Budapesti Műszaki és Gazdaságtudományi Egyetem
Automatizálási és Alkalmazott Informatikai Tanszék

INFORMATIKA 2

AUTOMATÁK ÉS NYELVEK

Vajk István

2010. március

Tartalomjegyzék

1. fejezet

Automaták és nyelvek

A hétköznapi életben fontos szerepet játszik a nyelv. A nyelv fontos eszköz az emberek közötti kommunikációban. Ezek után nem meglepő, hogy az ember és a számítógép, a számítógép és egy másik számítógép közötti kapcsolatban is nélkülözhetetlen használatuk. A számítógépes nyelvek azonban lényegesen eltérnek a természetes nyelvektől. Egyrészt sokkal egyszerűbbek. Másrészt sokkal merevebbek, sokkal egzaktabbak, matematikai pontosságúak.

A számítógépeket programozási nyelven megírt programokkal vezéreljük. A programozási nyelv megadását formális módszerekkel könnyítik meg. Például a B.W.Kernighan, D.M.Ritchie: A C programozási nyelv című könyvében a C nyelv referencia kézikönyv pontosságú leírása is formális eszközökkel készült. Az adatbázisok világában használt SQL nyelv pontos leírása is könnyen megadható nyelvi formalizmussal.

A számítógépek közötti kapcsolatokat szabályozó protokollok leírásánál is nyelvi eszközöket használnak. Az internet zavarmentes működését az RFC defacto szabványok használata teszi lehetővé, ami a Requests for Comments rövidítése. Az első RFC 1969-ben készült. Ma már az 5000-dik szabványon is túl vannak. A pontos kommunikációs formalizmus megadása formális nyelvi segítség nélkül szinte elképzelhetetlen.

A tárgy Automaták és nyelvek része megismerteti a hallgatókat a számítógépes nyelvészet legalapvetőbb fogalmaival. Megmutatja, hogy a nyelvek milyen szabályrendszerrel írhatók le. Bemutatja, hogy a számítógépekben használt nyelvek milyen strukturális bonyolultságú feladatok megoldását könnyítik meg. Megadja

az egy adott szabályrendszerhez tartozó automaták előállításának formális eszközeit, melyek megkönnyítik számos programozási, algoritmizálási probléma egyszerű és garantáltan hibamentes kezelését. Elméleti szempontból a formális elemzés nem jelent többet, mint annak a kérdésnek az eldöntését, hogy egy adott szöveg a nyelvnek a része, vagyis a nyelv egy mondata-e, vagy sem. Ugyanakkor a formai elemek mögött tartalmi elemek rejtőznek. A formai elemzés lehetőséget ad a szemantikai elemek beépítésére.

A jelen fejezet csak bevezetést kíván nyújtani a nyelvek és az automaták világába. A számítógépes nyelvészet iránt érdeklődő hallgatók Bach Iván: Formális nyelvek című Typotex kiadónál megjelent könyvéből bővebb ismeretekhez juthatnak.

1.1. Bevezetés

A jelen bevezető fejezetben alapvetően három kérdéssel foglalkozunk. Először megadjuk a nyelv matematikai fogalmat, a számítástechnikában használt definicióját. Majd megvizsgáljuk, hogy hogyan lehet a nyelveket leírni, milyen módszerek állnak rendelkezésre a nyelvek megadására, továbbá foglalkozunk a nyelvtan és az automata fogalmával és kapcoslatával. Végül pedig megnézzük, hogy hogyan lehet a nyelveket osztályozni, a nyelvek milyen hierachiát alkotnak, valamint a különböző bonyolultságú nyelvek milyen bonyolultságú automatákkal elemezhetőek.

1.1.1. A nyelv fogalma

Minden nyelv egy szótáron alapul. A szavakból mondatokat állítunk össze. A mondatok képzésénél a nyelv szabályait figyelembe kell venni. A szavak tetszőleges sorozata nem feltétlenül eredményez mondatot. A mondatok együttese alkotja a nyelvet. A formális nyelvek világában az atomi építőkő a szó helyett a karakter. A szótárt pedig a karakterek halmaza jelenti.

Karakterkészlet

A nyelvben használt szimbólumok - másként fogalmazva karakterek - halmazát karakterkészletnek nevezzük. A karakterhalmaz elemei tehát a karakterek.

Legyen Σ a nyelv karakterkészletének véges halmaza. Például:

$\Sigma = \{begin, end, while, if, then, else, for, do, a, b\}$

Egy másik nyelv karakterkészlete lehet:

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

A későbbiekben a nyelv karakterkészletét kisbetűkkel fogjuk jelölni. Például:

$\Sigma = \{a, b, c, d, e, f\}$.

