Csörnyei Zoltán **Kása Zoltán**

Formális nyelvek és fordítóprogramok

Csörnyei Zoltán Kása Zoltán

FORMÁLIS NYELVEK ÉS FORDÍTÓPROGRAMOK

2007

ISBN ...

Tartalomjegyzék

I. RÉSZ

Formális nyelvek és automaták

1. FEJEZET

Nyelvek és nyelvtanok

Tetszőleges szimbólumok (jelek) nem üres, véges halmazát ***ábécének*** nevezzük. Az ábécé elemeit ***betűknek*** nevezzük, de sokszor használjuk a jel és szimbólum neveket is. Ábécé például a $\Sigma = \{a, b, c, d, 0, 1, \sigma\}$, amelynek betűi $a, b, c, d, 0, 1$ és σ.

Az ábécé betűiből szavakat képezünk. Ha $a_1, a_2, \ldots, a_n \in \Sigma, n \geq 0$, akkor $a_1 a_2 \ldots a_n$ a Σ ábécé betűiből képzett ***szó*** (az a_i-k nem feltétlenül különbözők). A szót alkotó betűk száma, multiplicitással számolva, a szó ***hossza***. Ha $w = a_1 a_2 \ldots a_n$, akkor a w szó hossza $|w| = n$. Ha $n = 0$, akkor a szó egyetlen betűt sem tartalmaz, és ***üres szónak*** nevezzük. Jelölése: ε (sok könyvben λ). A Σ betűiből képezhető szavak halmazának jelölésére a Σ^* jelet használjuk:

$$\Sigma^* = \left\{ a_1 a_2 \ldots a_n \mid a_1, a_2, \ldots, a_n \in \Sigma,\, n \geq 0 \right\} .$$

A nem üres szavak halmaza $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. Az n hosszúságú szavak halmazát Σ^n-nel jelöljük, és természetesen $\Sigma^0 = \{\varepsilon\}$. Ekkor

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \cdots \cup \Sigma^n \cup \cdots \qquad \text{és} \qquad \Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \cdots \cup \Sigma^n \cup \cdots .$$

Az $u = a_1 a_2 \ldots a_m$ és $v = b_1 b_2 \ldots b_n$ szavak egyenlők (azaz $u = v$), ha $m = n$ és $a_i = b_i, \;\; i = 1, 2, \ldots, n$.

A Σ^*-on bevezetünk egy bináris műveletet, a konkatenációt. Az $u = a_1 a_2 \ldots a_m$ és $v = b_1 b_2 \ldots b_n$ szavak ***konkatenációján*** (illesztésén, szorzatán, összefűzésén) az $uv = a_1 a_2 \ldots a_m b_1 b_2 \ldots b_n$ szót értjük. Nyilvánvaló, hogy $|uv| = |u| + |v|$. Ez a művelet asszociatív, de nem kommutatív. Van egységeleme, az ε, mivel $\varepsilon u = u\varepsilon = u$ tetszőleges $u \in \Sigma^*$ szóra. Tehát Σ^* ezzel a konkatenáció művelettel monoidot (egységelemes félcsoportot) képez.

Bevezetjük a szavak hatványozását. Ha $u \in \Sigma^*$, akkor $u^0 = \varepsilon$, továbbá ha $n \geq 1$, akkor $u^n = u^{n-1} u$.

Az $u = a_1 a_2 \ldots a_n$ szó ***tükörképe*** $u^{-1} = a_n a_{n-1} \ldots a_1$. Az u tükörképének jelölésére néha használják még a következőket is: u^R, $\tilde{u}$. Nyilvánvaló, hogy $\left(u^{-1}\right)^{-1} = u$ és $(uv)^{-1} = v^{-1} u^{-1}$.

Azt mondjuk, hogy v ***kezdőszelete*** (prefixe) az u szónak, ha létezik a z szó

úgy, hogy $u = vz$. Ha $z \neq \varepsilon$, akkor v valódi kezdőszelete u-nak. Hasonlóképpen v ***végszelete*** (szuffixe) az u szónak, ha létezik az x szó úgy, hogy $u = xv$, és analóg módon értelmezhető a valódi végszelet is. A v szó ***részszava*** az u szónak, ha léteznek a p és q szavak úgy, hogy $u = pvq$. Ha $pq \neq \varepsilon$, akkor v ***valódi részszava*** u-nak. A kezdő- és végszeletek egyben részszavak is.

A Σ^* tetszőleges L részhalmazát a Σ ábécé feletti ***nyelvnek*** nevezzük. Mivel a szavaknak nem tulajdonítunk értelmet, gyakran ***formális nyelvről*** beszélünk, hogy megkülönböztessük a természetes és a mesterséges nyelvektől, amelyekben a szavakhoz valamilyen értelem is kapcsolódik. Megjegyezzük, hogy $\emptyset$ az üres nyelv, míg $\{\varepsilon\}$ az üres szóból álló nyelv.

1.1. Műveletek nyelvekkel

Ha L, L_1, L_2 egy-egy Σ feletti nyelv, akkor értelmezzük a következő műveleteket:

- *egyesítés*
$$L_1 \cup L_2 = \{u \in \Sigma^* \mid u \in L_1 \text{ vagy } u \in L_2\} ,$$
- *metszet*
$$L_1 \cap L_2 = \{u \in \Sigma^* \mid u \in L_1 \text{ és } u \in L_2\} ,$$
- *különbség*
$$L_1 \setminus L_2 = \{u \in \Sigma^* \mid u \in L_1 \text{ és } u \notin L_2\} ,$$
- *komplementum (komplemens* vagy *komplementer nyelv)*
$$\overline{L} = \Sigma^* \setminus L ,$$
- *szorzat*
$$L_1 L_2 = \{uv \mid u \in L_1, \text{ és } v \in L_2\} ,$$
- *nyelv hatványa*
$$L^0 = \{\varepsilon\}, \qquad L^n = L^{n-1}L, \text{ ha } n \geq 1 ,$$
- *nyelv iteráltja*
$$L^* = \bigcup_{i=0}^{\infty} L^i = L^0 \cup L \cup L^2 \cup \cdots \cup L^i \cup \cdots ,$$
- *tükrözés*
$$L^{-1} = \{u^{-1} \mid u \in L\}$$

Szoktuk még használni az L^+ jelölést is:

$$L^+ = \bigcup_{i=1}^{\infty} L^i = L \cup L^2 \cup \cdots \cup L^i \cup \cdots .$$

Az egyesítés, szorzás, iteráció műveleteket ***reguláris műveleteknek*** nevezzük.

1.2. Nyelvek megadása

Nyelveket többféleképpen adhatunk meg. Például:
1) felsoroljuk a szavait,
2) megadunk egy tulajdonságot, amellyel a nyelv minden szava rendelkezik, de más szavak nem,
3) nyelvtan segítségével.

1.2.1. Nyelvek megadása elemeik felsorolásával

Nyelvek például:
$L_1 = \{\varepsilon, 0, 1\}$,
$L_2 = \{a, aa, aaa, ab, ba, aba\}$.
Habár végtelen halmazokat nem tudunk felsorolni, a végtelen nyelvek is megadhatók felsorolással abban az esetben, ha az első néhány szó megadásából kiderül, hogyan kell a felsorolást folytatni. Ilyen például a következő nyelv:
$L_3 = \{\varepsilon, ab, aabb, aaabbb, aaaabbbb, \ldots\}$.

1.2.2. Nyelvek megadása tulajdonság segítségével

Nyelvek a következő halmazok is:
$L_4 = \{a^n b^n \mid n = 0, 1, 2, \ldots\}$,
$L_5 = \{uu^{-1} \mid u \in \Sigma^*\}$,
$L_6 = \{u \in \{a, b\}^* \mid n_a(u) = n_b(u)\}$,
ahol $n_a(u)$ az u szóban levő a betűk számát, $n_b(u)$ pedig a b betűk számát jelöli.

1.2.3. Nyelvek megadása nyelvtannal

Értelmezzük a ***generatív nyelvtan*** vagy röviden ***nyelvtan*** (grammatika) fogalmát.

1.2.1. értelmezés. ***Generatív nyelvtannak*** *nevezzük a* $G = (N, T, P, S)$ *rendezett négyest, ahol*

- N *a* ***változók*** *(vagy nemterminális jelek) ábécéje,*
- T *a* ***terminális jelek*** *ábécéje, ahol* $N \cap T = \emptyset$,
- $P \subseteq (N \cup T)^* N (N \cup T)^* \times (N \cup T)^*$ *véges halmaz, vagyis* P *az* (u, v) *alakú* ***helyettesítési szabályok*** *véges halmaza, ahol* $u, v \in (N \cup T)^*$, *és* u *tartalmaz legalább egy nemterminális jelet,*
- $S \in N$ *a nyelvtan* ***kezdőszimbóluma.***

Megjegyzés. Az (u, v) jelölés helyett gyakran használjuk az $u \to v$ jelölést, amely szemléletesebben utal a helyettesítésre (amint azt később látni fogjuk).

Az $u \to v$, azaz (u, v) szabályban u-t a szabály bal, v-t pedig a jobb oldalának nevezzük. Amennyiben a nyelvtanban több olyan szabály van, amelyeknek a bal oldala azonos, akkor ezeket egyszerűbben is írhatjuk:

$$u \to v_1,\ u \to v_2, \ldots, u \to v_r \quad \text{helyett} \quad u \to v_1 \mid v_2 \mid \ldots \mid v_r\ .$$

Értelmezzük az $(N \cup T)^*$ halmazban a ***közvetlen levezetés*** relációt:

$$u \Longrightarrow v, \qquad \text{ha} \quad u = p_1 p p_2, \quad v = p_1 q p_2 \quad \text{és} \quad (p, q) \in P\ .$$

Tulajdonképpen u-ban a p részszó valamely előfordulását q-val helyettesítjük, és eredményül v-t kapjuk. A közvetlen levezetésre még szokás a $\vdash$ vagy a $\models$ jeleket is használni.

Ha hangsúlyozni szeretnénk a G nyelvtant, amelynek szabályait használjuk, akkor a $\Longrightarrow$ jelölés helyett a $\underset{G}{\Longrightarrow}$ jelölést használjuk. A $\Longrightarrow$ reláció reflexív és tranzitív lezártját $\overset{*}{\Longrightarrow}$, míg a tranzitív lezártját $\overset{+}{\Longrightarrow}$ jelöli. A $\overset{*}{\Longrightarrow}$ reláció neve ***levezetés***.

A reflexív és tranzitív lezárt definíciójából következik, hogy $u \overset{*}{\Longrightarrow} v$, ha léteznek a $w_0, w_1, \ldots, w_n \in (N \cup T)^*$, $n \geq 0$ szavak, és $u = w_0,\ w_0 \Longrightarrow w_1,\ w_1 \Longrightarrow w_2,\ \ldots, w_{n-1} \Longrightarrow w_n,\ w_n = v$. Ezt röviden így is írhatjuk: $u = w_0 \Longrightarrow w_1 \Longrightarrow w_2 \Longrightarrow \ldots \Longrightarrow w_{n-1} \Longrightarrow w_n = v$. Ha $n = 0$, akkor $u = v$. Hasonlóképpen értelmezhető az $u \overset{+}{\Longrightarrow} v$ reláció, azzal a különbséggel, hogy itt $n \geq 1$, tehát elvégzünk legalább egy helyettesítést

1.2.2. értelmezés. *A $G = (N, T, P, S)$ nyelvtan által* ***generált nyelv****:*

$$L(G) = \{u \in T^* \mid S \overset{*}{\Longrightarrow} u\}\ .$$

Tehát $L(G)$ mindazokat a T ábécé betűiből képzett szavakat tartalmazza, amelyek az S kezdőszimbólumból levezethetők P szabályainak a segítségével.

1.2.3. értelmezés. *Ha $w \in T^*$, akkor w-t* ***mondat****nak nevezzük. Ha $\alpha \in (N \cup T)^*$, akkor α neve* ***mondatforma****.*

1.2.4. példa. Legyen $G = (N, T, P, S)$, ahol

$N = \{S\}$,
$T = \{a, b\}$,
$P = \{S \to aSb,\ S \to ab\}$.

Könnyű belátni, hogy ekkor $L(G) = \{a^n b^n \mid n \geq 1\}$, mivel

$$S \underset{G}{\Longrightarrow} aSb \underset{G}{\Longrightarrow} a^2 S b^2 \underset{G}{\Longrightarrow} \cdots \underset{G}{\Longrightarrow} a^{n-1} S b^{n-1} \underset{G}{\Longrightarrow} a^n b^n\ ,$$

ahol az utolsó előtti helyettesítésig mindig az első helyettesítési szabályt $(S \to aSb)$

használtuk, míg az utolsónál a második helyettesítési szabályt ($S \to ab$). Ezt a levezetést röviden így is írhatjuk: $S \overset{*}{\underset{G}{\Longrightarrow}} a^n b^n$. Tehát $a^n b^n$ tetszőleges n-re levezethető S-ből, és más szót nem lehet levezetni S-ből. □

1.2.5. értelmezés. *Azt mondjuk, hogy a G_1 és G_2 nyelvtanok* ***ekvivalensek,*** *és ezt $G_1 \cong G_2$-vel jelöljük, ha $L(G_1) = L(G_2)$.*

1.2.6. példa. A következő két nyelvtan ekvivalens, mivel mindegyik az $\{a^n b^n c^n \mid n \geq 1\}$ nyelvet generálja.
$G_1 = (N_1, T, P_1, S_1)$, ahol
$N_1 = \{S_1, X, Y\}$, $T = \{a, b, c\}$,
$P_1 = \{S_1 \to abc,\ S_1 \to aXbc,\ Xb \to bX,\ Xc \to Ybcc,\ bY \to Yb,\ aY \to aaX,$
$aY \to aa\}$.
$G_2 = (N_2, T, P_2, S_2)$, ahol
$N_2 = \{S_2, A, B, C\}$,
$P_2 = \{S_2 \to aS_2BC,\ S_2 \to aBC,\ CB \to BC,\ aB \to ab,\ bB \to bb,\ bC \to bc,$
$cC \to cc\}$.
Először teljes indukcióval megmutatjuk, hogy $n \geq 2$-re $S_1 \overset{*}{\underset{G_1}{\Longrightarrow}} a^{n-1} Y b^n c^n$. Ha $n = 2$, akkor

$$S_1 \underset{G_1}{\Longrightarrow} aXbc \underset{G_1}{\Longrightarrow} abXc \underset{G_1}{\Longrightarrow} abYbcc \underset{G_1}{\Longrightarrow} aYb^2c^2 \ .$$

Feltételezzük, hogy $S_1 \overset{*}{\underset{G_1}{\Longrightarrow}} a^{n-2} Y b^{n-1} c^{n-1}$. Alkalmazzuk az $aY \to aaX$ szabályt, majd $(n-1)$-szer az $Xb \to bX$ szabályt, egyszer az $Xc \to Ybcc$ szabályt, utána pedig szintén $(n-1)$-szer a $bY \to Yb$ szabályt. Tehát

$$S_1 \underset{G_1}{\Longrightarrow} a^{n-2} Y b^{n-1} c^{n-1} \underset{G_1}{\Longrightarrow} a^{n-1} X b^{n-1} c^{n-1} \overset{*}{\underset{G_1}{\Longrightarrow}} a^{n-1} b^{n-1} X c^{n-1}$$
$$\underset{G_1}{\Longrightarrow} a^{n-1} b^{n-1} Y b c^n \overset{*}{\underset{G_1}{\Longrightarrow}} a^{n-1} Y b^n c^n \ .$$

Ha most alkalmazzuk az $aY \to aa$ szabályt, azt kapjuk, hogy $S_1 \overset{*}{\underset{G_1}{\Longrightarrow}} a^n b^n c^n$, $n \geq 2$-re, de $S_1 \underset{G_1}{\Longrightarrow} abc$ az $S_1 \to abc$ szabály alapján, tehát $a^n b^n c^n \in L(G_1)$ tetszőleges $n \geq 1$-re. Be kell még bizonyítanunk, hogy a szabályok alkalmazásával más szót nem lehet generálni, csak $a^n b^n c^n$ alakút. Ezt könnyű belátni, hisz eredményes levezetés (amely csak terminális betűket tartalmazó szóban végződik) csak a fenti módon lehetséges.
Hasonlóképpen $n \geq 2$-re

$$S_2 \underset{G_2}{\Longrightarrow} aS_2BC \overset{*}{\underset{G_2}{\Longrightarrow}} a^{n-1} S_2 (BC)^{n-1} \underset{G_2}{\Longrightarrow} a^n (BC)^n \overset{*}{\underset{G_2}{\Longrightarrow}} a^n B^n C^n$$
$$\underset{G_2}{\Longrightarrow} a^n b B^{n-1} C^n \overset{*}{\underset{G_2}{\Longrightarrow}} a^n b^n C^n \underset{G_2}{\Longrightarrow} a^n b^n c C^{n-1} \overset{*}{\underset{G_2}{\Longrightarrow}} a^n b^n c^n \ .$$

Itt sorrendben a következő szabályokat alkalmaztuk: $S_2 \to aS_2BC$ ($n-1$-szer), $S_2 \to aBC$, $CB \to BC$ ($n-1$-szer), $aB \to ab$, $bB \to bb$ ($n-1$-szer), $bC \to bc$, $cC \to cc$ ($n-1$-szer). Ugyanakkor $S_2 \underset{G_2}{\Longrightarrow} aBC \underset{G_2}{\Longrightarrow} abC \underset{G_2}{\Longrightarrow} abc$, tehát $S_2 \overset{*}{\underset{G_2}{\Longrightarrow}} a^n b^n c^n$, $n \geq 1$. Itt is könnyű belátni, hogy más szavakat nem lehet generálni a G_2-ben.

A következő két nyelvtan:
$G_3 = (\{S\}, \{a, b\}, \{S \to aSb,\ S \to \varepsilon\}, S)$ és
$G_4 = (\{S\}, \{a, b\}, \{S \to aSb,\ S \to ab\}, S)$
nem ekvivalens, mert $L(G_3) = L(G_4) \cup \{\varepsilon\}$. □

1.2.7. tétel. *Létezik olyan formális nyelv, amelyet nem lehet nyelvtannal megadni.*

Bizonyítás. A bizonyításhoz a nyelvtanokat a $\{0, 1\}$ ábécé feletti szavakként kódoljuk. Egy adott $G = (N, T, P, S)$ nyelvtan esetében legyen $N = \{S_1, S_2, \ldots, S_n\}$, $T = \{a_1, a_2, \ldots, a_m\}$ és $S = S_1$. A kódolás a következő:

S_i kódja $10\underbrace{11\ldots11}_{i\text{-szer}}01$, $\qquad$ a_i kódja $100\underbrace{11\ldots11}_{i\text{-szer}}001$.

A szimbólumok kódját a kódolásban 000 választja el, a nyíl kódja 0000, a szabályokat pedig 00000 jelsorozattal választjuk el.

Nyilván elég csak a szabályokat kódolni. Példaként vegyük a következő nyelvtant:

$G = (\{S\}, \{a, b\}, \{S \to aSb,\ S \to ab\}, S)$.

S kódja 10101, a kódja 1001001, b kódja 10011001. A nyelvtan kódja:

$$\underbrace{10101}0000\underbrace{1001001}000\underbrace{10101}000\underbrace{10011001}00000\underbrace{10101}0000\underbrace{1001001}000\underbrace{10011001}\ .$$

Ebből a kódolásból következik, hogy a T terminális ábécével rendelkező nyelvtanok felsorolhatók[1] a következőképpen: $G_1, G_2, \ldots, G_k, \ldots$, és így ezen nyelvtanok halmazának számossága megszámlálhatóan végtelen.

Tekintsük most a T ábécé feletti összes nyelvek halmazát: $\mathcal{L}_T = \{L \mid L \subseteq T^*\}$, azaz $\mathcal{L}_T = \mathcal{P}(T^*)$. A T^* halmaz megszámlálhatóan végtelen, hisz szavai sorrendbe írhatók. Legyen ez a sorrend $s_0, s_1, s_2, \ldots$, ahol legyen $s_0 = \varepsilon$. Ekkor minden $L \in \mathcal{L}_T$ nyelvnek megfeleltethetünk egy $b_0, b_1, b_2, \ldots$ végtelen bináris sorozatot a következőképpen:

$$b_i = \begin{cases} 1, & \text{ha } s_i \in L \\ 0, & \text{ha } s_i \notin L \end{cases} \qquad i = 0, 1, 2, \ldots \ .$$

Könnyű belátni, hogy az így keletkezett bináris sorozatok halmaza nem megszámlálhatóan végtelen, hisz minden sorozat tekinthető úgy, mint egy 1-nél kisebb pozitív valós szám kettes számrendszerbeli ábrázolása (a tizedesvesszőt

[1] Tegyük fel, hogy a $\{0, 1\}$ ábécén adott egy $<$ lineáris rendezés, mondjuk $0 < 1$. Ezek után a nyelvtanokat kódoló szavak felsorolhatók úgy, hogy a szavakat előbb a hosszúságuk szerint rendezzük, majd azon belül ábécésorrendbe, az ábécé betűi közötti rendezés alapján. De lehet lexikografikus sorrend is, amely azt jelenti, hogy $u < v$ (u előbb van, mint v), ha u valódi kezdőszelete v-nek, vagy létezik az $u = xay$ és $v = xby'$ felbontás, ahol x, y, y' részszavak, a és b betűk, és $a < b$.

az első számjegy elé képzeljük). Fordítva is igaz, hogy minden 1-nél kisebb pozitív szám kettes számrendszerbeli ábrázolása, ha a tizedesvessző utáni részt vesszük, megfelel egy ilyen sorozatnak. Így a végtelen bináris sorozatok halmazának számossága megegyezik a $[0,1]$ intervallum kontinuum számosságával. Következésképpen az $\mathcal{L}_T$ halmaz is kontinuum számosságú. Rendeljük most hozzá minden T terminális ábécével rendelkező nyelvtanhoz az általa generált T feletti nyelvet. Mivel a nyelvtanok számossága megszámlálhatóan végtelen, lesz olyan $\mathcal{L}_T$-beli nyelv, amelyhez nem tartozik nyelvtan, más szóval, amelyet nem lehet nyelvtannal generálni. □

1.3. Chomsky-féle nyelvosztályok

Ha a helyettesítési szabályokra bizonyos megkötéseket teszünk, a következő négy nyelvtantípust különböztethetjük meg.

1.3.1. értelmezés. *A $G = (N, T, P, S)$ nyelvtanra a következő négy típust definiáljuk.*

*A G nyelvtan 0-típusú (**általános** vagy **mondatszerkezetű**), ha semmilyen megkötést nem teszünk a helyettesítési szabályaira.*

*A G nyelvtan 1-típusú **(környezetfüggő),** ha minden szabálya $\alpha A\gamma \to \alpha\beta\gamma$ alakú, ahol $A \in N$, $\alpha, \gamma \in (N \cup T)^*$, $\beta \in (N \cup T)^+$. Ezenkívül megengedhető az $S \to \varepsilon$ szabály is, ha S nem szerepel egyetlen szabály jobb oldalán sem.*

*A G nyelvtan 2-típusú **(környezetfüggetlen),** ha minden szabálya $A \to \beta$ alakú, ahol $A \in N$, $\beta \in (N \cup T)^+$. Ezenkívül megengedhető az $S \to \varepsilon$ szabály is, ha S nem szerepel egyetlen szabály jobb oldalán sem.*

*A G nyelvtan 3-típusú **(reguláris),** ha szabályai $A \to aB$ vagy $A \to a$ alakúak, ahol $a \in T$ és $A, B \in N$. Ezenkívül megengedhető az $S \to \varepsilon$ szabály is, ha S nem szerepel egyetlen szabály jobb oldalán sem.*

Ha G i-típusú nyelvtan, akkor az $L(G)$ nyelvet szintén i-típusúnak (általánosnak, környezetfüggőnek, környezetfüggetlennek, regulárisnak) nevezzük.

Ez a felosztás Noam Chomsky[2] nevéhez fűződik.

Egy L nyelv i-típusú ($i = 0, 1, 2, 3$), ha létezik egy i-típusú G nyelvtan, amelyik az L nyelvet generálja, vagyis $L = L(G)$.

Jelölje $\mathcal{L}_i$ ($i = 0, 1, 2, 3$) az i-típusú nyelvek osztályát. Ekkor bizonyítható, hogy

$$\mathcal{L}_0 \supset \mathcal{L}_1 \supset \mathcal{L}_2 \supset \mathcal{L}_3 \ .$$

[2]Noam Avram Chomsky (1928–) amerikai nyelvész és matematikus, aki bevezette a generatív grammatika fogalmát, amelyet ma a nyelvészetbwen és az informatikában egyaránt használnak.

A nyelvtantípusok fenti értelmezése alapján a tartalmazás ($\supseteq$) nyilvánvaló, de a szigorú tartalmazás ($\supset$) bizonyításra szorul.

1.3.2. példa. Adunk egy-egy példát a környezetfüggő, környezetfüggetlen és reguláris nyelvtanokra.

Környezetfüggő nyelvtan. $G_1 = (N_1, T_1, P_1, S_1)$, ahol $N_1 = \{S_1, A, B, C\}$, $T_1 = \{a, 0, 1\}$. P_1 elemei:

$$\begin{array}{lcl} S_1 & \to & ACA, \\ AC & \to & AACA \mid ABa \mid AaB, \\ B & \to & AB \mid A, \\ A & \to & 0 \mid 1. \end{array}$$

Az $L(G_1)$ nyelv olyan uav szavakból áll, ahol $u, v \in \{0,1\}^*$ és $|u| \neq |v|$.

Környezetfüggetlen nyelvtan. $G_2 = (N_2, T_2, P_2, S)$, ahol $N_2 = \{S, A, B\}$, $T_2 = \{+, *, (,), a\}$. P_2 elemei:

$$\begin{array}{lcl} S & \to & S + A \mid A, \\ A & \to & A * B \mid B, \\ B & \to & (S) \mid a. \end{array}$$

Az $L(G_2)$ olyan algebrai kifejezésekből áll, amelyek az a betűből a $+$ és $*$ műveleti jelekkel és a zárójelekkel „szabályosan" felépíthetők.

Reguláris nyelvtan. $G_3 = (N_3, T_3, P_3, S_3)$, ahol $N_3 = \{S_3, A, B\}$, $T_3 = \{a, b\}$. P_3 elemei:

$$\begin{array}{lcl} S_3 & \to & aA \\ A & \to & aB \mid a \\ B & \to & aB \mid bB \mid a \mid b. \end{array}$$

Az $L(G_3)$ nyelv olyan, a és b betűkből képezhető szavakból áll, amelyek legalább két a-val kezdődnek. □

Könnyű bebizonyítani, hogy minden véges nyelv reguláris. A nyelvtan szabályait úgy adjuk meg, hogy generálják az összes szót. Például, ha $u = a_1 a_2 \dots a_n$ eleme a nyelvnek, akkor bevezetjük a következő szabályokat: $S \to a_1 A_1$, $A_1 \to a_2 A_2$, $\dots A_{n-2} \to a_{n-1} A_{n-1}$, $A_{n-1} \to a_n$, ahol S a nyelvtan kezdőszimbóluma, $A_1, \dots, A_{n-1}$ pedig különböző nemterminálisok. Ezt a véges nyelv minden szavára megtesszük, oly módon, hogy különböző szavakhoz, az S kivételével, különböző nemterminálisokat használunk. Ha az üres szó is eleme a nyelvnek, akkor azt természetesen az $S \to \varepsilon$ szabállyal generáljuk.

Az üres halmaz szintén reguláris nyelv, hisz például a $G = (\{S\}, \{a\}, \{S \to aS\}, S)$ reguláris nyelvtan az üres halmazt generálja.

1.3.1. Átnevezések kiküszöbölése

Egy $A \to B$ alakú szabályt, ahol $A, B \in N$, ***átnevezésnek*** nevezzük. Az átnevezéseket ki lehet küszöbölni a G nyelvtanból úgy, hogy az újonnan kapott

nyelvtan ugyanolyan típusú legyen, mint G, és ugyanazt a nyelvet ismerje fel, mint G.

Legyen $G = (N, T, P, S)$ átnevezéseket tartalmazó nyelvtan. Definiáljuk a vele ekvivalens, de átnevezéseket nem tartalmazó $G' = (N, T, P', S)$ nyelvtant. A következő algoritmus megkonstruálja az új nyelvtan szabályait.

ÁTNEVEZÉS-KIZÁRÁS(G)
1 valahányszor $A \to B$ és $B \to C$ átnevezések P-ben vannak, mindannyiszor vegyük fel P-be az $A \to C$ átnevezést is mindaddig, amíg P bővíthető,
2 valahányszor $A \to B$ átnevezés és a $B \to \alpha$ ($\alpha \notin N$) szabály P-ben van, mindannyiszor vegyük fel P-be az $A \to \alpha$ szabályt is,
3 legyen P' azon P-beli szabályok halmaza, amelyek nem átnevezések.
4 **return** G'

Könnyen belátható, hogy G és G' ekvivalensek. Továbbá, ha G $i \in \{0, 1, 2, 3\}$ típusú, akkor G' is i típusú lesz.

1.3.3. példa. Alkalmazzuk az előbbi algoritmust a $G = \big(\{S, A, B, C\}, \{a, b\}, P, S\big)$ nyelvtanra, ahol a P elemei:

$S \to A$, $A \to B$, $B \to C$, $C \to B$, $D \to C$,
$S \to B$, $A \to D$, $C \to Aa$,
$A \to aB$,
$A \to b$.

Az algoritmus első lépése alapján a következő új átnevezések kerülnek be a szabályok közé:

$S \to D$ ($S \to A$ és $A \to D$ miatt),
$S \to C$ ($S \to B$ és $B \to C$ miatt),
$A \to C$ ($A \to B$ és $B \to C$ miatt),
$B \to B$ ($B \to C$ és $C \to B$ miatt),
$C \to C$ ($C \to B$ és $B \to C$ miatt),
$D \to B$ ($D \to C$ és $C \to B$ miatt).

Az algoritmus második lépése alkalmazásakor csak azok az átnevezések jöhetnek számításba, amelyeknek a jobb oldala vagy A vagy C, mivel csak az $A \to aB$, $A \to b$ és $C \to Aa$ szabályok alkalmazhatók (a többi szabály mind átnevezés). A következő új szabályok jelennek meg:

$S \to aB$ ($S \to A$ és $A \to aB$ miatt),
$S \to b$ ($S \to A$ és $A \to b$ miatt),
$S \to Aa$ ($S \to C$ és $C \to Aa$ miatt),
$A \to Aa$ ($A \to C$ és $C \to Aa$ miatt),
$B \to Aa$ ($B \to C$ és $C \to Aa$ miatt).

Az új $G' = \big(\{S, A, B, C\}, \{a, b\}, P', S\big)$ nyelvtan szabályai:

$S \to b$, $A \to b$, $B \to Aa$, $C \to Aa$,
$S \to aB$, $A \to aB$,
$S \to Aa$ $A \to Aa$, □

1.3.2. Normálalakú nyelvtanok

Ha egy nyelvtan szabályainak bal oldalán terminális betűk nem szerepelnek, akkor a nyelvtant *normálalakúnak* mondjuk.

Szükségünk lesz a következő fogalmakra. Adott Σ_1 és Σ_2 ábécékre *homomorfizmusnak* nevezzük a $h : \Sigma_1^* \to \Sigma_2^*$ függvényt, ha $h(u_1u_2) = h(u_1)h(u_2)$, $\forall u_1, u_2 \in \Sigma_1^*$. Könnyű belátni, hogy tetszőleges $u = a_1a_2 \dots a_n \in \Sigma_1^*$ esetében a $h(u)$ értéket egyértelműen meghatározza h-nak a Σ_1-re való megszorítása, ugyanis $h(u) = h(a_1)h(a_2) \dots h(a_n)$.

Ha a h függvény még bijektív is, akkor *izomorfizmusról* beszélünk.

1.3.4. tétel. *Tetszőleges nyelvtanhoz megkonstruálhatunk egy vele ekvivalens, azonos típusú nyelvtant, amely normálalakú.*

Bizonyítás.

A 2- és 3-típusú nyelvtanok szabályai a bal oldalon csak egy-egy nemterminális szimbólumot tartalmaznak, tehát eleve normálalakúak.

A bizonyítást csupán a 0- és 1-típusú nyelvtanokra kell elvégezni. Legyen az eredeti nyelvtan $G = (N, T, P, S)$ és definiáljuk a $G' = (N', T, P', S)$ normálalakú nyelvtant a következőképpen.

Legyenek $a_1, a_2, \dots, a_k$ azok a terminális szimbólumok, amelyek szerepelnek a szabályok bal oldalán. Ekkor vezessük be az $A_1, A_2, \dots, A_k$ új nemterminálisokat. Használjuk a következő jelöléseket: $T_1 = \{a_1, a_2, \dots, a_k\}$, $T_2 = T \setminus T_1$, $N_1 = \{A_1, A_2, \dots, A_k\}$ és $N' = N \cup N_1$.

Használjuk a $h : N \cup T \longrightarrow N' \cup T_2$ izomorfizmust, ahol:

$h(a_i) = A_i$, ha $a_i \in T_1$,
$h(X) = X$, ha $X \in N \cup T_2$

Most értelmezzük a P' szabályhalmazt:

$$P' = \Big\{ h(\alpha) \to h(\beta) \;\Big|\; (\alpha \to \beta) \in P \Big\} \cup \Big\{ A_i \longrightarrow a_i \;\Big|\; i = 1, 2, \dots, k \Big\}$$

Ebben az esetben $\alpha \overset{*}{\underset{G}{\Longrightarrow}} \beta$ akkor és csakis akkor, ha $h(\alpha) \overset{*}{\underset{G'}{\Longrightarrow}} h(\beta)$.

Innen pedig azonnal következik a tételünk, hisz $S \overset{*}{\underset{G}{\Longrightarrow}} u \;\Leftrightarrow\; S = h(S) \overset{*}{\underset{G'}{\Longrightarrow}} h(u) = u$. □

1.3.5. példa. Adott a $G = (\{S, D, E\}, \{a, b, c, d, e\}, P, S)$, ahol P szabályai:

$S \rightarrow aebc \mid aDbc$
$Db \rightarrow bD$
$Dc \rightarrow Ebccd$
$bE \rightarrow Eb$
$aE \rightarrow aaD \mid aae$

Bal oldali szabályokban az a, b, c terminálisok szerepelnek, ezért felvesszük a nemterminálisok közé az A, B, C új nemterminálisokat, és P'-be az $A \rightarrow a$, $B \rightarrow b$ és $C \rightarrow c$ szabályokat.

Az a, b, c terminálisokat minden szabályban helyettesítjük rendre az A, B, C nemterminálisokkal. Ekkor P' szabályai a következők:

$S \rightarrow AeBC \mid ADBC$
$DB \rightarrow BD$
$DC \rightarrow EBCCd$
$BE \rightarrow EB$
$AE \rightarrow AAD \mid AAe$
$A \rightarrow a$
$B \rightarrow b$
$C \rightarrow c$

Vizsgáljuk meg milyen szavakat generál ez a nyelvtan! $S \Longrightarrow AeBC \overset{*}{\Longrightarrow} aebc$ miatt $aebc \in L(G')$. $S \Longrightarrow ADBC \Longrightarrow ABDC \Longrightarrow ABEBCCd \Longrightarrow AEBBCCd \Longrightarrow AAeBBCCd \overset{*}{\Longrightarrow} aaebbccd$, tehát $aaebbccd \in L(G')$.

Feltételezzük, hogy $S \overset{*}{\Longrightarrow} A^{n-1}EB^nC(Cd)^{n-1}$, $n \geq 2$. Ezt matematikai indukcióval bizonyíthatjuk. Már láttuk, hogy feltevésünk igaz ha $n = 2$. Folytassuk az előbbi levezetést: $S \overset{*}{\Longrightarrow} A^{n-1}EB^nC(Cd)^{n-1} \Longrightarrow A^{n-2}AADB^nC(Cd)^{n-1} \overset{*}{\Longrightarrow} A^nB^nDC(Cd)^{n-1} \Longrightarrow A^nB^nEBCCd(Cd)^{n-1} \overset{*}{\Longrightarrow} A^nEB^{n+1}CCd(Cd)^{n-1} = A^nEB^{n+1}C(Cd)^n$, amit éppen bizonyítani kellett.

De $S \overset{*}{\Longrightarrow} A^{n-1}EB^nC(Cd)^{n-1} \Longrightarrow A^{n-2}AAeB^nC(Cd)^{n-1} \overset{*}{\Longrightarrow} a^neb^nc(cd)^{n-1}$. Tehát $a^neb^nc(cd)^{n-1} \in L(G')$, $n \geq 1$. Ezeket a szavakat G-ben is hasonlóan le lehet vezetni. □

1.4. Kiterjesztett nyelvtanok

Ebben a részben az 1-típusú, 2-típusú és a 3-típusú kiterjesztett nyelvtanokat mutatjuk be.

1-típusú kiterjesztett nyelvtan. Minden szabály $\alpha \rightarrow \beta$ alakú, ahol $|\alpha| \leq |\beta|$, kivéve esetleg az $S \rightarrow \varepsilon$ szabályt.

2-típusú kiterjesztett nyelvtan. Minden szabály $A \rightarrow \beta$ alakú, ahol $A \in N, \beta \in (N \cup T)^*$.

3-típusú kiterjesztett nyelvtan. Minden szabály $A \rightarrow uB$ vagy $A \rightarrow u$ alakú, ahol $A, B \in N, u \in T^*$.

1.4.1. tétel. *Tetszőleges kiterjesztett nyelvtanhoz megadható egy vele ekvivalens, ugyanolyan típusú nyelvtan.*

Bizonyítás. Jelöljük G_{ki}-vel a kiterjesztett nyelvtant és G-vel azt a nyelvtant, amelyet minden típusra külön értelmezünk, és amelyről meg fogjuk mutatni, hogy ekvivalens G_{ki}-vel.

1-típus. A G nyelvtan szabályait úgy kapjuk, hogy a G_{ki} nyelvtan $\alpha \to \beta$ szabályait, ahol $|\alpha| \leq |\beta|$, átírjuk a G esetében megengedett $\gamma_1\delta\gamma_2 \to \gamma_1\gamma\gamma_2$ alakú szabályokra a következőképpen.

Legyen $X_1X_2\ldots X_m \to Y_1Y_2\ldots Y_n$ $(m \leq n)$ a G_{ki} nyelvtan egy szabálya, amely nem megfelelő alakú. Vegyük fel G szabályhalmazába a következő szabályokat, ahol $A_1,\ A_2,\ \ldots,\ A_m$ új változók:

$$\begin{array}{lcl}
X_1X_2\ldots X_m & \to & A_1X_2X_3\ldots X_m \\
A_1X_2\ldots X_m & \to & A_1A_2X_3\ldots X_m \\
 & \ldots & \\
A_1A_2\ldots A_{m-1}X_m & \to & A_1A_2\ldots A_{m-1}A_m \\
A_1A_2\ldots A_{m-1}A_m & \to & Y_1A_2\ldots A_{m-1}A_m \\
Y_1A_2\ldots A_{m-1}A_m & \to & Y_1Y_2\ldots A_{m-1}A_m \\
 & \ldots & \\
Y_1Y_2\ldots Y_{m-2}A_{m-1}A_m & \to & Y_1Y_2\ldots Y_{m-2}Y_{m-1}A_m \\
Y_1Y_2\ldots Y_{m-1}A_m & \to & Y_1Y_2\ldots Y_{m-1}Y_mY_{m+1}\ldots Y_n.
\end{array}$$

Továbbá, G_{ki} minden megengedett, vagyis $\gamma_1\delta\gamma_2 \to \gamma_1\gamma\gamma_2$ alakú szabályát vegyük át változtatás nélkül G szabályhalmazába.

Ezek után a $L(G_{ki}) \subseteq L(G)$ tartalmazás abból következik, hogy a G_{ki} minden szabályának az alkalmazását egy, a belőle képzett G-beli szabályok alkalmazásával tudjuk szimulálni. Továbbá, mivel a G szabályai csak a felírt sorrendben alkalmazhatók, nem kapunk újabb szavakat, ezért $L(G) \subseteq L(G_{ki})$ is teljesül.

2-típus. Legyen $G_{ki} = (N, T, P, S)$. Ki kell küszöbölnünk az $A \to \varepsilon$ alakú szabályokat úgy, hogy csak $S \to \varepsilon$ maradhat, ha S nem szerepel szabály jobb oldalán. Ehhez felépítjük a következő halmazokat:

$U_0 = \{A \in N \mid (A \to \varepsilon) \in P\}$

$U_i = U_{i-1} \cup \{A \in N \mid (A \to w) \in P,\ w \in U_{i-1}^+\}$.

Mivel minden $i \geq 1$ esetében $U_{i-1} \subseteq U_i$ és $U_i \subseteq N$, továbbá N véges halmaz, léteznie kell egy olyan k-nak, amelyre $U_{k-1} = U_k$, és ezt a halmazt nevezzük el U-nak. Könnyű belátni, hogy egy A nemterminális akkor és csakis akkor eleme U-nak, ha $A \stackrel{*}{\Longrightarrow} \varepsilon$. (Mellesleg, $\varepsilon \in L(G_{ki})$, akkor és csakis akkor ha $S \in U$.)

G szabályait a következőképpen kapjuk G_{ki} szabályaiból. G_{ki} minden olyan $A \to \alpha$ szabálya esetében melyre $\alpha \neq \varepsilon$, vegyük fel a G szabályai közé ezt a szabályt és mellette azokat is, amelyeket úgy képezünk, hogy α-ból elhagyunk egy vagy több U-beli változót, de csak akkor, ha ezáltal a jobb oldal nem lesz ε. Nem nehéz belátni, hogy az így kapott G nyelvtan ugyanazt a nyelvet

generálja, mint G_{ki}, kivéve az ε szót, amelyet nem tud generálni. Ezért, ha $\varepsilon \notin L(G_{ki})$, akkor a bizonyítást befejeztük. Ha viszont $\varepsilon \in L(G_{ki})$, akkor két esetet különböztetünk meg. Ha az S kezdőszimbólum nem szerepel egyetlen szabály jobb oldalán sem, akkor az $S \to \varepsilon$ szabály bevezetésével a G nyelvtan már az üres szót is generálni fogja. Ha viszont S szerepel valamelyik szabály jobb oldalán, akkor egy új kezdőszimbólum (S') bevezetésével ε is generálható lesz, ha bevesszük a G szabályai közé az $S' \to S$ és $S' \to \varepsilon$ szabályokat.

3-típus. Először alkalmazzuk G_{ki}-re a 2-típus esetében használt eljárást az $A \to \varepsilon$ alakú szabályok kiküszöbölésére. A kapott nyelvtanból kiküszöböljük az átnevezéseket az ÁTNEVEZÉS-KIZÁRÁS algoritmus segítségével (11. oldal).

Az így kapott nyelvtan minden $A \to a_1 a_2 \dots a_n B$ szabálya esetén, ahol $B \in N \cup \{\varepsilon\}$, vegyük fel G szabályai közé a következő szabályokat:

$A \to a_1 A_1, \;\; A_1 \to a_2 A_2, \;\; A_{n-1} \to a_n B,$

ahol $A_1, A_2, \dots, A_{n-1}$ új változók. Könnyen igazolható, hogy az így megkonstruált G nyelvtan ekvivalens G_{ki}-vel. □

1.4.2. példa. Adva van a következő 1-típusú kiterjesztett nyelvtan: $G_{ki} = (N, T, P, S)$, ahol $N = \{S, B, C\}$, $T = \{a, b, c\}$ és P a következő szabályokból áll:

$$\begin{array}{lcllcl} S & \to & aSBC \mid aBC & CB & \to & BC \\ aB & \to & ab & bB & \to & bb \\ bC & \to & bc & cC & \to & cc\,. \end{array}$$

Az egyetlen szabály, amely nem környezetfüggő, az a $CB \to BC$. Ehelyett bevezetjük a következőket, a bizonyításban adott módszer alapján, ahol $A_1 = A$, $A_2 = D$:

$$\begin{array}{lcl} CB & \to & AB \\ AB & \to & AD \\ AD & \to & BD \\ BD & \to & BC \end{array}$$

Így az új nyelvtan $G = (\{S, A, B, C, D\}, \{a, b, c\}, P', S)$, ahol P' elemei:

$$\begin{array}{lcllcl} S & \to & aSBC \mid aBC & & & \\ CB & \to & AB & aB & \to & ab \\ AB & \to & AD & bB & \to & bb \\ AD & \to & BD & bC & \to & bc \\ BD & \to & BC & cC & \to & cc. \end{array}$$

már környezetfüggő. Igazolni lehet, hogy $L(G_{ki}) = L(G) = \{a^n b^n c^n \mid n \geq 1\}$. □

1.4.3. példa. Legyen $G_{ki} = (\{S, B, C\}, \{a, b, c\}, P, S)$ 2-típusú kiterjesztett nyelvtan, ahol P elemei:

$$\begin{array}{lcl} S & \to & aSc \mid B \\ B & \to & bB \mid C \\ C & \to & Cc \mid \varepsilon. \end{array}$$

Ekkor $U_0 = \{C\}$, $U_1 = \{B, C\}$, $U_3 = \{S, B, C\} = U$. Az új nyelvtan szabályai:

$S \rightarrow aSc \mid ac \mid B$
$B \rightarrow bB \mid b \mid C$
$C \rightarrow Cc \mid c.$

Mivel az eredeti nyelvtan generálja az üres szót is, és S szerepel szabály jobb oldalán, új kezdőszimbólumot kell bevezetnünk és még két szabályt: $S' \rightarrow S, S' \rightarrow \varepsilon$. Tehát az eredetivel ekvivalens környezetfüggetlen nyelvtan:
$G = (\{S', S, B, C\}, \{a, b, c\}, P', S')$, és a szabályok:

$S' \rightarrow S \mid \varepsilon$
$S \rightarrow aSc \mid ac \mid B$
$B \rightarrow bB \mid b \mid C$
$C \rightarrow Cc \mid c.$

Mindkét nyelvtan az $\{a^m b^n c^p \mid p \geq m \geq 0, n \geq 0\}$ nyelvet generálja. □

1.4.4. példa. Legyen $G_{ki} = (\{S, A, B\}, \{a, b\}, P, S)$ a vizsgálandó 3-típusú kiterjesztett nyelvtan, ahol P:

$S \rightarrow abA$
$A \rightarrow bB$
$B \rightarrow S \mid \varepsilon.$

Először küszöböljük ki a $B \rightarrow \varepsilon$ szabályt. Mivel $U_0 = U = \{B\}$, a szabályok a következők lesznek:

$S \rightarrow abA$
$A \rightarrow bB \mid b$
$B \rightarrow S.$

Ez utóbbi szabályt (amely átnevezés) ki lehet küszöbölni, bevezetve helyette a $B \rightarrow abA$ szabályt. Hátra van még az $S \rightarrow abA$ és $B \rightarrow abA$ szabályok jobb oldalának a feldarabolása. Mivel mindkét szabály jobb oldala ugyanaz, elég egy új változót bevezetnünk, és az $S \rightarrow abA$ helyett az $S \rightarrow aC$ és $C \rightarrow bA$ szabályokat használni. A $B \rightarrow abA$ helyett ekkor elég a $B \rightarrow aC$ szabályt venni. Az új nyelvtan: $G = (\{S, A, B, C\}, \{a, b\}, P', S)$, ahol P':

$S \rightarrow aC$
$A \rightarrow bB \mid b$
$B \rightarrow aC$
$C \rightarrow bA.$

Be lehet bizonyítani, hogy $L(G_{ki}) = L(G) = \{(abb)^n \mid n \geq 1\}$. □

1.5. A Chomsky-féle nyelvosztályok zártsági tulajdonságai

Bebizonyítjuk a következő tételt, amely szerint a Chomsky-nyelvosztályok mindegyike zárt a reguláris műveletekre nézve, azaz két i-típusú nyelv egyesítése és szorzata is i-típusú, i-típusú nyelv iteráltja is i-típusú ($i = 0, 1, 2, 3$).

1.5.1. tétel. *Az $\mathcal{L}_i$ ($i = 0, 1, 2, 3$) nyelvek osztálya zárt a reguláris műveletekre nézve.*

Bizonyítás. Kiterjesztett nyelvtanok segítségével végezzük a bizonyítást. Legyenek $G_1 = (N_1, T_1, P_1, S_1)$ és $G_2 = (N_2, T_2, P_2, S_2)$ i-típusú kiterjesztett nyelvtanok. Feltételezzük, hogy $N_1 \cap N_2 = \emptyset$.

Egyesítés. Legyen $G_\cup = (N_1 \cup N_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}, S)$.

Könnyű igazolni, hogy $L(G_\cup) = L(G_1) \cup L(G_2)$. Ha $i = 0, 2, 3$, akkor abból, hogy G_1 és G_2 i-típusú, következik, hogy $G_\cup$ is az lesz. Ha $i = 1$, akkor, ha valamelyik nyelvtan generálja az üres szót, akkor a $G_\cup$ szabályaiból kivesszük a megfelelő (esetleg mindkettő) $S_k \to \varepsilon$ ($k = 1, 2$) szabályt és helyettesítjük $S \to \varepsilon$ szabállyal.

Szorzat. Legyen $G_\times = (N_1 \cup N_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup \{S \to S_1 S_2\}, S)$.

Könnyű igazolni, hogy $L(G_\times) = L(G_1)L(G_2)$. Az $i = 0, 2$ típusoknál $G_\times$ is ugyanolyan típusú lesz. Az $i = 1$ típusnál, ha van P_1-ben $S_1 \to \varepsilon$ szabály, de P_2-ben nincs $S_2 \to \varepsilon$ szabály, akkor a $S_1 \to \varepsilon$ szabályt helyettesítjük az $S \to S_2$ szabállyal. Hasonlóképpen járunk el a szimmetrikus esetnél. Ha van P_1-ben $S_1 \to \varepsilon$ szabály és P_2-ben $S_2 \to \varepsilon$ szabály, akkor ezeket helyettesítjük az $S \to \varepsilon$ szabállyal.

Másképp kell megadni a szabályokat a reguláris nyelvtanok ($i = 3$) esetében, mert $S \to S_1 S_2$ nem reguláris szabály. Helyette a következő nyelvtant használjuk:

$G_\times = (N_1 \cup N_2, T_1 \cup T_2, P_1' \cup P_2, S_1)$, ahol P_1' annyiban különbözik P_1-től, hogy az $A \to u, u \in T^*$ szabályok helyett $A \to uS_2$ kerül be P_1'-be .

Iteráció. Legyen $G_* = (N_1 \cup \{S\}, T_1, P, S)$.

2-típusú nyelvtanoknál legyen $P = P_1 \cup \{S \to S_1 S,\ S \to \varepsilon\}$. Ekkor G_* is 2-típusú lesz.

A 3-típusnál, a szorzathoz hasonlóan átalakítjuk a szabályokat, azaz $P = P_1' \cup \{S \to S_1,\ S \to \varepsilon\}$, ahol P_1' abban különbözik P_1-től, hogy minden $A \to u$ ($u \in T^*$) alakú szabály helyett $A \to uS$ alakút veszünk, a többit változatlanul hagyjuk. Ekkor G_* is 3-típusú lesz.

$i = 0, 1$-re nem jók a 2-típusnál megadott szabályok, mert az $S \to S_1 S$ alkalmazása során megtörténhet, hogy a következő levezetésekhez jutunk: $S \overset{*}{\Longrightarrow} S_1 S_1 \alpha$, $S_1 \overset{*}{\Longrightarrow} \alpha_1 \beta_1$, $S_1 \overset{*}{\Longrightarrow} \alpha_2 \beta_2$, ahol $\beta_1 \alpha_2$ egy helyettesítési szabály bal oldala. Ekkor az $S \overset{*}{\Longrightarrow} \alpha_1 \beta_1 \alpha_2 \beta_2 \alpha$ levezetésben helyettesítve $\beta_1 \alpha_2$-t a neki megfelelő szabály jobb oldalával, olyan szót is generálhatunk, amelyik nincsen benne az iterált nyelvben. Hogy ezt elkerüljük, először feltételezzük, hogy a nyelvtan normálalakú, azaz szabályok bal oldalán nincsenek terminális jelek (*lásd* 12. oldal), majd bevezetünk egy új S' nemterminálist, tehát a nemterminálisok halmaza most $N_1 \cup \{S, S'\}$, a szabályok pedig a következők

lesznek: $P = P_1 \cup \{S \to \varepsilon,\ S \to S_1 S'\} \cup \{aS' \to aS \mid a \in T_1\}$. Így most már elkerülhetjük, hogy esetleg olyan szabályt is alkalmazzunk a levezetésben, amelynek bal oldala átnyúlik a szavak határán az iteráció miatt. Most már az előbbi levezetéseket csak úgy lehet alkalmazni, hogy az $S \Longrightarrow S_1 S'$ helyettesítéssel kezdjük, majd eljutunk az $S \overset{*}{\Longrightarrow} \alpha_1\beta_1 S'$ levezetéshez. Ezt csak akkor tudjuk S' helyettesítésével folytatni, ha β_1 utolsó betűje terminális, és miután alkalmaztuk valamelyik $aS' \to aS$ helyettesítési szabályt.

Igazolható, hogy mindegyik típus esetében $L(G_*) = L(G_1)^*$. □

Gyakorlatok

1-1. Adjunk meg egy nyelvtant, amely az $L = \left\{uu^{-1} \mid u \in \{a,b\}^*\right\}$ nyelvet generálja, és határozzuk meg a típusát.

1-2. Adott a $G = (N, T, P, S)$ kiterjesztett környezetfüggetlen nyelvtan, ahol
$N = \{S, A, C, D\},\ T = \{a, b, c, d, e\}$,
$P = \{S \to abCADe,\ C \to cC,\ C \to \varepsilon,\ D \to dD,\ D \to \varepsilon,\ A \to \varepsilon,$
$A \to dDcCA\}$.
Adjunk meg egy vele ekvivalens környezetfüggetlen nyelvtant.

1-3. Adjunk meg egy-egy nyelvtant a következő nyelvek generálására.
$L_1 = \{a^n b^m c^p \mid n \geq 1, m \geq 1, p \geq 1\}$,
$L_2 = \{a^{2n} \mid n \geq 1\}$,
$L_3 = \{a^n b^m \mid n \geq 0, m \geq 0\ \}$,
$L_4 = \{a^n b^m \mid n \geq m \geq 1\}$.

1-4. Adjunk meg egy nyelvtant az $L = \left\{u \in \{0,1\}^* \mid n_0(u) = n_1(u)\right\}$ nyelv generálására, ahol $n_0(u)$ az u szóban szereplő 0-k, $n_1(u)$ pedig az 1-ek számát jelenti.

1-5. Adjunk meg egy nyelvtant, amely a természetes számokat generálja.

1-6. Adott a $G = (N, T, P, S)$ kiterjesztett nyelvtan, ahol $N = \{S, A, B, C\}$, $T = \{a\}$, P pedig a következő helyettesítési szabályokat tartalmazza:

$$S \to BAB,\ BA \to BC,\ CA \to AAC,\ CB \to AAB,\ A \to a,\ B \to \varepsilon\ .$$

Milyen típusú ez a nyelvtan? Adjunk meg egy vele ekvivalens, ugyanolyan típusú nem kiterjesztett nyelvtant. Határozzuk meg a G nyelvtan által generált nyelvet.

1-7. Adjuk meg a következő nyelvtan által generált nyelvet:
$G = (N, T, P, S)$, ahol $N = \{S, A, B\}$, $T = \{a, *, +, (,)\}$ és P elemei:

$$\begin{aligned} S &\to S + A \mid A \\ A &\to A * B \mid B \\ B &\to (S) \mid a. \end{aligned}$$

1-8. Mutassuk meg, hogy egy r-betűs ábécé feletti véges nyelv, amelyben minden szó legfeljebb n hosszúságú, legfeljebb $\frac{r^{n+1}-1}{r-1}$ szót tartalmazhat.

1-9. Adjunk meg egy-egy reguláris, környezetfüggetlen és környezetfüggő nyelvtant az üres nyelv generálására.

2. FEJEZET

Véges automaták és reguláris nyelvek

2.1. Véges automaták értelmezése

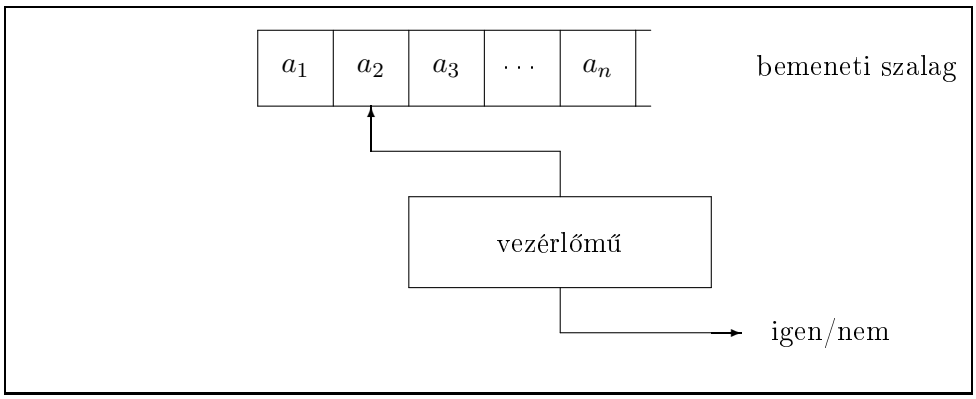
A véges automaták olyan számítási modellek, amelyek rendelkeznek egy bemeneti szalaggal és több állapottal (2.1. ábra). Az állapotok között vannak kezdőállapotnak, illetve végállapotnak nevezett állapotok. A véges automata egy szót kap bemenetként, amely a bemeneti szalagra van írva, és az olvasófeje a bemeneti szó első betűjére mutat. Az automata kezdőállapotból indul, a szalagról sorra olvassa a betűket, miközben állapotot válthat. Érzékeli, ha végigolvasta a szót, és amennyiben az utolsó állapot végállapot, akkor azt mondjuk, hogy felismerte az adott szót. Egy ilyen automata által felismert szavak halmazát az automata által felismert nyelvnek nevezzük.

2.1.1. értelmezés. ***Nemdeterminisztikus véges automatának*** *nevezzük az* $\mathrm{A} = (Q, \Sigma, E, I, F)$ *rendezett ötöst, ahol*

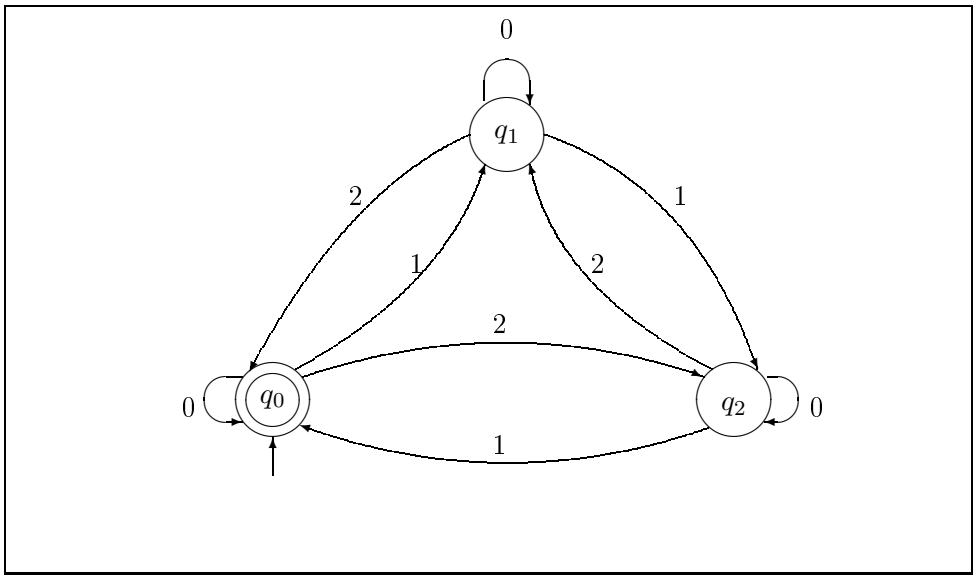
- Q *egy véges, nem üres halmaz, az* ***állapotok*** *halmaza,*
- Σ *a* ***bemeneti ábécé,***
- E *az* ***átmenetek*** *(vagy élek) halmaza, ahol* $E \subseteq Q \times \Sigma \times Q$*,*
- $I \subseteq Q$ *a* ***kezdőállapotok*** *halmaza,*
- $F \subseteq Q$ *a* ***végállapotok*** *halmaza.*

A nemdeterminisztikus véges automata tulajdonképpen egy olyan irányított, címkézett gráf, amelynek csúcsai az állapotok, és egy p csúcsából akkor vezet egy a betűvel megcímkézett él a q csúcsba, ha $(p, a, q) \in E$. Az állapotokat jelentő csúcsok között bizonyosak kezdő- és bizonyosak végállapotok. A kezdőállapotokat egy-egy befutó nyíl jelzi, míg a végállapotokat két-két koncentrikus kör. Ha két csúcs között több, ugyanolyan irányú él van, akkor ezeket helyettesítjük egyetlen éllel, amelyre a betűket vesszővel elválasztva írjuk. Ezt a gráfot átmenetgráfnak fogjuk hívni.

2.1.2. példa. Legyen $\mathrm{A} = (Q, \Sigma, E, I, F)$, ahol $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{0, 1, 2\}$,
$E = \big\{(q_0, 0, q_0), (q_0, 1, q_1), (q_0, 2, q_2),$
$(q_1, 0, q_1), (q_1, 1, q_2), (q_1, 2, q_0),$
$(q_2, 0, q_2), (q_2, 1, q_0), (q_2, 2, q_1)\big\}$, $I = \{q_0\}$, $F = \{q_0\}$. (2.2. ábra) □



2.1. ábra. Véges automata.



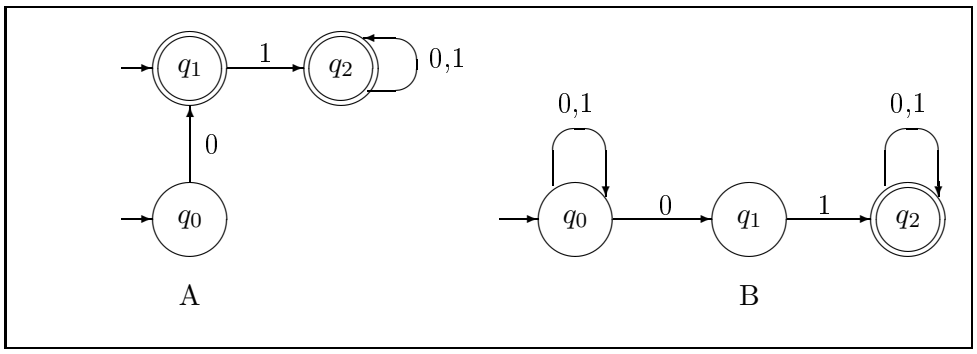
2.2. ábra. A 2.1.2. példában szereplő véges automata.

Egy (p, a, q) élnek p a kezdőpontja, q a végpontja, a pedig a címkéje. Értelmezzük a gráfoknál használatos ***séta*** fogalmát. A

$$(q_0, a_1, q_1), (q_1, a_2, q_2), \ldots, (q_{n-2}, a_{n-1}, q_{n-1}), (q_{n-1}, a_n, q_n)$$

élsorozat a nemdeterminisztikus véges automata egy sétája, amelynek címkéje az $a_1 a_2 \ldots a_n$ szó. Ha $n = 0$, akkor $q_0 = q_n$ és $a_1 a_2 \ldots a_n = \varepsilon$. Az ilyen sétát ***üres sétának*** nevezzük. A séta jelölése

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{n-1}} q_{n-1} \xrightarrow{a_n} q_n \ ,$$



2.3. ábra. Nemdeterminisztikus véges automaták.

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

A

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

B

2.4. ábra. A 2.3. ábrán látható két véges automata átmenettáblázata.

vagy ha $w = a_1 a_2 \dots a_n$, akkor röviden: $q_0 \xrightarrow{w} q_n$. Itt q_0 a séta kezdőpontja, q_n pedig a végpontja. A sétában szereplő állapotok nem feltétlenül különbözők. Egy séta ***produktív***, ha kezdőpontja kezdőállapot, végpontja pedig végállapot. Azt mondjuk, hogy a nemdeterminisztikus véges automata ***felismer*** egy szót, ha az a szó egy produktív séta címkéje. Az ε üres szót a nemdeterminisztikus véges automata akkor ismeri fel, ha van produktív üres séta, amely egyenértékű azzal, hogy van olyan kezdőállapot, amely egyben végállapot is.

Egy nemdeterminisztikus véges automata által felismert szavak halmazát a nemdeterminisztikus véges automata által felismert nyelvnek mondjuk. Az A ***nemdeterminisztikus véges automata által felismert nyelv***

$$L(\mathrm{A}) = \left\{ w \in \Sigma^* \mid \exists p \in I,\ \exists q \in F,\ \exists p \xrightarrow{w} q \right\} .$$

Az A_1 és A_2 véges automaták ***ekvivalensek,*** ha $L(\mathrm{A}_1) = L(\mathrm{A}_2)$.

Értelemzzük a következő átmenetfüggvényt:

$$\delta : Q \times \Sigma \to P(Q), \quad \delta(p, a) = \{ q \in Q \mid (p, a, q) \in E \} ,$$

amely sokszor hasznos lehet.

Ez a függvény egy p állapotnak és egy a betűnek megfelelteti azt az állapothalmazt, amelynek állapotaiba átmehet a véges automata, ha a p állapotban

van és az olvasófej az a betűre mutat. Jelöljük $|H|$-val a H halmaz elemeinek a számát.[1] Egy nemdeterminisztikus véges automatáról azt mondjuk, hogy ***determinisztikus,*** ha

$$|I| = 1 \text{ és } |\delta(q,a)| \leq 1,\ \forall q \in Q,\ \forall a \in \Sigma\ .$$

A 2.2. ábrán egy determinisztikus véges automata látható.

A $|\delta(q,a)| \leq 1$ feltételt helyettesíthetjük a következővel:

$$(p,a,q) \in E,\ (p,a,r) \in E \implies q = r\ , \forall p,q,r \in Q, \forall a \in \Sigma\ .$$

Ha a determinisztikus véges automata olyan, hogy $|\delta(q,a)| = 1$ minden $q \in Q$ állapotra és minden $a \in \Sigma$ betűre, akkor azt mondjuk, hogy ***teljes, determinisztikus véges automata***.

Minden determinisztikus véges automata teljessé tehető egy új állapot bevezetésével, amelyet csapdaállapotnak szokás nevezni. Legyen $\mathrm{A} = (Q, \Sigma, E, \{q_0\}, F)$ egy determinisztikus véges automata. A vele ekvivalens teljes, determinisztikus véges automata pedig $\mathrm{A}' = (Q \cup \{s\}, \Sigma, E', \{q_0\}, F)$, ahol s egy új állapot és $E' = E \cup \big\{(p,a,s) \mid \delta(p,a) = \emptyset, p \in Q, a \in \Sigma\big\} \cup \big\{(s,a,s) \mid a \in \Sigma\big\}$. Mivel ez az új állapot nem produktív, könnyű belátni, hogy $L(\mathrm{A}) = L(\mathrm{A}')$.

A következőkben, amennyiben csak véges automatát írunk, ezalatt mindig nemdeterminisztikus véges automatát értünk. Ha az automata determinisztikus, akkor ezt mindig hangsúlyozzuk.

Az átmenetfüggvény segítségével könnyen elkészíthetjük a véges automata átmenettáblázatát. A táblázat sorai Q elemeivel, oszlopai Σ elemeivel vannak indexelve. A $q \in Q$ sor és az $a \in \Sigma$ oszlop kereszteződésénél található elem $\delta(q,a)$. A 2.2. ábra esetében az átmenettáblázat a következő :

δ	0	1	2
q_0	$\{q_0\}$	$\{q_1\}$	$\{q_2\}$
q_1	$\{q_1\}$	$\{q_2\}$	$\{q_0\}$
q_2	$\{q_2\}$	$\{q_0\}$	$\{q_1\}$

A 2.3. ábrán látható két véges automata egyike sem determinisztikus, az első (az A véges automata) azért mert két kezdőállapota van, a második (a B véges automata) pedig azért, mert a q_0 állapotból 0-val a q_0 és q_1 állapotokba is el lehet jutni. E két véges automata átmenettáblázata a 2.4. ábrán látható. $L(\mathrm{A})$ azon szavak halmaza $\Sigma = \{0,1\}$ felett, amelyek nem kezdődnek két 0-val (természetesen az ε is eleme a nyelvnek), $L(\mathrm{B})$ pedig azon szavaké, amelyekben van 01 részszó.

[1] Nem értelemzavaró az, hogy ugyanazt a jelölést használjuk a halmaz számosságára, mint a szó hosszára, hiszen a szót mindig kisbetűvel jelöljük, a halmazt pedig nagybetűvel. Kivétel csak a $\delta(q,a)$ jelölés, amely nem téveszthető össze szóval.

2.1.1. Elérhetetlen állapotok kizárása

Legyen $\mathrm{A} = (Q, \Sigma, E, I, F)$ egy véges automata. Egy állapotot ***elérhetőnek*** nevezünk, ha egy kezdőállapotból valamely bemeneti szó hatására eljuthatunk ebbe az állapotba, azaz, ha létezik séta valamely kezdőállapotból ebbe az állapotba. A következő algoritmus meghatározza egy véges automatában az elérhető állapotokat az U_0, U_1, $U_2, \ldots$ halmazok felépítésével, ahol U_0 a kezdőállapotok halmaza, és tetszőleges $i \geq 1$ természetes számra U_i azon állapotok halmaza, amelyekbe el lehet jutni valamely kezdőállapotból egy legfeljebb i hosszúságú bemeneti szó hatására.

ELÉRHETŐ-ÁLLAPOTOK(A)
1 $U_0 \leftarrow I$
2 $i \leftarrow 0$
3 **repeat**
4 $\quad i \leftarrow i + 1$
5 $\quad U_i \leftarrow U_{i-1}$
6 $\quad$ **for** minden $q \in U_{i-1}$
7 $\quad\quad$ **do for** minden $a \in \Sigma$
8 $\quad\quad\quad$ **do** $U_i \leftarrow U_i \cup \delta(q, a)$
9 **until** $U_i = U_{i-1}$
10 $U \leftarrow U_i$
11 **return** U

A $Q \setminus U$ halmaz elemei nem elérhető állapotok, ezért kizárhatók a véges automatából anélkül, hogy az általa felismert nyelvet megváltoztatnánk.

