o expresie regulată, putem construi sistemul tranziţional care recunoaşte limbajul reprezentat de expresia respectivă. Cum orice limbaj recunoscut de un sistem tranziţional este regulat, rezultă că limbajele reprezentate de expresii regulate sunt limbaje regulate.

2.6 Analiza lexicală

Procesul de analiză lexicală este o fază a procesului de compilare în care se determină unităţile lexicale (cuvintele, atomii) ale unui program sursă, se furnizează codurile interne ale acestora şi se detectează eventualele erori lexicale. Analizorul lexical mai poate efectua o serie de operaţii auxiliare precum: eliminarea blank-urilor, ignorarea comentariilor, diferite conversiuni ale unor date, completarea

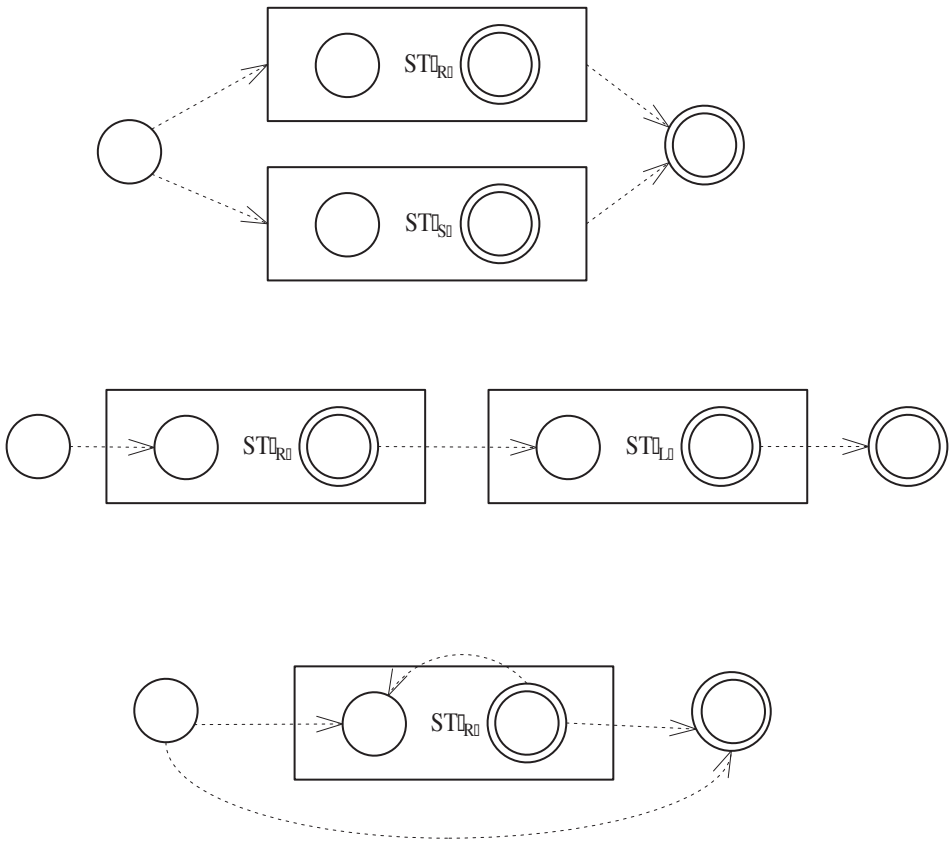

Figura 2.6: Sistemele tranziţionale ce recunosc limbajele $R|S$, $R \bullet S$ şi R^*.

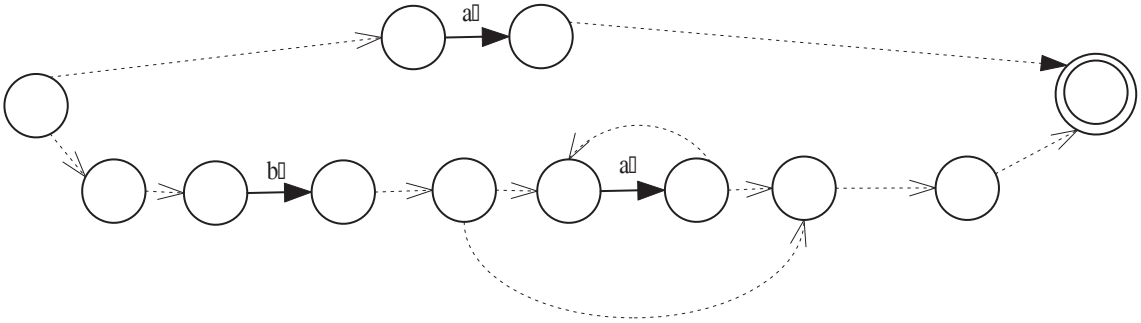

Figura 2.7: Sistemul tranziţional corespunzător expresiei regulate $R = a|b \bullet a^*$.

tabelelor compilatorului, gestiunea liniilor textului sursă.
Unităţi lexicale O unitate lexicală (Lexic = vocabular; totalitatea cuvintelor unei limbi) este o secvenţă din textul sursă care are o anumită unitate logică. Definiţia riguroasă a unităţilor lexicale ale unui limbaj particular se dă la definirea limbajului de programare respectiv. De obicei, în majoritatea limbajelor de programare, unităţile lexicale sunt: *cuvinte cheie, identificatori, constante, operatori, delimitatori.* Din punctul de vedere al analizei lexicale şi al modului de prelucrare, unităţile lexicale pot fi de două categorii:

- Unităţi lexicale simple, sunt unităţi lexicale care nu comportă atribute suplimentare, de exemplu, cuvintele cheie, operatorii;
- Unităţi lexicale compuse (atributive), sunt unităţi lexicale care comportă anumite atribute suplimentare, de exemplu, identificatorii şi constantele. Atributele sunt informaţii specifice, de exemplu, tipul identificatorului sau al constantei (întreg, real, etc.). Este necesară specificarea tipului unui identificator sau al unei constante din startul programului deoarece structura programului obiect sau reprezentarea internă a constantelor depinde de acest tip.

Reprezentarea internă a unităţilor lexicale se face în funcţie de categoria lor. Cele simple se reprezintă printr-un cod specific(număr întreg). Unităţile lexicale compuse se reprezintă prin cod şi informaţii (de natură semantică) asupra sa. De obicei compilatoarele utilizează tabele pentru stocarea atributelor (tabel de constante, tabel de variabile, tabel de etichete, etc.). In acest caz unitatea lexicală se reprezintă intern printr-un cod urmat de o referinţă într-un tabel. Informaţia conţinută de tabel ar putea fi pentru constante: cod, tip, valoare, iar pentru identificatori: cod, tip, indicator de iniţializare.

Este posibil ca o clasă de unităţi lexicale simple să se reprezinte printr-un cod unic şi un atribut pentru distingerea în cadrul clasei. De exemplu, operatorii aritmetici cu aceiaşi prioritate au o tratare similară din punct de vedere al analizei sintactice. Lista unităţilor lexicale şi definiţia riguroasă a acestora se dă la proiectarea compilatorului.

Un exemplu de unităţi lexicale şi coduri asociate ar putea fi cele din figura 2.8.

Analizorul primeşte textul sursă şi produce şirul de unităţi lexicale în codificarea internă. De exemplu secvenţa de text sursă următoare:

```
{if (unu < 2) return 0;
  a=33;
  }
```

va produce sirul de unităţi lexicale
LBRACE If LPAR [ID,22] [opr,1] [NUM, 40], RPAR RETURN [NUM, 42] SEMI [ID,24] opAssign [NUM,44] SEMI RBRACE.

Unitate lexicală	COD	ATRIBUT	Exemplu
if	if = 1	-	if, If, IF
else	else = 2	-	else ElSe
identificator	ID = 3	referinţă	Nelu v tabel
constantă întreagă	NUM = 4	referinţă	4 -3 233
constantă reală	FLOAT = 5	referinţă	4.32 -3.233
+	op = 6	1	
-	op = 6	2	
×	op = 6	3	
/	op = 6	4	
<	opr = 7	1	
>	opr = 7	2	
<=	opr = 7	3	
>=	opr = 7	4	
(	LPAR = 8	-	
)	RPAR = 9	-	
{	LBRACE = 10	-	
}	RBRACE = 11	-	

Figura 2.8: Coduri asociate unităţilor lexicale

In lista codurilor interne găsite atributul identificatorului este adresa relativa din tabela de identificatori, analog atributele constantelor numerice sunt adrese relative în tabelele de constante.

Majoritatea limbajelor evoluate conţin şi secvenţe de text care nu sunt unităţi lexicale, dar au acţiuni specifice asociate . De exemplu:

```
comentarii                  /* text */

directive de preprocesare   #include<stdio.h>
                            #define MAX 5.6
```

Înaintea analizei lexicale (sau ca subrutină) se preprocesează textul şi abia apoi se introduce rezultatul în analizorul lexical.

Specificarea unităţilor lexicale Definirea riguroasă a unităţilor lexicale se face de către proiectantul limbajului. O posibilitate de descriere este limbajul natural. De exemplu, pentru **C** şi **JAVA**:

"*Un identificator este o secvenţa de litere şi cifre: primul caracter trebuie să fie literă. Liniuţa de subliniere contează ca literă. Literele mari şi mici sunt diferite. Dacă şirul de intrare a fost împărţit în unităţi lexicale până la un caracter dat, noua unitate lexicală se consideră astfel încât să includă cel mai lung şir de caractere ce poate constitui o unitate lexicală. Spaţiile, taburile, newline şi comentariile sunt ignorate cu excepţia cazului când servesc la separarea unităţilor lexicale. Sunt necesare spaţii albe pentru separarea identificatorilor, cuvintelor cheie şi a constantelor.*"

Orice limbaj rezonabil poate fi folosit pentru implementarea unui analizor lexical.

Unităţile lexicale se pot specifica cu ajutorul limbajelor regulate, deci folosind gramatici de tipul 3 sau expresii regulate ce notează limbajele. Ambele specificaţii conduc la construirea de automate finite echivalente, care se pot uşor programa. În cele ce urmează, vom folosi ambele variante de specificare pentru un set uzual de unităţi lexicale întâlnit la majoritatea limbajelor evoluate.

Considerăm gramatica regulată ce generează identificatori, constante întregi, cuvinte cheie, operatori relaţionali şi aritmetici.

$$G: \begin{cases} < ul > \rightarrow < id > | < num > | < cc > | < op > | < opr > \\ < id > \rightarrow l < id1 > | l, < id1 > \rightarrow l < id1 > | c < id1 > | l | c \\ < num > \rightarrow c < num > | c \\ < cc > \rightarrow if | do | else | for \\ < op > \rightarrow + | - | * | / \\ < opr > \rightarrow < | <= | > | >= \end{cases},$$

unde l-litera, c-cifra.

Pornind de la această gramatică se poate construi una echivalentă în formă normală, apoi se extrage funcţia de evoluţie a automatului finit determinist echivalent ce recunoaşte unităţile lexicale.

Descrieri echivalente ale unităţilor lexicale cu ajutorul expresiilor regulate sunt

```
cuvinte cheie = if | do | else | for

identificatori = ( a|b|c|...z )( a|b|c|...z|0|1|...|9 )*

Numar = ( 0|1|...|9 )( 0|1|...|9 )*

Operatori aritmetici = + | - | * | /

Operatori relationali = < | <= | > | >=
```

Limbajul generat de gramatica precedentă se obţine prin suma expresiilor regulate. Menţionăm că există programe specializate (Lex, Flex, JavaCC) ce generează un analizor lexical (în C sau Java) pornind de la expresiile regulate. Sintaxa folosită în scrierea expresiilor regulate este dependentă de programul folosit.

Programarea unui analizor lexical Realizarea efectivă a unui analizor revine la simularea funcţionării unui automat finit. O variantă de programare este ataşarea unei secvenţe de program la fiecare stare a automatului. Dacă starea nu este stare finală atunci secvenţa citeşte următorul caracter din textul sursă şi găseşte următorul arc din diagrama de stări. Depinzând de rezultatul căutării se transferă controlul altei stări sau se returnează eşec (posibilă eroare lexicală). Dacă starea este finală atunci se apelează secvenţa de returnare a codului unităţii lexicale şi eventuala instalare a unităţii lexicale în tabelele compilatorului.

Pentru simplificarea implementării se caută următoarea unitate lexicală prin încercarea succesivă a diagramelor corespunzătoare fiecărei unităţi lexicale (într-o ordine prestabilită). Eroarea lexicală se semnalează doar atunci când toate încercările se încheie cu eşec.

De obicei textul sursă conţine şi secvenţe ce se pot descrie cu ajutorul expresiilor regulate, dar care nu sunt unităţi lexicale (de exemplu comentariile). Aceste secvenţe nu vor genera cod lexical, dar au asociate diverse acţiuni specifice. Pentru a evita apariţia unor caractere necunoscute in textul sursă se consideră şi limbajul ce constă din toate simbolurile ASCII. Astfel, indiferent de unde începe analiza textului, programul de analiză lexicală găseşte o potrivire cu o descriere. Spunem că specificaţia este **completă**.

Există posibilitatea ca mai multe secvenţe cu aceeaşi origine să corespundă la diferite descrieri ale unităţilor lexicale. Se consideră unitate lexicală cel mai lung şir ce se potriveşte unei descrieri (**longest match rule**). Dacă sunt două reguli care se potrivesc la acelaşi şir de lungime maximă atunci se consideră o prioritate asupra descrierilor (**rule priority**).

De exemplu, in textul următor,

```
i  if  if8
```

unităţile lexicale delimitate vor fi **i**–identificator, **if**–cuvânt cheie, **if8**–identificator. Regula de prioritate se aplică pentru potrivirea lui if cu identificator şi cuvânt cheie, iar criteriul de lungime maximă pentru if8.

Pentru depistarea celei mai lungi potriviri, din punt de vedere al programării analizorului, este suficient să prevedem un pointer suplimentar pe caracterul ce corespunde ultimei stări finale atinse pe parcursul citirii.

Studiu de caz. Se consideră problema realizării unui analizor lexical ce delimitează într-un text sursă cuvinte din limbajul ce conţine identificatori, cuvinte cheie (pentru simplificare folosim doar cuvântul cheie if), constante numerice (întregi fără semn). De asemenea se face salt peste spaţiile albe şi se ignoră comentariile. Presupunem că un comentariu începe cu două caractere slash şi se termină cu newline. Orice caracter ce nu se potriveşte descrierii este semnalat ca şi caracter ilegal in text.

Etapa I. Descriem secvenţele cu ajutorul expresiilor regulate

```
IF = "if"

ID = (a|b|c|...|z)(a|b|c|...|z|0|1|...|9)*

NUM = (0|1|...|9)(0|1|...|9)* = (0|1|...|9)+

WS = (\n|\t|" ")+

COMMENT = "//"(a|b|c|...|z|0|1|...|9|" ")*\n

ALL = a|b|...|z|0|...|9|\t| ...  toate caracterele ASCII
```

Etapa II. Corespunzător expresiilor avem următoarele automate finite deterministe echivalente, prezentate în figura 2.9 (stările au fost notate prin numere întregi):

Etapa III. Se construieşte sistemul tranziţional (vezi figura 2.10) ce recunoaşte limbajul reuniune, adăugând o nouă stare iniţială (notată cu 1), pe care o conectăm prin arce punctate (ce corespund λ-tranziţiilor). Construcţia provine din *legarea* sistemelor tranziţionale *în paralel*. S-au renumerotat stările sistemului tranziţional, asignând nume simbolice stărilor finale.

Etapa III. Construcţia automatului finit determinist general ce recunoaşte reuniunea limbajelor. Pentru aceasta sistemul tranziţional se transformă cu teorema de echivalenţă în automat finit determinist (practic se trece de la stări ale sistemului tranziţional la submulţimi de stări, ce devin stările automatului finit).

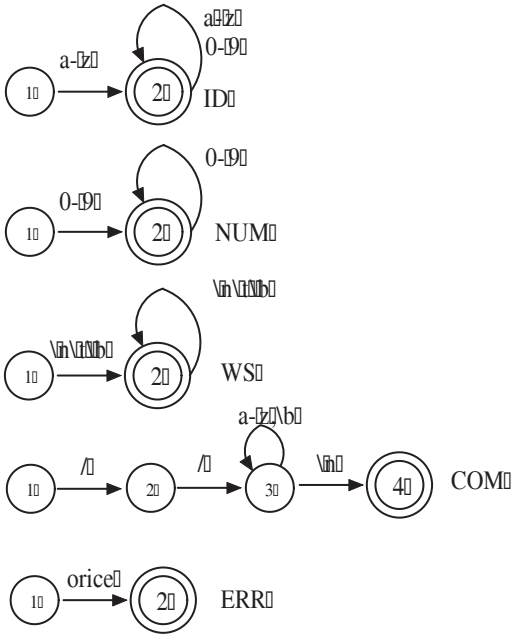

Figura 2.9: Automatele finite corespunzătoare expresiilor regulate

În cazul particular al automatului nostru, diagrama de stări este dată în figura 2.11.

Etapa IV. *Programarea analizorului lexical.*

Automatul finit obţinut are stările finale asociate cu clasele de cuvinte recunoscute. Se asociază acţiuni stărilor finale, corespunzător definiţiilor unităţilor lexicale (de exemplu pentru constante numerice se generează reprezentarea interna, se memorează în tabelul de constante şi se returnează codul unităţii lexicale NUM). Pentru programarea analizorului se folosesc trei variabile de tip pointer in textul sursă: **F**irst**S**ymbol, **C**urrent**S**ymbol, **L**ast**F**inal**S**ymbol, ce reţin indicele caracterului de început al secvenţei, indicele caracterului ce urmează la citire, indicele ultimului caracter ce corespunde atingerii unei stări finale. Cei trei pointeri au fost reprezentaţi prin semnele grafice $|$, $\perp$ respectiv $\top$. De asemenea considerăm o variabilă *State*, ce reţine starea curentă a automatului.

In tabelul 2.12 este indicată evoluţia analizorului lexical (inclusiv acţiunile asociate) pentru cazul analizei următorului text sursă.

```
if if8%// ha\n
```

Pentru simplificarea codificării, stările automatului finit determinist au fost redenumite prin numere întregi începând cu starea iniţială 1, starea $\{3, 6, 16\} = 2$, $\{4, 6\} = 3$, $\{6, 16\} = 4$, ş.a.m.d. de la stânga la dreapta şi de sus în jos. De obicei

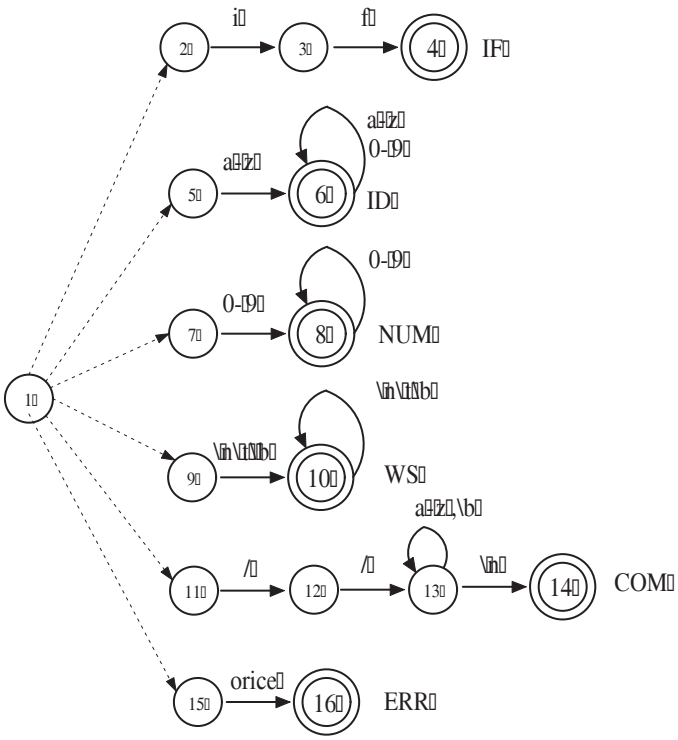

Figura 2.10: Sistemul tranziţional ce recunoaşte reuniunea limbajelor

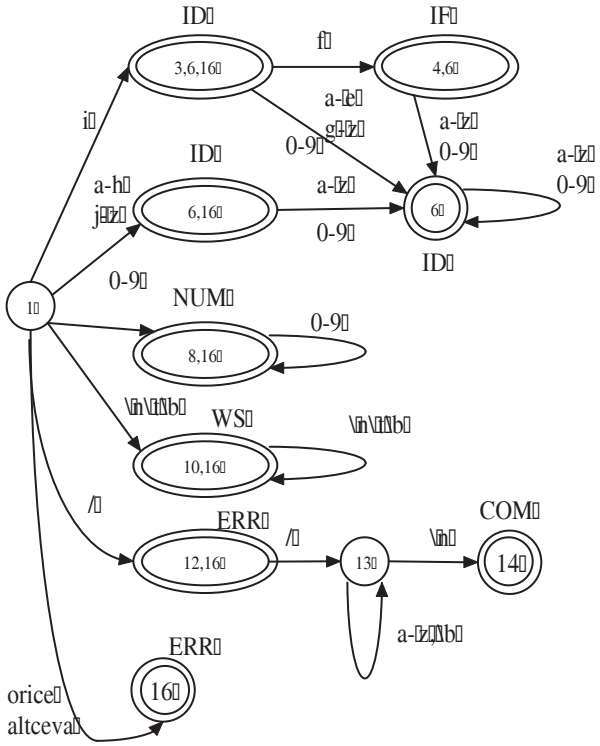

Figura 2.11: Automatul finit determinist ce recunoaşte reuniunea limbajelor

Last Final	Current State	Current Input	Accept Action
0	1	$\vert^{\top}_{\perp} i\ f\quad i\ f\ 8\ \%\ /\ /\quad h\ a\ \backslash n$	
2	2	$\vert\ i^{\top}_{\perp} f\quad i\ f\ 8\ \%\ /\ /\quad h\ a\ \backslash n$	
3	3	$\vert\ i\ f^{\top}_{\perp}\quad i\ f\ 8\ \%\ /\ /\quad h\ a\ \backslash n$	return $cc =< if >$
	0	$\vert\ i\ f^{\top}\ {}_{\perp} i\ f\ 8\ \%\ /\ /\quad h\ a\ \backslash n$	*resume*
0	1	$i\ f\vert^{\top}_{\perp}\ i\ f\ 8\ \%\ /\ /\quad h\ a\ \backslash n$	
7	7	$i\ f\vert\ {}^{\top}_{\perp} i\ f\ 8\ \%\ /\ /\quad h\ a\ \backslash n$	
	0	$i\ f\vert\ {}^{\top} i_{\perp} f\ 8\ \%\ /\ /\quad h\ a\ \backslash n$	*resume*
0	1	$i\ f\ \vert^{\top}_{\perp} i\ f\ 8\ \%\ /\ /\quad h\ a\ \backslash n$	
2	2	$i\ f\ \vert\ i^{\top}_{\perp} f\ 8\ \%\ /\ /\quad h\ a\ \backslash n$	
3	3	$i\ f\ \vert\ i\ f^{\top}_{\perp} 8\ \%\ /\ /\quad h\ a\ \backslash n$	
5	5	$i\ f\ \vert\ i\ f\ 8^{\top}_{\perp}\ \%\ /\ /\quad h\ a\ \backslash n$	return $id =< if8 >$
	0	$i\ f\ \vert\ i\ f\ 8^{\top}\ \%_{\perp}/\ /\quad h\ a\ \backslash n$	*resume*
0	1	$i\ f\quad i\ f\ 8\vert^{\top}_{\perp}\ \%\ /\ /\quad h\ a\ \backslash n$	
11	11	$i\ f\quad i\ f\ 8\vert\ \%^{\top}_{\perp}/\ /\quad h\ a\ \backslash n$	print ("illegal character: %");
	0	$i\ f\quad i\ f\ 8\vert\ \%^{\top}/_{\perp}/\quad h\ a\ \backslash n$	*resume*
0	1	$i\ f\quad i\ f\ 8\ \%\vert^{\top}_{\perp}/\ /\quad h\ a\ \backslash n$	
0	8	$i\ f\quad i\ f\ 8\ \%\vert^{\top}/_{\perp}/\quad h\ a\ \backslash n$	
0	9	$i\ f\quad i\ f\ 8\ \%\vert^{\top}/\ /_{\perp}\quad h\ a\ \backslash n$	
...	...	...	