A karakterkészlet számossága mindig véges.

Szöveg

A karakterkészlet karaktereiből alkotott sorozatot szövegnek nevezzük. Legyen

Σ^i a pontosan i hosszúságú szövegek halmaza;

Σ^0 a nulla hosszúságú szöveg, amelynek jele ε ;

Σ^* pedig az összes lehetséges szöveg halmaza, amely az i hosszúságú szövegek uniójaként képezhető: $\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$.

Σ^* számossága, vagyis a karakterekből képezhető szövegek száma megszámlálhatóan sok.

A szövegek között műveleteket értelmezhetünk. Legyen α és β két tetszőleges szöveg. Közöttük az alábbi műveleteket definiáljuk:

- $\alpha\beta$ folytatás (egyesítés, illesztés, konkatenáció),
- α^i α-t i-szer írjuk egymás után,
- α^0 megegyezés szerint legyen ε,
- α^{-1} tükörkép (a karakterek sorrendjének felcserélése),
- szöveg kezdőszelete, végszelete, része ...

Nyelv

A nyelv a Σ karakterkészletből felépített szövegek részhalmaza, azaz

$\omega \in L \subset \Sigma^*$,

ahol ω egy tetszőleges mondata az L nyelvnek.

Megszámlálhatóan sok elemet tartalmazó halmazból megszámlálhatatlanul sok részhalmaz képezhető, így a nyelvek kontinuum számosságúak.

Speciális nyelvek:

$L_0 = \{\}$ az egyetlen mondatot sem tartalmazó nyelv,

$L_\varepsilon = \{\varepsilon\}$ az a nyelv, aminek egyetlen mondata az ε.

A nyelvek között is értelmezhetünk műveleteket. A nyelvekre, mint szöveghalmazokra közvetlenül értelmezhetők a halmazelméleti alapműveletek:

$L_1 \cup L_2$ mindkét nyelv mondatait tartalmazza,

$L_1 \cap L_2$ a két nyelv közös mondatait tartalmazza,

$L_1 - L_2$	L_1 mondatai közül azokat tartalmazza, melyek nincsenek benne L_2-ben,
$\overline{L}$	komplemens halmaz.

Az illesztés (konkatenáció) és a tükrözés műveletét is kiterjeszthetjük szöveghalmazokra:

L_1L_2	eleme minden olyan $\alpha\beta$ páros, amelyre $\alpha \in L_1$, illetve $\beta \in L_2$;
L^{-1}	az L mondatainak tükörképeit tartalmazza.

A fenti definíciókból következik, hogy:

$$L_\varepsilon L = LL_\varepsilon = L,$$

illetve

$$L_0 L = LL_0 = L_0.$$

Az összefüggésekből látszik, hogy az L_ε nyelv az egység, az L_0 nyelv pedig a zérus elem szerepét játssza a nyelvek világában.

1.1.2. A nyelvek megadása

A nyelvek megadása azonos azzal a problémával, hogy hogyan adjuk meg egy halmaz részhalmazát. Ugyanis a nyelv a fenti definíció alapján nem más, mint a Σ^* szövegek halmazának egy részhalmaza. A következőkben három lehetséges megadási módot fogunk megvizsgálni:

- részhalmaz megadása felsorolással,
- nyelvtannal, valamint
- automatával.

1.1.2.1. Felsorolás

Egy nyelv megadható a benne előforduló mondatok felsorolásával. A mondatok számossága általában nem teszi lehetővé a nyelvben előforduló mondatok direkt felsorolását. Sok esetben megelégszünk a mondatok implicit megadásával. Például egy nyelvet definiálhatunk a következő formában:

$$a^i b^i \qquad i > 0.$$

Ezzel egy olyan nyelvet irtunk le, amelynek mondatai:

$$ab, aabb, aaabbb, ...$$

1.1.2.2. Nyelvtan (Grammatika)

A nyelveket megadhatjuk nyelvtanukkal is, vagyis egy szabályrendszerrel, aminek a felhasználásával a nyelv mondatai levezethetők. Formálisan a grammatikát az alábbi négyes határozza meg:

$$G(N, \Sigma, P, S)$$

ahol

- N nyelvtani kategóriák (nemterminálisok) halmaza,
- Σ karakterkészlet (terminálisok halmaza),
- P az úgynevezett levezetési, helyettesítési, szintaktikai szabályok összessége és
- S a mondatszimbólum (kiinduló szimbólum), ahol $S \in N$.