Ha $|Q| = n$ és $|\Sigma| = m$, akkor a fenti algoritmus lépésszáma legrosszabb esetben $O(n^2 m)$, mivel a két egymásba ágyazott ciklus lépésszáma legfeljebb nm, a **repeat** ciklusé pedig n.

Az így megkonstruált U halmaznak megvan az a tulajdonsága, hogy $L(\mathrm{A}) \neq \emptyset$ akkor és csakis akkor, ha $U \cap F \neq \emptyset$. Ezáltal a fenti algoritmus kiegészíthető az $U \cap F \neq \emptyset$ feltétellel, hogy eldöntse, hogy a felismert $L(\mathrm{A})$ nyelv üres-e vagy sem.

2.1.2. Nemproduktív állapotok kizárása

Legyen $\mathrm{A} = (Q, \Sigma, E, I, F)$ egy véges automata. Egy állapotot ***produktívnak*** nevezünk, ha abból az állapotból valamely bemeneti szó hatására eljuthatunk egy végállapotba, azaz, ha létezik séta ebből az állapotból egy végállapotba.

A produktív állapotok meghatározására szolgáló következő algoritmus használja a δ^{-1} függvényt, amelynek definíciója a következő:

$$\delta^{-1} : Q \times \Sigma \to \mathcal{P}(Q), \quad \delta^{-1}(p, a) = \{q \mid (q, a, p) \in E\} \ .$$

Ez a függvény egy p állapotra és egy a betűre megadja azt az állapothalmazt, amelynek elemeiből az a betű hatására el lehet jutni a p állapotba.

PRODUKTÍV-ÁLLAPOTOK(A)
1 $V_0 \leftarrow F$
2 $i \leftarrow 0$
3 **repeat**
4 $\quad i \leftarrow i + 1$
5 $\quad V_i \leftarrow V_{i-1}$
6 $\quad$ **for** minden $p \in V_{i-1}$
7 $\quad\quad$ **do for** minden $a \in \Sigma$
8 $\quad\quad\quad$ **do** $V_i \leftarrow V_i \cup \delta^{-1}(p, a)$
9 **until** $V_i = V_{i-1}$
10 $V \leftarrow V_i$
11 **return** V

A $Q \setminus V$ halmaz elemei nem produktív állapotok, ezért kizárhatók a véges automatából, anélkül, hogy az általa felismert nyelvet befolyásolnánk.

Ha n az állapotok száma és m a betűk száma, akkor a lépésszám ebben az esetben is $O(n^2m)$, akárcsak az ELÉRHETŐ-ÁLLAPOTOK algoritmus esetében.

Az így megkonstruált V halmaznak is megvan az a tulajdonsága, hogy $L(\mathrm{A}) \neq \emptyset$ akkor és csakis akkor, ha $V \cap I \neq \emptyset$. Ezért, kis módosítással, ez az algoritmus is használható annak eldöntésére, hogy $L(\mathrm{A})$ üres-e.

2.2. Nemdeterminisztikus véges automata átalakítása determinisztikus véges automatává

A következőkben megmutatjuk, hogy tetszőleges nemdeterminisztikus véges automata átalakítható olyan determinisztikus véges automatává, amelyik ekvivalens az eredetivel.

2.2.1. tétel. *Tetszőleges nemdeterminisztikus véges automatához mindig megkonstruálható egy vele ekvivalens determinisztikus véges automata.*

Bizonyítás. Legyen $\mathrm{A} = (Q, \Sigma, E, I, F)$ egy nemdeterminisztikus véges automata. Értelmezzük az $\overline{\mathrm{A}} = (\overline{Q}, \Sigma, \overline{E}, \overline{I}, \overline{F})$ determinisztikus véges automatát, ahol

$\overline{Q} = \mathcal{P}(Q) \setminus \emptyset$,

$\overline{E}$ élei azon (S, a, R) alakú hármasokból állnak, amelyekre $R, S \in \overline{Q}$, egyik sem üres, $a \in \Sigma$ és $R = \bigcup_{p \in S} \delta(p, a)$,

$\overline{I} = \{I\}$,

$\overline{F} = \{S \subseteq Q \mid S \cap F \neq \emptyset\}$.

Be kell bizonyítanunk, hogy $L(\mathrm{A}) = L(\overline{\mathrm{A}})$.

a) Bebizonyítjuk, hogy $L(\mathrm{A}) \subseteq L(\overline{\mathrm{A}})$. Legyen $w = a_1 a_2 \dots a_k \in L(\mathrm{A})$. Ekkor létezik a

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{k-1}} q_{k-1} \xrightarrow{a_k} q_k, \;\; q_0 \in I, \; q_k \in F$$

séta. Képezzük a következő halmazokat, felhasználva az $\overline{\mathrm{A}}$ véges automata $\overline{\delta}$ átmenetfüggvényét: $S_0 = \{q_0\}$, $\overline{\delta}(S_0, a_1) = S_1$, $\dots \overline{\delta}(S_{k-1}, a_k) = S_k$. Ekkor $q_1 \in S_1, \dots, q_k \in S_k$, és mivel $q_k \in F$, következik, hogy $S_k \cap F \neq \emptyset$, tehát $S_k \in \overline{F}$. Így létezik az

$$S_0 \xrightarrow{a_1} S_1 \xrightarrow{a_2} S_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{k-1}} S_{k-1} \xrightarrow{a_k} S_k, \;\; S_0 \subseteq I, \; S_k \in \overline{F}$$

séta. Vannak olyan $S'_0, \dots, S'_k$ halmazok, amelyekre $S'_0 = I$, továbbá minden $i = 0, 1, \dots, k$-ra $S_i \subseteq S'_i$ és

$$S'_0 \xrightarrow{a_1} S'_1 \xrightarrow{a_2} S'_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{k-1}} S'_{k-1} \xrightarrow{a_k} S'_k$$

is produktív séta. Ezért $w \in L(\overline{\mathrm{A}})$. Tehát $L(\mathrm{A}) \subseteq L(\overline{\mathrm{A}})$.

b) Bebizonyítjuk, hogy $L(\overline{\mathrm{A}}) \subseteq L(\mathrm{A})$. Legyen $w = a_1 a_2 \dots a_k \in L(\overline{\mathrm{A}})$. Ekkor létezik a

$$\overline{q_0} \xrightarrow{a_1} \overline{q_1} \xrightarrow{a_2} \overline{q_2} \xrightarrow{a_3} \cdots \xrightarrow{a_{k-1}} \overline{q}_{k-1} \xrightarrow{a_k} \overline{q}_k, \;\; \overline{q}_0 \in \overline{I}, \; \overline{q}_k \in \overline{F}$$

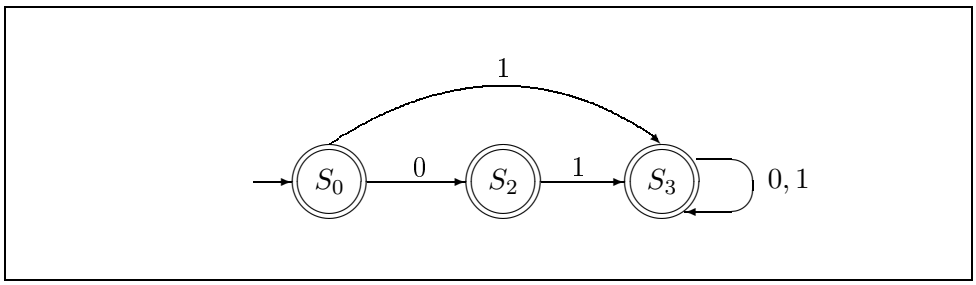
séta. Az $\overline{F}$ definíciója alapján $\overline{q}_k \cap F \neq \emptyset$, azaz létezik $q_k \in \overline{q}_k \cap F$, tehát $q_k \in F$ és $\overline{q}_k$ definíciója alapján létezik q_{k-1} úgy, hogy $(q_{k-1}, a_k, q_k) \in E$. Hasonlóképpen, léteznek a $q_{k-2}, \dots, q_1, q_0$ állapotok úgy, hogy $(q_{k-2}, a_k, q_{k-1}) \in E, \dots, (q_0, a_1, q_1) \in E$, ahol $q_0 \in \overline{q}_0 = I$, ezért létezik a

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{k-1}} q_{k-1} \xrightarrow{a_k} q_k, \;\; q_0 \in I, \; q_k \in F$$

séta, tehát $L(\overline{\mathrm{A}}) \subseteq L(\mathrm{A})$. □

A determinisztikus véges automata megkonstruálásában segítségünkre lehet ennek $\overline{\delta}$ átmenetfüggvénye, amely a δ segítségével a következőképpen értelmezhető

$$\overline{\delta}(\overline{q}, a) = \bigcup_{q \in \overline{q}} \delta(q, a), \;\; \forall \overline{q} \in \overline{Q}, \forall a \in \Sigma.$$



2.5. ábra.A 2.3. ábra A véges automatájával ekvivalens determinisztikus véges automata.

2.2.2. példa. Alkalmazzuk a 2.2.1 tételt a 2.3. ábra A véges automatájára, amely nemdeterminisztikus. Vezessük be a determinisztikus véges automata állapotaira a következő jelöléseket:

$S_0 := \{q_0, q_1\}$, $S_1 := \{q_0\}$, $S_2 := \{q_1\}$, $S_3 := \{q_2\}$,
$S_4 := \{q_0, q_2\}$, $S_5 := \{q_1, q_2\}$, $S_6 := \{q_0, q_1, q_2\}$,

amelyek közül S_0 a kezdőállapot. Ekkor alkalmazva az átmenetfüggvényt a következő átmenettáblázatot kapjuk:

$\overline{\delta}$	0	1
S_0	$\{S_2\}$	$\{S_3\}$
S_1	$\{S_2\}$	$\emptyset$
S_2	$\emptyset$	$\{S_3\}$
S_3	$\{S_3\}$	$\{S_3\}$
S_4	$\{S_5\}$	$\{S_3\}$
S_5	$\{S_3\}$	$\{S_3\}$
S_6	$\{S_5\}$	$\{S_3\}$

Ebben az automatában sok elérhetetlen állapot van. Az ELÉRHETŐ-ÁLLAPOTOK algoritmus szerint a véges automata elérhető állapotai a következők szerint határozhatók meg.

$U_0 = \{S_0\}$,
$U_1 = \{S_0, S_2, S_3\}$,
$U_2 = \{S_0, S_2, S_3\} = U_1 = U$.

Az S_0 kezdőállapot és egyben végállapot is. Az S_2 és S_3 mindegyike végállapot. Az S_1, S_4, S_5, S_6 elérhetetlen állapotok, ezért kizárjuk őket a véges automatából. Az így kapott véges automata átmenettáblázata a következő:

$\overline{\delta}$	0	1
S_0	$\{S_2\}$	$\{S_3\}$
S_2	$\emptyset$	$\{S_3\}$
S_3	$\{S_3\}$	$\{S_3\}$

Az ennek megfelelő determinisztikus véges automata átmenetgráfja a 2.5. ábrán látható. □

A 2.2.1 tétel által nyújtott algoritmus egyszerűsíthető. Nem kell a nemdeterminisztikus véges automata állapothalmazának minden részhalmazát figyelembe venni. Az $\overline{\mathrm{A}}$ véges automata állapotait fokozatosan kapjuk meg úgy, hogy elindulunk a $\overline{q}_0 = I$ állapottal, meghatározzuk a $\overline{\delta}(\overline{q}_0, a)$ állapotokat, minden $a \in \Sigma$ elemre. Az újonnan kapott állapotokra szintén meghatározzuk az átmenetek alapján a belőlük elérhető állapotokat. Ezt addig folytatjuk, amíg már nem kapunk új állapotokat.

Az előző példánkban legyen $\overline{q}_0 := \{q_0, q_1\}$ a kezdőállapot, és innen

$\overline{\delta}(\overline{q}_0, 0) = \{\overline{q_1}\}$, ahol $\overline{q}_1 := \{q_1\}$, $\quad \overline{\delta}(\overline{q}_0, 1) = \{\overline{q}_2\}$, ahol $\overline{q}_2 := \{q_2\}$,
$\overline{\delta}(\overline{q}_1, 0) = \emptyset$, $\quad \overline{\delta}(\overline{q}_1, 1) = \{\overline{q}_2\}$,
$\overline{\delta}(\overline{q}_2, 0) = \{\overline{q}_2\}$, $\quad \overline{\delta}(\overline{q}_2, 1) = \{\overline{q}_2\}$.

Az átmenettáblázat a következő:

$\overline{\delta}$	0	1
$\overline{q}_0$	$\{\overline{q}_1\}$	$\{\overline{q}_2\}$
$\overline{q}_1$	$\emptyset$	$\{\overline{q}_2\}$
$\overline{q}_2$	$\{\overline{q}_2\}$	$\{\overline{q}_2\}$

amely lényegében ugyanaz (ha eltekintünk a jelölésektől), mint az előbb kapott véges automata átmenettáblázata.

A következő algoritmus egy $\mathrm{A} = (Q, \Sigma, E, I, F)$ nemdeterminisztikus véges automatához megkonstruálja a vele ekvivalens $\overline{\mathrm{A}} = (\overline{Q}, \Sigma, \overline{E}, \overline{I}, \overline{F})$ determinisztikus véges automata M átmenettáblázatát, de nem tartalmazza annak megállapítását, hogy egy állapot végállapot-e vagy sem. Ez utóbbi azonban könnyűszerrel beépíthető. Az algoritmusban használjuk a BENNEVAN függvényt, amelyet nem írtunk le, de megjegyezzük, hogy a BENNEVAN$(\overline{q}, \overline{Q})$ értéke igaz, ha a $\overline{q}$ állapot már szerepel a $\overline{Q}$ halmazban, és hamis ellenkező esetben. Legyen $a_1, a_2, \ldots, a_m$ a Σ betűinek egy felsorolása.

NEMDET-DET(A)
1 $\overline{q}_0 \leftarrow I$
2 $\overline{Q} \leftarrow \{\overline{q}_0\}$
3 $i \leftarrow 0$ ▷ i a sorokat számolja.
4 $k \leftarrow 0$ ▷ k az állapotokat számolja.

```
5  repeat
6        for j = 1, 2, ..., m                          ▷ j az oszlopokat számolja.
7            do q̄ ← ⋃_{p∈q̄_i} δ(p, a_j)
8               if q̄ ≠ ∅
9                  then if Bennevan(q̄, Q̄)
10                           then M[i, j] ← {q̄}
11                           else k ← k + 1
12                                q̄_k ← q̄
13                                M[i, j] ← {q̄_k}
14                                Q̄ ← Q̄ ∪ {q̄_k}
15                 else M[i, j] ← ∅
16       i ← i + 1
17 until i = k + 1
18 return az Ā automata M átmenettáblázata
```

Mivel a **repeat** ciklust annyiszor végezzük el, ahány állapota van az új véges automatának, legrosszabb esetben ez exponenciális érték is lehet, hisz ha a nemdeterminisztikus véges automata állapotainak száma n, akkor az eredményautomatának akár $2^n - 1$ állapota is lehet. (Egy n elemű halmaz részhalmazainak a száma, beleértve az üres halmazt is, 2^n.)

A 2.2.1 tétel szerint tetszőleges nemdeterminisztikus véges automatához mindig hozzárendelhető egy vele ekvivalens determinisztikus véges automata. Ez fordítva is igaz, mivel az értelmezés szerint minden determinisztikus véges automata egyben nemdeterminisztikus is. Ezért a nemdeterminisztikus véges automaták ugyanazt a nyelvosztályt ismerik fel, mint a determinisztikus véges automaták.

2.3. Determinisztikus véges automaták ekvivalenciájának vizsgálata

Ebben a részben csak teljes, determinisztikus véges automatákkal dolgozunk. Ezen automaták esetében a $\delta(q, a)$ halmaz mindig egyetlen elemet tartalmaz. Néha egyszerűbb, bizonyos képletekben, a $\delta(q, a)$ halmaz helyett annak az elemét használni, ezért értelmezzük az egyelemű $A = \{a\}$ halmazra az $elem(A)$ függvényt, amely visszaadja az A halmaz egyetlen elemét, azaz $elem(A) = a$. Determinisztikus véges automaták ekvivalenciáját vizsgáljuk az azonos címkéjű, kezdőállapottal kezdődő séták segítségével a két véges automatában. Ha egyik séta végállapottal végződik, a másik pedig nem, akkor a két automata nyilvánvalóan nem lehet ekvivalens.

Adott két determinisztikus véges automata ugyanazon ábécé felett: $A = (Q, \Sigma, E, \{q_0\}, F)$ és $A' = (Q', \Sigma, E', \{q'_0\}, F')$, amelyek ekvivalenciáját vizsgáljuk. Készítünk egy táblázatot, amely (q, q') alakú állapotpárokat fog tartalmazni, ahol $q \in Q$ és $q' \in Q'$. A táblázat második oszlopától kezdődően a Σ ábécé minden betűjének megfeleltetünk egy oszlopot. Ha a táblázat i-edik sorának első eleme (q, q'), akkor az i-edik sor és az a betűhöz tartozó oszlop találkozásánál levő elem $\Big(elem\big(\delta(q,a)\big), elem\big(\delta'(q',a)\big)\Big)$ lesz.

	... a ...
...	...
(q, q')	$\Big(elem\big(\delta(q,a)\big), elem\big(\delta'(q',a)\big)\Big)$
...	...

A táblázat első sorának első oszlopába a (q_0, q'_0) állapotpár kerül, majd kitöltjük az első sort a fent leírtak alapján. Ha az első sor valamelyik oszlopában megjelenik egy olyan pár, amelyre egyik állapot végállapot, a másik meg nem, akkor az algoritmust befejezzük: ***a két deteminisztikus véges automata nem ekvivalens***. Amennyiben nincs ilyen pár, minden új párt beírunk az első oszlopba, és folytatjuk az algoritmust a következő olyan sorral, amelyik még nincs kitöltve. Ha már nem jelenik meg új állapotpár, és minden párra igaz, hogy mindkét eleme végállapot vagy egyik sem az, akkor az algoritmus szintén befejeződik, és ***a két deteminisztikus véges automata ekvivalens***.

AUTOMATA-EKVIVALENCIA(A, A′)

1 írjuk be a táblázat első sorának első oszlopába a (q_0, q'_0) állapotpárt
2 $i \leftarrow 0$
3 **repeat**
4 $\quad i \leftarrow i + 1$
5 $\quad$ legyen (q, q') a táblázat i-edik sorának első oszlopában levő állapotpár
6 $\quad$ **for** minden $a \in \Sigma$ betűre
7 $\quad\quad$ **do** írjuk be a táblázat i-edik sora a jelzésű oszlopába a $\Big(elem\big(\delta(q,a)\big), elem\big(\delta'(q',a)\big)\Big)$ állapotpárt
8 $\quad\quad$ **if** $\Big(elem\big(\delta(q,a)\big), elem\big(\delta'(q',a)\big)\Big)$ egyik állapota végállapot, a másik pedig nem
9 $\quad\quad\quad$ **then return** NEM

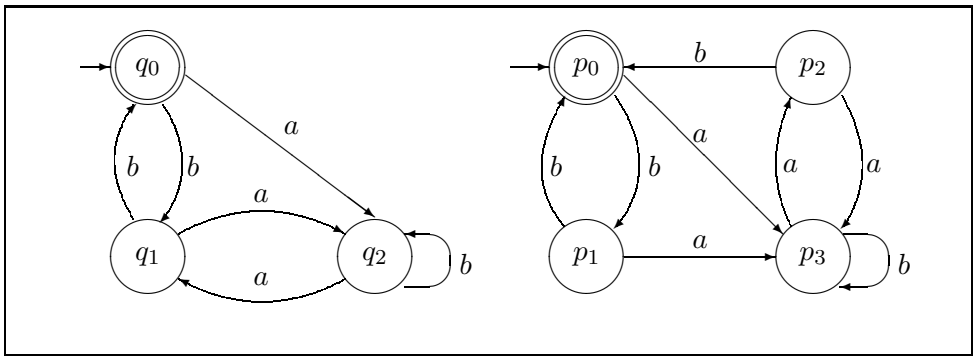
10 **else** írjuk be a $\Big(elem\big(\delta(q,a)\big), elem\big(\delta'(q',a)\big)\Big)$ párt az első oszlop következő üres sorába, ha még nem szerepel az első oszlopban

11 **until** $(i+1)$-edik sor első eleme üres

12 **return** IGEN

Ha $|Q| = n$, $|Q'| = n'$ és $|\Sigma| = m$, akkor figyelembe véve, hogy a **repeat** ciklust legrosszabb esetben nn'-szer kell végrehajtani, a **for** ciklust pedig m-szer, kiszámíthatjuk, hogy a maximális lépésszám legrosszabb esetben $O(nn'm)$, vagy ha $n = n'$, akkor $O(n^2m)$.

Algoritmusunkat két véges automata ekvivalenciájának vizsgálatára csak a teljes, determinisztikus véges automatákra írtuk le. Ha két tetszőleges nemdeterminisztikus véges automatáról szeretnénk eldönteni, hogy ekvivalensek-e, akkor előbb mindkettőt átalakítjuk determinisztikus véges automatává, majd alkalmazzuk a fentebb leírt algoritmust annak megállapítására, hogy ekvivalensek-e.



2.6. ábra. Ekvivalens véges automaták (2.3.1. példa)

2.3.1. példa. Vizsgáljuk meg, hogy a 2.6. ábrán látható két véges automata ekvivalens-e. Elkészítjük az állapotpárok következő táblázatát.

	a	b
(q_0, p_0)	(q_2, p_3)	(q_1, p_1)
(q_2, p_3)	(q_1, p_2)	(q_2, p_3)
(q_1, p_1)	(q_2, p_3)	(q_0, p_0)
(q_1, p_2)	(q_2, p_3)	(q_0, p_0)

A két véges automata ekvivalens, mivel minden lehetséges állapotpárt figyelembe vettünk, és minden pár mindkét eleme végállapot vagy egyik sem az. □

2.3.2. példa. A 2.7. ábrán látható két véges automata állapotpárjainak táblázata:

	a	b
(q_0, p_0)	(q_1, p_3)	(q_2, p_1)
(q_1, p_3)	(q_2, p_2)	(q_0, p_3)
(q_2, p_1)		
(q_2, p_2)		

A két véges automata nem ekvivalens, mivel a második sor utolsó oszlopában a (q_0, p_3) állapotpár első eleme végállapot, a második pedig nem az. □

2.4. Véges automaták és reguláris nyelvtanok ekvivalenciája

A nemdeterminisztikus véges automaták ugyanazt a nyelvosztályt ismerik fel, mint a determinisztikus véges automaták. A következő két tétel azt mutatja, hogy ez a nyelvosztály nem más, mint a reguláris nyelvek osztálya.

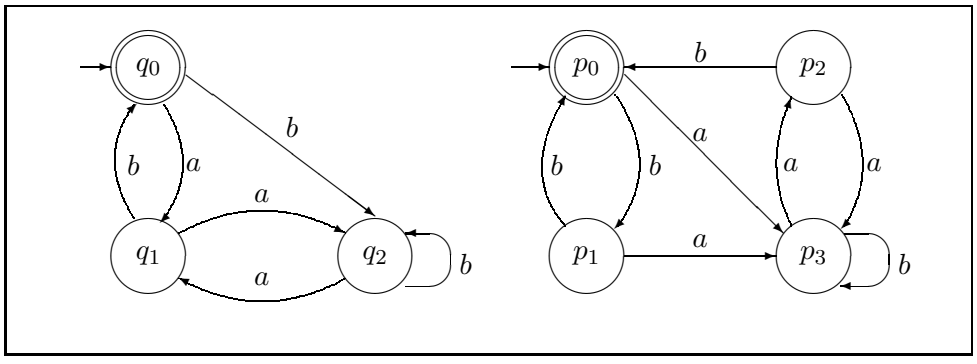
2.4.1. tétel. *Ha L egy tetszőleges determinisztikus véges automata által felismert nyelv, akkor megkonstruálható olyan reguláris nyelvtan, amelyik az L nyelvet generálja.*

Bizonyítás. Legyen $\mathrm{A} = (Q, \Sigma, E, \{q_0\}, F)$ az L nyelvet felismerő determinisztikus véges automata, azaz $L = L(\mathrm{A})$. Értelmezzük a $G = (Q, \Sigma, P, q_0)$ reguláris nyelvtant a következő szabályokkal:

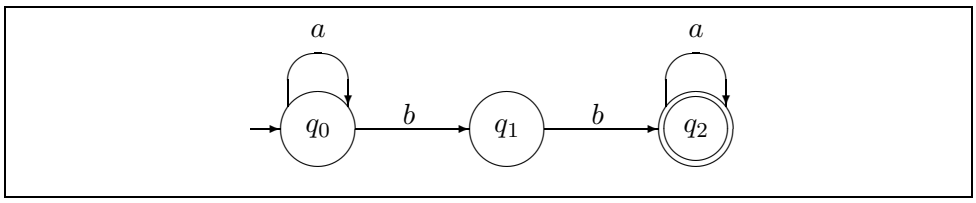
• Ha $(p, a, q) \in E$ valamilyen $p, q \in Q$ és $a \in \Sigma$-ra, akkor vegyük be P-be a $p \to aq$ szabályt.

• Ha $(p, a, q) \in E$ és $q \in F$, akkor a fenti szabály mellé még vegyük be P-be a $p \to a$ szabályt is.

Bebizonyítjuk, hogy $L(G) = L(\mathrm{A}) \setminus \{\varepsilon\}$.



2.7. ábra. Nem ekvivalens véges automaták (2.3.2. példa).



2.8. ábra. A 2.4.2. példa véges automatája.

Legyen $u = a_1 a_2 \dots a_n \in L(\mathrm{A})$ és $u \neq \varepsilon$. Ekkor, mivel az A véges automata felismeri az u szót, létezik a

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{n-1}} q_{n-1} \xrightarrow{a_n} q_n, \quad q_n \in F$$

séta. Ekkor P-ben léteznek a következő szabályok:

$$q_0 \to a_1 q_1, \quad q_1 \to a_2 q_2, \quad \dots, \quad q_{n-2} \to a_{n-1} q_{n-1}, \quad q_{n-1} \to a_n$$

(utóbbi szabály jobb oldalán nem szerepel q_n, mivel $q_n \in F$), tehát létezik a

$$q_0 \Longrightarrow a_1 q_1 \Longrightarrow a_1 a_2 q_2 \Longrightarrow \dots \Longrightarrow a_1 a_2 \dots a_{n-1} q_{n-1} \Longrightarrow a_1 a_2 \dots a_n$$

levezetés. Ezért $u \in L(G)$.

Fordítva, legyen $u = a_1 a_2 \dots a_n \in L(G)$, és $u \neq \varepsilon$. Ekkor létezik a

$$q_0 \Longrightarrow a_1 q_1 \Longrightarrow a_1 a_2 q_2 \Longrightarrow \dots \Longrightarrow a_1 a_2 \dots a_{n-1} q_{n-1} \Longrightarrow a_1 a_2 \dots a_n$$

levezetés, amelyben a

$$q_0 \to a_1 q_1, \quad q_1 \to a_2 q_2, \quad \dots, \quad q_{n-2} \to a_{n-1} q_{n-1}, \quad q_{n-1} \to a_n$$

szabályokat használtuk, amelyek értelmezés szerint azt jelentik, hogy az A véges automatában létezik a következő séta:

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{n-1}} q_{n-1} \xrightarrow{a_n} q_n,$$

és mivel q_n végállapot, következik, hogy $u \in L(\mathrm{A}) \setminus \{\varepsilon\}$.

Ha a determinisztikus véges automata ε-t is felismeri, akkor a fenti nyelvtan annyiban módosul, hogy bevezetünk egy új q_0' kezdőszimbólumot q_0 helyett, bevesszük a szabályok közé a $q_0' \to \varepsilon$ szabályt, majd minden $q_0 \to \alpha$ szabály mellé bevesszük a $q_0' \to \alpha$ szabályt is. □

2.4.2. példa. Adott az $\mathrm{A} = (\{q_0, q_1, q_2\}, \{a, b\}, E, \{q_0\}, \{q_2\})$ determinisztikus véges automata, ahol $E = \{(q_0, a, q_0), (q_0, b, q_1), (q_1, b, q_2), (q_2, a, q_2)\}$. A véges automata átmenettáblázata a következő:

δ	a	b
q_0	$\{q_0\}$	$\{q_1\}$
q_1	$\emptyset$	$\{q_2\}$
q_2	$\{q_2\}$	$\emptyset$

Az A átmenetgráfja a 2.8. ábrán látható. A 2.4.1 tétel alapján a $G = (\{q_0, q_1, q_2\}, \{a, b\}, P, q_0)$ reguláris nyelvtan P szabályai a következők:

$$q_0 \to aq_0 \mid bq_1, \qquad q_1 \to bq_2 \mid b, \qquad q_2 \to aq_2 \mid a.$$

Igazolható, hogy $L(\mathrm{A}) = \{a^m bba^n \mid m \geq 0, n \geq 0\}$. □

A 2.4.1 tétel bizonyításában megadott módszert könnyen átírhatjuk algoritmussá. Az $\mathrm{A} = (Q, \Sigma, E, \{q_0\}, F)$ determinisztikus automatából kapott $G = (Q, \Sigma, P, q_0)$ reguláris nyelvtan szabályait a következő algoritmussal határozzuk meg.

```
AUTOMATÁBÓL-REGULÁRIS-NYELVTAN(A)
 1  P ← ∅
 2  for minden p ∈ Q
 3      do for minden a ∈ Σ
 4              do for minden q ∈ Q
 5                      do if (p, a, q) ∈ E
 6                              then P ← P ∪ {p → aq}
 7                                   if q ∈ F
 8                                      then P ← P ∪ {p → a}
 9  if q0 ∈ F
10     then P ← P ∪ {q0 → ε}
11  return G
```

Amennyiben az automata felismeri az üres szót is, a fenti algoritmus esetleg kiterjesztett reguláris nyelvet generál.

Könnyű belátni, hogy az algoritmus futási ideje $\Theta(n^2 m)$, ha az állapotok száma n és a betűk száma m. A 2–4. sorokban lévő három ciklus helyett lehet csupán egyet venni, ha az E elemeit vizsgáljuk, ekkor a futási idő legrosszabb esetben $\Theta(p)$, ahol p az átmenetek száma. Ez szintén $O(n^2 m)$, mivel lehetséges, hogy minden átmenet jelen van. Ekkor az algoritmus a következőképpen írható le:

AUTOMATÁBÓL-REGULÁRIS-NYELVTAN'(A)
1 $P \leftarrow \emptyset$
2 **for** minden $(p, a, q) \in E$
3 **do** $P \leftarrow P \cup \{p \rightarrow aq\}$
4 **if** $q \in F$
5 **then** $P \leftarrow P \cup \{p \rightarrow a\}$
6 **if** $q_0 \in F$
7 **then** $P \leftarrow P \cup \{q_0 \rightarrow \varepsilon\}$
8 **return** G

2.4.3. tétel. *Ha $L = L(G)$ reguláris nyelv, akkor megkonstruálható olyan nemdeterminisztikus véges automata, amely az L nyelvet felismeri.*

Bizonyítás. Legyen $G = (N, T, P, S)$ az L nyelvet generáló reguláris nyelvtan. Definiáljuk az $\mathrm{A} = (Q, T, E, \{S\}, F)$ nemdeterminisztikus véges automatát a következőképpen.

- $Q = N \cup \{Z\}$, ahol $Z \notin N \cup T$ (vagyis egy új szimbólum),
- Minden $A \rightarrow aB$ szabályra bevesszük E-be az (A, a, B) átmenetet.
- Minden $A \rightarrow a$ szabályra bevesszük E-be az (A, a, Z) átmenetet.
- $F = \begin{cases} \{Z\} & \text{ha } G\text{-ben nem szerepel az } S \rightarrow \varepsilon \text{ szabály,} \\ \{Z, S\} & \text{ha } G\text{-ben szerepel az } S \rightarrow \varepsilon \text{ szabály.} \end{cases}$

Bebizonyítjuk, hogy $L(G) = L(A)$.
Legyen $u = a_1 a_2 \dots a_n \in L(G)$, $u \neq \varepsilon$. Ekkor létezik u-nak egy G-beli levezetése: $S \Longrightarrow a_1 A_1 \Longrightarrow a_1 a_2 A_2 \Longrightarrow \dots \Longrightarrow a_1 a_2 \dots a_{n-1} A_{n-1} \Longrightarrow a_1 a_2 \dots a_n$.
Ez a levezetés a következő szabályok alapján történt:

$$S \rightarrow a_1 A_1, \;\; A_1 \rightarrow a_2 A_2, \;\; \dots, \;\; A_{n-2} \rightarrow a_{n-1} A_{n-1}, \;\; A_{n-1} \rightarrow a_n.$$

Ekkor az A véges automata átmeneteinek értelmezése alapján létezik az

$$S \xrightarrow{a_1} A_1 \xrightarrow{a_2} A_2 \xrightarrow{a_3} \;\cdots\; \xrightarrow{a_{n-1}} A_{n-1} \xrightarrow{a_n} Z, \;\; Z \in F$$

séta. Ez azt jelenti, hogy $u \in L(\mathrm{A})$. Ha $\varepsilon \in L(G)$, akkor van $S \rightarrow \varepsilon$ szabály, de ekkor a kezdőállapot végállapot is, tehát $\varepsilon \in L(A)$. Ezért $L(G) \subseteq L(A)$.

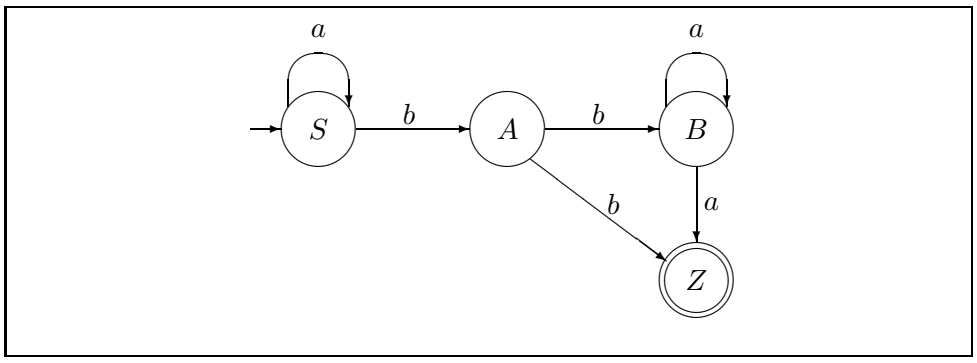
Legyen most $u = a_1 a_2 \dots a_n \in L(\mathrm{A})$. Ez azt jelenti, hogy létezik az

$$S \xrightarrow{a_1} A_1 \xrightarrow{a_2} A_2 \xrightarrow{a_3} \;\cdots\; \xrightarrow{a_{n-1}} A_{n-1} \xrightarrow{a_n} Z, \;\; Z \in F$$

séta. Ha u az üres szó, akkor Z helyett S van, amely szintén végállapot. Más esetben csak Z szerepelhet utolsóként. Tehát G-ben szerepelnek a következő szabályok: $S \rightarrow a_1 A_1, \;\; A_1 \rightarrow a_2 A_2, \;\; \dots, \;\; A_{n-2} \rightarrow a_{n-1} A_{n-1}, \;\; A_{n-1} \rightarrow a_n$, és így létezik az

$$S \Longrightarrow a_1 A_1 \Longrightarrow a_1 a_2 A_2 \Longrightarrow \dots \Longrightarrow a_1 a_2 \dots a_{n-1} A_{n-1} \Longrightarrow a_1 a_2 \dots a_n$$

levezetés, tehát $u \in L(G)$, és ekkor $L(A) \subseteq L(G)$. □



2.9. ábra. A 2.4.4. példa G nyelvtanához rendelt véges automata.

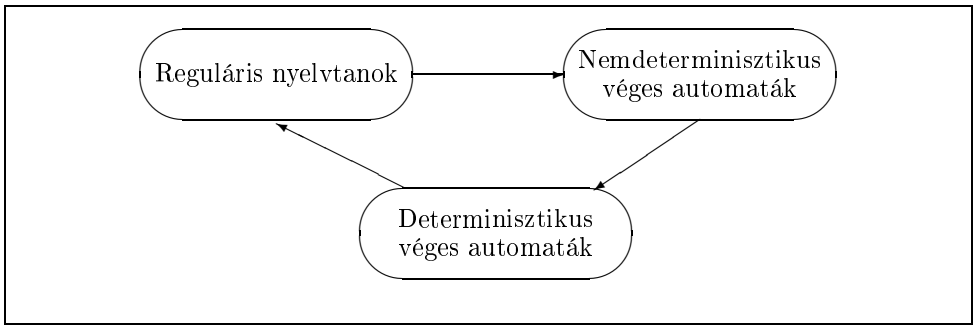
2.4.4. példa. Adott a $G = (\{S, A, B\}, \{a, b\}, \{S \to aS, S \to bA, A \to bB, A \to b, B \to aB, B \to a\}, S)$ reguláris nyelvtan. A hozzá rendelt véges automata $A = (\{S, A, B, Z\}, \{a, b\}, E, S, \{Z\})$, ahol $E = \big\{(S, a, S), (S, b, A), (A, b, B), (A, b, Z), (B, a, B), (B, a, Z)\big\}$. Ennek átmenettáblázata a következő:

δ	a	b
S	$\{S\}$	$\{A\}$
A	$\emptyset$	$\{B, Z\}$
B	$\{B, Z\}$	$\emptyset$
E	$\emptyset$	$\emptyset$

Az átmenetgráf 2.9. ábrán látható. Ez a véges automata egyszerűsíthető. A B és Z állapotok összevonhatók egyetlen végállapottá. □

Az előbbi tétel alapján írunk egy algoritmust, amely hozzárendeli a $G = (N, T, P, S)$ reguláris nyelvtanhoz az $\mathrm{A} = (Q, T, E, \{S\}, F)$ véges automatát.

```
REGULÁRIS-NYELVTANBÓL-AUTOMATA(G)
 1  E ← ∅
 2  Q ← N ∪ {Z}
 3  for minden A ∈ N
 4      do for minden a ∈ T
 5             do if (A → a) ∈ P
 6                   then E ← E ∪ {(A, a, Z)}
 7                for minden B ∈ N
 8                    do if (A → aB) ∈ P
 9                          then E ← E ∪ {(A, a, B)}
10  if (S → ε) ∉ P
11     then F ← {Z}
12     else  F ← {Z, S}
13  return A
```



2.10. ábra. Kapcsolat véges automaták és reguláris nyelvtanok között (2.4.5 tétel).

Akárcsak az AUTOMATÁBÓL-REGULÁRIS-NYELVTAN algoritmus esetében, a futási idő ebben az esetben is $\Theta(n^2m)$, ha a nemterminálisok száma n és a terminálisoké m. Lehetne a 3., 4. és 7. sorokban lévő ciklusokat helyettesíteni eggyel, amelyik a helyettesítési szabályokon megy végig. Ekkor az algoritmus lépésszáma $\Theta(p)$ lesz, ha p a szabályok száma. Az algoritmus a következő:

REGULÁRIS-NYELVTANBÓL-AUTOMATA'(G)
1 $E \leftarrow \emptyset$
2 $Q \leftarrow N \cup \{Z\}$
3 **for** minden $(A \rightarrow u) \in P$
4 **do if** $u = a$
5 **then** $E \leftarrow E \cup \{(A, a, Z)\}$
6 **if** $u = aB$
7 **then** $E \leftarrow E \cup \{(A, a, B)\}$
8 **if** $(S \rightarrow \varepsilon) \notin P$
9 **then** $F \leftarrow \{Z\}$
10 **else** $F \leftarrow \{Z, S\}$
11 **return** A

A 2.2.1, 2.4.1 és 2.4.3 tételek segítségével bebizonyítottuk, hogy a reguláris nyelvek osztálya egybeesik mind a determinisztikus véges automaták, mind a nemdeterminisztikus véges automaták által felismert nyelvek osztályával. A három tétel eredményét a 2.10. ábra szemlélteti és következő tétel foglalja össze.

2.4.5. tétel. *A következő három nyelvosztály megegyezik:*

- *a reguláris nyelvek osztálya,*
- *a determinisztikus véges automatákkal felismerhető nyelvek osztálya,*
- *a nemdeterminisztikus véges automatákkal felismerhető nyelvek osztálya.*

2.4.1. Műveletek reguláris nyelvekkel

A 1.5.1 tétel alapján tudjuk, hogy a reguláris nyelvek $\mathcal{L}_3$ halmaza zárt a reguláris műveletekre, azaz ha L_1, L_2 reguláris, akkor regulárisak a következő nyelvek is: $L_1 \cup L_2$, $L_1 L_2$, L_1^*. Ezenkívül a reguláris nyelvekre igazak a következő állítások is.

Egy reguláris nyelvnek a komplementuma is reguláris. Ez könnyen igazolható véges automaták segítségével. Legyen ugyanis L egy reguláris nyelv és $\mathrm{A} = (Q, \Sigma, E, \{q_0\}, F)$ egy, az L nyelvet felismerő teljes, determinisztikus véges automata. Könnyen belátható, hogy az $\overline{\mathrm{A}} = (Q, \Sigma, E, \{q_0\}, Q \setminus F)$ automata az $\overline{L}$ nyelvet ismeri fel. Így $\overline{L}$ is reguláris.

Két reguláris nyelvnek a metszete is reguláris. Mivel $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$, a metszet is reguláris.

Két reguláris nyelvnek a különbsége is reguláris. Mivel $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$, a különbség is reguláris.

2.5. ε-lépéses véges automaták és műveletek véges automatákkal

Az ε-lépéses véges automata annyiban különbözik a nemdeterminisztikus véges automatától, hogy megengedjük azt, hogy üres lépést is végezzen, azaz átmenjen egyik állapotból a másikba anélkül, hogy valamilyen bemeneti jelet olvasna. Az ε-lépéses $\mathrm{A} = (Q, \Sigma, E, I, F)$ véges automata átmeneteinek halmazára teljesül, hogy $E \subseteq Q \times \big(\Sigma \cup \{\varepsilon\}\big) \times Q$.

Az ε-lépéses véges automata átmenetfüggvénye a következő:

$$\delta : Q \times \big(\Sigma \cup \{\varepsilon\}\big) \to \mathcal{P}(Q), \quad \delta(p, a) = \{q \in Q \mid (p, a, q) \in E\} \ .$$

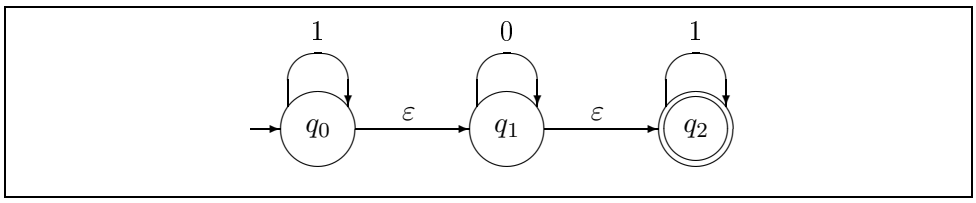
A 2.11. ábrán látható ε-lépéses véges automata az uvw alakú szavakat ismeri fel, ahol $u \in \{1\}^*$, $v \in \{0\}^*$ és $w \in \{1\}^*$.

2.5.1. tétel. *Tetszőleges ε-lépéses véges automatához mindig megkonstruálható egy vele ekvivalens nemdeterminisztikus véges automata, amely ε-lépés nélküli.*

Az $\mathrm{A} = (Q, \Sigma, E, I, F)$ ε-lépéses véges automatával ekvivalens nemdeterminisztikus véges automata $\overline{\mathrm{A}} = (Q, \Sigma, \overline{E}, I, \overline{F})$ lesz. Algoritmusunk az $\overline{F}$ és az $\overline{E}$ halmazokat határozza meg.

Egy q állapotra jelöljük $\Lambda(q)$-val azon állapotok halmazát, amelyekbe el lehet jutni q-ból csupa ε-lépéssel (beleértve magát q-t is). Terjesszük ki ezt definíciót halmazokra is, azaz legyen

$$\Lambda(S) = \bigcup_{q \in S} \Lambda(q), \quad \forall S \subseteq Q \ .$$



2.11. ábra. ε-lépéses véges automata.

Nyilvánvaló, hogy minden $q \in Q$-ra és $S \subseteq Q$-ra mind $\Lambda(q)$, mind $\Lambda(S)$ kiszámíthatók. A következőkben feltesszük, hogy ezek adottak.

A következő algoritmus az átmenetek meghatározását a $\overline{\delta}$ átmenetfüggvény segítségével végzi, amelyet az algoritmus 5. sorában értelmezünk.

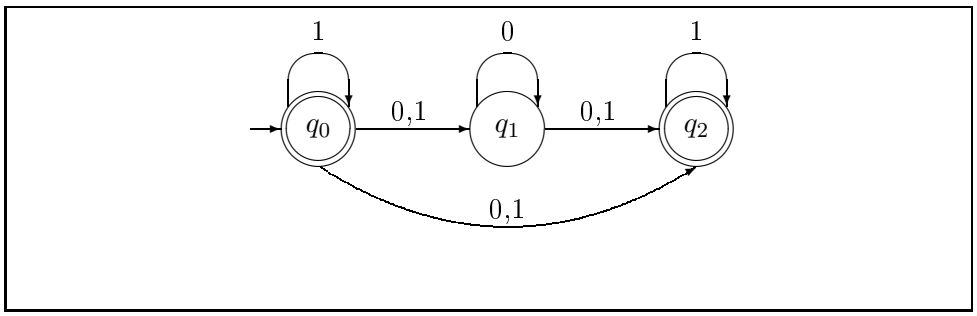
Ha $|Q| = n$ és $|\Sigma| = m$,, akkor a pszeudokód 2–6. soraiból látszik, hogy az algoritmus futási ideje legrosszabb esetben $O(n^2 m)$.

EPSZILON-MENTESÍTÉS(A)
1 $\overline{F} \leftarrow F \cup \{q \in I \mid \Lambda(q) \cap F \neq \emptyset\}$
2 **for** minden $q \in Q$
3 **do for** minden $a \in \Sigma$
4 **do** $\Delta \leftarrow \bigcup_{p \in \Lambda(q)} \delta(p, a)$
5 $\overline{\delta}(q, a) \leftarrow \Delta \cup \left(\bigcup_{p \in \Delta} \Lambda(p) \right)$
6 $\overline{E} \leftarrow \left\{ (p, a, q), \mid p, q \in Q,\ a \in \Sigma, q \in \overline{\delta}(p, a) \right\}$
7 **return** $\overline{\mathrm{A}}$

2.5.2. példa. Tekintsük a 2.11. ábrán lévő automatát, amelynek átmenettáblázata a következő:

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

Alkalmazzuk az EPSZILON-MENTESÍTÉS algoritmust.
$\Lambda(q_0) = \{q_0, q_1, q_2\},\ \Lambda(q_1) = \{q_1, q_2\},\ \Lambda(q_2) = \{q_2\}$
$\Lambda(I) = \Lambda(q_0)$, és ennek metszete az F-fel nem üres, ezért $\overline{F} = F \cup \{q_0\} = \{q_0, q_2\}$.
$(q_0, 0)$:
$\Delta = \delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0) = \{q_1\},\quad \{q_1\} \cup \Lambda(q_1) = \{q_1, q_2\}$
$\overline{\delta}(q_0, 0) = \{q_1, q_2\}$.



2.12. ábra. A 2.11. ábrán lévő ε-lépéses véges automatával ekvivalens ε-lépésmentes véges automata.

$(q_0, 1)$:
$\Delta = \delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1) = \{q_0, q_2\}$, $\{q_0, q_2\} \cup (\Lambda(q_0) \cup \Lambda(q_2)) = \{q_0, q_1, q_2\}$
$\overline{\delta}(q_0, 1) = \{q_0, q_1, q_2\}$
$(q_1, 0)$:
$\Delta = \delta(q_1, 0) \cup \delta(q_2, 0) = \{q_1\}$, $\{q_1\} \cup \Lambda(q_1) = \{q_1, q_2\}$
$\overline{\delta}(q_1, 0) = \{q_1, q_2\}$
$(q_1, 1)$:
$\Delta = \delta(q_1, 1) \cup \delta(q_2, 1) = \{q_2\}$, $\{q_2\} \cup \Lambda(q_2) = \{q_2\}$
$\overline{\delta}(q_1, 1) = \{q_2\}$
$(q_1, 1) : \Delta = \delta(q_2, 0) = \emptyset$
$\overline{\delta}(q_2, 0) = \emptyset$
$(q_2, 1)$:
$\Delta = \delta(q_2, 1) = \{q_2\}$, $\{q_2\} \cup \Lambda(q_2) = \{q_2\}$
$\overline{\delta}(q_2, 1) = \{q_2\}$.

Tehát a $\overline{\mathrm{A}}$ véges automata átmenettáblázata a következő:

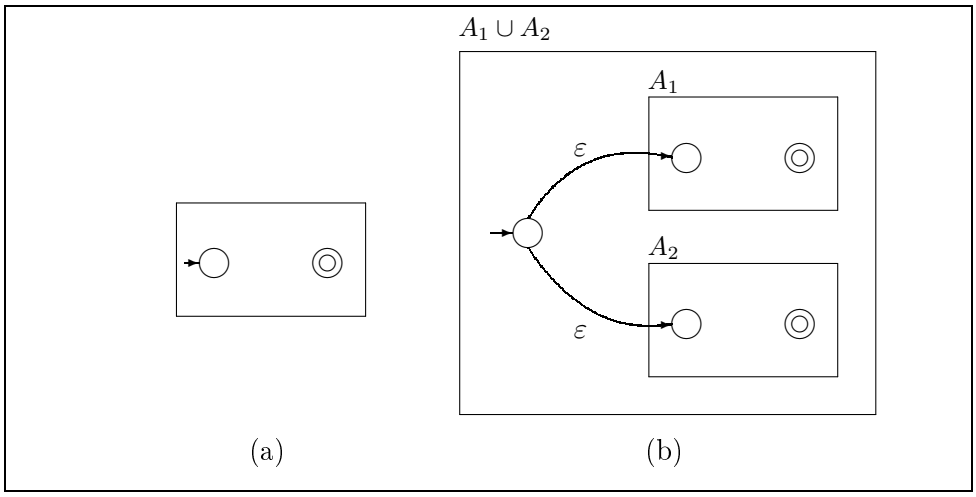
$\overline{\delta}$	0	1
q_0	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$
q_1	$\{q_1, q_2\}$	$\{q_2\}$
q_2	$\emptyset$	$\{q_2\}$

az átmenetdiagram pedig a 2.12. ábrán látható. □

A következőkben értelmezzük a véges automatákon a reguláris műveleteket: egyesítés, szorzat, iteráció. Eredményül ε-lépéses automatát kapunk.

A műveleteket szemléletesen ábrákkal is megadjuk, egy véges automatát a 2.13.(a) ábrán látható módon ábrázolunk. Egy nyíllal ellátott körrel jelöljük a kezdőállapotokat, és két koncentrikus körből álló jellel a végállapotokat.

Legyenek $\mathrm{A}_1 = (Q_1, \Sigma_1, E_1, I_1, F_1)$ és $\mathrm{A}_2 = (Q_2, \Sigma_2, E_2, I_2, F_2)$ véges automaták. A művelet eredménye az A-val jelölt $\mathrm{A} = (Q, \Sigma, E, I, F)$ ε-lépéses automata. Feltételezzük, hogy minden esetben $Q_1 \cap Q_2 = \emptyset$. Ha ez nem teljesül, akkor valamelyik állapothalmaz elemeit átnevezzük.



2.13. ábra.(a) Véges automata ábrázolása. Egy bemenő nyíl jelzi a kezdőállapotokat, míg két koncentrikus kör a végállapotokat. **(b)** Két véges automata egyesítése.

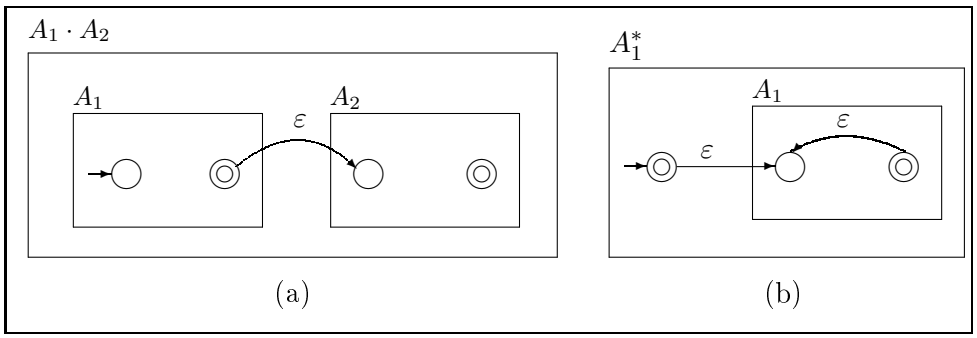
Egyesítés. $\mathrm{A} = \mathrm{A}_1 \cup \mathrm{A}_2$, ahol

$Q = Q_1 \cup Q_2 \cup \{q_0\}$,
$\Sigma = \Sigma_1 \cup \Sigma_2$,
$I = \{q_0\}$,
$F = F_1 \cup F_2$,
$E = E_1 \cup E_2 \cup \bigcup_{q \in I_1 \cup I_2} \{(q_0, \varepsilon, q)\}$.

Az eredményautomata a 2.13.(b) ábrán látható. Ugyanazt az eredményt kapjuk, ha kezdőállapot-halmaznak vesszük a $I_1 \cup I_2$ halmazt egy újabb kezdőállapot helyett. Ekkor egyáltalán nem lesznek ε-lépések. Az egyesítés definíciója alapján belátható, hogy $L(A_1 \cup A_2) = L(A_1) \cup L(A_2)$.

Szorzat. $\mathrm{A} = \mathrm{A}_1 \cdot \mathrm{A}_2$, ahol

$Q = Q_1 \cup Q_2$,
$\Sigma = \Sigma_1 \cup \Sigma_2$,
$F = F_2$,
$I = I_1$,
$E = E_1 \cup E_2 \cup \bigcup_{\substack{p \in F_1 \\ q \in I_2}} \{(p, \varepsilon, q)\}$



2.14. ábra. (a) Két véges automata szorzata. **(b)** Véges automata iteráltja.

Az eredményautomata a 2.14.(a) ábrán látható. Itt is beláthatjuk, hogy $L(A_1 \cdot A_2) = L(A_1)L(A_2)$.

Iteráció. $\mathrm{A} = \mathrm{A_1}^*$, ahol

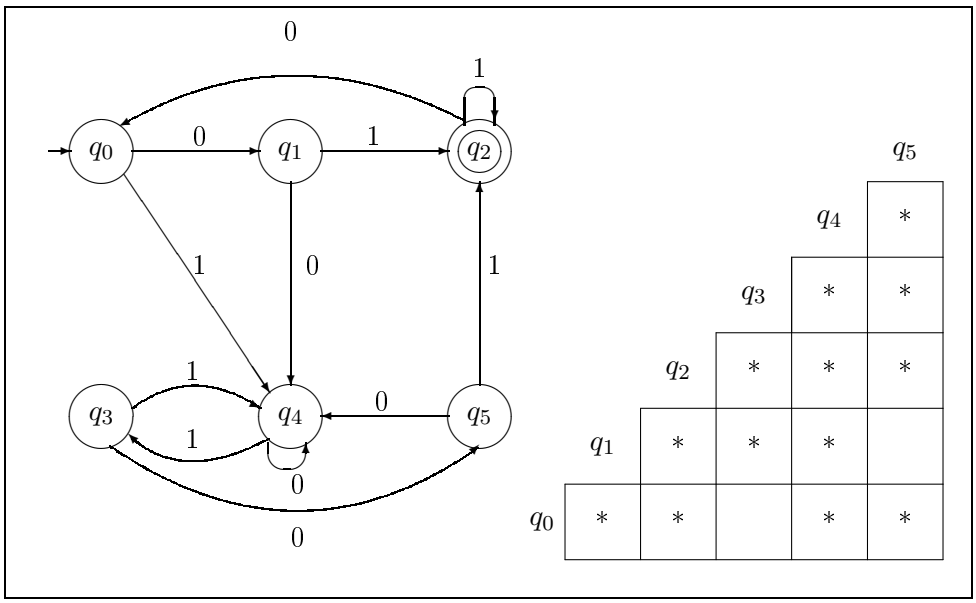
$Q = Q_1 \cup \{q_0\}$,
$\Sigma = \Sigma_1$,
$F = F_1 \cup \{q_0\}$,
$I = \{q_0\}$
$E = E_1 \cup \bigcup_{p \in I_1} \{(q_0, \varepsilon, p)\} \cup \bigcup_{\substack{q \in F_1 \\ p \in I_1}} \{(q, \varepsilon, p)\}$.

A véges automata iteráltja a 2.14.(b) ábrán látható. Erre a műveletre az teljesül, hogy $L(A_1^*) = \big(L(A_1)\big)^*$.

Az előbbiekben definiált három művelet segítségével újabb bizonyítását adtuk annak, hogy a reguláris nyelvek zártak a reguláris műveletekre nézve, ugyanis az eredményül kapott ε-lépéses automata átalakítható nemdeterminisztikus véges automatává.

2.6. Determinisztikus véges automaták minimalizálása

Egy $\mathrm{A} = (Q, \Sigma, E, \{q_0\}, F)$ teljes, determinisztikus véges automatát ***minimálisnak*** nevezünk, ha bármely vele ekvivalens $\mathrm{A}' = (Q', \Sigma, E', \{q_0'\}, F')$ teljes, determinisztikus véges automata esetében teljesül, hogy $|Q| \leq |Q'|$. A következőkben megadunk egy algoritmust, amely tetszőleges teljes, determinisztikus véges automatához megkonstruál egy vele ekvivalens minimális és teljes, determinisztikus véges automatát.



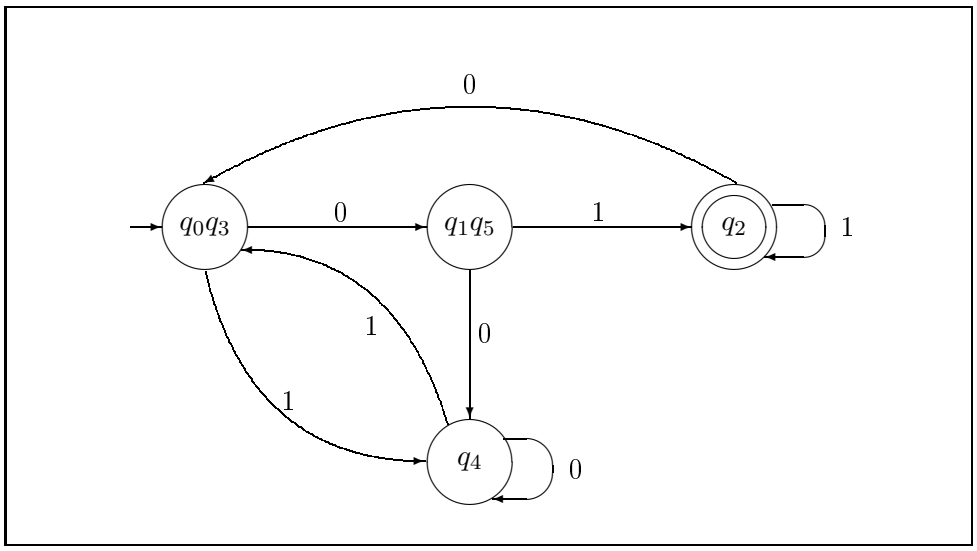
2.15. ábra. Véges automata minimalizálása.

Az $A = (Q, \Sigma, E, \{q_0\}, F)$ determinisztikus véges automata p és q állapotát ***ekvivalensnek*** mondjuk, ha tetszőleges u szóra, mindkettőből végállapotba jutunk vagy egyikből sem jutunk végállapotba, azaz

$$p \equiv q \text{ ha minden } u \in \Sigma^* \text{ szóra } \begin{cases} p \xrightarrow{u} r,\ r \in F \text{ és } q \xrightarrow{u} s,\ s \in F \text{ vagy} \\ p \xrightarrow{u} r,\ r \notin F \text{ és } q \xrightarrow{u} s,\ s \notin F\ . \end{cases}$$

Ha két állapot nem ekvivalens, akkor azt mondjuk, hogy megkülönböztethetők. Az alábbi algoritmusban csillaggal jelöljük a megkülönböztethető állapotpárokat, az egymással ekvivalenseket pedig összevonjuk. Az algoritmus során bizonyos (nem rendezett) állapotpárokhoz állapotpárokból álló listát rendelünk a későbbi megcsillagozás reményében, azaz ha az algoritmus során egy állapotpárt megcsillagoztunk, akkor megcsillagozzuk a hozzárendelt lista összes elemét is. Az alábbi algoritmust olyan determinisztikus véges automatára alkalmazzuk, amelyből már kizártuk az elérhetetlen állapotokat. Mivel a véges automata determinisztikus és teljes, a $\delta(p, a)$ pontosan egy elemet tartalmaz, itt is alkalmazzuk a 29. oldalon definiált *elem* függvényt, amely az egyelemű halmaz egyetlen elemét adja vissza.

AUTOMATA-MINIMALIZÁLÁSA(A)
1 jelöljük meg egy-egy csillaggal az összes olyan $\{p, q\}$ állapotpárt, amelyre $p \in F$ és $q \notin F$ vagy fordítva.



2.16. ábra.A 2.15. ábrán látható determinisztikus véges automata minimalizált változata.

2 minden jelöletlen $\{p, q\}$ állapotpárhoz rendeljünk egy üres listát,
3 minden jelöletlen $\{p, q\}$ állapotpárra és minden $a \in \Sigma$ betűre
vizsgáljuk meg az $\{elem(\delta(p, a)), elem(\delta(q, a))\}$ állapotpárokat,
ha az így kapott állapotpárok közül valamelyik meg van
csillagozva, akkor csillagozzuk meg a $\{p, q\}$ párt is,
egyetemben a már előzőleg a $\{p, q\}$ párhoz rendelt lista elemeivel,
különben, ha a fenti állapotpárok közül egy sincs megcsillagozva,
akkor írjuk be a $\{p, q\}$ párt a $\{elem(\delta(p, a)), elem(\delta(q, a))\}$
párokhoz rendelt lista mindegyikébe, feltéve, hogy $\delta(p, a) \neq \delta(q, a)$,
3 vonjuk össze a megjelöletlen (ekvivalens) állapotpárokat
4 **return** A

Az algoritmus befejeztével, ha a táblázatban egy cella nem tartalmaz csillagot, akkor a neki megfelelő sor és oszlop indexe két ekvivalens állapot, tehát összevonható. Az összevonást mindaddig folytatjuk, ameddig csak lehetséges. Általánosan, az ekvivalenciareláció az állapotok halmazát ekvivalenciaosztályokra bontja. Minden ilyen osztály állapotai egyetlen állapottá vonhatók össze.

Megjegyzés. Algoritmusunk abban az esetben is alkalmazható, ha a determinisztikus véges automata nem teljes, azaz vannak olyan állapotok, amelyekből adott bemeneti jelre nincs átmenet. Ekkor $\{\emptyset, \{q\}\}$ pár is előfordulhat, és ha q végállapot, úgy tekintjük, mintha ez a pár meg lenne csillagozva.

2.6.1. példa. Tekintsük a 2.15. ábrán látható determinisztikus véges automatát. Az algoritmus alkalmazásához egy táblázatot használunk, amelyben a csillagozást végezzük. A $\{p, q\}$ állapotpár megjelölését (megcsillagozását) a p sor és q oszlop (vagy q sor és p oszlop) találkozásánál elhelyezett csillag jelzi.

Először megcsillagozzuk a $\{q_2, q_0\}$, $\{q_2, q_1\}$, $\{q_2, q_3\}$, $\{q_2, q_4\}$ és $\{q_2, q_5\}$ párokat (mivel q_2 az egyetlen végállapot). Ezután sorra vesszük a csillaggal meg nem jelölt állapotpárokat, és az algoritmus szerint megvizsgáljuk őket. Kezdjük a $\{q_0, q_1\}$ párral. Hozzárendeljük a következő állapotpárokat: $\{elem\big(\delta(q_0, 0)\big), elem\big(\delta(q_1, 0)\big)\}, \{elem\big(\delta(q_0, 1)\big), elem\big(\delta(q_1, 1)\big)\}$, azaz $\{q_1, q_4\}$, $\{q_4, q_2\}$. Mivel $\{q_4, q_2\}$ már meg van jelölve, megjelöljük $\{q_0, q_1\}$-t is.

A $\{q_0, q_3\}$ pár esetében a két új pár $\{q_1, q_5\}$ és $\{q_4, q_4\}$. A $\{q_1, q_5\}$ párhoz hozzárendeljük egy listában a $\{q_0, q_3\}$-t, azaz $\{q_1, q_5\} \longrightarrow \{q_0, q_3\}$. Most $\{q_1, q_5\}$-tel folytatva, a $\{q_4, q_4\}$ és $\{q_2, q_2\}$ párokat kapjuk, amelyekhez az algoritmus szerint semmit sem rendelünk. Folytatjuk a $\{q_0, q_4\}$ párral. A hozzárendelt párok $\{q_1, q_4\}$ és $\{q_4, q_3\}$. Egyik sincs megcsillagozva, ezért hozzájuk rendeljük egy-egy listában a $\{q_0, q_4\}$ párt, azaz $\{q_1, q_4\} \longrightarrow \{q_0, q_4\}$, $\{q_4, q_3\} \longrightarrow \{q_0, q_4\}$. Most a $\{q_1, q_4\}$ párral folytatva a $\{q_4, q_4\}$, $\{q_2, q_3\}$ párokat kapjuk, és mivel ez utóbbi meg van jelölve csillaggal, megjelöljük a $\{q_1, q_4\}$ párt és a listában hozzárendelt $\{q_0, q_4\}$ párt is. Így folytatva, eljutunk a 2.15. ábrán látható táblázathoz, azaz azt kapjuk, hogy $q_0 \equiv q_3$ és $q_1 \equiv q_5$. Ezeket összevonva, a 2.16. ábrán látható determinisztikus véges automatát kapjuk, amelyik ekvivalens az eredetivel. □

2.7. Pumpáló lemma reguláris nyelvekre

A következő tétel, amelyet *pumpáló lemmának* nevezünk, jól használható arra, hogy egy nyelvről bebizonyítsuk, hogy nem reguláris. Ez a tétel egy szükséges feltételt ad meg arra, hogy egy nyelv reguláris legyen.

2.7.1. tétel. (pumpáló lemma) *Bármely L reguláris nyelv esetében létezik olyan $n \geq 1$ természetes szám (amely csak L-től függ), hogy L bármely legalább n hosszúságú u szava felírható $u = xyz$ alakban úgy, hogy*

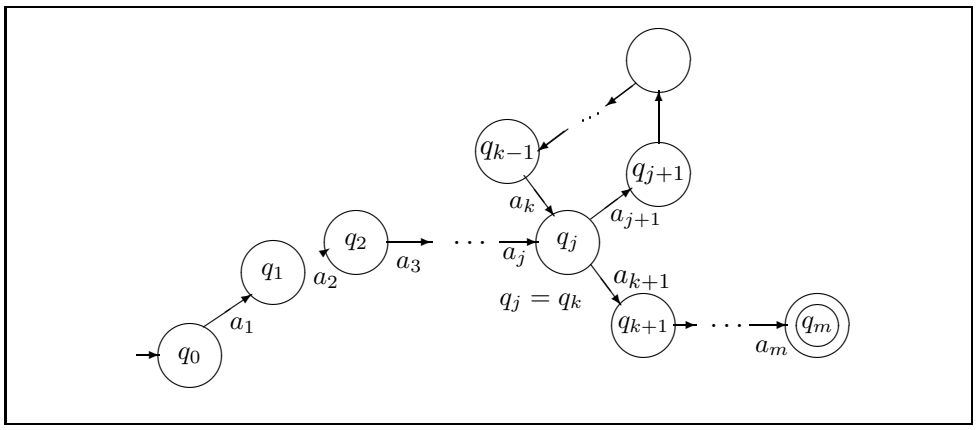
(1) $|xy| \leq n$,

(2) $|y| \geq 1$,

(3) $xy^iz \in L$ *minden* $i = 0, 1, 2, \ldots$ *értékre.*

Bizonyítás. Ha L reguláris nyelv, akkor létezik olyan determinisztikus véges automata, amely felismeri az L nyelvet (2.4.3 és 2.2.1 tételek alapján). Legyen ez a determinisztikus véges automata $\mathrm{A} = (Q, \Sigma, E, \{q_0\}, F)$, tehát $L = L(\mathrm{A})$. Legyen n az automata állapotainak száma, azaz $|Q| = n$. Legyen $u = a_1a_2 \ldots a_m \in L$ és $m \geq n$. Ekkor, mivel a determinisztikus véges automata az u szót felismeri, léteznek a $q_0, q_1, \ldots, q_m$ állapotok és a

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{m-1}} q_{m-1} \xrightarrow{a_m} q_m,\ q_m \in F$$



2.17. ábra.A pumpáló lemma bizonyításában használt determinisztikus véges automata rajza.

séta. Mivel összesen csak n állapotunk van, és $m \geq n$, a skatulya-elv[2] alapján a $q_0, q_1, \ldots, q_m$ állapotok között van legalább két megegyező (2.17. ábra). Legyen $q_j = q_k$, ahol $j < k$ és k a legkisebb ilyen index. Ekkor $j < k \leq n$. Bontsuk fel az u szót a következőképpen:

$x = a_1 a_2 \ldots a_j$
$y = a_{j+1} a_{j+2} \ldots a_k$
$z = a_{k+1} a_{k+2} \ldots a_m$.