Figura 2.12: Evoluţia analizorului lexical pentru textul *if if8*%// ha\n

```
int edges[][] = {  /* ...  0 1 2 ...  e f g h i j ...   */

/* state 0 */    {0,0, ...,0,0,0, ...,0,0,0,0,0,0, ... },

/* state 1 */    {0,0, ...,6,6,6, ...,4,4,4,4,2,4, ... },

/* state 2 */    {0,0, ...,5,5,5, ...,5,3,5,5,5,5, ... },

etc

}
```

Figura 2.13: Reprezentarea funcţiei de evoluţie a automatului finit

funcţia de evoluţie asociată automatului finit determinist se memorează sub forma unui tablou bidimensional de întregi, ca în figura 2.13. Starea 0 este asociată cu blocarea automatului. Ajungerea în această stare echivalează cu găsirea ultimei unităţi lexicale, între pointerii | şi $\top$. Se execută acţiunea asociată stării finale şi se reia căutarea (*resume*) următoarei unităţi lexicale începând cu caracterul imediat următor pointerului $\top$.

Observaţie: Cele mai costisitoare operaţiuni (ca timp) din analiza lexicală sunt ignorarea comentariilor şi tratarea erorilor lexicale. Primele generatoare automate de analizoare lexicale şi sintactice au apărut în anii '70 şi au fost incluse în sistemul de operare Unix.

2.7 Probleme propuse

1. Construiţi automate finite pentru recunoaşterea limbajelor:
   (a) $L = \{PSDR, PNL, PUNR\}$;
   (b) $L = \{w|$ şiruri de 0 şi 1 terminate cu 1 $\}$;
   (c) $L = \{w|w$ identificator PASCAL $\}$;
   (d) $L = \{w|w$ constantă întreagă cu semn în PASCAL $\}$;
   (e) $L = \{w \in \{0,1\}^* | w$ multiplu de 3 $\}$;
   (f) $L = \{a^i b^j | i, j > 0\}$;
   (g) $L = \emptyset$.
2. Construiţi automate finite echivalente cu gramaticile de tipul trei de la problema 1 capitolul 1.

3. Construiţi automate finite deterministe echivalente cu cele nedeterministe obţinute la problema precedentă.

4. Găsiţi gramatici regulate echivalente cu automatele de la problema 1.

5. Folosind lema de pompare pentru limbaje regulate dovediţi că următoarele limbaje nu sunt regulate:

   (a) $L = \{0^{i^2} | i \geq 1\}$;
   (b) $L = \{0^{2^n} | n \geq 1\}$;
   (c) $L = \{0^n | n$ este număr prim $\}$;
   (d) $L = \{0^m 1^n 0^{m+n} | m \geq 1, n \geq 1\}$;

6. Specificaţi limbajele denotate de următoarele expresii regulate:

   (a) $(11|0)^*(00|1)^*$;
   (b) $(1|01|001)^*(\lambda|0|00)$;
   (c) $10|(0|11)0^*1$;
   (d) $((0|1)(0|1))^*$;
   (e) $01^*|1$;
   (f) $((11)^*|101)^*$.

7. Construiţi sisteme tranziţionale ce recunosc limbajele specificate la problema precedentă. Pentru fiecare sistem tranziţional construiţi un automat finit determinist echivalent.

Capitolul 3

Limbaje independente de context

3.1 Arbori de derivare

Caracterizarea limbajelor independente de context cu ajutorul arborilor de derivare.

Una din caracteristicile de bază ale limbajelor independente de context este aceea că o derivare într-un astfel de limbaj poate fi reprezentată de un arbore, numit in acest context *arbore de derivare*. Această reprezentare este importantă în mod special pentru faptul că permite o imagine intuitivă simplă a unei derivări şi deci posibilitatea de a lucra uşor cu limbaje de tipul 2.

Vom prezenta în primul rând câteva noţiuni elementare de teoria grafurilor, cu scopul de a preciza notaţiile şi terminologia.

Un *graf orientat* $\mathcal{G}$ este o pereche $\mathcal{G} = (V, \Gamma)$ unde V este o mulţime finită iar Γ o aplicaţie $\Gamma : V \longrightarrow \mathcal{P}(V)$. Mulţimea V se numeşte mulţimea vârfurilor (nodurilor) grafului iar dacă $v_2 \in \Gamma(v_1)$, perechea (v_1, v_2) este un arc în graf; v_1 este *originea* iar v_2 este *extremitatea* arcului. Spunem că v_2 este *succesor direct* al nodului v_1, iar v_1 este *predecesor direct* al nodului v_2. Un *drum* de la vîrful v' la vârful v'' în graful $\mathcal{G}$ este o mulţime de arce $(v_1, v_2)(v_2, v_3) \dots (v_{n-1}, v_n)$ cu $v' = v_1$ şi $v'' = v_n$. Numărul $n-1$ este *lungimea drumului*. Un drum pentru care $v_1 = v_n$ se numeşte *circuit*. Un circuit de lungime 1 poartă numele de *buclă*.

Definiţie 3.1 *Un arbore orientat şi ordonat este un graf orientat $\mathcal{G}$, care satisface următoarele condiţii :*

1. $\exists v_0 \in V$ *numit* **rădăcina arborelui** *astfel încât* $v_0 \notin \Gamma(v), \forall v \in V$ *şi de la care există un drum la fiecare alt vârf al grafului;*

2. $\forall v \in V \setminus \{v_0\}$, $\exists! w$ *cu* $v \in \Gamma(w)$*; altfel spus orice vârf diferit de* v_0 *este extremitatea unui singur arc;*

3. Mulţimea succesorilor directi ai oricărui vârf $v \in V$ *este ordonată.*

Exemplu. $V = \{v_0, v_1, v_2, v_3, v_4\}$ iar funcţia Γ este dată de:

x	v_0	v_1	v_2	v_3	v_4
$\Gamma(x)$	$\{v_1, v_2\}$	$\emptyset$	$\{v_3, v_2\}$	$\emptyset$	$\emptyset$

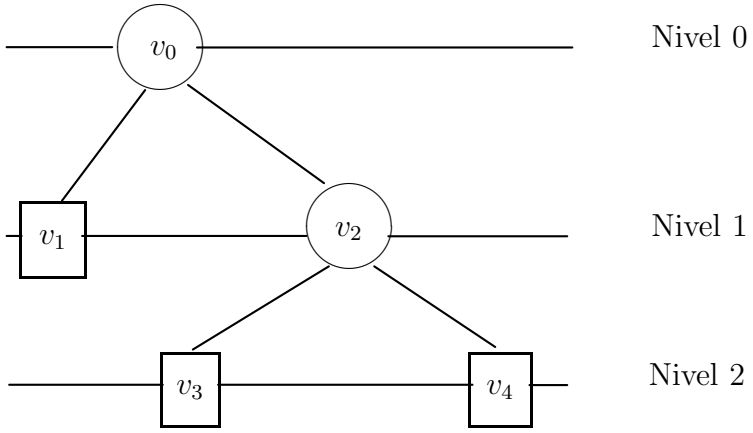

Figura 3.1: Reprezentarea grafică a arborelui $\mathcal{G} = (V, \Gamma)$

Considerând mulţimile ordonate de succesori direcţi (grafic ordinea succesorilor este de la stânga spre dreapta) şi sensul arcelor grafului de sus în jos, obţinem reprezentarea în plan a acestui arbore ca în figura 3.1

Nodurile v pentru care $\Gamma(v) = \emptyset$ se numesc noduri *terminale* (finale); celelalte se numesc *interne*. Mulţimea nodurilor terminale constituie *frontiera* arborelui. Spunem că un nod x precede pe y dacă drumul de la rădăcină la y trece prin x (y este succesor al lui x, sau y este descendent al lui x). Ordinea succesorilor direcţi ai unui nod induce în mod natural o ordine pe frontiera arborelui (daca v_1 este predecesor direct pentru v_2 atunci orice succesor al lui v_1 precede orice succesor al lui v_2, şamd). În general vom nota un arbore cu litere mari, specificând ca indici rădăcina şi frontiera; în figură $\mathcal{A}_{v_0,\, v_1v_3v_4}$. Un arbore comportă mai multe *ramuri*; în exemplu avem următoarele ramuri : v_0v_1, $v_0v_2v_3$, $v_0v_2v_4$. Un subarbore cu rădăcina x al unui arbore $\mathcal{A}_{r,w}$ se obţine prin extragerea nodului x împreună cu toţi succesorii săi (de exemplu $\mathcal{A}_{v_2,\, v_3v_4}$ este subarbore pentru $\mathcal{A}_{v_0,\, v_1v_3v_4}$).

Fie $G = (V_N, V_T, S, P)$ o gramatică de tipul 2.

Definiţie 3.2 *Un arbore de derivare în gramatica G este un arbore orientat şi ordonat cu următoarele trei proprietăţi .*

(1) Nodurile sunt etichetate cu elementele din $V_G \cup \{\lambda\}$;

(2) Dacă un nod v are descendenţi direcţi $v_1, \ldots, v_n$ (în această ordine) atunci $v \to v_1v_2 \ldots v_n \in P$.

(3) Dacă un nod are eticheta λ atunci el este singurul descendent al precedentului său direct.

Exemplu. $G = (\{A, B\}, \{a, b\}, A, P)$ unde

$$P = \{A \to aBA|Aa|a|\lambda, B \to AbB|ba|abb\}.$$

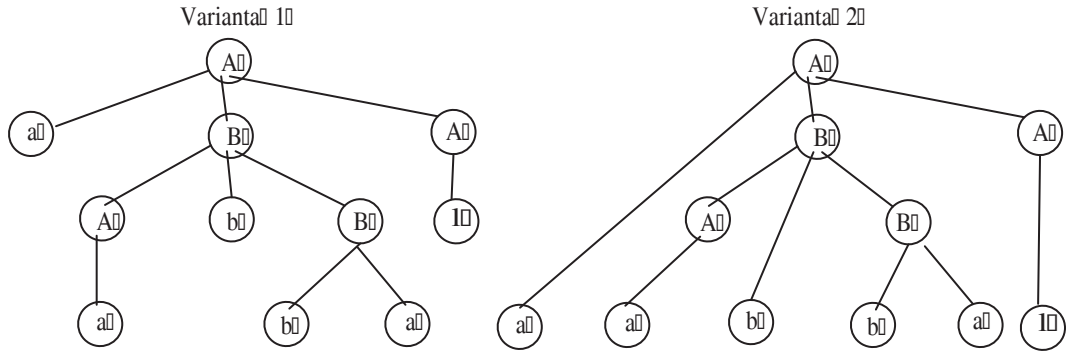

Figura 3.2: Variante de reprezentare a arborelui $\mathcal{A}_{A,\,aabba}$

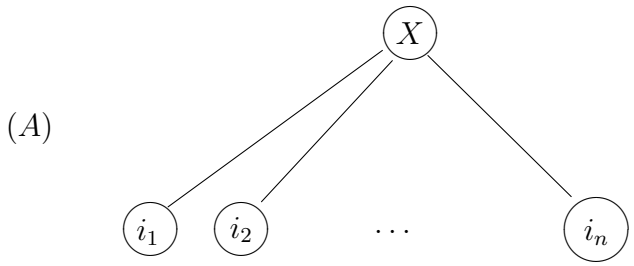

Figura 3.3: Arbore corespunzător unei derivări directe

Arborele $\mathcal{A}_{A,\,aabba}$ reprezentat în figura 3.2 (Varianta 1) este un arbore de derivare (poate fi desenat coborând frontiera pe nivelul ultim , Varianta 2):

Teorema 3.1 *Fie G o gramatică de tipul 2, $X \in V_N$ şi $p \in V_G{}^*$. Atunci $X \stackrel{*}{\Rightarrow} p$ dacă şi numai dacă există un arbore $\mathcal{A}_{X,\,p}$.*

Demonstraţie. $X \stackrel{*}{\Rightarrow} p$ implică $\exists \mathcal{A}_{X,\,p}$.

Procedăm prin inducţie asupra lungimii derivării l.
Dacă $l = 1$, $X \Rightarrow p = i_1 \dots i_n$ şi $X \to i_1 \dots i_n \in P$. Arborele din figura 3.3 corespunde cerinţelor teoremei.

Presupunem că proprietatea este adevărată pentru derivări de lungime l şi considerăm o derivare de lungime $l+1$, $X \stackrel{*}{\Rightarrow} p$. Punem în evidenţă prima derivare directă

$$X \Rightarrow X_1 \dots X_n \stackrel{*}{\Rightarrow} p$$

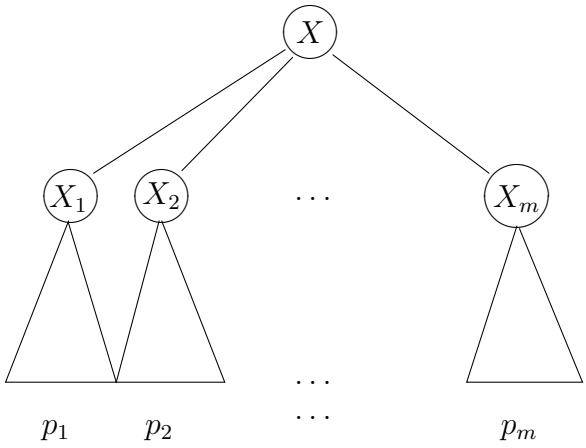

Figura 3.4: Construcţia arborelui $\mathcal{A}_{X,p_1\dots p_m}$.

Conform lemei de localizare, $p = p_1 \dots p_n$ şi $X_j \stackrel{*}{\Rightarrow} p_j$, $j = \overline{1,n}$. Putem face următoarea construcţie: conform ipotezei inductive, fiecărei derivări $X_j \stackrel{*}{\Rightarrow} p_j$ îi corespunde câte un arbore $\mathcal{A}_{X_j,p_j}$; dacă $a = X_j \in V_T$ atunci $p_j = a$; unim apoi toate nodurile X_j în nodul X plasat la nivelul zero. Obţinem astfel un arbore $\mathcal{A}_{X,p_1\dots p_m} = \mathcal{A}_{X,p}$ (vezi figura 3.4) care corespunde cerinţelor teoremei.

Pentru implicaţia , $\exists \mathcal{A}_{X,p} \Rightarrow X \stackrel{*}{\Rightarrow} p$, se parcurge o cale inversă, făcând o inducţie asupra numărului de nivele. De exemplu, dacă acest număr este 2, arborele de derivare trebuie să arate ca în 3.3 şi deci avem $X \to i_1 i_2 \dots i_n = p \in P$ şi $X \stackrel{*}{\Rightarrow} p$, etc. □

3.2 Decidabilitate şi ambiguitate în familia $\mathcal{L}_2$.

Decidabilitate. Problemele de decidabilitate sunt acele probleme în care se cere să decidem dacă un anumit fapt are sau nu loc. De obicei aceste probleme se rezolvă prin construirea unui algoritm de decizie.

Exemplu Fie gramatica

$$G = (\{A, B, C\}, \{a, b\}, A, \{A \to aA|bB|C,\ B \to abA|aC, C \to aabA\}).$$

Se poate uşor vedea că $L(G) = \emptyset$ (nu putem elimina neterminalele).
Problema: *Putem decide în general dacă limbajul generat de o gramatică de tipul 2 este vid sau nu*?

Teorema 3.2 *Faptul că limbajul generat de o gramatică de tipul 2 este nevid este decidabil.*

Demonstraţie. Vom construi un algoritm cu ajutorul căruia se poate decide dacă limbajul generat de o gramatică de tipul 2 este vid sau nu.

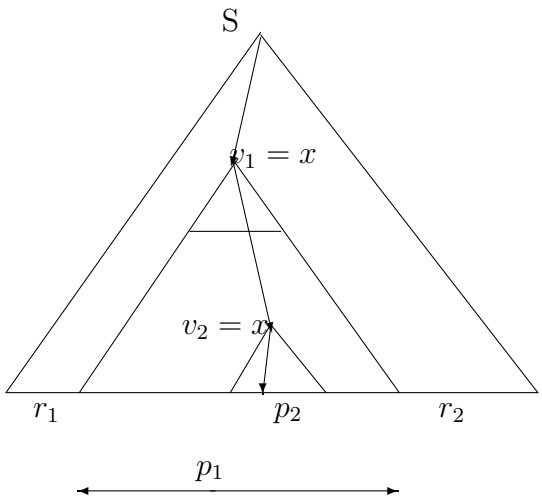

Figura 3.5:

Presupunem că limbajul nu este vid, $L(G) \neq \emptyset$ şi fie $p \in L(G)$. Există arborele $\mathcal{A}_{S,p}$. Să presupunem că în acest arbore există un drum cu două noduri interne etichetate cu acelaşi simbol, $v_1 = v_2 = X$. Descompunem arborele ca în figură 3.5

Avem $p = r_1 p_1 r_2$; evident $p_2 \in Sub(p_1)$. Efectuăm următoarea modificare: scoatem subarborele $\mathcal{A}_{v_1,p_1} = \mathcal{A}_{X,p_1}$ şi–l înlocuim cu subarborele $\mathcal{A}_{v_2,p_2} = \mathcal{A}_{X,p_2}$; obţinem în acest fel un arbore $\mathcal{A}_{S,\, r_1 p_2 r_2}$ (care este într–adevăr un arbore de derivare). Conform cu teorema de caracterizare a limbajelor de tipul 2, avem $S \stackrel{*}{\Rightarrow} r_1 p_2 r_2 \in L(G)$. Dar arborele corespunzător nu mai conţine perechea v_1, v_2 de noduri etichetate cu acelaşi simbol X.

Repetând procedeul, eliminăm pe rând nodurile de pe aceleaşi ramuri etichetate identic. În final vom obţine un arbore $\mathcal{A}_{S,\, q},\ q \in L(G)$ care are proprietatea că pe orice ramură nodurile sunt etichetate cu simboluri distincte. Ţinând cont că orice ramură are toate nodurile etichetate cu simboluri neterminale cu excepţia ultimului (de pe frontieră) care este etichetat cu un simbol terminal, rezultă că în $\mathcal{A}_{S,\, q}$ orice ramură conţine cel mult $card(V_N) + 1$ noduri. Dar mulţimea unor astfel de arbori este finită; obţinem următorul algoritm de decizie:

Construim toţi arborii cu rădăcina S şi care au proprietatea de mai sus; dacă printre aceştia se găseşte un arbore cu frontiera constituită numai din terminale, atunci $L(G) \neq \emptyset$ (evident) iar dacă nici unul din aceşti arbori nu au frontiera constituită numai din terminale, atunci $L(G) = \emptyset$. □

Ambiguitate. Fie G o gramatică de tipul 2. O derivare $S = u_0 \Rightarrow u_1 \Rightarrow \ldots \Rightarrow u_n$ în care la fiecare derivare directă se înlocuieşte simbolul neterminal cel mai din stânga (dreapta) se numeşte *derivare extrem stângă (dreaptă).* Să observăm că în particular într-o gramatică de tipul 3 orice derivare este extrem

dreaptă (scrierea drept liniară).

Definiţie 3.3 *O gramatică G de tipul 2 în care există un cuvânt $p \in L(G)$ care se poate obţine cu două derivări extrem stângi (drepte) distincte, se numeşte* **gramatică ambiguă**. *In caz contrar este* **neambiguă**.

Exemplu. Gramatica $A \to aBA|Aa|a,\ B \to AbB|ba|abb$ este ambiguă. Într-adevăr, avem

$$A \Rightarrow aBA \Rightarrow aBa \Rightarrow aAbBa \Rightarrow aAbbaa \Rightarrow aabbaa;$$

$$A \Rightarrow Aa \Rightarrow aBAa \Rightarrow aBaa \Rightarrow aabbaa.$$

Definiţie 3.4 *Un limbaj este ambigu dacă toate gramaticile care îl generează sunt ambigue. În caz contrar (adică dacă există o gramatică neambiguă care să îl genereze) limbajul este neambigu.*

Dacă G este ambiguă şi $p \in L(G)$ este un cuvânt care se poate obţine cu două derivări extrem stângi distincte, atunci există arborii $\mathcal{A}_{S,\,p}$ şi $\mathcal{A}'_{S,\,p}$, diferiţi, dar care au aceiaşi rădăcină şi frontieră.

Teorema 3.3 *Dacă L_1 şi L_2 sunt limbaje disjuncte neambigue, atunci $L_1 \cup L_2$ este neambigu.*

Demonstraţie. Fie $G_k = (V_{N_k}, V_{T_k}, S_k, P_k),\ k = 1, 2$ două gramatici de tipul 2 neambigue cu $L(G_k) = L_k$ şi fie $G = (V_{N_1} \cup V_{N_2} \cup \{S\}, V_{T_1} \cup V_{T_2}, S, P_1 \cup P_2 \cup \{S \to S_1|S_2\})$ gramatica ce generează limbajul reuniune $L(G_1) \cup L(G_2)$.

Să presupunem prin reducere la absurd că $L(G)$ este ambigu. Atunci există $p \in L(G)$ care se poate obţine cu două derivări extreme stângi diferite. Să presupunem că $p \in L(G_1),\ p \notin L(G_2)$. Atunci obţinem două derivări distincte în gramatica G (în care detalierem prima derivare directă)

$$(1)\ S \underset{G}{\Rightarrow} S_1 \overset{*}{\underset{G}{\Rightarrow}} p,\ deci\ S_1 \overset{*}{\underset{G}{\Rightarrow}} p;$$

$$(2)\ S \underset{G}{\Rightarrow} S_1 \overset{*}{\underset{G}{\Rightarrow}} p,\ deci\ S_1 \overset{*}{\underset{G}{\Rightarrow}} p,$$

deci şi două derivări extrem stângi distincte în gramatica G_1. Aceasta ar însemna că G_1 este ambiguă. Contradicţie cu ipoteza!□

Teorema 3.4 *Limbajele de tipul 3 sunt neambigue .*

Demonstraţie. Fie L un limbaj de tipul 3 şi G gramatica care îl generează; fie apoi AF automatul finit care recunoaşte limbajul L şi AFD automatul finit echivalent determinist. Construim gramatica G' astfel încât $L(G') = L(AFD)$. Reamintim că regulile lui G' se construiesc astfel $f(A, a) = B \Rightarrow A \to aB,\ \ f(A, a) \in \Sigma_f \Rightarrow A \to a$.

Să presupunem acum că L este ambigu; atunci orice gramatică care îl generează, inclusiv G', este ambiguă. Aceasta înseamnă că există un $p \in L(G')$ astfel încât

$$S \Rightarrow i_1A_1 \Rightarrow i_1i_2A_2 \Rightarrow \ldots \Rightarrow i_1 \ldots i_{n-1}A_{n-1} \left\{ \begin{array}{l} \Rightarrow i_1 \ldots i_nA'_n \Rightarrow \\ \Rightarrow i_1 \ldots i_nA''_n \Rightarrow \end{array} \right\} \stackrel{*}{\Rightarrow} p.$$

Deci există regulile $A_{n-1} \to i_nA'_n$ şi $A_{n-1} \to i_nA''_n$, adică în automatul AFD avem

$$f(A_{n-1}, i_n) = A'_n, \quad f(A_{n-1}, i_n) = A''_n,$$

ceea ce contrazice faptul că AFd este determinist.□

3.3 Forme normale pentru gramatici de tipul 2

Forma normală Chomsky.