A grammatikák azért alkalmasak egy nyelv megadására, mert segítségükkel egy nyelv mondatai a mondatszimbólumból kiindulva levezethetők. A levezetések sorozatában az egyik mondatszerű formából a következőt a levezetési szabályok alkalmazásával származtathatjuk. A mondatszerű forma abban különbözik a mondattól, hogy abban nemcsak a nyelv karakterkészletének elemei, hanem a grammatikai szimbólumok is szerepelhetnek. A sikeres levezetés során az utolsó lépés eredményeként olyan mondatszerű formát kapunk, ami már csak terminálisokat tartalmaz.

1. példa

Álljon egy nyelvtan az alábbi elemekből:

Σ: *kutyám, macskám, eszik, iszik*

N : <*alany*>, <*állítmány*>, <*mondat*>

P : <*mondat*> → <*alany*> <*állítmány*>
<*alany*> → *kutyám*
<*alany*> → *macskám*
<*állítmány*> → *eszik*
<*állítmány*> → *iszik*

S : <*mondat*>

A fenti nyelvtan alapján példaként az alábbi mondat vezethető le:
<*mondat*> ⇒ <*alany*> <*állítmány*> ⇒ *kutyám* <*állítmány*> ⇒ *kutyám iszik*

Könnyen belátható, hogy a most definiált nyelvnek összesen négy mondata van. Ezek: *kutyám eszik*, *kutyám iszik*, *macskám eszik*, *macskám iszik*.

A nemterminálisokat általában nagybetűvel, a karakterkészletet (terminálisokat) kisbetűvel szokták jelölni.

2. példa

A felsorolásnál (a 1.1.2.1. alfejezetben) leírt nyelv például a következőképpen is megadható lenne nyelvtan segítségével:

N: $\{S\}$

Σ: $\{a, b\}$

P: $S \to ab \mid aSb$

S: S

3. példa

N: $\{S, A, B\}$

Σ: $\{a, b, c\}$

P: $S \to aSAB \mid abB$

$BA \to AB$

$bA \to bb$

$bB \to bc$

$cB \to cc$

S: S

Nézzünk két lehetséges levezetést a fenti nyelvtan alapján:

a. eset:

$\underline{S} \Rightarrow a\underline{S}AB \Rightarrow aab\underline{BA}B \Rightarrow aa\underline{bA}BB \Rightarrow aab\underline{bB}B \Rightarrow aabb\underline{cB} \Rightarrow aabbcc$

b. eset:

$\underline{S} \Rightarrow a\underline{S}AB \Rightarrow aa\underline{bB}AB \Rightarrow aabcAB \Rightarrow ?$

Látható, hogy a második levezetésnél nem tudunk továbblépni, tehát nem tudunk mondatot generálni belőle. Ezt sikertelen levezetésnek hívjuk. A fenti szabályrendszer segítségével generálható nyelv az

$a^i b^i c^i \quad i > 0$

mondatokat tartalmazza.

Kiegészítések

Programozásból ismert, hogy egy nyelvet általában egy másik nyelvvel definiálhatunk. A programozási nyelvek világában egy szokásosan alkalmazott nyelv a BNF (Backus Naur Form) nyelv. Ennek elemei:

- terminálisok,
- nemterminálisok (Jele: <...>) ,
- értékadás ::= (Az értékadás formája: <....>::=<....><....>...),
- folytatás művelete. (Műveleti jele nincsen.)

Az EBNF, a BNF kiterjesztése az egyszerűbb nyelvmegadások érdekében további műveleteket vezet be:

- | az alternatíva, a *vagy* jelölésére,
- [...] *opcionális* részek jelölésére (nullászor vagy egyszer)
- {....} az *ismétlésekre* (nullászor vagy többször).

Példa BNF-ben leírt nyelvre:

<*számjegy*>::=0

<*számjegy*>::=1

...

<*számjegy*>::=9

<*szám*>::=<*számjegy*>

<*szám*>::=<*szám*> <*számjegy*>

Ugyanez kiterjesztett BNF-ben (EBNF-ben):

<*számjegy*>::=0|1|2|3|4|5|6|7|8|9

<*szám*>::=<*számjegy*> {<*számjegy*>}

A nyelvek *szintaktikája* a nyelv formai megkötéseit tartalmazza. A mondatok azonban jelentéstartalommal is bírnak. Ennek a leírását *szemantikának* nevezik. A szemantikának igazság tartalma van. Leírása bonyolultabb, mint a szintaktika megadása.

Egy nyelv több nyelvtannal is leírható, de nem minden nyelvhez található nyelvtan. Minket azonban csak azok a nyelvek érdekelnek, amelyekhez nyelvtan szerkeszthető.