Rögtön látszik, hogy $|xy| \leq n$ és $|y| \geq 1$. Bebizonyítjuk, hogy $xy^i z \in L$ tetszőleges i-re.
Mivel $u = xyz \in L$, létezik a

$$q_0 \xrightarrow{x} q_j \xrightarrow{y} q_k \xrightarrow{z} q_m,\ q_m \in F$$

séta, és $q_j = q_k$ miatt felírható

$$q_0 \xrightarrow{x} q_j \xrightarrow{y} q_j \xrightarrow{z} q_m,\ q_m \in F\ ,$$

alakban is. Ebből következik, hogy a $q_j \xrightarrow{y} q_j$ séta elhagyható vagy többször is beilleszthető. Tehát léteznek a következő séták:

$$q_0 \xrightarrow{x} q_j \xrightarrow{z} q_m,\ q_m \in F\ ,$$

$$q_0 \xrightarrow{x} q_j \xrightarrow{y} q_j \xrightarrow{y} \ldots \xrightarrow{y} q_j \xrightarrow{z} q_m,\ q_m \in F\ .$$

Ebből következik, hogy $xy^i z \in L$ tetszőleges i-re, és ezzel bebizonyítottuk a lemmát. □

[2] *Skatulya-elv:* Ha $k >$ 0-nál több elemet k dobozba kell elhelyeznünk, akkor legalább egy dobozba egynél több elem kerül.

2.7.2. példa. Bebizonyítjuk, hogy $L_1 = \{a^k b^k \mid k \geq 1\}$ nem reguláris. Tegyük fel, hogy L_1 reguláris, és legyen n a pumpáló lemma szerint az L_1-hez tartozó természetes szám. Mivel az $u = a^n b^n$ szó hossza $2n$, ezért ez a szó is felbontható a lemmában megadott módon. Bebizonyítjuk, hogy ez ellentmondáshoz vezet. Legyen ugyanis $u = xyz$ a felbontás. A lemma szerint ekkor $|xy| \leq n$, tehát x is és y is csak a-t tartalmazhatnak, és mivel $|y| \geq 1$, y legalább egy a-t tartalmaz. Ekkor $xy^i z$, $i \neq 1$-re, különböző számú a-t és b-t tartalmaz, tehát $xy^i z \notin L_1$ tetszőleges $i \neq 1$ értékre. Ez ellentmond a lemma állításának, tehát az a feltevésünk, hogy L_1 reguláris, hamis. Tehát $L_1 \notin \mathcal{L}_3$.

Mivel a $G_1 = (\{S\}, \{a, b\}, \{S \to ab, S \to aSb\}, S)$ környezetfüggetlen nyelvtan L_1-et generálja, így $L_1 \in \mathcal{L}_2$. E két állításból rögtön következik, hogy $\mathcal{L}_3 \subset \mathcal{L}_2$. □

2.7.3. példa. Bebizonyítjuk, hogy $L_2 = \{u \in \{0,1\}^* \mid n_0(u) = n_1(u)\}$ nem reguláris. ($n_0(u)$ az u-ban szereplő nullák, $n_1(u)$ pedig az 1-esek számát jelenti).

Az előbbi példához hasonlóan járunk el az $u = 0^n 1^n$ szóval, ahol n most a pumpáló lemmában az L_2-höz tartozó természetes szám. □

2.7.4. példa. Bebizonyítjuk, hogy $L_3 = \{uu \mid u \in \{a, b\}^*\}$ nem reguláris. Legyen $w = a^n b a^n b = xyz$, ahol n itt is a pumpáló lemma szerinti L_3-hoz tartozó természetes szám. Mivel $|xy| \leq n$, következik, hogy y csak a betűket tartalmazhat, és legalább egyet tartalmaz is. De ekkor a lemma szerint $xz \in L_3$, ami lehetetlen. Tehát L_3 nem reguláris. □

A pumpáló lemmának több érdekes következménye van.

2.7.5. következmény. *Az L reguláris nyelv akkor és csakis akkor nem üres, ha létezik $u \in L$, $|u| < n$, ahol n a pumpáló lemmában az L-hez tartozó természetes szám.*

Bizonyítás. Az állítás egyik irányba nyilvánvaló: ha létezik n-nél rövidebb szó L-ben, akkor $L \neq \emptyset$. Fordítva, legyen $L \neq \emptyset$, és legyen u a legrövidebb szó L-ben. Megmutatjuk, hogy $|u| < n$. Ha ugyanis $|u| \geq n$, akkor alkalmazzuk a pumpáló lemmát, és azt kapjuk, hogy $u = xyz$, $|y| > 1$ és $xz \in L$. Ellentmondás, mivel $|xz| < |u|$, és u a legrövidebb L-beli szó. Tehát $|u| < n$. □

2.7.6. következmény. *Létezik olyan algoritmus, amely eldönti, hogy egy reguláris nyelv üres-e.*

Bizonyítás. Tegyük fel, hogy $L = L(\mathrm{A})$, ahol $\mathrm{A} = (Q, \Sigma, E, \{q_0\}, F)$ egy determinisztikus véges automata. A 2.7.5 következmény és a 2.7.1 tétel szerint L akkor és csakis akkor nem üres, ha tartalmaz n-nél rövidebb szót, ahol n az A automata állapotainak a száma. Következésképpen, elegendő azt eldönteni, hogy van-e olyan n-nél rövidebb szó, amelyet A elfogad. Mivel az n-nél rövidebb szavak száma véges, a kérdés algoritmikusan eldönthető. □

Amikor a véges automaták elérhetetlen állapotainak meghatározására adtunk eljárást, akkor megjegyeztük, hogy az az eljárás használható annak eldöntésére is, hogy az automata által felismert nyelv üres-e. Mivel a véges automaták reguláris nyelveket ismernek fel, immár két eljárást ismerünk annak eldöntésére, hogy egy reguláris nyelv üres-e vagy sem. Sőt, van egy harmadik eljárásunk is, ha figyelembe vesszük, hogy a nem produktív állapotok kizárására szolgáló algoritmus is alkalmazható arra, hogy eldöntsük egy reguláris nyelvről, hogy üres-e vagy sem.

2.7.7. következmény. *Egy L reguláris nyelv akkor és csakis akkor végtelen, ha létezik $u \in L$ úgy, hogy $n \leq |u| < 2n$, ahol n a pumpáló lemmában az L-hez tartozó természetes szám.*

Bizonyítás. Ha L végtelen, akkor tartalmaz $2n$-nél hosszabb szót, és legyen u a legrövidebb, de $2n$-nél hosszabb L-beli szó. Mivel L reguláris, alkalmazható rá a pumpáló lemma, tehát $u = xyz$, ahol $|xy| \leq n$, tehát $|y| \leq n$ is igaz. A lemma szerint $u' = xz \in L$. Mivel $|u'| < |u|$, és a legrövidebb, de $2n$-nél hosszabb L-beli szó u, kapjuk, hogy $|u'| < 2n$. Másrészt $|y| \leq n$ miatt $|u'| \geq n$ is teljesül.

Fordítva, ha létezik $u \in L$ úgy, hogy $n \leq |u| < 2n$, akkor alkalmazva rá a pumpáló lemmát, következik, hogy $u = xyz$, $|y| \geq 1$ és $xy^iz \in L$ tetszőleges i-re, tehát L végtelen. □

Feltehetjük a kérdést, hogy alkalmazhatjuk-e a pumpáló lemmát egy véges nyelvre, hisz a pumpálással végtelen sok szót kapunk? A válasz abban rejlik, hogy egy véges L nyelvet felismerő bármely véges automata állapotainak száma nagyobb, mint L leghosszabb szavának a hossza. Ezért L-ben egyetlen szó sincs, amelynek a hossza legalább n, ahol n a pumpáló lemmában az L nyelvhez tartozó természetes szám. Tehát egyetlen L-beli szó sem bontható fel xyz alakban, ahol $|xyz| \geq n$, $|xy| \leq n$, $|y| \geq 1$, és ezért nem kaphatunk végtelen sok további L-beli szót.

2.8. Reguláris kifejezések

A következőkben tetszőleges Σ ábécé esetén bevezetjük a Σ feletti reguláris kifejezés és az általa jelölt nyelv fogalmát. A reguláris kifejezés egy formula, míg az általa jelölt nyelv egy Σ feletti nyelv lesz. Például, ha $\Sigma = \{a, b\}$, akkor az a^*, b^*, $a^* + b^*$ kifejezések Σ feletti reguláris kifejezések lesznek, amelyek rendre az $\{a\}^*$, $\{b\}^*$, $\{a\}^* \cup \{b\}^*$ nyelveket jelölik. A pontos definíció a következő.

$$
\begin{aligned}
x+y &\equiv y+x \\
(x+y)+z &\equiv x+(y+z) \\
(xy)z &\equiv x(yz) \\
(x+y)z &\equiv xz+yz \\
x(y+z) &\equiv xy+xz \\
(x+y)^* \equiv (x^*+y)^* &\equiv (x+y^*)^* \equiv (x^*+y^*)^* \\
(x+y)^* &\equiv (x^*y^*)^* \\
(x^*)^* &\equiv x^* \\
x^*x &\equiv xx^* \\
xx^*+\varepsilon &\equiv x^*
\end{aligned}
$$

2.1. táblázat. Reguláris kifejezések tulajdonságai.

2.8.1. értelmezés. *Rekurzívan értelmezzük a Σ feletti reguláris kifejezés és az általa jelölt nyelv fogalmát.*

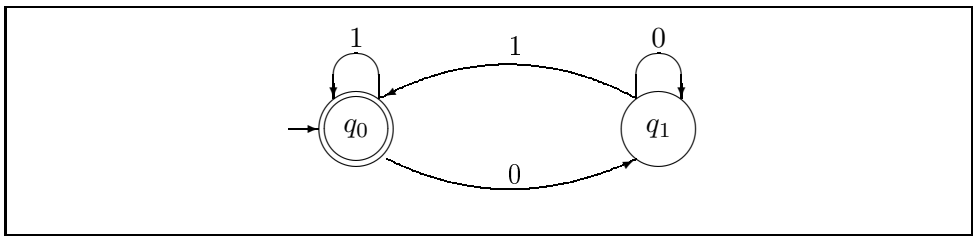
- *$\emptyset$ reguláris kifejezés és az üres nyelvet jelöli.*
- *ε reguláris kifejezés és az $\{\varepsilon\}$ nyelvet jelöli.*
- *Ha $a \in \Sigma$, a reguláris kifejezés és az $\{a\}$ nyelvet jelöli.*
- *Ha x, y reguláris kifejezések és az X, illetve Y nyelveket jelölik, akkor $(x+y)$, (xy), (x^*) is reguláris kifejezések és rendre az $X \cup Y$, XY és X^* nyelveket jelölik.*

Csak azok Σ feletti reguláris kifejezések, amelyeket a fenti szabályok véges sokszori alkalmazásával kapunk.

Egy reguláris kifejezésben bizonyos zárójeleket elhagyhatunk, amennyiben figyelembe véve a műveletek prioritási sorrendjét (iteráció, szorzat, egyesítés), nem változtatjuk meg az általa jelölt nyelvet. Például $((x^*)(x+y))$ helyett $x^*(x+y)$-t is írhatunk.

Két reguláris kifejezés ***ekvivalens,*** ha ugyanazt a nyelvet jelölik, azaz $x \equiv y$, ha $X = Y$, ahol X és Y rendre az x és y reguláris kifejezések által jelölt nyelvek. A 2.1. táblázatban néhány példát mutatunk ekvivalens reguláris kifejezésekre.

Megmutatjuk, hogy minden véges L nyelvhez megadható olyan x reguláris kifejezés, amely L-et jelöli. Ha $L = \emptyset$, akkor $x = \emptyset$. Ha $L = \{w_1, w_2, \ldots, w_n\}$, akkor $x = x_1 + x_2 + \ldots + x_n$, ahol minden $i = 1, 2, \ldots, n$ esetében x_i a $\{w_i\}$ nyelvet jelölő reguláris kifejezés. Ez utóbbit pedig a következőképpen adjuk meg. Ha $w_i = \varepsilon$, akkor $x_i = \varepsilon$. Különben, ha $w_i = a_1 a_2 \ldots a_m$, ahol $m \geq 1$ függ i-től, akkor az $x_i = a_1 a_2 \ldots a_m$, ahol elhagytuk a zárójeleket.



2.18. ábra. A 2.8.3. példában szereplő véges automata, amelyhez reguláris kifejezést rendelünk az 1. módszer alapján.

A következőkben bebizonyítjuk Kleene[3] tételét, amely kapcsolatot teremt a reguláris nyelvek és a reguláris kifejezések között.

2.8.2. tétel. (Kleene tétele) *Az $L \subseteq \Sigma^*$ nyelv pontosan akkor reguláris, ha van olyan Σ feletti reguláris kifejezés, amely éppen L-et jelöli.*

Bizonyítás. Először bebizonyítjuk, hogy ha x reguláris kifejezés, akkor az L nyelv, amelyet x jelöl, szintén reguláris. A bizonyítást a reguláris kifejezés felépítése szerinti indukcióval végezzük.

Ha $x = \emptyset$, $x = \varepsilon$, $x = a, \forall a \in \Sigma$, akkor $L = \emptyset$, $L = \{\varepsilon\}$, $L = \{a\}$. Mivel L mindhárom esetben véges, ezért reguláris.

Ha $x = (x_1 + x_2)$, akkor $L = L_1 \cup L_2$, ahol L_1 és L_2 rendre az x_1 és x_2 reguláris kifejezések által jelölt nyelvek. Az indukciós feltevésünk értelmében L_1 és L_2 reguláris nyelvek, így L is az, mivel a reguláris nyelvek osztálya zárt az egyesítésre. Az $x = (x_1 x_2)$ és $x = (x_1^*)$ esetek bizonyítása hasonló.

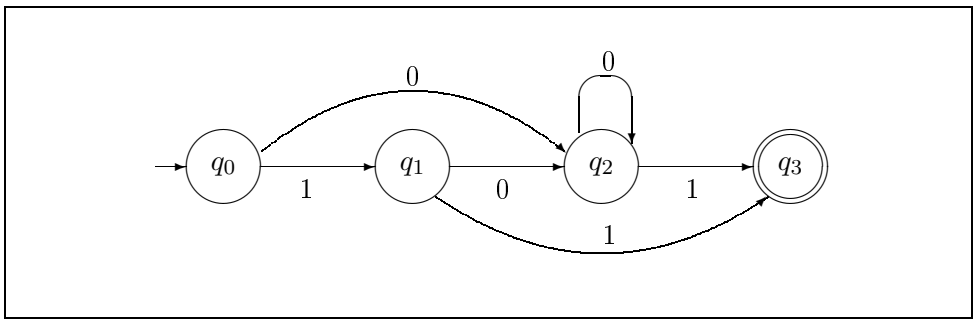
Fordítva, bebizonyítjuk, hogy ha L reguláris nyelv, akkor hozzárendelhető egy x reguláris kifejezés, amely éppen az L nyelvet jelöli. Ha L reguláris, akkor létezik egy $\mathrm{A} = (Q, \Sigma, E, \{q_0\}, F)$ determinisztikus véges automata, amelyre $L = L(\mathrm{A})$. Legyenek A állapotai $q_0, q_1, \ldots, q_n$. Értelmezzük az R_{ij}^k nyelveket, minden $-1 \leq k \leq n$ és $0 \leq i, j \leq n$ értékekre. R_{ij}^k azon szavak halmaza, amelyek hatására az A véges automata a q_i állapotból a q_j állapotba kerül úgy, hogy közben nem használja a k-nál nagyobb indexű állapotokat. Az átmenetgráfot tekintve, egy szó akkor és csakis akkor tartozik R_{ij}^k-ba, ha a q_i állapotból indulva élek mentén eljuthatunk a q_j állapotba úgy, hogy az élek címkéit összeolvasva éppen a szót kapjuk, és közben nem érintjük a $q_{k+1}, \ldots q_n$ állapot egyikét sem. Az R_{ij}^k halmazokat formálisan is leírhatjuk:

$R_{ij}^{-1} = \{a \in \Sigma \mid (q_i, a, q_j) \in E\}$, ha $i \neq j$,

$R_{ii}^{-1} = \{a \in \Sigma \mid (q_i, a, q_i) \in E\} \cup \{\varepsilon\}$,

$R_{ij}^k = R_{ij}^{k-1} \cup R_{ik}^{k-1} \left(R_{kk}^{k-1}\right)^* R_{kj}^{k-1}$ minden $i, j, k \in \{0, 1, \ldots, n\}$ értékre.

[3] Stephen Cole Kleene (1909–1994), amerikai matematikus. Nevének helyes kiejtése „kléni".



2.19. ábra.A 2.8.4. példában szereplő véges automata, amelyhez reguláris kifejezést rendelünk az 1. módszer alapján. A számításokat a 2.2. táblázat tartalmazza.

Indukcióval be lehet bizonyítani, hogy az R_{ij}^k halmazok leírhatók reguláris kifejezésekkel. Valóban, ha $k = -1$, akkor minden i-re és j-re az R_{ij}^k nyelv véges, és ezért megadható olyan reguláris kifejezés, amely ezt a véges nyelvet jelöli. Továbbá, ha minden i-re és j-re az R_{ij}^{k-1} nyelv jelölhető reguláris kifejezéssel, akkor az R_{ij}^k nyelv is jelölhető reguláris kifejezéssel, amelyet az R_{ij}^{k-1}, R_{ik}^{k-1}, R_{kk}^{k-1} és R_{kj}^{k-1} nyelveket jelölő reguláris kifejezésekből építünk fel alkalmas módon, az R_{ij}^k fentebb definiált képlete alapján.

Végül, ha $F = \{q_{i_1}, q_{i_2}, \ldots, q_{i_p}\}$ az A véges automata végállapotainak halmaza, akkor $L = L(\mathrm{A}) = R_{0i_1}^n \cup R_{0i_2}^n \cup \ldots \cup R_{0i_p}^n$ is megadható reguláris kifejezéssel az $R_{0i_1}^n, R_{0i_2}^n, \ldots, R_{0i_p}^n$ nyelveket jelölő reguláris kifejezésekből a $+$ művelet segítségével. □

A következőkben eljárásokat adunk meg, amelyek segítségével tetszőleges reguláris kifejezéshez megadhatjuk a megfelelő véges automatát, és fordítva.

2.8.1. Reguláris kifejezés hozzárendelése véges automatához

Három módszert mutatunk be, amelyek mindegyike tetszőleges véges automatához hozzárendeli a megfelelő reguláris kifejezést.

1. módszer. Felhasználjuk Kleene tételének az eredményét, azaz megkonstruáljuk az R_{ij}^k halmazokat, és felírjuk az $L = R_{0i_1}^n \cup R_{0i_2}^n \cup \ldots \cup R_{0i_p}^n$ nyelvet jelölő reguláris kifejezést, ahol $F = \{q_{i_1}, q_{i_2}, \ldots, q_{i_p}\}$ az automata végállapotainak halmaza.

2.8.3. példa. Tekintsük a 2.18. ábrán látható véges automatát.

$L(\mathrm{A}) = R_{00}^1 = R_{00}^0 \cup R_{01}^0 \left(R_{11}^0\right)^* R_{10}^0$

R_{00}^0 : $1^* + \varepsilon \equiv 1^*$

R_{01}^0 : 1^*0

R_{11}^0 : $11^*0 + \varepsilon + 0 \equiv (11^* + \varepsilon)0 + \varepsilon \equiv 1^*0 + \varepsilon$

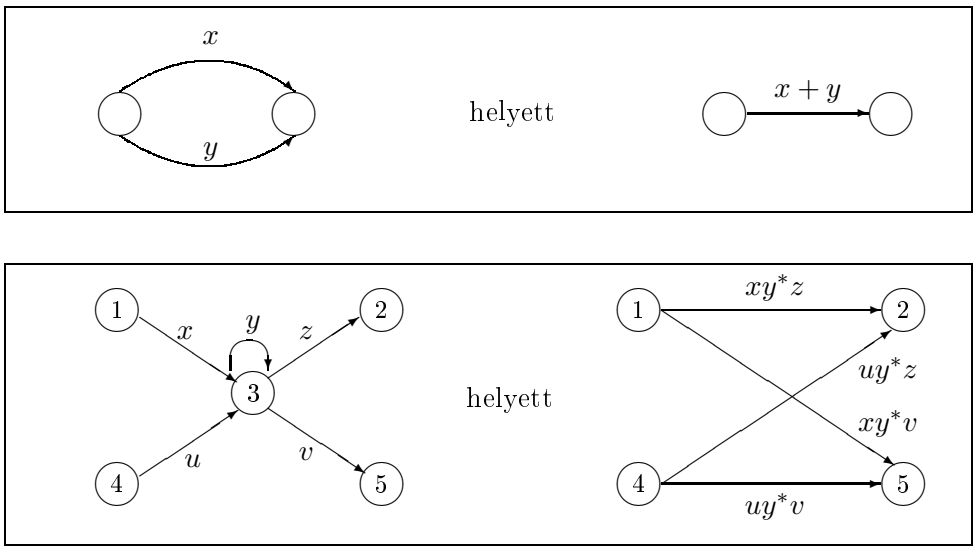
R_{10}^0 : 11^*

	$k=-1$	$k=0$	$k=1$	$k=2$	$k=3$
R_{00}^k	ε	ε	ε	ε	
R_{01}^k	1	1	1	1	
R_{02}^k	0	0	$0+10$	$(0+10)0^*$	
R_{03}^k	$\emptyset$	$\emptyset$	11	$11+(0+10)0^*1$	$11+(0+10)0^*1$
R_{11}^k	ε	ε	ε	ε	
R_{12}^k	0	0	0	00^*	
R_{13}^k	1	1	1	$1+00^*1$	
R_{22}^k	$0+\varepsilon$	$0+\varepsilon$	$0+\varepsilon$	0^*	
R_{23}^k	1	1	1	0^*1	
R_{33}^k	ε	ε	ε	ε	

2.2. táblázat.A 2.19. ábra véges automatájához rendelt reguláris kifejezés meghatározása az R_{ij}^k halmazok segítségével.

Ekkor az L(A)-nak megfelelő reguláris kifejezés: $1^* + 1^*0(1^*0+\varepsilon)^*11^* \equiv 1^* + 1^*0(1^*0)^*11^*$. □

2.8.4. példa. Keressük meg a 2.19. ábrán lévő véges automatához rendelt reguláris kifejezést. A számításokat a 2.2. táblázat tartalmazza. Az R_{03}^3-nak megfelelő reguláris kifejezés: $11+(0+10)0^*1$. □



2.20. ábra. Lehetséges ekvivalens átalakítások véges automatához rendelhető reguláris kifejezés meghatározására.

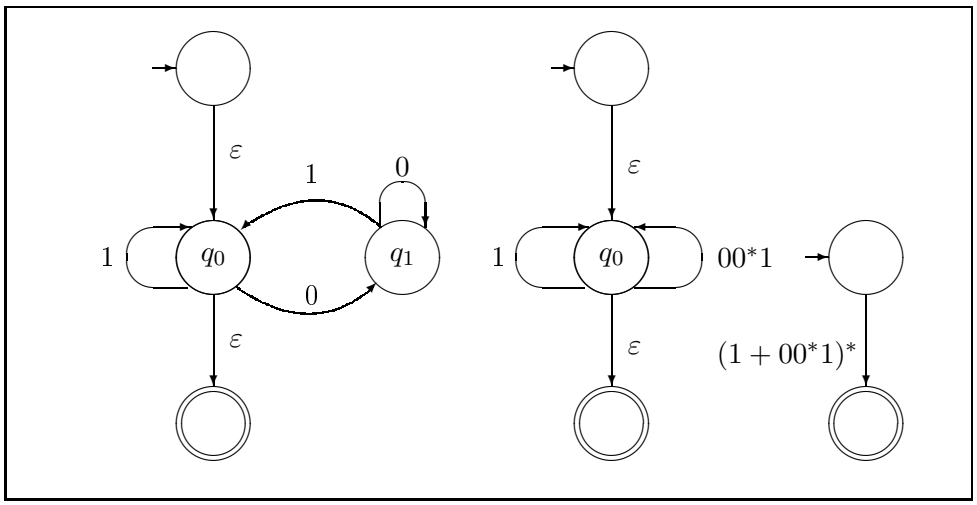
2. módszer. A véges automata fogalmát általánosítjuk úgy, hogy az automata gráfjának éleit nem betűkkel, hanem reguláris kifejezésekkel címkézzük meg. Egy ilyen automatában minden séta meghatároz egy reguláris kifejezést, amely meghatároz egy reguláris nyelvet. Az általánosított véges automata által felismert nyelven a produktív séták által meghatározott reguláris nyelvek egyesítését értjük. Könnyen belátható, hogy az ilyen általánosított véges automatákkal is éppen a reguláris nyelvek ismerhetők fel.

Ugyanakkor az általánosított véges automaták előnye, hogy rajtuk ekvivalens átalakításokat végezhetünk, amelyek csökkentik az automata gráfja éleinek a számát, ugyanakkor nem változtatják meg az automata által felismert nyelvet. Végül elérhetjük, hogy az általánosított véges automata gráfjának egyetlen éle legyen, amelynek címkéje éppen az eredeti automata által felismert nyelvet jelölő reguláris kifejezés.

Az ekvivalens átalakítások a 2.20. ábrán láthatók. Amennyiben az ábrán látható 1, 2, 4, 5 csúcsok közül bármelyik kettő egybeesik, a végeredményben ezeket összevonjuk, így hurokél is megjelenik.

Először átalakítjuk a véges automatát megfelelő ε-átmenetek segítségével úgy, hogy egyetlen kezdő- és végállapota legyen. Ezután addig alkalmazzuk rá az ekvivalens átalakításokat, amíg gráfja egyetlen élt tartalmaz, amelynek címkéje az eredeti véges automata által felismert nyelvet jelölő reguláris kifejezés.

2.8.5. példa. A 2.18. ábra esetében a 2.21. ábrán látható lépesekkel jutunk el az



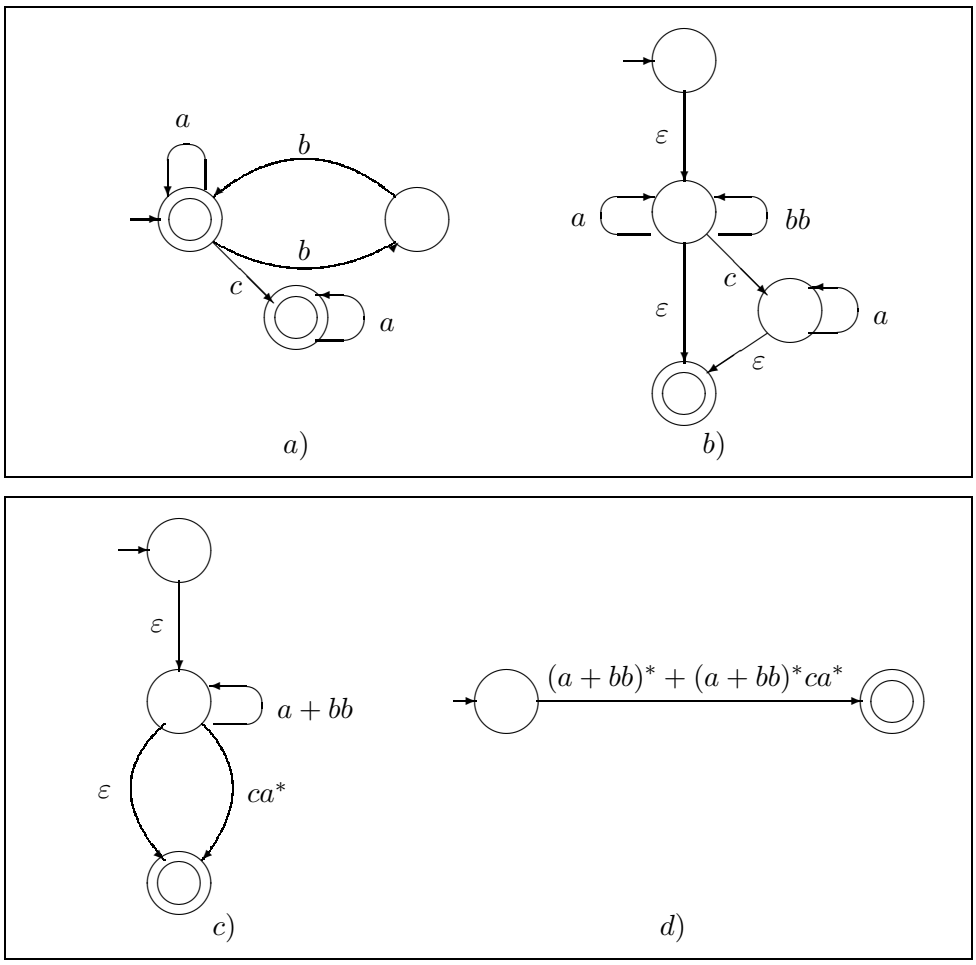
2.21. ábra. A 2.18. ábrán lévő véges automata átalakítása.

eredményhez. Tehát az eredmény $(1+00^*1)^*$, amely, habár más alakú, de ugyanazt a nyelvet jelenti, mint az előbbi módszerrel kapott kifejezés (2.8.3. példa) (*lásd* a 2-11. gyakorlatot). A 2.22. ábrán egy másik átalakítás látható. □

2.8.6. példa. A 2.19. ábra esetében nem szükséges új kezdő- és végállapotot bevezetni. Az átalakítás lépései a 2.23. ábrán láthatók. A kapott reguláris kifejezés még így is írható: $(0+10)0^*1+11$, amely azonos az előbbi módszerrel kapott kifejezéssel. □

3. módszer. Egy másik módszer a reguláris kifejezés felírására a formális egyenletek módszere. Minden állapothoz hozzárendelünk egy X változót (különböző állapotokhoz különbözőt) és egy egyenletet, amelynek bal oldalán X, jobb oldalán pedig Ya alakú kifejezések összege vagy ε állhatnak, ahol Y is egy állapothoz rendelt változó, a pedig egy bemeneti szimbólum. Ha az X változónak megfelelő állapotba nem vezet él, akkor az X baloldalú egyenlet jobb oldalán ε szerepel, különben az összes olyan Ya alakú tag összege, amelyekre teljesül, hogy a véges automata gráfjában az Y változónak megfelelő állapotból egy a-val címkézett él vezet az X változónak megfelelő állapotba. Amennyiben az X változónak megfelelő állapot kezdőállapot és egyben végállapot is, akkor az X baloldalú egyenlet jobb oldalán megjelenik egy ε tag is. Például a 2.19. ábra esetében legyenek ezek a változók X, Y, Z, U, amelyek a q_0, q_1, q_2, q_3 állapotoknak felelnek meg. A megfelelő egyenletek a következők:

$X = \varepsilon$
$Y = X1$
$Z = X0 + Y0 + Z0$
$U = Y1 + Z1.$



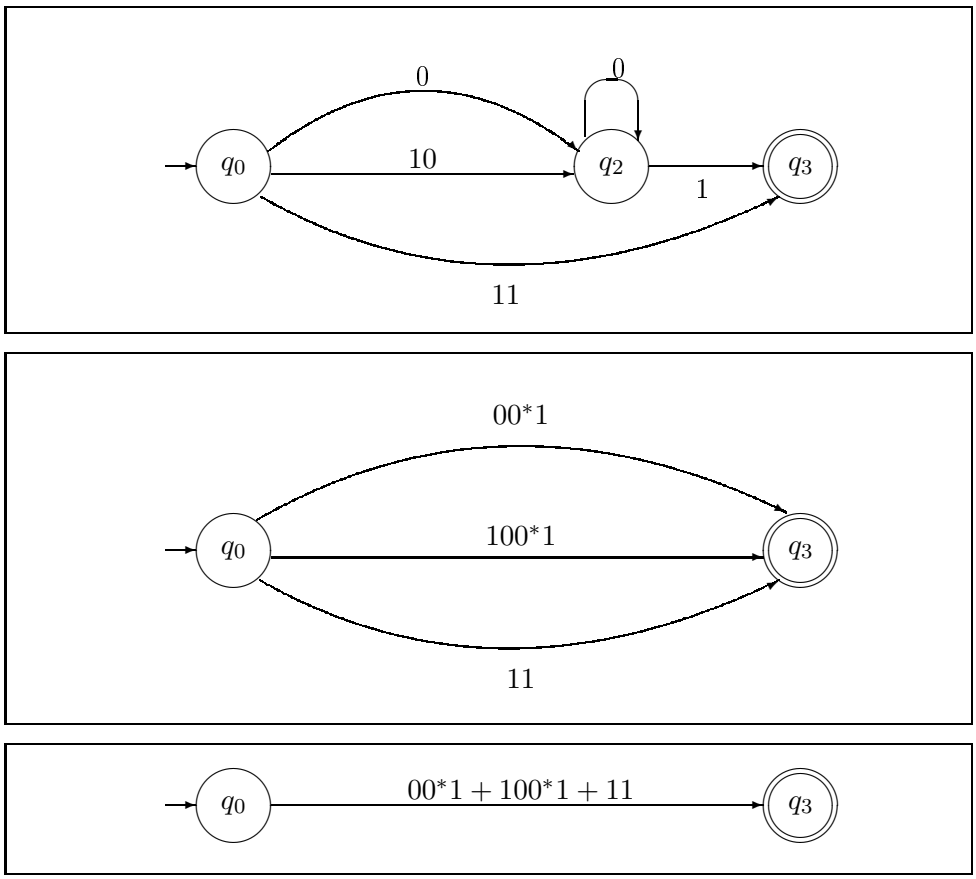
2.22. ábra. Reguláris kifejezés hozzárendelése véges automatához.

Ha egy egyenlet $X = X\alpha + \beta$ alakú, ahol α, β tetszőleges szavak, amelyek nem tartalmazzák az X változót, akkor könnyű ellenőrizni, egyszerű behelyettesítéssel, hogy $X = \beta\alpha^*$ megoldása az egyenletnek.

Mivel az egyenletek lineárisak a változókban, minden egyenlet felírható $X = X\alpha + \beta$ vagy $X = X\alpha$ alakban, ahol α nem tartalmaz egyetlen változót sem. Ezt behelyettesítve a többi egyenletbe, eggyel csökkentjük azok számát. Így a rendszer, megfelelő helyettesítésekkel, megoldható minden változóra.

Az egyenletrendszert megoldva a végállapotoknak megfelelő változók adják a megoldást jelentő reguláris kifejezést úgy, hogy összeadjuk az ezen változóknak megfelelő reguláris kifejezéseket.

Példánkban, a fenti egyenletrendszer első egyenletének segítségével azt kap-



2.23. ábra. A 2.8.6. példa lépései.

juk, hogy $Y = 1$. Innen $Z = 0 + 10 + Z0$, azaz $Z = Z0 + (0 + 10)$, és ezt megoldva azt kapjuk, hogy $Z = (0 + 10)0^*$. Innen pedig U egyszerűen megkapható: $U = 11 + (0 + 10)0^*1$.

Ezt a módszert alkalmazva a 2.18. ábra esetében, a következő egyenletekhez jutunk:

$X = \varepsilon + X1 + Y1$

$Y = X0 + Y0$

Kiemelés után:

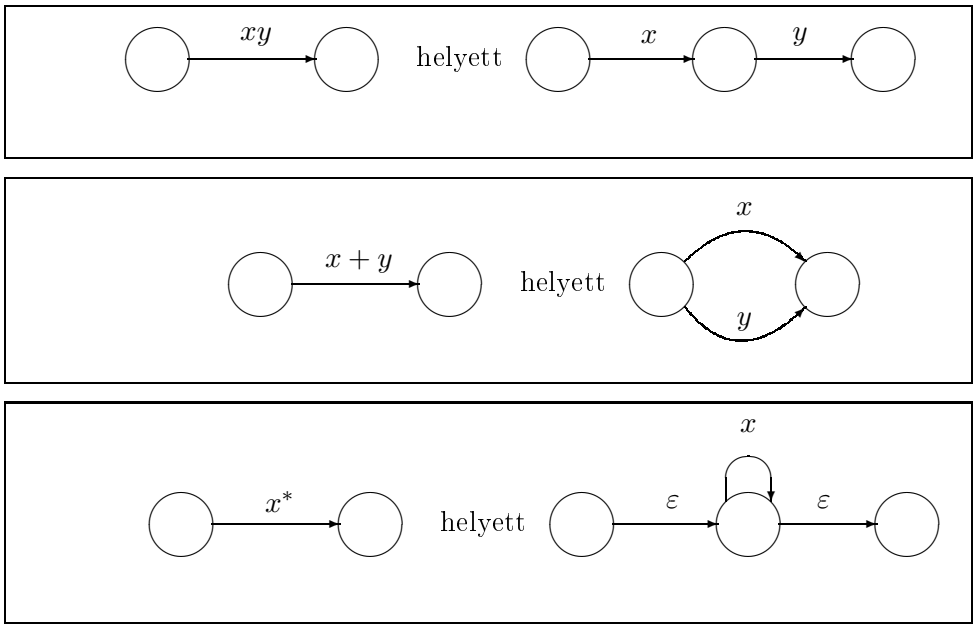
$X = \varepsilon + (X + Y)1$

$Y = (X + Y)0$.

A két egyenletet összeadva a következő egyenlethez jutunk:

$X+Y = \varepsilon+(X+Y)(0+1)$, ahonnan ($\varepsilon$-t β-nak, $(0+1)$-et α-nak tekintve) eredményül kapjuk a következőt:

$X + Y = (0 + 1)^*$.



2.24. ábra. Lehetséges átalakítások reguláris kifejezéshez rendelt véges automata meghatározásához.

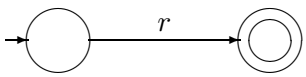
Innen – behelyettesítés után – megkapjuk X értékét:

$X = \varepsilon + (0+1)^*1$,

amely ekvivalens a másik módszerrel kapott kifejezéssel (*lásd* a 2-11. gyakorlatot).

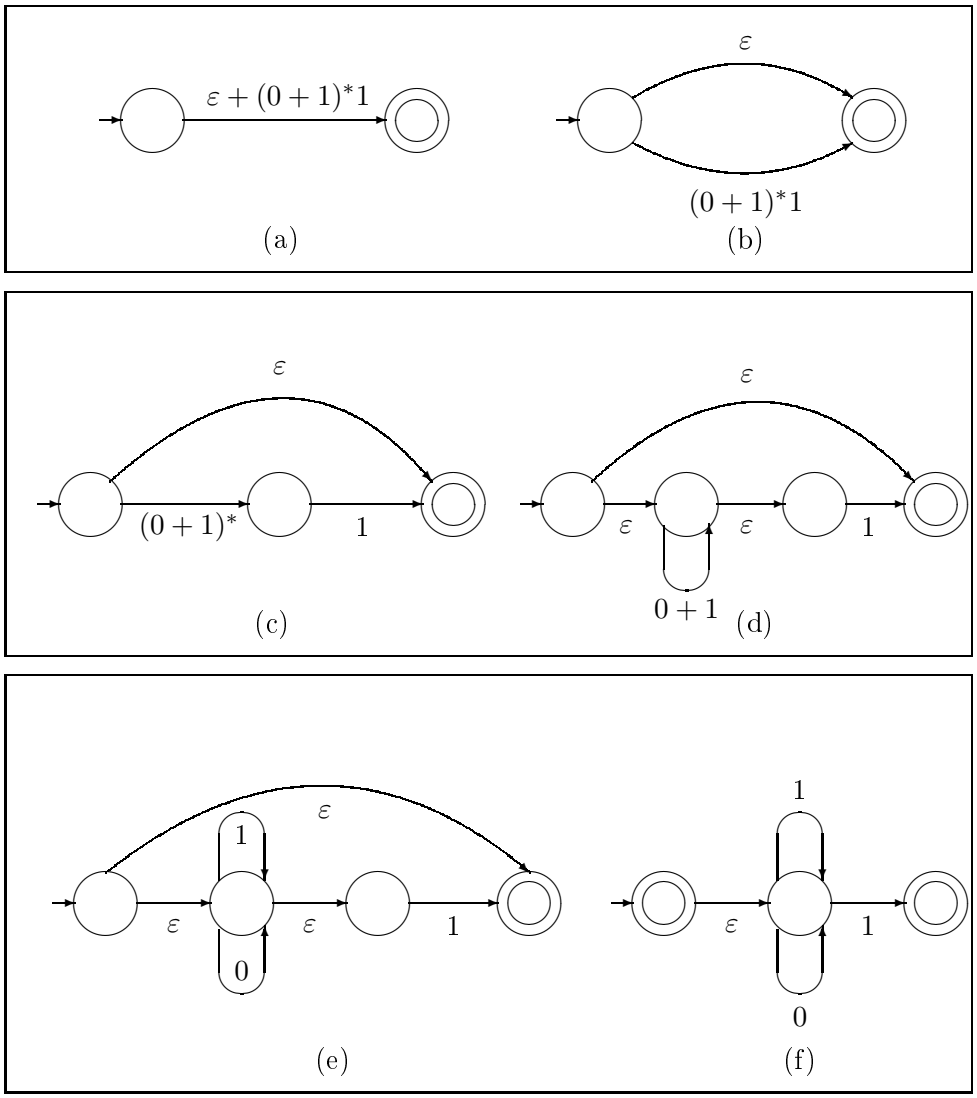
2.8.2. Véges automata hozzárendelése reguláris kifejezéshez

A r reguláris kifejezéshez hozzárendelünk egy általánosított véges automatát:

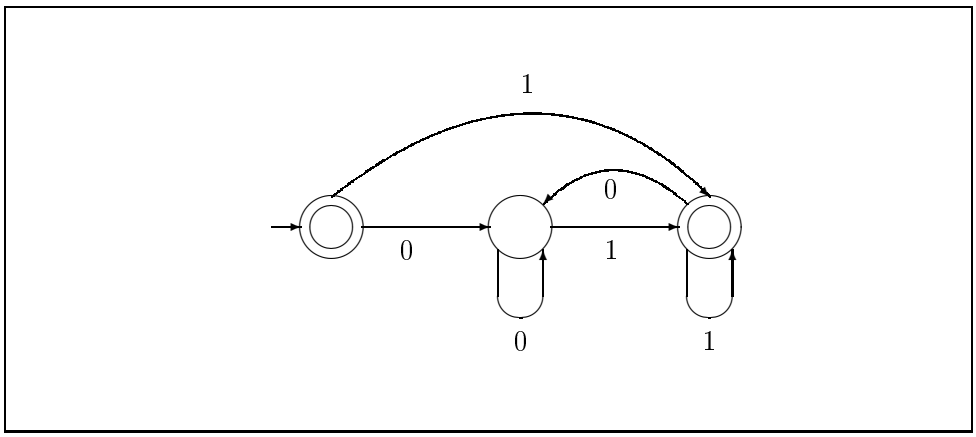


Azután lépésenként alkalmazzuk a 2.24. ábrán látható átalakításokat mindaddig, amíg a véges automata élei Σ elemeivel vagy ε-nal lesznek címkézve.

2.8.7. példa. Induljunk el az $\varepsilon + (0+1)^*1$ reguláris kifejezésből. Az átalakítás lépései a 2.25.(a)-(e) ábrákon láthatók. Az utolsó véges automata (a 2.25.(e) ábrán) egyszerűbb alakban is megadható, ez a 2.25.(f) ábrán látható. Ha ebből kiküszöböljük a ε-lépést, átalakítjuk determinisztikussá, akkor a 2.26. ábrán látható véges automatát kapjuk eredményül, amelyről be lehet bizonyítani, hogy ekvivalens a 2.18. ábrán látható véges automatával. □



2.25. ábra. Véges automata hozzárendelése az $\varepsilon + (0+1)^*1$ reguláris kifejezéshez.



2.26. ábra. Az $\varepsilon + (0+1)^*1$ kifejezéshez rendelt véges automata.

Gyakorlatok

2-1. Adjunk meg egy determinisztikus véges automatát, amely a 9-cel osztható természetes számokat ismeri fel.

2-2. Adjunk meg egy-egy determinisztikus véges automatát, amely

a. a páros számú 0-t és páros számú 1-et tartalmazó szavakból álló nyelvet ismeri fel,

b. a páros számú 0-t és páratlan számú 1-et tartalmazó szavakból álló nyelvet ismeri fel,

c. a páratlan számú 0-t és páros számú 1-et tartalmazó szavakból álló nyelvet ismeri fel,

d. a páratlan számú 0-t és páratlan számú 1-et tartalmazó szavakból álló nyelvet ismeri fel.

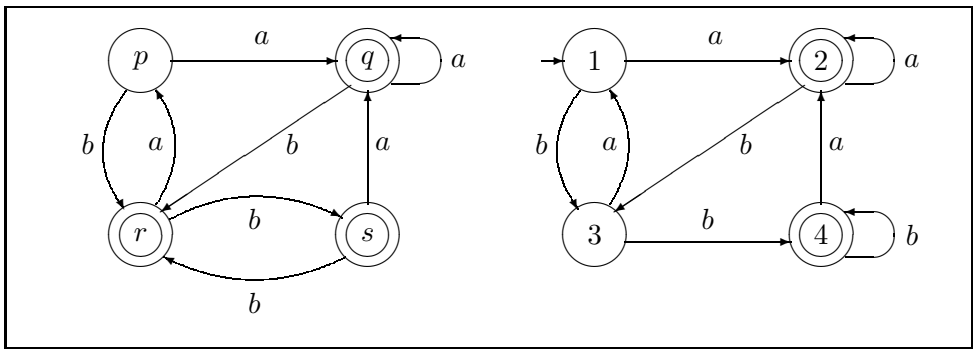
2-3. Adjunk meg egy-egy determinisztikus véges automatát a következő nyelvek felismerésére.

$L_1 = \{a^n b^m \mid n \geq 1, m \geq 0\}$, $\quad L_2 = \{a^n b^m \mid n \geq 1, m \geq 1\}$,
$L_3 = \{a^n b^m \mid n \geq 0, m \geq 0\}$, $\quad L_4 = \{a^n b^m \mid n \geq 0, m \geq 1\}$.

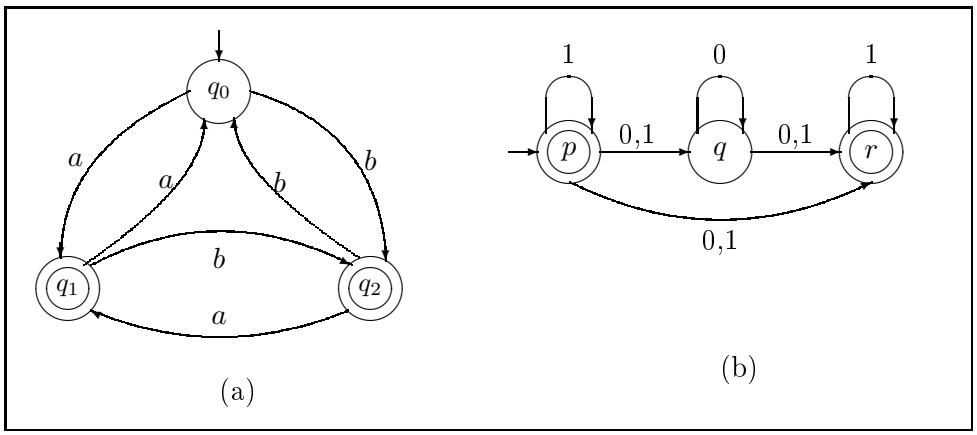
2-4. Adjunk meg egy nemdeterminisztikus véges automatát, amely a legalább két 0-át és tetszőleges számú 1-et tartalmazó szavakat ismeri fel. Adjunk meg egy vele ekvivalens determinisztikus véges automatát.

2-5. Minimalizáljuk a 2.27. ábrán lévő véges automatákat.

2-6. Mutassuk meg, hogy a 2.28.(a) ábrán látható véges automata minimális.



2.27. ábra. Minimalizálandó véges automaták a 2-5. gyakorlathoz.



2.28. ábra. Véges automaták a 2-6. és 2-7. gyakorlatokhoz.

2-7. Alakítsuk át a 2.28.(b) ábrán látható véges automatát determinisztikussá, majd minimalizáljuk.

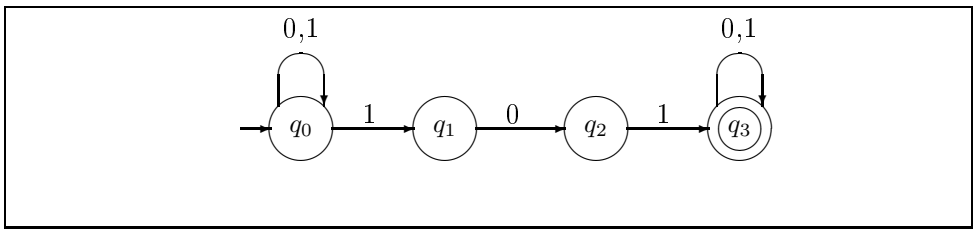
2-8. Értelmezzük az A_1 véges automatát, amely felismeri a $0(10)^n$ alakú szavakat ($n \geq 0$), az A_2-t, amely pedig az $1(01)^n$ ($n \geq 0$) alakúakat. Adjuk meg az $A_1 \cup A_2$ egyesített véges automatát, majd küszöböljük ki az ε-lépéseket.

2-9. Rendeljünk a 2.29. ábrán látható véges automatához egy reguláris kifejezést.

2-10. Rendeljünk az $ab^*ba^* + b + ba^*a$ reguláris kifejezéshez egy véges automatát.

2-11. Rendeljünk a következő reguláris kifejezések mindegyikéhez egy ε-lépéses véges automatát:

$$1^* + 1^*0(1^*0)^*11^*, \quad (1 + 00^*1)^*, \quad \varepsilon + (0 + 1)^*1.$$



2.29. ábra. Véges automata a 2-9. gyakorlathoz.

Az így kapott automatákhoz rendeljünk egy-egy ekvivalens nemdeterminisztikus véges automatát, majd pedig egy-egy ekvivalens determinisztikus véges automatát. Mutassuk meg, hogy ezek a determinisztikus véges automaták ekvivalensek egymással.

2-12. A pumpáló lemmát felhasználva bizonyítsuk be, hogy a következő nyelvek egyike sem reguláris:
$L_1 = \{a^n c b^n \mid n \geq 0\}$, $L_2 = \{a^n b^n a^n \mid n \geq 0\}$, $L_3 = \{a^p \mid p \text{ prím}\}$.

2-13. Bizonyítsuk be, hogy ha L reguláris nyelv, akkor az $\{u^{-1} \mid u \in L\}$ nyelv is reguláris.

2-14. Bizonyítsuk be, hogy ha $L \subseteq \Sigma^*$ reguláris nyelv, akkor regulárisak a következő nyelvek is:
$\text{pre}(L) = \{w \in \Sigma^* \mid \exists u \in \Sigma^*, wu \in L\}$,
$\text{suf}(L) = \{w \in \Sigma^* \mid \exists u \in \Sigma^*, uw \in L\}$.

2-15. Mutassuk meg, megfelelő reguláris nyelvtanok megadásával, hogy az alábbi nyelvek mind regulárisak.
$L_1 = \{ab^n cd^m \mid n > 0, m > 0\}$,
$L_2 = \{(ab)^n \mid n \geq 0\}$,
$L_3 = \{a^{kn} \mid n \geq 0,\ \ k \text{ állandó}\}$.

3. FEJEZET

Veremautomaták és környezetfüggetlen nyelvek

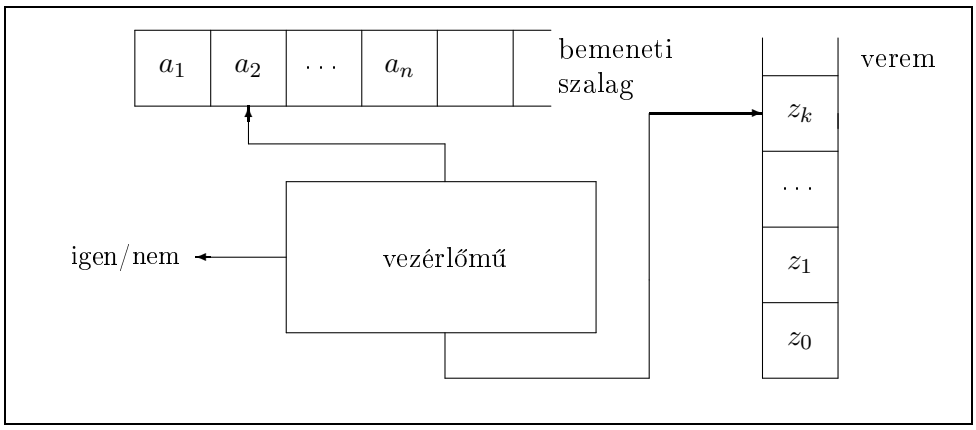
Ebben az alfejezetben veremautomatákkal és az általuk felismert nyelvek osztályával, a környezetfüggetlen nyelvekkel foglalkozunk. 0 Amint azt az 1. fejezetben láttuk, egy környezetfüggetlen nyelvtan olyan $G = (N, T, P, S)$ nyelvtan, amelynek szabályai $A \to \beta$ alakúak, $A \in N$, $\beta \in (N \cup T)^+$. Megengedhető az $S \to \varepsilon$ szabály is, amennyiben S nem szerepel egyetlen szabály jobb oldalán sem. Az $L(G) = \{u \in T \mid S \overset{*}{\underset{G}{\Longrightarrow}} u\}$ nyelv a G nyelvtan által generált környezetfüggetlen nyelv.

3.1. Veremautomaták

Láttuk, hogy a véges automaták a reguláris nyelvek osztályát ismerik fel. A következőkben olyan automatákkal, az ún. ***veremautomatákkal*** ismerkedünk meg, amelyek környezetfüggetlen nyelveket ismernek fel. A veremautomaták lényegében abban különböznek a véges automatáktól, hogy egyrészt akkor is válthatnak állapotot, ha nem lépnek tovább a bemeneti szóban (üres jelet olvasnak), másrészt rendelkeznek egy veremmemóriával. A veremben az ún. veremábécé jeleit tároljuk (3.1. ábra).

A veremautomata egy szót kap bemenetként, a kezdőállapotból indul ki, és a veremmemóriájában (a továbbiakban csak veremben) egy speciális szimbólum, a verem kezdőszimbóluma áll. A működése során az aktuális állapot, a következő bemeneti szimbólum (amely üres szó is lehet), és a verem tetején levő szimbólum ismeretében állapotot vált, és a verem tetején levő szimbólum helyére egy szót ír be (amely szintén lehet üres is).

Kétféle felismerési mód lehetséges. Végállapottal való felismerésről beszélünk, ha a veremautomata a bemeneti szó elolvasása után végállapotba kerül. Üres veremmel való felismerésről beszélünk, ha a bemeneti szó elolvasásának pillanatában a verem üres legyen. Meg fogjuk mutatni, hogy a két felismerési mód ekvivalens egymással.



3.1. ábra. Veremautomata.

3.1.1. értelmezés. ***Nemdeterminisztikus veremautomatának*** *nevezzük a*

$$\mathrm{V} = (Q, \Sigma, W, E, q_0, z_0, F)$$

rendezett hetest, ahol

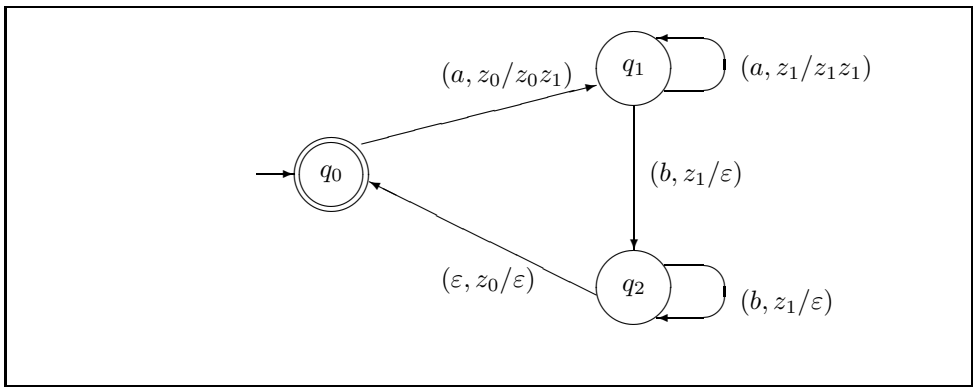
- Q *az* ***állapotok*** *véges, nem üres halmaza,*
- Σ *a* ***bemeneti ábécé,***
- W *a* ***veremábécé,***
- $E \subseteq Q \times \big(\Sigma \cup \{\varepsilon\}\big) \times W \times W^* \times Q$ *az* ***átmenetek*** *vagy* ***élek*** *halmaza,*
- $q_0 \in Q$ *a* ***kezdőállapot,***
- $z_0 \in W$ *a* ***veremmemória kezdőjele,***
- $F \subseteq Q$ *a* ***végállapotok*** *halmaza.*

Egy (p, a, z, w, q) átmenet azt jelenti, hogy ha a V veremautomata a p állapotban van, a bemeneti szalagról az a jelet olvassa (amely most üres szó is lehet), és a verem tetején levő szimbólum z, akkor q állapotba megy át és verembe a z helyére a w szót írja (betűnként). A w szó beírása a verembe követi a természetes sorrendet, azaz w betűi balról jobbra, sorrendben kerülnek a verembe (azaz a szó utolsó betűje lesz a verem tetején). A könnyebb olvashatóság kedvéért a (p, a, z, w, q) átmenet helyett a $\big(p, (a, z/w), q\big)$ jelölést használjuk, amely utal arra, hogy az a bemeneti jel elolvasása hatására a veremben kicseréljük z-t w-re.

Akárcsak a véges automaták esetében, most is értelmezhetünk egy átmenetfüggvényt a következőképpen:

$$\delta \ : \ Q \times (\Sigma \cup \{\varepsilon\}) \times W \to \mathcal{P}(W^* \times Q) \ ,$$

amely az aktuális állapothoz, bemeneti jelhez és verem tetején lévő elemhez hozzárendel (w, q) párokat, ahol $w \in W^*$ a verembe írt szó, $q \in Q$ pedig az



3.2. ábra. Példa veremautomatára.

új állapot. Mivel a veremautomata nemdeterminisztikus, az átmenetfüggvény esetében

$\delta(q, a, z) = \{(w_1, p_1), \ldots, (w_k, p_k)\}$ (ha az automata a bemeneti szalagról olvas egy jelet, majd az olvasófej továbblép), vagy

$\delta(q, \varepsilon, z) = \{(w_1, p_1), \ldots, (w_k, p_k)\}$ (ha nem mozdul el az olvasófej a bemeneti szalagon).

A veremautomata ***determinisztikus,*** ha $\forall q \in Q$ és $\forall z \in W$ esetében

- $|\delta(q, a, z)| \leq 1$, $\forall a \in \Sigma \cup \{\varepsilon\}$
- Ha $\delta(q, \varepsilon, z) \neq \emptyset$, akkor $\delta(q, a, z) = \emptyset$, $\forall a \in \Sigma$.

A veremautomatához mindig megadható egy átmenettáblázat, akárcsak a véges automaták esetében. A táblázat sorai Q elemeivel vannak indexelve, oszlopai pedig a $\Sigma \cup \{\varepsilon\}$ és W elemeivel (minden $a \in \Sigma \cup \{\varepsilon\}$ és $z \in W$-nek megfelel egy oszlop). A $q \in Q$ állapotnak megfelelő sor és az $a \in \Sigma \cup \{\varepsilon\}$ és $z \in W$-nek megfelelő oszlop találkozásánál találhatók a $(w_1, p_1), \ldots, (w_k, p_k)$ párok, ha $\delta(q, a, z) = \{(w_1, p_1), \ldots, (w_k, p_k)\}$.

Ugyanakkor könnyen értelmezhetjük az átmenetgráfot is, amely csupán annyiban különbözik a véges automatáknál használt gráftól, hogy a $\big(p, (a, z/w), q\big)$ átmenetnek megfelelően a (p, q) élre $(a, z/w)$ kerül.

3.1.2. példa. $\mathrm{V}_1 = (\{q_0, q_1, q_2\}, \{a, b\}, \{z_0, z_1\}, E, q_0, z_0, \{q_0\})$. Az E halmaz elemei:

$\big(q_0, (a, z_0/z_0z_1), q_1\big)$
$\big(q_1, (a, z_1/z_1z_1), q_1\big)$ $\qquad$ $\big(q_1, (b, z_1/\varepsilon), q_2\big)$
$\big(q_2, (b, z_1/\varepsilon), q_2\big)$ $\qquad$ $\big(q_2, (\varepsilon, z_0/\varepsilon), q_0\big)$.

Az átmenetfüggvény:

$\delta(q_0, a, z_0) = \{(z_0z_1, q_1)\}$
$\delta(q_1, a, z_1) = \{(z_1z_1, q_1)\}$ $\qquad$ $\delta(q_1, b, z_1) = \{(\varepsilon, q_2)\}$
$\delta(q_2, b, z_1) = \{(\varepsilon, q_2)\}$ $\qquad$ $\delta(q_2, \varepsilon, z_0) = \{(\varepsilon, q_0)\}$.

Az átmenettáblázat:

$\Sigma \cup \{\varepsilon\}$	a		b	ε
W	z_0	z_1	z_1	z_0
q_0	(z_0z_1, q_1)			
q_1		(z_1z_1, q_1)	(ε, q_2)	
q_2			(ε, q_2)	(ε, q_0)

Mivel az átmenetfüggvényben minden halmaz, amelyik nem üres, csak egy elemet tartalmaz (pl. $\delta(q_0, a, z_0) = \{(z_0z_1, q_1)\}$), a fenti átmenettáblázatban minden négyzetben csak egy elem szerepel, és nem használjuk a halmaz szokásos jelölését. Általában, ha egy halmaz egynél több elemet tartalmaz, akkor elemeit egymás alá írjuk. A veremautomata átmenetgráfja a 3.2. ábrán látható. □

Az aktuális állapot, a bemeneti szó még el nem olvasott része és a verem tartalma együtt képezik a veremautomata egy ***konfigurációját***, vagyis minden $q \in Q$, $u \in \Sigma^*$, és $v \in W^*$ esetében (q, u, v) egy konfiguráció. Úgy képzeljük el a v szót, mintha vermet jobbra döltöttük volna, azaz a verem teteje a szó utolsó betűje.

Ha $u = a_1a_2 \dots a_k$ és $v = x_1x_2 \dots x_m$, akkor a veremautomata kétféleképpen léphet (azaz konfigurációt válthat):

- $(q,\ a_1a_2 \dots a_k,\ x_1x_2 \dots x_{m-1}x_m) \Longrightarrow (p,\ a_2a_3 \dots a_k,\ x_1, x_2 \dots x_{m-1}w)$, ha $\big(q, (a_1, x_m/w), p\big) \in E$
- $(q,\ a_1a_2 \dots a_k,\ x_1x_2 \dots x_m) \Longrightarrow (p,\ a_1a_2 \dots a_k,\ x_1, x_2 \dots x_{m-1}w)$, ha $\big(q, (\varepsilon, x_m/w), p\big) \in E$.

A $\Longrightarrow$ reláció reflexív, tranzitív lezártját $\overset{*}{\Longrightarrow}$-gal jelöljük. A $\Longrightarrow$ jel helyett szokták még használni a $\vdash$ jelet is.

Az automata működése: elindulunk a $(q_0, a_1a_2 \dots a_n, z_0)$ kezdeti konfigurációból, majd meghatározzuk az összes lehetséges következő konfigurációt, majd ezekre a rákövetkezőket, és így tovább, ameddig lehet.

3.1.3. értelmezés. *Azt mondjuk, hogy a* V *veremautomata* ***végállapottal*** *felismer egy u szót, ha van* V*-beli konfigurációknak olyan sorozata, amelyre teljesülnek a következők:*

- *a sorozat első eleme* (q_0, u, z_0),
- *a sorozat minden eleméből van átmenet a sorozat következő elemébe, kivéve ha csak egy elemből áll,*
- *a sorozat utolsó eleme* (p, ε, w), *ahol* $p \in F$ *és* $w \in W^*$.

Tehát V akkor és csakis akkor ismeri fel egy u szót végállapottal, ha $(q_0, u, z_0) \overset{*}{\Longrightarrow} (p, \varepsilon, w)$, valamely $w \in W^*$-ra és $p \in F$-re. A V veremautomata által végállapottal felismert szavak halmazát a V által végállapottal felismert nyelvnek nevezzük, és $L(\mathrm{V})$-vel jelöljük.

3.1.4. értelmezés. *Azt mondjuk, hogy a* V *veremautomata* ***üres veremmel*** *felismer egy u szót, ha van* V*-beli konfigurációknak olyan sorozata, amelyre teljesülnek a következők:*

- *a sorozat első eleme (q_0, u, z_0),*
- *a sorozat minden eleméből van átmenet a sorozat következő elemébe,*
- *a sorozat utolsó eleme $(p, \varepsilon, \varepsilon)$, és p tetszőleges állapot.*

Tehát V akkor és csakis akkor ismer fel egy u szót üres veremmel, ha $(q_0, u, z_0) \overset{*}{\Longrightarrow} (p, \varepsilon, \varepsilon)$ valamely $p \in Q$-ra. A V veremautomata által üres veremmel felismert szavak halmazát a V által üres veremmel felismert nyelvnek nevezzük és $L_\varepsilon(\mathrm{V})$-vel jelöljük.

3.1.5. példa. Ha a 3.1.2. példa V_1 automatáját megvizsgáljuk, észrevehetjük, hogy az $\{a^n b^n \mid n \geq 0\}$ nyelvet ismeri fel végállapottal. Végezzük el a levezetést a következő szavakra: *aaabbb* és *abab*.

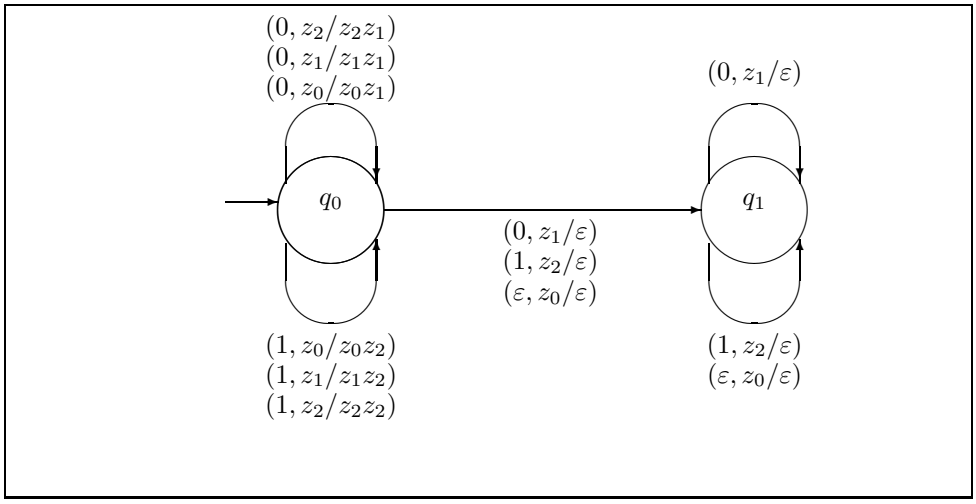
Az a^3b^3 szót felismeri az automata, mivel:
$(q_0, aaabbb, z_0) \Longrightarrow (q_1, aabbb, z_0z_1) \Longrightarrow (q_1, abbb, z_0z_1z_1) \Longrightarrow (q_1, bbb, z_0z_1z_1z_1)$
$\Longrightarrow (q_2, bb, z_0z_1z_1) \Longrightarrow (q_2, b, z_0z_1) \Longrightarrow (q_2, \varepsilon, z_0) \Longrightarrow (q_0, \varepsilon, \varepsilon)$, és mivel q_0 végállapot, a veremautomata felismeri a szót. Ugyanakkor, mivel a verem kiürült, üres veremmel is felismeri.

Mivel a kezdőállapot egyben végállapot is, az üres szót is felismeri végállapottal, ellenben üres veremmel nem.

Hogy bebizonyítsuk, hogy az *abab* szót nem ismeri fel, szükségünk van az összes lehetőség megvizsgálására. Könnyű belátni, hogy ebben az esetben csak egyetlen lehetőség van:
$(q_0, abab, z_0) \Longrightarrow (q_1, bab, z_0z_1) \Longrightarrow (q_2, ab, z_0) \Longrightarrow (q_0, ab, \varepsilon)$, de innen nincs átmenet, tehát nem ismeri fel az *abab* szót. □

3.1.6. példa. A $\mathrm{V}_2 = (\{q_0, q_1\}, \{0, 1\}, \{z_0, z_1, z_2\}, E, q_0, z_0, \emptyset)$ veremautomata átmenettáblázata:

	0			1			ε
W	z_0	z_1	z_2	z_0	z_1	z_2	z_0
q_0	(z_0z_1, q_0)	(z_1z_1, q_0) (ε, q_1)	(z_2z_1, q_0)	(z_0z_2, q_0)	(z_1z_2, q_0)	(z_2z_2, q_0) (ε, q_1)	(ε, q_1)
q_1		(ε, q_1)				(ε, q_1)	(ε, q_1)



3.3. ábra. A 3.1.6. példa átmenetgráfja.