Definiţie 3.5 *O gramatică în forma normală Chomsky este o gramatică cu reguli de forma*

$$A \to BC,$$
$$D \to i,$$

unde $A, B, C, D \in V_N$ şi $i \in V_T$. Se acceptă şi regula de completare $S \to \lambda$ cu condiţia ca S să nu apară în dreapta vreunei reguli.

Lema 3.1 *(lema substituţiei). Fie G o gramatică de tipul 2 şi $X \to uYv$ precum şi $Y \to p_1 \ldots p_n$ toate regulile din G care au Y în stânga . Atunci G este echivalentă cu o gramatică G' în care am făcut "substituţiile"; adică facem următoarea înlocuire*

$$X \to uYv \text{ se înlocuieşte cu } X \to up_1v| \ldots |up_nv$$

(Regulile $Y \to p_1| \ldots |p_n$ le vom păstra neschimbate).

Demonstraţie. Fie $p \in L(G)$ şi $S \stackrel{*}{\Rightarrow} p$. Punem în evidenţă doi paşi consecutivi oarecare:

$$G: \quad S \stackrel{*}{\Rightarrow} r \Rightarrow s \Rightarrow t \stackrel{*}{\Rightarrow} p.$$

Dacă în $r \Rightarrow s$ se utilizează regula $X \to uYv$ atunci în mod necesar în pasul următor se utilizează una din regulile $Y \to p_1| \ldots |p_n$, să presupunem $Y \to p_j$ (evident, este posibil ca această regulă să nu se aplice în pasul imediat următor, dar ea poate fi "adusă" în această poziţie). Prin urmare

$$(A) \quad G: \ r = r'Xr'' \Rightarrow r'uYvr'' \Rightarrow r'up_jvr'' = t.$$

Aceşti doi paşi se pot obţine şi în G' (într-un singur pas):

$$(B) \quad G': \ r = r'Xr'' \Rightarrow r'up_jr'' = t.$$

Deci $S \overset{*}{\underset{G'}{\Rightarrow}} p, p \in L(G')$ şi $L(G) \subseteq L(G')$.

Invers, dacă $p \in L(G')$ şi $S \overset{*}{\underset{G'}{\Rightarrow}} p$, atunci dacă la un pas se utilizează o regulă nou introdusă (pasul (B)), transformarea respectivă se poate obţine şi în G cu doi paşi (paşii (A)); deci $p \in L(G)$ şi $L(G') \subseteq L(G)$.□

Corolar 3.5 *Orice gramatică de tipul 2 este echivalentă cu o gramatică de acelaşi tip în care mulţimea de reguli nu conţine* redenumiri. *(O redenumire este o regulă de forma $A \to B,\ A, B \in V_N$).*

Intr-adevăr, dacă $A \to B \in P$ este o redenumire şi $B \to p_1|\ldots|p_n$ sunt toate regulile care au B în stânga, efectuăm substituţiile, deci înlocuim regula $A \to B$ cu $A \to p_1|\ldots|p_n$. În cazul în care printre acestea apare o nouă redenumire, repetăm procedeul.□

Exemplu. Gramatica G_E care generează expresii aritmetice $E \to E{+}T|T, T \to T * F|F, F \to (E)|i$ se poate pune sub următoarea formă (fără redenumiri) :

$$E \to E + T|T * F|(E)|i$$
$$T \to T * F|(E)|i$$
$$F \to (E)|i$$

Teorema 3.6 *(teorema lui Chomsky de existenţă a formei normale). Orice gramatică independentă de context este echivalentă cu o gramatică în forma normală Chomsky. ema2*

Demonstraţie. Putem porni de la o gramatică G care nu are redenumire şi ale cărei reguli cu terminale au forma $A \to i, A \in V_N, i \in V_T$. De asemenea presupunem că G nu are reguli de ştergere.

Rezultă că regulile lui G au una din formele:
(1) $A \to BC$,
(2) $D \to i$,
(3) $X \to X_1 \ldots X_n, n > 2$.

Construim o gramatică $G' = (V'_N, V_T, S, P')$ unde $V_N \subseteq V'_N$ şi P' conţine toate regulile din P de forma (1) şi (2). Fiecare regulă de forma (3) o înlocuim cu:

$$X \to X_1 Z_1,$$
$$Z_1 \to X_2 Z_2,$$
$$\ldots$$
$$Z_{n-2} \to X_{n-1} X_n$$

şi includem neterminalele $Z_1, \ldots, Z_{n-2}$ (altele pentru fiecare regulă de forma (3) în V'_N.

Se poate relativ uşor arăta că $L(G) = L(G')$. De exemplu, dacă $u \Rightarrow v$ (direct) în G şi de aplică o regulă de forma (1) sau (2), atunci evident derivarea respectivă se poate obţine şi în G'; în cazul în care se aplică o regulă de forma (3), avem

$$G: \quad u = u'Xu'' \Rightarrow u'X_1 \dots X_n u^{''} = v.$$

Această derivare se poate obţine şi în G' în mai mulţi paşi şi anume

$$G': \quad u = u'Xu'' \Rightarrow u'X_1Z_1u'' \Rightarrow u'X_1X_2Z_2u'' \Rightarrow \dots \Rightarrow u'X_1 \dots X_n u^{''} = v. \square$$

Observaţie. O gramatică ce are reguli de forma $A \to BC, A \to B, A \to a$ unde $A, B, C \in V_N$ şi $a \in V_T$ spunem că este în *forma 2–canonică.* Este evident că orice gramatică de tip 2 este echivalentă cu o gramatică în formă 2–canonică.

Gramatici recursive

Definiţie 3.6 *Un simbol neterminal X al unei gramatici de tipul 2 este recursiv dacă există o regulă de forma $X \to uXv,\ u, v \in V_G^*$.*

Dacă $u = \lambda$ ($v = \lambda$) simbolul X este stâng (drept) recursiv. O gramatică ce are cel puţin un simbol recursiv se numeşte recursivă. De exemplu, gramatica G_E care generează expresiile aritmetice are două simboluri stâng recursive, E şi T.

Existenţa simbolurilor stâng recursive poate provoca dificultăţi în aplicarea algoritmilor de analiză top-down. Într-adevăr, într-o astfel de gramatică, încercarea de a construi arborele de derivare corespunzător unui cuvânt p prin aplicarea întotdeauna a primei reguli pentru simbolul cel mai din stânga, poate să conducă la un ciclu infinit (de exemplu în G_E s-ar obţine $E \Rightarrow E + T \Rightarrow E + T + T \Rightarrow \dots$).

Teorema 3.7 *Orice limbaj de tipul 2 poate să fie generat de o gramatică fără recursie stângă.*

Demonstraţie. Fie $G = (V_N, V_T, S, P)$ o gramatică de tipul 2; presupunem că G are un singur simbol recursiv X şi fie

$$(A) \quad X \to u_1 | u_2 | \dots | u_n | Xv_1 | \dots | Xv_m$$

toate regulile care au X în stânga. Construim gramatica $G' = (V_N', V_T, S, P')$, unde $V_N \subset V_N',\ P \subset P'$ cu excepţia regulilor (A); acestea se înlocuiesc cu

$$\begin{aligned} &X \to u_1 | u_2 | \dots | u_n | u_1Y | u_2Y | \dots | u_nY, \\ &Y \to v_1 | \dots | v_m | v_1Y | \dots | v_mY \end{aligned}$$

G' este de tipul 2 şi nu are simboluri stâng recursive; se vede însă că Y este un simbol drept recursiv.

Fie $p \in L(G), S \overset{*}{\underset{G}{\Rightarrow}} p$. Dacă în această derivare nu intervine simbolul recursiv, atunci evident că $S \overset{*}{\underset{G'}{\Rightarrow}} p$. Să presupunem că X intervine la un anumit pas: $S \Rightarrow u \Rightarrow p$, unde $u = u'Xu^{''}$. Putem aplica, începând de la u spre dreapta, în primul rând regulile pentru X şi să urmărim numai subarborele respectiv, deci

$$G: \ X \underset{G}{\Rightarrow} Xv_{j_1} \underset{G}{\Rightarrow} Xv_{j_2}v_{j_1} \underset{G}{\Rightarrow} \ldots \underset{G}{\Rightarrow} Xv_{j_s}\ldots v_{j_1} \underset{G}{\Rightarrow} u_j v_{j_s}\ldots v_{j_1}.$$

Aceeaşi formă propoziţională o putem obţine şi în gramatica G' astfel

$$G': \ X \underset{G'}{\Rightarrow} u_j Y \underset{G'}{\Rightarrow} u_j v_{j_s} Y \underset{G'}{\Rightarrow} \ldots \underset{G'}{\Rightarrow} u_j v_{j_s}\ldots v_{j_1}.$$

Prin urmare avem $S \underset{G'}{\Rightarrow} u \underset{G'}{\Rightarrow} p$, adică $p \in L(G')$ şi $L(G) \subseteq L(G')$. Analog, $L(G') \subseteq L(G)$.□

Forma normală Greibach.

Definiţie 3.7 *O gramatică în forma normală Greibach este o gramatică cu reguli de forma*

$$A \to ip, \ \ unde \ \ A \in V_N, \ i \in V_T \ p \in V_N^*.$$

Se acceptă şi regula de completare $S \to \lambda$ cu condiţia ca S să nu apară în dreapta vreunei reguli.

Teorema 3.8 *(Teorema de existenţă a formei normale Greibach). Orice gramatică de tipul 2 este echivalentă cu o gramatică în forma normală Greibach.*

Demonstraţie. Fie G o gramatică de tipul 2 în forma normală Chomsky şi fie $V_N = \{S = X_1, X_2, \ldots, X_n\}$. Vom construi o gramatică echivalentă care să satisfacă cerinţele din forma normală Greibach în mai multe etape.
Etapa I. Vom modifica regulile de generare astfel încât toate regulile care nu sunt de forma $X \to i$ să satisfacă condiţia $X_j \to X_k p, \ j < k, \ p \in V_N^*$. Acest lucru îl facem cu un algoritm pe care îl prezentăm într-un limbaj nestandard de publicare (tip PASCAL):

$j := 1;$
e1: **begin**
Se elimină recursiile stângi; neterminalele
noi le notăm cu $Y_1, Y_2, \ldots$
end
if $j = n$ **then** STOP;
$j := j + 1;$
$l := 1;$
e2: **begin**
Fie $X_j \to X_l p, \ p \in V_N^*$ şi $X_l \to p_1 \ldots p_m$
toate regulile care au X_l în stânga; se efectuează toate substituţiile.
end
$l := l + 1;$
if $l < j - 1$ **then goto** e2
goto e1

Să observăm că pentru $j = 1$ şi după eliminarea recursiilor stângi condiţia cerută este evident îndeplinită; în plus, dacă au fost recursii, vom avea reguli cu partea stângă neterminale noi $Y_1, Y_2, \ldots$. Mai departe, luăm toate regulile care au în stânga X_2 ($j := j + 1 = 2$) şi efectuăm substituţiile; acestea se vor transforma în $X_2 \to X_k p$ cu $k \geq 2$ şi după o nouă eliminare a recursiilor stângi vom avea $k > 2$ plus reguli cu partea stângă neterminale noi. În felul acesta toate regulile care au în stânga X_1 şi X_2 satisfac condiţia cerută; în continuare $j := j + 1 = 3$, etc.

Etapa II. Avem acum trei categorii de reguli:

(1) $X_j \to i$;

(2) $X_j \to X_k p,\ j < k, p \in V_N^*$;

(3) $Y \to iq,\ q \in V_N^*,\ i = 1, \ldots, m$.

Aranjăm toate neterminalele într-un şir unic, la început $Y_1, \ldots, Y_m$ apoi $X_1, \ldots, X_n$ şi le redenumim, de exemplu cu $X_1, \ldots, X_{m+n}$:

$Y_1, \quad Y_2, \quad \ldots, \quad Y_m, \quad X_1, \quad X_2, \quad \ldots, \quad X_n$

$X_1, \quad X_2, \quad \ldots, \quad X_m, \quad X_{m+1}, \quad X_{m+2}, \quad \ldots, \quad X_{m+n}$

Vom nota $n + m = N$. În felul acesta regulile gramaticii vor avea numai formele (1) şi (2).

Etapa III. Toate regulile care au X_N în stânga vor avea forma (1). Fie $X_{n-1} \to X_N p_1 | \ldots | X_N p_n$ toate regulile care au X_{N-1} în stânga şi care nu sunt de forma (1). Efecuăm substituţiile lui X_N; în acest fel regulile care au X_N şi X_{N-1} în stânga satisfac cerinţele din forma normală Greibach. În continuare, considerăm toate regulile care au X_{N-2} în stânga şi efectuăm substituţiile, etc.□

Forma normală operator.

Forma normală operator

Una din formele importante pentru gramatici independente de context, utilizată în analiza sintactică prin metoda precedenţei, este forma operator a acestor gramatici.

Definiţie 3.8 *O gramatică independentă de context $G = (V_N, V_T, S, P)$ se spune că este în* forma normală operator *dacă oricare ar fi producţia $A \to \beta \in P$, în β nu apar două neterminale (*variabile*) consecutive, adică*

$$P \subseteq V_N \times [(V_N \cup V_T)^* \setminus (V_N \cup V_T)^* V^2 (V_N \cup V_T)^*].$$

Teorema 3.9 *Orice gramatică independentă de context este echivalentă cu o gramatică în forma normală operator.*

Demonstraţie. Fie $G = (V_N, V_T, S, P)$ o gramatică de tipul 2 şi $L(G)$ limbajul generat. Fără a restrânge generalitatea presupunem că $\lambda \notin L(G)$ şi G este în forma 2–canonică (regulile sunt de forma $A \to BC,\ A \to B,\ A \to a$ vezi teorema **??**). Definim o gramatică echivalentă $G' = (V'_N, V_T, S, P')$ astfel: $V'_N =$

$\{S\} \cup (V_N \times V_T)$, iar $P' = P_1 \cup P_2 \cup P_3 \cup P_4$ unde

$i)$ $P_1 = \{S \to (S,a)a |\ a \in V_T\}$;
$ii)$ $P_2 = \{(A,a) \to \lambda |\ A \in V_N, a \in V_T, A \to a \in P\}$;
$iii)$ $P_3 = \{(A,a) \to (B,a) |\ A,B \in V_N, a \in V_T, A \to B \in P\}$;
$iv)$ $P_4 = \{(A,a) \to (B,b)b(C,a) |\ A,B,C \in V_N, a,b \in V_T, A \to BC \in P\}$.

Să observăm că G' este în forma normală operator. Pentru a demonstra că $L(G) = L(G')$ vom defini mai întâi o funcţie $\phi : P_2 \cup P_3 \cup P_4 \longrightarrow P$ astfel:

$\phi((A,a) \to \lambda) = A \to a$ *pentru* $(A,a) \to \lambda \in P_2$;
$\phi((A,a) \to (B,a)) = A \to B$ *pentru* $(A,a) \to (B,a) \in P_3$;
$\phi((A,a) \to (B,b)b(C,a)) = A \to BC$ *pentru* $(A,a) \to (B,b)b(C,a) \in P_4$.

Funcţia ϕ se extinde în mod natural la $\phi' : (P_2 \cup P_3 \cup P_4)^* \longrightarrow P^*$. Vom arăta că în gramatica G, $\forall w \in V_T^*,\ \forall a \in V_T$ are loc derivarea extrem dreaptă $A \underset{G}{\overset{*}{\Rightarrow}} wa$ folosind producţiile $\pi_1, \pi_2, \ldots, \pi_n$ dacă şi numai dacă există în P' producţiile $\pi'_1, \pi'_2, \ldots, \pi'_n$ astfel ca $\phi(\pi'_i) = \pi_i,\ 1 \leq i \leq n$ şi în G' are loc derivarea extrem dreaptă $(A,a) \underset{G'}{\overset{*}{\Rightarrow}} w$ folosind producţiile $\pi'_1, \pi'_2, \ldots, \pi'_n$.

Să demonstrăm afirmaţia prin inducţie după n, lungimea derivării.

Dacă $n = 1$ atunci $w = \lambda, A \to a \in P$ şi în P' există producţia $(A,a) \to \lambda$, deci $(A,a) \Rightarrow \lambda$ şi $\phi((A,a) \to \lambda) = A \to a$.

Invers, dacă $(A,a) \Rightarrow w$ în G' atunci $w = \lambda$ (după forma producţiilor din G') şi are loc proprietatea enunţată.

Să presupunem afirmaţia adevărată pentru derivări de lungime cel mult $n-1$ şi să o demonstrăm pentru derivări de lungime $n > 1$. Fie aşadar $A \underset{G}{\overset{*}{\Rightarrow}} wa$ o derivare de lungime n în gramatica G şi punem în evidenţă prima derivare directă. Distingem două cazuri:

I. Prima producţie utilizată în derivare este $A \to B$. Atunci,

$$A \underset{G}{\Rightarrow} B \underset{G}{\overset{*}{\Rightarrow}} wa$$

şi conform ipotezei inductive avem în G' o derivare $(B,a) \underset{G'}{\overset{*}{\Rightarrow}} w$ (de lungine $n-1$) cu producţii satisfăcând condiţiile arătate. Dar cum $A \to B \in P$, în P' avem producţia $(A,a) \to (B,a)$ deci $(A,a) \underset{G'}{\overset{*}{\Rightarrow}} w$ în gramatica G'.

II. Prima producţie este de forma $A \to BC$. Atunci

$$A \underset{G}{\Rightarrow} BC \underset{G}{\overset{*}{\Rightarrow}} wa.$$

În acest caz $wa = ubva$ (conform lemei de localizare), astfel că $B \overset{*}{\underset{G}{\Rightarrow}} ub$ şi $C \overset{*}{\underset{G}{\Rightarrow}} va$. După ipoteza inductivă, vom avea în G' derivările:

$$(B,b) \overset{*}{\underset{G'}{\Rightarrow}} u, \quad (C,a) \overset{*}{\underset{G'}{\Rightarrow}} v.$$

Cum $A \to BC \in P$ vom avea în P' producţia $(A,a) \to (B,b)b(C,a)$ şi în G' putem scrie derivarea extrem dreaptă

$$(A,a) \Rightarrow (B,b)b(C,a) \overset{*}{\underset{G'}{\Rightarrow}} (B,b)bv \overset{*}{\underset{G'}{\Rightarrow}} ubv = w$$

şi producţiile care s-au aplicat îndeplinesc condiţiile din enunţ.

În mod analog se demonstrează reciproca.

Din această afirmaţie, luând în particular $A = S$, obţinem:

$$S \overset{*}{\underset{G}{\Rightarrow}} wa \quad \Leftrightarrow \quad (S,a) \overset{*}{\underset{G'}{\Rightarrow}} w, \ \ \forall w \in V_T^*, \forall a \in V_T.$$

Cum în G' există şi producţia $S \to (S,a)a$, am găsit: $wa \in L(G)$ dacă şi numai $wa \in L(G')$, deci cele două gramatici sunt echivalente.

Pentru a încheia demonstraţia trebuie să considerăm şi cazul $\lambda \in L(G)$. Aplicăm construcţia de mai sus unei gramatici ce generează $L(G) \setminus \{\lambda\}$ şi obţinem $G' = (V'_N, V_T, S, P')$ gramatica operator corespunzătoare. Considerăm acum gramatica $G_1 = (V_{N_1}, V_T, S_1, P_1)$ unde $V_1 = V' \cup \{S_1\}$, $P_1 = P' \cup \{S_1 \to \lambda, S_1 \to S\}$ care este în forma normală operator şi $L(G_1) = L(G)$.□

3.4 Automate push-down (APD)

Automatele push-down sunt mecanisme pentru recunoaşterea limbajelor independente de context.

Un APD se compune din (vezi figura 3.6):

1. O bandă de intrare care conţine simboluri ale unui *alfabet de intrare*; aceste simboluri constituie pe o bandă un anumit cuvânt peste alfabetul de intrare. Banda se mişcă numai spre stânga;

2. O *memorie push-down* (memorie inversă, stivă, pilă, etc) care conţine simboluri ale unui alfabet propriu, numit *alfabetul memoriei push-down.* Această memorie funcţionează ca o stivă - ultimul introdus, primul extras (Last In, First Out);

3. Un *dispozitiv de comandă* care se află permanent într-o anumită stare internă aparţinând unei mulţimi finite de stări. Dispozitivul de comandă posedă un *dispozitiv de citire* de pe banda de intrare şi un *dispozitiv de scriere-citire* în memoria push-down.

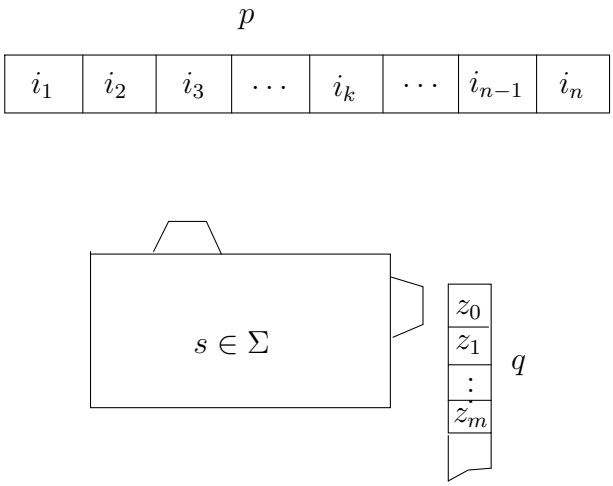

Figura 3.6: Reprezentare schematică a unui automat push-down

Ca şi un automat finit, un automat push-down funcţionează în paşi discreţi; un pas de funcţionare comportă:

1. Dispozitivul de comadă citeşte simbolul de pe banda de intrare din dreptul dispozitivului de citire şi mută banda cu o poziţie spre stânga.

2. În funcţie de starea internă, de simbolul citit şi de simbolul din vârful memoriei push-down dispozitivul de comandă efectuează operaţiile:
   (a) Trece într-o nouă stare;
   (b) Scrie în memoria push-down un anumit cuvânt peste alfabetul memoriei push-down; în particular, acesta poate să fie cuvântul vid, ceea ce are ca efect ştergerea simbolului din vârful memoriei push-down.

Funcţionarea unui APD se termină în general după ce s-a citit ultimul simbol al cuvântului scris pe banda de intrare dar este posibil ca el să efectueze un anumit număr de paşi, citind de fiecare dată de pe bandă cuvântul vid λ. De asemenea, este posibil ca în timpul funcţionării, deci înainte de a ajunge la ultimul simbol, automatul să se blocheze. De exemplu, automatul ajunge într-o configuraţie (stare, simbol pe bandă, simbol în vârful memoriei push-down) inadmisibilă sau se goleşte memoria push-down dar nu s-a epuizat cuvântul de pe bandă, etc.

Matematic, un APD se defineşte astfel:

Definiţie 3.9 *Un automat push-down este un sistem*
$APD = (\Sigma, I, Z, f, s_0, z_0)$
unde:

Σ *este mulţimea de stări (finită şi nevidă);*
I *este alfabetul de intare;*
Z *este alfabetul memoriei push-down;*

$f : \Sigma \times (I \cup \{\lambda\}) \times Z \longrightarrow \mathcal{P}(\Sigma \times Z^*)$ *este funcţia de evoluţie;*
$s_0 \in \Sigma$ *este starea iniţială;*
$z_0 \in Z$ *este simbolul iniţial al memoriei push-down.*

Un APD are în general o funcţioanre nedeterministă, $card\ f(s,i,z) \geq 1$; mulţimea automatelor push-down deterministe formează o clasă specială.