1.1.2.3. Automata

Egy nyelv feldolgozójának, fordítójának nem az a feladata, hogy a mondatokat generálja, hanem hogy a mondatokat felismerje és elemezze. Erre kiváló eszköz a nyelvekhez konstruálható automata, amelynek felépítését a 1.1. ábrán láthatjuk. Az automata képes eldönteni a bemenetre érkező karakterek sorozatából alkotott szövegről, hogy eleme-e a nyelvnek.

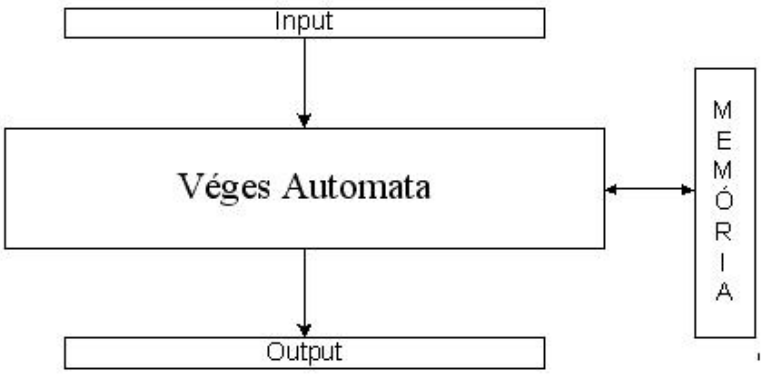
Az automaták rendkívül egyszerű gépek. A bemeneti szalagról, az inputról képes a szöveg karaktereit olvasni. A bemeneti szalag, mint neve is jelzi, csak bemenetként szolgál, csak olvasható. Az automata a feldolgozás eredményét a kimenetre írja. A bonyolultabb automaták egy írható és olvasható memóriával is rendelkeznek. A véges állapotú automata a bemenetről olvasott karakter, a (sok esetben veremszerkezetként működő) memóriában található információ, valamint az automata állapota szerint hozza meg döntéseit. Ezek:

- mit írjon a kimenetre,
- mit írjon a memóriába,
- milyen állapotba lépjen.

Az automaták legegyszerűbb formái csak arra a kérdésre keresnek választ, hogy a szöveg a nyelvtani szabályok szerint értelmezhető-e. Ebben az esetben a kimenet egyetlen egy "lámpa" is lehetne, amely felvillanásával jelezhetné, hogy a felismerés sikerült. A fordító automaták esetén azonban a kimenet tényleges szöveg. Az automata egy transzformációt végez, egy leképezést valósít meg. Vannak olyan automaták is, amelyeknek nincsen bemenete. Feladatuk, hogy véletlenszerűen generálják a nyelv mondatait. A tárgy keretében alapvetően a felismerő automatákkal foglalkozunk.

1.1.3. A nyelvek osztályozása

Chomsky a helyettesítési szabályok komplexitása alapján négy nyelvosztályt definiált, s ezeket számokkal jelölte 0-tól 3-ig. Az osztály számának növekedésével



1.1. ábra. Az automata felépítése

egyre szigorúbb megkötéseket írt elő.

3-as nyelvosztály - Reguláris (szabályos) nyelvek

A reguláris nyelvtanban csak az alábbi szabálytípusok alkalmazhatók:

- Jobbreguláris nyelvtan esetén:
  $A \to a \mid bB$
- Balreguláris nyelvtan esetén pedig:
  $A \to a \mid Bb$

Látható, hogy a különbség abban áll, hogy a terminálisnak melyik oldalán áll a nemterminális. Ebbe a nyelvosztályba tartoznak a programozás alapvető építőkövei. Például: a változók, számok, kulcsszavak feldolgozása. Ezeket a fordítás során tokenizálásnak nevezik.

Nézzünk néhány példát:

$S \to a \mid aS$

Ez a nyelv olyan mondatokat tartalmaz, ami csak a karakterből áll:

$a^i \qquad i > 0.$

Egy másik példa reguláris nyelvekre, amely az $a^i b^j \ i, j > 0$ mondatokból áll.
Ez a nyelv az

$S \to aA$

$A \to aA \mid bB \mid b$

$B \rightarrow bB \mid b$

nyelvtannal is megadható.