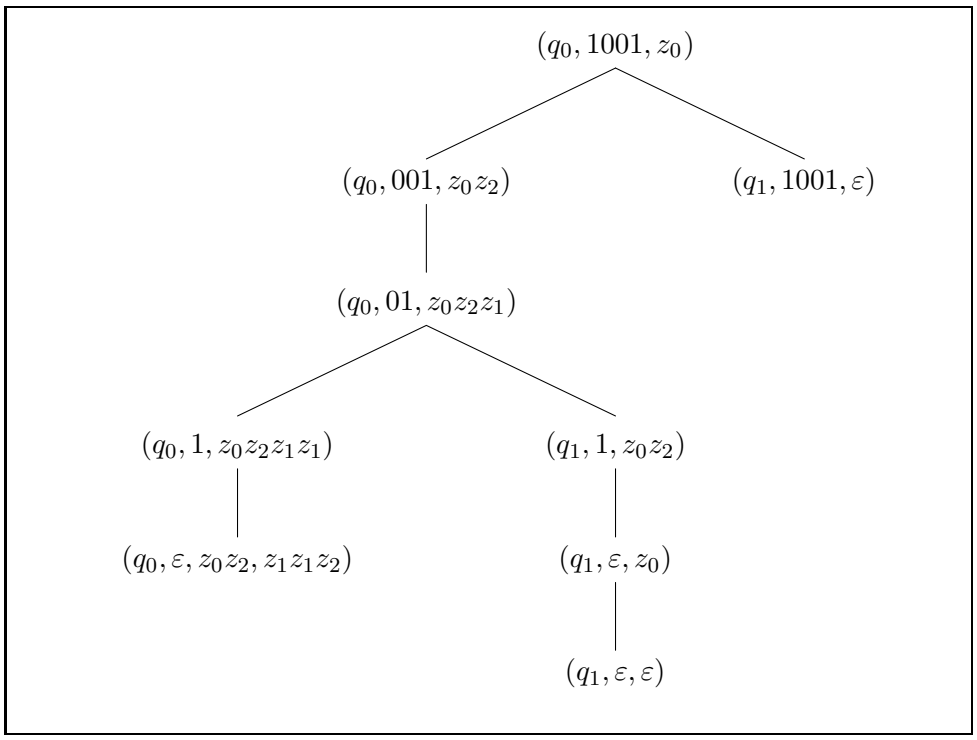
Az átmenetgráf a 3.3. ábrán látható. A V_2 veremautomata az $\{uu^{-1} \mid u \in \{0,1\}^*\}$ nyelvet ismeri fel. Mivel V_2 nemdeterminisztikus, a (q_0, u, z_0) kezdőkonfigurációból elérhető összes konfigurációt egy ún. számítási fában tudjuk ábrázolni. Például a $(q_0, 1001, z_0)$ kezdőkonfigurációhoz tartozó számítási fa a 3.4. ábrán látható. A számítási fából megállapíthatjuk, hogy mivel $(q_1, \varepsilon, \varepsilon)$ a fa egyik levele, a V_2 veremautomata üres veremmel felismeri az 1001 szót. A 3.5. ábrán látható számítási fa annak bizonyítéka, hogy a V_2 veremautomata nem ismeri fel az 101 szót. A levelekben lévő konfigurációkat nem lehet folytatni, és egyik sem $(q, \varepsilon, \varepsilon)$ alakú. □

3.1.7. tétel. *Egy L nyelv akkor és csakis akkor ismerhető fel valamely V_1 nemdeterminisztikus veremautomatával üres veremmel, ha felismerhető valamely V_2 nemdeterminisztikus veremautomatával végállapottal.*

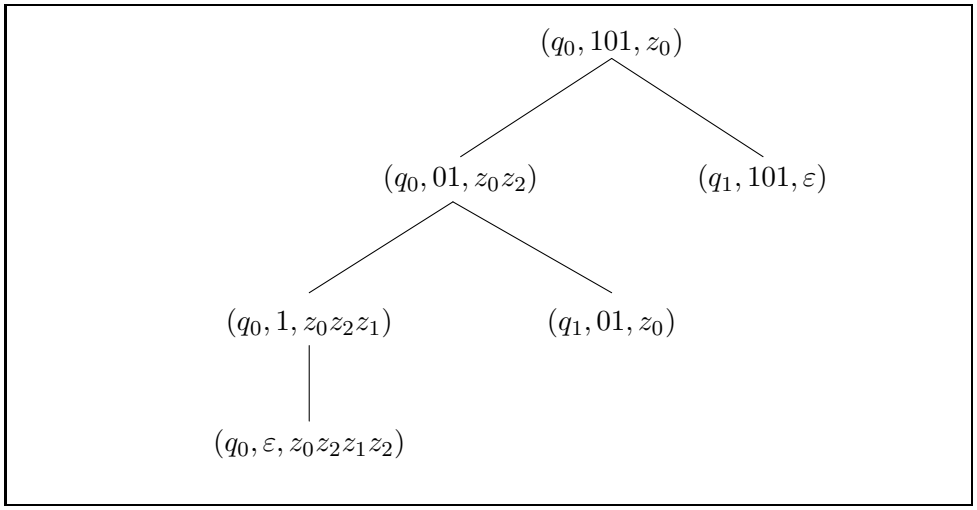
Bizonyítás. *a)* Legyen $\mathrm{V}_1 = (Q, \Sigma, W, E, q_0, z_0, \emptyset)$ veremautomata, amely üres veremmel ismeri fel az L nyelvet. Definiáljuk a $\mathrm{V}_2 = (Q \cup \{p_0, p\}, \Sigma, W \cup \{x\}, E', p_0, x, \{p\})$ veremautomatát, ahol $p, p_0 \notin Q$, $, x \notin W$ és

$$E' = E \cup \Big\{\big(p_0, (\varepsilon, x/xz_0), q_0\big)\Big\} \cup \Big\{\big(q, (\varepsilon, x/\varepsilon), p\big) \mid q \in Q\Big\}.$$

V_2 **működése:** V_2 először egy ε-lépéssel átmegy a V_1 kezdőállapotába, beírva a verembe x mellé z_0-t, V_1 kezdőszimbólumát. Ettől kezdve úgy működik, mint V_1. Ha V_1 egy adott szóra kiüríti a saját vermét, akkor V_2-nél még mindig marad egy x a veremben, amelyet V_2 ε-lépéssel töröl és végállapotba kerül. V_2 csak akkor kerülhet végállapotba, ha V_1 kiürítette a vermét.



3.4. ábra. Az 1001 szó felismerését bizonyító számítási fa (3.1.6. példa).



3.5. ábra. Számítási fa annak bizonyítására, hogy a 3.1.6. példa veremautomatája nem ismeri fel az 101 szót.

b) Legyen $\mathrm{V}_2 = (Q, \Sigma, W, E, q_0, z_0, F)$ egy veremautomata, amely az L nyelvet végállapottal ismeri fel. Definiáljuk a $\mathrm{V}_1 = (Q \cup \{p_0, p\}, \Sigma, W \cup \{x\}, E', p_0, x, \emptyset)$ veremautomatát, ahol $p_0, p \notin Q$, $x \notin W$ és

$$E' = E \cup \Big\{\big(p_0, (\varepsilon, x/xz_0), q_0\big)\Big\} \cup \Big\{\big(q, (\varepsilon, z/\varepsilon), p\big) \mid q \in F, p \in Q, z \in W\Big\}$$
$$\cup \Big\{\big(p, (\varepsilon, z/\varepsilon), p\big) \mid p \in Q, z \in W \cup \{x\}\Big\}$$

V_1 **működése:** V_1 először ε-lépéssel beírja a verembe x mellé z_0-t, V_2 vermének kezdőszimbólumát, ettől kezdve mint V_2 működik, azaz végállapotba jut minden felismert szóra. Innen V_1 ε-lépéssel kiüríti a vermet. V_1 csak akkor ürítheti ki a vermet, ha V_2 végállapotba kerül. □

A következő két tétel azt bizonyítja, hogy a nemdeterminisztikus veremautomaták által felismert nyelvek halmaza éppen a környezetfüggetlen nyelvek halmaza.

3.1.8. tétel. *Ha G környezetfüggetlen nyelvtan, akkor létezik egy olyan* V *nemdeterminisztikus veremautomata, amely üres veremmel felismeri az $L(G)$ nyelvet, azaz $L_\varepsilon(\mathrm{V}) = L(G)$.*

Csak a bizonyítás ötletét adjuk meg. Legyen $G = (N, T, P, S)$ egy környezetfüggetlen nyelvtan. Értelmezzük a $\mathrm{V} = (\{q\}, T, N \cup T, E, q, S, \emptyset)$ veremautomatát, ahol $q \notin N \cup T$, az E átmenethalmaz értelmezése pedig:

• Ha létezik a G nyelvtan szabályai között $A \to \alpha$ szabály, akkor vegyük be E-be a $\big(q, (\varepsilon, A/\alpha^{-1}), q\big)$ átmenetet,

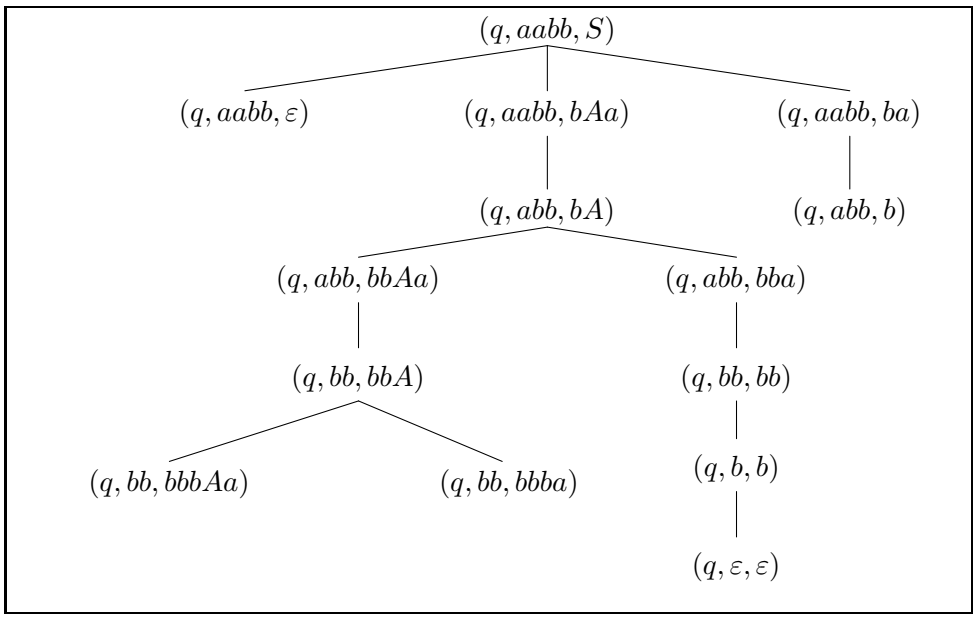
• Minden $a \in T$ jelre vegyük be E-be a $\big(q, (a, a/\varepsilon), q\big)$ átmenetet.

Ha van $S \to \alpha$ szabály G-ben, a veremautomata egy ε-lépéssel beírja a verembe az α tükörképét. Ha a beolvasott betű egyezik a verem tetején lévővel, akkor törli azt a veremből. Ha a verem tetején az A nemterminális betű van, akkor beviszi a verembe valamelyik A-val kezdődő szabály jobb oldalának a tükörképét. Ha a szó beolvasása végén a verem kiürül, a veremautomata felismerte a szót.

A következő algoritmus egy $G = (N, T, P, S)$ környezetfüggetlen nyelvtanhoz megkonstruálja azt a $\mathrm{V} = (\{q\}, T, N \cup T, E, q, S, \emptyset)$ veremautomatát, amelyik üres veremmel felismeri a G által generált nyelvet.

KÖRNYEZETFÜGGETLEN-NYELVTANBÓL-VEREMAUTOMATA(G)
1 **for** minden $A \to \alpha$ szabályra
2 **do** vegyük be E-be a $\big(q, (\varepsilon, A/\alpha^{-1}), q\big)$ átmenetet
3 **for** minden $a \in T$ terminálisra
4 **do** vegyük be E-be a $\big(q, (a, a/\varepsilon), q\big)$ átmenetet
5 **return** V

Ha a G nyelvtan szabályainak száma n, a terminális betűké pedig m, akkor az algoritmus lépésszáma $\Theta(n + m)$.



3.6. ábra. Szó felismerése üres veremmel (3.1.9. példa).

3.1.9. példa. Legyen $G = (\{S, A\}, \{a, b\}, \{S \to \varepsilon,\ S \to ab,\ S \to aAb,\ A \to aAb,\ A \to ab\}, S)$. Ekkor $\mathrm{V} = (\{q\}, \{a, b\}, \{a, b, A, S\}, E, q, S, \emptyset)$, a következő átmenettáblázattal.

$\Sigma \cup \{\varepsilon\}$	a	b	ε	
W	a	b	S	A
q	(ε, q)	(ε, q)	(ε, q) (ba, q) (bAa, q)	(bAa, q) (ba, q)

Nézzük meg, hogyan ismeri fel a V veremautomata az $aabb$ szót, amelyet a G nyelvtanban a következőképpen lehet levezetni.

$$S \Longrightarrow aAb \Longrightarrow aabb,$$

ahol alkalmaztuk az $S \to aAb$ és $A \to ab$ szabályokat. A felismerés üres veremmel történik (3.6. ábra). □

3.1.10. tétel. *Ha* V *nemdeterminisztikus veremautomata, akkor létezik egy olyan G környezetfüggetlen nyelvtan, hogy* V *üres veremmel felismeri az $L(G)$ nyelvet, azaz $L_\varepsilon(\mathrm{V}) = L(G)$.*

Bizonyítás helyett megadjuk, hogyan kell definiálni a G nyelvtant. Legyen a nemdeterminisztikus veremautomata $\mathrm{V} = (Q, \Sigma, W, E, q_0, z_0, \emptyset)$.

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

$N = \{S\} \cup \{S_{p,z,q} \mid p, q \in Q, z \in W\}$ és $T = \Sigma$.

A P szabályait pedig a következőképpen kapjuk meg.

• Minden q állapotra vegyük be P-be az $S \to S_{q_0,z_0,q}$ szabályt.

• Ha $\big(q, (a, z/z_k \dots z_2 z_1), p\big) \in E$, ahol $q \in Q$, $z, z_1, z_2, \dots z_k \in W$ $(k \geq 1)$ és $a \in \Sigma \cup \{\varepsilon\}$, vegyük be P-be minden lehetséges $p_1, p_2, \dots, p_k$ állapotra az $S_{q,z,p_k} \to a S_{p,z_1,p_1} S_{p_1,z_2,p_2} \dots S_{p_{k-1},z_k,p_k}$ szabályokat.

• Ha $\big(q, (a, z/\varepsilon), p\big) \in E$, ahol $p, q \in Q, z \in W$, és $a \in \Sigma \cup \{\varepsilon\}$, vegyük be P-be az $S_{q,z,p} \to a$ szabályt.

Az így értelmezett nyelvtan kiterjesztett környezetfüggetlen nyelvtan, amelyhez, mint tudjuk, mindig hozzárendelhető egy vele ekvivalens környezetfüggetlen nyelvtan. A tétel bizonyítása azon alapszik, hogy minden konfigurációsorozatnak, amelynek segítségével a V veremautomata felismer egy szót, megfelel egy levezetés a G nyelvtanban. Azt, hogy ez levezetés éppen az adott szót generálja, az $S_{q,z,p_k} \to a S_{p,z_1,p_1} S_{p_1,z_2,p_2} \dots S_{p_{k-1},z_k,p_k}$ alakú szabályok biztosítják azáltal, hogy minden lehetséges $p_1, p_2, \dots, p_k$ állapotra értelmeztük őket. A 3.1.6. példa segítségével megmutatjuk, hogyan rendelhető hozzá egy levezetés az adott konfigurációsorozathoz. A példában értelmezett veremautomata a

$(q_0, 00, z_0) \Longrightarrow (q_0, 0, z_0 z_1) \Longrightarrow (q_1, \varepsilon, z_0) \Longrightarrow (q_1, \varepsilon, \varepsilon)$

konfigurációsorozattal ismeri fel a 00 szót, amely sorozat az

$\big(q_0, (0, z_0/z_0 z_1), q_0\big)$,

$\big(q_0, (0, z_1/\varepsilon), q_1\big)$,

$\big(q_1, (\varepsilon, z_1/\varepsilon), q_1\big)$.

átmeneteken alapszik.

A G nyelvtan értelmezése szerint ezeknek rendre megfelelnek a következő helyettesítési szabályok

(1) $S_{q_0,z_0,p_2} \longrightarrow 0 S_{q_0,z_1,p_1} S_{p_1,z_0,p_2}$ minden $p_1, p_2 \in Q$ állapotra,

(2) $S_{q_0,z_1,q_1} \longrightarrow 0$,

(3) $S_{q_1,z_0,q_1} \longrightarrow \varepsilon$.

Továbbá, minden q állapotra definiáltuk az $S \longrightarrow S_{q_0,z_0,q}$ szabályokat is.

Az $S \longrightarrow S_{q_0,z_0,q}$ szabály alapján létezik az $S \Longrightarrow S_{q_0,z_0,q}$ levezetés ahol q tetszőlegesen választható. Válasszuk a fenti (1) szabályban p_2-t is q-nak. Ekkor létezik az

$S \Longrightarrow S_{q_0,z_0,q} \Longrightarrow 0 S_{q_0,z_1,p_1} S_{p_1,z_0,q}$

levezetés is, ahol $p_1 \in Q$ tetszőlegesen megválasztható. Ha $p_1 = q_1$, akkor az

$S \Longrightarrow S_{q_0,z_0,q} \Longrightarrow 0 S_{q_0,z_1,q_1} S_{q_1,z_0,q} \Longrightarrow 00 S_{q_1,z_0,q}$

levezetést kapjuk. Most q-t választhatjuk q_1-nek, és ekkor

$S \Longrightarrow S_{q_0,z_0,q_1} \Longrightarrow 0S_{q_0,z_1,q_1}S_{q_1,z_0,q_1} \Longrightarrow 00S_{q_1,z_0,q_1} \Longrightarrow 00,$
amely azt bizonyítja, hogy a 00 szó levezethető a fent definiált nyelvtanban.

A következő algoritmus egy $\mathrm{V} = (Q, \Sigma, W, E, q_0, z_0, \emptyset)$ veremautomatához megkonstruálja azt a $G = (N, T, P, S)$ környezetfüggetlen nyelvtant, amely a V veremautomata által üres veremmel felismert környezetfüggetlen nyelvet generálja.

VEREMAUTOMATÁBÓL-KÖRNYEZETFÜGGETLEN-NYELVTAN(V)
1 **for** minden $q \in Q$
2 **do** vegyük be P-be az $S \to S_{q_0,z_0,q}$ szabályt
3 **for** minden $\big(q,(a,z/z_k \dots z_2 z_1),p\big) \in E$
$\triangleright\ q \in Q,\ z, z_1, z_2, \dots z_k \in W\ (k \geq 1),\ a \in \Sigma \cup \{\varepsilon\}$
4 **do for** minden $p_1, p_2, \dots, p_k$ állapotra
5 **do** vegyük be P-be az $S_{q,z,p_k} \to aS_{p,z_1,p_1}S_{p_1,z_2,p_2} \dots S_{p_{k-1},z_k,p_k}$ szabályokat
6 **for** minden $\big(q(a,z/\varepsilon),p\big) \in E$ $\triangleright\ p, q \in Q, z \in W,\ a \in \Sigma \cup \{\varepsilon\}$
7 **do** vegyük be P-be az $S_{q,z,p} \to a$ szabályt
8 **return** G

Ha az automata állapotainak száma n, átmeneteinek száma pedig m, akkor a fenti algoritmus legfeljebb $n+mn+m$ lépést hajt végre, tehát a lépésszáma legrosszabb esetben $O(nm)$.

Végül bizonyítás nélkül megemlítjük, hogy a determinisztikus veremautomatákkal felismerhető nyelvek osztálya valódi része a nemdeterminisztikus veremautomatákkal felismerhető nyelvek osztályának. Ebből a szempontból a veremautomaták a véges automatáktól eltérő módon viselkednek.

3.1.11. példa. Példaként, tekintsük az előbbi példában (3.1.9. példa) meghatározott V veremautomatát: $\mathrm{V} = (\{q\}, \{a,b\}, \{a,b,A,S\}, E, q, S, \emptyset)$. A G nyelvtan a következő:

$$G = (\{S, S_a, S_b, S_S, S_A, \}, \{a,b\}, P, S)\ ,$$

ahol minden $z \in \{a,b,S,A\}$ esetén S_z az $S_{q,z,q}$ jelölés rövidítése. Felírjuk a veremautomata átmeneteit:

$\big(q,(a,a/\varepsilon),q\big)$, $\big(q,(b,b/\varepsilon),q\big)$,
$\big(q,(\varepsilon,S/\varepsilon),q\big)$, $\big(q,(\varepsilon,S/ba),q\big)$, $\big(q,(\varepsilon,S/bAa),q\big)$,
$\big(q,(\varepsilon,A/ba),q\big)$, $\big(q,(\varepsilon,A/bAa),q\big)$.

Ezek alapján a következő szabályokat definiálhatjuk:

$S \to S_S$
$S_a \to a$
$S_b \to b$
$S_S \to \varepsilon \mid S_aS_b \mid S_aS_AS_b$
$S_A \to S_aS_AS_b \mid S_aS_b.$

Könnyen belátható, hogy S_S-t kiiktathatjuk, így a szabályok, miután az elsőt elhagyjuk, a következő lesznek:

$S \to \varepsilon \mid S_a S_b \mid S_a S_A S_b,$
$S_A \to S_a S_A S_b \mid S_a S_b,$
$S_a \to a, \;\; S_b \to b,$

ezek a szabályok pedig helyettesíthetők a következőkkel:

$S \to \varepsilon \mid ab \mid aAb,$
$A \to aAb \mid ab.$ □

3.2. Környezetfüggetlen nyelvek

Vegyünk egy $G = (N, T, P, S)$ környezetfüggetlen nyelvtant. G egy ***levezetési fájának*** egy olyan véges, rendezett, címkézett fát nevezünk, amelynek gyökere az S kezdőszimbólummal van címkézve, minden belső csúcs címkéje egy nemterminális szimbólum, és levelei terminális szimbólumokkal vannak címkézve. Teljesül továbbá az a feltétel, hogy ha egy belső csúcs címkéje az A nemterminális, és a csúcsnak k közvetlen leszármazottja van, akkor P-ben van egy olyan $A \to a_1 a_2 \dots a_k$ szabály, hogy ezek a közvetlen leszármazottak rendre az $a_1, a_2, \dots a_k$ szimbólumokkal vannak címkézve. A levezetési fa ***eredménye*** az a T feletti szó, amelyet úgy kapunk, hogy a fa leveleinek címkéjét balról jobbra haladva összeolvassuk. A levezetési fát még ***szintaxisfának*** is nevezzük.

Tekintsük a $G = (\{S, A\}, \{a, b\}, \{S \to aA,\ S \to a,\ S \to \varepsilon,\ A \to aA,\ A \to aAb,\ A \to ab,\ A \to b\}, S)$ környezetfüggetlen nyelvtant. Ez a nyelvtan az $L(G) = \{a^n b^m \mid n \geq m \geq 0\}$ nyelvet generálja. Az $a^4 b^2 \in L(G)$ szó levezetése a következő:

$$S \Longrightarrow aA \Longrightarrow aaA \Longrightarrow aaaAb \Longrightarrow aaaabb.$$

Ezt a levezetést ábrázolhatjuk a 3.7. ábrán lévő levezetési fával, amelynek eredménye az *aaaabb* szó.

Minden levezetéshez hozzárendelhető egy levezetési fa. Fordítva, egy levezetési fához több levezetés is rendelhető. Például, a 3.7. ábrán látható fához, a fenti levezetésen kívül a

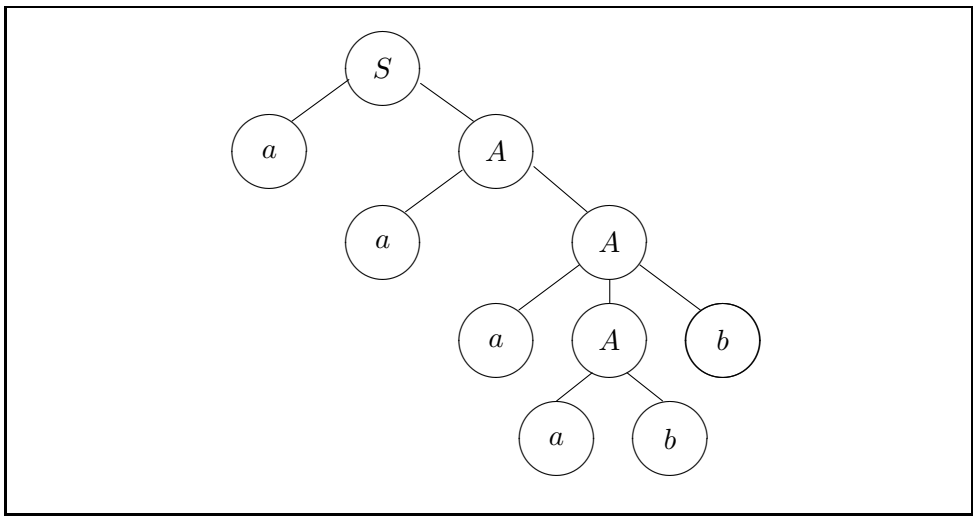
$$S \Longrightarrow aA \Longrightarrow aaAb \Longrightarrow aaaAb \Longrightarrow aaaabb.$$

levezetés is hozzárendelhető.

3.2.1. értelmezés. *Az $\alpha_0 \Longrightarrow \alpha_1 \Longrightarrow \dots \Longrightarrow \alpha_n$ levezetést* ***legbaloldalibb levezetésnek*** *nevezzük, ha minden $i = 1, 2, \dots, n-1$ esetén van olyan $u_i \in T^*$, $\beta_i \in (N \cup T)^*$ szó és $(A_i \to \gamma_i) \in P$ szabály, hogy teljesülnek az*

$$\alpha_i = u_i A_i \beta_i \quad \text{és} \quad \alpha_{i+1} = u_i \gamma_i \beta_i$$

összefüggések.



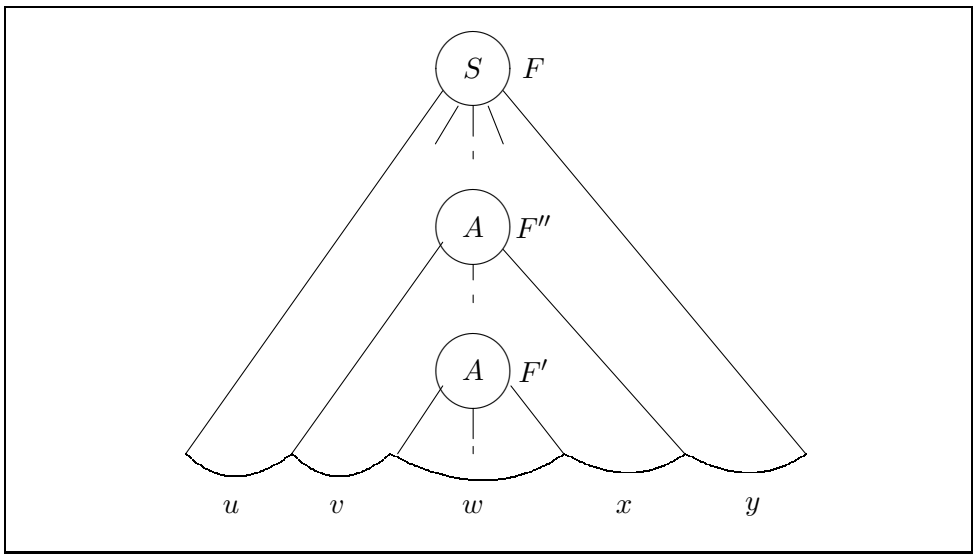
3.7. ábra. Az *aaaabb* szó levezetési fája.

Tekintsük a következő nyelvtant:
$G = (\{S, A\}, \{a, b, c\}, \{S \to bA,\ S \to bAS,\ S \to a,\ A \to cS,\ A \to a\}, S)$.
A *bcbaa* szónak két, egymástól különböző legbaloldalibb levezetése van:
$S \Longrightarrow bA \Longrightarrow bcS \Longrightarrow bcbAS \Longrightarrow bcbaS \Longrightarrow bcbaa$,
$S \Longrightarrow bAS \Longrightarrow bcSS \Longrightarrow bcbAS \Longrightarrow bcbaS \Longrightarrow bcbaa$.

3.2.2. értelmezés. *Egy G környezetfüggetlen nyelvtan* ***nem egyértelmű****, ha $L(G)$-ben van olyan szó, amelynek egynél több legbaloldalibb levezetése van. Különben a G nyelvtan* ***egyértelmű****.*

Az előbbi G nyelvtan nem egyértelmű, mert a *bcbaa* szónak találtunk két legbaloldalibb levezetését. Egy nyelvet több nyelvtan is generálhat, és ezek között lehetnek egyértelműek és nem egyértelműek is. Egy környezetfüggetlen nyelv ***alapvetően nem egyértelmű,*** ha nem létezik egyetlen egyértelmű nyelvtan sem, amely generálja.

3.2.3. példa. Vizsgáljuk meg a következő két nyelvtant.
A $G_1 = (\{S\}, \{a, +, *\}, \{S \to S + S,\ S \to S * S,\ S \to a\}, S)$ nem egyértelmű, mert
$S \Longrightarrow S+S \Longrightarrow a+S \Longrightarrow a+S*S \Longrightarrow a+a*S \Longrightarrow a+a*S+S \Longrightarrow a+a*a+S$
$\Longrightarrow a + a * a + a$ és
$S \Longrightarrow S*S \Longrightarrow S+S*S \Longrightarrow a+S*S \Longrightarrow a+a*S \Longrightarrow a+a*S+S \Longrightarrow a+a*a+S$
$\Longrightarrow a + a * a + a$.
A $G_2 = (\{S, A\}, \{a, *, +\}, \{S \to A + S \mid A,\ A \to A * A \mid a\}, S)$ nyelvtan egyértelmű.
Be lehet bizonyítani, hogy $L(G_1) = L(G_2)$. □



3.8. ábra. A pumpáló lemma bizonyításában szereplő fa felbontása.

3.3. Pumpáló lemma környezetfüggetlen nyelvekre

Környezetfüggetlen nyelvekre is létezik a reguláris nyelveknél megismert pumpáló lemmához hasonló lemma.

3.3.1. tétel. (pumpáló lemma) *Tetszőleges L környezetfüggetlen nyelvhez megadható egy olyan n természetes szám (amely csak az adott nyelvtől függ) úgy, hogy a nyelv minden n-nél hosszabb z szavát fel lehet írni $uvwxy$ alakban, és igazak a következők:*

(1) $|w| \geq 1$,
(2) $|vx| \geq 1$,
(3) $|vwx| \leq n$,
(4) uv^iwx^iy *is eleme* L*-nek, minden* $i \geq 0$ *értékre.*

Bizonyítás. Legyen $G = (N, T, P, S)$ egy olyan, átnevezéseket nem tartalmazó környezetfüggetlen nyelvtan, amely L-et generálja. Legyen $m = |N|$ a nemterminális jelek száma, és legyen ℓ a P-beli szabályok jobb oldalai hosszának maximuma, azaz $\ell = \max\{|\alpha| \mid \exists A \in N : (A \to \alpha) \in P\}$. Legyen $n = \ell^{m+1}$ és $z \in L(G)$ úgy, hogy $|z| > n$. Ekkor létezik egy olyan F levezetési fa, amelynek eredménye z. Legyen F magassága (a gyökértől a levelekig vezető utak hosszának a maximuma) h. Mivel F-ben minden belső csúcsnak legfeljebb ℓ leszármazottja van, ezért F-nek legfeljebb ℓ^h levele van, vagyis $|z| \leq \ell^h$. Másrészt, mivel $|z| > \ell^{m+1}$, kapjuk, hogy $h > m + 1$. Ebből következik, hogy az F levezetési fában van olyan út gyökértől levélig, amelyben

$(m+1)$-nél több csúcs van. Tekintsünk egy ilyen utat. Mivel G nemterminálisainak száma m és ezen az úton minden, levéltől különböző csúcs nemterminálissal van címkézve, a skatulya-elv szerint van olyan nemterminális, amelyik legalább kétszer fordul elő ezen az úton.

Tekintsük azt a nemterminálist, amelyik a levéltől a gyökérig haladva legelőször ismétlődik az úton és jelöljük A-val. Jelöljük F'-vel azt részfát, amelynek gyökere A ezen előfordulása. Hasonlóképpen, jelöljük F''-vel azt a részfát, amelynek gyökere az A következő (második) előfordulása az úton. Legyen F' eredménye w. Akkor F'' eredménye vwx, míg F eredménye $uvwxy$ alakban írható. Az F levezetési fa és z ezen felbontása a 3.8.. ábrán látható. Megmutatjuk, hogy z felbontása kielégíti a lemmában megkövetelt (1)–(4) feltételeket.

Mivel P-ben nincsenek ε-szabályok (kivéve esetleg az $S \to \varepsilon$ szabályt), ezért $|w| \geq 1$. Továbbá, mivel a levezetési fa minden csúcsából és így F'' gyökeréből is legalább két él fut ki (ugyanis nincsenek átnevezések sem), ezért $|vx| \geq 1$. Mivel A az a nemterminális, amelyik a vizsgált úton a levéltől a gyökérig haladva legelőször megismétlődik, ezért F'' magassága legfeljebb $m+1$, amiből következik, hogy $|vwx| \leq \ell^{m+1} = n$.

Ha F-ből eltávolítjuk F'' csúcsait, csak a gyökeret hagyva meg, akkor az így kapott fa eredménye uAy, azaz $S \overset{*}{\underset{G}{\Longrightarrow}} uAy$.

Hasonlóképpen kapjuk F''-ből eltávolítva F'-t, hogy $A \overset{*}{\underset{G}{\Longrightarrow}} vAx$, és végül F' definíciója miatt, hogy $A \overset{*}{\underset{G}{\Longrightarrow}} w$. Tehát $S \overset{*}{\underset{G}{\Longrightarrow}} uAy$, $A \overset{*}{\underset{G}{\Longrightarrow}} vAx$ és $A \overset{*}{\underset{G}{\Longrightarrow}} w$. Innen $S \overset{*}{\underset{G}{\Longrightarrow}} uAy \overset{*}{\underset{G}{\Longrightarrow}} uwy$ és $S \overset{*}{\underset{G}{\Longrightarrow}} uAy \overset{*}{\underset{G}{\Longrightarrow}} uvAxy \overset{*}{\underset{G}{\Longrightarrow}} \ldots \overset{*}{\underset{G}{\Longrightarrow}} uv^iAx^iy \overset{*}{\underset{G}{\Longrightarrow}} uv^iwx^iy$ tetszőleges $i \geq 1$ értékre. Tehát tetszőleges $i \geq 0$-ra $S \overset{*}{\Longrightarrow} uv^iwx^iy$, vagyis tetszőleges $i \geq 0$-ra $uv^iwx^iy \in L(G)$. □

Bemutatjuk a pumpáló lemma két következményét.

3.3.2. következmény. $\mathcal{L}_2 \subset \mathcal{L}_1$.

Bizonyítás. A következmény azt állítja, hogy létezik olyan környezetfüggő nyelv, amely nem környezetfüggetlen, tehát a tartalmazás szigorú. Ennek bizonyításához elég, ha találunk egy olyan környezetfüggő nyelvet, amelyre az előbbi pumpáló lemma nem teljesül. Legyen ez $L = \{a^mb^mc^m \mid m \geq 1\}$.

Hogy ez a nyelv környezetfüggő, azt könnyen lehet igazolni, megfelelő nyelvtan megadásával. A 1.2.6. példában megadott két nyelvtan mindegyike kiterjesztett környezetfüggő nyelvtan. De tudjuk, hogy tetszőleges i típusú kiterjesztett nyelvtanhoz mindig hozzárendelhető ugyanolyan típusú és vele ekvivalens nyelvtan.

Legyen n a pumpáló lemmában az L-hez tartozó természetes szám, és tekintsük a $z = a^nb^nc^n$ szót. Mivel $|z| = 3n > n$, z felírható $z = uvwxy$ alakban

úgy, hogy teljesülnek a pumpáló lemmában szereplő (1)–(4) feltételek. Megmutatjuk, hogy ez ellentmondáshoz vezet.

Először megmutatjuk, hogy a v és x szavak mindegyike legfeljebb egyféle betűt tartalmaz. Valóban, ha v és x valamelyike egynél többféle betűt tartalmaz, akkor az $uvvwxxy$ szóban a betűk sorrendje nem az a, b, c sorrend, tehát $uvvwxxy \notin L(G)$, ami ellentmond a pumpáló lemmában szereplő (4) feltételnek.

Ha viszont v és x mindegyike legfeljebb egyféle betűt tartalmaz, akkor az uwy szóban valamelyik betű többször fordul elő, mint a másik kettő, tehát $uwy \notin L(G)$. Ez is ellentmond a pumpáló lemmában szereplő (4) feltételnek, tehát L nem környezetfüggetlen. □

3.3.3. következmény. *A környezetfüggetlen nyelvek osztálya nem zárt a metszetre.*

Bizonyítás. Megadunk két olyan környezetfüggetlen nyelvet, amelyeknek metszete nem környezetfüggetlen. Legyen $N = \{S, A, B\}$, $T = \{a, b, c\}$ és $G_1 = (N, T, P_1, S)$ ahol P_1 :

$S \to AB$,
$A \to aAb \mid ab$,
$B \to cB \mid c$,

és $G_2 = (N, T, P_2, S)$, ahol P_2 :

$S \to AB$,
$A \to Aa \mid a$,
$B \to bBc \mid bc$.

Tehát az $L(G_1) = \{a^n b^n c^m \mid n \geq 1, m \geq 1\}$ és az $L(G_2) = \{a^n b^m c^m \mid n \geq 1, m \geq 1\}$ nyelvek környezetfüggetlenek. Ugyanakkor

$$L(G_1) \cap L(G_2) = \{a^n b^n c^n \mid n \geq 1\}$$

nem környezetfüggetlen, mint ahogy azt a 3.3.2 következmény bizonyításában láttuk. □

3.4. Környezetfüggetlen nyelvtanok normálalakjai

Tetszőleges nyelvtanok esetében a normálalakot úgy értelmeztük (lásd 12. oldal), hogy a szabályok bal oldalán csak nemterminálisok szerepelnek. Környezetfüggetlen nyelvtanok esetében a normálalak a szabályok jobb oldalára adott bizonyos megkötéseket jelent. A következőkben a környezetfüggetlen nyelvtanoknak két normálalakját vizsgáljuk meg, a Chomsky-, illetve

a Greibach-féle[1] normálalakokat.

3.4.1. Chomsky-féle normálalak

3.4.1. értelmezés. *Egy* $G = (N, T, P, S)$ *környezetfüggetlen nyelvtant Chomsky-normálalakú nevezünk, ha minden szabálya* $A \to a$ *vagy* $A \to BC$ *alakú, ahol* $A, B, C \in N$, $a \in T$.

3.4.2. példa. A $G = (\{S, A, B, C\}, \{a, b\}, \{S \to AB,\ S \to CB,\ C \to AS,\ A \to a,\ B \to b\}, S)$ nyelvtan Chomsky-normálalakú és $L(G) = \{a^n b^n \mid n \geq 1\}$. □

Minden ε-mentes környezetfüggetlen nyelvtanhoz megadható egy vele ekvivalens Chomsky-normálalakú nyelvtan. Megadunk egy algoritmust, amely a $G = (N, T, P, S)$ ε-mentes környezetfüggetlen nyelvtant átalakítja a $G' = (N', T, P', S)$ Chomsky-normálalakúvá.

CHOMSKY-ALAK(G)
1 $N' \leftarrow N$
2 küszöböljük ki a szabályokban az átnevezéseket, és legyen P' az új szabályhalmaz (lásd ÁTNEVEZÉS-KIZÁRÁS algoritmus, 11. oldal)
3 a P' minden olyan szabályában, amelynek jobb oldala legalább két szimbólumból áll, minden a terminális jelet helyettesítsünk egy új A nemterminálissal, amelyet vegyünk fel N'-be, és vegyük fel a szabályok közé az $A \to a$ új szabályt
4 minden $B \to A_1 A_2 \dots A_k$ alakú szabályt, ahol $k \geq 3$ és $A_1, A_2, \dots, A_k \in N$, helyettesítsünk a következőkkel:

$$\begin{aligned} B &\to A_1 C_1, \\ C_1 &\to A_2 C_2, \\ &\dots \\ C_{k-3} &\to A_{k-2} C_{k-2}, \\ C_{k-2} &\to A_{k-1} A_k, \end{aligned}$$

ahol $C_1, C_2, \dots, C_{k-2}$ új nemterminális szimbólumok, amelyeket felveszünk N'-be.
5 **return** G'

3.4.3. példa. Legyen $G = (\{S, D\}, \{a, b, c\}, \{S \to aSc,\ S \to D,\ D \to bD,\ D \to b\}, S)$. Könnyű belátni, hogy $L(G) = \{a^n b^m c^n \mid n \geq 0, m \geq 1\}$. A Chomsky-féle normálalakú nyelvtanná való alakítás lépései a következők:

[1] Sheila Greibach (1939–), amerikai informatikus.

1. lépés: $N' = \{S, D\}$

2. lépés: Az $S \to D$ átnevezés kiküszöbölése után, a szabályok a következők:

$S \to aSc \mid bD \mid b$,

$D \to bD \mid b$.

3. lépés: Mivel a szabályokban három terminális szerepel, három új nemterminálist vezetünk be (legyenek ezek A, B, C). Ekkor a szabályok:

$S \to ASC \mid BD \mid b$,

$D \to BD \mid b$,

$A \to a$,

$B \to b$,

$C \to c$.

4. lépés: Mivel egyetlen szabály kivételével, amelynek jobb oldala három szimbólumból áll, minden más szabály jobb oldala legfeljebb kettő hosszúságú, egyetlen új nemterminálist kell bevezetnünk, legyen ez E. Ezért $N' = \{S, A, B, C, D, E\}$, a P' szabályai pedig:

$S \to AE \mid BD \mid b$,

$D \to BD \mid b$,

$A \to a$,

$B \to b$,

$C \to c$,

$E \to SC$. □

3.4.2. Greibach-féle normálalak

3.4.4. értelmezés. *Egy $G = (N, T, P, S)$ környezetfüggetlen nyelvtant Greibach-normálalakúnak nevezünk, ha minden szabálya $A \to aw$ alakú, ahol $A \in N,\ a \in T,\ w \in N^*$.*

3.4.5. példa. A $G = (\{S, B\}, \{a, b\}, \{S \to aB,\ S \to aSB,\ B \to b\}, S)$ nyelvtan Greibach-normálalakú és $L(G) = \{a^n b^n \mid n \geq 1\}$. □

Minden ε-mentes környezetfüggetlen nyelvtanhoz megadható egy vele ekvivalens Greibach-normálalakú nyelvtan. Megadunk egy algoritmust, amely a Chomsky-normálalakban levő $G = (N, T, P, S)$ környezetfüggetlen nyelvtant átalakítja a $G' = (N', T, P', S)$ Greibach-normálalakúvá.

Először rögzítjük a nemterminális jelek egy $A_1, A_2, \ldots, A_n$ sorrendjét úgy, hogy A_1 legyen a kezdőszimbólum. Az algoritmusban a következő jelöléseket használjuk: $x \in N'^+$, $\alpha \in TN'^* \cup N'^+$.

GREIBACH-ALAK(G)
1 $N' \leftarrow N$
2 $P' \leftarrow P$
3 **for** $i \leftarrow 2$ **to** n $\triangleright A_i \to A_j x$, $j < i$ esete.
4 **do for** $j \leftarrow 1$ **to** $i-1$
5 **do** minden $A_i \to A_j x$ és minden $A_j \to \alpha$ alakú szabályra
(ahol α nem kezdődik A_j-vel)
vegyük fel P'-be az $A_i \to \alpha x$ szabályt,
töröljük P'-ből az $A_i \to A_j x$ szabályokat
6 **if** létezik $A_i \to A_i x$ alakú szabály $\triangleright A_i \to A_i x$ esete.
7 **then** vegyük fel N'-be az új B_i nemterminálist,
minden $A_i \to A_i x$ alakú szabályra vegyük fel P'-be a
$B_i \to x B_i$ és $B_i \to x$ szabályokat,
töröljük P'-ből az $A_i \to A_i x$ szabályt,
minden $A_i \to \alpha$ alakú szabályra (ahol α nem kezdődik A_i-vel)
vegyük fel P'-be az $A_i \to \alpha B_i$ szabályt
8 **for** $i \leftarrow n-1$ **downto** 1 $\triangleright A_i \to A_j x$, $j > i$ esete.
9 **do for** $j \leftarrow i+1$ **to** n
10 **do** minden $A_i \to A_j x$ és minden $A_j \to \alpha$ alakú szabályra
vegyük fel P'-be az $A_i \to \alpha x$ szabályt és
töröljük P'-ből az $A_i \to A_j x$ szabályokat,
11 **for** $i \leftarrow 1$ **to** n $\triangleright B_i \to A_j x$ esete.
12 **do for** $j \leftarrow 1$ **to** n
13 **do** minden $B_i \to A_j x$ és minden $A_j \to \alpha$ alakú szabályra
vegyük fel P'-be a $B_i \to \alpha x$ szabályt és
töröljük P'-ből a $B_i \to A_j x$ szabályokat
14 **return** G'

Az algoritmus az első lépésben az $A_i \to A_j x$, $j < i$ alakú szabályokat átalakítja úgy, hogy azok $A_i \to A_j x$, $j \geq i$ vagy $A_i \to \alpha$ alakúak legyenek, ahol ez utóbbi már Greibach-normálalakú. A második lépésben, új nemterminális bevezetésével, kiküszöböli az $A_i \to A_i x$ alakú szabályokat, majd helyettesítésekkel eléri, hogy az $A_i \to A_j x$, $j > i$ és $B_i \to A_j x$ alakú szabályok is Greibach-normálalakúak legyenek.

3.4.6. példa. Alakítsuk át a következő Chomsky-normálalakú szabályokat Greibach-normálalakúvá:

$A_1 \to A_2 A_3 \mid A_2 A_4$
$A_2 \to A_2 A_3 \mid a$
$A_3 \to A_2 A_4 \mid b$
$A_4 \to c$

Az algoritmus lépései:

3–5: Az $A_3 \to A_2A_4$ szabályt kell átalakítani. Erre csak az $A_2 \to a$ szabály alkalmas. Ezért felvesszük a szabályok közé az $A_3 \to aA_4$ szabályt és töröljük az $A_3 \to A_2A_4$ szabályt.
Tehát a szabályok:

$A_1 \to A_2A_3 \mid A_2A_4$
$A_2 \to A_2A_3 \mid a$
$A_3 \to aA_4 \mid b$
$A_4 \to c$

6-7: Az $A_2 \to A_2A_3$ kiküszöbölése a következő szabályokkal történik:

$B_2 \to A_3B_2$
$B_2 \to A_3$
$A_2 \to aB_2$

Tehát, a **6–7.** lépések után, a szabályok:

$A_1 \to A_2A_3 \mid A_2A_4$
$A_2 \to aB_2 \mid a$
$A_3 \to aA_4 \mid b$
$A_4 \to c$
$B_2 \to A_3B_2 \mid A_3$

8–10: Az A_1 baloldalú szabályoknál végzünk helyettesítéseket. Az eredmény:

$A_1 \to aA_3 \mid aB_2A_3 \mid aA_4 \mid aB_2A_4$

11–13: Hasonlóképpen járunk el a B_2 baloldalú szabályokkal:

$B_2 \to aA_4B_2 \mid aA_3A_4B_2 \mid aA_4 \mid aA_3A_4$

Miután kitöröltük a 8–13. lépésekben a helyettesített szabályokat, a következőket kapjuk, amelyek már mind Greibach-alakú szabályok:

$A_1 \to aA_3 \mid aB_2A_3 \mid aA_4 \mid aB_2A_4$
$A_2 \to aB_2 \mid a$
$A_3 \to aA_4 \mid b$
$A_4 \to c$
$B_2 \to aA_4B_2 \mid aA_3A_4B_2 \mid aA_4 \mid aA_3A_4$ □

3.4.7. példa. Nézzünk meg egy másik példát. Megadunk egy nyelvtant a következő nyelv generálására.

$$L = \{a^n b^k c^{n+k} \mid n \geq 0, k \geq 0, n + k > 0\}.$$

Bebizonyítható, hogy a következő nyelvtan generálja L-et.

$$G = \{\{S, R\}, \{a, b, c\}, \{S \to aSc,\ S \to ac,\ S \to R, R \to bRc,\ R \to bc\}, S\}.$$

Először kiküszöböljük az átnevezéseket (itt most csupán egy van), azután megadunk egy vele ekvivalens Chomsky-alakú nyelvtant, majd egy ezzel ekvivalens Greibach-alakút. Az $S \to R$ átnevezés kiküszöbölése után a következő szabályokot kapjuk:

$S \to aSc \mid ac \mid bRc \mid bc$
$R \to bRc \mid bc.$

Bevezetjük az $A \to a, B \to b, C \to c$ szabályokat, majd a terminálisokat helyettesítjük minden szabály jobb oldalán a megfelelő változóval:

$S \to ASC \mid AC \mid BRC \mid BC,$
$R \to BRC \mid BC,$
$A \to a,\ B \to b,\ C \to c.$

Két új változó (D, E) bevezetése után:

$S \to AD \mid AC \mid BE \mid BC,$
$D \to SC,$
$E \to RC,$
$R \to BE \mid BC,$
$A \to a,\ B \to b,\ C \to c.$

Ez már Chomsky-féle normálalak. Induljunk ki ebből a nyelvtanból, miután átírjuk a változókat A_i alakúra, hogy könnyebben alkalmazhassuk az algoritmust. Tehát, a következő átnevezés után

S helyett A_1, A helyett A_2, B helyett A_3, C helyett A_4, D helyett A_5,
E helyett A_6, R helyett A_7,

átnevezés után nyelvtanunk a következő szabályokat tartalmazza:

$A_1 \to A_2A_5 \mid A_2A_4 \mid A_3A_6 \mid A_3A_4,$
$A_2 \to a,\ A_3 \to b,\ A_4 \to c,$
$A_5 \to A_1A_4,$
$A_6 \to A_7A_4,$
$A_7 \to A_3A_6 \mid A_3A_4.$

Az algoritmus 3–5. lépéseinek alkalmazásakor a következő új szabályok jelennek meg:

$A_5 \to A_2A_5A_4 \mid A_2A_4A_4 \mid A_3A_6A_4 \mid A_3A_4A_4$ majd
$A_5 \to aA_5A_4 \mid aA_4A_4 \mid bA_6A_4 \mid bA_4A_4$
$A_7 \to A_3A_6 \mid A_3A_4$, majd
$A_7 \to bA_6 \mid bA_4.$

Tehát:

$A_1 \to A_2A_5 \mid A_2A_4 \mid A_3A_6 \mid A_3A_4,$
$A_2 \to a,\ A_3 \to b,\ A_4 \to c,$
$A_5 \to aA_5A_4 \mid aA_4A_4 \mid bA_6A_4 \mid bA_4A_4$
$A_6 \to A_7A_4,$
$A_7 \to bA_6 \mid bA_4.$

A 6–7. lépéseket átugorjuk, hisz nincs balrekurzív szabály. A 8–10. lépésekben a megfelelő helyettesítések után:

$A_1 \to aA_5 \mid aA_4 \mid bA_6 \mid bA_4$,
$A_2 \to a$,
$A_3 \to b$,
$A_4 \to c$,
$A_5 \to aA_5A_4 \mid aA_4A_4 \mid bA_6A_4 \mid bA_4A_4$
$A_6 \to bA_6A_4 \mid bA_4A_4$,
$A_7 \to bA_6 \mid bA_4$. □

Gyakorlatok

3-1. Adjunk meg egy-egy veremautomatát a következő nyelvek felismerésére:
$L_1 = \{a^n c b^n \mid n \geq 0\}$,
$L_2 = \{a^n b^{2n} \mid n \geq 1\}$,
$L_3 = \{a^{2n} b^n \mid n \geq 0\} \cup \{a^n b^{2n} \mid n \geq 0\}$,

3-2. Adjunk meg egy olyan környezetfüggetlen nyelvtant, amely az $L = \{a^n b^n c^m \mid n \geq 0, m \geq 0\}$ nyelvet generálja, majd írjuk át Chomsky-, illetve Greibach-normálalakúvá. Adjunk meg egy veremautomatát, amely felismeri az L nyelvet.

3-3. Milyen nyelvet generálnak a következő környezetfüggetlen nyelvtanok?
$G_1 = (\{S\}, \{a, b\}, \{S \to SSa, \to b\}, S)$,
$G_2 = (\{S\}, \{a, b\}, \{S \to SaS, \to b\}, S)$

3-4. Adott a következő nyelvtan: $G = (V, T, P, S)$, ahol
$V = \{S\}$,
$T = \{\textit{if, then, else, a, c}\}$,
$P = \{S \to \textit{if a then } S, \quad S \to \textit{if a then } S \textit{ else } S, \quad S \to c\}$,
Mutassuk meg, hogy az *if a then if a then c else c* szónak létezik két különböző legbaloldalibb levezetése.

3-5. Bizonyítsuk be, hogy ha L környezetfüggetlen, akkor
$L^{-1} = \{u^{-1} \mid u \in L\}$
is környezetfüggetlen.

3-6. Adjunk meg egy környezetfüggetlen nyelvtant, amely olyan szavakat generál, amelyben egyenlő számban vannak az a és b betűk.

3-7. Bizonyítsuk be a pumpáló lemma alkalmazásával, hogy az a nyelv, amelynek minden szava ugyanannyi a, b és c betűt tartalmaz, nem környezetfüggetlen.

Feladatok

I-1. A révész problémája
A révésznek át kell szállítania a folyó túlsó partjára egy kecskét, egy káposztát és egy farkast egy olyan csónakkal, amelyben rajta kívül csak egyetlen egyet vihet a kecske, a káposzta és a farkas közül. A farkas nem maradhat a kecskével egyik parton sem, hasonlóan a kecske sem a káposztával. Oldjuk meg a révész problémáját véges automata segítségével. Hány megoldás van?

I-2. Lineáris nyelvtanok
Az olyan $G = (N, T, P, S)$ nyelvtant, amelynek minden szabálya $A \to u_1 B u_2$ vagy $A \to u$ alakú, ahol $A, B \in N,\ u, u_1, u_2 \in T^*$, ***lineáris nyelvtannak*** nevezzük. Amennyiben egy lineáris nyelvtanban minden szabály $A \to Bu$ vagy $A \to v$ alakú, akkor ballineáris nyelvtanról beszélünk. Bizonyítsuk be, hogy minden ballineáris nyelvtan által generált nyelv reguláris.

I-3. Operátornyelvtanok
Egy ε-mentes környezetfüggetlen nyelvtant ***operátornyelvtannak*** nevezünk, ha a szabályai jobb oldalán nincs két nemterminális szimbólum egymás mellett. Igazoljuk, hogy minden ε-mentes környezetfüggetlen nyelvtanhoz megkonstruálható egy vele ekvivalens operátornyelvtan.

I-4. Környezetfüggetlen nyelvek komplementuma és metszete reguláris nyelvvel
Bizonyítsuk be, hogy a környezetfüggetlen nyelvek osztálya nem zárt a komplementumra. Bizonyítsuk be, hogy egy környezetfüggetlen nyelv metszete egy reguláris nyelvvel mindig környezetfüggetlen. Hasonlóan egy környezetfüggetlen nyelv különbsége egy reguláris nyelvvel is.

I-5. Prímnyelvek
Egy környezetfüggetlen nyelvet ***prímnek*** nevezünk, ha nem írható fel nyelvek szorzataként. Milyen körülmények között lehet egy nyelvet egyértelműen felírni prímnyelvek szorzataként?

I-6. Önbeágyazó nyelvek
Egy környezetfüggetlen nyelvet ***önbeágyazónak*** nevezünk, ha van olyan A változója, amelyre $A \overset{*}{\Longrightarrow} \alpha A \beta$, ahol $\alpha, \beta \in (N \cup T)^+$. Bizonyítsuk be, hogy ha egy környezetfüggetlen nyelv nem önbeágyazó, akkor reguláris.

Megjegyzések

A véges automata definíciójában eltértünk a hagyományos értelmezéstől, az átmenetfüggvény helyett az átmenetgráfot használtuk. Ezt a szemléletmódot követtük a veremautomata esetében is. Ez sok esetben hasznosnak bizonyult, nagyban egyszerűsítvén a bizonyításokat.

Az automatákról és a formális nyelvekről sok klasszikus könyv létezik. Ezek közül megemlítjük a következőket: Aho és Ullman két könyve [2, 3] 1972-ből és 1973-ból, Gécseg Ferenc és Peák István [17] angol nyelvű könyve 1972-ből, Salomaa két könyve [45, 46] 1969-ból és 1973-ból, Hopcroft és Ullman [24] könyve 1979-ből, Harrison [22] könyve 1978-ból, Manna [34] könyve, amely 1981-ben magyar fordításban is megjelent. Megemlítjük még Sipser [48] 1997-es könyvét, valamint Rozenberg és Salomaa [43] monográfiáját. Lothaire (francia szerzők közös neve) [31] szókombinatorikai könyvében egyéb típusú automatákról is olvashatunk. Giammarresi és Montalbano cikke [18] az általánosított véges automatákkal foglalkozik. A témakör friss angol nyelvű monográfiája Hopcroft, Motwani és Ullman [23] műve. Német nyelven Asteroth és Baier [4] tankönyvét ajánljuk. A Greibach-féle normálalakra való hozás algoritmusának tömör leírása innen való.

Magyar nyelven is több jegyzet és könyv tárgyalja az automaták és formális nyelvek témáját: Bach Iván [5], a Demetrovics–Denev–Pavlov szerzőhármas [13], Fülöp Zoltán [16], Peák István [39, 40, 41], Révész György [44], Kása Zoltán [29], Iványi Antal (alkotó szerkesztő) [27] könyve, valamint Dömösi–Fazekas–Horváth–Mecsei [14] és Hunyadvári–Manhertz [26] elektronikus kézirata. Jelen rész a [27] könyv 19. fejezetének átdolgozott változata.

A román nyelvű könyvek közül megemlítjük a következőket: Marcus [35], Livovschi–Georgescu–Popovici–Ţăndăreanu [30], Căzănescu [12], Grigoraş [20], Jucan [28], Moldovan [36].

A fordítóprogramokról szóló rész végén egyéb, a témához kapcsolódó könyvekre is találunk utalást.

II. RÉSZ

Fordítóprogramok

4. FEJEZET

Compiler, interpreter

A továbbiakban a *magasszintű programozási nyelvek* fordításával foglalkozunk, és most csak az *imperatív programozási nyelvek* fordítási algoritmusait tanulmányozzuk.

Ha a forrásnyelv egy magasszintű nyelv, akkor a forrásnyelv és a számítógép által végrehajtható gépi kód nagyon különböznek egymástól. Ezt a különbséget kétféleképpen lehet áthidalni.

Az első módszer az, hogy a magasszintű nyelven írt programból egy alacsonyabb szintű (például assembly nyelvű vagy gépi kódú) programot készítünk. Az ilyen átalakítást végző fordítóprogramot *compilernek* nevezzük.

A compiler inputja a *forrásnyelvű program*, vagy röviden a *forrásprogram*, a fordítóprogram ezt lefordítja, és eredményül a *tárgynyelvű programot*, vagy röviden a *tárgyprogramot* adja.

Azt az időt, amikor a fordítás történik, *fordítási időnek*, és azt az időt, amikor a lefordított, futtatható tárgyprogramot a programhoz szükséges adatokkal végrehajtjuk, *futtatási időnek* nevezzük. A fordítási idő és a futtatási idő két egymástól jól elkülöníthető időpont.

A forrásnyelv és a gépi kód közötti különbség áthidalásának második módszere az, hogy készítünk egy olyan gépet, amelyik a magasszintű nyelvet értelmezi, azaz amelyiknek a gépi kódja ez a magasszintű nyelv. Ha ezt a gépet hardver úton valósítjuk meg, akkor ezt a számítógépet *formulavezérlésű számítógépnek*, ha programmal valósítjuk meg, akkor ezt a programot *interpreternek* nevezzük. Az interpreterrel tehát egy olyan kétszintű gépet hozunk létre, amelyiknek az alsó szintje a tényleges fizikai gép, és erre épül a magasszintű nyelvet értelmező, programmal megvalósított magasszintű gép. Az interpreter működésekor a fordítási és a futási időpont egybeesik, és az interpreter nem készít tárgyprogramot.

Látható, hogy míg a compilerek a generált kódot tárgyprogramba helyezik, az interpreterek ezt a kódot azonnal végrehajtják. Mivel a mi témánk a fordítási algoritmusok vizsgálata, szempontunkból, a fordítás szempontjából a compilereket és az interpretereket nem kell megkülönböztetnünk.

4.1. A fordítóprogram szerkezete

A compiler a forrásnyelvű programot egy tárgyprogramra fordítja le. A fordítás folyamatáról, a fordítás eredményéről egy listát is készít, amely a programozó számára visszaigazolja a forrásnyelvű szöveget, tartalmazza a felfedezett hibákat, és információt ad a tárgyprogramról is, például közli az egyes utasításoknak a program elejéhez viszonyított relatív kezdőcímét. A fordítóprogram struktúrája tehát a következő:

compiler(forrásnyelvű program)(tárgyprogram, lista),

ahol a fenti leírás a *program(bemenet)(kimenet)* jelölésmódnak felel meg. Ezt a jelölésmódot alkalmazva, a továbbiakban a compiler struktúráját finomítjuk.

A forrásnyelvű program egy adathordozón, általában egy fájlban, az operációs rendszertől függő formátumban helyezkedik el. A fordítóprogram első lépése ezért egy olyan program végrehajtása lesz, amelyik ezt a forrásnyelvű programot a fordítás számára könnyen hozzáférhető karaktersorozattá alakítja át. Ezt a műveletet végzi el az *input-handler*:

input-handler(forrásnyelvű program)(karaktersorozat).

Az input-handler a forrásnyelvű program egy vagy több rekordját egy pufferben helyezi el, meghívásakor innen olvassa a forrásnyelvű program soron következő sorát, levágva például már a további feldolgozás szempontjából közömbös kocsivissza és soremelés karaktereket is.

Ezekből a forrásnyelvű sorokból készül a lista. A lista összeállítását az *output-handler* végzi, amely a listát szintén a háttértárolón, egy fájlban, az operációs rendszertől függő formátumban helyezi el:

output-handler(forrásnyelvű program, hibák)(lista).

Itt is felhívjuk a figyelmet arra, hogy a fordítóprogramok szinte mindig hibás programokat fordítanak, hibátlan programokkal csak elhanyagolhatóan kevés alkalommal foglalkoznak, ezért a fordítóprogramok íróinak rendkívül nagy figyelmet kell fordítaniuk a lista formájára, a hibajelzés módjára.

Az output-handlerhez hasonlóan, a compiler által készített tárgykódot is háttértárolón, egy fájlban, például relokálható bináris formátumban kell elhelyezni, a formátum természetesen ismét az operációs rendszertől függ. Ezt a műveletet a *code-handler* végzi el:

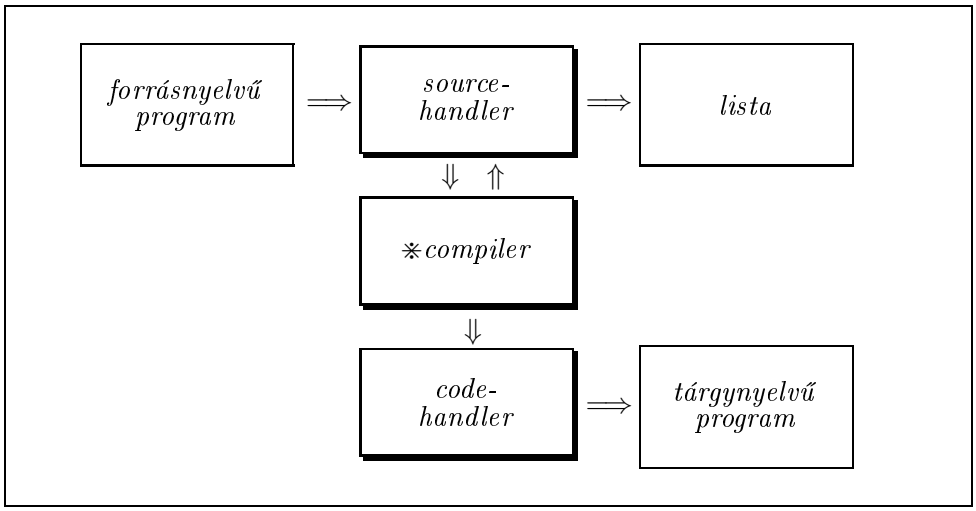
code-handler(tárgykód)(tárgyprogram).

Mivel mind az input-handler, mind az output-handler bemenete a forrásnyelvű program, célszerű ezeket egy programba összefogni. Nevezzük ezt a programot *source-handlernek*:

source-handler(forrásnyelvű program, hibák)(karaktersorozat, lista).

Így tehát a fordítóprogram struktúrája a következő lesz (4.1. ábra):
source-handler(forrásnyelvű program, hibák)(karaktersorozat, lista),
**compiler(karaktersorozat)(tárgykód, hibák)*,
code-handler(tárgykód)(tárgyprogram).

Ez a felbontás nem szekvenciát jelöl, a három programelem nem szekvenciáli-



4.1. ábra. A compiler felépítése

san hajtódik végre. A fenti felbontással a fordítóprogramot három egymástól jól elkülöníthető funkcionális, működési egységre bontottuk fel. Az egyes működési egységek kapcsolatát az egységek bemenete és kimenete jelzi.

A *source-handler* és a *code-handler* végzi el az összes perifériafüggő és az összes operációs rendszertől függő műveletet, a **compiler* most már ténylegesen csak a fordítással foglalkozik. A két handlerrel, elsősorban a számítógéptől, a perifériáktól és az operációs rendszerektől való függőségük miatt, a továbbiakban nem foglalkozunk, bár a fordítóprogramok külső jellemzőit, kezelhetőségét, a felhasználóval való kapcsolatát alapvetően ezek határozzák meg.

A középső programelemnek, a **compiler*-nek két nagy feladatot kell megoldania: elemeznie kell a bemenetén kapott karaktersorozatot, és szintetizálnia kell a tárgykódot. Az analízis a forrásnyelvű program karaktersorozatát részekre bontja és ezt vizsgálja, míg a szintézis az egyes részeknek megfelelő tárgykódokból építi fel a program teljes tárgykódját.

Az *analízis* első feladata az, hogy a karaktersorozatban meghatározza az egyes szimbolikus egységeket, a konstansokat, változókat, kulcsszavakat, operátorokat. Azt a programelemet, amelyik ezt a feladatot végzi, *lexikális elem-*

zőnek nevezzük. A lexikális elemző a karaktersorozatból szimbólumsorozatot készít:

lexikális elemző(karaktersorozat)(szimbólumsorozat, lexikális hibák).

A létrehozott szimbólumsorozat ábrázolása hatékonysági okokból általában kódolt, egy szimbólumot egy típuskód és egy cím határoz meg, a típuskód a szimbólum típusára utal, a cím pedig a szimbólumtáblának az a címe, ahol az elemző a szimbólumhoz tartozó szöveget elhelyezte.

A lexikális elemzőnek kell kiszűrnie a szóköz karaktereket, a forrásnyelvű programba írt kommenteket, mivel ezekből tárgykód nem származik. A magasszintű programnyelvek egy utasítása több sorba is írható, a lexikális elemző feladata egy több sorba írt utasítás összeállítása is.

A lexikális elemző által készített szimbólumsorozat a bemenete a *szintaktikai elemzőnek*. A szintaktikai elemzőnek a feladata a program struktúrájának a felismerése és ellenőrzése. A szintaktikus elemző azt vizsgálja, hogy a szimbólumsorozatban az egyes szimbólumok a megfelelő helyen vannak-e, a szimbólumok sorrendje megfelel-e a programnyelv szabályainak, nem hiányzik-e esetleg valahonnan egy szimbólum. Ez a folyamat hasonlít ahhoz, amikor a magyar nyelvtan órán egy mondat alanyát, állítmányát, tárgyát, határozóit és jelzőit határozzák meg. A szintaktikai elemző működésének az eredménye az elemzett program szintaxisfája, vagy valamilyen ezzel ekvivalens struktúra:

szintaktikai elemző(szimbólumsorozat)(szintaktikusan elemzett program, szintaktikai hibák).

Az analízis harmadik programja a *szemantikai elemző*, amelynek a feladata bizonyos szemantikai jellegű tulajdonságok vizsgálata. A szemantikai elemző feladata például az $\langle azonosító \rangle + \langle konstans \rangle$ kifejezés elemzésekor az, hogy az összeadás műveletének felismerése után megvizsgálja, hogy az $\langle azonosító \rangle$-val megadott szimbólum deklarálva van-e, a $\langle konstans \rangle$-nak van-e értéke, és típusuk azonos-e.

szemantikai elemző(szintaktikusan elemzett program)(elemzett program, szemantikai hibák).

A szemantikai elemző kimenete lesz a szintetizálást végző programok bemenő adata. A szintaxisfa alakjában, vagy például lengyel-formában ábrázolt elemzett programból készül el a tárgyprogram. A szintézis első lépése a kódgenerálás, amelyet a *kódgenerátor* program végez el:

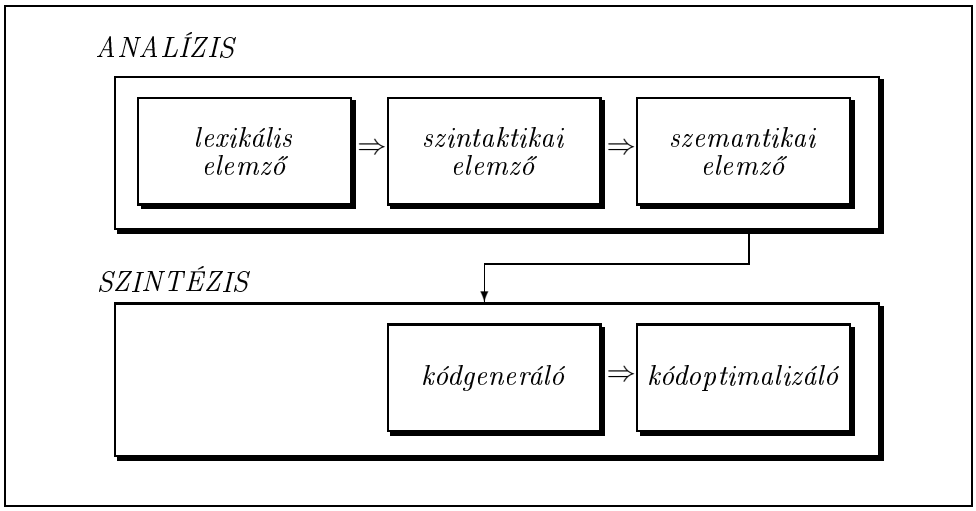
kódgenerátor(elemzett program)(tárgykód).

A tárgykód a forrásnyelvű program egy közbülső programformájának tekinthető. Ez a kód lehet a számítógéptől és az operációs rendszertől független speciális kód, de a legtöbb fordítóprogram tárgykódként az adott számítógép

assembly nyelvű vagy gépi kódú programját állítja elő. A szintetizálás következő lépése a *kódoptimalizálás*:

kódoptimalizáló(tárgykód)(tárgykód).

A kódoptimalizálás a legegyszerűbb esetben a tárgykódban levő azonos programrészek felfedezését és egy alprogramba való helyezését, vagy a hurkok ciklusváltozótól független részeinek megkeresését és a hurkon kívül való elhelyezését jelenti. Bonyolultabbak a gépfüggő kódoptimalizáló programok, amelyek például optimális regiszter-használatot biztosítanak. A fordítóprogramok íróinak a célja, hogy a kódoptimalizáló jobb és hatékonyabb tárgyprogramot állítson elő, mint amit egy (az assembly nyelvű programozásban) gyakorlott programozó készíteni tud.