În termenii funcţiei de evoluţie, un pas de evoluţie comportă citirea simbolului i de pe bandă, citirea simbolului z din vârful memoriei push-down, apoi, în funcţie de starea internă s şi de aceste două simboluri, automatul trece într-o nouă stare s' şi scrie în memoria push-down un cuvânt $q \in Z^*$ astfel încât $(s', q) \in f(s, i, z)$. În cazul în care $f(s, i, z) = \emptyset$ evoluţia este oprită; este situaţia în care automatul se blochează.

O *stare* a automatului (sau configuraţie) este un sistem $\delta = (s, p, q)$ unde $s \in S$ este starea internă, $p \in I^*$ este subcuvântul de pe banda de intrare rămas de citit (inclusiv simbolul din dreptul dispozitivului de citire), iar $q \in Z^*$ este subcuvântul din memoria push-down.

Vom spune că un APD trece direct din starea $\delta_1 = (s_1, p_1, q_1)$ în starea $\delta_2 = (s_2, p_2, q_2)$ şi vom scrie $\delta_1 \longmapsto \delta_2$ dacă se execută un pas de evoluţie; dacă $p_1 = ip'_1, q_1 = zq'_1$ putem avea $(s_2, q) \in f(s_1, i, z)$ şi atunci $p_2 = p'_1, q_2 = qq'_1$ sau $(s_2, q) \in f(s_1, \lambda, z)$ şi atunci $p_2 = p_1, q_2 = qq'_1$.

Vom spune că automatul evoluează (fără specificaţia direct) din starea δ' în stare δ'' şi vom scrie $\delta' \overset{*}{\longmapsto} \delta''$ dacă:

(1) $\delta' = \delta''$;

(2) $\exists \delta_1, \ldots, \delta_n$ astfel încât $\delta' = \delta_1 \longmapsto \delta_2 \longmapsto \ldots \longmapsto \delta_{n-1} \longmapsto \delta_n = \delta''$.

Limbajul recunoscut de un APD se poate defini în două moduri:

(1) Limbajul recunoscut de un APD *cu golirea memoriei push-down*, este, prin definiţie

$$L(APD) = \{p | p \in I^*\ (s_0, p, z_0) \overset{*}{\longmapsto} (s, \lambda, \lambda)\}.$$

Aceata înseamnă că, pornind din starea internă s_0 şi având în vârful memoriei push-down simbolul z_0, cu ajutorul cuvântului p de pe banda de intrare, automatul poate să evolueze astfel încât să golească memoria push-down după citirea cuvântului. Menţionăm că golirea memoriei push-down nu trebuie neaparat să coincidă cu citirea ultimului simbol al lui p; este posibil ca automatul să mai efectueze câţiva paşi citind de pe bandă simbolul λ.

(2) Limbajul recunoscut de un APD *cu stări finale*; în definiţia automatului se adaugă o submulţime Σ_f a lui Σ numită mulţimea de stări finale. Prin definiţie, limbajul recunoscut de un APD cu stări finale este:

$$L(APD) = \{p | p \in I^*\ (s_0, p, z_0) \overset{*}{\longmapsto} (s, \lambda, q), s \in \Sigma_f, q \in Z^*\}.$$

Prin urmare, este necesar ca după citirea lui p, eventual după încă câţiva paşi, APD să ajungă într-o stare finală. Vom vedea că cele două definiţii sunt echivalente.

Limbaje recunoscute de automate push-down cu golirea memoriei. Vom arăta că familia limbajelor recunoscute de APD cu stări finale coincide cu familia limbajelor independente de context. În felul acesta, APD constituie mecanisme analitice de definire a limbajelor de tipul 2.

Teorema 3.10 *Un limbaj este independent de context dacă şi numai dacă este recunoscut de un automat push–down cu golirea memoriei push–down.*

Demonstraţie. Partea I $E \in \mathcal{L}_2 \Rightarrow E = L(APD)$.

Fie $G = (V_N, V_T, S, P)$ o gramatică de tipul 2 în forma normală Greibach care generează limbajul E. Construim un automat pushdown astfel:
$APD = (\{s\}, V_T, V_N, f, s, S)$, funcţia de evoluţie fiind definită de:

$$A \to ip \in P \Rightarrow (s, p) \in f(s, i, A),$$

altfel $\emptyset$.

Fie $p \in L(G),\ p = i_1 \dots i_n,\ S \overset{*}{\underset{G}{\Rightarrow}} p$. Această derivare trebuie să aibă forma (extrem stângă):

$$(A)\ \ S \Rightarrow i_1 X_1 u_1 \Rightarrow i_1 i_2 X_2 u_2 u_1 \Rightarrow i_1 i_2 i_3 X_3 u_3 u_2 u_1 \Rightarrow \dots \Rightarrow i_1 \dots i_n,$$

unde $u_1, u_2, u_3, \dots \in V_N^* = Z^*$.

Observaţie. Aparent, partea $u_s u_{s-1} \dots u_1$ se măreşte cu fiecare derivare directă. În realitate, unele din cuvintele u_j sunt vide, şi anume atunci când se aplică o regulă de forma $X \to i$; în particular, în ultimele derivări directe se aplică numai reguli de această formă.

Avem

$$\begin{aligned}
&S \to i_1 X_1 u_1 \Rightarrow (s, X_1 u_1) \in f(s, i_1, S),\\
&X_1 \to i_2 X_2 u_2 \Rightarrow (s, X_2 u_2) \in f(s, i_2, X_1),\\
&X_2 \to i_3 X_3 u_3 \Rightarrow (s, X_3 u_3) \in f(s, i_3, X_2),\\
&\dots .
\end{aligned}$$

Prin urmare automatul poate să aibă următoarea evoluţie:

$$(s, i_1 i_2 i_3 i_4 \dots i_n, S) \longmapsto (s, i_2 i_3 i_4 \dots i_n, X_1 u_1) \longmapsto$$

$$\longmapsto (s, i_3 i_4 \dots i_n, X_2 u_2 u_1) \longmapsto (s, i_4 \dots i_n, X_3 u_3 u_2 u_1) \longmapsto \dots .$$

Dacă comparăm această evoluţie cu derivarea (A) putem observa că pe banda de intrare avem la fiecare pas partea complementară a cuvântului (faţă de derivare) iar în memoria push-down se reproduce partea de neterminale din formele propoziţionale ale derivării. Cum în derivare se ajunge la $i_1 \dots i_n$, în evoluţie se va ajunge la (s, λ, λ).

Deci $p \in L(APD)$ şi $L(G) \subseteq L(APD)$.

Fie acum $p \in L(APD)$; va trebui să arătăm că $p \in L(G)$, deci că $S \underset{G}{\overset{*}{\Rightarrow}} p$. Vom arăta o implicaţie ceva mai generală, şi anume, pentru orice $u \in V_N^*$, avem

$$(s, p, u) \overset{*}{\longmapsto} (s, \lambda, \lambda) \Rightarrow u \underset{G}{\overset{*}{\Rightarrow}} p.$$

În particular, dacă $u = S$ obţinem implicaţia dorită.

Procedăm prin inducţie asupra lungimii lui p.

Dacă $|p| = 1$, atunci $p = i, u = X$ iar evoluţia va avea un singur pas $(s, i, X) \longmapsto (s, \lambda, \lambda)$, deci $(s, \lambda) \in f(s, i, X)$ şi $X \to i \in P$. Putem scrie $u = X \underset{G}{\Rightarrow} i = p$.

Presupunem că implicaţia este adevărată pentru un cuvânt $|p| = l$ şi considerăm un p astfel încât $|p| = l + 1$. Fie i şi X primele simboluri din p şi u, deci $p = ip'$ şi $u = Xu'$. În evoluţia $(s, p, u) \overset{*}{\longmapsto} (s, \lambda, \lambda)$ punem în evidenţă prima evoluţie directă

$$(s, p, u) = (s, ip', Xu') \longmapsto (s, p', vu') \overset{*}{\longmapsto} (s, \lambda, \lambda).$$

Din definiţia evoluţiei directe rezultă că $(s, v) \in f(s, i, X)$ deci $X \to iv \in P$. Pe de altă parte din ipoteza inductivă rezultă că $vu' \underset{G}{\overset{*}{\Rightarrow}} p'$. Avem

$$u = Xu' \underset{G}{\Rightarrow} ivu' \underset{G}{\overset{*}{\Rightarrow}} ip' = p,$$

ceea ce demonstrează implicaţia.

Prin urmare $p \in L(G)$ şi $L(APD) \subseteq L(G)$, de unde $L(G) = L(APD)$.□

Partea II. $E = L(APD)$ $E \in \mathcal{L}_2$

Fie $APD = (\Sigma, I, Z, f, s_0, z_0)$. Construim G de forma $G = (V_N, V_T, S, P)$ unde $V_N = \{s_0\} \cup \{(s, z, s') | s, s' \in \Sigma, z \in Z\}$, $V_T = I$, $S = s_0$, iar regulile de generare le definim astfel:

(1) $s_0 \to (s_0, z_0, s)$, $\forall s \in \Sigma$;

(2) Dacă $(s_1, z_1 \dots z_m) \in f(s, i, z)$ vom introduce în P reguli de forma

$$(s, z, s') \to i(s_1, z_1, s_2)(s_2, z_2, s_3) \dots (s_m, z_m, s'),$$

unde $s', s_2, \dots, s_m \in \Sigma$;

(3) Dacă $(s', \lambda) \in f(s, i, z)$ vom introduce în P reguli de forma

$$(s, z, s') \to i,$$

unde $s' \in \Sigma$.

Să observăm că gramatica astfel construită este independentă de context, şi anume în forma normală Greibach.

Fie $p \in L(APD)$, deci $(s_0, p, z_0) \overset{*}{\longmapsto} (s', \lambda, \lambda)$; trebuie să arătăm că $s_0 \overset{*}{\underset{G}{\Rightarrow}} p$.

Vom arăta implicaţia ceva mai generală

$$(s, p, z) \overset{*}{\longmapsto} (s', \lambda, \lambda) \quad \Rightarrow \quad (s, z, s') \overset{*}{\underset{G}{\Rightarrow}} p.$$

În particular pentru $s = s_0, z = z_0$ rezultă $(s_0, z_0, s') \overset{*}{\underset{G}{\Rightarrow}} p$ şi putem scrie $s_0 \Rightarrow (s_0, z_0, s') \overset{*}{\underset{G}{\Rightarrow}} p$, adică $p \in L(G)$.

Procedăm prin inducţie asupra lungimii evoluţiei l.

Dacă $l = 1$ atunci $(s, p, z) \longmapsto (s', \lambda, \lambda)$, deci $p = i$ şi $(s', \lambda) \in f(s, i, z)$ şi $(s, z, s') \to i$ este o regulă, adică putem scrie $(s, z, s') \Rightarrow i = p$.

Presupunem că implicaţia este adevărată pentru evoluţii de lungime oarecare l şi considerăm o evoluţie de lungime $l + 1$; punem în evidenţă prima evoluţie directă

$$(s, p, z) = (s, i_1 p', z) \longmapsto (s_1, p', z_1 \dots z_m) \overset{*}{\longmapsto} (s', \lambda, \lambda).$$

Descompunem cuvântul p' în forma $p' = p_1 \dots p_m$ astfel încât

$$\begin{array}{lcl} (s_1, p_1, z_1) & \overset{*}{\longmapsto} & (s_2, \lambda, \lambda), \\ (s_2, p_2, z_2) & \overset{*}{\longmapsto} & (s_3, \lambda, \lambda), \\ & \dots & \\ (s_m, p_m, z_m) & \overset{*}{\longmapsto} & (s', \lambda, \lambda). \end{array}$$

Observaţie. Putem pune în evidenţă felul în care se defineşte cuvântul p_1 urmărind evoluţia lui APD;

$$\begin{array}{ccccc} (s_1, i_1 i_2 \dots i_n, z_1 z_2 \dots z_m) & \longmapsto & (s_1', i_2 i_3 \dots i_n, q z_2 \dots z_m) & \longmapsto & \\ (a) & & (b) & & \\ \dots & \longmapsto & (s_2, i_{j_1} \dots i_n, z_2 \dots z_m) & & \\ & & (c) & & \end{array}$$

La primul pas (situaţia a) automatul este în starea s_1, pe bandă este i_1 iar în memoria push-down este z_1. După efectuarea unui pas, automatul trece în starea s_1', mută banda cu o poziţie spre stânga, extrage pe z_1 şi scrie în memoria push-down un cuvânt q (situaţia b). Se poate observa că z_2 a "coborât"; cum ştim că memoria push-down se goleşte ($p \in L(APD)$), trebuie ca la un moment dat z_2 să ajungă în vârful stivei (situaţia c). În acest moment partea din p citită va fi p_1 iar starea în care a ajuns automatul o notăm cu s_2. Este clar că dacă pe bandă am avea scris numai p_1 am avea evoluţia $(s_1, p_1, z_1) \longmapsto (s_2, \lambda, \lambda)$.

Analog $p_2, \dots, p_m$.

Din definiţia derivării directe $(s, i_1 p', z) \longmapsto (s_1, p', z_1 \dots z_m)$ avem $(s_1, z_1 \dots z_m) \in f(s, i_1, z)$ iar în P va exista regula

$$(s, z, s') \to i_1(s_1, z_1, s_2)(s_2, z_2, s_3) \dots (s_m, z_m, s')$$

unde luăm stările $s_2, \dots, s_m$ cele rezultate la descompunerea lui p'. Pe de altă parte, din ipoteza inductivă, avem

$$\begin{aligned}&(s_1, z_1, s_2) \stackrel{*}{\Rightarrow} p_1,\\ &(s_2, z_2, s_3) \stackrel{*}{\Rightarrow} p_2,\\ &\dots\\ &(s_m, z_m, s') \stackrel{*}{\Rightarrow} p_m.\end{aligned}$$

Putem scrie derivarea

$$(s, z, s') \Rightarrow i_1(s_1, z_1, s_2)(s_2, z_2, s_3) \dots (s_m, z_m, s') \stackrel{*}{\Rightarrow} i_1 p_1 \dots p_m = i_1 p' = p.$$

După cum am văzut, rezultă mai departe $p \in L(G)$ şi $L(APD) \subseteq L(G)$.

Pentru a demonstra incluziunea inversă, vom arăta mai întâi implicaţia $(s, z, s') \stackrel{*}{\Rightarrow} p(s_1, z_1 s_2) \dots (s_m, z_m, s')$ implică $(s, p, z) \stackrel{*}{\longmapsto} (s_1, \lambda, z_1 \dots z_m)$.

Procedăm prin inducţie asupra lungimii derivării l.

Dacă $l = 1$ atunci $p = i$ şi se aplică regula

$$(s, z, s') \to i(s_1, z_1, s_2) \dots (s_m, z_m, s')$$

deci $(s_1, z_1 \dots z_m) \in f(s, i, z)$ şi $(s, i, z) \longmapsto (s_1, \lambda, z_1 \dots z_m)$.

Presupunem că implicaţia este adevărată pentru l oarecare şi considerăm o derivare de lungime $l + 1$. Fie $p = p'i$ şi punem în evidenţă ultimul pas.

$(s, z, s') \stackrel{*}{\Rightarrow} p'(s'_{j-1}, z'_{j-1}, s'_j)(s_j, z_j, s_{j+1}) \dots (s_m, z_m, s')$

$\Rightarrow p'i(s_1, z_1, s_2) \dots (s_{j-1}, z_{j-1}, s_j)(s_j, z_j, s_{j+1}) \dots (s_m, z_m, s')$,

unde $s'_j = s_j$; la ultimul pas s-a aplicat regula

$$(s'_{j-1}, z'_{j-1}, s_j) \to i(s_1, z_1, s_2) \dots (s_{j-1}, z_{j-1}, s_j).$$

Rezultă $(s_1, z_1 \dots z_{j-1}) \in f(s'_{j-1}, i, z'_{j-1})$ şi putem scrie evoluţia $(s'_{j-1}, i, z'_{j-1}) \longmapsto (s_1, \lambda, z_1 \dots z_{j-1})$.

Pe de altă parte, conform ipotezei inductive, avem

$$(s, p', z) \stackrel{*}{\longmapsto} (s'_{j-1}, \lambda, z'_{j-1} z_j \dots z_m)$$

Prin urmare

$$(s, p, z) = (s, p'i, z) \stackrel{*}{\longmapsto} (s'_{j-1}, i, z'_{j-1} z_j \dots z_m) \longmapsto (s_1, \lambda, z_1 \dots z_m)$$

şi implicaţia este demonstrată.

Fie acum $p \in L(G)$, deci $s_0 \overset{*}{\underset{G}{\Rightarrow}} p$. Ţinând seama de forma regulilor din G, în această derivare se va aplica prima dată o regulă de forma (1), apoi regula de forma (2) iar la şfârşit reguli de forma (3). La aplicarea regulilor (2) putem rescrie la fiecare pas simbolul neterminal cel mai din stânga, deci să obţinem o derivare extrem stângă. Să observăm că în acest caz structura formelor propoziţionale intermediare este cea menţionată, $p(s_1, z_1, s_2)(s_2, z_2, s_3) \dots (s_m, z_m, s')$.

Prin urmare, derivarea va avea forma

$$s_0 \Rightarrow (s_0, z_0, s') \overset{*}{\Rightarrow} p(s_1, z_1, s_2) \dots (s_m, z_m, s') \overset{*}{\Rightarrow} p.$$

Trebuie să avem regulile $(s_j, z_j, s_{j+1}) \to \lambda, j = 1, \dots, m,\ s_{m+1} = s'$ şi putem scrie

$$(s_0, p, z_0) \overset{*}{\longmapsto} (s_1, \lambda, z_1 \dots z_m) \longmapsto (s_2, \lambda, z_2 \dots z_m) \longmapsto \dots \longmapsto (s', \lambda, \lambda)$$

adică $p \in L(APD)$ şi $L(G) \subseteq L(APD)$.□

Automate push–down cu stări finale. Vom nota un automat push-down cu stări finale cu APD_f.

Teorema 3.11 *Un limbaj este recunoscut de un automat push–down dacă şi numai dacă este recunoscut de un automat push–down cu stări finale.*

Demonstraţie. Partea I $E = l(APD) \Rightarrow E \in L(APD_f)$.

Dacă $APD = (\Sigma, I, Z, f, s_0, z_0)$ construim un automat push–down cu stări finale astfel

$$APD_f = (\Sigma \cup \{s_0', s_f\}, I, Z \cup \{z_0'\}, f', s_0', z_0')$$

unde mulţimea de stări finale este $\{s\}$ iar funcţia de evoluţie este definită de:

$f'(s, i, z) = f(s, i, z), s \in \Sigma, i \in I \cup \{\lambda\}, z \in Z$;

$f'(s_0', \lambda, z_0') = (s_0, z_0 z_0')$;

$f(s, \lambda, z_0') = (s_f, \lambda), s \in \Sigma$;

în rest $\emptyset$.

Fie $p \in L(APD)$; atunci $(s_0, p, z_0) \overset{*}{\longmapsto} (s, \lambda, \lambda)$. Evident că aceiaşi evoluţie o poate avea şi automatul push–down cu stări finale. Putem scrie în APD_f evoluţia

$$(APD_f): \quad (s_0', p, z_0') \overset{*}{\longmapsto} (s_0, \lambda, z_0') \longmapsto (s, \lambda, \lambda),$$

deci $p \in L(APD_f)$ şi $L(APD) \subseteq L(APD_f)$.

Invers, fie $p \in L(APD_f)$, atunci (în APD_f)

$$(s_0', p, z_0') \longmapsto (s, p, z_0 z_0') \overset{*}{\longmapsto} (s_f, \lambda, q).$$

Ultimul pas trebuie să fie de forma $(s, \lambda, z_0') \longmapsto (s_f, \lambda, \lambda)$ pentru că nu există altă valoare a lui f care să ne ducă într-o stare finală. Deci (în APD_f)

$$(s_0', p, z_0 z_0') \overset{*}{\longmapsto} (s, \lambda, z_0') \longmapsto (s_f, \lambda, \lambda)$$

şi putem scrie în APD evoluţia $(s_0, p, z_0) \vdash^* (s, \lambda, \lambda)$, adică $p \in L(APD)$ şi $L(APD_f) \subseteq L(APD)$.□

Partea II $E = L(APD_f \Rightarrow E \in L(APD)$.

Fie $APD_f = (\Sigma, I, Z, f, s_0, z_0, \Sigma_f)$ un automat push–down cu stări finale (mulţimea stărilor finale este Σ_f) şi construim un APD astfel

$$APD = (\Sigma \cup \{s_0', s'\}, I, Z \cup \{z_0'\}, f', s_0', z_0')$$

unde

$f'(s, i, z) = f(s, i, z), s \in \Sigma, i \in I \cup \{\lambda\}, z \in Z$;

$f(s_0', \lambda, z_0') = (s, z_0 z_0')$;

$f(s, \lambda, z) = (s', \lambda), s \in \Sigma_f \cup \{s'\}, z \in Z \cup \{z_0'\}$;

în rest $\emptyset$.

Fie $p \in L(APD_f)$, atunci $(s_0, p, z_0) \vdash^* (s, \lambda, q), s \in \Sigma_f$. Este evident că în APD avem evoluţia $(s_0, p, z_0) \vdash^* (s, \lambda, q)$. Putem scrie

$$APD: \quad (s_0', p, z_0') \longmapsto (s_0, p, z_0 z_0') \vdash^* (s, \lambda, q z_0') \vdash^* (s', \lambda, \lambda),$$

deci $p \in L(APD)$ şi $L(APD_f) \subseteq L(APD)$.

Invers, fie $p \in L(APD)$. Avem

$$APD: \quad (s_0', p, z_0') \vdash^* (s_0, p, z_0 z_0') \vdash^* (s, \lambda, \lambda).$$

Simbolul z_0' nu poate fi şters decât cu o regulă de forma $f(s, \lambda, z) = (s', \lambda)$, $s \in \Sigma_f \cup \{s'\}$, deci APD trebuie să ajungă într-o stare $s \in \Sigma_f$, apoi să rămână în s'.

$$APD: \quad (s_0, p, z_0 z_0') \vdash^* (s, \lambda, q z_0'), s \in \Sigma_f.$$

Putem scrie

$$APD_f: \quad (s_0, p, z_0) \vdash^* (s, \lambda, q), s \in \Sigma_f$$

şi deci $p \in L(APD_f)$, adică $L(APD) \subseteq L(APD_f)$.□

3.5 Automate push–down deterministe

Funcţionarea unui APD este în general nedeterministă, $card\ f(s, i, z) \geq 1$. Pentru ca un APD să aibă o funcţionare deterministă nu ete suficient să impunem condiţia $card\ f(s, i, z) = 1$, deoarece dacă pentru un anumit $s \in \Sigma$ şi $z \in Z$ avem $f(s, \lambda, z) \neq \emptyset$ şi $f(s, i, z) \neq \emptyset$, putem face un pas citind λ sau citind i.

Definiţie 3.10 *Un automat push–down este determinist dacă*

(1) card $f(s, i, z) \geq 1, s \in \Sigma, i \in I \cup \{\lambda\}, z \in Z$;

(2) dacă $f(s, \lambda, z) \neq \emptyset$, *atunci* $f(s, i, z) = \emptyset, \forall i \in I$.

Un limbaj recunoscut de un APD determinist îl vom numi limbaj independent de context (sau de tipul doi) determinist. Familia limbajelor independente de context deterministe este inclusă (strict) în familia limbajelor de tipul 2 (după cum vom vedea).

Un APD se poate bloca în următoarel două situaţii

1. Automatul ajunge în starea s, în vârful memoriei push–down se află simbolul z, pe banda de intrare urmează i şi $f(s,i,z) = f(s,\lambda,z) = \emptyset$;

2. Intră într-un ciclu infinit citind de pe bandă λ; de exemplu $f(s,\lambda,z) = (s,z)$ şi $f(s,i,z) = \emptyset$ pentru o anumită pereche (s,z).