4.2. ábra. Az analízis és szintézis programjai

A fentiek alapján tehát egy compiler a következő részekre bontható (az elemzést és szintetizálást végző programok a 4.2. ábrán láthatók):

source-handler(forrásnyelvű program, hibák)(karaktersorozat, lista),

lexikális elemző(karaktersorozat)(szimbólumsorozat, lexikális hibák),

szintaktikai elemző(szimbólumsorozat)(szintaktikusan elemzett program, szintaktikai hibák),

szemantikai elemző(szintaktikusan elemzett program)(elemzett program, szemantikai hibák),

kódgenerátor(elemzett program)(tárgykód),

kódoptimalizáló(tárgykód)(tárgykód),

code-handler(tárgykód)(tárgyprogram).

A compilereknek a 4.2. ábrán szereplő struktúrája arra utalhatna, hogy egy compiler öt menetben tudja elvégezni a feladatát. Ez azonban nincs így, vannak olyan compilerek, amelyek egymenetesek, de például az IBM első PL/1 compilere több mint 30 menetből állt.

A fordítóprogramok analízist és szintézist végző részének algoritmusa a következőképpen írható le:

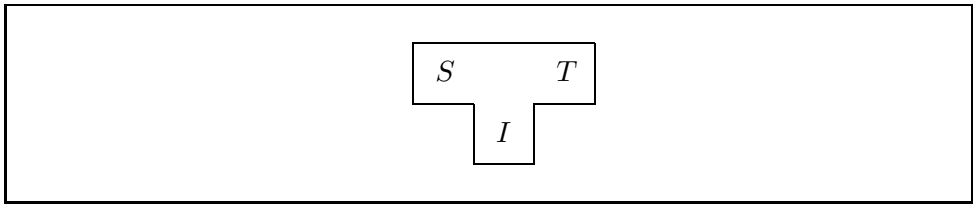
∗FORDÍTÓPROGRAM
1 határozzuk meg a forrásnyelvű program szövegében a lexikális szimbólumokat
2 ellenőrizzük a szimbólumsorozat szintaktikus helyességét
3 ellenőrizzük a szimbólumsorozat szemantikus helyességét
4 határozzuk meg a tárgyprogramba kerülő kódot
5 optimalizáljuk a tárgyprogram kódját

Ezekkel a programokkal foglalkozunk a következő fejezetekben.

4.2. Kezdő lépések

A fordítóprogramok bonyolult programok, célszerű őket valamilyen magasszintű nyelven írni. Egy fordítóprogram három nyelvvel jellemezhető, a forrásnyelvvel, a tárgynyelvvel és azzal a nyelvvel, amelyen a fordítóprogramot megírták, azaz implementálták. Ez a három nyelv természetesen lényegesen különbözhet egymástól. Ha a fordítóprogram nem annak a gépnek a kódjára fordít, mint amelyiken fut, akkor a fordítóprogramot *keresztfordítóprogramnak* nevezzük.

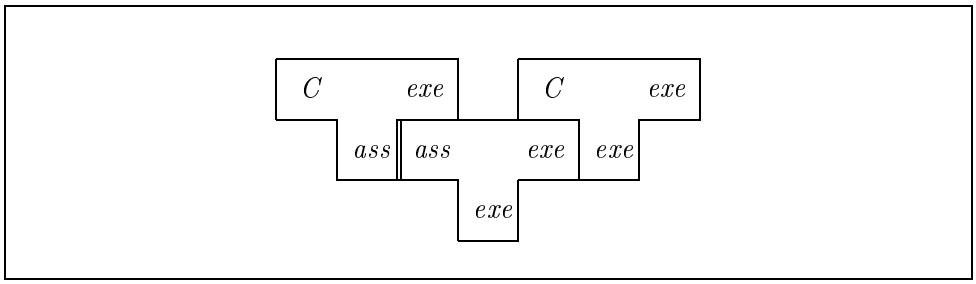
A fordítóprogram három nyelvét egy *T-diagrammal* ábrázolhatjuk, a 4.3. ábrán S-sel a forrásnyelvet, T-vel a tárgynyelvet, I-vel az implementáció nyelvét jelöltük. A programot az I nyelven implementált $S \rightarrow T$ fordítóprogramnak nevezzük. Két T-diagram egymáshoz illeszthető egy olyan ponton, ahol



4.3. ábra. A T-diagram

a két diagramban azonos nyelv van. Az ábrákon az illesztési pontokat dupla vonallal jelöljük. Például, ha van egy *ass* assembly nyelven írt $C \rightarrow exe$ fordítóprogramunk, és van egy *exe* végrehajtható kódú $ass \rightarrow exe$ assemblerünk,

akkor ezzel az assemblerrel elő tudunk állítani egy végrehajtható $C \rightarrow exe$ fordítóprogramot. Ezt a folyamatot T-diagramokkal a 4.4. ábrán látható módon írjuk le.



4.4. ábra. A $C \rightarrow exe$ fordítóprogram

A *bootstrapping*, az indítás, azaz a „kezdő lépések" témakör foglalkozik azokkal a kérdésekkel, hogy

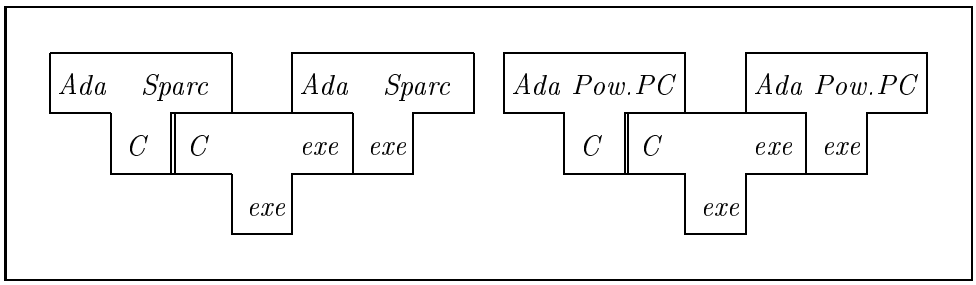
1. hogyan lehet az egyik gépen már működő fordítóprogramot a lehető legkönnyebben átírni úgy, hogy az egy másik gép kódjára fordítson,
2. hogyan fordították le az első magasszintű nyelven megírt fordítóprogramot akkor, amikor még nem volt erre a nyelvre fordítóprogram,
3. hogyan lehet egy olyan fordítóprogramot előállítani, amit a fordítóprogram forrásnyelvén írtak meg,
4. tudja-e egy optimalizált kódot előállító fordítóprogram optimalizálni saját végrehajtási kódját.

Ezek a problémák már az 1950-es években felmerültek.

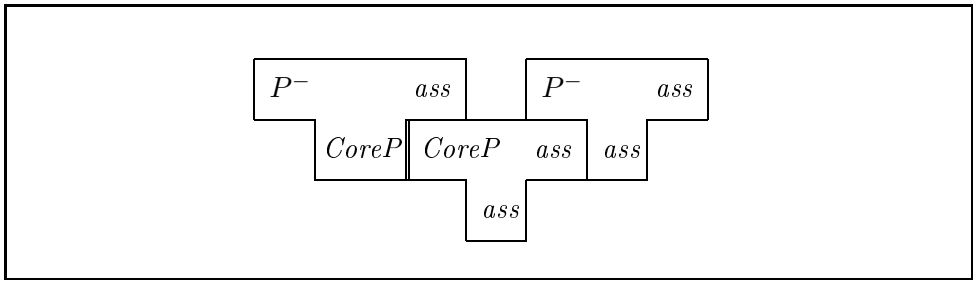
Először vizsgáljuk meg az 1. pontban felvetett problémát, azaz a kereszt-fordítóprogramok előállítását. Például olyan *Ada* fordítóprogramokat szeretnénk készíteni, amelyek a *Sparc* és *PowerPC* gépekre készítenek tárgyprogramot. Tegyük fel, hogy van egy magasszintű nyelvet, például C-t fordító $C \rightarrow exe$ fordítóprogramunk. Ekkor a kereszt-fordítóprogramokat elég ezen a C nyelven megírni, hiszen a rendelkezésre álló C fordítóprogrammal a kívánt programokat elő tudjuk állítani (4.5. ábra).

A 2. és a 3. pontban leírt probléma nem független egymástól, hiszen az első magasszintű nyelven írt fordítóprogramnak nyilván ugyanazt a magasszintű nyelvet kellett fordítania, amelyen a fordítóprogramot megírták. Tehát elég azzal a problémával foglalkoznunk, hogy hogyan lehet egy P nyelven megírt $P \rightarrow exe$ fordítóprogramot készíteni.

A módszer a következő.

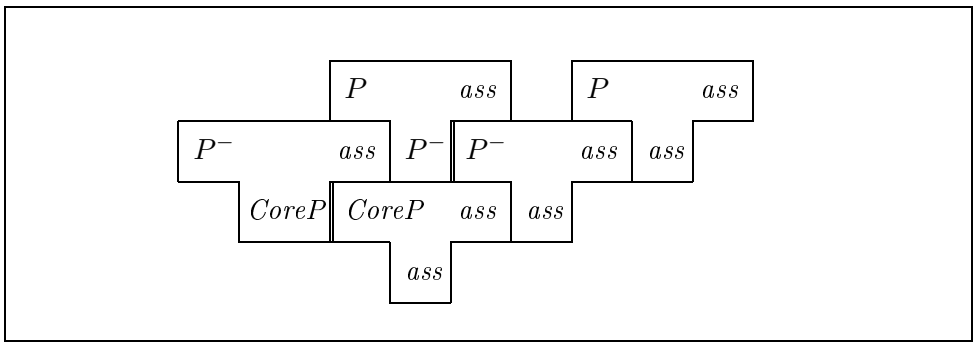


4.5. ábra. Az *Ada* → *Sparc* és az *Ada* → *PowerPC* fordítóprogram

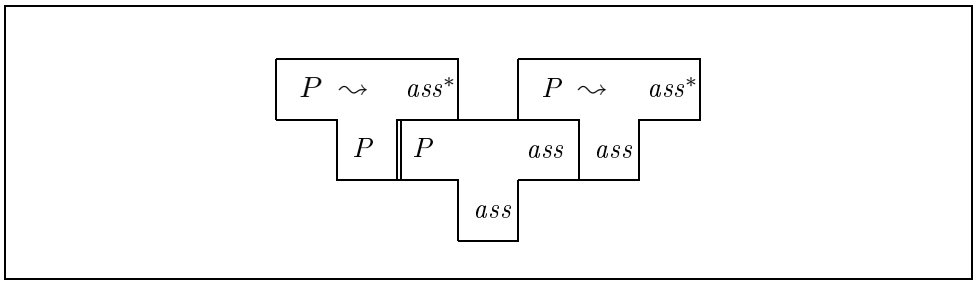


4.6. ábra. A P^- → *ass* fordítóprogram

- Először írjunk egy assemblert, csak azért, hogy ne gépi kódban kelljen programozni, az assembly nyelven írt programokat ezzel az assemblerrel *exe* végrehajtható kódú programra le tudjuk fordítani.
- Ezután a P nyelvet egyszerűsítjük úgy, hogy elhagyunk belőle néhány bonyolultabb dolgot, például néhány bonyolult deklarációt, típust, utasítást. Így megkapjuk a P^- nyelvet. Ezután ezt a nyelvet is tovább (akár több lépésben is) egyszerűsítjük, úgy, hogy végül azért még egy, ha nehezen is, de használható *CoreP* nyelvet kapjunk.
- Most következik egy nehéz programozási feladat, és a fordítóprogram létrehozásában ez az egyedüli nehéz munka, meg kell írni assembly nyelven a *CoreP* → *ass* fordítóprogramot.
- Ezután már használhatjuk implementációs nyelvként a *CoreP* nyelvet, ezen a nyelven már könnyebben megírhatjuk a P^- → *ass* fordítóprogramot, amit az előző lépésben már megírt fordítóprogrammal le tudunk fordítani (4.6. ábra).
- Egyre bővülő, nagyobb hatásfokú programnyelvet, most már a P^--t használjuk implementációs nyelvként, most ezen a nyelven kell megírnunk a P → *ass* fordítóprogramot, amiből könnyen előállíthatjuk az assembly nyelvű P → *ass* programot (4.7. ábra).
- Mivel a P nyelv jobb, mint a P^- nyelv, valószínű, hogy a P nyelven



4.7. ábra. A P^- nyelven implementált $P \to ass$ fordítóprogram

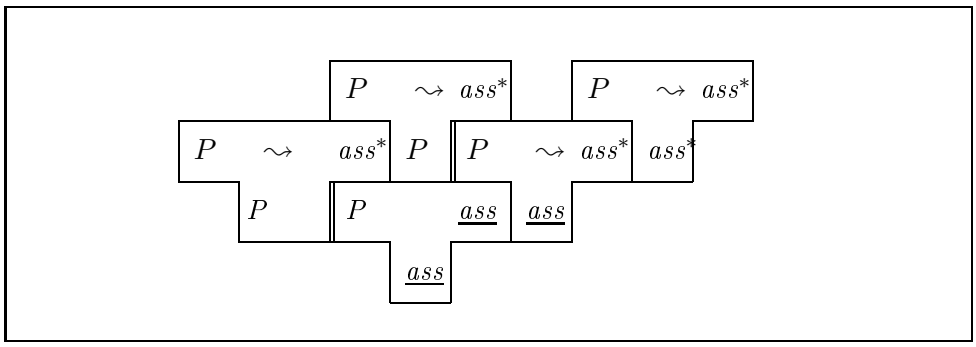


4.8. ábra. A $P \rightsquigarrow ass^*$ fordítóprogram

jobb fordítóprogramot tudunk írni, mint a P^- nyelven. Például, ez a fordítóprogram már optimalizálja a generált *ass* tárgyprogramot. Az optimalizálást jelöljük $\rightsquigarrow$ nyíllal, az optimalizált tárgykódot ass^*-gal, ezt a P nyelven implementált $P \rightsquigarrow ass^*$ fordítóprogramot az előbbi $P \to ass$ fordítóprogrammal már le tudjuk fordítani (4.8. ábra).

Nézzük meg a 4. pontban felvetett problémát, ha már van egy optimalizált kódot előállító fordítóprogramunk, hogyan tudjuk magának a fordítóprogramnak a kódját optimalizálni. Az implementáció kódjának az optimalizálását két lépésben tudjuk megvalósítani.

- Viszonylag nem nagy munkával készíthető egy nem feltétlenül optimális kódú, lassú fordítóprogram, ami ráadásul nagyon gyenge minőségű tárgykódot állít elő. Legyen ez az $\underline{ass}$ kódú $P \to \underline{ass}$ program. Ezzel fordítsuk le a $P \rightsquigarrow ass^*$ programot, eredményül egy optimalizált tárgykódot készítő, de lassú fordítóprogramot kapunk.
- Most ezzel a fordítóprogrammal fordítsuk az eredeti $P \rightsquigarrow ass^*$ fordítóprogramot, a fordítás eredménye egy optimalizált kódú és optimalizált tárgyprogramot előállító fordítóprogram (4.9. ábra).



4.9. ábra. Az optimalizált kódú $P \rightsquigarrow ass^*$ fordítóprogram

Gyakorlatok

4-1. A megadott jelölésrendszert használva, adjuk meg az interpreterek szerkezetét.

4-2. Válasszunk egy programnyelvet, és ezt a nyelvet használva mutassunk néhány olyan programrészletet, amiben lexikális, szintaktikus vagy szemantikus hiba van.

4-3. Adjunk meg olyan szempontokat, amelyek szerint a *kódoptimalizáló* optimalizálhatja a tárgykódot.

5. FEJEZET

A lexikális elemzés

A forrásnyelvű programból a *source-handler* egy karaktersorozatot készít. A lexikális elemző alapvető feladata az, hogy ebben a karaktersorozatban felismerje az összetartozó, a *szimbólumokat* alkotó karaktersorozatokat. A lexikális elemző a karaktersorozatot egy szimbólumsorozatra alakítja át.

Különböző programnyelvekben az azonos, vagy hasonló fogalmakat jelentő szimbolikus egységek lényegesen különbözhetnek, például van, ahol a *program*, és van, ahol a *main* szó jelenti a főprogramot, az egyik nyelvben a *case*, a másikban a *switch* a többirányú elágazás kulcsszava. Különböző szimbólumhoz azonos karakterek vagy karaktersorozatok is tartozhatnak, például a − karakter a legtöbb nyelvben a kivonás jele, de van olyan nyelv is, ahol ez a hosszú azonosító szimbólumok tagolására szolgál, vagy például a ; karakter az egyik nyelvben az utasítás végét, míg egy másik nyelvben két egymásutáni utasítás elkülönítését jelzi.

5.1. A lexikális elemző működése

A lexikális elemző működésének alapelve az, hogy egy szimbólumot mindig a lehető *leghosszabb karaktersorozatból* kell felépíteni, például a *cicamica* szöveg nem nyolc egybetűs, hanem egy nyolcbetűs szimbólum lesz.

A szimbolikus egységek precíz megadása *reguláris nyelvtannal* vagy más néven *Chomsky 3-as típusú nyelvtannal*, *reguláris kifejezésekkel* vagy *determinisztikus véges automatával* írható le.

A lexikális elemző inputjában benne van az összes szóköz és tabulátor jel, hiszen a source-handlerről csak annyit tételeztünk fel, hogy a kocsivissza és soremelés karaktereket hagyja el. A legtöbb programozási nyelv tetszőlegesen sok szóköz és tabulátor karaktert megenged az egyes szimbólumok között. Ezeknek a karaktereknek a fordítás szempontjából a szimbólumok felismerése után már nincs szerepük, ezért ezeket *fehér szóközöknek* nevezzük. A fehér szóközöket a

$(space \mid tab)^*$

reguláris kifejezéssel adhatjuk meg, ahol a *space* a szóköz, a *tab* a tabulátor karaktert jelenti.[1] A fehér szóközökkel a felismerés után a lexikális elemzőnek semmi teendője nincs, ezt a szimbólumot nem kell továbbadnia a szintaktikus elemzőnek.

A lexikális elemzők létrehozásának módszere az, hogy megadjuk a szimbolikus egységek leírását, például a reguláris kifejezések nyelvén, megkonstruáljuk az ekvivalens determinisztikus véges automatát, és elkészítjük a determinisztikus véges automata implementációját. Az elemző a szimbólumot az automata végállapotaival ismeri fel.

A lexikális elemző egy szimbólumhoz egy előre megadott kódot rendel, és ez a kód kerül bele a lexikális elemző outputjába. Egy szimbólumhoz kiegészítő információ is tartozhat, ilyen például a *konstans* szimbólum *típusa*, *értéke*. Ezekre az információkra a következő elemző programnak, a szintaktikus elemzőnek nincs szüksége, de például a szemantikus elemző már a típusokat ellenőrzi, a kódgeneráláshoz pedig ezek az információk feltétlenül szükségesek. Tehát a lexikális elemzőnek a szimbólumokhoz tartozó kiegészítő információkat tárolnia kell, erre a feladatra a lexikális elemzők általában egy *szimbólumtáblát* használnak, és a szimbólum kódja után egy pointert helyeznek el, ami a szimbólumhoz tartozó szimbólumtábla bejegyzésre mutat.

5.1.1. példa. Tegyük fel, hogy a szimbólumok kódjai a következők:

azonosító	1
konstans	200
if	50
then	51
else	52
=	800
:=	700
;	999

Ekkor a lexikális elemző az

if cica12 = 12 then alma := 55 else c1 := 99;

programsorból az

50 1-0001 800 200-0002 51 1-0003 700 200-0002 52 1-0004 700 200-0005 999

sorozatot készíti, ahol 0001 , . . . , 0005 a szimbólumtábla bejegyzésekre mutató pointerek. □

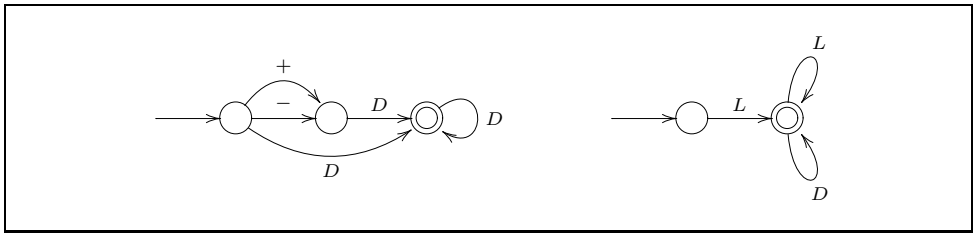
A lexikális elemző outputját, a szimbólumsorozatot arra is használhatjuk, hogy a kódokból visszaállítsuk az eredeti forrásnyelvű programot, rendezett, szépen tabulált, tagolt formában.

[1] Ebben a szakaszban, ellentétben a 2.8. szakasz jelölésével, a reguláris kifejezések „vagy" műveletét a | jellel jelöljük.

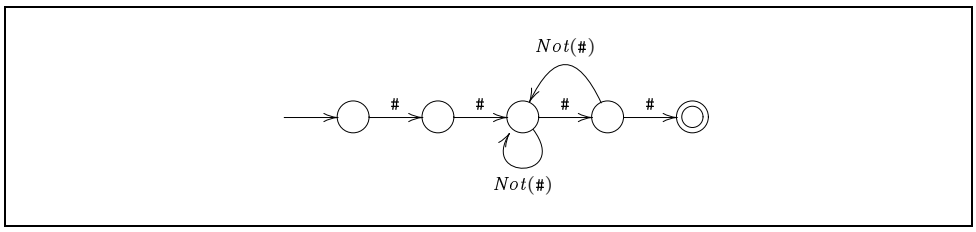
A következőkben néhány példát adunk reguláris kifejezésekre. Megjegyezzük, hogy a szimbólumok leírására azért használunk reguláris kifejezéseket, mert ezekkel a szimbólumok egyszerűbben és könnyebben írhatók le, mint a reguláris nyelvtanokkal.

5.1.2. példa. Vezessük be a következő jelöléseket: jelöljön D egy tetszőleges számjegyet és L egy tetszőleges betűt, azaz $D \in \{0, 1, \ldots, 9\}$, és $L \in \{a, b, \ldots, z, A, B, \ldots, Z\}$, a nem látható karaktereket jelöljük a rövid nevükkel (például *space*, *Eol*, *Eof*), és legyen ε az üres karaktersorozat jele. $Not(a)$ jelentsen egy a-tól, $Not(a|b)$ egy a-tól és b-től különböző karaktert. A reguláris kifejezések:

1. egész szám:
   $(+ \mid - \mid \varepsilon)D^+$
2. egyszerű azonosító szimbólum:
   $L(L \mid D)^*$
3. ## karakterpárokkal határolt komment:
   $\#\#((\# \mid \varepsilon)Not(\#))^*\#\#$
4. /∗ és ∗/ karakterpárokkal határolt komment:
   $/*\,(Not(*))^*\,*^+\,(Not(*|/)Not(*)^**^+)^*/$
5. karakterstring:
   $''(Not('') \mid '' \; '')^* \; ''$.



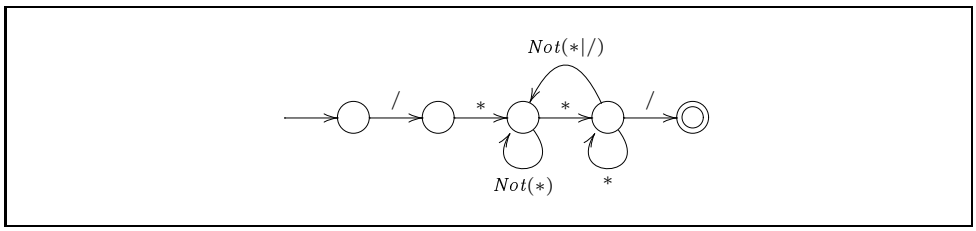
5.1. ábra. Az 5.1.2. példa 1. és 2. automatája



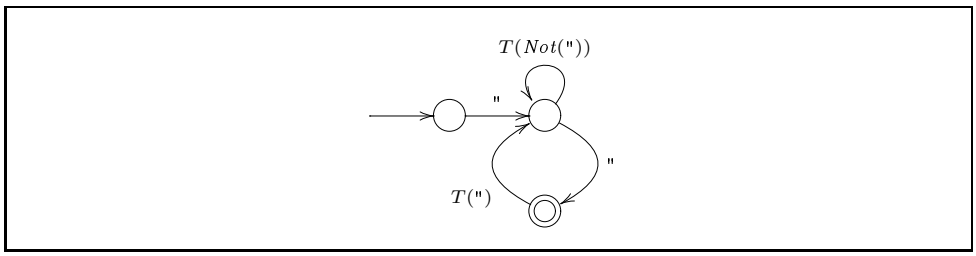
5.2. ábra. Az 5.1.2. példa 3. kifejezésének felismerő automatája

Az 1. és 2. reguláris kifejezéshez konstruálható determinisztikus véges automaták az 5.1. ábrán láthatók. □

A lexikális elemző feladata a szimbólum szövegének a meghatározása, azonban például a fenti 5. reguláris kifejezésben nem minden karakter tartozik a



5.3. ábra. Az 5.1.2. példa 4. kifejezésének felismerő automatája



5.4. ábra. Az 5.1.2. példa 5. kifejezésének felismerő automatája

szimbólumhoz, a kezdő és a befejező " karakterek nem elemei a szimbólumnak. Ezért a lexikális elemzőhöz rendeljünk hozzá egy puffert, egy szimbólum felismerése után a szimbólumot alkotó karakterek ebben a pufferben lesznek. A determinisztikus véges automatát pedig egészítsük ki egy T átviteli függvénnyel, ahol $T(a)$ jelentse azt, hogy az a karakter a pufferbe kerül.

5.1.3. példa. A 3. és 4. reguláris kifejezések automatái (5.2. és 5.3. ábrák), mivel egy kommentet ismernek fel, egyáltalán nem tartalmaznak T függvényt. Az 5. kifejezés automatája (5.4. ábra) például az *"Ez egy ""idézet"""* szövegből az *Ez egy "idézet"* szöveget viszi a pufferbe. □

Most adjuk meg egy determinisztikus véges automatával megadott lexikális elemzés algoritmusát (az egyszerűség kedvéért az egyelemű állapot halmazokat a halmazok egyetlen elemével jelöljük).

Az elemzést végző $A = (Q, \Sigma, \delta, q_0, F)$ determinisztikus véges automata Σ ábécéjét egészítsük ki egy új jellel, jelöljük *egyéb*-bel a nem Σ-beli karaktereket. Ennek megfelelően a δ átmenetfüggvény a következőképpen módosul:

$$\delta'(q,a) = \begin{cases} \delta(q,a), & \text{ha } a \neq \textit{egyéb}\ , \\ \emptyset, & \text{egyébként}\ . \end{cases}$$

Az így kapott A' kiegészített automatával az elemzés a következő lesz:

```
LEX-ELEMEZ(x#, A')
 1  q ← q_0, a ← x első karaktere
 2  s' ← elemez
 3  while a ≠ # és s' = elemez
 4        do if δ'(q, a) ≠ ∅
 5              then q ← δ'(q, a)
 6                   a ← x következő karaktere
 7              else s' ← hiba
 8  if s' = elemez és q ∈ F
 9    then s' ← O.K.
10    else s' ← HIBA
11  return s', a
```

Az algoritmus bemenő paramétere a # jellel lezárt elemezendő karaktersorozat és az elemző automata. Az 1. sorban az elemző állapotát az automata q_0 állapotára állítjuk, és meghatározzuk az elemezendő szöveg első karakterét. Az s' változó az elemző működését jelzi, a 2. sorban a változóba az *elemez* szöveget töltjük. Az 5. sorban az automata állapotátmeneteit hajtjuk végre, és látható, hogy az eredeti automata fenti kiegészítésére azért volt szükség, hogy az automata működése az *egyéb* hibás karakterre is befejeződjék. A 8–10. sorokban *O.K.* azt jelenti, hogy az elemzett karaktersorozat helyes, *HIBA* a lexikális hiba megjelenését jelzi. Sikeres elemzés esetén a a # karaktert, hiba esetén éppen a hibás karaktert tartalmazza.

Megjegyezzük, hogy a LEX-ELEMEZ algoritmus csak egy szimbólumot ismer fel, és ezután működése befejeződik. Egy program nyilvánvalóan sok szimbólumból áll, és egy szimbólum meghatározása után a lexikális elemzést a következő szimbólum meghatározásával kell folytatni, azaz az elemzést az automata kezdőállapotával újra kell indítani. A teljes lexikális elemzés algoritmusának meghatározását a feladatok között tűzzük ki.

5.2. Speciális problémák

5.2.1. Kulcsszavak, standard szavak

Minden programozási nyelvben vannak olyan azonosítók, amelyeknek speciális célra fenntartott nevük, előre megadott jelentésük van, ezek a *kulcsszavak*. A kulcsszavak eredeti jelentésüktől eltérően nem használhatók. Vannak azonban olyan azonosítók is, amelyekre szintén fennáll, hogy előre megadott jelentésük van, de ez a jelentés a programban megváltoztatható. Ezeket a szavakat *standard szavaknak* nevezzük.

A kulcsszavak és a standard szavak száma programnyelvenként változik. A COBOL nyelvben több mint 300 ilyen szó van, a nulla érték jelölésére például három kulcsszó is van: *zero*, *zeros* és *zeroes*.

Foglalkozzunk azzal a problémával, hogy a lexikális elemző hogyan ismeri fel a kulcsszavakat és a standard szavakat, és hogy hogyan különbözteti meg őket a felhasználó által használt azonosítóktól.

A kulcsszavak és standard szavak felismerése akkor nagyon egyszerű, ha azokat speciális karakterekkel írják, vagy speciális elő- és utókarakterekkel jelölik meg.

A standard szavaknak az eredeti értelemtől eltérő felhasználása azonban nemcsak a lexikális elemző feladatát nehezíti meg, hanem a program olvashatóságát is erősen rontja, mint például a következő utasításban:

if if then else = then;

vagy, hogyha egy *begin* és egy *end* nevű eljárást deklarálunk:

```
begin
  begin; begin end; end; begin end;
end;
```

A kulcsszavak kezelésére két módszert adunk.

1. Minden kulcsszót egy reguláris kifejezéssel írunk le, és megadjuk a reguláris kifejezéshez tartozó automata implementációját. Ennek a módszernek a hátránya az, hogy még akkor is nagyon nagyméretű programot kapunk, ha a kulcsszavak leírását az azonos kezdőbetűk szerint összevonjuk.

2. A kulcsszavakat egy külön táblázatban tároljuk. A karaktersorozatban a szavakat egy általános azonosító-felismerővel határozzuk meg, majd egy kereső algoritmust alkalmazva megnézzük, hogy az azonosító benne van-e a táblázatban. Ha igen, akkor a szimbólum egy kulcsszó, ellenkező esetben egy, a felhasználó által megadott azonosító. Ez a módszer nagyon egyszerű, de a keresés sebessége függ a táblázat felépítésétől, a keresési algoritmustól. Egy jól megválasztott leképező függvény és egy lineárisan szétszórt altáblákból felépített kulcsszó-táblázat nagyon hatékony lehet.

Ha a programnyelv lehetővé teszi standard szavak használatát, akkor a lexikális elemző a kulcsszavakra alkalmazott módszerrel meghatározhatja, hogy a vizsgált szimbólum standard szó-e. Az, hogy a standard szó az eredeti jelentésben használt szimbólum, vagy hogy a szimbólumot a felhasználó újraértelmezte, a szimbólum környezetétől függ. Ennek eldöntése a szintaktikus elemző feladata lesz.

5.2.2. Az előreolvasás

Mivel a lexikális elemző a leghosszabb karaktersorozatból álló szimbólum felismerésére törekszik, a szimbólum jobb oldali végpontjának meghatározására gyakran egy vagy több karaktert is előre kell olvasnia. Erre a klasszikus példa a következő két FORTRAN utasítás:

DO 123 K = 9.92
DO 123 K = 9,92

ahol, mivel a FORTRAN nyelvben a szóköz karakterek semmilyen szerepet nem játszanak, a *9* és *92* közötti jel dönti el, hogy az utasítás egy *DO* kulcsszóval kezdődő ciklusutasítás, vagy a *DO123K* azonosítóra vonatkozó értékadás.

A reguláris kifejezések leírásában vezessük be a szimbólum jobb oldali végpontjának a jelölésére a / jelet, és nevezzük ezt *előreolvasási operátornak*. Így a fenti *DO* kulcsszó értelmezése a következő:

$DO\ /\ (betű \mid számjegy)^* = (betű \mid számjegy)^*,$

Ez az értelmezés tehát azt jelenti, hogy a lexikális elemző csak akkor tudja eldönteni, hogy az első két karakteren a *D* és az *O* betű a *DO* kulcsszó, ha előreolvasva, betűk vagy számjegyek, egy egyenlőségjel és ismét betűk vagy számjegyek után egy „ , ” karaktert talál. Az előreolvasási operátor azt is jelenti, hogy a következő szimbólum keresését a *DO* utáni karakterrel kell kezdeni. Megjegyezzük, hogy az elemző ezzel az előreolvasással a *DO* szimbólumot azonosítja akkor is, ha a *DO* után programhiba van, mint például a *DO2A=3B,* karaktersorozatban, de helyes értékadó utasításban sohasem fogja az azonosító első két *D* és *O* karakterét a *DO* kulcsszónak értelmezni.

A fenti példánál bonyolultabb esetek is előfordulhatnak, mint például a következő programsorban:

DO 123 I = (P12(3,"),)")),5

A lexikális elemzőnek fel kell ismernie, hogy az egyenlőség jel után a két paraméteres *P12* függvény *P12(3,"),)")* értéke van, az első paraméter *3*, a második egy idézőjelek közé tett karaktersorozat. Ezután következik egy vessző, és ez a vessző fogja azt jelenteni, hogy a *DO* egy kulcsszó, a függvényérték a ciklusváltozó kezdőértéke, és *5* a végérték.

Az előreolvasás karakterszámát a programozási nyelv leírásából lehet meghatározni. A modern programozási nyelveknél egy szimbólum felismeréséhez a szimbólum karakterei után egy, pesszimális esetben négy karakter előreolvasása szükséges.

5.2.1. példa.
A C++ nyelv néhány kétkarakteres szimbóluma: ++, +=, --, -=, ->, >=, >>, háromkarakteres szimbólumai: <<=, >>=.

A Java nyelv háromkarakteres szimbólumai: >>=, >>>, <<=,
és egy négykarakteres szimbólum: >>>=. □

5.2.3. Direktívák

A forrásnyelvekben a *direktívák* a fordítóprogram működésének vezérlésére szolgálnak. A direktívákat és a direktívák operandusaiban szereplő szimbólumokat is a lexikális elemzőnek kell meghatároznia, de ezekkel az elemzőnek további teendői is vannak.

Ha a direktíva például egy feltételes fordítás *if* direktívája, akkor fel kell ismernie a direktíva összes szimbólumát, majd ki kell értékelnie az elágazás feltételét. Ha ez *false*, akkor a további sorokban szereplő szimbólumokat nem szabad elemeznie egészen addig, amíg egy *else*, vagy az *if* végét jelentő *endif* direktívát nem talál. Ez azt jelenti, hogy a lexikális elemzőnek már szintaktikus és szemantikus ellenőrzéseket is kell végeznie, és kódjellegű információt kell előállítania. A feladatot különösen bonyolíthatja, ha a nyelv lehetőséget ad a feltételek skatulyázására.

Egy másik tipikus direktíva a makróhelyettesítés, vagy egy adott nevű állomány bemásolása a forrásnyelvű szövegbe, aminek a végrehajtása szintén távol áll a lexikális elemző eredeti alapfeladatától.

A legtöbb fordítóprogramban ezeket a problémákat úgy oldják meg, hogy a szintaktikus elemző előtt egy előfeldolgozó programot működtetnek, amelynek a feladata a direktívák által megadott feladatok végrehajtása.

5.2.4. Hibakezelés

Ha a lexikális elemző egy karaktersorozatnak nem tud egy szimbólumot sem megfeleltetni, akkor azt mondjuk, hogy a karaktersorozatban *lexikális hiba* van.

A lexikális hibákat legtöbbször az okozza, hogy illegális karakterek kerülnek a szövegbe, karakterek felcserélődnek vagy esetleg karakterek hiányoznak. Mivel nem célszerű egy lexikális hiba detektálásakor a fordítást megszakítani, valamilyen hibaelfedő algoritmust kell alkalmazni, amelynek az a célja, hogy az elemzés a lehető legkevesebb karakter kihagyásával folytatódjon.

Tegyük fel, hogy az elemző a *defiξne* karaktersorozatot vizsgálja, ahol a ξ egy nem megengedett karakter. Ekkor a ξ karakterre egy hibajelzést ad, és az egyik lehetőség az, hogy nem foglalkozik a már beolvasott karakterekkel, az elemzést a következő, még nem vizsgált karakterrel folytatja. Az elemző így az *ne* karakterekből például egy logikai reláció szimbólumot ismer fel. Egy másik lehetőség az, hogy egyszerűen kihagyja, vagy egy tetszőleges karakterrel, általában a *space* szóköz karakterrel helyettesíti az illegális karaktert. Így a

fenti szövegben vagy a *define* szimbólumot, vagy a *defi* és *ne* szimbólumokat határozza meg.

Hiányzó karaktereknek gyakran az a hatásuk, hogy a lexikális elemző nem tudja a szimbólumokat elkülöníteni egymástól. Például, ha az *alma+112* karaktersorozatból a + jel kimarad, akkor az elemző minden hibajelzés nélkül az *alma112* azonosítót ismeri fel. Ha a *112+alma*-ból hiányzik a + jel, akkor a lexikális elemző ezt egy *112* konstansra és egy *alma* azonosítóra bontja, és azt, hogy közülük egy műveleti jel hiányzik, majd csak a szintaktikus elemző fogja felfedezni.

Hasonlóan, a következő C++ programsor is meglepő eredményt ad, a *"sárga"* és *"narancs"* szövegek közül kimaradt a vessző:

*char *szín[] = { "piros" , "kék", "sárga" "narancs" } ;*

a harmadik elem a *sárganarancs* lesz.

Vannak olyan lexikális hibák is, amelyeknek a kezelésére különösen nagy gondot kell fordítani. Ilyenek a karakterstringek és a kommentek befejező karaktereinek hiányai, ezek a hibák gyakran azt okozzák, hogy a program további része, egészen a program végéig, karakterstring vagy komment lesz.

Gyakorlatok

5-1. Adjuk meg egy nyelv kommentjének reguláris kifejezését, ha a kommentet a -- jelek kezdik, és a komment a sorvége karakterig tart.

5-2. Vizsgáljuk meg, hogy az előző példában szereplő reguláris kifejezést hogyan kell megváltoztatni, ha a kommentek skatulyázhatók.

5-3. Írjunk pozitív valós számokhoz olyan reguláris kifejezést, ami nem engedi meg az elő- és utónullákat. Adjuk meg a reguláris kifejezéshez tartozó determinisztikus véges automatát is.

5-4. Írjunk egy olyan programot, ami a lexikális elemzés szimbólumsorozat kimenetéből visszaállítja az eredeti programszöveget. Ügyeljünk a visszaállított karaktersorozatok helyes és szép pozicionálására.

6. FEJEZET

A szintaktikai elemzés

A környezetfüggetlen nyelvtanokkal leírható programnyelvi tulajdonságok vizsgálatát *szintaktikai elemzésnek* nevezzük. A programnyelv szintaktikájának azon követelményei, amelyek nem írhatók le környezetfüggetlen nyelvtannal, a *statikus szemantikát* alkotják. Ezeknek a tulajdonságoknak az ellenőrzésével a *szemantikai elemző* program foglalkozik.

Ebben a fejezetben a szintaktikai elemzésekkel foglalkozunk, és itt nyelvtan alatt mindig környezetfüggetlen nyelvtant, vagy más néven *Chomsky 2-es típusú nyelvtant* értünk, és feltesszük, hogy a környezetfüggetlen nyelvtan *kiterjesztett nyelvtan*, azaz helyettesítési szabályai

$$A \to \alpha \quad (A \in N,\ \alpha \in (N \cup T)^*)$$

alakúak.

6.1. A szintaktikai elemzés alapfogalmai

Most megadjuk a formális nyelvek elméletének azokat az alapfogalmait, amelyeket a szintaktikai elemzésben használni fogunk.

6.1.1. értelmezés. *Legyen $G = (N, T, P, S)$ egy nyelvtan. Ha $S \overset{*}{\Longrightarrow} \alpha$, ahol $\alpha \in (N \cup T)^*$, akkor az α-t **mondatformának** nevezzük. Ha $S \overset{*}{\Longrightarrow} x$, ahol $x \in T^*$, akkor az x a nyelvtan által generált nyelv egy **mondata**.*

Az elemzés szempontjából a mondatnak kiemelt szerepe van, hiszen a programozó által írt program is terminális szimbólumok sorozata, de ez a szimbólumsorozat csak akkor lesz a nyelvnek egy mondata, ha a program szintaktikusan helyes.

6.1.2. értelmezés. *Legyen a $G = (N, T, P, S)$ nyelvtannak $\alpha = \alpha_1\beta\alpha_2$ egy mondatformája ($\alpha, \alpha_1, \alpha_2, \beta \in (N \cup T)^*$). A β-t az α egy **részmondatának** nevezzük, ha van olyan $A \in N$ szimbólum, amelyre $S \overset{*}{\Longrightarrow} \alpha_1 A \alpha_2$ és $A \overset{*}{\Longrightarrow} \beta$. Az α-nak β egy **egyszerű részmondata**, ha a fentiekben az $A \to \beta \in P$ teljesül.*

Megjegyezzük, hogy minden mondat egyben mondatforma is, és felhívjuk a figyelmet arra, hogy a mondatforma „részét" is részmondatnak, és nem részmondatformának nevezzük. A legbaloldalibb egyszerű részmondat fontos szerepet játszik a szintaktikai elemzésben, külön elnevezést is kapott.

6.1.3. értelmezés. *Egy mondatforma legbaloldalibb egyszerű részmondatát a mondatforma* ***nyelének*** *nevezzük.*

A mondat *szintaxisfájának* levelei a nyelvtan terminális szimbólumai, a szintaxisfa többi pontja a nemterminális szimbólumokat reprezentálja, a gyökérelem pedig a nyelvtan kezdőszimbóluma.

Egy nemegyértelmű nyelvtanban van legalább egy olyan mondat, amelyhez több szintaxisfa tartozik. Ez az elemzés szempontjából azt jelenti, hogy ezt a mondatot többféleképpen is lehet elemezni, azaz a különböző elemzésekhez különböző tárgyprogramok is tartozhatnak. Ezért fordítóprogramokkal csak az *egyértelmű nyelvtanok* által generált nyelvek fordítását végezzük el.

A továbbiakban feltesszük azt is, hogy a G nyelvtanra a következő feltételek teljesülnek:

1. a nyelvtan *ciklusmentes,* azaz nem tartalmaz $A \overset{+}{\Longrightarrow} A$ $(A \in N)$ helyettesítési szabálysorozatot,
2. a nyelvtan *redukált*, azaz nincs benne „felesleges" nemterminális szimbólum, minden nemterminális szimbólum előfordul legalább egy levezetésben, és minden nemterminális szimbólumból levezethető legalább egy mondatnak egy része, azaz minden $A \in N$-re fennáll, hogy $S \overset{*}{\Longrightarrow} \alpha A \beta \overset{*}{\Longrightarrow} \alpha y \beta \overset{*}{\Longrightarrow} xyz$, ahol $A \overset{*}{\Longrightarrow} y$ és $|y| > 0$ $(\alpha, \beta \in (N \cup T)^*,\ x, y, z \in T^*)$.

6.2. A szintaktikai elemzési módszerek

A programozó által írt programot a lexikális elemző egy terminális szimbólumokból álló sorozattá alakítja, ez a terminálisokból álló sorozat a szintaktikai elemzés bemenete. A szintaktikai elemzés feladata eldönteni azt, hogy ez a szimbólumsorozat a nyelv egy mondata-e. A szintaktikai elemzőnek ehhez például meg kell határoznia a szimbólumsorozat szintaxisfáját, ismerve a szintaxisfa gyökérelemét és a leveleit, elő kell állítania a szintaxisfa többi pontját és élét, vagyis meg kell határoznia a program egy levezetését.

Ha ez sikerül, akkor azt mondjuk, hogy a program eleme a nyelvnek, azaz a program ***szintaktikusan helyes.***

A továbbiakban csak a *balról-jobbra* haladó elemzésekkel foglalkozunk, azaz azokkal az elemzésekkel, amelyek a program jeleit balról jobbra olvasva dolgozzák fel.

A szintaxisfa belső részének felépítésére több módszer létezik. Az egyik az, amikor az S szimbólumból kiindulva építjük fel a szintaxisfát, ezt *felülről-lefelé* haladó elemzésnek nevezzük. Ha a szintaxisfa építése a levelekből kiindulva halad az S szimbólum felé, akkor *alulról-felfelé* elemzésről beszélünk.

A felülről-lefelé haladó modern elemzési módszerekkel a 6.3. szakaszban foglalkozunk, az „igazi" compilerekben jelenleg is használt alulról-felfelé haladó elemzéseket pedig a 6.4. szakaszban tárgyaljuk.

6.3. $LL(1)$ elemzés

Felülről-lefelé elemezve a nyelvtan kezdőszimbólumától, a szintaxisfa gyökérpontjától indulunk el, és így próbáljuk felépíteni a szintaxisfát. A célunk az, hogy a szintaxisfa levelein az elemezendő programszöveg terminális szimbólumai legyenek.

Most először áttekintjük azokat a fogalmakat, amelyeket a felülről-lefelé elemzésben használunk, majd a táblázatos $LL(1)$ elemzéseket és a rekurzív leszállás módszerét tanulmányozzuk.

6.3.1. Az $LL(k)$ nyelvtanok

Mivel felülről-lefelé építjük a szintaxisfát és a szövegben balról-jobbra haladunk, ezért arra kell törekednünk, hogy az elemzéskor kapott mondatformák bal oldalán a lehető leghamarabb megjelenjenek az elemezendő szöveg terminálisai.

6.3.1. értelmezés. *Ha $A \to \alpha \in P$, akkor az $xA\beta$ mondatforma ($x \in T^*$, $\alpha, \beta \in (N \cup T)^*$) **legbaloldalibb helyettesítése** $x\alpha\beta$, azaz*

$$xA\beta \underset{legbal}{\Longrightarrow} x\alpha\beta \ .$$

6.3.2. értelmezés. *Ha az $S \overset{*}{\Longrightarrow} x$ ($x \in T^*$) levezetésben minden helyettesítés legbaloldalibb helyettesítés, akkor ezt a levezetést **legbaloldalibb levezetésnek** nevezzük, és így jelöljük:*

$$S \underset{legbal}{\overset{*}{\Longrightarrow}} x \ .$$

A legbaloldalibb levezetésben a terminális szimbólumok a mondatforma bal oldalán jelennek meg, ezért a felülről-lefelé levezetésekben mindig legbaloldalibb helyettesítéseket alkalmazunk, így a felülről-lefelé elemzés a legbaloldalibb levezetésnek felel meg. Ezért a továbbiakban, amikor felülről-lefelé elemzésről lesz szó, a nyilak alá már nem is írjuk oda a „*legbal*" szót.

A felülről-lefelé elemzés egyik módszere az lehetne, hogy előállítjuk az összes lehetséges szintaxisfát. Egy szintaxisfa levelein levő szimbólumokat balról-jobbra olvasva a nyelv egy mondatát kapjuk meg. Ha az elemezendő szöveg megegyezik valamelyik mondattal, akkor a mondathoz tartozó szintaxisfáról az elemzés lépései már leolvashatók. Ezt a módszert természetesen nem célszerű és nem is lehet a gyakorlatban megvalósítani.

A gyakorlatban megvalósítható módszer a következő:

Kiindulunk a kezdőszimbólumból, és megpróbálunk legbaloldalibb helyettesítések egymás utáni alkalmazásával eljutni az elemezendő szöveghez. A szintaxisfa építésekor azonban egyáltalán nem biztos, hogy jó helyettesítési szabályokat alkalmazunk, mert például egy lépés után előfordulhat, hogy a következő lépésben nem találunk alkalmazható szabályt, vagy a mondatforma elejére kerülő terminális szimbólumok nem egyeznek meg az elemezendő szöveg terminális szimbólumaival. A terminális szimbólumokra a következőket állíthatjuk:

6.3.3. tétel. *Ha* $S \overset{*}{\Longrightarrow} x\alpha \overset{*}{\Longrightarrow} yz$ $(\alpha \in (N \cup T)^*,\ x, y, z \in T^*)$ *és* $|x| = |y|$, *akkor* $x = y$.

A tétel állítása triviális, egy mondatforma bal oldalán a terminálisokból álló x sorozatot a környezetfüggetlen nyelvtan helyettesítési szabályai nem változtathatják meg.

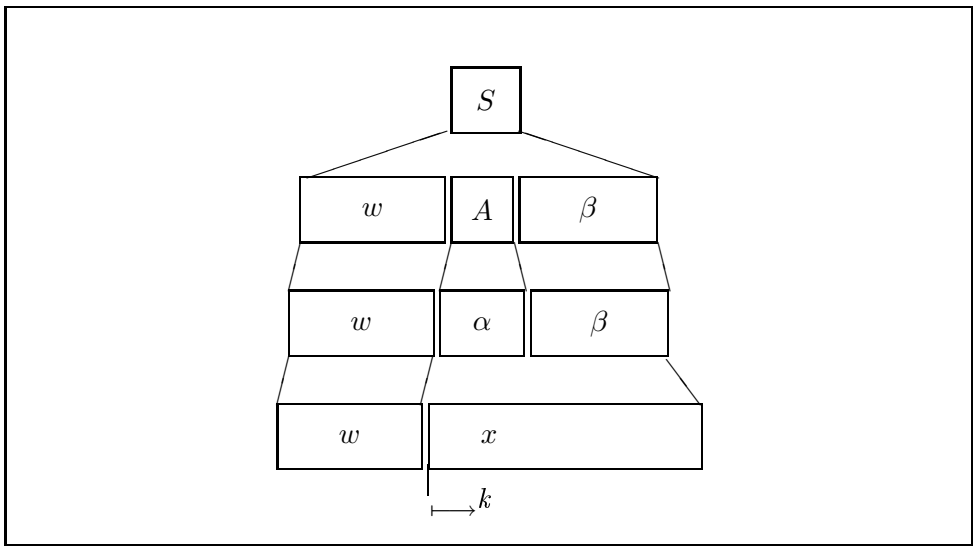
Ez a „kezdőszelet-egyeztetés" arra használható, hogy megállapíthassuk, ha a szintaxisfa építésekor a bal oldali terminálisok nem egyeznek meg az elemezendő szöveg bal oldalán álló terminálisokkal, akkor a szintaxisfa építése biztosan rossz irányban halad. Ekkor egy lépést vissza kell lépni, és ott egy másik helyettesítési szabályt kell alkalmazni, és még egy lépést vissza kell lépni akkor, ha az adott pontban már nincs több alkalmazható szabály.

Az általános felülről-lefelé elemzést tehát visszalépéses algoritmussal lehet megvalósítani. A visszalépések azonban rendkívül lelassíthatják az elemzést, ezért a továbbiakban csak olyan nyelvtanokkal foglalkozunk, amelyekre visszalépés nélküli elemzések adhatók meg.

Az $LL(k)$ nyelvtanok alaptulajdonsága az, hogy ha az $S \overset{*}{\Longrightarrow} wx$ $(w, x \in T^*)$ legbaloldalibb levezetés építésekor eljutunk az $S \overset{*}{\Longrightarrow} wA\beta$ mondatformáig $(A \in N,\ \beta \in (N \cup T)^*)$, és az $A\beta \overset{*}{\Longrightarrow} x$-t szeretnénk elérni, akkor az A-ra alkalmazható $A \to \alpha$ helyettesítést egyértelműen meghatározhatjuk, ha ismerjük az x első k darab szimbólumát.

A k szimbólum előreolvasására megadjuk az $Első_k$ függvényt:

6.3.4. értelmezés. *Legyen* $\boldsymbol{Első_k(\alpha)}$ $(k \geq 0,\ \alpha \in (N \cup T)^*)$ *az* α*-ból levezethető szimbólumsorozatok* k *hosszúságú kezdő terminális sorozatainak hal-*



6.1. ábra. Az $LL(k)$ nyelvtan.

maza, azaz

$$Első_k(\alpha) = \{x \mid \alpha \overset{*}{\Longrightarrow} x\beta \text{ és } |x| = k\} \cup \{x \mid \alpha \overset{*}{\Longrightarrow} x \text{ és } |x| < k\}$$

$$(x \in T^*,\ \beta \in (N \cup T)^*) .$$

Tehát az $Első_k(x)$ halmaz az x első k darab szimbólumát, $|x| < k$ esetén pedig a teljes x-t tartalmazza. Ha $\alpha \overset{*}{\Longrightarrow} \varepsilon$, akkor természetesen $\varepsilon \in Első_k(\alpha)$.

6.3.5. értelmezés. *A G nyelvtan* ***LL(k) nyelvtan*** *($k \geq 0$), ha bármely két*

$$S \overset{*}{\Longrightarrow} wA\beta \Longrightarrow w\alpha_1\beta \overset{*}{\Longrightarrow} wx ,$$
$$S \overset{*}{\Longrightarrow} wA\beta \Longrightarrow w\alpha_2\beta \overset{*}{\Longrightarrow} wy$$

($A \in N,\ x, y, w \in T^*,\ \alpha_1, \alpha_2, \beta \in (N \cup T)^*$) levezetésre

$$Első_k(x) = Első_k(y)$$

esetén

$$\alpha_1 = \alpha_2 .$$

A fenti értelmezés szerint, ha egy nyelvtan $LL(k)$ nyelvtan, akkor a már elemzett w utáni k darab terminális szimbólum az A-ra alkalmazható helyettesítési szabályt egyértelműen meghatározza (6.1. ábra).

Az értelmezésből az is látható, hogy ha egy nyelvtan $LL(k_0)$ nyelvtan, akkor minden $k > k_0$-ra is $LL(k)$ nyelvtan. Ha $LL(k)$ nyelvtanról beszélünk, akkor k alatt mindig azt a legkisebb k-t értjük, amelyre az értelmezésben megadott tulajdonság teljesül.

6.3.6. példa. A következő nyelvtan egy $LL(1)$ nyelvtan. Legyen $G = (\{A, S\}, \{a, b\}, P, S)$, ahol a helyettesítési szabályok a következők:
$S \to AS \mid \varepsilon$
$A \to aA \mid b$
Az S szimbólumra az $S \to AS$ szabályt kell alkalmazni, ha az elemezendő szöveg következő szimbóluma a vagy b, és az $S \to \varepsilon$ szabályt, ha a következő szimbólum a # jel. □

Nem minden környezetfüggetlen nyelvtan $LL(k)$ nyelvtan. Például a következő nyelvtan semmilyen k-ra sem $LL(k)$ nyelvtan.

6.3.7. példa. A $G = (\{A, B, S\}, \{a, b, c\}, P, S)$ nyelvtan helyettesítési szabályai a következők:
$S \to A \mid B$
$A \to aAb \mid ab$
$B \to aBc \mid ac$
Az $L(G)$ az $a^i b^i$ és $a^i c^i$ $(i \geq 1)$ alakú mondatokat tartalmazza. Az $a^{k+1}b^{k+1}$ elemzésének már a kezdetekor sem lehet eldönteni k szimbólum előreolvasásával, hogy az $S \to A$ és az $S \to B$ közül melyik helyettesítést kell először alkalmazni, mivel minden k-ra $\mathit{Első}_k(a^k b^k) = \mathit{Első}_k(a^k c^k) = a^k$. □

Az $LL(k)$ nyelvtan értelmezése szerint, ha legbaloldalibb helyettesítésekkel a $wA\beta$ mondatformát kaptuk, akkor a w-t követő k darab terminális szimbólum egyértelműen meghatározza az A-ra alkalmazható szabályt. Ezt mondja ki a következő tétel.

6.3.8. tétel. *A G nyelvtan akkor és csak akkor $LL(k)$ nyelvtan, ha minden*

$$S \overset{*}{\Longrightarrow} wA\beta, \text{ és } A \to \gamma \mid \delta \quad (\gamma \neq \delta,\ w \in T^*,\ A \in N,\ \beta, \gamma, \delta \in (N \cup T)^*)$$

esetén

$$\mathit{Első}_k(\gamma\beta) \cap \mathit{Első}_k(\delta\beta) = \emptyset \ .$$

Ha a nyelvtanban az A-ra van egy $A \to \varepsilon$ szabály is, akkor a $\mathit{Első}_k$ halmazokban a β-ból származó terminális sorozatok k hosszúságú kezdősorozatai is szerepelnek. Ez pedig azt jelenti, hogy az $LL(k)$ tulajdonság eldöntéséhez nem elegendő csak a szabályokat vizsgálni, hanem a nem feltétlenül véges darabszámú levezetéseket is figyelembe kell venni.

A gyakorlatban jól használható vizsgálati módszert csak az $LL(1)$ nyelvtanokra lehet adni. Ehhez egy új fogalmat értelmezünk, egy szimbólumsorozatot követő k hosszúságú terminális sorozatok halmazát.

6.3.9. értelmezés. ***Követő$_k$(β)***$= \{x \mid S \overset{*}{\Longrightarrow} \alpha\beta\gamma$ *és* $x \in$ *Első*$_k(\gamma)\}$*, és ha* $\varepsilon \in$ *Követő*$_k(\beta)$*, akkor legyen* *Követő*$_k(\beta) =$ *Követő*$_k(\beta)\backslash\{\varepsilon\}\cup\{\#\}$ $(\alpha, \beta, \gamma \in (N \cup T)^*,\ x \in T^*)$.

Az értelmezés második részében levő átalakítás azért szükséges, mert ha az $\alpha\beta\gamma$ levezetésben a β után nem áll semmilyen szimbólum, azaz $\gamma = \varepsilon$, akkor a β utáni jel csak a mondatokat lezáró # jel lehet.

A *Követő*$_1(A)$ $(A \in N)$ tehát azokat a terminális szimbólumokat tartalmazza, amelyek az

$$S \overset{*}{\Longrightarrow} \alpha A\gamma \overset{*}{\Longrightarrow} \alpha Aw\ (\alpha, \gamma \in (N \cup T)^*,\ w \in T^*)$$

levezetésben közvetlenül az A szimbólum mögött állhatnak.

6.3.10. tétel. *A* G *nyelvtan akkor és csak akkor* $LL(1)$ *nyelvtan, ha minden* A *nemterminális szimbólum* $A \to \gamma \mid \delta$ *helyettesítési szabályaira*

$$\textit{Első}_1(\gamma\textit{Követő}_1(A)) \cap \textit{Első}_1(\delta\textit{Követő}_1(A)) = \emptyset\ .$$

A tételben az *Első*$_1(\gamma$*Követő*$_1(A))$ kifejezés azt jelenti, hogy a γ-t a *Követő*$_1(A)$ halmaz minden elemével külön-külön konkatenálni kell, és az így kapott halmaz minden elemére alkalmazni kell az *Első*$_1$ függvényt.

Látható, hogy a 6.3.10. tétel jól használható annak eldöntésére, hogy egy nyelvtan vajon $LL(1)$-es-e, hiszen legfeljebb csak annyi halmazt kell a vizsgálathoz előállítani, ahány szabálya van a nyelvtannak.

A továbbiakban az $LL(1)$ nyelvtanok által meghatározott $LL(1)$ nyelvekkel foglalkozunk, az $LL(1)$ nyelvek elemzési módszereit vizsgáljuk. Az egyszerűbb jelölés érdekében az *Első*$_1$ és *Követő*$_1$ függvények nevéből az indexet elhagyjuk.

Az *Első*(α) halmaz elemei a következő algoritmussal határozhatók meg.

```
ELSŐ(α)
1  if α = ε
2     then F ← {ε}
3  if α = a, ahol a ∈ T
4     then F ← {a}
```

```
 5  if α = A, ahol A ∈ N
 6     then if A → ε ∈ P
 7             then F ← {ε}
 8             else  F ← ∅
 9          for minden A → Y1Y2...Ym ∈ P-re  (m ≥ 1)
10             do F ← F ∪ (Első(Y1) \ {ε})
11                for k ← 1 to m − 1
12                   do if Y1Y2...Yk ⟹* ε
13                         then F ← F ∪ (Első(Yk+1) \ {ε})
14                if Y1Y2...Ym ⟹* ε
15                   then F ← F ∪ {ε}
16  if α = Y1Y2...Ym (m ≥ 2)
17     then F ← (Első(Y1) \ {ε})
18          for k ← 1 to m − 1
19             do if Y1Y2...Yk ⟹* ε
20                   then F ← F ∪ (Első(Yk+1) \ {ε})
21          if Y1Y2...Ym ⟹* ε
22             then F ← F ∪ {ε}
23  return F
```

Az 1–4. sorokban az ε és egy a terminális szimbólum argumentumra adjuk meg a halmazt, az 5–15. sorokban az A nemterminális szimbólumra határozzuk meg halmaz elemeit. A 6–7. sorokban és a 14–15. sorokban a halmazba betesszük az ε szimbólumot, ha az A-ból az ε levezethető. Az algoritmus 16–22. sorai arra az esetre adják meg a halmaz elemeit, ha az argumentum egy szimbólumsorozat. Megjegyezzük, hogy a 11. és 18. sor **for** ciklusát már akkor is befejezhetjük, ha $Y_k \in T$, mivel ekkor $Y_1Y_2 \dots Y_k$-ból biztosan nem vezethető le az ε.

A 6.3.10. tételben, és a továbbiakban is, szükséges a *Követő*(A) halmaz elemeinek ismerete. Ezeknek a meghatározására a következő algoritmust adjuk.

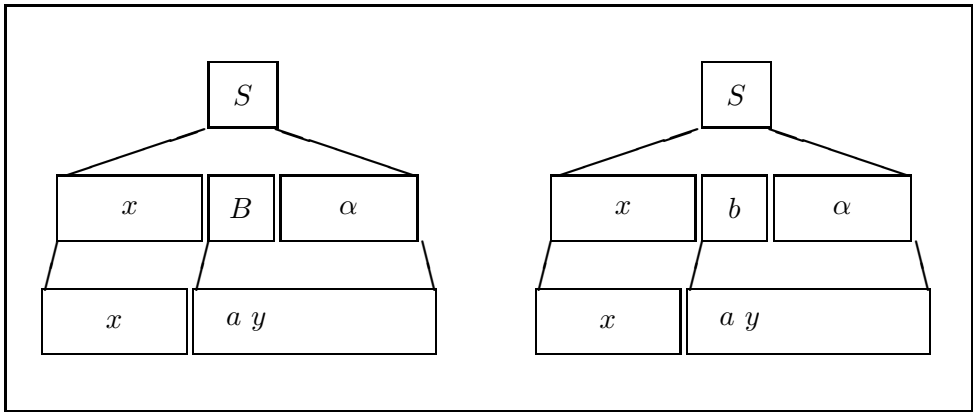
```
Követő(A)
1  if A = S
2     then F ← {#}
3     else  F ← ∅
```

```
4  for minden B → αAβ ∈ P szabályra
5      do if |β| > 0
6            then F ← F ∪ (Első(β) \ {ε})
7                 if β ⟹* ε
8                    then F ← F ∪ Követő(B)
9            else  F ← F ∪ Követő(B)
10 return F
```

A *Követő*(A) halmaz elemeit az F halmazba helyezzük. A 4–9. sorokban megvizsgáljuk, hogy ha az argumentum egy szabály jobb oldalán szerepel, milyen terminális szimbólumok állhatnak közvetlenül utána. Látható, hogy az ε nem kerülhet bele a halmazba, és a # is csak akkor, ha az argumentum egy mondatforma legjobboldalibb szimbóluma.

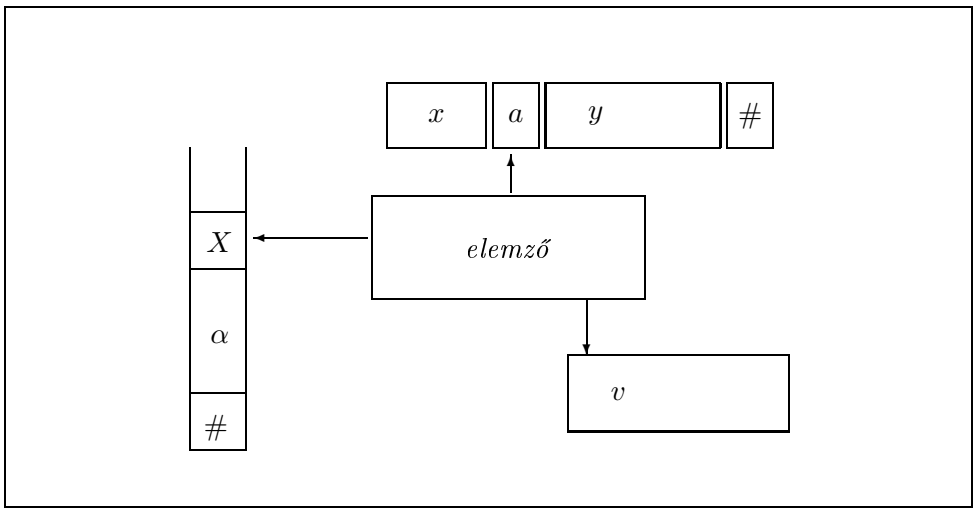
6.3.2. Táblázatos elemzés

Tegyük fel, hogy az elemezendő terminális sorozat xay, amiből az x szöveget már szintaktikai hiba detektálása nélkül elemeztük. Mivel felülről lefelé elemzünk, tehát legbaloldalibb helyettesítéseket alkalmazunk, az elemezendő mondatformánk $xY\alpha$, azaz $xB\alpha$ vagy $xb\alpha$ alakú ($Y \in (N \cup T)$, $B \in N$, $a, b \in T$, $x, y \in T^*$, $\alpha \in (N \cup T)^*$) (6.2. ábra).



6.2. ábra. A mondatforma és az elemezendő szöveg.

Az első esetben a szintaxisfa építésében most a B egy helyettesítése a következő lépés. Ismerjük a bemenő szimbólumsorozat következő a elemét, ezért egyértelműen meghatározhatjuk azt a $B \to \beta$ szabályt, amelyikre $a \in$ *Első*(β*Követő*(B)). Ha van ilyen szabály, akkor az $LL(1)$ nyelvtan értelmezése alapján pontosan egy van, ha nincs, akkor ez az eset egy *szintaktikai hiba* megtalálását jelenti.



6.3. ábra. Az $LL(1)$ elemző szerkezete.

A második esetben a mondatforma következő szimbóluma a b terminális szimbólum, tehát azt várjuk, hogy az elemezendő szöveg következő szimbóluma is b legyen. Ha ez teljesül, azaz $a = b$, akkor az a szimbólum egy helyes szimbólum, továbbléphetünk, mind a mondatformában, mind az elemezendő szövegben az a szimbólum átkerülhet a már elemzett jelsorozatba. Ha $a \neq b$, akkor ez egy *szintaktikai hibát* jelent. Láthatjuk, hogy mindkét szintaktikai hiba esetén ismerjük a hiba helyét is, az a a hibás szimbólum.

Az elemző működését a következőképpen írjuk le. Jelöljük a # jellel az elemezendő szöveg jobb oldali végpontját, azaz legyen a # az elemezendő szöveg utolsó szimbóluma. Az elemzéshez egy vermet is használunk, a verem alját jelöljük szintén egy # jellel. A helyettesítési szabályokat valamilyen sorrendben, például a felsorolásuk sorrendjében sorszámozzuk meg. Az elemzés során alkalmazott szabályok sorszámát felírjuk egy listába, az elemzés végén ez a lista arra fog szolgálni, hogy az elemzett szöveg szintaxisfáját felépíthessük (6.3. ábra).

Az *elemzés állapotait* az $(ay\#, X\alpha\#, v)$ hármassal jelöljük. Az $ay\#$ a még nem elemzett szöveg, $X\alpha\#$ az elemzés mondatformájának még nem elemzett része, ez van a veremben, úgyhogy X van a verem tetején, v pedig a szabályok sorszámait tartalmazó lista. Az elemzés úgy fog működni, hogy mindig a verem tetején levő X szimbólumot és a még nem elemzett szöveg első szimbólumát, az a-t fogjuk vizsgálni. Az a-t *aktuális szimbólumnak* nevezzük. A verem tetejére és az aktuális szimbólumra az elemzőből egy-egy pointer mutat.

Mivel felülről lefelé elemzünk, a verem kezdeti tartalma legyen $S\#$. Ha a teljes elemezendő szöveget xay-nal jelöljük, akkor az elemzés kezdetén az elemzés állapotát, azaz a *kezdőállapotot* az $(xay\#, S\#, \varepsilon)$ hármassal írhatjuk le, ahol most ε az üres listát jelöli.

Az elemzést egy T *elemző táblázat* segítségével fogjuk végezni. A táblázat sorai a verem tetején álló szimbólumot, az oszlopai pedig az input következő elemezendő szimbólumát jelölik, a # jelet a táblázat utolsó sorához és utolsó oszlopához írjuk.

A táblázat $T[X, a]$ eleme legyen a következő:

$$T[X,a] = \begin{cases} (\beta, i), & \text{ha } X \to \beta \text{ az } i\text{-edik helyettesítési szabály ,} \\ & a \in \textit{Első}(\beta) \text{ vagy} \\ & (\varepsilon \in \textit{Első}(\beta) \text{ és } a \in \textit{Követő}(X)) \text{ ,} \\ \textit{pop}, & \text{ha } X = a \text{ ,} \\ \textit{elfogad}, & \text{ha } X = \# \text{ és } a = \# \text{ ,} \\ \textit{hiba} & \text{egyébként .} \end{cases}$$

A táblázat kitöltését a következő algoritmussal végezhetjük:

LL(1)-TÁBLÁZATOT-KITÖLT(G)

```
 1  for minden A ∈ N-re
 2      do if A → α ∈ P az i-edik szabály
 3            then for minden a ∈ ELSŐ(α)-ra
 4                    do T[A, a] ← (α, i)
 5                    if ε ∈ ELSŐ(α)
 6                       then for minden a ∈ KÖVETŐ(A)-ra
 7                               do T[A, a] ← (α, i)
 8  for minden a ∈ T-re
 9      do T[a, a] ← pop
10  T[#, #] ← elfogad
11  for minden X ∈ (N ∪ T ∪ {#}) és minden a ∈ (T ∪ {#})-ra
12      do if T[X, a] = „üres"
13            then T[X, a] ← hiba
14  return T
```

A 10. sorban a táblázat jobb alsó sorába az *elfogad* szöveget írjuk, a 8–9. sorokban a terminálisokkal címkézett sorok és oszlopok rész-táblázatának főátlójába a *pop* szöveget írjuk, az 1–7. sorokban levő algoritmussal a megadott pozícióra a szabály jobb oldalának szimbólumai és a szabály sorszáma kerül.