Definiţie 3.11 *Un APD determinist este* neblocabil *dacă pentru orice cuvânt $p \in I^*$ există o evoluţie de forma $(s_0,p,z_0) \overset{*}{\longmapsto} (s,\lambda,q)$.*

Într-un APD determinist neblocabil orice cuvânt peste I poate fi citit. Evident, de aici nu rezultă că orice cuvânt este recunoscut de APD.

Lema 3.2 *Un APD determinist cu stări finale este echivalent cu un APD determinist cu stări finale neblocabil (relativ la prima situaţie de blocare).*

*Demonstraţie.*Fie $APD_f = (\Sigma, I, Z, f, s_0, z_0, \Sigma_f)$. Construim $APD'_f = (\Sigma \cup \{s'_0, s'\}, I, Z \cup \{z'_0\}, f', s'_0, z'_0, \Sigma_f)$ unde:
(1) $f'(s,i,z) = f(s,i,z)$ dacă $f(s,i,z) \neq \emptyset, s \in \Sigma, i \in I \cup \{\lambda\}, z \in Z$;
(2) $f'(s,i,z) = (s',z)$ dacă $f(s,i,z) = f(s,\lambda,z) = \emptyset, s \in \Sigma, i \in I, z \in Z$;
(3) $f'(s',i,z) = (s',z),\ i \in I, z \in Z$;
(4) $f'(s',\lambda,z'_0) = (s_0, z_0 z'_0)$.
Avem

$$p \in L(APD_f) \Leftrightarrow (s_0,p,z_0) \overset{APD_f}{\models} (s,\lambda,q), s \in \Sigma_f \Leftrightarrow$$

$$(s'_0,p,z'_0) \overset{APD'_f}{\models} (s_0,p,z_0z'_0) \overset{APD'_f}{\models} (s,\lambda,qz'_0), s \in \Sigma_f \Leftrightarrow p \in L(APD'_f).$$

Deci $L(APD_f) = L(APD'_f$.□

Observaţie. Într-o situaţie de blocare (s,ip,zq) şi $f(s,i,z) = f(s,\lambda,z) = \emptyset$ putem scrie

$$(s,ip,zq) \overset{APD'_f}{\models} (s',p,zq) \overset{APD'_f}{\models} \ldots \overset{APD'_f}{\models} (s',\lambda,zq).$$

Teorema 3.12 *Orice limbaj independent de context determinist este recunoscut de un APD determinist neblocabil.*

Demonstraţie. Fiind dat un APD determinist vom construi un APD determinist neblocabil echivalent. Conform lemei anterioare putem presupune că nu are loc prima situaţie de blocare.

Dacă are loc a doua situaţie de blocare, putem avea două cazuri:

1. conţinutul memoriei push–down se măreşte nelimitat;

2. lungimea cuvintelor scrise în memoria push–down nu depăşeşte un anumit număr.

Fie $card(\Sigma) = n$, $card(Z) = k$, $l = max\{|q|, q \in Z^* \mid (s', q) \in f(s, i, z)\}$.

Cazul 1. Există în total un număr nk de perechi de forma (s, z). Dacă în evoluţia lui APD s-ar succede numai configuraţii cu perechi de forma (s, z) distincte atunci lungimea cuvântului din memoria push–down ar creşte la maximum nkl simboluri. Prin urmare, dacă $(s', \lambda, \alpha') \overset{*}{\longmapsto} (s", \lambda, \alpha")$ şi $|\alpha"| - |\alpha'| > nkl$, atunci în această evoluţie trebuie să existe două configuraţii cu aceiaşi stare s şi cu acelaşi simbol z în vârful memoriei push–down. Deci

$$(s', \lambda, \alpha') \overset{*}{\longmapsto} (s, \lambda, zr) \overset{*}{\longmapsto} (s, \lambda, zqr) \overset{*}{\longmapsto} (s", \lambda, \alpha"),$$

de unde urmează că

$$(s', \lambda, \alpha') \overset{*}{\longmapsto} (s, \lambda, zr) \overset{*}{\longmapsto} (s, \lambda, q^m r), \forall m \in N.$$

Cazul II Lungimea cuvântului scris în memoria push–down nu depăşeşte nkl, căci dacă $|\alpha"| - |\alpha'| > nkl$, ar rezulta că în memoria push–down am avea $zq^m, \forall m \in N$ şi lungimea cuvântului nu ar fi finită.□

Teorema 3.13 *Dacă E este un limbaj independent de context determinist atunci limbajul complementar $I^* \setminus E$ este de asemenea independent de context determinist.*

Demonstraţie. Fie $APD_f = (\Sigma, I, Z, f, s_0, z_0, \Sigma_f)$ automatul push–down determinist care recunoaşte limbajul E. Construim un automat push–down determinist care va recunoaşte limbajul $I^* \setminus E$ în modul următor

$$APD'_f = (\Sigma \times \{\sigma_1, \sigma_2, \sigma_3\}, I, Z, f', s'_0, z_0, \Sigma'_f)$$

unde

$$s'_0 = \begin{cases} (s_0, \sigma_1) & pentru\ s_0 \in \Sigma_f, \\ (s_0, \sigma_2) & pentru\ s_0 \notin \Sigma_f, \end{cases}$$

mulţimea de stări finale este $\Sigma'_f = \{(s, \sigma_3) | s \in \Sigma\}$ iar funcţia de evoluţie este definită de

(1) dacă $f(s,i,z)=(s',q)$ atunci
$$f'((s,\sigma_1),i,z)=((s',k'),q),$$
$$f'((s,\sigma_2),i,z)=((s,\sigma_3),q),$$
$$f'((s,\sigma_3),i,z)=((s',k'),q);$$
(2) dacă $f(s,\lambda,z)=(s',q)$ atunci
$$f'((s,\sigma_1),\lambda,z)=((s',k'),q),$$
$$f'((s,\sigma_3),\lambda,z)=((s',k'),q);$$

unde $k'=1$ dacă $s'\in\Sigma_f$ şi $k'=2$ dacă $s'\notin\Sigma_f$.

Fie $p\in L(APD'_f)$, atunci $((s_0,\sigma_k),p,z_0)\overset{*}{\longmapsto}((s,\sigma_3),\lambda,q), k=1,2$. În mod necesar, ultima configuraţie trebuie să fie precedată de o configuraţie de forma $((s',\sigma_2),\lambda,q)$, deci

$$(A)\ ((s_0,\sigma_k),p,z_0)\overset{*}{\longmapsto}((s',\sigma_2),\lambda,q)\longmapsto((s,\sigma_3),\lambda,q)$$

de unde rezultă că $(s_0,p,z_0)\overset{*}{\longmapsto}(s',\lambda,q)$ şi $s'\notin\Sigma_f$. Deci $p\in I^*\setminus E$ şi $L(APD'_f)\subseteq I^*\setminus E$.

Invers, fie $p\in I^*\setminus E$, atunci în APD'_f avem evoluţia $(s_0,p,z_0)\overset{*}{\longmapsto}(s,\lambda,q), s\notin\Sigma_f$ şi putem scrie evoluţia (A). Prin urmare $p\in L(APD'_f)$, adică $I^*\setminus E\subseteq L(APD'_f)$.□

Consecinţă. Putem enunţa proprietatea de mai sus astfel: Familia limbajelor independente de context deterministe este închisă la complementariere. Cum familia limbajelor independente de context nu este închisă la complementariere şi cum ea coincide cu familia limbajelor recunoscute de automatele push–down nedeterministe putem mai departe obţine următorul rezultat:

APD nedeterministe nu sunt echivalente cu APD deterministe.

3.6 Lema Bar–Hillel

Ca şi în cazul limbajelor regulate, lema Bar–Hillel pune în evidenţă următoarea proprietate a unui limbaj independent de context: orice cuvânt al unui astfel de limbaj, suficient de lung, conţine un subcuvânt (nevid) pe care multiplicându–l de un număr arbitrar de ori, obţinem noi cuvinte care aparţin de asemenea limbajului. Mai poartă denumirea de "lema de pompare" sau "lema *uvwxy*".

Ne vom referi în cele ce urmează la gramatici de tipul 2 în formă normală Chomsky. Într-o gramatică de această formă arborii de derivare sunt <u>întotdeauna</u> arbori binari.

Lema 3.3 *Fie G o gramatică în forma normală Chomsky şi $X\overset{*}{\Rightarrow}p, p\in V_G^*$. Dacă cea mai lungă ramură din arborele $\mathcal{A}_{X,p}$ conţine m noduri, atunci $|p|\leq 2^{m-1}$.*

Demonstraţie. Procedăm prin inducţie asupra lui m.

Pentru $m=2$ arborele are două nivele şi în mod evident $|p|\leq 2=2^{m-1}$.

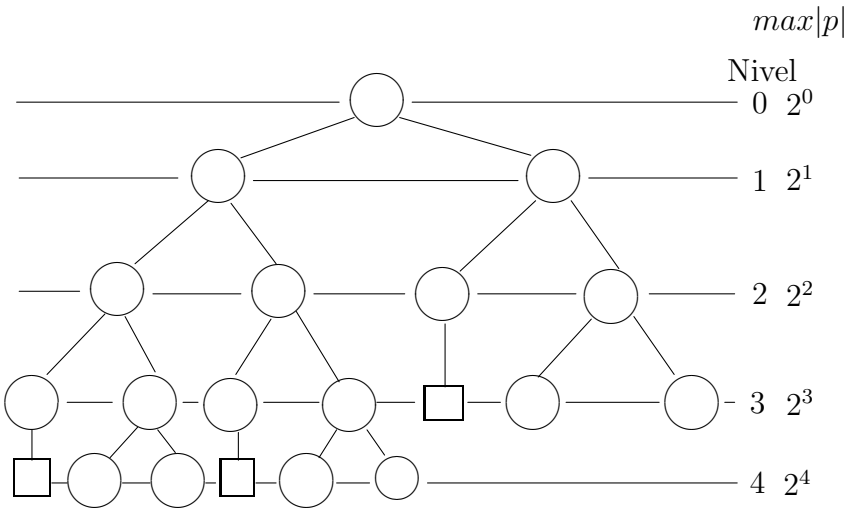

Figura 3.7: Exemplu de arbore

Presupunem că proprietatea este adevărată pentru un m oarecare şi considerăm un arbore care are pe cea mai lungă ramură $m+1$ noduri. În derivarea $X \stackrel{*}{\Rightarrow} p$ punem în evidenţă prima derivare directă

$$X \Rightarrow Y Z \stackrel{*}{\Rightarrow} p.$$

Conform lemei de localizare, $p = p_1 p_2$ şi $Y \stackrel{*}{\Rightarrow} p_1$, $Z \stackrel{*}{\Rightarrow} p_2$. Arborii de derivare $\mathcal{A}_{Y,p_1}$ şi $\mathcal{A}_{Z,p_2}$ conţin, fiecare pe cea mai lungă ramură cel mult m noduri; conform ipotezei de inducţie, avem $|p_1| \leq 2^{m-1}, |p_2| \leq 2^{m-1}$. Prin urmare

$$|p| = |p_1 p_2| = |p_1| + |p_2| \leq 2^{m-1} + 2^{m-1} = 2^m. \square$$

Exemplu. Considerăm un arbore de formă dată în figura 3.7. Cea mai lungă ramură are $m = 5$ noduri (nivelul ultim plus 1). Se poate uşor vedea că pe ultimul nivel putem avea cel mult 2^{m-1} noduri (în acest caz toate nodurile sunt neterminale).

Observaţie. Dacă $X \stackrel{*}{\Rightarrow} p$ şi $|p| > 2^{m-1}$ atunci există în arborele $\mathcal{A}_{X,p}$ cel puţin o ramură cu $m+1$ noduri.

Lema 3.4 *Fie G o gramatică de tipul 2 şi fie $X \Rightarrow X_1 \ldots X_m \stackrel{*}{\Rightarrow} p$. Atunci, conform lemei de localizare, $p = p_1 \ldots p_m$ şi $X_j \stackrel{*}{\Rightarrow} p_j, j = 1, \ldots, m$. Fie $\mathcal{A}_{Y,q} \subseteq \mathcal{A}_{X,p}$. atunci q este subcuvânt într–unul din cuvintele p_j.*

Demonstraţie. Neterminalul Y aparţine unuia din subarborii $\mathcal{A}_{X_j,p_j}$, fie acesta $\mathcal{A}_{X_k,p_k}$; atunci $\mathcal{A}_{Y,q} \subseteq \mathcal{A}_{X_k,p_k}$ şi $q \in Sub(p_k)$. Imaginea grafică este dată de figura 3.8.$\square$

Lema 3.5 *(Lema Bar–Hillel) Fie E un limbaj independent de context. Atunci există un număr natural k astfel încât, dacă $p \in E$ şi $|p| > k$ atunci p se poate descompune în forma $p = uvwxy$ cu următoarele proprietăţi:*

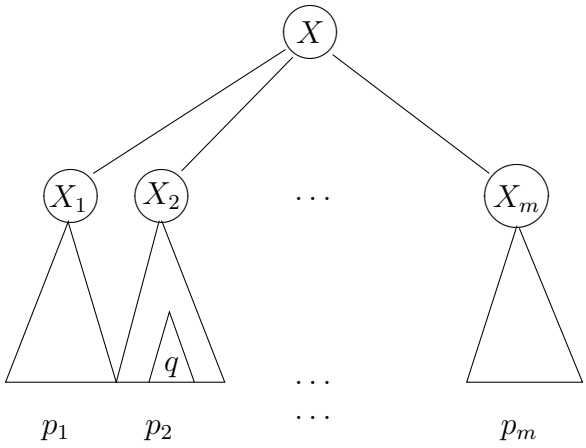

Figura 3.8: Reprezentarea grafică a incluziunii $q \in Sub(p_k)$.

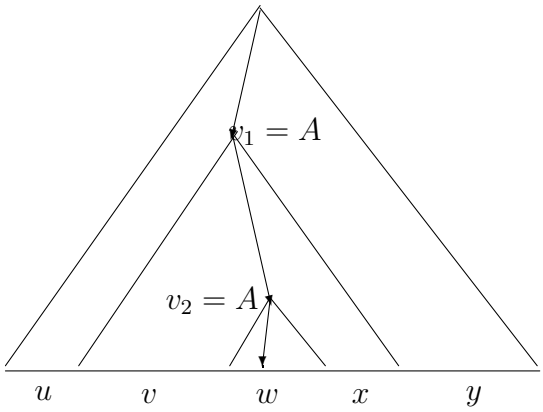

Figura 3.9: Descompunerea cuvantului p.

1. $vx \neq \lambda$;

2. $|vwx| \leq k$;

3. $uv^jwx^jy \in E, \forall j \in N$,

Demonstraţie. Fie $n = card(V_N)$. Luăm $k = 2^n$. Cum $|p| > k = 2^n$, conform observaţiei de la prima lemă, există în arborele $\mathcal{A}_{S,p}$ cel puţin o ramură cu $n+2$ noduri; pe această ramură ultimul nod este terminal, deci există $n+1$ noduri neterminale, din care minim două sunt etichetate cu acelaşi simbol. Parcurgem ramura de lungime maximă începând de la frunza şi alegem primul simbol neterminal A care se repetă pe aceasta ramură. Fie v_1 şi v_2 nodurile cu eticheta A găsite prin această procedură. Descompunem cuvântul p ca în figura 3.9, $p = uvwxy$.

(1) Considerăm subarborele $\mathcal{A}_{A,vwx}$ căruia îi corespunde derivarea $A\stackrel{*}{\Rightarrow}vwx$. Punem în evidenţă primul pas

$$A\Rightarrow BC\stackrel{*}{\Rightarrow}vwx,$$

şi vwx se descompune în $vwx = p_Bp_C, B\stackrel{*}{\Rightarrow}p_B, C\stackrel{*}{\Rightarrow}p_C$ şi $p_B, p_C \neq \lambda$. Cum $\mathcal{A}_{A,w} \subseteq \mathcal{A}_{A,vwx}$, rezultă că w este subcuvânt în p_B sau în p_C. Să presupunem că $w \in Sub(p_C)$; atunci $p_B \in Sub(v)$ şi cum $p_B \neq \lambda$ rezultă $v \neq \lambda$.

(2)Din modul de alegere al nodurilor $v_1 = A$ şi $v_2 = A$ rezultă că pe ramura punctată începând de la v_1 în jos până la frontieră sunt cel mult $n + 2$ noduri, ultimul fiind terminalul. Deoarece arborele $\mathcal{A}_{A,vwx}$ contine doar terminale pe frontieră, iar acestea se obţin numai prin aplicarea de reguli de forma $D \to i$, numarul frunzelor este egal cu numarul neterminalelor ce pot fi obtinute intr-un arbore cu $n + 1$ noduri cea mai lunga ramura, deci conform lemei precedente, $|vwx| \leq 2^n$.

(3) putem scrie derivările $A\stackrel{*}{\Rightarrow}w, A\stackrel{*}{\Rightarrow}vAx$. Deci

$$S\stackrel{*}{\Rightarrow}uAy\stackrel{*}{\Rightarrow}uvAxy\stackrel{*}{\Rightarrow}uv^2Ax^2y\stackrel{*}{\Rightarrow}\ldots\stackrel{*}{\Rightarrow}uv^jwx^jy.\square$$

Problema închiderii familiei $\mathcal{L}_2$ la intersecţie

Teorema 3.14 *Familia $\mathcal{L}_2$ nu este închisă la intersecţie.*

Demonstraţie. Considerăm limbajul $L_3 = \{a^nb^nc^n | n \geq 1\}$. Să rătăm că $L_3 \notin \mathcal{L}_2$. Într–adevăr, să presupunem că $L_3 \in \mathcal{L}_2$ şi fie k constanta din lema Bar–Hillel. Luăm $n > k/3$ şi fie $p = a^nb^nc^n$; avem $|p| > k$, deci conform acestei leme p se descompune în forma $p = uvwxy$ cu $vx \neq \lambda$ şi $uv^jwx^jy \in L_3, j \in N$.

Vom analiza posibilităţile de constituire a subcuvintelor v şi x. Să presupunem că în v (sau x) intră două din simbolurile a, b, c; de exemplu $v = aabbb$. atunci considerăm cuvântul $p_2 = uv^2wx^2y = uaabbaabbwx^2y$ care nu are structura cuvintelor lui L_3 ("a" nu poate să urmeze după "b") dar conform lemei Bar-Hillel aparţine lui L_3. Deci, în v (sau x) nu pot intra două (sau trei) din simbolurile a, b, c. Să presupunem că intră un singur simbol; de exemplu $v \in \{a\}^*$ şi $x \in \{b\}^*$. Atunci multiplicând în p subcuvintele v şi x, puterile simbolurilor "a" şi "b" se măresc iar "c" rămâne pe loc, contrazicându–se din nou structura cuvintelor din L_3. În concluzie L_3 nu este independent de context.

Fie acum limbajele

$$L_3' = \{a^mb^nc^n | m, n \geq 1\}$$
$$L_3^{"} = \{a^nb^nc^m | m, n \geq 1\}.$$

Se poate vedea uşor că aceste limbaje sunt de tipul 2; gramaticile care le generează sunt $S \to aS|aA, A \to bAc|bc$ şi respectiv $S \to Sc|Ac, A \to aAb|ab$.

Avem $L_3' \cap L_3^{"} = L_3$, deci intersecţia a două limbaje de tipul 2 este un limbaj care nu aparţine lui $\mathcal{L}_2$.$\square$

Corolar 3.15 *Familia $\mathcal{L}_2$ nu este închisă la complementariere.*

Într–adevăr, să presupunem că $\mathcal{L}_2$ este închisă la complementariere şi fie $E_1, E_2 \in \mathcal{L}_2$. Cum familia $\mathcal{L}_2$ este închisă la reuniune, ar rezulta $C(E_1) \cup C(E_2) \in \mathcal{L}_2$. Dar (de Morgan) $C(E_1) \cup C(E_2) = C(E_1 \cap C(E_2) \ \in \mathcal{L}_2$.

Complementul limbajului $C(E_1 \cap E_2)$ este $E_1 \cap E_2$ şi conform presupunerii ar trebui ca $E_1 \cap E_2 \in \mathcal{L}_2$, oricare ar fi E_1, E_2, ceea ce nu este adevărat.□

Generalizări ale lemei Bar–Hillel Lema lui Bar–Hillel reprezintă o condiţie necesară ca un limbaj să fie independent de context. Cu ajutorul ei se poate demonstra relativ uşor că anumite limbaje nu sunt independente de context. Ideea este aceea că prin multiplicarea subcuvintelor v şi x, de un număr suficient de ori, se contrazice structura limbajului.

Totuşi, nu orice limbaj care nu este de tipul 2 poate fi respins de lema Bar–Hillel. De exemplu, această lemă nu poate respinge limbajul

$$L = \{a^n b^{2^m} | n, m \geq 1\} \cup \{b\}^*,$$

care nu este de tipul 2 (se demonstrează pe altă cale). Într–adevăr, pentru orice $p = a^n b^{2^m}$ luăm

$$u = \lambda, v = a, w = \lambda, x = \lambda, y = a^{n-1} b^{2^m}.$$

Putem itera subcuvintele v şi x fără să contrazicem structura cuvintelor limbajului.

O generalizare a lemei Bar–Hillel este

Lema 3.6 *(Lema lui Ogden) pentru orice limbaj independent de context L există o constantă k astfel încât orice $p \in L$ pentru care cel puţin k simboluri sunt "marcate", se poate descompune în forma $p = uvwxy$ astfel încât*

1. *sau u, v, w sau w, x, y conţin fiecare câte un simbol marcat;*

2. *vwx conţine cel mult k simboluri marcate;*

3. *$uv^j wx^j y \in L, \forall j \in N$*

Cu ajutorul acestei leme se poate arăta că limbajul de mai sus nu este de tipul 2, considerând marcate simbolurile b.

3.7 Închiderea familiei $\mathcal{L}_2$ la substituţii

Definiţie 3.12 *Vom spune că o familie de limbaje $\mathcal{L}$ este închisă la substituţii dacă oricare ar fi $L \in \mathcal{L}$ şi oricare ar fi substituţia $s : V^* \longrightarrow \mathcal{P}(U^*)$ astfel încât $s(i) \in \mathcal{L}, \forall i \in V$, avem $s(L) \in \mathcal{L}$.*

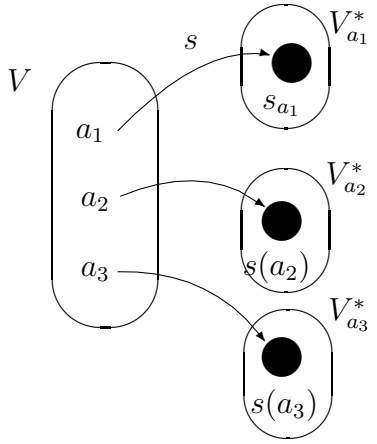

Figura 3.10: Reprezentare grafică a substituţiei

Se cunosc o serie de proprietăţi de închidere a familiilor de limbaje din clasificarea Chomsky faţă de substituţii particulare; o parte din acestea sunt prezentate în tabelul următor.