Az algoritmus 12–13. sorában a táblázat üresen maradt helyeire a szintaktikai hibát jelző *hiba* szöveget írjuk.

Az elemzés működése állapotátmenetekkel adható meg. A kezdőállapot tehát $(x\#, S\#, \varepsilon)$, ha az elemezendő szöveg x, és az elemzés sikeresen befejeződik akkor, ha az elemző a $(\#, \#, w)$ állapotba, a *végállapotba* kerül. Ha a még nem elemzett szöveg az $ay\#$, és a verem tetején az X szimbólum áll, az állapotátmenetek a következők:

$$(ay\#, X\alpha\#, v) \;\rightarrow\; \begin{cases} (ay\#, \beta\alpha\#, vi), & \text{ha } T[X,a] = (\beta, i)\ , \\ (y\#, \alpha\#, v), & \text{ha } T[X,a] = \mathit{pop}\ , \\ \mathit{O.K.}, & \text{ha } T[X,a] = \mathit{elfogad}\ , \\ \mathit{HIBA}, & \text{ha } T[X,a] = \mathit{hiba}\ . \end{cases}$$

Az *O.K.* azt jelenti, hogy az elemzett szimbólumsorozat szintaktikusan helyes, a *HIBA* pedig egy szintaktikai hiba detektálását jelzi.

Az elemző működésére a következő algoritmust adhatjuk meg.

```
LL(1)-ELEMEZ(xay#, T)
 1  s ← (xay#, S#, ε), s' ← elemez
 2  repeat
 3        if s = (ay#, Aα#, v) és T[A, a] = (β, i)
 4          then s ← (ay#, βα#, vi)
 5          else if s = (ay#, aα#, v)
 6                  then s ← (y#, α#, v)          ▷ Ekkor T[a, a] = pop.
 7                  else  if s = (#, #, v)
 8                        then s' ← O.K.          ▷ Ekkor T[#, #] = elfogad.
 9                        else  s' ← HIBA         ▷ Ekkor T[A, a] = hiba.
10  until s' = O.K. vagy s' = HIBA
11  return s', s
```

Az algoritmus bemenő paramétere az xay elemezendő szöveg és a T elemző táblázat. Az s' változó az elemző működését jelzi, működés közben az s' értéke *elemez,* az elemzés befejezésekor *O.K.* vagy *HIBA*. Az elemző az elemzett szöveg a aktuális szimbóluma és a verem tetején levő szimbólum alapján a T táblázatból meghatározza az elvégzendő műveletet. A 3–4. sorban a szintaxisfát építi az $A \rightarrow \beta$ szabály alapján. Az 5–6. sorban léptetés történik, mivel a verem tetején is az a szimbólum található. Az algoritmus a 8–9. sorban az elemzés befejezését jelzi, ha a verem kiürült és az elemezendő szöveg végére ért, akkor az elemzett szöveg helyes, egyebként az elemző egy szintaktikai hibát fedezett fel. Az algoritmus végeredménye ennek megfelelően az *O.K.* vagy *HIBA* jelzés, és kimenetként mindkét esetben megjelenik az elemző állapotának s hármasa is. Helyes szöveg esetén a hármas harmadik eleméből, v-ből a szabályok sorszámai alapján felépíthető a szintaxisfa, szintaktikai hiba esetén a hármas első elemének első szimbóluma a hiba helyét adja meg.

6.3.11. példa. Legyen a G nyelvtan a következő: $G = (\{E, E', T, T', F\}, \{+, *, (,), i\}, P, E)$, ahol a P helyettesítési szabályok a következők:

$E \to TE'$

$E' \to +TE' \mid \varepsilon$

$T \to FT'$

$T' \to *FT' \mid \varepsilon$

$F \to (E) \mid i$

A szabályokból a *Követő*(A) halmazok meghatározhatók, az elemző táblázat kitöltéséhez a következő halmazok szükségesek:

Első$(TE') = \{(, i\}$,

Első$(+TE') = \{+\}$,

Első$(FT') = \{(, i\}$,

Első$(*FT') = \{*\}$,

Első$((E)) = \{(\}$,

Első$(i) = \{i\}$,

Követő$(E') = \{), \#\}$,

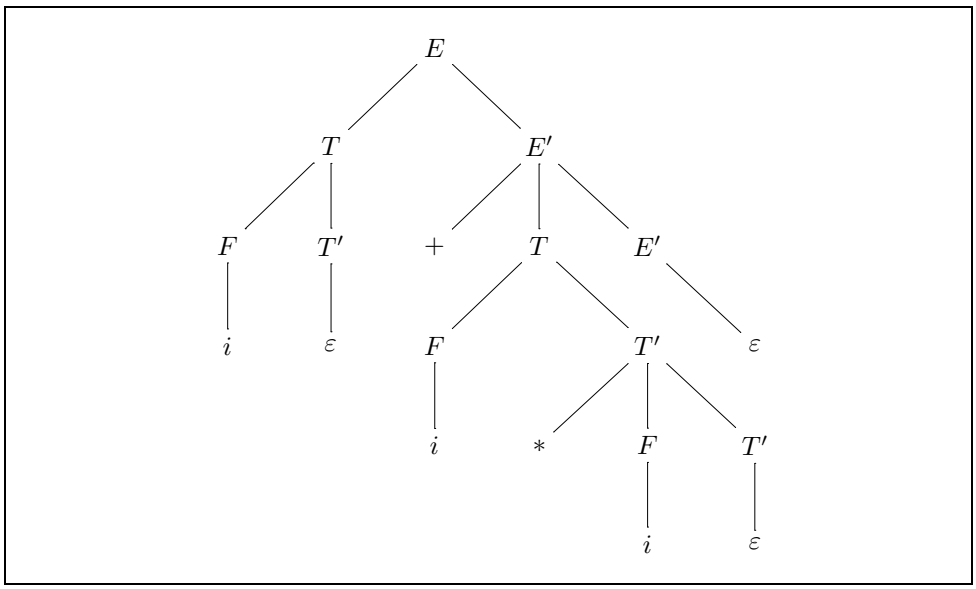
Követő$(T') = \{+,), \#\}$.

Az elemző táblázat a következő, a táblázatban az üres helyek a *hibát* jelentik.

	$+$	$*$	$($	$)$	i	$\#$
E			$(TE', 1)$		$(TE', 1)$	
E'	$(+TE', 2)$			$(\varepsilon, 3)$		$(\varepsilon, 3)$
T			$(FT', 4)$		$(FT', 4)$	
T'	$(\varepsilon, 6)$	$(*FT', 5)$		$(\varepsilon, 6)$		$(\varepsilon, 6)$
F			$((E), 7)$		$(i, 8)$	
$+$	*pop*					
$*$		*pop*				
$($			*pop*			
$)$				*pop*		
i					*pop*	
$\#$						*elfogad*

□

6.3.12. példa. Az előző példában szereplő nyelvtan elemző táblázatának felhasználásával elemezzük az $i + i * i$ szöveget. Az elemzés a következő:



6.4. ábra. Az $i + i * i$ mondat szintaxisfája.

$$
\begin{array}{lll}
(i+i*i\#,\, S\#,\, \varepsilon) & \xrightarrow{(TE',1)} & (\quad i+i*i\#,\quad TE'\#,\quad 1\quad) \\
& \xrightarrow{(FT',4)} & (\quad i+i*i\#,\quad FT'E'\#,\quad 14\quad) \\
& \xrightarrow{(i,8)} & (\quad i+i*i\#,\quad iT'E'\#,\quad 148\quad) \\
& \xrightarrow{pop} & (\quad +i*i\#,\quad T'E'\#,\quad 148\quad) \\
& \xrightarrow{(\varepsilon,6)} & (\quad +i*i\#,\quad E'\#,\quad 1486\quad) \\
& \xrightarrow{(+TE',2)} & (\quad +i*i\#,\quad +TE'\#,\quad 14862\quad) \\
& \xrightarrow{pop} & (\quad i*i\#,\quad TE'\#,\quad 14862\quad) \\
& \xrightarrow{(FT',4)} & (\quad i*i\#,\quad FT'E'\#,\quad 148624\quad) \\
& \xrightarrow{(i,8)} & (\quad i*i\#,\quad iT'E'\#,\quad 1486248\quad) \\
& \xrightarrow{pop} & (\quad *i\#,\quad T'E'\#,\quad 1486248\quad) \\
& \xrightarrow{(*FT',5)} & (\quad *i\#,\quad *FT'E'\#,\quad 14862485\quad) \\
& \xrightarrow{pop} & (\quad i\#,\quad FT'E'\#,\quad 14862485\quad) \\
& \xrightarrow{(i,8)} & (\quad i\#,\quad iT'E'\#,\quad 148624858\quad) \\
& \xrightarrow{pop} & (\quad \#,\quad T'E'\#,\quad 148624858\quad) \\
& \xrightarrow{(\varepsilon,6)} & (\quad \#,\quad E'\#,\quad 1486248586\quad) \\
& \xrightarrow{(\varepsilon,3)} & (\quad \#,\quad \#,\quad 14862485863\quad) \\
& \xrightarrow{elfogad} & O.K.
\end{array}
$$

Az elemzett mondat szintaxisfája az 6.4. ábrán látható. □

6.3.3. A rekurzív leszállás módszere

A visszalépés nélküli felülről-lefelé elemzésekre a táblázatos módszeren kívül gyakran alkalmazunk egy olyan módszert, amelynek lényege az, hogy a nyelvtanhoz egy programot rendelünk.

A nyelvtan szimbólumaihoz eljárásokat adunk meg, és elemzés közben a rekurzív eljáráshívásokon keresztül a programnyelv implementációja valósítja meg az elemző vermét és a veremkezelést. A felülről-lefelé elemzés és a rekurzív eljáráshívások miatt ezt a módszert a ***rekurzív leszállás módszerének*** nevezzük.

A terminális szimbólumok vizsgálatára vezessük be az *Vizsgál* eljárást. Legyen az eljárás paramétere a „várt szimbólum", azaz a mondatforma legbaloldalibb, még nem vizsgált terminális szimbóluma, és tartalmazza az *aktuális_szimbólum* globális változó a vizsgált terminális sorozat soron következő szimbólumát.

```
procedure Vizsgál(a);
begin
   if aktuális_szimbólum = a
      then Következő_szimbólum
      else Hibajelzés
end;
```

A *Következő_szimbólum* az előreolvasásra szolgál, ez a lexikális elemzőt meghívó eljárás neve. Ez az eljárás a bemenő szimbólumsorozatból meghatározza a következő szimbólumot és ezt az *aktuális_szimbólum* változóba tölti. A *Hibajelzés* eljárás szintaktikai hibajelzést ad, és ezután befejezi az elemző program futását.

A nyelvtan minden nemterminális szimbólumához rendeljünk hozzá egy eljárást. Az A szimbólumhoz tartozó eljárás legyen a következő:

```
procedure  A;
begin
   T(A)
end;
```

ahol `T(A)`-t az A-ra vonatkozó helyettesítési szabályok jobb oldalán álló szimbólumok határozzák meg.

Az elemzéshez használt nyelvtanok *redukáltak,* ami többek között azt jelenti, hogy nincs bennük „felesleges" nemterminális szimbólum, minden nemterminális szimbólum szerepel legalább egy helyettesítési szabály bal oldalán.

Tehát ha az A szimbólumot vizsgáljuk, biztosan van legalább egy $A \to \alpha$ helyettesítési szabály.

1. Ha az A szimbólumra csak egy helyettesítési szabály van:

   (a) az $A \to a$ szabályhoz rendelt program legyen a `Vizsgál(a)`,
   (b) az $A \to B$ szabályhoz rendeljük hozzá a `B` eljáráshívást,
   (c) az $A \to X_1 X_2 \ldots X_n$ $(n \geq 2)$ szabályhoz tartozzon a következő blokk:

```
begin
   T(X_1);
   T(X_2);
   ...
   T(X_n)
end;
```

2. Ha az A szimbólumra több helyettesítési szabály van:

   (a) Ha az $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$ szabályok ε-mentesek, azaz α_i-ből $(1 \leq i \leq n)$ nem vezethető le az ε, akkor `T(A)` legyen

```
case aktuális_szimbólum of
   Első(alpha_1) : T(alpha_1);
   Első(alpha_2) : T(alpha_2);
   ...
   Első(alpha_n) : T(alpha_n)
end;
```

   ahol `Első(alpha_i)` az *Első*(α_i) programbeli jelölése. Felhívjuk a figyelmet arra, hogy a rekurzív leszállás módszerében most először használjuk ki azt, hogy a nyelvtan $LL(1)$-es.

   (b) Az $LL(1)$ nyelvtan programnyelvet ír le, ezért a nyelvtan ε-mentességét nem célszerű megkövetelni. Az $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_{n-1} \mid \varepsilon$ szabályokhoz a következő `T(A)` programot rendeljük:

```
case aktuális_szimbólum of
   Első(alpha_1)     : T(alpha_1);
   Első(alpha_2)     : T(alpha_2);
   ...
   Első(alpha_(n-1)) : T(alpha_(n-1));
   Követő(A)         : skip
end;
```

   ahol `Követő(A)` a *Követő*(A)-nak felel meg.

   Speciálisan, ha az $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$ szabály esetén egy i-re $(1 \leq i \leq n)$ $\alpha_i \stackrel{*}{\Longrightarrow} \varepsilon$, azaz $\varepsilon \in$ *Első*(α_i), akkor a `case` utasítás i-edik sora lesz a `Követő(A) : skip` sor.

A `T(A)`-ban a `case` helyett, ha lehetséges, használhatunk `if-then-else` vagy `while` utasítást is.

A rekurzív leszállás módszerével készített elemző program kezdő eljárása, azaz főprogramja a nyelvtan kezdőszimbólumához írt eljárás lesz.

A rekurzív leszállás módszerével működő elemző programot a következő REK-LESZÁLL-KÉSZÍT algoritmussal hozhatjuk létre. Az algoritmus bemenete a G nyelvtan, és az algoritmus eredményül az elemző P programját adja. Az algoritmusban használunk egy PROGRAMOT-ÍR eljárást, ami az argumentumában megadott programsorokat a már meglévő P programhoz fűzi. Ezt az algoritmust nem részletezzük.

```
REK-LESZÁLL-KÉSZÍT(G)
 1  P ← ∅
 2  PROGRAMOT-ÍR(
 3       procedure Vizsgál(a);
 4       begin
 5       if aktuális_szimbólum = a
 6          then Következő_szimbólum
 7          else Hibajelzés
 8       end;
 9       )
10  for a G nyelvtan minden A ∈ N szimbólumára
11      do if A = S
12           then PROGRAMOT-ÍR(
13                    program S;
14                    begin
15                      REK-LESZÁLL-UT(S, P)
16                    end.
17                    )
18           else PROGRAMOT-ÍR(
19                   procedure A;
20                   begin
21                     REK-LESZÁLL-UT(A, P)
22                   end;
23                   )
24  return P
```

Az algoritmus a 2–9. sorokban elkészíti a *Vizsgál* eljárást, majd a bemeneteként megadott G nyelvtan minden nemterminális szimbólumára a REK-LESZÁLL-UT algoritmus felhasználásával készíti a szimbólumhoz tartozó eljárást. A 11–17. sorokban látható, hogy a nyelvtan kezdőszimbólumához az

elemző főprogramja fog tartozni. Az algoritmus kimenete az elemző program lesz.

```
REK-LESZÁLL-UT(A, P)
1  if csak egy A → α szabály van
2     then REK-LESZÁLL-UT1(α, P)                    ▷ A → α.
3     else REK-LESZÁLL-UT2(A, (α_1, ..., α_n), P)   ▷ A → α_1 | ··· | α_n.
4  return P
```

Mivel a létrehozandó elemző program utasításai lényegesen függnek attól, hogy az A nemterminális szimbólumra a nyelvtanban hány helyettesítési szabály van, a REK-LESZÁLL-UT algoritmus a további műveleteket két részre osztja. A REK-LESZÁLL-UT1 algoritmus foglalkozik azzal az esettel, amikor csak egy helyettesítési szabály van, és a REK-LESZÁLL-UT2 készíti az alternatívákra vonatkozó elemző programot.

```
REK-LESZÁLL-UT1(α, P)
 1  if α = a
 2     then PROGRAMOT-ÍR(
 3               Vizsgál(a)
 4               )
 5  if α = B
 6     then PROGRAMOT-ÍR(
 7               B
 8               )
 9  if α = X_1 X_2 ... X_n (n ≥ 2)
10     then PROGRAMOT-ÍR(
11               begin
12                 REK-LESZÁLL-UT1(X_1, P) ;
13                 REK-LESZÁLL-UT1(X_2, P) ;
14                 ...
15                 REK-LESZÁLL-UT1(X_n, P)
16               end;
17  return P
```

```
REK-LESZÁLL-UT2(A, (α_1, ..., α_n), P)
 1  if α_1, ..., α_n szabályok ε-mentesek
 2    then PROGRAMOT-ÍR(
 3                case aktuális_szimbólum of
 4                  Első(alpha_1) : REK-LESZÁLL-UT1(α_1, P) ;
 5                  ...
 6                  Első(alpha_n) : REK-LESZÁLL-UT1(α_n, P)
 7                end;
 8                )
 9  if van ε-szabály, α_i = ε (1 ≤ i ≤ n)
10    then PROGRAMOT-ÍR(
11                case aktuális_szimbólum of
12                  Első(alpha_1)     : REK-LESZÁLL-UT1(α_1, P) ;
13                  ...
14                  Első(alpha_(i-1)) : REK-LESZÁLL-UT1(α_{i-1}, P) ;
15                  Követő(A)         : skip;
16                  Első(alpha_(i+1)) : REK-LESZÁLL-UT1(α_{i+1}, P) ;
17                  ...
18                  Első(alpha_n)     : REK-LESZÁLL-UT1(α_1, P)
19                end;
20                )
21  return P
```

A fenti két algoritmus a korábban már részletesen leírt programot hozza létre.

Az elemezendő szöveg végének az ellenőrzése a rekurzív leszállás módszerével úgy valósítható meg, hogy a szöveg végét jelző # szimbólumot beépítjük egy új helyettesítési szabályba. Ha a nyelvtan kezdőszimbóluma S, akkor létrehozunk egy $S' \to S\#$ szabályt, és az új S' lesz az új nyelvtan kezdőszimbóluma. A # szimbólumot terminális szimbólumnak tekintjük. Az így kibővített nyelvtanra készítjük el a rekurzív leszállás elemző programját.

6.3.13. példa. A 6.3.11. példában szereplő nyelvtant egészítsük ki a fenti módon. A szabályok tehát a következők.

$S' \to E\#$
$E \to TE'$
$E' \to +TE' \mid \varepsilon$
$T \to FT'$
$T' \to *FT' \mid \varepsilon$
$F \to (E) \mid i$

A 6.3.11. példában megadtuk a táblázatos elemző létrehozásához szükséges *Első* és *Követő* halmazokat. Ezek közül most a következőkre van szükség:

Első$(+TE') = \{+\}$,

$Első(*FT') = \{*\}$,
$Első((E)) = \{(\}$,
$Első(i) = \{i\}$,
$Követő(E') = \{), \#\}$,
$Követő(T') = \{+,), \#\}$.

A halmazok felhasználásának helyét a program sorainak kommentjeiben adjuk meg, a kommenteket a -- karakterpárral kezdjük.

A rekurzív leszállás módszerének elemző programja a következő lesz.

```
program S';
begin
   E;
   Vizsgál(#)
end.
procedure E;
begin
   T;
   E'
end;
procedure E';
begin
   case aktuális_szimbólum of
   +     : begin                 -- Első(+TE')
               Vizsgál(+);
               T;
               E'
           end;
   ),#   : skip                  -- Követő(E')
   end
end;
procedure T;
begin
   F;
   T'
end;
procedure T';
begin
   case aktuális_szimbólum of
   *     : begin                 -- Első(*FT')
               Vizsgál(*);
               F;
               T'
           end;
   +,),# : skip                  -- Követő(T')
   end
end;
```

```
procedure F;
begin
   case aktuális_szimbólum of
   (     : begin                -- Első((E))
               Vizsgál(();
               E;
               Vizsgál())
           end;
   i     : Vizsgál(i)            -- Első(i)
   end
end;
```

Látható, hogy az elemző főprogramja a nyelvtan S' kezdőszimbólumához tartozó eljárás lett. □

6.4. $LR(1)$ elemzés

Alulról-felfelé elemezve az elemezendő szimbólumsorozatból indulunk ki, megkeressük a mondatforma nyelét (a nyél fogalmát már a 6.1.3. értelmezésben megadtuk), és ezt a nyelet helyettesítjük a hozzátartozó nemterminális szimbólummal. Ezt ismételve próbáljuk felépíteni a szintaxisfát. A célunk az, hogy elérjük a nyelvtan kezdőszimbólumát, ez lesz majd a szintaxisfa gyökérpontja, a fa levelein pedig az elemezendő programszöveg terminális szimbólumai lesznek.

Először áttekintjük azokat a fogalmakat, amelyeket az alulról-felfelé haladó elemzésekben használunk.

Alulról-felfelé elemezve mindig a mondatforma nyelét kell meghatároznunk. A probléma tehát az, hogy hogyan lehet a nyelet meghatározni, és ha a nyél helyettesítésére több lehetőség is van, akkor a nyelet melyik nemterminális szimbólummal kell helyettesíteni.

6.4.1. értelmezés. *Ha $A \to \alpha \in P$, akkor a βAx mondatforma ($x \in T^*$, $\alpha, \beta \in (N \cup T)^*$) **legjobboldalibb helyettesítése** $\beta\alpha x$, azaz*

$$\beta Ax \underset{legjobb}{\Longrightarrow} \beta\alpha x \ .$$

6.4.2. értelmezés. *Ha az $S \overset{*}{\Longrightarrow} x$ ($x \in T^*$) levezetésben minden helyettesítés legjobboldalibb helyettesítés, akkor ezt a levezetést **legjobboldalibb levezetésnek** nevezzük, és így jelöljük:*

$$S \underset{legjobb}{\overset{*}{\Longrightarrow}} x \ .$$

A legjobboldalibb levezetésben a terminális szimbólumok a mondatforma jobb oldalán jelennek meg. A nyél és a legjobboldalibb helyettesítés kapcsolata alapján, ha a legjobboldalibb levezetés lépéseit „visszafelé" alkalmazzuk, akkor éppen az alulról-felfelé haladó elemzés lépéseit kapjuk meg. Az alulról-felfelé elemzés tehát a legjobboldalibb levezetés „inverzének" felel meg. Ezért a továbbiakban, amikor alulról-felfelé elemzésről lesz szó, a helyettesítéseket és a levezetéseket jelölő nyilak alá már nem is írjuk oda a „*legjobb*" szót.

Az általános alulról-felfelé elemzést, a felülről-lefelé elemzésekhez hasonlóan, visszalépéses algoritmussal lehet megvalósítani. A visszalépések azonban rendkívül lelassíthatják az elemzést, ezért csak olyan nyelvtanokkal fogunk foglalkozni, amelyekre visszalépés nélküli elemzések adhatók meg.

A további szakaszokban bemutatunk egy hatékony, a környezetfüggetlen nyelvtanok igen nagy osztályára alkalmazható elemzési módszert. Ez a nyelvtan-osztály a gyakorlatban használt programnyelvek nyelvtanait is tartalmazza.

Az elemzést $LR(k)$ elemzésnek, a nyelvtant $LR(k)$ nyelvtannak nevezzük, ahol az LR a balról jobbra („*L*eft to *R*ight") történő elemzésre utal, a k pedig azt jelenti, hogy k szimbólumot előreolvasva egyértelműen meghatározható a mondatforma nyele. Az $LR(k)$ elemzés visszalépés nélküli, léptetés-redukálás típusú elemzés.

Mint majd látni fogjuk, elegendő az $LR(1)$-es elemzőkkel foglalkoznunk, mivel minden $LR(k)$ $(k > 1)$ nyelvtanhoz létezik vele ekvivalens $LR(1)$ nyelvtan. Ez rendkívül fontos számunkra, mivel így egy szöveg elemzésekor mindig elég csak egy szimbólumot előreolvasni.

Az $LR(k)$ elemzés hátrányának hozható fel, hogy az elemző táblázatának „kézi" megkonstruálása nem könnyű. Léteznek azonban olyan programok (például a UNIX `yacc` programja), amelyek egy adott nyelvtan szabályaiból létrehozzák a teljes elemző programot, és így az elemző megírása sem jelent problémát.

Az $LR(k)$ nyelvtanok vizsgálata után az $LALR(1)$ elemzést, a programnyelvek fordítóprogramjaiban jelenleg használt elemzési módszert tanulmányozzuk.

6.4.1. Az $LR(k)$ nyelvtanok

Mint már korábban is tettük, jelöljük az elemezendő szöveg, az elemezendő terminális sorozat jobb oldalát a # szimbólummal. Vezessünk be egy új S' nemterminális szimbólumot és egy új $S' \to S$ szabályt.

6.4.3. értelmezés. *Legyen a $G = (N, T, P, S)$ nyelvtanhoz tartozó G'* ***kiegészített nyelvtan*** *a következő:*

$$G' = (N \cup \{S'\}, T, P \cup \{S' \to S\}, S') \ .$$

Sorszámozzuk meg a helyettesítési szabályokat, az $S' \to S$ szabály legyen a nulladik szabály. Így, ha redukáláskor a nulladik szabályt kell alkalmazni, akkor ez az elemzés végét, és az elemzett szöveg szintaktikus helyességét fogja jelenteni.

Megjegyezzük, hogy ha az eredeti S kezdőszimbólum nem szerepel egyik helyettesítési szabály jobb oldalán sem, akkor az $S' \to S$ kiegészítésre nincs is szükség. Az általánosság kedvéért azonban az $LR(k)$ tulajdonságot csak kiegészített nyelvtanokra értelmezzük.

6.4.4. értelmezés. *Egy G' kiegészített nyelvtan* ***LR(k) nyelvtan*** *($k \geq 0$), ha bármely két*

$$S' \overset{*}{\Longrightarrow} \alpha A w \Longrightarrow \alpha\beta w \ ,$$
$$S' \overset{*}{\Longrightarrow} \gamma B x \Longrightarrow \gamma\delta x = \alpha\beta y$$

$(A, B \in N,\ x, y, w \in T^*,\ \alpha, \beta, \gamma, \delta \in (N \cup T)^*)$ levezetésre

$$\mathit{Első}_k(w) = \mathit{Első}_k(y)$$

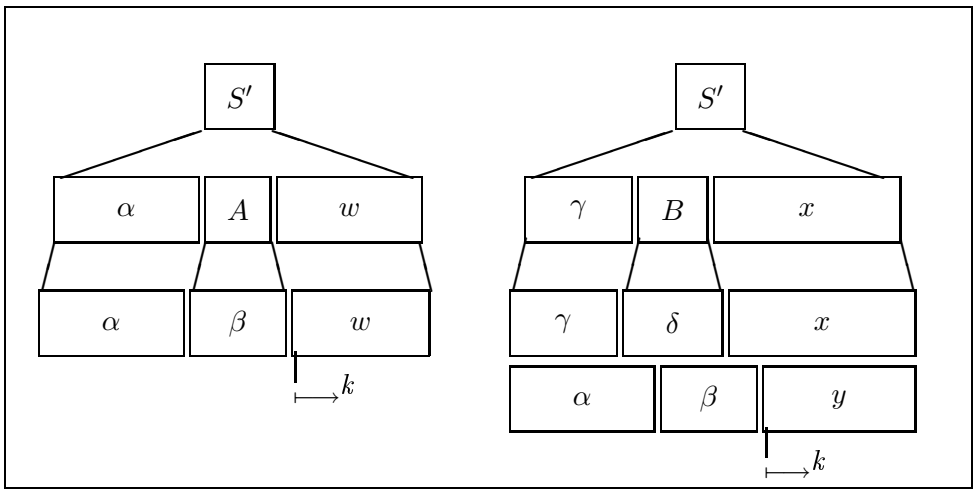
esetén

$$\alpha = \gamma, A = B \text{ és } x = y \ .$$

Az $LR(k)$ nyelvtanokra az a jellemző, hogy az $\alpha\beta w$ mondatformában a w első szimbólumától kezdve előreolvasva k darab szimbólumot, egyértelműen meghatározható, hogy valóban β a nyél, és az, hogy az $A \to \beta$ szabállyal kell redukálni, azaz az $\alpha\beta w$ mondatforma az $\alpha A w$ mondatformára redukálható.

Tegyük fel ugyanis, hogy az $\alpha\beta w$ és az $\alpha\beta y$ mondatformákban, amelyeknek tehát az $\alpha\beta$ prefixük azonos, $\mathit{Első}_k(w) = \mathit{Első}_k(y)$, és mégis az $\alpha\beta w$ az $\alpha A w$-re, az $\alpha\beta y$ pedig a $\gamma B x$-re redukálható. Az $LR(k)$ tulajdonság miatt ekkor csak $\alpha = \gamma$ és $A = B$ lehet. Ez azt jelenti, hogy a nyél vagy soha nem a β, vagy pedig mindig az (6.5. ábra).

6.4.5. példa. A $G' = (\{S', S\}, \{a\}, P', S')$ nyelvtan, ahol a helyettesítési szabályok
$S' \to S$
$S \to Sa \mid a$
nem $LR(0)$ nyelvtan, mivel, feltüntetve az értelmezés jelöléseit,



6.5. ábra. Az $LR(k)$ nyelvtan.

$$S' \overset{*}{\Longrightarrow} \begin{matrix}\varepsilon & S' & \varepsilon\\ \alpha & A & w\end{matrix} \Longrightarrow \begin{matrix}\varepsilon & S & \varepsilon\\ \alpha & \beta & w\end{matrix},$$

$$S' \overset{*}{\Longrightarrow} \begin{matrix}\varepsilon & S' & \varepsilon\\ \gamma & B & x\end{matrix} \Longrightarrow \begin{matrix}\varepsilon & Sa & \varepsilon\\ \gamma & \delta & x\end{matrix} = \begin{matrix}\varepsilon & S & a,\\ \alpha & \beta & y\end{matrix}$$

esetén $Első_0(\varepsilon) = Első_0(a) = \varepsilon$, de $\gamma Bx \neq \alpha Ay$. □

6.4.6. példa. A következő nyelvtan egy $LR(1)$ nyelvtan. $G = (\{S', S\}, \{a, b\}, P', S')$, ahol a helyettesítési szabályok:

$S' \to S$

$S \to SaSb \mid \varepsilon$ □

A következő példában megmutatjuk, hogy van olyan környezetfüggetlen nyelvtan, amelyik nem $LR(k)$ nyelvtan egyetlen k-ra sem ($k \geq 0$).

6.4.7. példa. Legyenek a $G' = (\{S', S\}, \{a\}, P', S')$ nyelvtan helyettesítési szabályai a következők:

$S' \to S$

$S \to aSa \mid a$

Ekkor minden k-ra ($k \geq 0$)

$$S' \overset{*}{\Longrightarrow} a^k S a^k \Longrightarrow a^k a a^k = a^{2k+1},$$

$$S' \overset{*}{\Longrightarrow} a^{k+1} S a^{k+1} \Longrightarrow a^{k+1} a a^{k+1} = a^{2k+3},$$

és

$$Első_k(a^k) = Első_k(a a^{k+1}) = a^k,$$

de

$$a^{k+1} S a^{k+1} \neq a^k S a^{k+2}.$$ □

Míg egy tetszőleges $LL(k)$ $(k > 1)$ nyelvtanra nem biztos, hogy lehet vele ekvivalens $LL(1)$ nyelvtant megadni, addig az $LR(k)$ nyelvtanokra jobb eredményt lehet elérni:

6.4.8. tétel. *Minden $LR(k)$ $(k > 1)$ nyelvtanhoz létezik vele ekvivalens $LR(1)$ nyelvtan.*

A fenti tétel rendkívül nagy jelentősége az, hogy $LR(k)$ $(k > 1)$ nyelvtanok és nyelvek helyett elegendő csak az $LR(1)$ nyelvtanokkal és nyelvekkel foglalkozni.

6.4.2. $LR(1)$ kanonikus halmazok

Értelmezzük az LR elemzések egyik központi fogalmát.

6.4.9. értelmezés. *Legyen az $\alpha\beta x$ $(\alpha, \beta \in (N \cup T)^*, x \in T^*)$ mondatforma nyele β. Ekkor az $\alpha\beta$ jelsorozat prefixeit az $\alpha\beta x$* ***járható prefixeinek*** *nevezzük.*

6.4.10. példa. Tekintsük a következő nyelvtant: $G' = (\{E, T, S'\}, \{i, +, (,)\}, P', S')$, ahol a helyettesítési szabályok a következők (a helyettesítési szabályokat sorszámmal láttuk el):

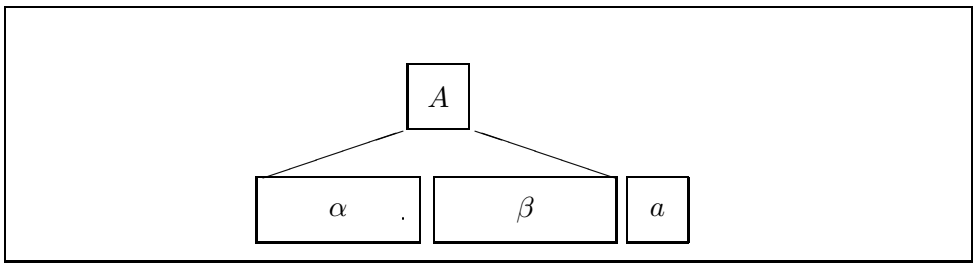
(0) $S' \rightarrow E$
(1) $E \rightarrow T$
(2) $E \rightarrow E + T$
(3) $T \rightarrow i$
(4) $T \rightarrow (E)$

A nyelvtan egy mondatformája $E + (i + i)$, ahol az első i a mondatforma nyele. Ennek a mondatformának a járható prefixei a következők: E, $E+$, $E + ($, $E + (i$. □

Az értelmezés szerint a járható prefixek a mondatforma nyele utáni szimbólumokat nem tartalmazhatják. Így, mivel az alulról-felfelé elemzésben a feladat a mondatforma nyelének a meghatározása, ez a feladat visszavezethető a mondatforma leghosszabb járható prefixének meghatározására.

Ha adott egy nyelvtan, akkor a nyelvtan helyettesítési szabályaiból a járható prefixek halmaza meghatározható. Ugyanakkor nyilvánvaló, hogy az egy nyelvtanhoz tartozó járható prefixek darabszáma nem feltétlenül véges.

A járható prefixek jelentősége az elemzésben a következő: a nyelvtan járható prefixeiből képezett halmazokhoz hozzárendelhetők egy determinisztikus véges automata állapotai, az állapotátmenetekhez pedig a nyelvtan szimbólumai úgy, hogy kiindulva az automata kezdőállapotából, egy állapothoz mindig egy járható prefix szimbólumain keresztül jutunk el. Ezt a tulajdonságot ismerve fogunk egy olyan módszert adni, amellyel az elemzést végző automata meghatározható.



6.6. ábra. Az $[A \to \alpha.\beta, a]$ $LR(1)$-elem.

6.4.11. értelmezés. *Ha a G' nyelvtan egy helyettesítési szabálya $A \to \alpha\beta$, akkor a nyelvtan* ***LR(1)-eleme***

$$[A \to \alpha.\beta, a]\,, \quad (a \in T \cup \{\#\})\,,$$

ahol az $A \to \alpha.\beta$ az $LR(1)$-elem ***magja,*** *és a az $LR(1)$-elem* ***előreolvasási szimbóluma.***

Az előreolvasási szimbólumnak csak akkor van szerepe, ha az $LR(1)$-elem redukciót ír elő, azaz $[A \to \alpha., a]$ alakú. Ez azt jelenti, hogy redukciót majd csak abban az esetben szabad végrehajtani, ha az α-t, azaz a mondat nyelét az a szimbólum követi.

6.4.12. értelmezés. *Egy G' nyelvtan $[A \to \alpha.\beta, a]$ $LR(1)$-eleme* ***érvényes*** *a $\gamma\alpha$ járható prefixre nézve, ha*

$$S' \overset{*}{\Longrightarrow} \gamma Ax \Longrightarrow \gamma\alpha\beta x \ (\gamma \in (N \cup T)^*,\ x \in T^*)\,,$$

és az a az x első szimbóluma, vagy ha $x = \varepsilon$, akkor $a = \#$.

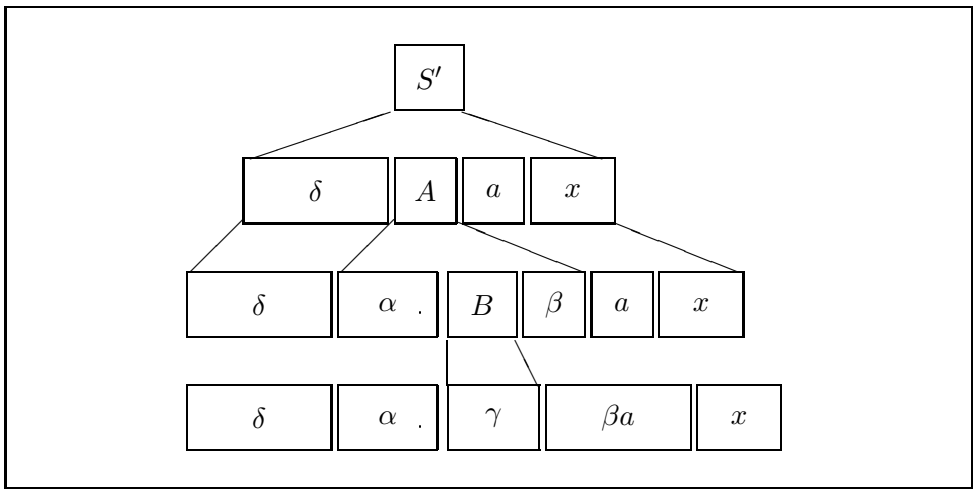
6.4.13. példa. Legyenek a $G' = (\{S', S, A\}, \{a, b\}, P', S')$ nyelvtan helyettesítési szabályai a következők:

(0) $S' \to S$
(1) $S \to AA$
(2) $A \to aA$
(3) $A \to b$

Ekkor $S' \overset{*}{\Longrightarrow} aaAab \Longrightarrow aaaAab$. Az aaa egy járható prefix, és az $[A \to a.A, a]$ érvényes elem erre a járható prefixre nézve. Hasonlóan, $S' \overset{*}{\Longrightarrow} AaA \Longrightarrow AaaA$. Az Aaa járható prefixre nézve az $[A \to a.A, \#]$ $LR(1)$-elem érvényes. □

Az $LR(1)$ elemző felépítéséhez meg kell konstruálni az $LR(1)$-elemek kanonikus halmazait, és ehhez értelmezni kell az $LR(1)$-elemhalmazokra a *closure* „lezárás" és a *read* „olvasás" függvényeket.

6.4.14. értelmezés. *Legyen a $\mathcal{H}$ halmaz egy nyelvtan egy $LR(1)$-elemhalmaza. Ekkor a* ***closure($\mathcal{H}$)*** *halmaz a következő $LR(1)$-elemeket tartalmazza:*



6.7. ábra. A *closure*($[A \to \alpha.B\beta, a]$) függvény.

1. *a* $\mathcal{H}$ *halmaz minden eleme legyen eleme a closure*($\mathcal{H}$) *halmaznak is,*

2. *ha* $[A \to \alpha.B\beta, a] \in$ *closure*($\mathcal{H}$) *és* $B \to \gamma$ *a nyelvtan egy helyettesítési szabálya, akkor legyen* $[B \to .\gamma, b] \in$ *closure*($\mathcal{H}$) *minden* $b \in$ *Első*(βa)*-ra,*

3. *a closure*($\mathcal{H}$) *halmazt a 2. pontban leírt művelettel addig kell bővíteni, ameddig az lehetséges.*

Az értelmezés szerint, ha a $\delta\alpha$ járható prefixre nézve az $[A \to \alpha.B\beta, a]$ egy érvényes $LR(1)$-elem, akkor ugyanerre a prefixre a $[B \to .\gamma, b]$ is egy érvényes $LR(1)$-elem lesz, ahol $b \in$ *Első*(βa). (6.7. ábra). Látható az is, hogy a *closure* művelet a $\delta\alpha$ prefixre az összes érvényes LR(1)-elemet meghatározza.

Egy $\mathcal{H}$ $LR(1)$-elemhalmaz lezárását, azaz a *closure*($\mathcal{H}$) halmazt a következő algoritmussal határozhatjuk meg. A lezárás eredménye a $\mathcal{K}$-ba kerül.

```
ELEMHALMAZ-LEZÁR(H)
1  K ← ∅
2  for minden E ∈ H LR(1)-elemre
3       do K ← K ∪ ELEM-LEZÁR(E)
4  return K
```

```
ELEM-LEZÁR(E)
 1  K_E ← {E}
 2  if E LR(1)-elem [A → α.Bβ, a] alakú
 3     then I ← ∅
 4          J ← K_E
 5          repeat
 6                 for minden [C → γ.Dδ, b] ∈ J alakú LR(1)-elemre
 7                     do for minden D → η ∈ P szabályra
 8                            do for minden c ∈ ELSŐ(δb) szimbólumra
 9                                   do I ← I ∪ [D → .η, c]
10                 J ← I
11                 if I ≠ ∅
12                    then K_E ← K_E ∪ I
13                         I ← ∅
14          until J ≠ ∅
15  return K_E
```

Az ELEM-LEZÁR algoritmus egy E elem $\mathcal{K}_E$ lezárását adja meg. Ha az argumentumban a „pont" után egy terminális szimbólum van, akkor az eredmény halmazában csak ez az egy elem lesz (1. sor). Ha ez a szimbólum egy B nemterminális szimbólum, akkor a B bal oldalú szabályok mindegyikéből tudunk egy új LR(1)-elemet készíteni, ez található a 9. sorban. Mivel az elemek vizsgálatát minden új elemre is el kell végezni, az 5–14. sorok egy **repeat** ciklust tartalmaznak. Ezeket a lépéseket addig kell végezni, amíg új elemeket kapunk (14. sor). A J halmaz tartalmazza a megvizsgálandó elemeket, és I az új elemeket, a $J \leftarrow I$ művelet a 10. sorban látható.

6.4.15. értelmezés. *Legyen a $\mathcal{H}$ halmaz egy nyelvtan egy LR(1)-elemhalmaza. Ekkor a* ***read($\mathcal{H}$, X)*** *($X \in (N \cup T)$) halmaz a következő LR(1)-elemeket tartalmazza:*

1. *ha $[A \to \alpha.X\beta, a] \in \mathcal{H}$, akkor a closure($[A \to \alpha X.\beta, a]$) minden eleme legyen a read($\mathcal{H}, X$) halmaz eleme,*
2. *a read($\mathcal{H}, X$) halmazt az 1. művelettel addig kell bővíteni, ameddig az lehetséges.*

Szemléletesen, a *read*$(\mathcal{H}, X)$ függvény a $\mathcal{H}$ halmaz elemeiben az X szimbólumot olvassa, a „pont" jel az eredmény halmaz elemeiben már az X jobb oldalán van. Ha $\mathcal{H}$ a γ járható prefixekre nézve érvényes $LR(1)$-elemeket tartalmazza, akkor a *read*$(\mathcal{H}, X)$ a γX-re nézve érvényes $LR(1)$-elemek halmaza lesz.

A *read* műveletet az ELEMHALMAZ-OLVAS algoritmus valósítja meg, az eredményt a $\mathcal{K}$-ban kapjuk meg.

ELEMHALMAZ-OLVAS($\mathcal{H}, Y$)
1 $\mathcal{K} \leftarrow \emptyset$
2 **for** minden $E \in H$
3 **do** $\mathcal{K} \leftarrow \mathcal{K} \cup$ ELEM-OLVAS(E, Y)
4 **return** $\mathcal{K}$

ELEM-OLVAS(E, Y)
1 **if** $E = [A \rightarrow \alpha.X\beta, a]$ és $X = Y$
2 **then** $\mathcal{K}_{E,Y} \leftarrow$ ELEM-LEZÁR($[A \rightarrow \alpha X.\beta, a]$)
3 **else** $\mathcal{K}_{E,Y} \leftarrow \emptyset$
4 **return** $\mathcal{K}_{E,Y}$

A második algoritmus 2. sorában látható, hogy az összes olyan LR(1)-elemet meghatározzuk, ami az olvasás utáni állapotot írja le.

Az $LR(1)$-elemek felsorolásának rövidebb leírása érdekében vezessük be a következő jelölést:

$$[A \rightarrow \alpha.X\beta, a/b]$$

jelentse az

$$[A \rightarrow \alpha.X\beta, a] \text{ és } [A \rightarrow \alpha.X\beta, b]$$

$LR(1)$-elemeket.

6.4.16. példa. A 6.4.13. példában szereplő nyelvtan egy $LR(1)$-eleme $[S' \rightarrow .S, \#]$. Erre
$closure([S' \rightarrow .S, \#]) = \{[S' \rightarrow .S, \#], [S \rightarrow .AA, \#], [A \rightarrow .aA, a/b],$
$[A \rightarrow .b, a/b]\}$. □

Az $LR(1)$-*elemek kanonikus halmazait,* vagy röviden az $LR(1)$-*kanonikus halmazokat* a következő módszerrel határozzuk meg:

6.4.17. értelmezés. *A $\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_m$ $LR(1)$-elemek* ***kanonikus halmazai*** *a következők:*

- *Legyen $\mathcal{H}_0 = closure([S' \rightarrow .S, \#])$,*
- *Ezután képezzük egy X szimbólumra a $read(\mathcal{H}_0, X)$ halmazt. Ha az így kapott halmaz nem üres, és nem egyezik meg a $\mathcal{H}_0$ kanonikus halmazzal, akkor legyen ez a következő kanonikus halmaz, azaz $\mathcal{H}_1$.*

  Ismételjük meg ezt a műveletet az összes lehetséges X terminális és nemterminális szimbólumra úgy, hogy ha olyan nem üres halmazt kapunk, amelyik nem egyezik meg egyik korábbi kanonikus halmazzal sem, akkor ez a halmaz legyen egy új kanonikus halmaz, és indexe legyen 1-gyel nagyobb, mint az eddigi maximális index.

- *Ezután ismételjük meg ezt a műveletet a már korábban előállított összes kanonikus halmazra és a nyelvtan minden szimbólumára, egészen addig, amíg csak új kanonikus halmazt kapunk.*

Az így létrehozott

$$\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_m$$

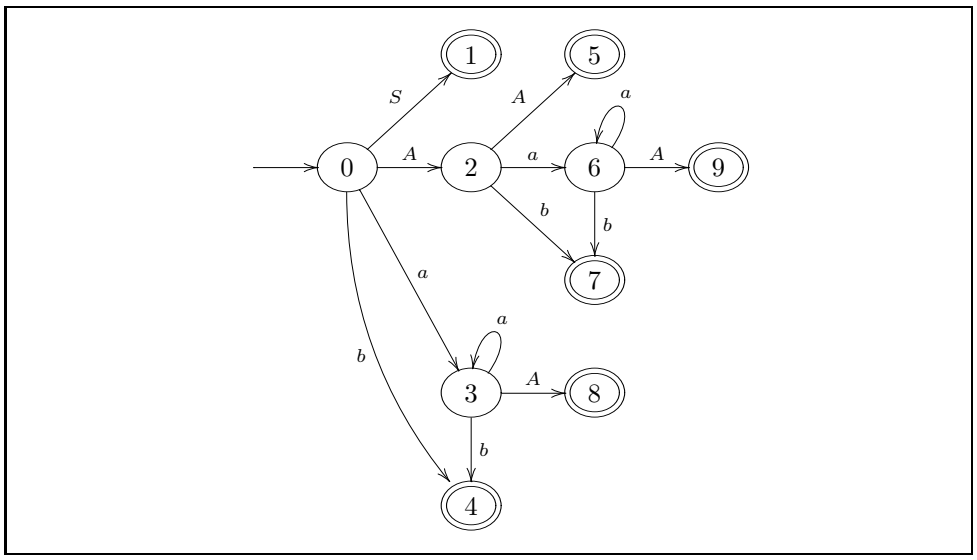
halmazokat nevezzük a G nyelvtan $LR(1)$-kanonikus halmazainak.

Mivel egy nyelvtanra az $LR(1)$-elemek darabszáma véges, az $LR(1)$-kanonikus halmazok létrehozása biztosan véges lépésben befejeződik.

A G nyelvtan kanonikus halmazait a következő algoritmus állítja elő:

```
KANONIKUS-HALMAZOKAT-KÉSZÍT(G)
 1 i ← 0
 2 H_i ← ELEM-LEZÁR([S' → .S, #])
 3 I ← {H_i}, K ← {H_i}
 4 repeat
 5         L ← K
 6         for minden M ∈ I-re
 7             do I ← I \ M
 8                 for minden X ∈ T ∪ N-re
 9                     do J ← ELEMHALMAZ-LEZÁR
                                (ELEMHALMAZ-OLVAS(M, X))
10                        if J ≠ ∅ és J ∉ K
11                          then i ← i + 1
12                               H_i ← J
13                               K ← K ∪ {H_i}
14                               I ← I ∪ {H_i}
15 until K = L
16 return K
```

Az algoritmus a K-ban adja a kanonikus halmazokat, a 2. sorban látható, hogy az első kanonikus halmaz a $\mathcal{H}_0$ lesz. További halmazokat a már meglévő kanonikus halmazokból az ELEMHALMAZ-LEZÁR(ELEMHALMAZ-OLVAS) függvénnyel képezünk a 9. sorban. A 10. sor programja azt vizsgálja, hogy ez az új halmaz vajon különbözik-e az eddigiektől, és ha igen, akkor a 11–12. sorban ez a halmaz egy új kanonikus halmaz lesz. A 6–14. sorok **for** ciklusa azt biztosítja, hogy ezeket a műveleteket a már meglévő minden kanonikus halmazra elvégezzük. A 3–14. sorokban lévő **repeat** ciklus szerint a kanonikus halmazok generálását addig végezzük, amíg új kanonikus halmazokat kapunk.



6.8. ábra. A 6.4.13. példa járható prefixeit felismerő automata.

6.4.18. példa. A 6.4.13. példában szereplő nyelvtan $LR(1)$-elemeinek kanonikus halmazai a következők:

$$
\begin{array}{lll}
\mathcal{H}_0 & = closure([S' \to .S, \#]) & = \{[S' \to .S, \#], [S \to .AA, \#], \\
 & & [A \to .aA, a/b], [A \to .b, a/b]\} \\
\mathcal{H}_1 = read(\mathcal{H}_0, S) & = closure([S' \to S., \#]) & = \{[S' \to S., \#]\} \\
\mathcal{H}_2 = read(\mathcal{H}_0, A) & = closure([S' \to A.A, \#]) & = \{[S \to A.A, \#], [A \to .aA, \#], \\
 & & [A \to .b, \#]\} \\
\mathcal{H}_3 = read(\mathcal{H}_0, a) & = closure([A \to a.A, a/b]) & = \{[A \to a.A, a/b], [A \to .aA, a/b], \\
 & & [A \to .b, a/b]\} \\
\mathcal{H}_4 = read(\mathcal{H}_0, b) & = closure([A \to b., a/b]) & = \{[A \to b., a/b]\} \\
\mathcal{H}_5 = read(\mathcal{H}_2, A) & = closure([S \to AA., \#]) & = \{[S \to AA., \#]\} \\
\mathcal{H}_6 = read(\mathcal{H}_2, a) & = closure([A \to a.A, \#]) & = \{[A \to a.A, \#], [A \to .aA, \#], \\
 & & [A \to .b, \#]\} \\
\mathcal{H}_7 = read(\mathcal{H}_2, b) & = closure([A \to b., \#]) & = \{[A \to b., \#]\} \\
\mathcal{H}_8 = read(\mathcal{H}_3, A) & = closure([A \to aA., a/b]) & = \{[A \to aA., a/b]\} \\
read(\mathcal{H}_3, a) & = \mathcal{H}_3 & \\
read(\mathcal{H}_3, b) & = \mathcal{H}_4 & \\
\mathcal{H}_9 = read(\mathcal{H}_6, A) & = closure([A \to aA., \#]) & = \{[A \to aA., \#]\} \\
read(\mathcal{H}_6, a) & = \mathcal{H}_6 & \\
read(\mathcal{H}_6, b) & = \mathcal{H}_7 &
\end{array}
$$

Az elemző automatája a 6.8. ábrán látható. □

6.4.3. Az *LR*(1) elemző

Ha egy G' kiegészített nyelvtanhoz meghatároztuk az $LR(1)$-elemek

$$\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_m$$

kanonikus halmazait, akkor egy automata k állapotához rendeljük hozzá a $\mathcal{H}_k$ halmazt. Az automata állapotai és az $LR(1)$-elemek kanonikus halmazai közötti kapcsolatot a következő, az ***LR*(1)-elemzés nagy tételének*** is nevezett állítás mondja ki:

6.4.19. tétel. *Egy γ járható prefixre érvényes $LR(1)$-elemek halmaza az a $\mathcal{H}_k$ kanonikus elemhalmaz, amelyik az elemző véges determinisztikus automatájának ahhoz a k állapotához tartozik, amelyikbe az automata a kezdőállapotból a γ hatására kerül.*

A tétel azt mondja ki, hogy a járható perfixeket felismerő automata felépíthető a kanonikus halmazok ismeretében. Állítsuk elő tehát az $LR(1)$-elemek kanonikus halmazaiból az $LR(1)$ elemzőt.

A járható prefixeket felismerő determinisztikus véges automata leírható egy táblázattal, ezt $LR(1)$ *elemző táblázatnak* nevezzük. A táblázat sorait az automata állapotaihoz rendeljük hozzá.

Az elemző táblázat két részből áll. Az első neve az *action* táblázat. Mivel az elemezendő szöveg szimbóluma határozza meg az elvégzendő műveletet, az *action* táblázatot oszlopokra bontjuk, és az oszlopokhoz a terminális szimbólumokat rendeljük. Az *action* táblázat azt tartalmazza, hogy az adott állapotban, ha az oszlophoz tartozó terminális szimbólum a bemenő jel, léptetést vagy redukciót kell-e végrehajtani. A léptetés műveletét jelöljük sj-vel, ahol s a léptetést, j a léptetés utáni állapotot jelenti. A redukció jele legyen ri, ahol i az alkalmazott helyettesítési szabály sorszáma. Mivel a nulladik szabály szerinti redukció azt jelenti, hogy elemzés befejeződött és az elemzett szöveg szintaktikusan helyes, jelöljük ezt a táblázatban az *elfogad* szóval.

A második rész a *goto* táblázat. Ebbe az az információ kerül, hogy a nemterminális szimbólumok hatására az automata egy adott állapotból melyik állapotba megy át. (A terminális szimbólumok állapot-átmeneteit az *action* táblázat sj bejegyzései tartalmazzák.)

Az automata állapotainak halmaza legyen a $\{0, 1, \ldots, m\}$ halmaz, az elemző táblázatok i-edik sorát a $\mathcal{H}_i$ $LR(1)$-elemeiből töltjük ki.

Az *action* táblázat i-edik sora:

- ha $[A \to \alpha.a\beta, b] \in \mathcal{H}_i$ és $read(\mathcal{H}_i, a) = \mathcal{H}_j$, akkor legyen $action[i, a] = sj$,
- ha $[A \to \alpha., a] \in \mathcal{H}_i$ és $A \neq S'$, akkor legyen $action[i, a] = rl$, ahol az $A \to \alpha$ a nyelvtan l-edik szabálya,

- ha $[S' \to S., \#] \in \mathcal{H}_i$, akkor legyen $action[i, \#] = elfogad$.

A *goto* táblázat kitöltésének módszere:

- ha $read(\mathcal{H}_i, A) = \mathcal{H}_j$, akkor legyen $goto[i, A] = j$.
- Mindkét táblázatban az üresen maradt helyeket a *hiba* szöveggel töltsük ki.

Az $LR(1)$-elemek kanonikus halmazaiból létrehozott *action* és *goto* táblázatokat $LR(1)$ vagy *kanonikus elemző táblázatoknak* nevezzük.

6.4.20. tétel. *A G' kiegészített nyelvtan akkor és csak akkor $LR(1)$ nyelvtan, ha a nyelvtanhoz készített kanonikus elemző táblázatok kitöltése konfliktusmentes.*

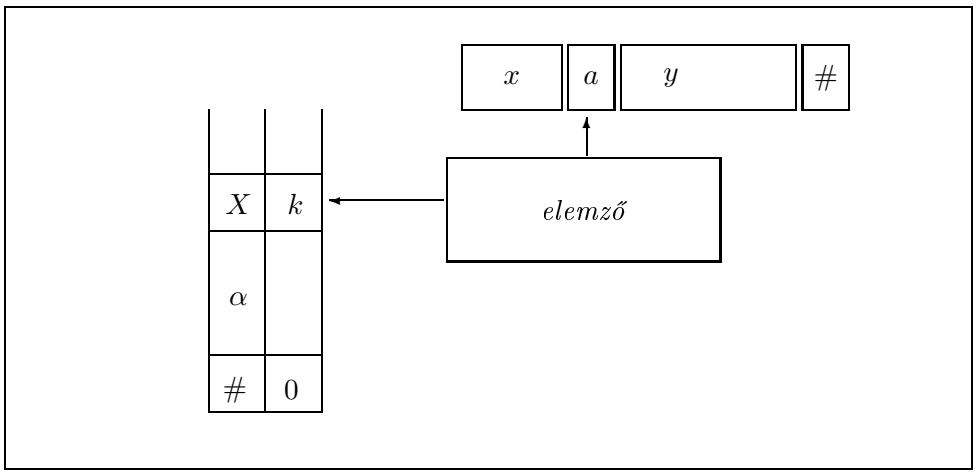
A táblázat kitöltését a következő algoritmussal végezhetjük:

LR(1)-TÁBLÁZATOT-KITÖLT(G)
1 **for** minden $\mathcal{H}_i$ LR(1) kanonikus halmazra
2 **do for** minden LR(1)-elemre
3 **if** $[A \to \alpha.a\beta, b] \in \mathcal{H}_i$ és $read(\mathcal{H}_i, a) = \mathcal{H}_j$
4 **then** $action[i, a] = sj$
5 **if** $[A \to \alpha., a] \in \mathcal{H}_i$ és $A \neq S'$ és $A \to \alpha$ az l-edik szabály
6 **then** $action[i, a] = rl$
7 **if** $[S' \to S., \#] \in \mathcal{H}_i$
8 **then** $action[i, \#] = elfogad$
9 **if** $read(\mathcal{H}_i, A) = \mathcal{H}_j$
10 **then** $goto[i, A] = j$
11 **for** minden $a \in (T \cup \{\#\})$
12 **do if** $action[i, a] =$ „üres"
13 **then** $action[i, a] \leftarrow hiba$
14 **for** minden $X \in N$
15 **do if** $goto[i, X] =$ „üres"
16 **then** $goto[i, X] \leftarrow hiba$
17 **return** *action*, *goto*

A táblázatokat soronként töltjük ki, a 2–6. sorokban az *action* táblázatot, a 9–10. sorokban a *goto* táblázatot. Az algoritmus 11–13. soraiban a táblázatok sorainak üresen maradt helyeire a szintaktikai hibát jelző *hiba* szöveget írjuk.

Az $LR(1)$ elemző működése a következőképpen adható meg (6.9. ábra).

Az elemző verme egy „dupla verem", azaz egy *push* vagy *pop* művelettel két információt írunk vagy olvasunk. A verem szimbólumpárokat tartalmaz, a párok első elemében egy terminális vagy nemterminális szimbólumot tárolunk, a második elemben pedig az automata állapotának sorszámát. A verem



6.9. ábra. Az $LR(1)$ elemző szerkezete.

kezdeti tartalma legyen $\#0$.

Az *elemző állapotát* egy kettőssel írjuk le, a kettős első eleme legyen a verem tartalma, a második elem pedig a bemenő szimbólumsorozat még nem elemzett része. Az elemző *kezdőállapota* tehát $(\#0, z\#)$, ahol z az elemezendő szimbólumsorozat. Az elemzés sikeresen befejeződik, azaz az elemző a *végállapotba* kerül, ha a verem tartalma ismét $\#0$, és az elemzéssel az elemezendő szimbólumsorozat végére értünk.

Tegyük fel, hogy az elemző pillanatnyi állapota a $(\#0 \dots Y_k i_k, ay\#)$ kettőssel írható le. Ekkor az elemző következő lépését az $action[i_k, a]$ adat határozza meg.

Az állapotátmenetek a következők:

- Ha $action[i_k, a] = sl$, azaz az automata egy léptetést hajt végre, akkor a bemenet soron következő a szimbóluma és az új állapot i_l sorszáma kerüljön a verembe, azaz

$$(\#0 \dots Y_k i_k, ay\#) \to (\#0 \dots Y_k i_k a i_l, y\#) \; .$$

- Ha $action[i_k, a] = rl$, akkor az l-edik szabály, az $A \to \alpha$ szabály szerint kell redukálni. Először töröljük a verem $|\alpha|$ darab sorát, azaz $2|\alpha|$ elemét. Ezután határozzuk meg a *goto* táblázatból, hogy az automata a törlés után a verem tetejére kerülő állapotból az A hatására melyik állapotba kerül, majd az A szimbólumot és a meghatározott állapotsorszámot írjuk be a verembe.

$$(\#0 \dots Y_{k-r} i_{k-r} Y_{k-r+1} i_{k-r+1} \dots Y_k i_k, y\#) \to$$

$$(\#0 \dots Y_{k-r} i_{k-r} A i_l, y\#) \ ,$$

ahol $|\alpha| = r$, és $goto[i_{k-r}, A] = i_l$.

- Ha $action[i_k, a] = elfogad$, akkor az elemzés a veremből való törlés után befejeződik, az elemző az elemzett szöveget elfogadja.
- Ha $action[i_k, a] = hiba$, akkor az elemzés befejeződik, az elemző az elemzett szövegben az a szimbólumnál egy *szintaktikai hibát* detektált.

Az $LR(1)$ elemzőt gyakran *kanonikus* $LR(1)$ *elemzőnek* is nevezik.

Ha T-vel jelöljük az *action* és *goto* táblákat, az elemző működésére a következő algoritmust adhatjuk meg.

```
LR(1)-ELEMEZ(xay#, T)
 1  s ← (#0, xay#), s' ← elemez
 2  repeat
 3        s = (#0 ... Y_{k-r} i_{k-r} Y_{k-r+1} i_{k-r+1} ... Y_k i_k, ay#)
 4        if action[i_k, a] = sl
 5           then s ← (#0 ... Y_k i_k a i_l, y#)
 6           else if action[i_k, a] = rl és A → α az l-edik szabály és
 7                    |α| = r és goto[i_{k-r}, A] = i_l
 8                    then s ← (#0 ... Y_{k-r} i_{k-r} A i_l, ay#)
 9                    else if action[i_k, a] = elfogad
10                             then s' ← O.K.
11                             else  s' ← HIBA
12  until s' = O.K. vagy s' = HIBA
13  return s', s
```

Az algoritmus bemenő paramétere az xay elemezendő szöveg és a T elemző táblázat. Az s' változó az elemző működését jelzi, működés közben az s' értéke *elemez*, az elemzés befejezésekor *O.K.* vagy *HIBA*. A 3. sorban az elemző állapotát írjuk fel részletesen, erre majd a 6–8. sorokban levő művelet esetén lesz szükség. Az elemző az automatának a verem tetején levő x_k állapota és az a aktuális szimbólum alapján a *action* táblázatból meghatározza az elvégzendő műveletet. A 4–5. sorban a léptetés műveletét hajtjuk végre, a 6–8. sorokban a redukálás művelete található. Az algoritmus a 9–11. sorban az elemzés befejezését jelzi, ha az elemezendő szöveg végére ért és a verem tetején a 0 állapot van, akkor az elemzett szöveg helyes, egyebként az elemző egy szintaktikai hibát fedezett fel. Az algoritmus végeredménye ennek megfelelően az *O.K.* vagy *HIBA* jelzés, és kimenetként mindkét esetben megjelenik az elemző állapota is. szintaktikai hiba esetén az elemző állapot második elemének első szimbóluma a hiba helyét adja meg.

6.4.21. példa. A 6.4.13. példában megadott nyelvtan $LR(1)$ elemzőjének *action* és *goto* táblázatai a következők, az üres helyek most is a *hibát* jelentik.

állapot	*action*			*goto*	
	a	b	$\#$	S	A
0	$s3$	$s4$		1	2
1			*elfogad*		
2	$s6$	$s7$			5
3	$s3$	$s4$			8
4	$r3$	$r3$			
5			$r1$		
6	$s6$	$s7$			9
7			$r3$		
8	$r2$	$r2$			
9			$r2$		

□

6.4.22. példa. Elemezzük az előző példában megadott táblázat felhasználásával az $abb\#$ szöveget.

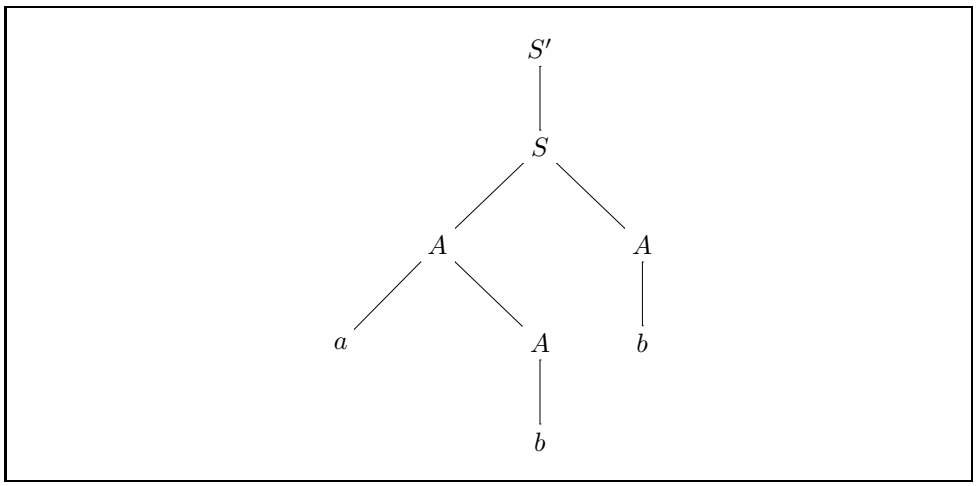
			szabály
$(\#0,$	$aab\#)$		
	$\xrightarrow{s3}$	$(\#0a3,\quad bb\#)$	
	$\xrightarrow{s4}$	$(\#0a3b4,\quad b\#)$	
	$\xrightarrow{r3}$	$(\#0a3A8,\quad b\#)$	$A \to b$
	$\xrightarrow{r2}$	$(\#0A2,\quad b\#)$	$A \to aA$
	$\xrightarrow{s7}$	$(\#0A2b7,\quad \#)$	
	$\xrightarrow{r3}$	$(\#0A2A5,\quad \#)$	$A \to b$
	$\xrightarrow{r1}$	$(\#0S1,\quad \#)$	$S \to AA$
	$\xrightarrow{elfogad}$	*O.K.*	

Az elemzett mondat szintaxisfája a 6.10. ábrán látható. □

6.4.4. Az $LALR(1)$ elemző

Mivel az elemző program állapotszámától nemcsak az elemző mérete, hanem a sebessége is függ, most célul az állapotok számának csökkentését tűzzük ki, és arra törekszünk, hogy ezzel az elemezhető nyelvek halmaza az $LR(1)$ nyelvekhez viszonyítva lényegesen ne csökkenjen.

Vegyük azt észre, hogy az $LR(1)$-elemek kanonikus halmazai között vannak olyan halmazpárok, hogy az egyik halmazban levő minden $LR(1)$-elemnek van egy megfelelője a másik halmazban, úgy, hogy ezeknek az elemeknek a magjuk azonos, és legfeljebb csak az előreolvasási szimbólumokban különböznek. Egyesítsük ezeket a halmazpárokat.



6.10. ábra. Az *aab* mondat szintaxisfája.

Ha a $\mathcal{H}_i$ és a $\mathcal{H}_j$ halmazok egyesíthetők, akkor legyen $\mathcal{K}_{[i,j]} = \mathcal{H}_i \cup \mathcal{H}_j$.

Végezzük el az $LR(1)$-kanonikus halmazok összes lehetséges egyesítését, az indexek átsorszámozása után az így kapott $\mathcal{K}_0, \mathcal{K}_1, \ldots, \mathcal{K}_n$ halmazokat nevezzük *egyesített* $LR(1)$ *kanonikus halmazoknak,* vagy $LALR(1)$*-kanonikus halmazoknak.*

Ezekből az egyesített halmazokból létrehozható elemzőt fogjuk majd $LALR(1)$ elemzőnek nevezni.

6.4.23. példa. A 6.4.18. példában szereplő kanonikus halmazok közül a következőket lehet egyesíteni:
$\mathcal{H}_3$ és $\mathcal{H}_6$,
$\mathcal{H}_4$ és $\mathcal{H}_7$,
$\mathcal{H}_8$ és $\mathcal{H}_9$.
A 6.8. ábrán is látható, hogy az összevonható halmazok az automatában azonos, vagy legalábbis hasonló részstruktúrát alkotnak. □

A *read* függvény az egyesített halmazokra nem okozhat problémát, azaz ha

$$\mathcal{K} = \mathcal{H}_1 \cup \mathcal{H}_2 \cup \ldots \cup \mathcal{H}_k \ ,$$

$$read(\mathcal{H}_1, X) = \mathcal{H}_1^{'}, read(\mathcal{H}_2, X) = \mathcal{H}_2^{'}, \ldots, read(\mathcal{H}_k, X) = \mathcal{H}_k^{'} \ ,$$

$$\mathcal{K}' = \mathcal{H}_1^{'} \cup \mathcal{H}_2^{'} \cup \ldots \cup \mathcal{H}_k^{'} \ ,$$

akkor

$$read(\mathcal{K}, X) = \mathcal{K}' \ .$$

Ezt a következőképpen lehet belátni. A *read* függvény értelmezése szerint a

$read(\mathcal{H}, X)$ csak a $\mathcal{H}$ $LR(1)$-elemeinek magjaitól függ, és nem függ az előreolvasási szimbólumoktól. Így, mivel a $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_k$ halmazokban az $LR(1)$-elemek magjai azonos halmazokat alkotnak, a

$$read(\mathcal{H}_1, X), read(\mathcal{H}_2, X), \ldots, read(\mathcal{H}_k, X)$$

$LR(1)$-elemeinek magjaiból alkotott halmazok is azonosak, tehát ezek a halmazok is egyesíthetők egy $\mathcal{K}'$ halmazba, és így valóban $read(\mathcal{K}, X) = \mathcal{K}'$.