	$\mathcal{L}_0$	$\mathcal{L}_1$	$\mathcal{L}_2$	$\mathcal{L}_3$
Substituţie	DA	NU	DA	DA
Substituţie λ–liberă	DA	DA	DA	DA
Homomorfisme	DA	NU	DA	DA
Homomorfisme λ–libere	DA	DA	DA	DA

În cele ce urmează vom considera noţiunea de substituţie într–o accepţiune ceva mai generală (faţă de cea din Capitolul I)

Definiţie 3.13 *Fie V un alfabet; pentru orice $a \in V$ considerăm un alfabet V_a şi notăm $V_g = \bigcup_{a\in V} V_a$. Fie $s : V \longrightarrow \mathcal{P}(V_g^*)$ o aplicaţie cu proprietatea $s(a) \in V_a^*, a \in V$. Prelungirea canonică a lui s la V^* o numim substituţie.*

Observaţie. Prelungirea canonică se defineşte recursiv astfel:

$$s(\lambda) = \{\lambda\}, \;\; s(a_1 \dots a_n) = s(a_1) \dots s(a_n).$$

O reprezentare grafică este prezentată în figura 3.10. Observăm că
$V_g^* \neq \bigcup_{a\in V} V_a^*$ şi $V_a^* \subseteq V_g^*$.

Definiţia substituţiei se poate da şi direct:

Definiţie 3.14 *Vom spune că $s : V^* \longrightarrow \mathcal{P}(V_g^*)$ este o substituţie dacă*

1. $s(\lambda) = \{\lambda\}$;
2. $s(a) \subseteq V_a^*, \forall a \in V$;
3. $s(a_1 \dots a_n) = s(a_1) \dots s(a_n),\ a_1, \dots, a_n \in V$.

Definiţie 3.15 *Imaginea unui limbaj $L \in V^*$ prin substituţia s este*

$$s(L) = \bigcup_{a \in L} s(a).$$

Definiţie 3.16 *Vom spune că o familie de limbaje $\mathcal{L}$ este închisă la substituţii dacă oricare ar fi $L \in \mathcal{L}$ şi oricare ar fi substituţia $s : V^* \longrightarrow \mathcal{P}(V_g^*$ astfel încât $s(a) \in \mathcal{L}, \forall a \in V$, avem $s(L) \in \mathcal{L}$.*

Observaţie. În cazul particular în care $V_a = U, \forall a \in V$, avem $V_g = U$ şi condiţia $s(a) \subseteq V_a^*$ este în mod evident satisfăcută.

Teorema 3.16 *Familia limbajelor independente de context este închisă la substituţii.*

Demonstraţie. Fie $L \in \mathcal{L}_2$ şi $s : V^* \longrightarrow \mathcal{P}(V_g^*)$ o substituţie astfel încât $s(a) \in \mathcal{L}_2, \forall a \in V$.

Fie

$$G = (V_N, V, S, P) \ \ cu \ \ L(G) = L,$$
$$G_a = (V_N^a, V_a, S_a, P_a) \ \ cu \ \ L(G_a) = s(a),$$

G şi G_a fiind gramatici de tipul 2. Vom presupune că aceste gramatici nu conţin reguli de stergere (cu excepţia regulilor de completare) şi că alfabetele neterminale sunt disjuncte.

Definim $\mu : V_N \cup V \cup \{\lambda\} \longrightarrow V_N \cup \{S_a | a \in V\} \cup \{\lambda\}$ prin

$$\mu(\lambda) = \lambda,$$
$$\mu(X) = X, X \in V_N,$$
$$\mu(a) = S_a, a \in V$$

şi prelungim canonic aplicaţia la $(V_N \cup V)^*$.

Considerăm acum gramatica $G' = (V_N', V_T', S, P')$ unde

$$V_N' = V_N \cup (\bigcup_{a \in V} V_N^a),$$
$$V_T' = \bigcup_{a \in V} V_a,$$
$$P' = (\bigcup_{a \in V} P_a) \cup \{X \to \mu(\alpha) | X \to \alpha \in P\}.$$

Observaţie. P' conţine toate regulile din toate gramaticile G_a nemodificate; regulile din G le modifică, şi anume terminalele din părţile drepte le înlocuim cu simbolurile de start din gramaticile G_a.

Să mai observăm că G' este de tipul 2.

Fie $u \in s(L)$. Deoarece $s(L) = \bigcup_{v \in L} s(v)$ rezultă că există $v \in L$ astfel încât $u \in s(v)$.

Dacă $v = \lambda$ înseamnă că $s \to \lambda \in P$ şi $S \to \mu(\lambda) = \lambda \in P'$. Ţinând cont că $s(\lambda) = \{\lambda\}$ rezultă $u = \lambda$ şi $\lambda \in L(G')$.

Să presupunem că $v \neq \lambda$ şi fie $v = v_1 \dots v_n$; avem $s(v) = s(v_1) \dots s(v_n)$. Prin urmare $u = u_1 \dots u_n$ şi $u_1 \in s(v_1), \dots, u_n \in s(v_n)$.

Avem $S \overset{*}{\underset{G}{\Rightarrow}} v$ şi $S_{a_i} \overset{*}{\Rightarrow} u_i$ în $G_{a_i}, i = 1, \ldots, n$.

Atunci

$$G' : \quad S \Rightarrow \mu(v_1) \Rightarrow \mu(v_2) \Rightarrow \ldots \Rightarrow \mu(v_n) = \mu(v).$$

Cum v conţine numai terminale, rezultă că $\mu(v) = S_{a_1} \ldots S_{a_n}$. Deci

$$G' : \quad S \overset{*}{\Rightarrow} X_{a_1} \ldots X_{a_n} \overset{*}{\Rightarrow} u_1 \ldots u_n = u,$$

şi $u \in L(G')$, adică $s(L) \subseteq L(G')$.

Invers, fie $u \in L(G')$. deoarece $(\cup V_N^a) \cap V_N = \emptyset$, derivările din G' care utilizează reguli din P_a nu introduc simboluri din V_N. În derivarea $S \overset{*}{\underset{G'}{\Rightarrow}} u$ efectuăm mai întâi derivările directe care utilizează reguli de forma $X \to \mu(\alpha)$. Vom avea

$$G' : \quad S \overset{*}{\Rightarrow} S_{a_1} \ldots S_{a_n} \overset{*}{\Rightarrow} u.$$

Cuvântul u se descompune în $u = v_1 \ldots v_n$ şi $S_{a_j} \overset{*}{\underset{G'}{\Rightarrow}} v_j$. Deoarece alfabetele gramaticilor G_a sunt disjuncte, putem scrie $S_{a_j} \overset{*}{\Rightarrow} v_j$ (în gramatica G_{a_j}) şi deci $v_j \in L(G_{a_j}) = s(a_j)$. Aşadar

$$u = v_1 \ldots v_n \in s(a_1 \ldots s(a_n) = s(a_1 \ldots a_n) \subseteq s(L).$$

Rezultă $L(G') \subseteq s(L)$ şi în final $s(L) = L(G')$.□

Proprietatea de închidere a familiei $\mathcal{L}_2$ la substituţii se poate utiliza pentru a reobţine proprietăţi cunoscute. De exemplu:

Corolar 3.17 *Familia $\mathcal{L}_2$ este închisă la operaţiile regulate (reuniune, produs, închidere Kleene).*

Demonstraţie. Limbajele $\{a, b\}$, $\{ab\}$, $\{a\}^*$ sunt regulate, deci şi independente de context.

Definim o substituţie s astfel: $V = \{a, b\}, V_a, V_b$ oarecare şi fie $L_1 \subseteq V_a^*, L_2 \subseteq V_b^*$ două limbaje independente de context. Considerăm $s : \{a, b\} \longrightarrow \mathcal{P}(\ (V_a \cup V_b)^*\ )$ definită de

$$s(a) = L_1, s(b) = L_2$$

Conform teoremei anterioare rezultă că limbajele $s(\{a, b\}), s(\{ab\}), s(\{a\}^*)$ sunt independente de context; dar

$s(\{a, b\})L_1 \cup L_2, s(\{ab\}) = L_1 L_2, s(\{a\}^*) = L_1^*$.

De exemplu

$$s(\{a, b\}) = s(a) \cup s(b) = L_1 \cup L_2, \ etc.$$

3.8 Caracterizarea limbajelor independente de context

Limbajul lui Dyck Cuvintele limbajului Dyck, numit şi limbaj parametric, constituie şiruri de paranteze corect scrise într-o expresie aritmetică. De exemplu, dacă în expresia

$$\{(a+b)*[(c+d)-(e+f)]\}$$

se elimină toţi operanzii se obţine şirul de paranteze $\{()[()()]\}$, care după cum se poate observa sunt corect scrise. Definiţia generală a unui şir de paranteze corect scrise nu este foarte simplă; de exemplu, nu este suficient să spunem că fiecărei paranteze deschise trebuie să–i corespundă o paranteză închisă.

Şirurile de paranteze corect scrise se pot defini elegant cu ajutorul gramaticilor de tipul 2. Fie $G_n = (\{X\}, V_T, X, P)$ o gramatică de tipul 2, unde

$$V_T = \{i_1, \ldots, i_n, i'_1, \ldots, i'_n\},$$
$$P = \{X \to Xi_kXi'_kX,\ k = 1, \ldots, n\ X \to \lambda\}.$$

Prin definiţie, *limbajul lui Dyck de ordinul n* este $D_n = L(G_n)$.

Exemplu. Presupunem $n = 3$; putem scrie derivarea
$X \Rightarrow Xi_3Xi'_3X \Rightarrow Xi_3Xi_1Xi'_1Xi'_3X \Rightarrow Xi_3Xi_1Xi'_1Xi_2Xi'_2Xi'_3X$
$Xi_3Xi_1Xi'_1Xi_2Xi_1Xi'_1Xi'_2Xi'_3X \Rightarrow Xi_3Xi_1Xi'_1Xi_2Xi_1Xi_1Xi'_1Xi'_1Xi'_2Xi'_3X.$

În final, aplicând regula de ştergere, obţinem

$$X \overset{*}{\Rightarrow} i_3i_1i'_1i_2i_1i_1i'_1i'_1i'_2i'_3.$$

Dacă asociem simbolurilor i_k, i'_k perechi de paranteze, de exemplu: $i_1, i'_1 \leftrightarrow (,)$; $i_2, i'_2 \leftrightarrow [,]$; $i_3, i'_3 \leftrightarrow \{,\}$ obţinem $\{()[()()]\}$.

Proprietăţi ale limbajului lui Dyck

1. $i_kD_ni'_k \subseteq D_n,\ k = 1, \ldots, n$;
2. $D_nD_n \subseteq D_n$;
3. $D_n \subseteq (\bigcup_{k=1}^{n} i_kD_ni'_kD_n) \cup \{\lambda\}$;
4. dacă $p, pq \in D_n$ atunci $q \in D_n$.

Demonstraţie. (1) Un cuvânt din $i_kD_ni'_k$ trebuie să aibă forma $i_kpi'_k, p \in D_n$ (deci $X \overset{*}{\Rightarrow} p$). Putem scrie

$$X \Rightarrow Xi_kXi'_kX \overset{*}{\Rightarrow} i_kXi'_k \overset{*}{\Rightarrow} i_kpi'_k \in D_n.\square$$

(2) Fie $p \in D_nD_n$, deci $p = p_1p_2, p_1 \in D_n, p_2 \in D_n$ sau $X \overset{*}{\Rightarrow} p_1, X \overset{*}{\Rightarrow} p_2$; fie $p = i_1 \ldots i_n$. Derivarea $X \overset{*}{\Rightarrow} p_1$ o putem detalia lăsând la sfârşit ştergerile:

$$X \overset{*}{\Rightarrow} Xi_1X_2X \ldots Xi_nX \overset{*}{\Rightarrow} i_1 \ldots i_n.$$

Putem scrie

$$X\overset{*}{\Rightarrow}Xi_1X_2X\dots Xi_nX\overset{*}{\Rightarrow}i_1\dots i_nX=p_1X\overset{*}{\Rightarrow}p_1p_2=p\in D_n.\square$$

(3) fie $p\in D_n$; trebuie să arătăm că $p\in(\bigcup_{k=1}^{n}i_kD_ni'_kD_n)\cup\{\lambda\}$. Procedăm prin inducţie asupra lungimii lui p. Dacă $|p|=0$, atunci $p=\lambda$ şi apartenenţa este evidentă.

Observaţie. Putem lua $|p|=2$, deci $p=i_ki'_k$; avem

$$i_kD_ni'_kD_n=i_k\{\lambda,\dots\}i'_k\{\lambda,\dots\}=\{i_ki'_k,\dots\}$$

şi deci $i_ki'_k\in i_kD_ni'_kD_n$.

Presupunem acum că proprietatea este adevărată pentru $|p|$ oarecare şi considerăm cazul $|p|+1$. punem în evidenţă prima derivare directă $X\Rightarrow Xi_kXi'_kX\overset{*}{\Rightarrow}p$. Conform lemei de localizare, $p=p_1i_kp_2i'_kp_3$ şi $X\overset{*}{\Rightarrow}p_j$, $j=1,2,3$, adică $p_j\in D_n$. Dacă $p_1=\lambda$ proprietatea este demonstrată. Presupunem că $p_1\neq\lambda$. Cum $p_1\in D_n$, avem $p_1=i_jp'_1i'_jp''_1, p'_1, p''_1\in D_n$, în conformitate cu ipoteza inductivă. Prin urmare

$$p=i_jp'_1i'_j\underbrace{p''_1\underbrace{i_kp_2i'_k}_{\in D_n}p_3}_{=q\in D_n}=i_jp'_1i'_jq.\square$$

(4) Fie $p,pq\in D_n$. Procedăm prin inducţie asupra lui $|pq|$.

Dacă $|pq|=0$, rezultă $q=\lambda\in D_n$.

Presupunem că proprietatea este adevărată pentru $|pq|$ oarecare şi considerăm cazul $|pq|+1$.

Din $p,q\in D_n$ rezultă

$$pq=i_ksi'_kt,\ s,t\in D_n.$$

Avem două cazuri

$$I.\ pq=\underbrace{i_ks_1s_2\dots s_\alpha i'_k\overbrace{t_1t_2}^{t'}}_{p}\underbrace{\dots t_p}_{q}$$

Deci

$$\underbrace{p}_{\in D_n}=\underbrace{i_ksi'_k}_{\in D_n}\Rightarrow(ip.ind.)\ t'\in D_n,$$

$$\underbrace{t}_{\in D_n}=\underbrace{t'q}_{\in D_n}\Rightarrow(ip.ind.)\ q\in D_n.$$

$$II.\ pq=\underbrace{i_ks_1s_2}_{p}\underbrace{\dots s_\alpha}_{s'}\underbrace{i'_kt_1t_2\dots t_p}_{q}.$$

Avem

$$i_ksi'_k=ps'i'_k=i_kp'i'_kp''s'i'_k, p', p''\in D_n,$$

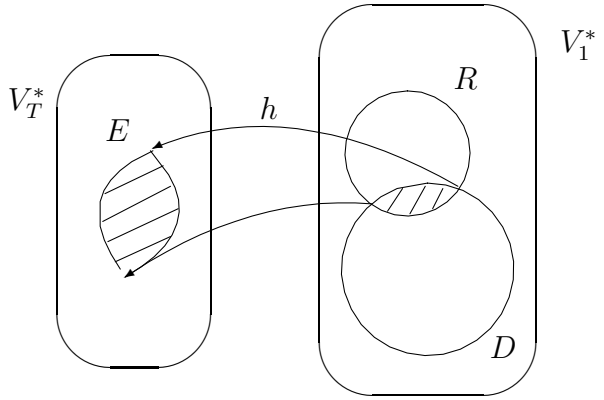

Figura 3.11: Imagine grafică a teoremei de caracterizare

$$\underbrace{s}_{\in D_n} = \underbrace{p'}_{\in D_n}\ i'_k p'' s' \Rightarrow (ip.ind.)\ i'_k p'' s' \in D_n$$

ceea ce nu este posibil.□

Teorema de caracterizare

Teorema 3.18 *Un limbaj este independent de context dacă şi numai dacă este imaginea printr-un homomorfism a intersecţiei dintre un limbaj regulat şi un limbaj Dyck.*

Demonstraţie. Vom arăta numai implicaţia directă, adică dacă E este un limbaj independent de context atunci există un limbaj regulat R, un limbaj Dyck şi un homomorfism h astfel încât $E = h(R \cap D)$.

Imaginea geometrică este prezentată în figura 3.11. Fie $G = (V_N, V_T, S, P)$ în forma normală Chomsky astfel încât $E = L(G)$. Presupunem că
$V_T = \{i_1, \ldots i_m\}$.

(a) Construcţia lui R.

Luăm gramatica $G' = (V_N, V_1, S, P')$ unde
$V_1 = \{i_1, i_2, i_m, i_{m+1}, \ldots, i_{m+n}, i'_1, \ldots, i'_{m+n}\}$ iar P' se construieşte astfel:

$$X \to i_k \in P \ \Rightarrow \ \begin{cases} X \to i'_k i_k \in P', \\ X \to i_k i'_k i_{n+l} Z \in P', \end{cases} \ l = \overline{1,n},\ Z \in V_N;$$
$$X \to YZ \in P \ \ atunci \ \ X \to i_{m+l} Y \in P', l = \overline{1,n}.$$

De fiecare dată alegem un i_{m+l} diferit (de aici rezultă n).

(b) Construcţia lui D.

Luăm $D = D_{m+n}$ cu alfabetul terminal V_1 şi cu regulile cunoscute.

(c) Definiţia lui h.

Luăm $h : V_1 \longrightarrow V_T \cup \{\lambda\}$ astfel:

$$h(i_k) = \begin{cases} i_k, k \geq m \\ \lambda, k > m, \end{cases} \quad h(i'_k) = \lambda,\ k = 1, \ldots, m+n.$$

Prelungim h în mod canonic la V_1^*. Se poate arăta uşor că h este un homomorfism.

Să arătăm că $E \subseteq h(R \cap D)$.

Fie $p \in E = L(G)$; va trebui să arătăm că există $q \in R,\ D$ astfel încât $p \in h(q)$. Vom arăta implicaţia ceva mai generală:

Dacă $X \underset{G}{\overset{*}{\Rightarrow}} p$ atunci $\exists q, X \underset{G'}{\overset{*}{\Rightarrow}} q, q \in D, p = h(q)$.

Procedăm prin inducţie asupra lungimii l a derivării.

Dacă $l = 1$, atunci $X \underset{G}{\Rightarrow} p = i_k, X \to i_k i'_k \in P'$. Luăm $q = i_k i'_k$ şi avem în G' derivarea $X \Rightarrow i_k i'_k$, iar $h(q) = h(i_k)h(i'_k) = i_k = p$.

Presupunem că implicaţia este adevărată pentru derivări de lungime oarecare l şi considerăm cazul $l + 1$; punem în evidenţă prima derivare directă

$$X \underset{G}{\Rightarrow} YZ \underset{G}{\overset{*}{\Rightarrow}} p.$$

Conform lemei de localizare, $p = p_1 p_2, Y \underset{G}{\overset{*}{\Rightarrow}} p_1, Z \underset{G}{\overset{*}{\Rightarrow}} p_2$. Ţinând cont de ipoteza inductivă avem în gramatica G'

$$G' : \begin{array}{l} \exists q_1,\ Y \overset{*}{\Rightarrow} q_1,\ q_1 \in D,\ h(q_1) = p_1; \\ \exists q_2,\ Z \overset{*}{\Rightarrow} q_2,\ q_2 \in D,\ h(q_2) = p_2. \end{array}$$

Ne fixăm atenţia asupra derivării $Y \underset{G'}{\overset{*}{\Rightarrow}} q_1$; fiind o derivare într-o gramatică de tipul 3, în toate derivările directe se aplică reguli de categoria I ($A \to pB$) cu excepţia ultimei unde se aplică o regulă de categoria II ($c \to q$). În cazul nostru, această ultimă regulă trebuie să fie de forma $X \to i_k i'_k$. Odată cu această regulă avem şi $X \to i_k i'_k k i'_{m+l} Z$; aplicând–o pe aceasta, avem

$$Y \underset{G'}{\overset{*}{\Rightarrow}} q_1 i'_{m+l} Z \underset{G'}{\overset{*}{\Rightarrow}} q_1 i'_{m+l} q_2.$$

Pe de altă parte, deoarec $X \to YZ \in P$ avem $X \to i_{m+l} Y \in P'$, aşa încât

$$X \underset{G'}{\overset{*}{\Rightarrow}} i_{m+l} Y \underset{G'}{\overset{*}{\Rightarrow}} i_{m+l} q_1 i'_{m+l} q_2 \overset{def}{=} q.$$

Avem $X \underset{G'}{\overset{*}{\Rightarrow}} q, q \in D$ şi $h(q) = h(q_1)h(q_2) = p_1 p_2 = p$.

Să arătăm acum că $h(R \cap D) \subseteq E$.

Fie $q \in R, D$ şi $p \in h(q)$, atunci $p \in L(G)$.

Mai general

$$X \underset{G'}{\overset{*}{\Rightarrow}} q, q \in D \ \ atunci \ \ X \underset{G}{\overset{*}{\Rightarrow}} h(q).$$

Procedăm prin inducţie asupra lungimii derivării l.

Dacă $l = 1$ atunci $X \Rightarrow q = i_k i'_k;\ q \in D_n, h(q) = i_k$. S–a aplicat regula $X \to i_k i'_k \in P'$; rezultă $X \to i_k \in P$ deci $X \underset{G}{\Rightarrow} i_k = h(q)$.

Presupunem că implicaţia este adevărată pentru un l oarecare şi considerăm cazul $l+1$. Punem în evidenţă prima derivare directă (evident, aceasta nu poate să fie de forma $X \to i_k i'_k$). Distingem două cazuri

I. Se aplică o regulă de forma $X \to i_k i'_k i'_{m+l} Z$. Avem

$$X \Rightarrow i_k i'_k i'_{m+l} Z \overset{*}{\underset{G'}{\Rightarrow}} q,$$

$$\underbrace{q}_{\in D} = \underbrace{i_k i'_k}_{\in D} i'_{m+l} q_1 \Rightarrow i'_k i'_{m+l} q_1 \in D,\ \textit{nu este posibil}.$$

II. Se aplică o regulă de forma $X \to i_{m+l} Y$. Avem

$$G' :\ X \Rightarrow i_{m+l} Y \overset{*}{\Rightarrow} q,\ q = i_{m+l} q' i'_{m+l} q^{"}, q', q^{"} \in D.$$

Detaliat, derivarea de mai sus trebuie să aibă forma

$$G' :\ X \Rightarrow i_{m+l} Y \overset{*}{\Rightarrow} i_{m+l} q' Z \overset{*}{\Rightarrow} i_{m+l} q' i'_{m+l} q^{"}.$$

De aici $Y \overset{*}{\underset{G'}{\Rightarrow}} q', Z \overset{*}{\underset{G'}{\Rightarrow}} q^{"}$ şi conform ipotezei inductive avem $Y \overset{*}{\underset{G}{\Rightarrow}} h(q'), Z \overset{*}{\underset{G}{\Rightarrow}} h(q^{"})$. Putem scrie

$$G :\ X \Rightarrow YZ \overset{*}{\Rightarrow} h(q')h(q^{"}) = h(i_{m+l})h(q')h(i_{m+l})h(q^{"}) = h(i_{m+l} q' i'_{m+l} q^{"}) = h(q). \square$$

1. Să se arate că $L(G) = \{(ab)^n a | n \geq 0\}$ unde G are producţiile $S \to SbS,\ S \to a$. Să se construiască o gramatică echivalentă cu G care să fie neambiguă.

2. Eliminaţi redenumirile din gramatica G_E ce generează expresiile aritmetice simple.

3. Aduceţi la forma normală Chomsky gramaticile ce au regulile:

   (a) $S \to TbT, T \to TaT | ca$;

   (b) $S \to aAC, A \to aB | bAB, B \to b, C \to c$;

   (c) $S \to A.A, A \to 0A | 1A | \ldots 9A | \lambda$.

4. Găsiţi forma normală Greibach pentru gramaticile:

   (a) $A_1 \to A_2 A_3,\ A_2 \to A_1 A_2 | 1,\ A_3 \to A_1 A_3 | 0$.

   (b) G_E care generează expresiile aritmetice simple.