Az $LR(1)$-elemek kanonikus halmazainak egyesítése után azonban az egyesített halmazon belül maguk az $LR(1)$-elemek okozhatnak problémát. Tegyük fel, hogy

$\mathcal{K}_{[i,j]} = \mathcal{H}_i \cup \mathcal{H}_j$.

- *Léptetés-léptetés konfliktus* az összevonás után nem léphet fel. Ha

  $$[A \to \alpha.a\beta, b] \in \mathcal{H}_i \ ,$$

  $$[B \to \gamma.a\delta, c] \in \mathcal{H}_j \ ,$$

  akkor az összevonás után az a szimbólumra továbbra is egy léptetést írunk elő, és a fentiekben láttuk, hogy a *read* függvény sem okoz problémát, azaz a $read(\mathcal{K}_{[i,j]}, a)$ éppen a $read(\mathcal{H}_i, a) \cup read(\mathcal{H}_j, a)$-val egyenlő.
- Ha $\mathcal{H}_i$ kanonikus halmazban egy

  $$[A \to \alpha.a\beta, b] \ ,$$

  a $\mathcal{H}_j$-ben egy

  $$[B \to \gamma., a]$$

  elem szerepelne, akkor az egyesítés után az a szimbólum miatt egy inadekvát állapotot kapnánk, *léptetés-redukálás konfliktus* lépne fel. Ez az eset azonban sohasem állhat fenn, mivel ekkor mindkét elemnek szerepelnie kell mind a $\mathcal{H}_i$, mind a $\mathcal{H}_j$ halmazban, legfeljebb csak az előreolvasási szimbólumokban különbözhetnek, hiszen ezért tudtuk egyesíteni őket. Tehát a $\mathcal{H}_j$ halmazban is kell lennie egy $[A \to \alpha.a\beta, c]$ elemnek. Ekkor pedig a 6.4.20. tétel alapján a nyelvtan nem lenne $LR(1)$ nyelvtan; már a $\mathcal{H}_j$ halmazból léptetés-redukálás konfliktus következne, az a szimbólumot előreolvasva nem lehetne eldönteni, hogy az elemzésben milyen műveletet kell alkalmazni.
- Az összevonás után azonban *redukálás-redukálás konfliktus* előfordulhat, az $LR(1)$ nyelvtan tulajdonságai ezt nem zárják ki. A következő példában egy ilyen esetet mutatunk be.

6.4.24. példa. Tekintsük a $G' = (\{S', S, A, B\}, \{a, b, c, d, e\}, P', S')$ nyelvtant, ahol a helyettesítési szabályok:
$S' \to S$
$S \to aAd \mid bBd \mid aBe \mid bAe$
$A \to c$
$B \to c$
A nyelvtan egy $LR(1)$ nyelvtan. Az ac járható prefixre az

$$\{[A \to c., d], [B \to c., e]\},$$

valamint a bc járható prefixre az

$$\{[A \to c., e], [B \to c., d]\}$$

$LR(1)$-elemek egy-egy kanonikus halmazt alkotnak.

A két halmaz egyesítése után redukálás-redukálás konfliktus lép fel. Ha a bemenő szimbólum d vagy e, a c mondatnyél azonosítható, de nem dönthető el, hogy az $A \to c$ és a $B \to c$ szabály szerinti redukciók közül melyiket kell végrehajtani. □

Most az $LALR(1)$ elemző táblázatainak kitöltési szabályait adjuk meg. Miután meghatároztuk az $LR(1)$-elemek

$$\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_m$$

kanonikus halmazait, egyesítsük egy halmazba azokat a kanonikus halmazokat, amelyekben az $LR(1)$-elemek magjaiból alkotott halmazok azonosak. Legyenek ezek a halmazok

$$\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_n \quad (n \leq m).$$

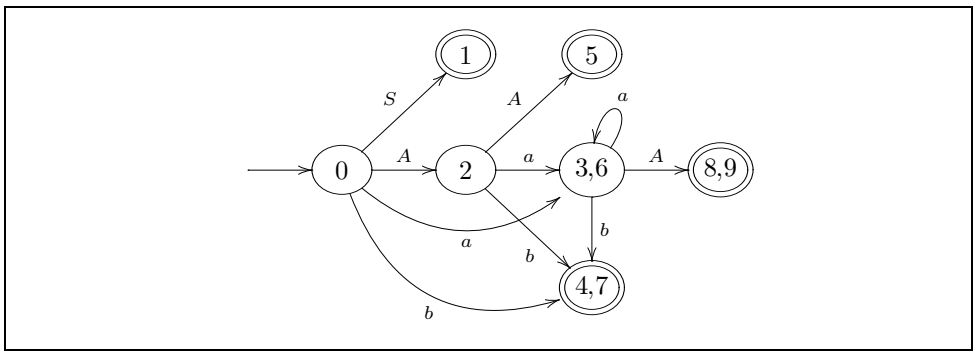
Az *action* és a *goto* táblázatok méretének a meghatározására és a táblázatok kitöltésére a $\mathcal{K}_i \quad (1 \leq i \leq n)$ halmazokat kell használni, a táblázatok kitöltésének módszere teljesen megegyezik az $LR(1)$ elemzőnél leírtakkal. A táblázatokat $LALR(1)$ *elemző táblázatoknak* nevezzük.

6.4.25. értelmezés. *Ha a G' kiegészített nyelvtanra a $LALR(1)$ elemző táblázatok kitöltése konfliktusmentes, akkor a nyelvtant* ***LALR(1) nyelvtannak*** *nevezzük.*

Az $LALR(1)$ elemző működése az $LR(1)$ elemző működésével egyezik meg.

6.4.26. példa. A $\mathcal{H}_i$ és a $\mathcal{H}_j$ kanonikus halmazok egyesítéséből származó $\mathcal{K}_{[i,j]}$ halmazhoz tartozó állapotot jelöljük $[i, j]$ -vel.

A 6.4.13. példában szereplő nyelvtan $LR(1)$-elemeinek kanonikus halmazait a 6.4.18. példában adtuk meg, és a 6.4.23. példában láttuk az egyesíthető halmazpárokat. Így a nyelvtanhoz a következő $LALR(1)$ elemző táblázatokat lehet elkészíteni.



6.11. ábra. A 6.4.26. példa járható prefixeit felismerő automatája.

állapot	*action*			*goto*	
	a	b	$\#$	S	A
0	$s\,[3,6]$	$s\,[4,7]$		1	2
1			*accept*		
2	$s\,[3,6]$	$s\,[4,7]$			5
$[3,6]$	$s\,[3,6]$	$s\,[4,7]$			$[8,9]$
$[4,7]$	$r3$	$r3$	$r3$		
5			$r1$		
$[8,9]$	$r2$	$r2$	$r2$		

Az $LALR(1)$-táblázatok kitöltése konfliktusmentes, a nyelvtan tehát egy $LALR(1)$ nyelvtan. Az elemző automatája a 6.11. ábrán látható. □

6.4.27. példa. Elemezzük az előző példában megadott táblázat felhasználásával az $abb\#$ szöveget.

			szabály
$(\#0,$	$aab\#)$		
	$\xrightarrow{s[3,6]}$	$(\#0a\,[3,6]\,,\quad bb\#)$	
	$\xrightarrow{s[4,7]}$	$(\#0a\,[3,6]\,b\,[4,7]\,,\quad b\#)$	
	$\xrightarrow{r3}$	$(\#0a\,[3,6]\,A[8,9],\quad b\#)$	$A \to b$
	$\xrightarrow{r2}$	$(\#0A2,\quad b\#)$	$A \to aA$
	$\xrightarrow{s[4,7]}$	$(\#0A2b\,[4,7]\,,\quad \#)$	
	$\xrightarrow{r3}$	$(\#0A2A5,\quad \#)$	$A \to b$
	$\xrightarrow{r1}$	$(\#0S1,\quad \#)$	$S \to AA$
	$\xrightarrow{elfogad}$	$O.K.$	

Az elemzett mondat szintaxisfája a 6.10. ábrán látható. □

Az $LALR(1)$ nyelvtanok egyben $LR(1)$ nyelvtanok is, mint az a fenti példából is látható, de ez fordítva nem áll fenn. A 6.4.24. példában éppen egy olyan nyelvtan szerepelt, amelyik $LR(1)$, de nem $LALR(1)$ nyelvtan.

A programnyelvek generálhatók $LALR(1)$ nyelvtannal, a programnyelvek fordítóprogramjaiban leggyakrabban alkalmazott elemzési módszer az $LALR(1)$ elemzés. Az $LALR(1)$ elemző előnye az $LR(1)$ elemzővel szemben az, hogy táblázatainak mérete lényegesen kisebb.

Például, a Pascal nyelvre az $LALR(1)$ -táblázatok néhányszor száz sort tartalmaznak, míg az $LR(1)$ elemző táblázatai több ezer sorból állnak.

Gyakorlatok

6-1. Határozzuk meg, hogy a következő nyelvtanok közül melyek $LL(1)$ nyelvtanok. A nyelvtanoknak csak a helyettesítési szabályait adjuk meg.

1. $S \rightarrow ABc$
   $A \rightarrow a \mid \varepsilon$
   $B \rightarrow b \mid \varepsilon$
2. $S \rightarrow Ab$
   $A \rightarrow a \mid B \mid \varepsilon$
   $B \rightarrow b \mid \varepsilon$
3. $S \rightarrow ABBA$
   $A \rightarrow a \mid \varepsilon$
   $B \rightarrow b \mid \varepsilon$
4. $S \rightarrow aSe \mid A$
   $A \rightarrow bAe \mid B$
   $B \rightarrow cBe \mid d$

6-2. Bizonyítsuk be, hogy a következő nyelvtanok $LL(1)$ nyelvtanok. A nyelvtanoknak csak a helyettesítési szabályait adjuk meg.

1. $S \rightarrow Bb \mid Cd$
   $B \rightarrow aB \mid \varepsilon$
   $C \rightarrow cC \mid \varepsilon$
2. $S \rightarrow aSA \mid \varepsilon$
   $A \rightarrow c \mid bS$
3. $S \rightarrow AB$
   $A \rightarrow a \mid \varepsilon$
   $B \rightarrow b \mid \varepsilon$

6-3. Bizonyítsuk be, hogy a következő nyelvtanok nem $LL(1)$ nyelvtanok. A nyelvtanoknak csak a helyettesítési szabályait adjuk meg.

1. $S \rightarrow aAa \mid Cd$
   $A \rightarrow abS \mid c$
2. $S \rightarrow aAaa \mid bAba$
   $A \rightarrow b \mid \varepsilon$
3. $S \rightarrow abA \mid \varepsilon$
   $A \rightarrow Saa \mid b$

6-4. Mutassuk meg, hogy az $LL(0)$ nyelvtannak csak egy mondata van.

6-5. Bizonyítsuk be, hogy a következő nyelvtanok $LR(0)$ nyelvtanok. A nyelvtanoknak csak a helyettesítési szabályait adjuk meg.

1. $S' \rightarrow S$
   $S \rightarrow aSa \mid aSb \mid c$
2. $S' \rightarrow S$
   $S \rightarrow aAc$
   $A \rightarrow Abb \mid b$

6-6. Bizonyítsuk be, hogy a következő nyelvtanok $LR(1)$ nyelvtanok. A nyelvtanoknak csak a helyettesítési szabályait adjuk meg.

1. $S' \rightarrow S$
   $S \rightarrow aSS \mid b$
2. $S' \rightarrow S$
   $S \rightarrow SSa \mid b$

6-7. Bizonyítsuk be, hogy a következő nyelvtanok nem $LR(k)$ nyelvtanok egyetlen k-ra sem. A nyelvtanoknak csak a helyettesítési szabályait adjuk meg.

1. $S' \rightarrow S$
   $S \rightarrow aSa \mid bSb \mid a \mid b$
2. $S' \rightarrow S$
   $S \rightarrow aSa \mid bSa \mid ab \mid ba$

6-8. Bizonyítsuk be, hogy a következő nyelvtanok $LR(1)$, de nem $LALR(1)$ nyelvtanok. A nyelvtanoknak csak a helyettesítési szabályait adjuk meg.

1. $S' \rightarrow S$
   $S \rightarrow Aa \mid bAc \mid Bc \mid bBa$
   $A \rightarrow d$
   $B \rightarrow d$

2. $S' \rightarrow S$
   $S \rightarrow aAcA \mid A \mid B$
   $A \rightarrow b \mid Ce$
   $B \rightarrow dD$
   $C \rightarrow b$
   $D \rightarrow CcS \mid CcD$

6-9. A fenti példákban szereplő $LL(1)$ nyelvtanokhoz készítsük el az elemző táblázatokat.

6-10. A fenti példákban szereplő $LL(1)$ nyelvtanokhoz írjuk meg a rekurzív leszállás elemző programjait.

6-11. A fenti példákban szereplő $LR(1)$ nyelvtanokhoz készítsük el a kanonikus halmazokat és az elemző táblázatokat.

6-12. A fenti példákban szereplő $LALR(1)$ nyelvtanokhoz készítsük el az összevont kanonikus halmazokat és az elemző táblázatokat.

7. FEJEZET

A szemantikai elemzés

A szintaktikus elemzés meghatározta az elemzendő szöveg szintaxisfáját. A szintaxisfa pontjaihoz olyan attribútumokat rendelünk, amelyek leírják az adott pont tulajdonságait. Ezeknek az attribútumoknak a meghatározása és az attribútumok konzisztenciájának vizsgálata a szemantikai elemzés feladata lesz.

Mint a 6. fejezetben már említettük, a szemantikai elemző a ***statikus szemantikával***, azaz az olyan tulajdonságok vizsgálatával foglalkozik, amelyek nem írhatók le környezetfüggetlen nyelvtannal. A szemantikai elemzők általában a következő tulajdonságokat vizsgálják:

- változók deklarációja és a változók hatásköre, láthatósága,
- változók többszörös deklarációja, a deklaráció hiánya,
- operátorok és operandusaik közötti típuskompatibilitás,
- eljárások, tömbök formális és aktuális paraméterei közötti kompatibilitás,
- túlterhelések egyértelműsége.

Bár történtek kísérletek arra, hogy a szemantikai elemzőt környezetfüggő nyelvtanból készítsék el, ezt a módszert elvetették, mert az elemzők rendkívül bonyolultak és lassúak voltak. A szemantikai elemzésre az a módszer terjedt el, hogy az egyes szemantikai tulajdonságok vizsgálatára önálló programokat írnak.

7.1. A fordítási nyelvtanok

A helyettesítési szabályokban speciális jelekkel, ***akciószimbólumokkal*** jelölhetjük azt, hogy a szintaxisfa felépítése folyamán az adott szabály alkalmazása esetén szemantikai elemzési tevékenységeket is kell végezni. Ha a szemantikai tevékenységet egy *proc* eljárás írja le, akkor ezt az eljárást ***szemantikai rutinnak*** nevezzük és a hozzátartozó akciószimbólumot *@proc*-cal jelöljük.

A $@s$ akciószimbólum az $S \overset{*}{\Longrightarrow} \alpha @s \beta \overset{*}{\Longrightarrow} x$ levezetésben azt jelenti, hogy az elemzéskor a $@s$ sorra kerülése esetén az s eljárást kell meghívni, és az elemzés csak ennek a programelemnek a lefutása után folytatódhat.

7.1.1. értelmezés. *Ha egy G környezetfüggetlen nyelvtan szabályainak jobb oldalait akciószimbólumokkal egészítjük ki, akkor a nyelvtant* ***fordítási nyelvtannak*** *nevezzük.*

A fordítási nyelvtanokat TG-vel jelöljük.

7.1.2. példa. Legyen egy nyelvtannak az

⟨*értékadó-utasítás*⟩ → ⟨*változó*⟩ := ⟨*kifejezés*⟩

egy szabálya, és az ebből kialakított fordítási nyelvtannak a szabálya legyen

⟨*értékadó-utasítás*⟩ → ⟨*változó*⟩ := ⟨*kifejezés*⟩ *@CheckType.*

A *@CheckType* akciószimbólumhoz tartozó program a ⟨*változó*⟩ és a ⟨*kifejezés*⟩ típusának azonosságát vizsgálja. □

A szemantikai elemzést megnehezítheti az, hogy az akciószimbólumokkal jelölt szemantikai rutinoknak nincs paraméterük, az akciószimbólumok által jelölt eljárásokba bele kell építeni azt, hogy az eljárások a szükséges paramétereket hol találják meg.

Mivel mind a felülről lefelé, mind az alulról felfelé elemzések az elemzéshez egy vermet használnak, célszerűnek látszik, hogy a szimbólumok attribútumait is egy verembe helyezzük el, és ez a verem a szemantikai rutinok egymás közötti paraméterátadására is szolgálhat. Ezt a vermet ***szemantikai veremnek*** hívjuk. A szemantikai rutinok paraméterátadása megvalósítható, hiszen a paraméterek az SP veremmutatóhoz relatív címzéssel elérhetők.

A továbbiakban azonban egy ennél sokkal jobb és általánosabb módszerrel foglalkozunk. A TG szimbólumaihoz attribútumokat rendelünk, ezek az attribútumok szolgálnak majd arra, hogy a szemantikai információt továbbítsák, és ebből majd a szemantikai elemző programot is generálni tudjuk.

7.2. Attribútum fordítási nyelvtanok

Legyen TG egy fordítási nyelvtan. A továbbiakban jelöljük a TG fordítási nyelvtan *akciószimbólumainak halmazát* $@\mathcal{S}$-sel, és jelöljük X-szel a nyelvtan tetszőleges terminális vagy nemterminális szimbólumát, vagy egy tetszőleges akciószimbólumát, azaz legyen $X \in T \cup N \cup @\mathcal{S}$.

Legyen $\mathcal{A}$ egy véges, nem üres halmaz, az ***attribútumok halmaza***. Ekkor minden X szimbólumhoz rendeljük hozzá az $\mathcal{A}$ egy $\mathcal{A}(X)$ részhalmazát, speciálisan, a $@s$ szimbólumra $\mathcal{A}(@s)$ legyen az s szemantikai rutin paramétereinek halmaza.

Az $x \in L(TG)$ mondat szintaxisfájában az X ponthoz az $\mathcal{A}(X)$-beli attribútumok *egy-egy példányát* rendeljük, tehát a szintaxisfa két különböző helyén levő X pontban két azonos nevű attribútum között nincs semmilyen kapcsolat.

A továbbiakban jelöljük az X szimbólum $a \in \mathcal{A}(X)$ attribútumát $X.a$-val. Az attribútumokat a nyelvtan helyettesítési szabályaiba, a szimbólumok megismétlése nélkül, közvetlenül a szimbólum mellé is beírhatjuk. Ha egy szimbólumnak több attribútuma is van, akkor ezeket vesszővel választjuk el. Ha egy helyettesítési szabályban egy szimbólum többször is előfordul, akkor a szimbólumokat indexeléssel különböztetjük meg, utalva arra, hogy azonos nevű attribútumaik különbözőek.

Ha két különböző szimbólumhoz azonos nevű attribútumokat rendelünk, akkor az attribútum elé a hozzátartozó szimbólumot mindig meg kell adni. Az a azonossága az $X.a$ és az $Y.a$ $(X \neq Y)$ attribútumokban semmilyen összefüggésre nem utal, az attribútumok között lévő kapcsolatot a szemantikai függvények írják le.

Az ***attribútumértékek halmazát*** jelöljük $\mathcal{V}$-vel, a $\mathcal{V}$-re semmilyen megkötést nem teszünk.

Legyen $\mathcal{R}$ a ***szemantikai függvények halmaza***, és minden $p \in P$ helyettesítési szabályhoz rendeljük hozzá az $\mathcal{R}$ egy $\mathcal{R}(p)$ részhalmazát. Ha a $p \in P$ helyettesítési szabályhoz tartozó attribútum-előfordulások halmazát $\mathcal{A}(p)$-vel jelöljük, akkor az $\mathcal{R}(p)$ minden szemantikai függvényének minden argumentuma az $\mathcal{A}(p)$ egy eleme, és értéke az $\mathcal{A}(p)$ egy elemének az értéke. A $@s$ akciószimbólumhoz tartozó s eljárást egy szemantikai függvény programjának tekintjük.

Az X szimbólum attribútumértékeinek egyértelműeknek kell lenniük, azaz minden $x \in L(TG)$ mondatra az x-hez tartozó szintaxisfa minden X pontjában minden attribútum értékét legfeljebb csak egy szemantikai függvény határozhatja meg. Megjegyezzük, ha $p : X \to \alpha$ és $q : Y \to \beta X \gamma$ két helyettesítési szabály, akkor lehetnek olyan $\mathcal{A}(X)$-beli attribútumok is, amelyek értékének meghatározására sem $\mathcal{R}(p)$-ben, sem $\mathcal{R}(q)$-ban nincs szemantikai függvény.

Legyen $\mathcal{C}$ a ***logikai feltételek halmaza***, és minden $p \in P$ helyettesítési szabályhoz rendeljük hozzá a $\mathcal{C}$ egy $\mathcal{C}(p)$ elemét. A $\mathcal{C}(p)$ egy logikai állítást mond ki a p helyettesítési szabályhoz tartozó $\mathcal{A}(p)$ attribútumokra. Ha az állítás a p szabály feldolgozásakor az $\mathcal{A}(p)$ attribútumokra nem teljesül, akkor a szemantikai elemzőnek hibajelzést kell adnia. A $\mathcal{C}$ halmaz lehet üres halmaz is. Tehát a p szabályhoz tartozó szemantikai tulajdonságokat a $\mathcal{C}(p)$ kiértékelésével tudjuk ellenőrizni.

7.2.1. értelmezés. *Az* $ATG = (TG, \mathcal{A}, \mathcal{V}, \mathcal{R}, \mathcal{C})$ *ötöst* ***attribútum fordítási nyelvtannak*** *nevezzük, ahol*

- TG *egy fordítási nyelvtan,*
- $\mathcal{A}$ *az attribútumok halmaza,*
- $\mathcal{V}$ *az attribútumértékek halmaza,*
- $\mathcal{R}$ *a szemantikai szabályok halmaza, és*
- $\mathcal{C}$ *a logikai feltételek halmaza.*

Az $X \in T \cup N$ szimbólum egy $X.a$ attribútumát ***szintetizáltnak*** nevezzük, ha értékét egy szemantikai függvény abban az esetben határozza meg, amikor az X szimbólum egy helyettesítési szabály bal oldalán áll. Az $X.a$ attribútum ***örökölt***, ha értékét egy szemantikai függvény akkor határozza meg, amikor az X szimbólum egy helyettesítési szabály jobb oldalán áll. Tehát az információt egy szintaxisfában a szintetizált attribútumok alulról-felfelé, az örökölt attribútumok pedig felülről-lefelé és egy helyettesítési szabály jobb oldalát alkotó pontokon továbbítják.

A nyelvtan S kezdőszimbólumához nem tartoznak örökölt attribútumok, és a terminális szimbólumokhoz nem tartoznak szintetizált attribútumok. A terminális szimbólumoknak azonban vannak „szintetizált jellegű", úgynevezett ***kitüntetett szintetizált attribútumaik***. Ilyen attribútum például egy konstans terminális szimbólum esetén a konstans értéke és típusa, vagy egy azonosító szimbólum esetén a szimbólum neve. Feltehetjük azt, hogy a kitüntetett szintetizált attribútumok értékét konstans értékű szemantikai függvények határozzák meg, és ezeket a konstans értékeket egy, az attribútum nyelvtantól független külső eljárás, a lexikális elemző adja át.

A $@s$ akciószimbólum $@s.a$ attribútuma *örökölt*, ha az a az s eljárás bemenő paramétere, és a $@s.a$ attribútum *szintetizált*, ha az a az s eljárás kimenő paramétere, azaz értékét az s eljárás határozza meg.

Jelölje a továbbiakban egy ATG nyelvtan $X \in T \cup N \cup @\mathcal{S}$ szimbólumának szintetizált és örökölt attribútumait $\mathcal{S}(X)$ és $\mathcal{I}(X)$.

Legyen $AF(p)$ az $\mathcal{R}(p)$ szemantikai függvények által *meghatározott értékű attribútumok halmaza*, azaz legyen

$$AF(p) = \{X.a \mid X.a = f(\ldots) \in \mathcal{R}(p)\}.$$

Az $X.a$ szintetizált attribútum, ha létezik egy olyan $p : X \to \alpha$ helyettesítési szabály, melyre $X.a \in AF(p)$, és az $X.a$ örökölt attribútum, ha létezik egy olyan $q : A \to \alpha X \beta$ helyettesítési szabály, melyre $X.a \in AF(q)$.

Ha az X szimbólum $X.a$ attribútumának szintetizált vagy örökölt jellegét is meg akarjuk különböztetni, akkor az attribútumot $X \uparrow a$ vagy $X \downarrow a$-val jelöljük. Az $X \uparrow a, b \downarrow c, d$ az $X \uparrow a$, $X \uparrow b$, $X \downarrow c$ és $X \downarrow d$ rövid jelölése. A

nyelvtanok helyettesítési szabályaiban az $X\uparrow a, b\downarrow c, d$ az X szimbólumot és a szimbólum négy attribútumát jelöli. Az akciószimbólumok attribútumait, utalva arra, hogy ezek egy eljárás paraméterei, zárójelbe tesszük, például $@s(\uparrow a, b, \downarrow c, d)$.

Ha $X \in T \cup N$, és $X.a \in \mathcal{S}(X) \cap \mathcal{I}(X)$ lenne, akkor létezne egy olyan $p : X \to \alpha$ helyettesítési szabály, melyre $X.a \in AF(p)$, és létezne egy olyan $q : Y \to \beta X \gamma$ szabály, melyre $X.a \in AF(q)$. Mivel $S \overset{*}{\Longrightarrow} \varphi Y \psi \Longrightarrow \varphi\beta X\gamma\psi \Longrightarrow \varphi\beta\alpha\gamma\psi \overset{*}{\Longrightarrow} x$, az $X.a$ kiszámítására legalább kettő szemantikai függvény lenne, ami ellentmond a kiszámíthatóságra tett feltételnek.

Hasonlóan, ha $@s \in @\mathcal{S}$, és $@s.a \in \mathcal{I}(@s)$, akkor létezik olyan p szabály, amelyre $@s \in AF(p)$. Ha $@s.a \in \mathcal{S}(@s)$ is fennállna, akkor a $@s.a$ értékét az s szemantikai rutin is meghatározná, ami szintén ellentmond a kiszámíthatóságára tett feltételnek.

Így a szintetizált és örökölt attribútumokra érvényes a következő állítás:

7.2.2. tétel. *Az ATG nyelvtan minden $X \in T \cup N \cup @\mathcal{S}$ szimbólumára $\mathcal{S}(X) \cap \mathcal{I}(X) = \emptyset$.*

Ha egy szintaxisfa pontjaiban meg akarjuk határozni az attribútumok értékét, akkor először csak a levelekhez tartozó kitüntetett szintetizált attribútumok értékei ismertek. A szemantikai elemzés fogja az összes többi attribútum értéket meghatározni. Az egyes értékek meghatározásának sorrendje közömbös, de azt megköveteljük, hogy egy szemantikai függvény alkalmazása esetén az argumentumok értéke már ismert legyen.

Először vizsgáljuk meg azt, hogy egy helyettesítési szabály attribútumai közül melyeket kell a szabály alkalmazásakor meghatároznunk.

7.2.3. értelmezés. *Egy attribútum fordítási nyelvtant* ***teljes attribútum fordítási nyelvtannak*** *nevezünk, ha minden $p : X_0 \to X_1X_2 \ldots X_n$ helyettesítési szabályra*

- $\mathcal{S}(X_i) \subseteq AF(p)$, *ha* $i = 0$, *vagy* $X_i = @s$,
- $\mathcal{I}(X_i) \subseteq AF(p) \quad (1 \leq i \leq n)$,
- $\mathcal{A}(X_i) = \mathcal{S}(X_i) \cup \mathcal{I}(X_i) \quad (0 \leq i \leq n)$.

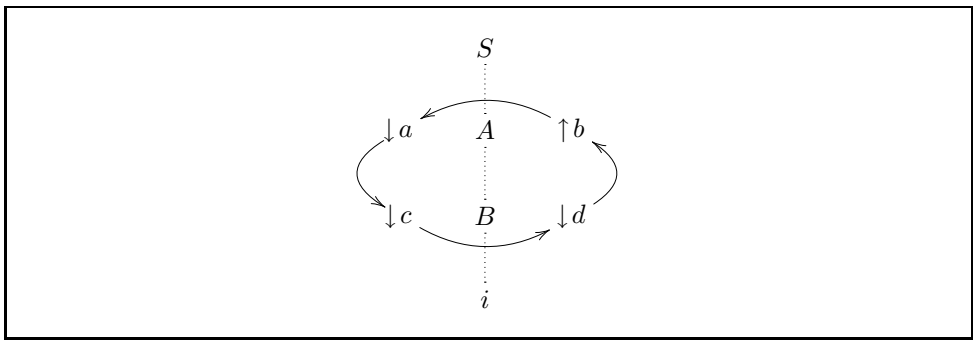
A teljes ATG azonban csak az egyszintű részfákra biztosítja az attribútumok kiszámíthatóságát, ami még nem jelenti azt, hogy egy szintaxisfa minden attribútumának az értéke meghatározható.

7.2.4. példa. Legyen egy ATG-ben

$N = \{S, A, B\}, \quad T = \{i\},$

$\mathcal{A} = \{A\downarrow a, A\uparrow b, B\downarrow c, B\downarrow d\},$

és a P és az $\mathcal{R}$ halmaz a következő:



7.1. ábra. Az $S \overset{+}{\Longrightarrow} i$ levezetés attribútumai

$$
\begin{array}{lcll}
S & \to & A{\downarrow}a{\uparrow}b & \{A{\downarrow}a \leftarrow A{\uparrow}b\} \\
A{\downarrow}a{\uparrow}b & \to & B{\downarrow}c,d & \{A{\uparrow}b \leftarrow B{\downarrow}d,\ B{\downarrow}d \leftarrow B{\downarrow}c,\ B{\downarrow}c \leftarrow A{\downarrow}a\} \\
B{\downarrow}c,d & \to & i & \emptyset
\end{array}
$$

Látható, hogy a nyelvtan egy teljes attribútum nyelvtan. Az $S \overset{+}{\Longrightarrow} i$ levezetéshez tartozó, attribútumokkal kiegészített szintaxisfa a 7.1. ábrán látható. Az ábrán (és a további ábrákon is,) a szintaxisfa éleit pontokkal, az attribútum értékadásokat nyilakkal ábrázoljuk. Az attribútumok egy gráf csúcspontjai, és az attribútumok értékeinek meghatározását a gráf élei jelölik.

Az ábráról leolvasható, hogy az attribútumok értékeit nem tudjuk meghatározni, mivel az attribútumok értékeit meghatározó gráf egy kört tartalmaz. □

A fordításhoz természetesen olyan ATG-k kellenek, amelyekben a szintaxisfák összes attribútumának az értéke meghatározható.

7.2.5. értelmezés. *Egy attribútum fordítási nyelvtant* ***jól definiált attribútum fordítási nyelvtannak*** *nevezünk, ha a nyelvtan által generált nyelv minden mondatára teljesül, hogy a mondat szintaxisfájának minden pontjában minden attribútum értéke egyértelműen kiszámítható.*

Nyilvánvaló, hogy minden jól definiált attribútum fordítási nyelvtan teljes, de ez fordítva nem áll fenn, mint azt a 7.2.4. példában is láttuk.

Ha egy $X.a$ attribútum értéke szükséges az $Y.b$ attribútum értékének meghatározásához, akkor ezt az $(X.a, Y.b)$ párossal jelöljük.

7.2.6. értelmezés. *A* $p : X_0 \to X_1 X_2 \dots X_n$ *helyettesítési szabályhoz tartozó* ***direkt attribútumfüggőségek*** *a következők:*

$$DP(p) = \{(X_i.a, X_j.b) \ \mid \ X_j.b = f(\dots, X_i.a, \dots) \in \mathcal{R}(p), \quad (0 \le i, j \le n)\ \}.$$

A direkt attribútumfüggőségek egy adott szintaxisfára egy ***függőségi gráfot*** generálnak, ahol a gráf pontjai az attribútumok, az irányított élek pedig a fenti kapcsolatot leíró relációk. A 7.2.4. példában éppen egy ilyen függőségi gráfot ábrázoltunk.

Egy attribútum fordítási nyelvtant ***lokálisan aciklikusnak*** nevezünk akkor, ha minden $p \in P$ helyettesítési szabályra a $DP(p)$ függőségi gráf körmentes.

Legyen $DT(x)$ az x mondat levezetésében felhasznált összes p szabályhoz tartozó $DP(p)$ direkt attribútum függőségek halmaza. A $DT(x)$-hez tartozó direkt attribútumfüggőségeket gráfban is ábrázolhatjuk.

A teljesség, a jól definiáltság és a $DT(x)$ direkt attribútumfüggőségre teljesül a következő állítás.

7.2.7. tétel. *Egy teljes attribútum fordítási nyelvtan jól definiált, ha a nyelvtan által generált nyelv minden x mondatára a $DT(x)$ gráf nem tartalmaz kört.*

A jól definiáltság tehát azt jelenti, hogy a nyelvtan nem csak lokálisan aciklikus, hanem minden mondatának szintaxisfájára teljesül az is, hogy az attribútumok között nincs cirkuláris függőség. A fordítóprogramokban nyilvánvaló, hogy csak jól definiált attribútum fordítási nyelvtanok alkalmazhatók.

A jól definiált ATG-kben az attribútumok értékei meghatározhatók. Ezekkel a nyelvtanokkal a problémát azonban az okozza, hogy a függőségi gráfok körmentességét csak exponenciális idejű algoritmussal lehet eldönteni. Ezért a gyakorlatban az attribútum fordítási nyelvtanokra a jól definiáltságon kívül további megszorításokat kell tennünk.

7.3. Particionált attribútum fordítási nyelvtanok

A feladatunk az, hogy minden X szimbólumra olyan attribútumfüggőségeket határozzunk meg, amelyek megadják az attribútumok kiszámítási sorrendjét. Ha $(X.a, X.b)$ eleme ennek a függőségnek, akkor az X minden előfordulására az $X.a$ kiszámításának meg kell előznie az $X.b$ kiszámítását. Ha az $(X.a, X.b)$, $(X.b, X.a)$ függőségek egyikét sem tudjuk megállapítani, akkor az $X.a$ és $X.b$ kiszámítási sorrendje tetszőleges. Ezután, a szimbólumokra megállapított függőségeket figyelembe véve, a szabályok attribútumainak értékét kiszámító programot fogjuk megadni.

Ezt természetesen nem lehet minden ATG-re elvégezni. Mint majd látni fogjuk, az attribútumok kiszámítási sorrendje csak a „particionált" attribútum nyelvtanokra adható meg, ezekben minden szabályhoz megadható a szabály attribútumait bejáró „látogatási sorozat", amelyet követve az attribútumértékeket a meghatározott relációknak megfelelő sorrendben határozhatjuk meg.

7.3.1. értelmezés. *Az $\mathcal{A}(X)$ egy $\mathcal{A}_1(X), \mathcal{A}_2(X), \ldots, \mathcal{A}_{m(X)}(X)$ particionálását* ***megengedett particionálásnak*** *nevezzük, ha*

- $\mathcal{A}_{m(X)}(X), \mathcal{A}_{m(X)-2}(X), \mathcal{A}_{m(X)-4}(X), \ldots \subseteq \mathcal{S}(X)$,
- $\mathcal{A}_{m(X)-1}(X), \mathcal{A}_{m(X)-3}(X), \mathcal{A}_{m(X)-5}(X), \ldots \subseteq \mathcal{I}(X)$.

Megjegyezzük, hogy a fenti értelmezésben egyes $\mathcal{A}_i(X)$ halmazok lehetnek üres halmazok is.

7.3.2. értelmezés. *Egy ATG-t* ***particionáltnak*** *nevezünk, ha*

- *lokálisan aciklikus, és*
- *minden X szimbólumához létezik egy olyan $\mathcal{A}_1(X), \mathcal{A}_2(X), \ldots, \mathcal{A}_{m(X)}(X)$ megengedett particionálás, hogy az X attribútumainak értékei a partíciók növekvő sorrendjében meghatározhatók.*

Mivel egy particionált ATG-ben minden attribútum kiszámítható, minden particionált ATG egyben jól definiált is, de majd mint látni fogjuk, ez fordítva nem áll fenn.

A kiszámíthatóság miatt az X szimbólum $\mathcal{A}_i(X)$ $(1 \leq i \leq m(X))$ partícióit azoknak a p szabályoknak a $DP(p)$ direkt függőségei figyelembevételével kell meghatározni, amely p szabályokban az X szimbólum szerepel. A $DP(p)$ reláció tranzitív, ezért először képezzük a $DP(p)$ reláció $NDP(p)$ normalizált tranzitív lezárását.

7.3.3. értelmezés. $NDP(p) = DP^+(p) \setminus \{(X.a, X.b) \mid X.a, X.b \in AF(p)\}$, *az $NDP(p)$ relációkat* ***normalizált direkt függőségeknek*** *nevezzük.*

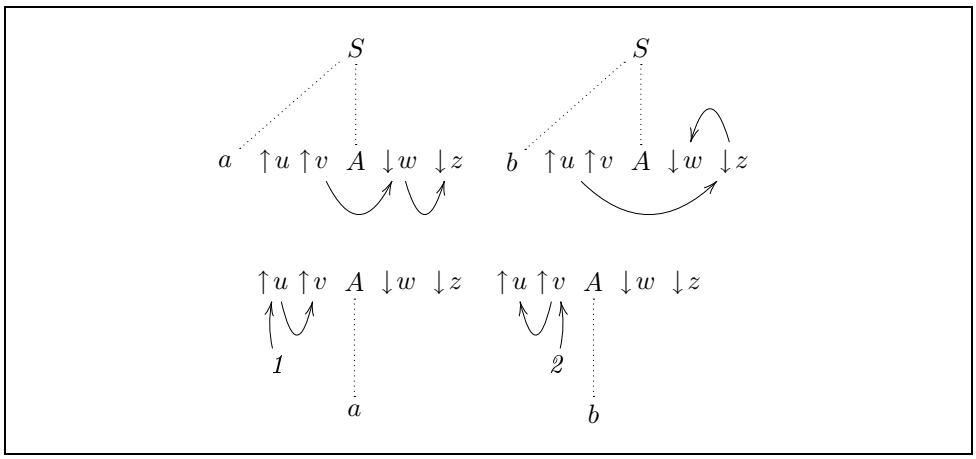
Az $NDP(p)$ reláció tehát nem tartalmazza azokat a függőségeket, amelyeknek mindkét komponensét egy-egy $\mathcal{R}(p)$-beli függvény határozza meg. Ennek a normalizálásnak a jelentőségét a 7.3.5. példában mutatjuk meg.

A $p : X \to \alpha$ és $q : B \to \beta X \gamma$ szabályok esetén az X szimbólum attribútumainak meghatározásához mind az $\mathcal{R}(p)$, mind az $\mathcal{R}(q)$ szükséges, hiszen $\mathcal{S}(X) \subseteq AF(p)$, és $\mathcal{I}(X) \subseteq AF(q)$. Azt is mondhatjuk, hogy az X az interfész az $\mathcal{R}(p)$ és $\mathcal{R}(q)$ között. Az X szimbólum interfész jellegéből és a $DP(p)$ reláció tranzitivitásából következnek az X-re vonatkozó indukált függőségek. Jelölje $IDP(p)$ a p szabályra vonatkozó, és $IDS(X)$ az X szimbólum attribútumai között fennálló *indukált attribútumfüggőségeket.*

7.3.4. értelmezés. *Egy ATG* ***indukált attribútumfüggőségeit*** *a következőképpen határozzuk meg:*

1. *Minden $p \in P$-re legyen $IDP(p) = NDP(p)$,*
2. *minden X-re legyen*

   $IDS(X) = \{(X.a, X.b) \mid \exists\, q \in P$, *melyre* $(X.a, X.b) \in IDP^+(q)\}$,



7.2. ábra. A 7.3.5. példa *DP* függőségei

3. *minden* $p : X_0 \to X_1X_2...X_n \in P$*-re legyen*
   $IDP(p) = IDP(p) \cup IDS(X_0) \cup IDS(X_1) \cup \ldots \cup IDS(X_n)$,
4. *a 2. és 3. lépést addig kell ismételni, amíg az IDP és IDS halmazok változnak.*

Megjegyezzük, hogy az értelmezés 2. lépésében szereplő q szabály vagy $X \to \alpha$, vagy $A \to \alpha X \beta$ alakú.

Az $IDS(X)$ halmaz tehát már tartalmazza azokat az $(X.a, X.b)$ függőségeket is, amelyeket az X gyökérpontú részfa attribútumainak kiértékelésekor tudunk meghatározni, és egy X bal oldalú helyettesítési szabály esetén tartalmazza azokat a függőségeket is, amelyek egy olyan szabályból származnak, ahol X a jobb oldal egy szimbóluma.

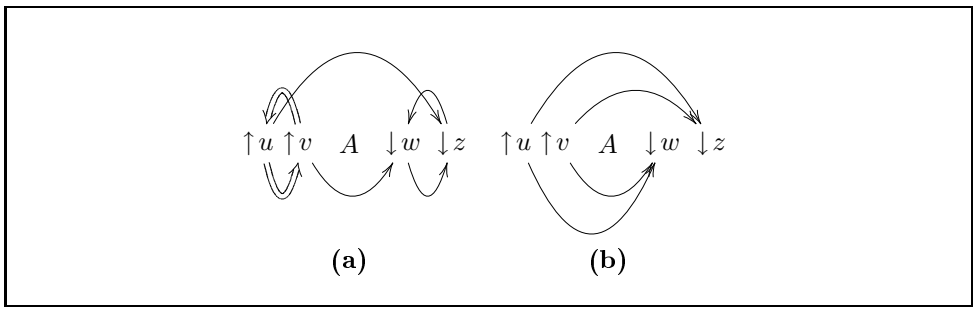
Az *IDS* képzésekor kapott új függőségeket az ábrákon dupla nyíllal fogjuk jelölni.

Most megmutatjuk, hogy az *IDS* relációk meghatározásában milyen szerepet játszik az, hogy az értelmezésben nem a *DP*-t, hanem az *NDP* normalizált tranzitív lezárást használtuk.

7.3.5. példa. Legyen egy *ATG* nyelvtan P és $\mathcal{R}$ halmaza a következő (7.2. ábra):

$$
\begin{array}{lcll}
S & \to & aA\uparrow u, v\downarrow w, z & \{A\downarrow w \leftarrow A\uparrow v, A\downarrow z \leftarrow A\downarrow w\} \\
S & \to & bA\uparrow u, v\downarrow w, z & \{A\downarrow z \leftarrow A\uparrow u, A\downarrow w \leftarrow A\downarrow z\} \\
A\uparrow u, v\downarrow w, z & \to & a & \{A\uparrow u \leftarrow 1, A\uparrow v \leftarrow A\uparrow u\} \\
A\uparrow u, v\downarrow w, z & \to & b & \{A\uparrow v \leftarrow 2, A\uparrow u \leftarrow A\uparrow v\}
\end{array}
$$

Ha az $IDP(p)$-nek a 7.3.4. értelmezés első lépésében a *DP* halmazt választjuk, akkor



7.3. ábra. Az $IDS(A)$ függőségek **(a)** a DP és **(b)** az NDP függőségekből

$IDS(A) = \{(A.u, A.v), (A.u, A.z), (A.v, A.u), (A.v, A.w), (A.w, A.z), (A.z, A.w)\}$,

azaz az $IDS(A)$ két kört is tartalmaz (7.3.(a) ábra), míg az $IDP(p) = NDP(p)$ választással

$IDS(A) = \{(A.u, A.w), (A.u, A.z), (A.v, A.w), (A.v, A.z)\}$,

azaz az $IDS(A)$ körmentes (7.3.(b) ábra).

Ugyanakkor az is látható, hogy az

- $\mathcal{A}_1(A) = \{A.u, A.v\}, \mathcal{A}_2(A) = \{A.w, A.z\}, \ \mathcal{A}_3(A) = \emptyset$

választással a nyelvtan particionált. □

Az $IDP(p)$ és $IDS(X)$ függőségek pesszimálisak abban az értelemben, hogy miközben akkumulálják a p-re és az X-re vonatkozó függőségeket, az összes függőség egyidejű jelenlétét is feltételezik, azaz X-re olyan függőségeket is előírhatnak, amelyek egyszerre egy szintaxisfa egyik X csúcsában sem fordulnak elő. Ezt mutatjuk meg a következő példában.

7.3.6. példa. Legyen egy ATG nyelvtan P és $\mathcal{R}$ halmaza a következő (7.4. ábra):

S	$\rightarrow$	$A{\downarrow}x{\uparrow}y$	$\{A{\downarrow}x \leftarrow 1\}$
$A{\downarrow}x{\uparrow}y$	$\rightarrow$	$aB{\downarrow}u, v{\uparrow}w, z$	$\{A{\uparrow}y \leftarrow B{\uparrow}z, B{\downarrow}u \leftarrow A{\downarrow}x, B{\downarrow}v \leftarrow B{\uparrow}w\}$
$A{\downarrow}x{\uparrow}y$	$\rightarrow$	$bB{\downarrow}u, v{\uparrow}w, z$	$\{A{\uparrow}y \leftarrow B{\uparrow}w, B{\downarrow}u \leftarrow B{\uparrow}z, B{\downarrow}v \leftarrow A{\downarrow}x\}$
$B{\downarrow}u, v{\uparrow}w, z$	$\rightarrow$	c	$\{B{\uparrow}w \leftarrow 2, B{\uparrow}z \leftarrow B{\downarrow}v\}$
$B{\downarrow}u, v{\uparrow}w, z$	$\rightarrow$	d	$\{B{\uparrow}w \leftarrow B{\downarrow}u, B{\uparrow}z \leftarrow 3\}$

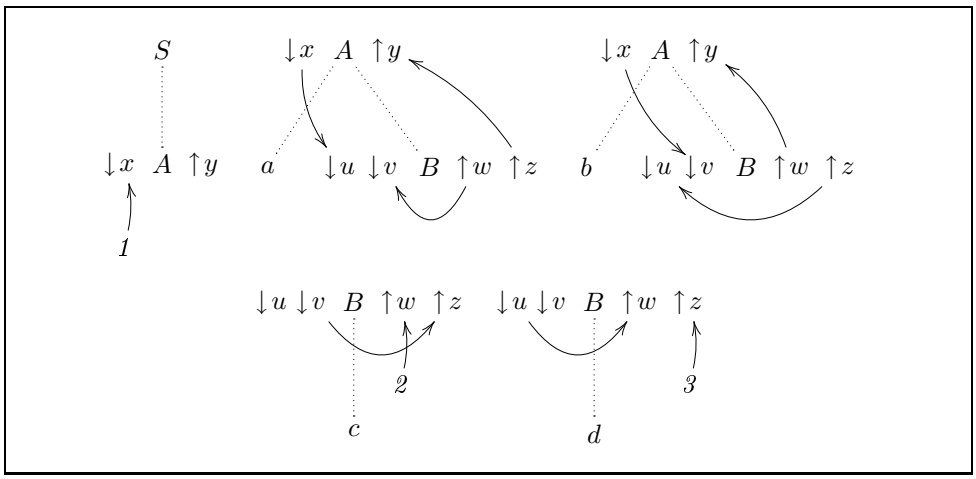
Látható, hogy

$IDS(B) = \{(B{\uparrow}w, B{\downarrow}v), (B{\uparrow}z, B{\downarrow}u), (B{\downarrow}v, B{\uparrow}z), (B{\downarrow}u, B{\uparrow}w)\}$,

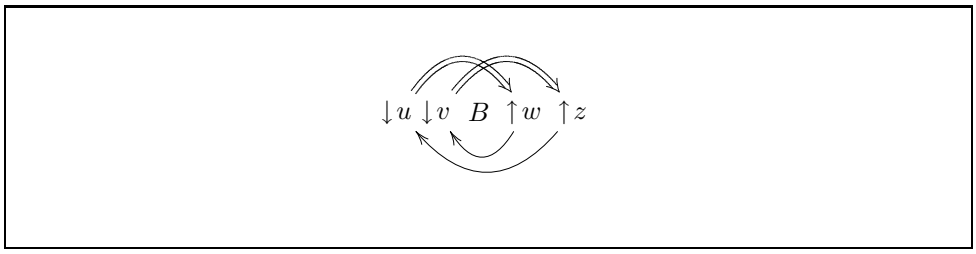
azaz az $IDS(B)$ egy kört tartalmaz (7.5. ábra).

A nyelvtannak azonban nincs egyetlen mondata sem, amelyre egyidejűleg mind a négy reláció fennállna. □

Az $IDS(X)$ halmazok és a particionált attribútum fordítási nyelvtanok közötti kapcsolatot mutatja a következő tétel.



7.4. ábra. A 7.3.6. példa *DP* függőségei



7.5. ábra. Az *IDS*(B) függőségek kört tartalmaznak

7.3.7. tétel. *Ha egy ATG particionált, akkor minden p-re és X-re az IDP(p) és IDS(X) függőségek gráfja nem tartalmaz kört. Ha $(X.a, X.b) \in IDS(X)$, akkor $X.a \in \mathcal{A}_i(X)$ és $X.b \in \mathcal{A}_j(X)$, ahol $i \leq j$.*

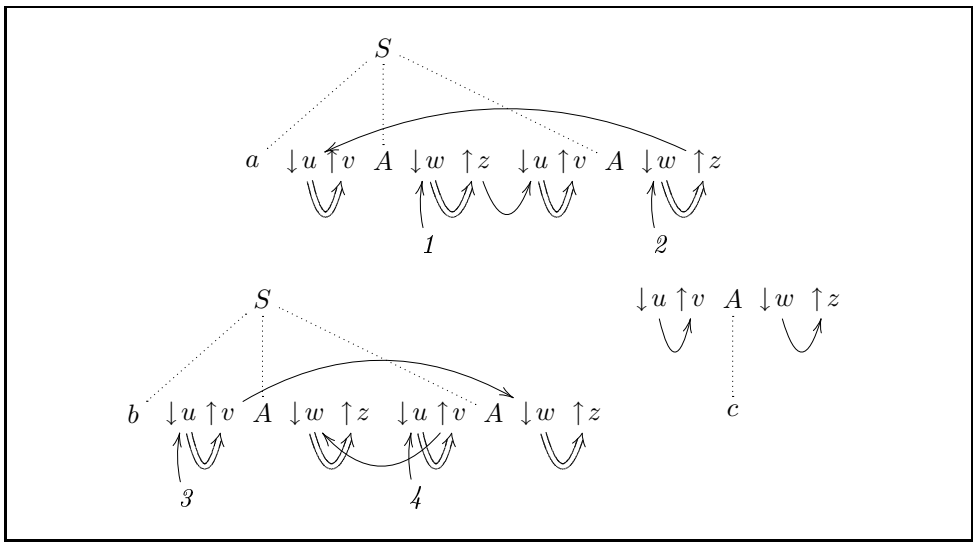
A 7.3.6. példában szereplő nyelvtan szemantikai függvényeiről azonnal látható, hogy a nyelvtan jól definiált. A fenti tétel alapján azonban nem lehet particionált, hiszen az *IDS*(B) egy kört tartalmazott.

A tétel egy szükséges, de nem elégséges feltételt ad a megengedett particionálásra. A következő példa azt mutatja, hogy az *IDP* és *IDS* gráfok körmentessége valóban nem elegendő a megengedett partíciók meghatározásához.

7.3.8. példa. Legyen egy *ATG* P és $\mathcal{R}$ halmaza a következő:

$$
\begin{array}{lll}
S & \rightarrow aA_1{\downarrow}u{\uparrow}v{\downarrow}w{\uparrow}zA_2{\downarrow}u{\uparrow}v{\downarrow}w{\uparrow}z & \{A_1{\downarrow}u \leftarrow A_2{\uparrow}z, A_1{\downarrow}w \leftarrow \mathtt{1}, \\
 & & A_2{\downarrow}u \leftarrow A_1{\uparrow}z, A_2{\downarrow}w \leftarrow \mathtt{2}\} \\
S & \rightarrow bA_1{\downarrow}u{\uparrow}v{\downarrow}w{\uparrow}zA_2{\downarrow}u{\uparrow}v{\downarrow}w{\uparrow}z & \{A_1{\downarrow}u \leftarrow \mathtt{3}, A_1{\downarrow}w \leftarrow A_2{\uparrow}v, \\
 & & A_2{\downarrow}u \leftarrow \mathtt{4}, A_2{\downarrow}w \leftarrow A_1{\uparrow}v\} \\
A{\downarrow}u{\uparrow}v{\downarrow}w{\uparrow}z & \rightarrow c & \{A{\uparrow}v \leftarrow A{\downarrow}u, A{\uparrow}z \leftarrow A{\downarrow}w\}
\end{array}
$$

A 7.6. ábrán is látható, hogy az *IDP* és az *IDS* gráfok nem tartalmaznak kört.



7.6. ábra. A 7.3.8. példa *IDP* és *IDS* függőségei

Az $IDS(A) = \{(A.u, A.v), (A.w, A.z)\}$, és a 7.3.7. tételt a következő particionálások elégíti ki:

- $\{A.u\}, \{A.v\}, \{A.w\}, \{A.z\}$,
- $\{A.w\}, \{A.z\}, \{A.u\}, \{A.v\}$,
- $\{A.u, A.w\}, \{A.v, A.z\}$.

Az $S \Longrightarrow aAA \Longrightarrow acc$ és az $S \Longrightarrow bAA \Longrightarrow bcc$ levezetésekből azonban látszik, hogy egyik particionálás sem használható az A attribútumainak meghatározására. □

Mint az előbbi példából is kitűnik, a problémát az okozza, hogy egy particionálás további függőségeket indukál, hiszen két különböző partícióból vett attribútumpárra a partíciók automatikusan egy relációt írnak elő, függetlenül attól, hogy eredetileg volt-e közöttük reláció. Bővítsük ki az *IDP* függőségeket ezekkel a relációkkal.

7.3.9. értelmezés. *Legyen* $p : X_0 \to X_1X_2...X_n$ *és legyen* $\mathcal{A}_1(X_i), \mathcal{A}_2(X_i), \ldots, \mathcal{A}_{m(X_i)}(X_i)$ *az* $\mathcal{A}(X_i)$ $(0 \leq i \leq n)$ *egy megengedett particionálása. Az* $EDP(p)$ ***kiterjesztett attribútumfüggőségek*** *legyenek a következők:*

$$EDP(p) = IDP(p) \cup \{(X_i.a, X_i.b) \mid X_i.a \in \mathcal{A}_j(X_i), X_i.b \in \mathcal{A}_k(X_i), (0 \leq i \leq n, 0 \leq j < k \leq m(X_i))\}.$$

A kiterjesztett függőségekre vonatkozik a következő tétel.

7.3.10. tétel. *Egy ATG nyelvtan akkor és csak akkor particionált, ha minden* $p \in P$*-re az* $EDP(p)$ *gráfok körmentesek.*

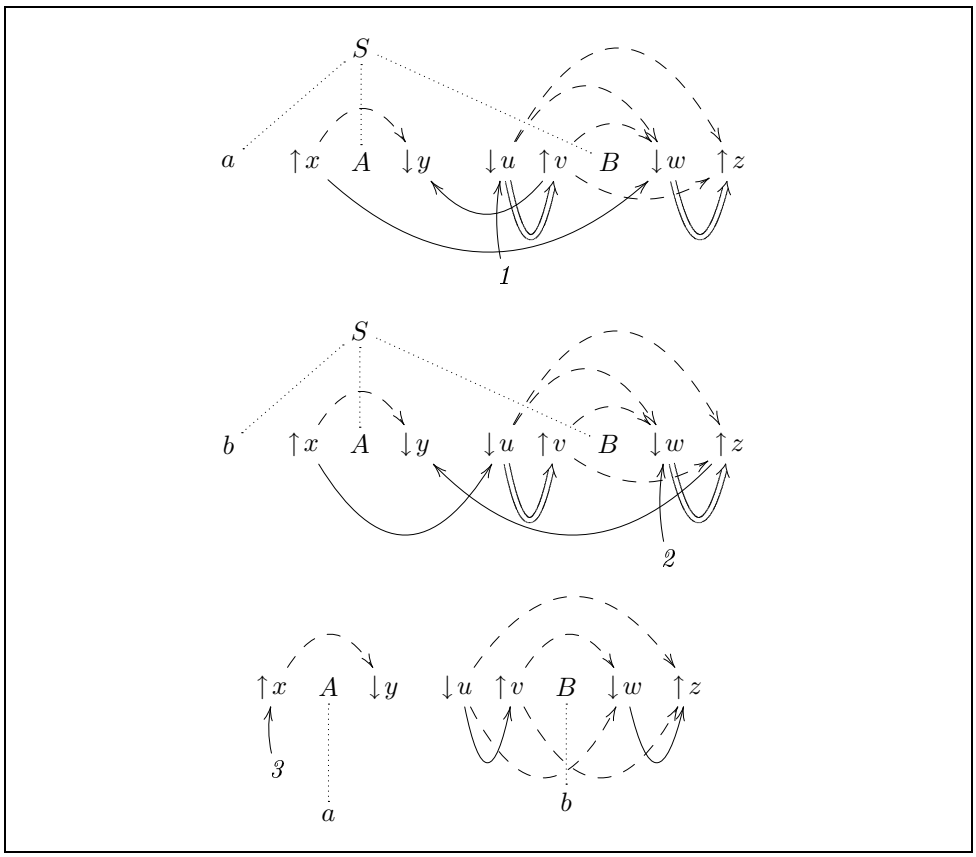
A tétel sajnos nem ad módszert a partíciók meghatározására, ezért a particionálásra először intuitív módszereket alkalmazunk. A következő szakaszban a particionált attribútum fordítási nyelvtanok speciális részhalmazára egy olyan algoritmust fogunk adni, amellyel az attribútumok partíciói is meghatározhatók.

7.3.11. példa. Határozzuk meg a következő nyelvtan partícióit.

S	$\rightarrow$	$aA\uparrow x\downarrow yB\downarrow u\uparrow v\downarrow w\uparrow z$	$\{A\downarrow y \leftarrow B\uparrow v, B\downarrow w \leftarrow A\uparrow x, B\downarrow u \leftarrow 1\}$
S	$\rightarrow$	$bA\uparrow x\downarrow yB\downarrow u\uparrow v\downarrow w\uparrow z$	$\{A\downarrow y \leftarrow B\uparrow z, B\downarrow w \leftarrow 2, B\downarrow u \leftarrow A\uparrow x\}$
$A\uparrow x\downarrow y$	$\rightarrow$	a	$\{A\uparrow x \leftarrow 3\}$
$B\downarrow u\uparrow v\downarrow w\uparrow z$	$\rightarrow$	b	$\{B\uparrow z \leftarrow B\downarrow w, B\uparrow v \leftarrow B\downarrow u\}$

Mivel $IDS(A) = \emptyset$ és $IDS(B) = \{(B.u, B.v), (B.w, B.z)\}$, az A-ra és B-re egy nyilvánvaló particionálás a következő:

- $\mathcal{A}_1(A) = \{A.y\}, \mathcal{A}_2(A) = \{A.x\}$,
- $\mathcal{A}_1(B) = \{B.u, B.w\}, \mathcal{A}_2(B) = \{B.v, B.z\}$,



7.7. ábra. A 7.3.11. példa *EDP* függőségei

de ez a particionálás az $EDP(1)$-ben és az $EDP(2)$-ben is kört okoz, hiszen

$(A.y, A.x), (B.w, B.v) \in EDP(1)$,

$(A.y, A.x), (B.u, B.z) \in EDP(2)$.

Azonban az

- $\mathcal{A}_1(A) = \{A.x\},\ \mathcal{A}_2(A) = \{A.y\},\ \mathcal{A}_3(A) = \emptyset$
- $\mathcal{A}_1(B) = \{B.u\},\ \mathcal{A}_2(B) = \{B.v\},\ \mathcal{A}_3(B) = \{B.w\},\ \mathcal{A}_4(B) = \{B.z\}$

már megengedett particionálások lesznek, és az *EDP* gráfok sem tartalmaznak kört. Az *EDP* függőségek gráfjában (7.7. ábra) a particionálásból származó új relációkat szaggatott nyíllal jelöltük. □

7.3.1. Látogatási sorozatok

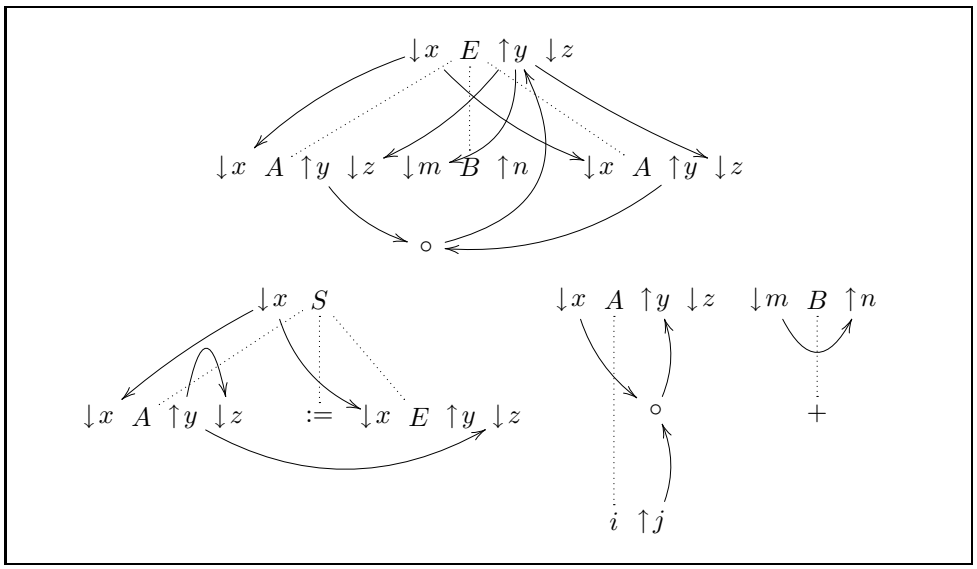
Most az attribútumértékeket meghatározó algoritmussal foglalkozunk. Az algoritmus a következő elven alapul: minden $p \in P$ szabályhoz meghatározzuk a szabályhoz tartozó szintaxisfa egy olyan $VP(p)$ bejárását, amely a p szabályhoz tartozó attribútumokat értékeli ki. A bejárást ***látogatási sorozatnak*** nevezzük.

Tegyük fel, hogy a particionált attribútum fordítási nyelvtanhoz már minden X szimbólumra meghatároztuk az $\mathcal{A}_1(X), \mathcal{A}_2(X), \ldots, \mathcal{A}_{m(X)}(X)$ partíciókat. Most határozzuk meg a $p : X_0 \to X_1 X_2 \ldots X_n$ helyettesítési szabály $VP(p)$ látogatási sorozatát:

1. Először meghatározzuk az attribútumok közötti összes függőséget úgy, hogy az $EDP(p)$-n kívül a $DP(p)$ relációkat is figyelembe vesszük, hiszen az $EDP(p)$ meghatározásában csak az $NDP(p)$ normalizált attribútumfüggőségeket vettük figyelembe. Tüntessük fel az $A_k(X_i) = \emptyset$ halmazokat is, az ábrákon ezeket az $\emptyset$ jellel jelöljük. Az attribútumfüggőségek az attribútumok között egy parciális rendezést adnak.
2. Jól ismert az az állítás, hogy minden parciálisan rendezett halmaz topologikus rendezése eredményül egy olyan lineáris rendezést ad, amelyik a parciális rendezés relációit megtartja. A topologikus rendezés egy egyszerű algoritmussal megvalósítható. Végezzük el az attribútumok topologikus rendezését.
3. A topologikus rendezés eredményeként az attribútumokra kapott lineáris rendezés egy utat határoz meg, ezt az utat bejárva minden attribútumot pontosan egyszer érintünk. Ez az út lesz a $VP(p)$ látogatási sorozat.
4. Mivel minden $VP(p)$-nek a $\mathcal{C}(p)$ logikai feltétel kiértékelésével kell befejeződnie, ezért az előbbi pontban meghatározott út utolsó pontja után helyezzünk el egy olyan pontot, ahol a $\mathcal{C}(p)$ logikai feltétel kiértékelése a feladat.

5. Egy látogatási sorozat pontjaihoz a következő eljárást rendeljük: A p szabályhoz tartozó eljárás neve legyen `P(p)`, az eljárás törzsét a `begin` és `end` kulcsszavak közé írjuk, a törzs utolsó utasítása legyen a `return` utasítás. Mint majd látni fogjuk, egy eljárás több „rész-eljárásból" is állhat, az eljárások részeit, amelyek egyébként önmaguk is önálló eljárások, egy `P(p)`-n belül az `1`, `2`, ... címkékkel jelöljük. A `P(p)` eljárás első utasításának címkéje legyen `1`.

   - A látogatási sorozat minden pontjához egy utasítást rendelünk, az utasítás megjegyzés mezőjébe az attribútumokat írjuk.
   - Ha a ponthoz nem tartozik attribútum, azaz itt az attribútumok halmaza az üres halmaz, akkor ehhez a ponthoz a `nop` utasítást rendeljük.
   - Ha a pont egy $AF(p)$-beli attribútum, akkor az attribútum értékét az attribútumhoz tartozó szemantikai függvénnyel biztosan itt határozzuk meg, így ehhez a ponthoz rendeljünk hozzá egy `eval` utasítást, amelynek operandusa az attribútum. Megjegyezzük, hogy ezek az attribútumok az X_0 szintetizált attribútumai, az X_i $(i \neq 0)$ szimbólum örökölt és kitüntetett szintetizált attribútumai.
   - Az X_0 szimbólum első örökölt attribútumának értékét nem a p szabályban határozzuk meg, a p szabályhoz tartozó program hívásakor ez az attribútum már biztosan ismert. Ehhez a ponthoz rendeljünk egy `nop` utasítást.
   - Az X_0 szimbólum nem első örökölt attribútumát sem itt határozzuk meg, hanem a p-t hívó szabály programjában. Ezért ehhez a ponthoz rendeljünk egy `return` utasítást. Ez az utasítás az elemzéskor visszalép az elemzett mondat szintaxisfájában a p-t meghívó programhoz. A `P(p)` eljárás következő meghívásakor az eljárást a `return` utáni utasítással kell kezdeni, ezért a következő utasításnak a címkéje legyen a `P(p)`-n belüli eddigi maximális címkénél eggyel nagyobb. Ez a szám éppen azt jelzi, hogy a látogatási sorozat bejárásakor az X_0 szimbólum hányadik örökölt attribútumánál tartunk.
   - Ha a ponthoz az X_i $(i \neq 0)$ szimbólum szintetizált attribútuma tartozik, akkor ezt az értéket sem ebben a programban határozzuk meg, hanem a $q : X_i \rightarrow \alpha$ szabályhoz tartozó `P(q)` eljárásban. Ezért ehhez a ponthoz rendeljünk hozzá egy olyan `call P(q):n` eljáráshívást, amelyik a vezérlést a `P(q)` program `n` címkéjű utasítására adja át. Az `n` jelzi, hogy a `P(p)` eljárásban hányadszor hívjuk meg a `P(q)` programot, azaz a látogatási sorozat bejárásakor az X_i szimbólum hányadik szintetizált attribútumánál tartunk.



7.8. ábra. A 7.3.12. példa *DP* függőségei

A particionált attribútum fordítási nyelvtanok esetén az attribútumértékek meghatározásának sorrendje tehát független a nyelv mondataitól, és csak a nyelvtantól függ. Ezért az attribútumkiértékelő algoritmus már a compiler generálásakor meghatározható, és például $LL(1)$ vagy $LALR(1)$ nyelvtanok esetén az elemző táblázattal egyidőben elkészíthető.

7.3.12. példa. Egy értékadó utasítást és egy egyszerű kifejezést feldolgozó nyelvtan legyen a következő:

$$
\begin{array}{lll}
S{\downarrow}x & \to\ A{\downarrow}x{\uparrow}y{\downarrow}z := E{\downarrow}x{\uparrow}y{\downarrow}z & \{A{\downarrow}x \leftarrow S{\downarrow}x, A{\downarrow}z \leftarrow A{\uparrow}y, \\
 & & \quad E{\downarrow}x \leftarrow S{\downarrow}x, E{\downarrow}z \leftarrow A{\uparrow}y\} \\
E{\downarrow}x{\uparrow}y{\downarrow}z & \to\ A_1{\downarrow}x{\uparrow}y{\downarrow}z B{\downarrow}m{\uparrow}n\ A_2{\downarrow}x{\uparrow}y{\downarrow}z & \{A_1{\downarrow}x \leftarrow E{\downarrow}x, A_2{\downarrow}x \leftarrow E{\downarrow}x, \\
 & & \quad E{\uparrow}y \leftarrow A_1{\uparrow}y \circ A_2{\uparrow}y, \\
 & & \quad A_1{\downarrow}z \leftarrow E{\uparrow}y, A_2{\downarrow}z \leftarrow E{\uparrow}y, \\
 & & \quad B{\downarrow}m \leftarrow E{\uparrow}y\} \\
A{\downarrow}x{\uparrow}y{\downarrow}z & \to\ i & \{A{\uparrow}y \leftarrow A{\downarrow}x \circ i{\uparrow}j\} \\
B{\downarrow}m{\uparrow}n & \to\ + & \{B{\uparrow}n \leftarrow B{\downarrow}m\}
\end{array}
$$

A nyelvtan szabályaiból az $i := i + i$ utasítás szimbólumsorozata vezethető le, a megadott szemantikai függvények pedig típusellenőrzést végeznek.