5. Construiţi un automat push–down pentru recunoaşterea limbajului:

(a) $L = \{w | w \in \{a, b\}^*$, numărul literelor a în w este egal cu numărul literelor b în w $\}$;

(b) $L = \{w | w \in \{a, b\}^*, w = \tilde{w}\}$;

(c) $L = \{w | w \in \{(,)\}^*, w$ este un cuvânt în care fiecare paranteză deschisă are o pereche, paranteză închisă $\}$.

6. Folosind lema de pompare să se arate că următoarele limbaje nu sunt independente de context:

(a) $L = \{a^i b^j c^k | i < j < k\}$;

(b) $L = \{a^i b^j | j = i^2\}$;

(c) $L = \{a^n b^n c^n | n \geq 1\}$;

(d) $L = \{a^i | i$ prim $\}$;

(e) $L = \{a^i b^i c^j | j \geq i\}$;

Capitolul 4

Limbaje dependente de context

4.1 Gramatici monotone

Reamintim că o gramatică este monotonă dacă orice regulă $u \to v$ satisface condiţia $|u| \leq |v|$. Este permisă şi λ–regula, $S \to \lambda$ (care evident nu satisface condiţia de monotonie) cu condiţia ca S să nu apară în dreapta vreunei reguli.

Lema 4.1 *Fie $G_1 = (V_N, V_T, S, P \cup \{u \to v\})$ unde $u \to v$ este o regulă care satisface condiţia de monotonie iar P sunt reguli de tipul 1. Atunci G_1 este echivalentă cu o gramatică dependentă de context.*

Demonstraţie. Presupunem că P satisface condiţia din lema 1.2, deci $u = X_1 \dots X_m$ şi $v = Y_1 \dots Y_n$ cu $X_i, Y_j \in V_N$ şi $m \leq n$. Putem presupune că $m \geq 2$. Fie $Z_1, \dots, Z_m$ neterminale noi şi considerăm regulile R:

$$
\begin{aligned}
&X_1X_2 \dots X_m \to Z_1X_2 \dots X_m,\\
&Z_1X_2 \dots X_m \to Z_1Z_2 \dots X_m,\\
&\dots\\
&Z_1Z_2 \dots Z_{m-1}X_m \to Z_1Z_2 \dots Z_mY_{m+1} \dots Y_n,\\
&Z_1Z_2 \dots Z_mY_{m+1} \dots Y_n \to Y_1Z_2 \dots Z_mY_{m+1} \dots Y_n,\\
&\dots\\
&Y_1Y_2 \dots Y_{m-1}Z_mY_{m+1} \dots Y_n \to Y_1Y_2 \dots Y_n.
\end{aligned}
$$

Luăm $G_1' = (V_N \cup \{Z_1, \dots, Z_m\}, V_T, S, P \cup R)$. Evident G_1' este de tipul 1. Să arătăm că $L(G_1') = L(G)$.

Fie $p \in L(G)$. Dacă în derivarea $S \overset{*}{\underset{G_1}{\Rightarrow}} p$ nu se utilizează regula $u \to v$, este clar că putem scrie în G_1' derivarea $S \overset{*}{\Rightarrow} p$ şi $p \in L(G_1')$. Să presupunem că la un anumit pas se utilizează regula $u \to v$; detaliem

$$(G_1): \quad S \overset{*}{\Rightarrow} \alpha = \alpha' u \alpha'' \Rightarrow \alpha' v \alpha'' = \beta \overset{*}{\Rightarrow} p.$$

Pasul $\alpha \Rightarrow \beta$ se poate obţine şi în gramatica G_1' utilizând regulile R:

$$(G_1'): \quad \alpha = \alpha' u \alpha'' = \alpha' X_1 \dots X_m \alpha'' \Rightarrow \dots \Rightarrow \alpha' v \alpha'' = \beta$$

Rezultă că $S \overset{*}{\Rightarrow} p$ şi deci $p \in L(G_1')$, adică $L(G_1) \subseteq L(G_1')$.

Invers, fie $p \in L(G_1')$. Dacă în derivarea $S \overset{*}{\Rightarrow} p$ s–au folosit numai reguli din P atunci evident $S \underset{G_1}{\overset{*}{\Rightarrow}} p$. Să presupunem că la un anumit pas s–a utilizat pentru prima dată o regulă din R; aceasta nu poate să fie decât prima deoarece până în acest moment nu pot apărea simboluri Z. În continuare trebuie aplicate succesiv toate regulile din R, altfel nu putem elimina neterminalele Z. Paşii respectivi (de la α la β) vor avea forma (G_1') de unde rezultă forma (G_1). Prin urmare, $p \in L(G_1)$, mai departe $L(G_1') \subseteq L(G_1)$ şi $L(G_1') = L(G_1)$. □

Să observăm că orice gramatică dependentă de context este în mod evident monotonă. Vom demonstra în cele ce urmează afirmaţia reciprocă.

Teorema 4.1 *Orice gramatică monotonă este echivalentă cu o gramatică dependentă de context.*

Demonstraţie. Orice gramatică monotonă se poate pune sub forma $G_n = (V_N, V_T, S, P \cup \{u_i \to v_i, i = 1, \dots, n\})$ unde regulile $u_i \to v_i$ satisfac condiţia de monotonie iar P sunt reguli de tipul 1 (P conţine cel puţin o regulă, anume cea care are S în stânga). Este clar că G_n este o generalizare a lui G_1; procedând ca în lemă, putem construi o gramatică dependentă de context G_n' echivalentă cu G_n.□

Corolar 4.2 *Familia limbajelor generate de gramatici monotone coincide cu familia $\mathcal{L}_2$.*

Gramatici liniar mărginite.

Definiţie 4.1 *O gramatică monotonă este de ordinul n dacă orice regulă $u \to v$ satisface condiţia $|v| \leq n$.*

Lema 4.2 *O gramatică monotonă de ordinul $n \geq 3$ este echivalentă cu o gramatică monotonă de ordinul $n - 1$.*

Demonstraţie. Fie $G = (V_N, V_T, S, P)$ o gramatică monotonă de ordinul $n \geq 3$. Putem presupune din nou că G are forma din lema 1.2. Construim $G' = (V_N', V_T', S, P')$ cu $V_N \subset V_N'$ şi în plus

1. Dacă $u \to v \in P, |u|, |v| \leq 2$ atunci $u \to v \in P'$;

2. Fie $u \to v \in P, |v| > 2$; atunci $v = Y_1 Y_2 Y_3 v'$.

(a) dacă $u = X_1$ introducem în P' regulile

$$X_1 \to Z_1 Z_2,$$
$$Z_1 \to Y_1,$$
$$Z_2 \to Y_2 Y_3 v'.$$

(b) dacă $u = X_1 X_2 u'$ introducem în P' regulile

$$X_1 X_2 \to Z_1 Z_2,$$
$$Z_1 \to Y_1,$$
$$Z_2 u' \to Y_2 Y_3 v'.$$

În ambele cazuri $Z_1, Z_2 \in V'_N$.

Este clar că G' este monotonă, are ordinul $n-1$ şi $L(G') = L(G)$.□

Observaţie. Procedând în mod analog putem reduce ordinul unei gramatici monotone până la valoarea 2. Prin urmare, orice gramatică monotonă este echivalentă cu o gramatică monotonă de ordinul 2.

Forma normală Kuroda.

Definiţie 4.2 *O gramatică se numeşte liniar mărginită (sau în forma normală Kuroda) dacă are reguli de generare de forma*

1. $S \to SA$,
2. $B \to C|i$,
3. $DE \to FH$,

unde, ca de obicei, S este simbolul de start, $A, B, C, D, E, F, H \in V_N$ şi $i \in V_T$ cu condiţia $C, F, H \neq S$.

Teorema 4.3 *Orice gramatică monotonă este echivalentă cu o gramatică liniar mărginită.*

Demonstraţie. Fie G o gramatică monotonă pe care o presupunem de ordinul 2. Prin urmare regulile din G au formele:

1. $B \to C|i$,
2. $DE \to FH$,
3. $X \to YZ$.

Regulile (1) şi (2) satisfac condiţiile de la gramaticile liniar mărginite, iar cele de forma (3) presupunem că nu satisfac aceste condiţii, deci că $X \neq S$ sau $Y \neq S$. În general, putem avea mai multe astfel de reguli; pentru simplificare presupunem că există două reguli de forma (3):

$$(3'); \quad X_1 \to Y_1 Z_1, \quad X_2 \to Y_2 Z_2.$$

Construim o gramatică liniar mărginită $G' = (V_N \cup \{S', \bar{X}_1, \bar{X}_2\}, V_T, S', P')$, unde P' conţine toate regulile care convin (de formele (1) şi (2)) precum şi

$$\begin{array}{l} S' \to S, \\ S' \to S'\bar{X}_1 | S'\bar{X}_2, \\ X_1\bar{X}_1 \to Y_1 Z_1, X_2\bar{X}_2 \to Y_2 Z_2. \end{array}$$

De asemenea vom include în P' următoarel reguli de comutare (relativ la simbolurile nou introduse $\bar{X}_1, \bar{X}_2$):

$$\begin{array}{l} A\bar{X}_1 \to \bar{X}_1 A, \\ A\bar{X}_2 \to \bar{X}_2 A, \end{array} , \forall A \in V_N.$$

Se vede că G' este o gramatică liniar mărginită. Să arătăm că $L(G) = L(G')$.

Fie $p \in L(G)$, deci $S \underset{G}{\overset{*}{\Rightarrow}} p$. Presupunem că în această derivare există o singură secvenţă $u \overset{*}{\Rightarrow} v$ în care se aplică reguli de forma (3′), adică

$$G: \quad S \overset{*}{\Rightarrow} u \overset{*}{\Rightarrow} v \overset{*}{\Rightarrow} p.$$

Presupunem că secvenţa $u \overset{*}{\Rightarrow} v$ are forma

$$u = u_1 X_1 u_2 X_2 u_3 X_2 u_4 \overset{*}{\Rightarrow} u_1 Y_1 Z_1 u_2 Y_2 Z_2 u_2 Y_2 Z_3 u_4 = v,$$

unde $u_1, u_2, u_3, u_4 \in V_N^*$. În G' putem scrie

$$\begin{array}{l} S' \Rightarrow S'\bar{X}_2 \Rightarrow S'\bar{X}_2\bar{X}_2 \Rightarrow S'\bar{X}_1\bar{X}_2\bar{X}_2 \Rightarrow S\bar{X}_1\bar{X}_2\bar{X}_2 \\ \overset{*}{\Rightarrow} u\bar{X}_1\bar{X}_2\bar{X}_2 = u_1 X_1 u_2 X_2 u_3 X_2 u_4 \bar{X}_1\bar{X}_2\bar{X}_2 \overset{*}{\Rightarrow} u_1 X_1 \bar{X}_1 u_2 X_2 \bar{X}_2 u_3 X_2 \bar{X}_2 u_4 \\ \overset{*}{\Rightarrow} u_1 Y_1 Z_1 u_2 Y_2 Z_2 u_3 Y_2 Z_2 u_4 = v \overset{*}{\Rightarrow} p \end{array}$$

Prin urmare, $S' \underset{G'}{\overset{*}{\Rightarrow}} p$ şi $p \in L(G')$.

Invers, fie $p \in L(G')$, deci $S' \underset{G'}{\overset{*}{\Rightarrow}} p$. Dacă în derivarea $S' \underset{G'}{\overset{*}{\Rightarrow}} p$ apar simbolurile $\bar{X}_1, \bar{X}_2$ ele nu pot apărea dacât în urma aplicării regulilor $S' \to S'\bar{X}_1$ şi $S' \to S'\bar{X}_2$, deci la începutul derivării, apoi singura posibilitate de continuare este $S' \to S$; de asemenea, aceste simboluri nu pot fi eliminate decât cu regulile $X_1\bar{X}_1 \to Y_1 Z_1$ şi $X_2\bar{X}_2 \to Y_2 Z_2$, etc. Astfel derivarea noastră trebuie să aibă forma descrisă mai sus, de unde rezultă $S \underset{G}{\overset{*}{\Rightarrow}} p, p \in L(G)$.□

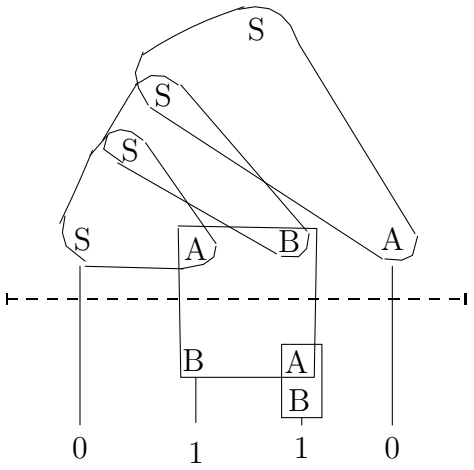

Figura 4.1: Hiperarbori asociaţi unei derivări

Observaţie. Ţinând cont că gramaticile dependente de context sunt echivalente cu gramaticile monotone, rezultă că orice gramatică dependentă de context este echivalentă cu o gramatică liniar mărginită, cu alte cuvinte admite forma normală.

Structura derivărilor într-o gramatică liniar mărginită. Deoarece $C, F, H \neq S$ rezultă că în orice derivare dintr-o gramatică liniar mărgintă se aplică mai întâi un număr oarecare (să zicem m) de reguli de forma 1),apoi reguli de forma 2) şi 3). Este clar că lungimea cuvântului este $m + 1$.

Exemplu. Considerăm gramatica

$$S \to SA|SB,$$
$$A \to B,\ AB \to BA,$$
$$S \to 0,\ A \to 0,\ B \to 1.$$

Putem scrie derivarea

$$S \Rightarrow SA \Rightarrow SBA \Rightarrow SABA \Rightarrow SBAA \Rightarrow SBBA \overset{*}{\Rightarrow} 0110.$$

Corespunzător acestei derivări putem figura hiperarborele din figura 4.1. Să observăm că până la linia punctată s-au aplicat trei reguli de forma $S \to SA$, obţinându–se cuvântul $SABA$ apoi două reguli de forma (2),(3) şi apoi reguli de forma $X \to i$.

4.2 Automate liniar mărginite

Un automat liniar mărginit (ALM) se compune dintr–o bandă de intrare şi un dispozitiv de comandă care posedă un dispozitiv de scriere–citire pe banda de

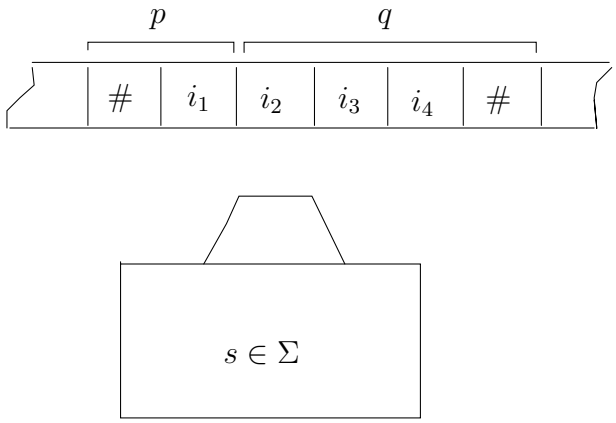

Figura 4.2: Reprezentarea schematică a unui automat liniar mărginit

intrare. Banda de intrare conţine simboluri ale unui alfabet de intrare, constituind un cuvânt de intrare. Alfabetul de intrare conţine un simbol special numit "marcaj" şi notat "#"(diez) utilizat pentru delimitarea cuvintelor. Banda se poate mişca în ambele sensuri, sau să rămână pe loc; prin convenţie vom nota, deplasarea spre stânga cu $+1$, deplasarea spre dreapta cu -1 şi rămânerea pe loc cu 0. Dispozitivul de comandă se află întotdeauna într-o anumită stare internă, element al unei mulţimi finite de stări.

Schematic un ALM are forma din figura 4.2. Un ALM funcţionează în paşi discreţi. Un pas de funcţionare comportă:

(a) Dispozitivul de comandă citeşte simbolul din dreptul dispozitivului de scriere–citire;

(b) În funcţie de starea internă şi de simbolul citit, automatul trece într-o nouă stare, scrie pe bandă un nou simbol (în locul simbolului citit) şi mută banda cu o poziţie (stânga, dreapta) sau o lasă pe loc.

Observaţie. Oricare ar fi starea internă a dispozitivului de comandă, citirea marcajului provoacă scrierea din nou a marcajului; de asemenea scrierea marcajului are loc numai în această situaţie (adică dacă s-a citit marcaj). Prin urmare, cele două marcaje care delimitează un cuvânt nu se modifică în urma funcţionării.

Funcţionarea unui ALM încetează în momentul în care s-au citit toate simbolurile cuvântului scris pe banda de intrare, mai exact în momentul în care s-a citit al doilea marcaj. Starea dispozitivului de comandă în acest moment este starea finală a automatului.

Matematic, un ALM este un sistem de forma $ALM = (\Sigma, I, f, s_0, \Sigma_f)$ unde:

- Σ este mulţimea de stări (finită şi nevidă);
- I este alfabetul de intrare;
- $f : \Sigma \times I \longrightarrow \mathcal{P}(\Sigma \times I \times \{+1, 0, -1\})$ este funcţia de evoluţie;

- $s_0 \in \Sigma$ şi constituie starea iniţială;
- $\Sigma_f \subseteq \Sigma$ este mulţimea de stări finale.

Observaţie. Avem

$$f(s,i) = (s', \#, k) \quad \Leftrightarrow \quad i = \#.$$

O stare sau o configuraţie instantanee a unui ALM este un triplet de forma (s,p,q) unde p este partea din cuvântul de pe banda de intrare din stânga dispozitivului de scriere–citire, q este partea din dreapta a acestui cuvânt, inclusiv simbolul din dreptul dispozitivului de scriere–citire, iar s starea internă. Zicem că o stare $\sigma_1 = (p_1, s_1, q_1)$ *se transformă* (sau *evoluează*) direct în starea $\sigma_2 = (p_2, s_2, q_2)$ (notăm $\sigma_1 \vdash\!\!\longrightarrow \sigma_2$) dacă

$$\begin{array}{l}(1)\ q_1 = i_1 q_1',\ f(s_1,i_1) = (s_2, i_2, +1),\ p_2 = p_1 i_2, q_2 = q_1';\\ (2)\ q_1 = i_1 q_1',\ f(s_1,i_1) = (s_2, i_2, 0),\ p_2 = p_1, q_2 = i_2 q_1';\\ (3)\ q_1 = i_1 q_1',\ p_1 = p_1' j,\ f(s_1,i_1) = (s_2, i_2, -1),\ p_2 = p_1', q_2 = j i_2 q_1'.\end{array}$$

Vom spune că starea σ' *evoluează* (fără specificaţia direct) în starea σ'' şi vom scrie $\sigma' \vdash\!\!\overset{*}{\longrightarrow} \sigma''$ dacă $\sigma' = \sigma''$ sau dacă $\exists \sigma_1, \ldots, \sigma_n$ astfel încât

$$\sigma' = \sigma_1 \vdash\!\!\longrightarrow \sigma_2 \vdash\!\!\longrightarrow \ldots \vdash\!\!\longrightarrow \sigma_n = \sigma''.$$

Definiţie 4.3 *Limbajul recunoscut de un ALM este*

$$L(ALM) = \{p | p \in I^*, (\lambda, s_0, p) \vdash\!\!\overset{*}{\longrightarrow} (q, s, \lambda), s \in \Sigma_f\}.$$

Limbajele recunoscute de ALM. Automatele liniar mărginite sunt mecanisme care recunosc limbajele dependente de context, deci reprezintă o modalitate analitică de definire a acestor limbaje.

Teorema 4.4 *Un limbaj este dependent de context dacă şi numai dacă este recunoscut de un automat liniar mărginit.*

Demonstraţie. Partea I: $E \in \mathcal{L}_1 \ \Rightarrow \ E = L(ALM)$.

Fie $G = (V_N, V_T, S, P)$ o gramatică de tipul 1 presupusă în forma normală, astfel încât $E = L(G)$. Construim automatul liniar mărginit $ALM = (\Sigma, I, f, s_0, \Sigma_f)$ unde

$\Sigma = \{s_0, s_1, s_2, s_3, s_4\} \cup \{s_D | D \in V_N, DE \to FH \in P\}$,

$I = V_N \cup V_T \cup \{\#, \flat\}$,

$\Sigma_f = \{s_3\}$,

iar funcţia de evoluţie o definim punând în evidenţă patru categorii de relaţii:

$$Cat.I; \quad \begin{array}{l}(1) f(s_0, \#) = (s_0, \#, 1),\\ (2) f(s_3, \#) = (s_3, \#, 1),\end{array}$$

Figura 4.3:

$$Cat.II;\quad \left.\begin{array}{l}(3) f(s_0,\#) = (s_4,\#,1),\\ (4) f(s_4,\#) = (s_3,\#,1),\end{array}\right\} pentru\ \lambda \in E;$$

$$Cat.III;\quad \begin{array}{l}(5) f(s_0,x) = \{(s_0,x,1),(s_0,x,-1)\}, \forall x \in V_N \cup V_T;\\ (6) f(s_0,y) = (s_0,B,0),\ B \to y \in P,\ y \in V_N \cup V_T,\\ (7) f(s_0,F) = (s_D,D,1),\ DE \to FH \in P\\ (8) f(s_D,H) = (s_0,E,0),\ DE \to FH \in P\end{array}$$

$$Cat.IV;\quad \begin{array}{l}(9) f(s_3,A) = (s_0,S,0),\ S \to SA \in P\\ (10) f(s_0,S) = (s_1,S,-1),\\ (11) f(s_1,\#) = (s_2,\#,1),\\ (12) f(s_2,S) = (s_3,\flat,1),\\ (13) f(s_1,\flat) = (s_2,\flat,1)\end{array}$$

Relaţiile din categoria I servesc la "pornirea" şi respectiv "oprirea" automatului. Categoria II apar numai dacă gramatica conţine regula $S \to \lambda$ şi servesc la recunoaşterea cuvântului vid λ. Regulile din categoria III se utilizează pentru reducerea succesivă a hiperarborelui de derivare şi anume partea construită cu reguli de forma 2)şi 3) (partea situată sub linia punctată de la exemplul din figura 4.1). În adevăr, cu ajutorul relaţiei (5) putem plasa dispozitivul de comandă sub oricare din simbolurile cuvântului curent. Apoi, cu ajutorul relaţiilor (6) şi respectiv (7),(8) putem reduce subhiperarbori de forma prezentată în figura 4.3. corespunzători regulilor de forma $B \to y$ şi $DE \to FH$. De exemplu, avem următoarea funcţionare (vezi figura 4.4).

În acest fel ajungem la partea triunghiulară a hiperarborelui (situată în exemplu deasupra liniei punctate) care va avea pe frontieră un cuvânt de forma $\#SXY \ldots Z\#$ cu $X, Y, \ldots, Z \in V_N$. Această parte va fi redusă cu relaţiile din categoria IV. Astfel, după plasarea dispozitivului de citire–scriere pe S, cu ajutorul relaţiei (10), dispozitivul se plasează pe marcajul de început şi trece în starea s_1, după care revine pe S în starea s_2 (relaţia 11). În continuare au loc secvenţe de forma din figura 4.5.

În urma acestor secvenţe pe bandă se va scrie cuvântul $\#\flat\flat \ldots \flat\#$ iar dispozitivul se va plasa pe marcajul de sfârşit în starea s_3. În sfârşit, cu ajutorul relaţiei (2) funcţionarea automatului este oprită.

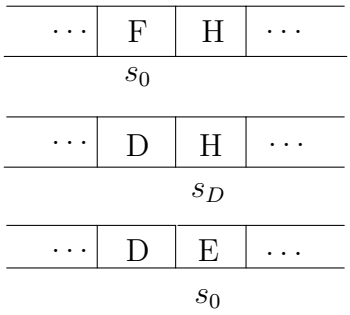

Figura 4.4: Reducerea regulii $DE \to FH$.