Tegyük fel, hogy az $S{\downarrow}x$ attribútum adott, és az x attribútum az utasítás feltételezett, várt típusát írja le. Az x attribútumok mindenhol ezt a feltételezett típust tartalmazzák.

A j attribútumban van az i konstans tényleges típusa, az $A \to i$ szabály alkalmazásakor a $\circ$ jellel jelölt típusellenőrzés, esetleg típusmódosítás hajtódik végre, az eredményül kapott típus az $A.y$-ba kerül.

Az $E \to A_1 B A_2$ szabály alkalmazásakor a + művelet két operandusának típusát kell ellenőrizni, esetleg módosítani, a $\circ$-val jelölt művelet eredményének típusa az $E.y$-ba kerül, ahonnan az $A_1.z$, $A_2.z$ és $B.m$ attribútumokba jut. Az $A_1.z$ és $A_2.z$ attribútumok tehát az összeadás művelet bemenő értékeinek az összeadás elvégzéséhez szükséges típusát, és így az eredmény típusát adják meg. A $B.m$ az összeadás típusát tartalmazza, a $B.n$ pedig a típusnak megfelelő műveletet.

Az $S \to A := E$ szabálynál az $A.y$ átkerül az $A.z$-be és az $E.z$-be. Az értékadás műveleténél tehát a kifejezés $E.y$ tényleges típusát $E.z$-re kell átalakítani.

Látható, hogy a nyelvtanban mindig az y és a z attribútumokban tárolt típusok között kell az ellenőrzést elvégezni.

Határozzuk meg a partíciókat.

$$\begin{aligned} IDS(S) &= \emptyset, \\ IDS(A) &= \{(A.x, A.y), (A.y, A.z)\}, \\ IDS(E) &= \{(E.x, E.y)\}, \\ IDS(B) &= \{(B.m, B.n)\}, \\ IDS(i) &= \emptyset, \end{aligned}$$

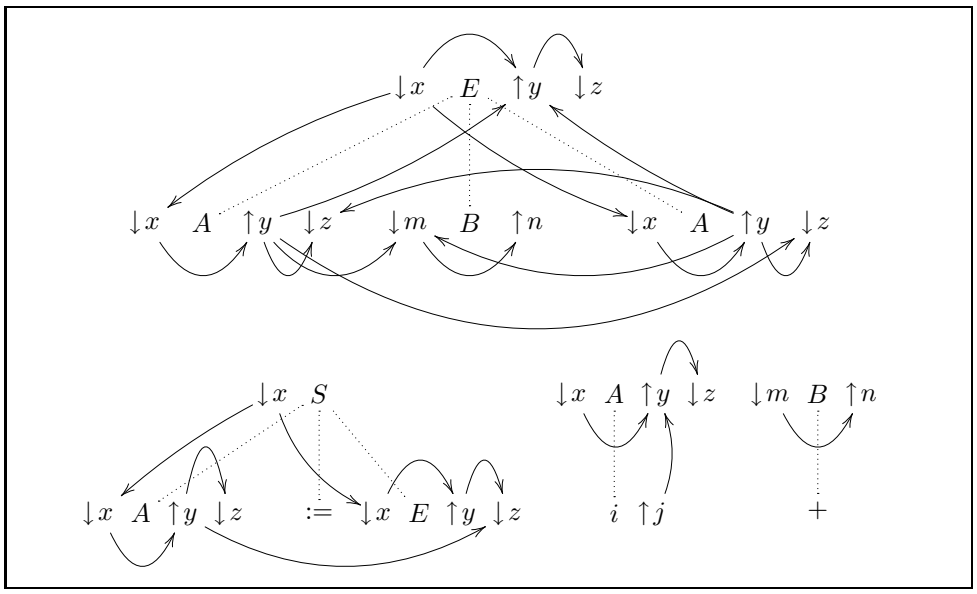
így egy lehetséges particionálás a következő:

- $\mathcal{A}_1(S) = \{S.x\}, \mathcal{A}_2(S) = \emptyset$,
- $\mathcal{A}_1(A) = \{A.x\}, \mathcal{A}_2(A) = \{A.y\}, \mathcal{A}_3(A) = \{A.z\}, \mathcal{A}_4(A) = \emptyset$,
- $\mathcal{A}_1(E) = \{E.x\}, \mathcal{A}_2(E) = \{E.y\}, \mathcal{A}_3(E) = \{E.z\}, \mathcal{A}_4(E) = \emptyset$,
- $\mathcal{A}_1(B) = \{B.m\}, \mathcal{A}_2(B) = \{B.n\}$,
- $\mathcal{A}_1(i) = \{i.j\}$.

A 7.8. ábrán a *DP* függőségeket ábrázoltuk. A 7.9. ábrán látható, hogy az *EDP* gráfok körmentesek, tehát a fenti particionálással valóban particionált attribútum fordítási nyelvtant kaptunk. Az ábrán minden attribútumfüggőséget vékony nyíllal jelöltünk. □

7.3.13. példa. Határozzuk meg a 7.3.12. példában szereplő *ATG* látogatási sorozatait, a fent megadott módszerrel írjuk meg az attribútumok értékeit meghatározó eljárásokat. A bejárási utak a 7.10. ábrán láthatók.

```
--
-- S →  A := E
P(1): begin
      nop                -- S↓x
      eval     A.x       -- A↓x
      call     P(3):1    -- A↑y
      eval     E.x       -- E↓x
      call     P(2):1    -- E↑y
      eval     A.z       -- A↓z
      call     P(3):2    -- A↑∅
      eval     E.z       -- E↓z
      call     P(2):2    -- E↑∅
      nop                -- S↑∅
```

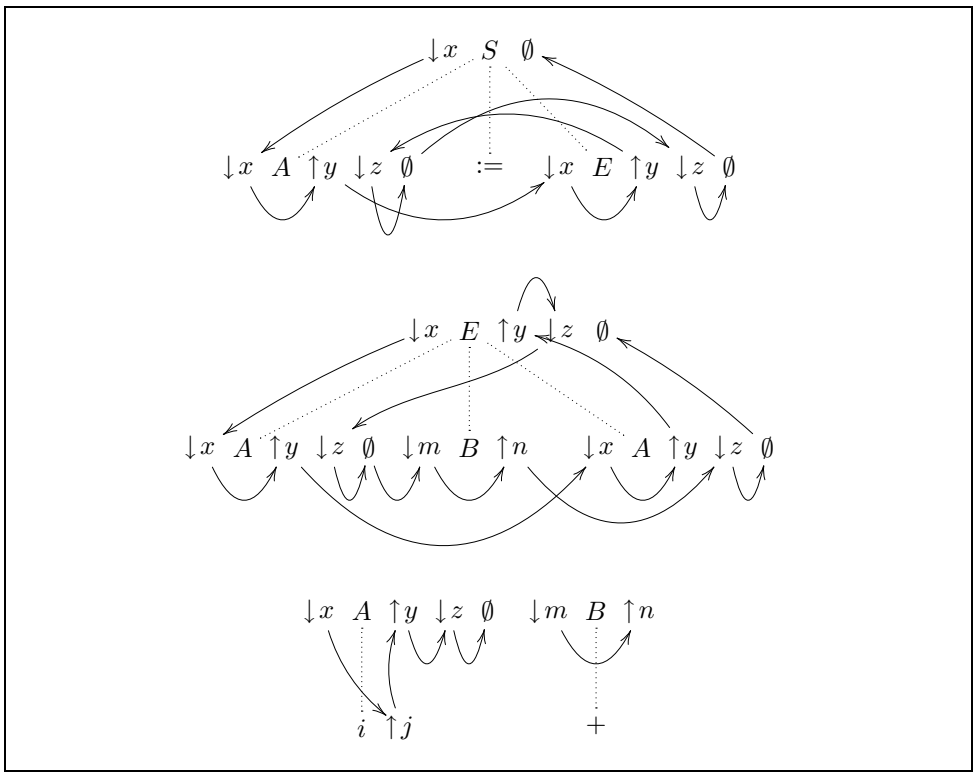


7.9. ábra. A 7.3.12. példa *EDP* függőségei

```
      call     C(1)
      end

-- E → A_1 B A_2
P(2): begin
1:    nop                 -- E↓x
      eval     A_1.x      -- A_1↓x
      call     P(3):1     -- A_1↑y
      eval     A_2.x      -- A_2↓x
      call     P(3):1     -- A_2↑y
      eval     E.y        -- E↑y
      return              -- E↓z
2:    eval     A_1.z      -- A_1↓z
      call     P(3):2     -- A_1↑∅
      eval     B.m        -- B↓m
      call     P(4):1     -- B↑n
      eval     A_2.z      -- A_2↓z
      call     P(3):2     -- A_2↑∅
      nop                 -- E↑∅
      call     C(2)
      return
      end

-- A → i
P(3): begin
1:    nop                 -- A↓x
```



7.10. ábra. A 7.3.12. példa *VP* látogatási sorozatai

```
      eval     i.j          -- i↑j
      eval     A.y          -- A↑y
      return                -- A↓z
2:    nop                   -- A↑∅
      call     C(3)
      return
      end

-- B → +
P(4): begin
1:    nop                   -- B↓m
      eval     B.n          -- B↑n
      call     C(4)
      return
      end
```

□

7.4. Rendezett attribútum fordítási nyelvtanok

Most egy módszert adunk a partíciók meghatározására, és egyben a rendezett attribútum fordítási nyelvtanokat is értelmezzük.

7.4.1. értelmezés. *Az alábbi algoritmussal az* $\mathcal{A}(X)$ *halmaz* $\mathcal{A}_1(X), \mathcal{A}_2(X), \ldots, \mathcal{A}_{m(X)}(X)$ *partícióit határozzuk meg. Egy ATG nyelvtan* ***rendezett attribútum fordítási nyelvtan****, ha minden* X *szimbólumra az* $\mathcal{A}(X)$ *így meghatározott partícióival particionált fordítási nyelvtant kapunk.*

- $T_{-1}(X) = \emptyset,$
  $T_0(X) = \emptyset,$
- *minden* $1 \leq k \leq m(X)$*-re, ahol* $m(X)$ *az a legkisebb olyan* i *index, amelyre* $T_{i-1}(X) \cup T_i(X) = \mathcal{A}(X)$*, legyen*

  $$T_k(X) = \{X.a \mid X.a \in \mathcal{S}(X),\ \textit{ha } k \textit{ páratlan},\ X.a \in \mathcal{I}(X),\ \textit{ha } k \textit{ páros},\ \textit{és } \nexists X.b,\ \textit{melyre } (X.a, X.b) \in \textit{IDS}(X),\ \textit{vagy } (X.a, X.b) \in \textit{IDS}(X) \textit{ esetén } X.b \in T_j(X)\ (j \leq k)\ \},$$

- *minden* X*-re, az* X *szimbólum partíciói legyenek*

  $\mathcal{A}_1(X), \mathcal{A}_2(X), \ldots, \mathcal{A}_{m(X)}(X)$*, ahol*

  $\mathcal{A}_i(X) = T_{m(X)-i+1}(X) \setminus T_{m(X)-i-1}(X) \quad (i = 1, 2, \ldots, m(X)).$

Az értelmezés szerint tehát először meghatározzuk a szimbólumok partícióit, és ha így particionált fordítási nyelvtant kapunk, azaz az *EDP* gráfok körmentesek, akkor a nyelvtant a 7.3.10. tétel alapján rendezett attribútum fordítási nyelvtannak nevezzük.

Az értelmezésből nyilvánvaló, hogy minden rendezett attribútum fordítási nyelvtan particionált, ez azonban fordítva nem áll fenn, mint azt a következő példából is láthatjuk.

7.4.2. példa. Határozzuk meg a 7.3.12. példában szereplő nyelvtan A szimbólumának partícióit.

$\textit{IDS}(A) = \{(A.x, A.y), (A.y, A.z)\},$

így

$$\begin{aligned} T_{-1}(A) &= \emptyset \\ T_0(A) &= \emptyset \\ T_1(A) &= \emptyset \\ T_2(A) &= \{A.z\} \\ T_3(A) &= \{A.y\} \\ T_4(A) &= \{A.x, A.z\} \end{aligned}$$

Mivel $T_3(A) \cup T_4(A) = \mathcal{A}(A)$, az $m(A) = 4$. A partíciók a következők:

$$\begin{array}{lllllllll}
\mathcal{A}_1(A) & = & T_{4-1+1}(A) \setminus T_{4-1-1}(A) & = & T_4(A) \setminus T_2(A) & = & \{A.x\} \\
\mathcal{A}_2(A) & = & T_{4-2+1}(A) \setminus T_{4-2-1}(A) & = & T_3(A) \setminus T_1(A) & = & \{A.y\} \\
\mathcal{A}_3(A) & = & T_{4-3+1}(A) \setminus T_{4-3-1}(A) & = & T_2(A) \setminus T_0(A) & = & \{A.z\} \\
\mathcal{A}_4(A) & = & T_{4-4+1}(A) \setminus T_{4-4-1}(A) & = & T_1(A) \setminus T_{-1}(A) & = & \emptyset
\end{array}$$ □

7.4.3. példa. A 7.3.11. példában szereplő nyelvtanról láttuk, hogy létezik olyan megengedett particionálása, amelyre az *EDP* gráfok körmentesek, tehát a nyelvtan particionált.

A fenti értelmezésben szereplő algoritmust alkalmazva azonban nem ilyen partíciókat kapunk, a következő halmazokat kapjuk eredményül:

$$\begin{array}{llll}
T_{-1}(A) = \emptyset & & T_{-1}(B) = \emptyset \\
T_0(A) = \emptyset & & T_0(B) = \emptyset \\
T_1(A) = \{A.x\} & & T_1(B) = \{B.z, B.v\} \\
T_2(A) = \{A.y\} & & T_1(B) = \{B.u, B.w\}
\end{array}$$

ezekből pedig az

- $\mathcal{A}_1(A) = \{A.y\},\ \mathcal{A}_2(A) = \{A.x\}$,
- $\mathcal{A}_1(B) = \{B.u, B.w\},\ \mathcal{A}_2(B) = \{B.v, B.z\}$

particionálást kapjuk, amelyről ugyanabban a példában láttuk, hogy nem eredményezett particionált nyelvtant. □

A továbbiakban megvizsgáljuk a rendezett attribútum fordítási nyelvtanok speciális eseteit.

7.4.1. *S*-attribútum fordítási nyelvtanok

Szemantikai elemzés folyamán előfordulhat, hogy a szemantikai információ a szintaxisfában csak alulról felfelé terjed, azaz az egyszintű részfákat tekintve csak a levelekről kerül információ a gyökérelemhez, és fordított irányú információáramlás nincs.

7.4.4. értelmezés. *Egy ATG-t* ***S-attribútum fordítási nyelvtannak*** *nevezünk és S-ATG-vel jelölünk, ha minden $A \to X_1 X_2 \ldots X_n$ helyettesítési szabályhoz csak*

$$A{\uparrow}a = f(X_1.b, \ldots, X_n.c)$$

alakú szemantikai függvény tartozik, és akciószimbólum csak a helyettesítési szabály jobb oldalának első szimbóluma lehet.

Az *S-ATG* nyelvtanok jól használhatók alulról-felfelé elemzéseknél. Az elemző program a szimbólumhoz tartozó attribútumokat az elemző vermében, a szimbólumhoz kapcsolva tárolhatja. Ha az elemző egy $A \to \alpha$ szabály szerinti redukciót hajt végre, akkor az A-hoz tartozó szintetizált attribútum értékét a verem tetején levő, az α-hoz tartozó attribútum értékekből meghatározhatja.

Egy $LR(1)$ elemző műveleteit csak az attribútum értékeknek a verembe történő beírásával, a veremből való kiolvasásával vagy törlésével kell bővíteni.

Az *S-ATG* nyelvtanok az attribútum fordítási nyelvtanoknak csak nagyon szűk osztályát alkotják, hiszen örökölt attribútumokat nem tartalmazhatnak.

7.4.2. *L*-attribútum fordítási nyelvtanok

Szemantikai elemzésnél az lenne a célszerű, ha az attribútum értékek kiszámítása a szintaxisfa építésével párhuzamosan történne, azaz ha egy új szimbólum kerül a szintaxisfába, akkor ennek a szimbólumnak az attribútum értékei azonnal meghatározhatók lennének. Egy ilyen tulajdonságú nyelvtannal foglalkozunk a következőkben.

7.4.5. értelmezés. *Egy ATG-t* ***L-attribútum fordítási nyelvtannak*** *nevezünk és L-ATG-vel jelölünk, ha minden $A \to X_1X_2 \dots X_n$ helyettesítési szabályra az attribútumok kiszámítási sorrendje a következő:*

$\mathcal{I}(A), \mathcal{I}(X_1), \mathcal{S}(X_1), \mathcal{I}(X_2), \mathcal{S}(X_2), \dots, \mathcal{I}(X_n), \mathcal{S}(X_n), \mathcal{S}(A)$.

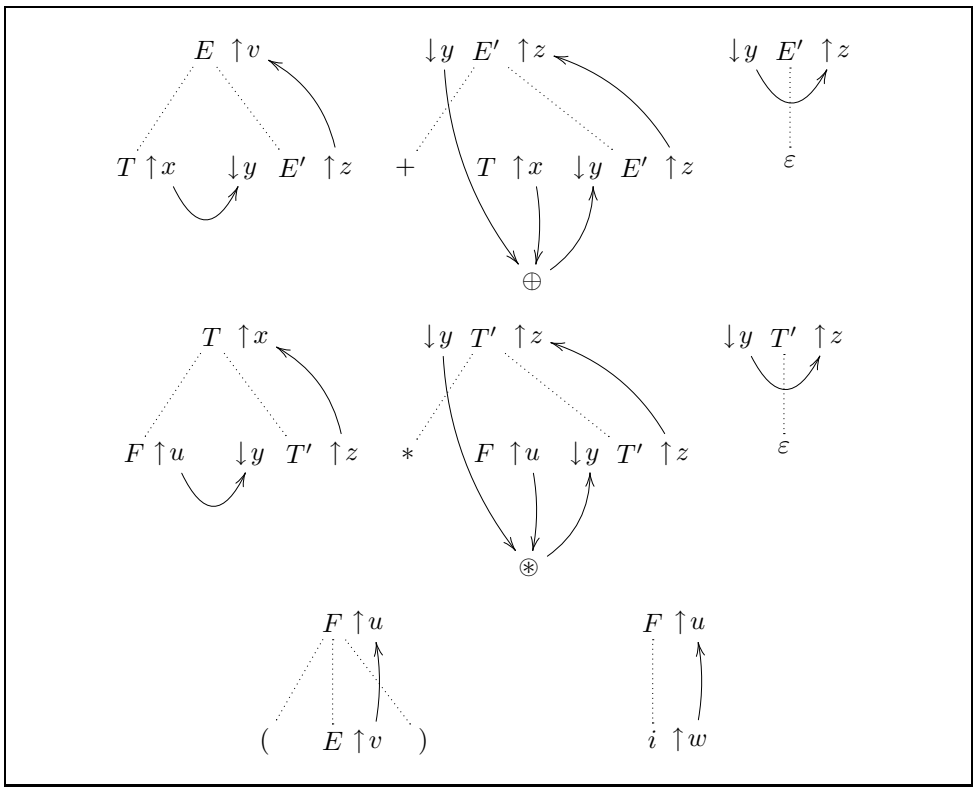
Egy *L-ATG* nyelvtanban tehát az $A \to X_1X_2 \dots X_n$ helyettesítési szabály X_i $(1 \leq i \leq n)$ szimbólumának örökölt attribútumai csak az A örökölt és az $X_1, X_2, \dots X_{i-1}$ szimbólumok örökölt vagy szintetizált attribútumaitól függnek, és az X_i szimbólum szintetizált attribútuma ezenkívül még a saját örökölt attribútumának lehet függvénye. Az A szimbólum szintetizált attribútuma a saját örökölt attribútumaitól, és a helyettesítési szabály jobb oldalán levő szimbólumok örökölt vagy szintetizált attribútumaitól függ.

Látható tehát, hogy az *L-ATG* nyelvtan egy $A \to X_1X_2 \dots X_n$ helyettesítési szabályának gráfját nézve a szemantikai információ az A örökölt attribútumától kiindulva lefelé, majd a leveleken balról jobb halad, és az utolsó levél után felfelé, a csúcsban levő A szintetizált attribútuma felé terjed.

A következő tétel állítása a 7.4.5. értelmezésből következik.

7.4.6. tétel. *Egy ATG akkor és csak akkor L-ATG, ha lokálisan aciklikus, és minden $p : A \to X_1X_2 \dots X_n$ helyettesítési szabály minden $(X_i.a, X_j.b) \in DP(p)$ direkt függőségi relációjára a következő esetek egyike fennáll:*

- $i < j$,
- $i = j$ *és* $X_i.a \in \mathcal{I}(X_i)$,
- $X_i = A$ *és* $X_i.a \in \mathcal{I}(A)$,
- $X_j = A$.



7.11. ábra. A 7.4.7. példa *DP* függőségei

7.4.7. példa. A 6.3.11. példában szereplő nyelvtant egészítsük ki attribútumokkal úgy, hogy egy elemzés végén az $E \uparrow v$ attribútum az aritmetikai kifejezés értékét tartalmazza. Egy attribútumnevet több különböző szimbólumhoz is hozzárendelünk, és feltételezzük, hogy a $+, *, (,)$ terminális szimbólumokhoz nem tartozik attribútum. Az i terminális szimbólum egy természetes számot jelöl, a hozzátartozó attribútum értéke legyen ez a szám.

$$
\begin{array}{lcll}
E\uparrow v & \to & T\uparrow xE'\downarrow y\uparrow z & \{E'\downarrow y \leftarrow T\uparrow x,\ E\uparrow v \leftarrow E'\uparrow z\} \\
E'_1\downarrow y\uparrow z & \to & +T\uparrow xE'_2\downarrow y\uparrow z & \{E'_2\downarrow y \leftarrow E'_1\downarrow y + T\uparrow x,\ E'_1\uparrow z \leftarrow E'_2\uparrow z\} \\
 & | & \varepsilon & \{E'_1\uparrow z \leftarrow E'_1\downarrow y\} \\
T\uparrow x & \to & F\uparrow uT'\downarrow y\uparrow z & \{T'\downarrow y \leftarrow F\uparrow u,\ T\uparrow x \leftarrow T'\uparrow z\} \\
T'_1\downarrow y\uparrow z & \to & *F\uparrow uT'_2\downarrow y\uparrow z & \{T'_2\downarrow y \leftarrow T'_1\downarrow y * F\uparrow u,\ T'_1\uparrow z \leftarrow T'_2\downarrow z\} \\
 & | & \varepsilon & \{T'_1\uparrow z \leftarrow T'_1\downarrow y\} \\
F\uparrow u & \to & (E\uparrow v) & \{F\uparrow u \leftarrow E\uparrow v\} \\
 & | & i\uparrow w & \{F\uparrow u \leftarrow i\uparrow w\}
\end{array}
$$

A nyelvtan *DP* függőségeit a 7.11. ábrán ábrázoltuk. Látható, hogy a nyelvtan egy *L-ATG* nyelvtan. □

A szemantikai függvények tetszőleges aritmetikai és logikai kifejezéseket tartalmazhatnak. Az attribútumokra vonatkozó kifejezések értékének meghatározása új szemantikai rutinok és új attribútumok bevezetésével áttehető az elemző programba, és így elérhető, hogy a szemantikai függvények csak értékadó utasításokat tartalmazzanak.

7.4.8. értelmezés. *Egy L-ATG nyelvtan* ***egyszerű értékadó formában*** *van, ha minden* $A \to X_1X_2 \dots X_n$ *helyettesítési szabályra*

- *az* $X_i{\downarrow}\, a$ $(1 \le i \le n)$ *attribútum értéke konstans, vagy megegyezik az* $\mathcal{I}(A)$*, vagy az* $\mathcal{S}(X_j)$ $(1 \le j < i)$ *egy értékével,*
- *az* $A{\uparrow}\, b$ *attribútum értéke konstans, vagy megegyezik az* $\mathcal{I}(A)$*, vagy az* $\mathcal{S}(X_i)$ $(1 \le i \le n)$ *egy értékével.*

7.4.9. példa. A 7.4.7. példában szereplő második és ötödik helyettesítési szabály szemantikai függvényei nincsenek egyszerű értékadó formában. Ezek a következőképpen alakíthatók át:

$$
\begin{array}{ll}
E'_1{\downarrow}y{\uparrow}z \;\to\; + \; T{\uparrow}x \; @add\,({\downarrow}a, b, {\uparrow}c) \; E'_2{\downarrow}y{\uparrow}z & \{\, @add{\downarrow}b \leftarrow E'_1{\downarrow}y, \\
 & \quad @add{\downarrow}a \leftarrow T{\uparrow}x, \\
 & \quad E'_2{\downarrow}y \leftarrow @add{\uparrow}c, \\
 & \quad E'_1{\uparrow}z \leftarrow E'_2{\uparrow}z\} \\
T'_1{\downarrow}y{\uparrow}z \;\to\; *F{\uparrow}u @mul\,({\downarrow}a, b, {\uparrow}c) T'_2{\downarrow}y{\uparrow}z & \{\, @mul{\downarrow}a \leftarrow T'_1{\downarrow}y, \\
 & \quad @mul{\downarrow}b \leftarrow F{\uparrow}u, \\
 & \quad T'_2{\downarrow}y \leftarrow @mul{\uparrow}c, \\
 & \quad T'_1{\uparrow}z \leftarrow T'_2{\downarrow}z\}
\end{array}
$$

□

Az *L-ATG* nyelvtanok leírása tovább egyszerűsíthető. Az értékadó utasítások is elhagyhatók, ha bevezetjük az ***attribútumváltozó*** fogalmát. Az attribútumok értékét az attribútumváltozók veszik fel, és így a különböző X és Y szimbólumok azonos nevű attribútumának az értéke bizonyos esetekben a rájuk vonatkozó értékadó utasítás nélkül is azonos lesz.

Az attribútumváltozó fogalma hasonló a programnyelvek változó fogalmához. Egy attribútumváltozó lokális egy helyettesítési szabályra, azaz hatásköre egy helyettesítési szabály, és ha az attribútumváltozó értéket kap, akkor az attribútumváltozó láthatósága határozza meg, hogy ez az érték az azonos nevű változókban a helyettesítési szabály melyik szimbóluma mellett jelenik meg.

Egy egyszerű értékadó formában levő és attribútum változókat tartalmazó *L-ATG* értékadó utasításai el is hagyhatók, ha az $X_0 \to X_1X_2 \dots X_n$ $(X_0 \in N)$ helyettesítési szabályba beírt attribútumok értékének meghatározásakor a következő szabályokat betartjuk:

1. az $X_i{\downarrow}a$ $(1 \leq i \leq n)$ értéke az X_j $(0 \leq j < i)$ szimbólumokhoz tartozó legjobboldalibb azonos nevű örökölt vagy szintetizált attribútum értéke lesz,
2. az $X_0{\downarrow}\,a$ az elemzés egy másik lépésében kap értéket, akkor, amikor a szintaxisfa egy egyszintű részfájának X_0 leveléhez tartozó örökölt attribútumokat határozzuk meg,
3. ha X_i egy akciószimbólum, akkor az $X_i{\uparrow}\,a$ $(1 \leq i \leq n)$ értékét szemantikai rutin határozza meg, egyébként az $X_i{\uparrow}\,a$ $(1 \leq i \leq n)$ az elemzés egy későbbi lépésében kap értéket, akkor, amikor a szintaxisfa X_i gyökérelemű egyszintű részfájának szintetizált attribútumértékét határozzuk meg,
4. az $X_0{\uparrow}\,a$ értéke az X_j $(1 \leq j \leq n)$ szimbólumokhoz tartozó legjobboldalibb azonos nevű attribútum értéke lesz.

Különböző attribútumváltozók közötti adatátvitelre vezessük be a $\uparrow b \leftarrow{\downarrow}a$ átvitelt végrehajtó *@echo* $({\downarrow}a,{\uparrow}b)$ szemantikai rutint.

7.4.10. példa. A 7.4.7. és a 7.4.9. példában szereplő nyelvtan attribútumváltozókat tartalmazó formája a 7.12. ábrán látható, a balról-jobbra terjedő információt a sor alatt, a jobbról-balra terjedő információt a sor fölött jelöltük. □

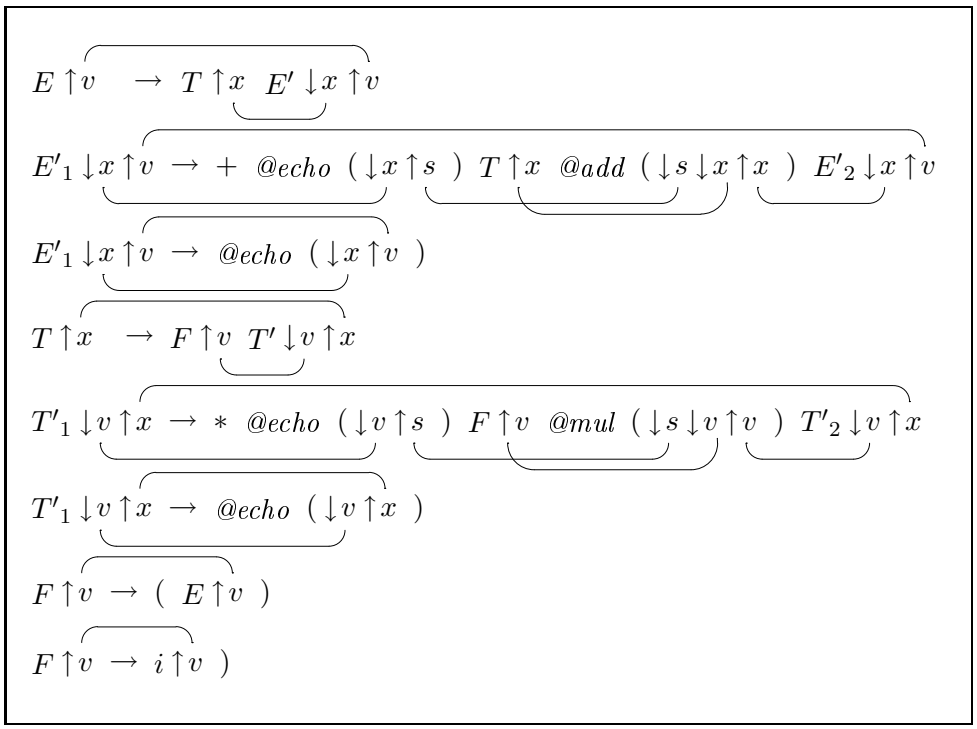
Az *L-ATG* nyelvtanok jól használhatók felülről-lefelé elemzésekben, például az $LL(1)$ elemzőkben. Jó attribútumkiértékelő tulajdonságaikat felhasználva célszerű lenne az alulról-felfelé elemzésekben is *L-ATG* nyelvtanokat használni, de ezekben az elemzésekben az örökölt attribútumok kiértékelése problémát okozhat.

Ha az $A \rightarrow X_1 \ldots X_n$ szabályban az X_i $(1 \leq i \leq n)$ szimbólumnak van $X_i{\downarrow}\,a$ attribútuma, akkor a szabályban az X_i-t cseréljük ki a B_iX_i szimbólumpárra, és vezessük be az új $B_i \rightarrow \varepsilon$ szabályt. A B_i vegye át X_i-től az örökölt attribútum „tárolását" addig, amíg az X_i-hez tartozó részfa attribútumai kiértékelődnek, azaz ha $X_i{\downarrow}\,a = f(\ldots)$, akkor legyen $B_i{\downarrow}\,a = f(\ldots)$, és legyen $X_i{\downarrow}\,a \leftarrow B_i{\downarrow}\,a$.

Ha $X_i{\downarrow}\,a \leftarrow A{\downarrow}\,b$, vagy $X_i{\downarrow}\,a \leftarrow X_j.b$ $(j < i)$, akkor a B_i bevezetésére nincs szükség, hiszen az $X_i{\downarrow}\,a$-t meghatározó attribútumértékek X_i feldolgozásakor már biztosan ismertek.

Ez a módszer alkalmazható az $A{\downarrow}\,a$ meghatározására is, mivel $A = S$ esetén az A-nak nem lehet örökölt attribútuma, $A \neq S$ esetén pedig az A biztosan előfordul egy helyettesítési szabály jobb oldalán.

Sajnos, alulról-felfelé elemezve nem minden $LR(1)$ nyelvtanban lehet az örökölt attribútumok értékét ezzel a módszerrel meghatározni, még akkor sem, ha a nyelvtan *L-ATG* nyelvtan. A következő példa ezt mutatja meg.



7.12. ábra. A 7.4.7. példa szabályai

7.4.11. példa. A nyelvtan helyettesítési szabályai és szemantikai függvényei a következők:

S	$\rightarrow$	$L\downarrow a$	{	$L\downarrow a$	$\leftarrow$	0	}
$L_1\downarrow a$	$\rightarrow$	$L_2\downarrow a\ x$	{	$L_2\downarrow a$	$\leftarrow$	$(L_1\downarrow a)+1$	}
	$\mid$	ε	{	*print(a)*			}

Alulról-felfelé elemezve a legelső művelet az $L \rightarrow \varepsilon$ szabály szerinti redukció lesz, az ehhez tartozó szemantikai függvény azonban semmilyen módon sem kaphatja meg az a attribútum értékét, azaz a mondatban levő x karakterek darabszámát. □

Gyakorlatok

7-1. Határozzuk meg a 7.3.12. példában szereplő nyelvtan *DP*, *NDP*, *IDP* és *IDS* függőségeit.

7-2. Mutassuk meg, hogy a 7.4.7. példában szereplő nyelvtan minden szabálya kielégíti a 7.4.6. tétel következtetéseit.

8. FEJEZET

A kódgenerálás

Ebben a fejezetben azt vizsgáljuk, hogy a fordítóprogramok milyen módszerek felhasználásával építik fel a program kódját. A már megismert *L-ATG* nyelvtanok felhasználhatóságát mutatja, hogy kódgenerálás feladata is leírható *L-ATG* nyelvtannal.

Mivel a könyvünk II. részének célja a fordítási folyamat és a formális nyelvek kapcsolatának bemutatása, a kódgenerálás feladatai közül csak néhányat mutatunk be. A kódoptimalizálás vizsgálatát a feladatok között tűzzük ki.

8.1. Az aktivációs rekord

Futási időben a program i-edik blokkjának adatait az AR_i *aktivációs rekord* tartalmazza. Az éppen futó blokk aktivációs rekordját *aktív aktivációs rekordnak* nevezzük. A fordítóprogram olyan kódot generál, hogy az aktivációs rekord a program futtatásakor, a blokk hívásakor épüljön fel a számítógép run-time vermébe.

Az aktivációs rekord három részből áll, a *lokális változók területéből*, a *display-területből* és a *paraméterterületből*.

A display-terület egyik adata a *call* utasítással indított blokkból a hívó programra való visszatéréshez szükséges utasításcím, az a cím, amit a *call* utasítás végrehajtásakor a processzor ír a verembe. A *statikus pointer* a blokkból elérhető változókat tartalmazó aktivációs rekordra mutat. A display-területen található meg a hívó blokk aktivációs rekordjának címe is. A hívó blokk aktivációs rekordjának címére azért van szükség, hogy a blokkból való kilépéskor visszaállítható legyen a run-time veremnek a blokkba való belépéskor fennálló állapota. Ezt az adatot *dinamikus pointernek* nevezzük.

Az aktivációs rekord paraméterterülete a blokk aktuális paramétereit tartalmazza. Érték szerinti paraméterátadásnál a paraméter értéke, hivatkozás szerinti paraméterátadásnál a paraméter memóriacíme található ezen a területen.

Az IBM PC gépeken a verem tetejét az *SP* regiszter jelzi. Az aktív aktivációs rekord dinamikus pointerére a *BP* regiszter mutat, a $[BP]+2$ címen a visszatérési cím, a $[BP]+4$ címen a statikus pointer található.

A fordítóprogramok a főprogram lokális változóit, azaz a globális változókat az adatszegmensbe helyezik, így a főprogram aktivációs rekordja csak a dinamikus pointert és a főprogramból a rendszerhez való visszatéréshez egy visszatérési címet tartalmaz (8.1.(a) ábra). A főprogram *p* nevű kétbájtos lokális változójának deklarációjából a

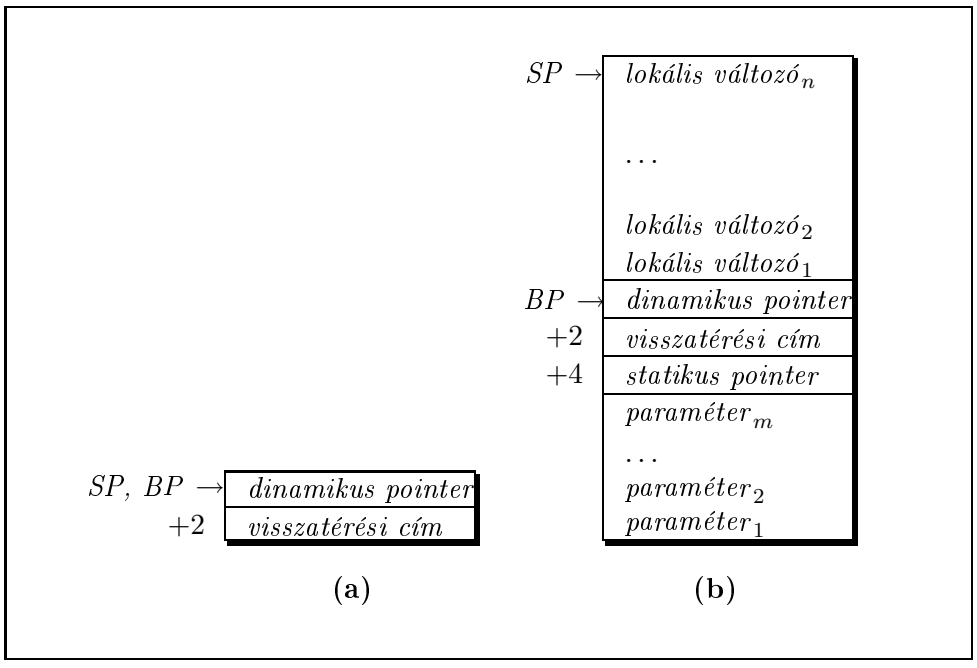
```
p       dw  ?
```

utasítássor származik.

Az alprogramok lokális változói az aktivációs rekordba kerülnek, azaz majd a futási időben a run-time verembe. Egy n paramétert és m lokális változót tartalmazó eljáráshoz a 8.1.(b) ábrán látható aktivációs rekord tartozik. A fordítóprogram egy belső blokk *q* nevű kétbájtos lokális változójának deklarációjából a

```
p       equ word ptr [bp]-k
```

programsort készíti, ahol $[bp]-k$ a változó helyét határozza meg.



8.1. ábra. Az aktivációs rekordok szerkezete: **(a)** főprogram, **(b)** alprogram

8.2. A kifejezések fordítása

A kifejezések fordítását egy egyszerű, a 6.3.11. példában szereplő nyelvtannal generált kifejezéseken tanulmányozzuk.

Ha az $F \to i$ helyettesítési szabályban szereplő i egy egyszerű változó, akkor legyen

$F \to i\uparrow n\ @SearchVar(\downarrow n,\uparrow t,q)\ @StLoadAX(\downarrow q)$,

ahol az n attribútum a változó azonosítója, t a típusa, és q a deklarációban meghatározott címe. A $@SearchVar(\downarrow n,\uparrow t,q)$ megkeresi az n szimbólumot a szimbólumtáblában. Ha van ilyen nevű szimbólum, akkor ellenőrzi a láthatóságát, és outputként adja a szimbólum típusát és a szimbólumtáblában levő címét. A $@StLoadAX(\downarrow q)$ szemantikus rutin a változó értékét az AX regiszterbe töltő utasítást generálja:

```
mov     ax,q        ; @StLoadAX(↓q)
```

Így a p nevű globális változóra a

```
mov     ax,p        ; @StLoadAX(↓q)
```

a lokális, azaz a veremben elhelyezett aktivációs rekord változójára a

```
mov     ax,word ptr [bp]-k   ; @StLoadAX(↓q)
```

utasítás fog generálódni.

A 7.4.10. példában az s attribútumot használtuk közbülső eredmények tárolására. Mivel egy művelet eredménye mindig az AX regiszterben van, most ezt a funkciót a *push AX* utasítással tudjuk megvalósítani. Generálja a *@StPushAX* szemantikus rutin ezt az utasítást. A $@add(\downarrow s,x,\uparrow x)$ és a $@mul(\downarrow s,v,\uparrow v)$ szemantikus rutinok végezték a műveletek végrehajtását, ezt a *pop BX*; *add AX,BX* és *pop BX*; *imul BX* utasításpárokkal tudjuk leírni. Generálják a *@StPopBX*, *@StAddBX* és *@StImulBX* szemantikus rutinok ezeket az utasításokat.

A kétbájtos *integer* kifejezést leíró nyelvtant tehát a következőképpen adhatjuk meg:

$$
\begin{array}{lcl}
E & \to & TE' \\
E' & \to & +\ @StPushAX\ T\ @StPopBX\ @StAddBX\ E' \ \mid \varepsilon \\
T & \to & FT' \\
T' & \to & *\ @StPushAX\ F\ @StPopBX\ @StImulBX\ T' \ \mid \varepsilon \\
F & \to & (\ E\ ) \\
 & \mid & i\uparrow n\ @SearchVar(\downarrow n,\uparrow t,q)\ @StLoadAX(\downarrow q).
\end{array}
$$

8.2.1. példa. Határozzuk meg a $2+j*(3+k)$ kifejezéshez a fenti nyelvtannal generálható programot. Legyen j egy globális szimbólum, k pedig az aktivációs rekord első lokális változója.

```
        mov     ax,2        ; 2 @StLoadAX
        push    ax          ; + @StPushAX
        mov     ax,j        ; j @StLoadAX
        push    ax          ; * @StPushAX (
        mov     ax,3        ; 3 @StLoadAX
        push    ax          ; + @StPushAX
        mov     ax,word ptr [bp]-2 ; k @StLoadAX
        pop     bx          ; @StPopBX
        add     ax,bx       ; @StAddBX )
        pop     bx          ; @StPopBX
        imul    bx          ; @StImulBX
        pop     bx          ; @StPopBX
        add     ax,bx       ; @StAddBX
```

□

8.3. Az *if* utasítás fordítása

Vizsgáljuk az *if* utasítás következő formáját:

⟨*if-utasítás*⟩ → *if* ⟨*kifejezés*⟩ *then* ⟨*utasítás*$_1$⟩ ⟨*if-tail*⟩
⟨*if-tail*⟩ → *else* ⟨*utasítás*$_2$⟩ *endif*
| *endif*

Az utasításból a következő felépítésű kódok egyikét kell generálnunk:

```
        <kifejezés>
        cmp     ax,0
        jz      L_0001
        <utasítás1>
L_0001:
```

vagy

```
        <kifejezés>
        cmp     ax,0
        jz      L_0001
        <utasítás1>
        jmp     L_0002
L_0001:
        <utasítás2>
L_0002:
```

Látható, hogy az *if* utasításhoz egyedi címkét kell generálni. A címke nevének generálását végezze a *@GenLabel*(↑r) szemantikus rutin.

A *@GenLabel*(↑r) feladata hasonló az assemblerek makrófordításkor működő címkegenerálásához, azaz amikor a *label* direktívával megadott címkékre

a makróhelyettesítésekben az assembler a *??nnnn* címkéket állítja elő, ahol *nnnn* a címkéket megkülönböztető, minden helyettesítésben eggyel megnövelt természetes szám. Az *nnnn* érték kerül az r attribútumba, és a generált címke *L_ nnnn* alakú.

A *@GenLabel*$(\uparrow r)$ szemantikus rutinnal meghatározott r címkenévvel a *@StLabel*$(\downarrow r)$ szemantikus rutin az

```
L_r     equ     $           @StLabel(↓r)
```

vagy az ezzel ekvivalens

```
L_r:                        @StLabel(↓r)
```

sort állítja elő. A *@StJmp*$(\downarrow r)$ rutin generálja a következő feltétel nélküli ugró utasítást:

```
        jmp     L_r         ; @StJmp(↓r)
```

A ⟨*kifejezés*⟩ feldolgozása az AX regiszterben nullát ad, ha a kifejezés értéke *false*. Ezért generálja a *@StJFalse*$(\downarrow r)$ szemantikus rutin a következő utasításokat:

```
        cmp     ax,0        ; @StJFalse(↓r)
        jnz     $+5
        jmp     L_r
```

A *jnz* és *jmp* utasításpárra azért van szükség, mert az assembly nyelvben nincs *near* típusú feltételes ugró utasítás. Ha az ugrás távolsága megfelel a *short* típusú ugrásnak, akkor a *@StJFalse*$(\downarrow r)$ rutinnal generált utasítások egyszerűbben is megadhatók:

```
        cmp     ax,0        ; @StJFalse(↓r)
        jz      L_r
```

Így a kódgeneráláshoz az *if* utasítás fenti leírása a következőképpen alakítható át:

⟨*if-utasítás*⟩ → *if* ⟨*kifejezés*⟩ *@GenLabel*$(\uparrow r)$ *@StJFalse*$(\downarrow r)$
then ⟨*utasítás*$_1$⟩ ⟨*if-tail*⟩$\downarrow r$

⟨*if-tail*⟩$\downarrow r$ → *else @GenLabel*$(\uparrow s)$ *@StJmp*$(\downarrow s)$ *@StLabel*$(\downarrow r)$
⟨*utasítás*$_2$⟩ *endif @StLabel*$(\downarrow s)$
| *endif @StLabel*$(\downarrow r)$

8.3.1. példa. Az *if a then i := 1 else i := 2 endif* forrásnyelvű programsorból a következő assembly nyelvű programot kapjuk:

```
                    ; if
mov     ax,a        ; <kifejezés>
                    ; @GenLabel(↑r)   (r = 0001)
cmp     ax,0        ; @StJFalse(↓r)
jz      L_0001
                    ; then
```

```
        mov    i,1        ; <utasítás1>
                          ; else @GenLabel(↑s)  (s = 0002)
        jmp    L_0002     ; @StJmp(↓s)
L_0001:                   ; @StLabel(↓r)
        mov    i,2        ; <utasítás2>
                          ; endif
L_0002:                   ; @StLabel(↓s)
```

□

Gyakorlatok

8-1. Adjuk meg egy adott programnyelven írt program deklarációiból származó assembly kódot. Írjuk le a kódgenerálást ATG nyelvtannal.

8-2. Adjuk meg a logikai kifejezések kódgenerálásakor keletkező assembly kódot. Írjuk le a kódgenerálást ATG nyelvtannal.

8-3. Határozzuk meg az érték és a hivatkozás szerinti paraméterátadás fordításakor generált kódot. Írjuk le a kódgenerálást ATG nyelvtannal.

Feladatok

II-1. Programszöveg lexikális elemzése
Az 5. fejezetben szereplő LEX-ELEMEZ algoritmus csak *egy* reguláris kifejezéssel, vagy a megfelelő véges determinisztikus automatával leírható szövegek elemzését adta meg, azaz csak egy szimbólumot ismert fel. Készítsünk egy olyan automatát, ami egy teljes programnyelv lexikális elemzését elvégzi, és adjuk meg a teljes elemzés LEX-ELEMEZ-NYELV algoritmusát. Az algoritmus egyik bemenete legyen egy programszöveg, kimenete pedig a szimbólumsorozat. Nyilvánvaló, hogy ha az automata egy végállapotba került, azaz felismert egy szimbólumot, akkor ezután a működését a kezdőállapottal kell folytatnia, hogy a programszöveg következő szimbólumát meghatározza. Az algoritmus működésének csak akkor kell befejeződnie, ha az elemzés a programszöveg végére ért, vagy ha egy lexikális hibát talált.

II-2. Szimbólumsorozat a szimbólumok adataival
Módosítsuk az előző feladat algoritmusát úgy, hogy a szimbólumsorozat tartalmazza a felismert szimbólum jellemző adatait is, például egy azonosító szimbólum mellé adjuk meg a szimbólumot alkotó karaktersorozatot, vagy egy szám mellé a szám típusát és értékét is. A szimbólumok kódja mellé, az egységes kezelés érdekében, célszerű nem az adatokat, hanem az adatokra mutató pointereket írni.

II-3. *LALR*(1) elemző *LR*(0)-kanonikus halmazokból

Ha az $LR(1)$-elemekből elhagyjuk az előreolvasási szimbólumokat, akkor az ***LR*(0)-elemeket** kapjuk meg. Az $LR(0)$-elemekre is értelmezhetjük a *closure* és *read* függvényeket, úgy, hogy az előreolvasási szimbólumokat nem vesszük figyelembe. Az $LR(1)$ kanonikus halmazokhoz hasonló módszerrel meghatározhatjuk az

$$\mathcal{I}_0, \mathcal{I}_1, \ldots, \mathcal{I}_n$$

***LR*(0) kanonikus halmazokat** is. Megfigyelhetjük, hogy $LALR(1)$ nyelvtanok esetén az egyesítés után kapott $LALR(1)$ kanonikus halmazok darabszáma megegyezik az $LR(0)$-kanonikus halmazok darabszámával, hiszen az egyesített halmazokban az $LR(1)$-elemek magjai éppen az $LR(0)$-kanonikus halmazok elemeinek felelnek meg. Így az $LALR(1)$ elemzőnek pontosan ugyanannyi állapota lesz, mint egy $LR(0)$ elemzőnek lenne.

Ez a tulajdonság adja az $LALR(1)$-kanonikus halmazoknak az $LR(0)$-kanonikus halmazokból történő meghatározását, hiszen ha az $LR(0)$-kanonikus halmazok elemeit kiegészítjük a megfelelő előreolvasási szimbólumokkal, akkor az egyesített $LALR(1)$-kanonikus halmazokat fogjuk megkapni.

Vegyük észre, hogy a nem $\mathcal{H}_0$ $LR(1)$-kanonikus halmazokban csak az olyan $LR(1)$-elemekben kezdődik a mag jobb oldala ponttal, amelyek a kanonikus halmazban levő $LR(1)$-elemekből a *closure* függvény alkalmazásával származnak. Így az $LR(1)$-elemek egy kanonikus halmazát nem kell az összes elemének felsorolásával megadni. Nevezzük a $\mathcal{H}_0$ kanonikus halmaz ***törzsének*** az $[S' \to .S, \#]$ $LR(1)$-elemet, egy nem $\mathcal{H}_0$ kanonikus halmaz *törzsének* pedig a kanonikus halmaz azon elemeit, amelyekben a mag jobb oldala nem a pont metaszimbólummal kezdődik. Egy $LR(1)$-kanonikus halmazt tehát a törzsével is megadhatunk, hiszen a törzsből az összes többi $LR(1)$-elem előállítható.

Könnyen belátható, hogy a kanonikus halmaz törzséből a léptetés és a redukció műveletek is meghatározhatók.

Ha az $LR(0)$-kanonikus halmazok törzseinek elemeit kiegészítjük az előreolvasási szimbólumokkal, akkor az egyesített $LR(1)$-kanonikus halmazok törzseit fogjuk megkapni, azaz ha $\mathcal{I}_j$ az $LR(0)$-elemek kanonikus halmazának a törzse, $\mathcal{I}_j$-ből a kiegészítéssel létrehozott egyesített $LR(1)$-kanonikus halmaz törzse $\mathcal{K}_j$ lesz.

Az $LR(0)$-elemekre, ha ismerjük $\mathcal{I}_j$-t, akkor a $read(\mathcal{I}_j, X)$ könnyen meghatározható, hiszen ha $[B \to \gamma.C\delta] \in \mathcal{I}_j$, $C \stackrel{*}{\to} A\eta$ és $A \to X\alpha$, akkor nyilván $[A \to X.\alpha] \in read(\mathcal{I}_j, X)$. Az $LR(1)$-elemekre ez már nem ilyen egyszerű, ha $[B \to \gamma.C\delta, b] \in \mathcal{K}_j$, $C \stackrel{*}{\to} A\eta$ és $A \to X\alpha$, akkor még az előreolvasási szimbólumot is meg kell határozni, azaz azt, hogy milyen a-ra lesz $[A \to X.\alpha, a] \in read(\mathcal{K}_j, X)$.

Ha $\eta\delta \neq \varepsilon$, és $a \in \mathit{Első}(\eta\delta b)$, akkor biztosan $[A \to X.\alpha, a] \in \mathit{read}(\mathcal{K}_j, X)$, és azt mondjuk, hogy az a előreolvasási szimbólum a $\mathit{read}(\mathcal{K}_j, X)$ halmaznak ehhez az eleméhez ***spontán generálható,*** hiszen a b szimbólumnak semmilyen szerepe sincs az új előreolvasási szimbólum meghatározásában.

Ha $\eta\delta = \varepsilon$, akkor $[A \to X.\alpha, b]$ lesz a $\mathit{read}(\mathcal{K}_j, X)$ eleme, azaz ebben az elemben a b lesz az előreolvasási szimbólum. Ekkor azt mondjuk, hogy a b előreolvasási szimbólum a $\mathcal{K}_j$-ből ***öröklődik*** a $\mathit{read}(\mathcal{K}_j, X)$ halmaznak ebbe az elemébe.

Ha adott egy $LR(0)$-kanonikus halmaznak az $\mathcal{I}_j$ törzse, akkor tetszőleges X szimbólumra a $\mathit{read}(\mathcal{K}_j, X)$-hez spontán generálható és öröklődő előreolvasási szimbólumok a következő módszerrel határozhatók meg: minden $[B \to \gamma.\delta] \in \mathcal{I}_j$-re határozzuk meg a $\mathcal{K}_j = \mathit{closure}([B \to \gamma.\delta, @])$ halmazt, ahol @ egy dummy-szimbólum,

- ha $[A \to \alpha.X\beta, a] \in \mathcal{K}_j$ és $a \neq @$, akkor $[A \to \alpha X.\beta, a] \in \mathit{read}(\mathcal{K}_j, X)$ és az a szimbólum a $\mathit{read}(\mathcal{K}_j, X)$ halmaznak ebbe az elemébe spontán generálható,
- ha $[A \to \alpha.X\beta, @] \in \mathcal{K}_j$, akkor $[A \to \alpha X.\beta, @] \in \mathit{read}(\mathcal{K}_j, X)$ és a @ öröklődik $\mathcal{K}_j$-ből a $\mathit{read}(\mathcal{K}_j, X)$ halmaznak ebbe az elemébe.

A $\mathcal{K}_0$ kanonikus halmaz törzsének egy eleme van, az elem magja $[S' \to .S]$, és ehhez rendeljük hozzá a # előreolvasási szimbólumot. Mivel az összes $\mathcal{K}_j$ kanonikus halmaz törzsének magja már adott, a fenti módszerrel meghatározható, hogy milyen szimbólumok lesznek a spontán generálható és öröklődő előreolvasási szimbólumok.

Adjuk meg azt az algoritmust, ami a spontán generálás és az öröklődés, valamint az $[S' \to .S, \#]$ elem ismeretében meghatározza az $LR(0)$-kanonikus halmazokból az $LALR(1)$-kanonikus halmazokat.

II-4. Szemantikai elemzés veremmel

A fejezetben utaltunk rá, hogy a szemantikai elemzés megvalósítható egy szemantikai verem használatával is. Egészítsük ki a szintaktikai elemzést a szemantikai vermet használó szemantikai elemzéssel, és adjuk meg az alulról-felfelé és a felülről-lefelé haladó elemzések algoritmusait.

II-5. Kiértékelő stratégiák

Láttuk, hogy a jól definiált ATG-kben az attribútumok értékei meghatározhatók. Minden jól definiált attribútum fordítási grammatikához megadható a következő nemdeterminisztikus algoritmus, amelyik egy tetszőleges mondat szintaxisfájának minden pontjában meghatározza az attribútumok értékeit:

Minden egyes attribútumérték kiszámításához rendeljünk hozzá egy folyamatot. Egy folyamat működése indul, ha a hozzátartozó attribútum kiszámí-

táshoz szükséges összes attribútumérték már ismert. Ha egy folyamat befejeződik, akkor az általa meghatározott értékkel további folyamatok indulhatnak el. Ez a folyamat a kitüntetett szintetizált attribútumok felhasználásával kezdődik, hiszen azok értékei ismertek, és a jól definiáltság biztosítja azt, hogy ezzel a módszerrel minden attribútum értéke meghatározható, azaz az algoritmus terminál.

Írjuk meg egy többprocesszoros gépre a fenti algoritmus programját.

II-6. Menetvezérelt kiértékelők

A *menetvezérelt kiértékelők* az attribútumok értékét a szintaxisfa postorder bejárásaival határozzák meg. A bejárások száma szerint megkülönböztethetünk egy- vagy többmenetes stratégiákat.

Írjuk meg a kiértékelő algoritmusát.

II-7. *ASE*-kiértékelők

Ha a kiértékelő a szintaxisfát minden menetben postorder bejárással járja be, akkor *L-R kiértékelőről* beszélünk. Ha a postorder bejárást megváltoztatjuk úgy, hogy a leveleket jobbról-balra haladva járjuk be, akkor a kiértékelőt *R-L kiértékelőnek* nevezzük.

Azokat a többmenetes attribútumkiértékelőket, amelyek az attribútumokat páratlan menetekben *L-R*, a páros menetekben *R-L* stratégiával értékelik ki, *alternáló kiértékelőknek*, röviden *ASE kiértékelőknek* nevezzük.

Írjuk meg az *ASE*-kiértékelő algoritmusát.

II-8. Kódoptimalizálás

Az optimalizálás célja az, hogy a generált kód minősége javuljon, ami alatt azt értjük, hogy a program végrehajtási ideje és a program mérete csökkenjen. Az optimalizáló eljárásokat a kódgenerálás lépésében vagy a kódgenerálás után alkalmazhatjuk, tehát az optimalizálandó program a tárgyprogram.

Az optimalizálással szemben támasztott legfontosabb követelmény a biztonság, amely azt jelenti, hogy az optimalizált programnak ugyanazt az eredményt kell adnia, mint az eredetinek.

Az optimalizálás természetesen nem jelenti az optimális kódú program meghatározását. A fordítóprogram általában nem tudja meghatározni, hogy a program egy része, például egy feltételes elágazás egyik ága a program futásakor semmilyen esetben sem hajtódik végre, tehát törölhető, ezért biztosan nem tud optimális méretű programot létrehozni. Egyes esetekben előfordulhat az is, hogy az optimalizálás a program minőségét rontja.

A kódoptimalizálás lehet *gépfüggő*, amikor például jobb regiszterfelhasználással gyorsítjuk a program futását, vagy lehet *gépfüggetlen*, amikor olyan stratégiákat alkalmazunk, amelyek függetlenek a kódot végrehajtó környezettől.

Tanulmányozzuk a kódoptimalizálás témakörével foglalkozó könyveket.

Megjegyzések

A fordítóprogramok elmélete és írásának gyakorlata egyidős a számítógépekkel, az első programnyelvekkel. Az első fordítóprogramok megírása az 50-es évek elejére tehető. A fordítóprogramok írása sokáig igen nehéz feladat volt, például az első FORTRAN compiler létrehozása 18 emberév munkáját emésztette fel [3]. Ettől kezdve egyre pontosabban határozták meg a fordítás problémáit, egyre jobb fordítási módszereket dolgoztak ki, és egyre jobb program-eszközöket hoztak létre a fordítóprogram írás megkönnyítésére.

A munkában nagy előrelépést jelentett a formális nyelvek és az automaták elméletének fejlődése, és azt lehet mondani, hogy ezek fejlődését elsősorban a fordítóprogramok írásának igénye ösztönözte. Napjainkra az elemző programok írása egyszerű rutin-feladattá vált, lényeges új eredmények most már elsősorban a kódoptimalizálás területén találhatók.

A nemdeterminisztikus, visszalépéses algoritmusok az 1960-as évek elején jelentek meg. Az első két sikeres algoritmus a CYK (Cocke–Younger–Kasami) algoritmus volt 1965–67-ből, és az Earley-algoritmus 1965-ből. A precedencia elemzések különböző fajtáinak kialakulása az 1960-as évek végére, az 1970-es évek elejére tehető. Az LR(k) nyelvtanokat Knuth értelmezte 1965-ben, az LL(k) nyelvtanok első értelmezése az 1970-es évek elejéről származik. Az $LALR(1)$ nyelvtanokat először De Remer tanulmányozta 1971-ben, az $LALR(1)$ elemzés kidolgozása és tanulmányozása az 1980-as évek elejére fejeződött be [1, 2, 3].

Az 1980-as évek közepére nyilvánvalóvá vált, hogy a fordítóprogramokban az LR elemzési módszerek a valóban hatékony módszerek, és azóta a fordítóprogramokban ezeket a módszereket, elsősorban az $LALR(1)$ elemzést alkalmazzák [1].

A fordítóprogramok elméletével és a programok írásával nagyon sok kiváló könyv foglalkozott, közülük az első talán legsikeresebb, de ma már túlhaladott módszereket tárgyaló könyv Gries [19] könyve volt, ebben elsősorban a precedencia nyelvtanokra vonatkozó eredmények találhatók meg. Az új fordítási módszerekkel foglalkozó első, nagy sikert aratott könyv Aho és Ullman [2] kétkötetes műve volt, ebben részletesen megtalálható a CYK- és Earley-algoritmus is. Ezt az úgynevezett „sárkányos könyv" követte, szintén Aho és Ullman munkája [3]. A könyv bővített, javított kiadása 1986-ban jelent meg, az Aho–Ullman–Sethi szerzőhármastól [1].

A teljesség igénye nélkül megemlítjük még Fischer és LeBlanc [15], Tremblay és Sorenson [49], Waite és Goos [51], Hunter[25], Pittman [42] és Mak [33] könyveit. A legújabb eredményeket tartalmazzák többek között Muchnick [38], Grune, Bal, Jacobs és Langendoen [21] művei, vagy a legújabbak, Cooper és Torczon [7] könyve és Louden [32] könyvfejezete.

Magyarul Csörnyei Zoltán [8, 9] kétkötetes egyetemi jegyzete dolgozza fel a fordítóprogramok elméletének és írásának témakörét. A jegyzetek kissé átdolgozott, rövidített anyaga egyetemi tankönyv formájában is megjelent [10]. A Fordítóprogramok című [11] könyv áttekinti a fordítási algoritmusokat a kezdetektől a mai modern módszerekig.

A román nyelvű könyvek közül Şerbănaţi [47] és Simona Motogna [37] könyveit ajánljuk.

Néhány, a formális nyelvek és automaták elméletét tárgyaló könyv is foglalkozik a fordítással. Egy-egy fejezetet találunk az elméletnek a fordítóprogramokban való alkalmazásáról például Bach Iván [5], Fülöp Zoltán [16], Révész György [44] és Varga László [50] könyvében, valamint Dömösi Pál, Fazekas Attila, Horváth Géza és Mecsei Zoltán [14] elektronikus kéziratában. [5, 14, 44] a CYK- és Earley-algoritmust tárgyalják, a precedencia elemzésekről a [5] és [50] könyvekben olvashatunk, [5, 6, 16] az LR elemzésekkel is foglalkozik.

Könyvészet

[1] Aho, Alfred V. – Sethi, Ravi – Ullman, Jeffrey D.: *Compilers, Principles, Techniques and Tools.* Addison-Wesley, 1986.

[2] Aho, Alfred V. – Ullman, Jeffrey D.: *The Theory of Parsing, Translation and Compiling* Vol. I: *Parsing.* Prentice-Hall, 1972.

[3] Aho, Alfred V. – Ullman, Jeffrey D.: *The Theory of Parsing, Translation and Compiling* Vol. II: *Compiling.* Prentice-Hall, 1973.

[4] Asteroth, Alexander – Baier, Christel: *Teoretische Informatik.* Pearson Studium, 2002.

[5] Bach Iván: *Formális nyelvek.* Typotex, Budapest, 2001.

[6] Bánkfalvi Judit – Bánkfalvi Zsolt – Bognár Gábor: *A formális nyelvek szintaktikus elemzése.* Közgazdasági és Jogi Kiadó, Budapest, 1978.

[7] Cooper, Keith D. – Torczon, Linda: *Engineering a Compiler.* Morgan Kaufmann, 2004.

[8] Csörnyei Zoltán: *Bevezetés a fordítóprogramok elméletébe I.* Egyetemi jegyzet. Tankönyvkiadó, Budapest, 1996.

[9] Csörnyei Zoltán: *Bevezetés a fordítóprogramok elméletébe II.* Egyetemi jegyzet. ELTE Kiadó, Budapest, 1996.

[10] Csörnyei Zoltán: *Fordítási algoritmusok.* Erdélyi Tankönyvtanács, Kolozsvár, 2001.

[11] Csörnyei Zoltán: *Fordítóprogramok.* Typotex, Budapest, 2006.

[12] Căzănescu, Virgil Emil: *Introducere in teoria limbaje formale.* Editura Academiei, Bucureşti, 1983.

[13] Demetrovics János – Denev Jordan – Pavlov Radiszlav: *A számítástudomány matematikai alapjai.* Nemzeti Tankönyvkiadó, Budapest, 1999.

[14] Dömösi Pál – Fazekas Attila – Horváth Géza – Mecsei Zoltán:. Formális nyelvek és automaták. Elektronikus kézirat, http://www.inf.unideb.hu/~mecseiz/hallgato/fonya_OK.pdf, 2004.

[15] Fischer, Charles N. – LeBlanc, Richard J.: *Crafting a Compiler*. Benjamin/Cummings, 1988.

[16] Fülöp Zoltán: *Formális nyelvek és szintaktikus elemzésük*. Polygon, Szeged, 2004.

[17] Gécseg, F. – Peák, I.: *Algebraic Theory of Automata*. Akadémiai Kiadó, Budapest, 1972.

[18] Giammarresi, Dora – Montalbano, Rosa: Geterministic generalized automata. *Theoretical Computer Science*, 215. évf. (1999), 191–208. p.

[19] Gries, David: *Compiler Construction for Digital Computers*. John Wiley & Sons, 1971.

[20] Grigoraş, Gheorghe: *Limbaje formale şi tehnici de compilare*. Univ. Al. I. Cuza Iaşi, 1985.

[21] Grune, Dick – Bal, Henri E. – Jacobs, Ceriel J. H. – Langendoen, Koen G.: *Modern Compiler Design*. John Wiley & Sons, 2000.

[22] Harrison, Michael A.: *Introduction to formal language theory*. Addison-Wesley, 1978.

[23] Hopcroft, John E. – Motwani, Rajeev – Ullman, Jeffrey D.: *Introduction to automata theory, languages, and computation*. Addison-Wesley, 2001.

[24] Hopcroft, John E. – Ullmann, J.D.: *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[25] Hunter, Robin W.: *Compilers, Their Design and Construction using Pascal*. John Wiley & Sons, 1985.

[26] Hunyadvári László – Manhertz Tamás:. Automaták és formális nyelvek. Elektronikus kézirat, http://aszt.inf.elte.hu/~hunlaci/book.pdf, 2004.

[27] Iványi Antal (alkotó szerkesztő): *Informatikai algoritmusok II*. ELTE Eötvös Kiadó, Budapest, 2005.

[28] Jucan, Teodor: *Limbaje formale şi automate*. Matrix Rom, Bucureşti, 1999.

[29] Kása Zoltán: *Formális nyelvek és automaták.* Farkas Gyula Egyesület a Matematikáért és Informatikáért, Kolozsvár, 2004.

[30] Livovschi, Leon – Georgescu, Horia – Popovici, Constantin – Ţăndăreanu, Nicolae: *Bazele informaticii.* Editura Didactică şi Pedagogică, Bucureşti, 1981.

[31] Lothaire, M.: *Algebraic Combinatorics on Words.* Cambridge University Press, 2002.

[32] Louden, Kenneth C.: Compilers and interpreters. In Tucker, Allen B. (szerk.): *Handbook of Computer Science.* Chapman & Hall/CRC, 2004, 99/1–99/30. p.

[33] Mak, Ronald: *Writing Compilers and Interpreters.* Addison-Wesley, 1991.

[34] Manna, Zohar: *Mathematical Theory of Computation.* McGraw-Hill Book Co., 1974 (Magyarul: *Programozáselmélet*, Műszaki Könyvkiadó, 1981.).

[35] Marcus, Solomon: *Gramatici şi automate finite.* Editura Academiei, Bucureşti, 1964.

[36] Moldovan, Grigor: *Limbaje formale şi teoria automatelor.* EduSoft, Bacău, 2005.

[37] Motogna, Simona: *Metode de proiectare a compilatoarelor.* Editura Albastră, Cluj-Napoca, 2006.

[38] Muchnick, Steven S.: *Advanced Compiler Design and Implementation.* Morgan Kaufman, 1997.

[39] Peák István: *Bevezetés az automaták elméletébe. Az automaták mint információ átalakító rendszerek 1.* Tankönyvkiadó, Budapest, 1977.

[40] Peák István: *Bevezetés az automaták elméletébe. Az automaták mint felismerő rendszerek 1.* Tankönyvkiadó, Budapest, 1978.

[41] Peák István: *Bevezetés az automaták elméletébe. Automaták kompozíciói. Struktúratételek 3.* Tankönyvkiadó, Budapest, 1980.

[42] Pittman, Thomas: *The Art of Compiler Design, Theory and Practice.* Prentice-Hall, 1992.

[43] Rozenberg, Grzegorz – Salomaa, Arto: *Handbook of formal languages, vol. I–III,.* Springer-Verlag, 1997.

[44] Révész György: *Bevezetés a formális nyelvek elméletébe.* Akadémiai Kiadó, 1979.

[45] Salomaa, Arto: *Theory of Automata.* Pergamon Press, 1969.

[46] Salomaa, Arto: *Formal Languages.* Academic Press, 1973.

[47] Şerbănaţi, Luca D.: *Limbaje de programare şi compilatoare.* Editura Academiei, Bucureşti, 1987.

[48] Sipser, Michael: *Introduction to the Theory of Computation.* PWS Publishing Company, 1997.

[49] Tremblay, Jean-Paul – Sorenson, Paul G.: *The Theory and Practice of Compiler Writing.* McGraw-Hill, 1985.

[50] Varga László: *Rendszerprogramok elmélete és gyakorlata.* Akadémiai Kiadó, Budapest, 1978.

[51] Waite, William M. – Goos, Gerhard: *Compiler Construction.* Springer-Verlag, 1984.

Tárgy- és névmutató