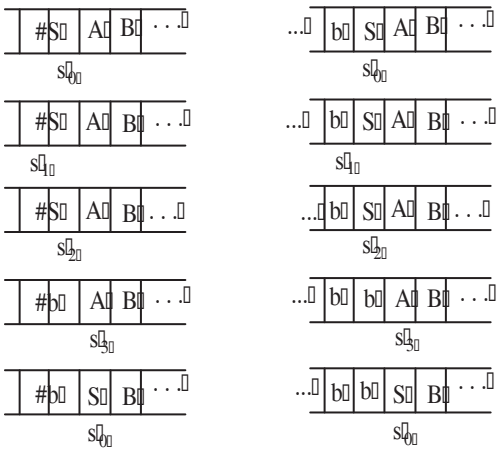

Figura 4.5: Reducerea regulilor $S \to SA$.

Este clar că dacă $p \in E$ atunci automatul definit mai sus va avea o evoluţie de acest tip şi reciproc, deci $E = L(ALM)$.□

Exemplu: Evoluţia ALM corespunzător gramaticii de la exemplul din paragraful precedent pentru cuvântul #0110# este:

$(\lambda, s_0, \#0110\#) \longmapsto (\#, s_0, 0110\#) \longmapsto (\#0, s_0, 110\#) \longmapsto (\#01, s_0, 10\#) \longmapsto$
$(\#01, s_0, B0\#) \longmapsto \ldots \longmapsto (\#S, s_0, BAA\#) \longmapsto (\#SA, s_D, AA\#) \longmapsto$
$(\#SA, s_0, BA\#) \longmapsto \ldots \longmapsto (\#, s_0, SABA\#) \longmapsto (\lambda, s_1, \#SBAA\#) \longmapsto$
$(\#, s_2, SABA\#) \longmapsto (\#\flat, s_3, ABA\#) \longmapsto (\#\flat, s_0, SBA\#) \longmapsto (\#, s_1, \flat SBA\#) \longmapsto$
$(\#\flat, s_2, SBA\#) \longmapsto (\#\flat\flat, s_3, BA\#) \longmapsto (\#\flat\flat, s_0, SA\#) \longmapsto \ldots \longmapsto (\#\flat\flat\flat\flat, s_3, \#) \longmapsto$
$(\#\flat\flat\flat\flat\#, s_3, \lambda)$

Capitolul 5

Limbaje de tipul zero

5.1 Forma normală

Limbajele de tipul 0 sunt generate de gramatici Chomsky fără restricţii asupra regulilor de derivare. Se poate arăta că orice gramatică de tipul zero acceptă următoarea formă normală

$$S \to SA,$$
$$B \to C|i|\lambda,$$
$$DE \to FH,$$

unde $A, C, F, H \neq S$. Se observă că singura deosebire faţă de gramaticile dependente de context este aceea că putem avea şi λ–reguli.

Demonstraţia existenţei formei normale urmează în linii mari aceiaşi cale cu cea de la limbaje dependente de context. În prealabil gramatica se completează cu neterminale noi astfel încât orice regulă care nu este de ştergere să respecte condiţia de monotonie. Acest lucru se poate realiza după cum urmează. Fie

$$X_1 \dots X_n \to Y_1 \dots Y_m, n > m,$$

o regulă care nu respectă condiţia de monotonie. Vom pune

$$X_1 \dots X_n \to Y_1 \dots Y_m Z_{m+1} \dots Z_n,\ Z_{m+1} \to \lambda, \dots, Z_n \to \lambda.$$

Este clar că orice derivare directă $u \Rightarrow v$ în gramatica iniţială se poate obţine şi în gramatica modificată şi reciproc (neterminalele Z nu pot apărea în stânga altor relaţii). În acest fel, gramatica se va transforma astfel încât să conţină două categorii de reguli: reguli care respectă condiţia de monotonie şi λ–reguli.

În continuare reducem ordinul regulilor monotone, urmând aceiaşi procedură ca la limbaje dependente de context, până la ordinul 2 inclusiv (λ–regulile nu intervin cu nimic în acest proces). În final obţinem reguli de următoarele categorii:

(1) $B \to C|i|\lambda$,

(2) $DE \to FH$,

(3) $X \to YZ$.

Regulile de forma (3) cu $X \neq S$ sau $Y \neq S$ le eliminăm cu procedura cunoscută plus un simbol de start nou, cu reguli stâng recursive cunoscute pentru neterminalele nou introduse.

Relativ la puterea generativă a gramaticilor de tipul zero se poate arăta că $\mathcal{L}_1 \subset \mathcal{L}_0$ (strict). Demonstraţia se face pe o cale indirectă (nu s-a găsit un exemplu de limbaj de tipul zero care să nu poată fi generat de o gramatică dependentă de context).

După cum am văzut în primul capitol limbajele de tipul zero sunt închise faţă de operaţiile regulate. Se poate arăta că familia $\mathcal{L}_0$ este închisă la intersecţie.

Menţionăm şi următoarea "teoremă a spaţiului de lucru". Fie G o gramatică de tipul zero. Pentru o derivare

$$D: \quad S = u_0 \Rightarrow u_1 \Rightarrow \ldots \Rightarrow u_n = p \in V_T^*$$

definim

$$WS_D(p) = \max_{i=1,\ldots,n} |u_i|$$

şi

$$WS(p) = \min_{D} \{WS_D(p)\}.$$

Observaţie. WS este prescurtare de la *working space*.

Spunem că o gramatică de tipul zero G are spaţiul de lucru mărginit dacă există $k \in N$ astfel încât pentru $\forall p \in L(G)$ să avem $WS(p) \leq k|p|$. Să observăm că orice gramatică care nu are reguli de ştergere (excepţie $S \to \lambda$ şi atunci S nu apare în dreapta) satisface această condiţie cu $k = 1$.

Teorema 5.1 *(teorema spaţiului de lucru) dacă o gramatică G are spaţiul de lucru mărginit, atunci $L(G) \in \mathcal{L}_1$.*

5.2 Maşina Turing

Conceptul de maşină Turing. Maşina Turing este un mecanism de recunoaştere a limbajelor de tipul zero. În felul acesta, familiilor de limbaje din clasificarea Chomsky le corespund automate specifice de recunoaştere.

Familia $\mathcal{L}_3$ - automate finite,
Familia $\mathcal{L}_2$ - automate push–down,
Familia $\mathcal{L}_1$ - automate liniar mărginite,
Familia $\mathcal{L}_0$ - maşina Turing.

O maşină Turing (prescurtat MT) se compune dintr–un dispozitiv de comandă care posedă un dispozitiv de scriere–citire şi o bandă de intrare. Banda de intrare se consideră mărginită la stânga şi nemărginită la dreapta; ea conţine simboluri

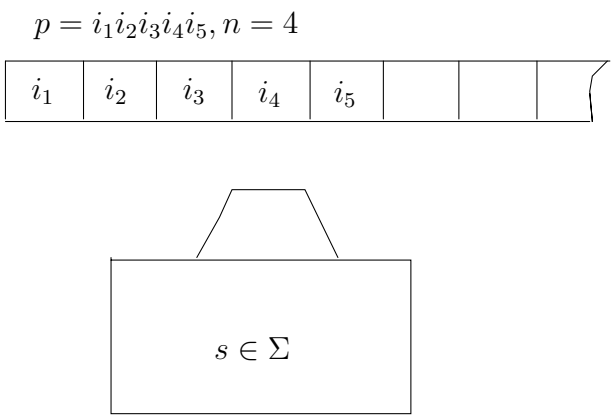

Figura 5.1: Reprezentarea schematică a maşinii Turing

ale unui alfabet de intrare, începând din prima poziţie. Alfabetul are un simbol special numit blanc şi notat $\flat$ (sau spaţiu), care se consideră înregistrat în toată partea neocupată a benzii. Indicele simbolului din dreptul dispozitivului de scriere–citire îl notăm cu n şi el va constitui un element al stării maşinii Turing. Dispozitivul de comandă se află într-o anumită stare internă, element al unei mulţimi finite de stări.

Schema unei maşini Turing este dată în figura 5.1. O maşină Turing funcţionează în paşi discreţi. Un pas de funcţionare constă din:

Dispozitivul de comandă citeşte simbolul aflat în dreptul dispozitivului de scriere–citire; în funcţie de simbolul citit, maşina Turing trece într-o nouă stare internă, scrie pe bandă un nou simbol (întotdeauna diferit de blanc) în locul simbolului citit şi mută banda cu o poziţie (spre stânga sau dreapta) sau o lasă pe loc. Convenţional, vom nota cu $+1$ o mişcare spre stânga, cu -1 spre dreapta şi cu 0 pe loc.

Din punct de vedere matematic, o maşină Turing este un sistem

$$MT = (\Sigma, I, f, s_0, \Sigma_f)$$

unde:

Σ este mulţimea de stări;

I este alfabetul de intrare;

$f : (\Sigma \times I)' \longrightarrow \Sigma \times (I - \{\flat\}) \times \{1, 0, -1\},\ (\Sigma \times I)' \subseteq \Sigma \times I,$
este funcţia de evoluţie;

$s_0 \in \Sigma$ este simbolul de start;

$\Sigma_f \subseteq \Sigma$ este mulţimea de stări finale.

Observaţie. Funcţia f nu este definită pe întregul produs $\Sigma \times I$; dacă maşina ajunge într-o stare s iar în dreptul dispozitivului de scriere–citire se află simbolul i şi $(s, i) \notin (\Sigma \times I)'$ spunem că maşina se blochează. Există şi alte cazuri de blocare, de exemplu situaţia în care dispozitivul de scriere–citire se află în dreptul primului simbol, iar pasul de funcţionare prevede o mişcare a benzii spre dreapta.

O stare (sau configuraţie) a unei maşini Turing este un triplet de forma $\sigma = (s, p, n)$ unde s este starea internă, p este cuvântul scris pe bandă iar n este indicele simbolului din dreptul dispozitivului de scriere–citire.

Vom spune că starea $\sigma_1 = (s_1, p_1, n_1)$ *evoluează direct* în starea $\sigma_2 = (s_2, p_1, n_2)$ şi vom scrie $\sigma_1 \vdash\!\!\!\longrightarrow \sigma_2$ dacă se efectuează un pas de evoluţie. În termenii funcţiei de evoluţie, avem

$(1) f(s_1, i_{n_1}) = (s_2, i, +1),\ p_2 = i_1 \dots i_{n_1-1} i i_{n_1+1} \dots i_m,\ n_2 = n_1 + 1;$
$(2) f(s_1, i_{n_1}) = (s_2, i, -1),\ p_2 = i_1 \dots i_{n_1-1} i i_{n_1+1} \dots i_m,\ n_2 = n_1 - 1;$
$(3) f(s_1, i_{n_1}) = (s_2, i, 0),\ p_2 = i_1 \dots i_{n_1-1} i i_{n_1+1} \dots i_m,\ n_2 = n_1;$
$(4) f(s_1, \flat) = (s_2, i, +1),\ p_2 = p_1 i,\ n_2 = n_1 + 1;$
$(5) f(s_1, \flat) = (s_2, i, -1),\ p_2 = p_1 i,\ n_2 = n_1 - 1;$
$(6) f(s_1, \flat) = (s_2, i, 0),\ p_2 = p_1 i,\ n_2 = n_1;$

Vom spune că σ' *evoluează* (fără specificaţia direct) în σ'' şi vom nota $\sigma' \vdash\!\!\!\overset{*}{\longrightarrow} \sigma''$ dacă $\sigma' = \sigma''$ sau dacă există $\sigma_1, \dots, \sigma_n$ astfel încât

$$\sigma' = \sigma_1 \vdash\!\!\!\longrightarrow \sigma_2 \vdash\!\!\!\longrightarrow \dots \vdash\!\!\!\longrightarrow \sigma_n = \sigma''.$$

Limbajul recunoscut de o maşină Turing este prin definiţie

$$L(MT) = \{p | p \in I^*, (s_0, p, 1) \vdash\!\!\!\overset{*}{\longrightarrow} (s, q, \epsilon), s \in \Sigma_f\}.$$

Observaţie. Este posibil ca maşina să ajungă într-o stare finală înainte de citirea integrală a lui p; analiza stării finale trebuie făcută numai după parcurgerea cuvântului.

Exemplu. Considerăm maşina turing $MT = (\Sigma, I, f, s_0, \Sigma_f)$ unde $\Sigma = \{s_0, s_1, s_2\}$, $I = \{0, 1\}$, $\Sigma_f = \{s_1\}$ iar funcţia de evoluţie este dată de

f	s_0	s_1	s_2
$\flat$	$(s_2, 1, 1)$		$(s_0, 0, -1)$
0	$(s_1, 0, 1)$		
1	$(s_0, 1, 1)$	$(s_0, 0, 1)$	

Evoluţia maşinii pentru $p = 001$ este

$$(s_0, 011, 1) \vdash\!\!\!\longrightarrow (s_1, 011, 2) \vdash\!\!\!\longrightarrow (s_0, 001, 3) \vdash\!\!\!\longrightarrow (s_0, 001, 4) \vdash\!\!\!\longrightarrow$$

$$(s_2, 0011, 5) \vdash\!\!\!\longrightarrow (s_0, 0011, 4) \vdash\!\!\!\longrightarrow (s_0, 00110, 5) \vdash\!\!\!\longrightarrow (s_1, 00110, 6)$$

Se poate observa că după citirea întregului cuvânt maşina poate să efectueze un număr de paşi suplimentari până la ajungerea într-o stare finală, deci este posibil ca $|p| < |q|$.

Situaţii în care o maşină Turing se blochează (în aceste situaţii cuvântul scris pe bandă nu este recunoscut):

1. MT ajunge într-o stare s, în dreptul dispozitivului de citire–scriere se află simbolul i şi $(s, i) \notin (\Sigma \times I)'$;

2. MT este în starea s, a citit simbolul din prima poziţie i şi $f(s,i) = (s', i', -1)$;

3. MT efectuează un ciclu infinit în interiorul cuvântului.

Definiţie 5.1 *Vom spune că o maşină Turing este nestaţionară dacă*

$$f : (\Sigma \times I)' \to \Sigma \times (I \setminus \{\flat\}) \times \{-1, +1\}.$$

Prin urmare, o maşină Turing nestaţionară nu lasă în nici o situaţie banda pe loc.

Lema 5.1 *Orice maşină Turing este echivalentă cu o maşină Turing nestaţionară.*

Demonstraţie. Pornind de la o MT dată construim o MT' nestaţionară astfel:
Dacă $f(s,i) = (s', i', \pm 1)$ vom pune $f'(s,i) = f(s,i)$;
Dacă $f(s,i) = (s', i', 0)$ vom pune $f'(s,i) = (s'', i', -1)$ şi $f'(s'', j) = (s', j, +1), \forall j \in I$, unde s'' este o stare nou introdusă.

Astfel, în cazul unei rămâneri pe loc a lui MT, noua maşină va face un pas spre stânga şi unul spre dreapta, fără să modifice nici starea şi nici conţinutul benzii. Evident, cele două maşini sunt echivalente.

Limbajele recunoscute de maşini Turing.

Teorema 5.2 *Un limbaj este recunoscut de o maşină Turing dacă şi numai dacă este de tipul zero.*

Demonstraţie. Partea I. $E = L(MT) \Rightarrow E \in \mathcal{L}_0$.

Fie $MT = (\Sigma, I, f, s_0, \Sigma_f)$ o maşină Turing astfel încât $E = L(MT)$. Putem presupune că MT este nestaţionară. Fie $I_\lambda = (I \setminus \{\flat\}) \cup \{\lambda\}$.

Construim o gramatică de tipul zero astfel: $G = (V_N, V_T, S, P)$ unde

$$V_N = \Sigma \cup \{(i,j) | i \in I_\lambda, j \in I\} \cup \{S, X_1, X_2\};\ V_T = I \setminus \{\flat\}.$$

Definiţia lui P:

$S \to s_0 X_1,\ X_1 \to (i,i)X_1,\ i \in I \setminus \{\flat\}$,

$X_1 \to X_2,\ X_2 \to (\lambda, \flat)X_2,\ X_2 \to \lambda$,

dacă $f(s,i) = (s', i', 1)$ atunci $s(i_1, i) \to (i_1, i')s',\ \forall i_1 \in I_\lambda$,

dacă $f(s,i) = (s', i', -1)$ atunci $(i_1, i_2)s(i_3, i) \to s'(i_1, i_2)(i_3, i'),\ i_1, i_3 \in I_\lambda,\ i_2 \in I$,

dacă $s \in \Sigma_f$, atunci $s(i_1, i_2) \to s i_1 s$ şi $(i_1, i_2)s \to s i_1 s$.

Fie $p \in L(MT), p = i_1 \dots i_n$. Presupunem că MT utilizează în recunoaştere m poziţii de pe bandă situate la dreapta cuvântului p. În G avem

$$S \Rightarrow s_0 X_1 \overset{*}{\Rightarrow} s_0(i_1, i_1) \dots (i_n, i_n) X_2 \overset{*}{\Rightarrow} s_0(i_1, i_1) \dots (i_n, i_n)(\lambda, \flat)^m.$$

Cuvântul p fiind recunoscut de MT avem

$$(1)\ \ (s_0, i_1 \dots i_n, 1) \overset{*}{\longmapsto} (s_f, i'_1 \dots i'_h, k),\ h \geq n.$$

Vom arăta că (1) implică existenţa unei derivări de forma

$$2) s_0(i_1, i_1) \dots (i_n, i_n)(\lambda, \flat)^m \overset{*}{\Rightarrow} (i_1, i'_1) \dots (i_{k-1}, i'_{k-1}) s_f (i_k, i'_k) \dots (i_{n+m}, i'_{n+m}),$$

unde

$$i_1, \dots, i_n \in I \setminus \{\flat\},\ i_{n+1}, \dots, i_{n+m} = \lambda,$$
$$i'_1, \dots, i'_h \in I \setminus \{\flat\},\ i'_{h+1}, \dots, i'_{n+m} = \lambda.$$

Demonstraţie prin inducţie asupra lungimii l a evoluţiei.

Pentru $l = 0$ avem

$$(s_0, i_1 \dots i_n, 1) \overset{*}{\longmapsto} (s_0, i_1 \dots i_n, 1), k = 1, h = n, i'_j = i_j.$$

Partea dreaptă a lui 2) va avea forma $s_0(i_1, i_1) \dots (i_n, i_n)(\lambda, \flat)^m$ şi deci 2) este adevărată.

Presupunem că implicaţia este adevărată pentru l oarecare şi considerăm o evoluţie de lungime $l + 1$. Punem în evidenţă ultima evoluţie directă

$$(s_0, i_1 \dots i_n, 1) \overset{*}{\longmapsto} (s, i''_1 \dots i''_g, j) \longmapsto (s_f, i'_1 \dots i'_h, k).$$

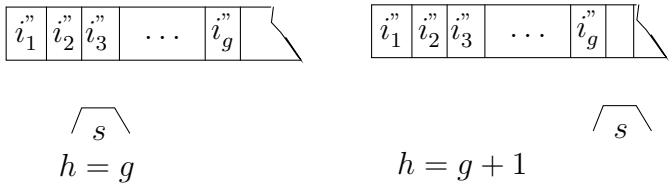

Figura 5.2: Configuraţii posibile ale maşinii Turing

În general $h = g$ sau $h = g + 1$ în conformitate cu următoarele două cazuri (figura 5.2).

Întotdeauna $k = j \pm 1$. apoi $i_t'' = i_t'$, $t = 1, \ldots, g, t \neq j$, adică

$$i_1'', i_2'', \ldots, i_{j-1}'', i_j'', i_{j+1}'', \ldots, i_g'';$$
$$i_1', i_2', \ldots, i_{j-1}', i_j', i_{j+1}', \ldots, i_g'.$$

În urma ultimei evoluţii directe vor diferi numai simbolurile i_j'', i_j'.

Din ipoteza inductivă rezultă

$$s_0(i_1, i_1) \ldots (i_n, i_n)(\lambda, \flat)^m \overset{*}{\Rightarrow} (i_1, i_1'') \ldots (i_{j-1}, i_{j-1}'')s(i_j, i_j'') \ldots (i_{n+m}, i_{n+m}'').$$

În conformitate cu definiţia evoluţiei directe avem

$$f(s, i_j'') = (s_f, i_j', 1),\ k = j + 1, s(i_j, i_j'') \to (i_j, i_j'')s_f;$$
$$f(s, i_j'') = (s_f, i_j', -1),\ k = j - 1, (i_{j-1}, i_{j-1}'')s(i_j, i_j'') \to s_f(i_j, i_{j-1}'')(i_j, i_j').$$

În ambele cazuri

$$s_0(i_1, i_1) \ldots (i_n, i_n)(\lambda, \flat)^m \overset{*}{\Rightarrow} (i_1, i_1') \ldots (i_{k-1}, i_{k-1}')s_f(i_k, i_k') \ldots (i_{n+m}, i_{n+m}').$$

Acum, deoarece $s_f \in \Sigma_f$, putem scrie

$$\begin{aligned} S &\overset{*}{\Rightarrow} s_0(i_1, i_1) \ldots (i_n, i_n)(\lambda, \flat)^m \\ &\overset{*}{\Rightarrow} (i_1, i_1') \ldots (i_{k-1}, i_{k-1}')s_f(i_k, i_k') \ldots (i_{m+n}, i_{m+n}') \\ &\Rightarrow (i_1, i_1') \ldots (i_{k-1}, i_{k-1}')s_f i_k s_f(i_{k+1}, i_{k+1}') \ldots (i_{m+n}, i_{m+n}') \\ &\Rightarrow (i_1, i_1') \ldots (i_{k-2}, i_{k-2}')s_f i_{k-1} s_f i_k s_f i_{k+1} s_f(i_{k+2}, i_{k+2}') \ldots (i_{m+n}, i_{m+n}') \\ &\overset{*}{\Rightarrow} s_f i_1 s_f i_2 s_f \ldots s_f i_n s_f \overset{*}{\Rightarrow} i_1 \ldots i_n = p. \end{aligned}$$

Prin urmare $p \in L(G)$ şi $L(MT) \subseteq L(G)$. Analog se arată şi incluziunea inversă şi deci $L(MT) = L(G)$.□

Cuprins

Bibliografie

1. Octavian C. Dogaru, *Bazele informaticii. Limbaje formale*, Tipografia Universităţii din Timişoara, 1989.

2. Gheorghe Grigoraş, *Limbaje formale şi tehnici de compilare*, Tipografia Universităţii "Alexandru Ioan Cuza", Iaşi, 1984.

3. J. E. Hopcroft şi J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Reading Mass., 1979.

4. Solomon Marcus, *Gramatici şi automate finite*, Editura Academiei, Bucureşti, 1964.

5. Ştefan Măruşter, *Curs de Limbaje formale şi tehnici de compilare*, Tipografia Universităţii din Timişoara, 1980.

6. Gheorghe Orman, *Limbaje formale*, Editura tehnică, Bucureşti, 1982.

7. Gheorghe Păun, *Probleme actuale în teoria Limbajelor formale*, Editura Ştiinţifică şi Enciclopedică, Bucureşti, 1984.

8. Teodor Rus, *Mecanisme formale pentru specificarea limbajelor*, Editura Academiei, Bucureşti, 1983.

9. Arto Salomma, *Formal languages*, Academic Press, New York, 1973.

10. Dan Simovici, *Limbaje formale şi tehnici de compilare*, Editura didactică şi pedagogică, Bucureşti, 1978.

11. Luca Dan Şerbănaţi, *Limbaje de programare şi compilatoare*, Editura Academiei, Bucureşti, 1987.