2-es nyelvosztály - Környezetfüggetlen nyelvek (CF (context free) nyelvek)

Ezen nyelvosztályba tartozó nyelvek levezetési szabályai

$A \rightarrow \alpha$

formájúak. Bal oldalon egy nemterminális áll, míg a jobb oldalon tetszőleges terminálisok és nemterminálisok sorozata.
A programozási nyelveknél például az aritmetikai kifejezések leírása, valamint minden olyan struktúrának a megadása, amely egymásba ágyazható szerkezeteket (nyitó és csukó zárójelek, begin-end,...) tartalmaz, ezzel a nyelvosztállyal adható meg.

Nézzünk egy egyszerű példát:

$S \rightarrow ab \mid aSb$

Ez a nyelvtan olyan mondatokat generál, amelyekben ugyanannyi a és b karakter van egymás után:

$a^i b^i \quad i > 0.$

1-es nyelvosztály - Környezetfüggő nyelvek (CS (context sensitive) nyelvek)

Helyettesítési szabályainak korlátait kétféle módon is megadhatjuk.

- Az első megadási mód szerint a levezetési szabályok alakja:

  $\beta A \gamma \rightarrow \beta \alpha \gamma$

  A fenti formájú szabály azt jelenti, hogy egy nemterminális csak akkor helyettesíthető a megfelelő terminálisok és nemterminálisok sorozatával, ha előtte és utána a szabály által előírt terminálisok és nemterminálisok állnak.
- Második megadási mód szerint:

  $\alpha \rightarrow \beta$, ahol $|\alpha| \leq |\beta|$.

  A fenti megkötés azt fejezi ki, hogy a helyettesítési szabály jobboldala nem lehet rövidebb a baloldalnál. Ezen tulajdonság alapján ezeket a nyelveket nem csökkentő nyelveknek is nevezik.

A nyelvosztályra megadott két, látszólag különböző meghatározás valójában ugyanazt a halmazt definiálja.

Egy lehetséges nyelv ebből a nyelvosztályból: $a^i b^i c^i \quad i > 0$. Egy másik példa a környezetfüggő nyelvre az a nyelv, amelyben csak a karakter fordul elő, és a szöveg hossza kettő egész számú hatványa.

0-ás nyelvosztály - Rekurzívan felsorolható nyelvek

A 0-ás nyelvosztályban alkalmazható helyettesítési szabályokra nincs megkötés. A levezetési szabályok általános formája:

$\alpha \rightarrow \beta$

azaz semmilyen megkötés nincsen. Ezeket a nyelveket rekurzíven felsorolható nyelveknek is nevezik.

Nézzünk egy példát ebbe a nyelvosztályba tartozó nyelvre. Ilyen nyelv lehet az a nyelv, amelynek a szavai csak a karakterekből állnak, és a karaktersorozat hossza prímszám.

Annak ellenére, hogy az $A \rightarrow \varepsilon$ szabály formailag a 0-s nyelvosztályba tartozik, megengedjük megjelenését valamennyi nyelvosztályban.

A grammatikákat osztályozó szabályok típusa alapján könnyen belátható, hogy minden 3. típusú grammatika egyben 2. típusú is, és minden 2. típusú egyben 1. típusú is. Hasonlóképpen minden 1. típusú 0. típusú is. A nyelvosztályokhoz tartozó nyelvek hierarchiát alkotnak:

$\mathbf{L}_3 \subset \mathbf{L}_2 \subset \mathbf{L}_1 \subset \mathbf{L}_0$.

A gyakorlati alkalmazások szempontjából kiemelkedő szerepe van a reguláris és a környezetfüggetlen nyelveknek. A későbbi fejezetekben részletesen foglalkozunk ezen nyelvek elemzésével.

A nyelvosztályok és az egyes nyelvosztályokba tartozó mondatok felismerésére szolgáló automaták típusa között szoros kapcsolat van. Az egyes nyelvosztályokba tartozó nyelvek mondatai különböző bonyolultságú automatákkal elemezhetők.

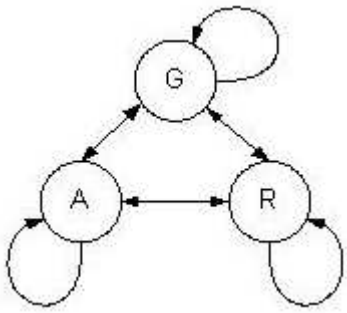
A regulás nyelveket véges automatákkal, a környezetfüggetlen nyelveket egy vermet tartalmazó automatával, a környezetfüggő nyelveket kettős veremautomatákkal vagy lineárisan korlátozott automatákkal (szöveg hosszával arányos hosszúságú szalaggal rendelkező Turing-gép), a 0-ás nyelvosztály szövegeit pedig Turing-gépekkel lehet elemezni. A Turing-gép egy olyan automata, amely egy szalag jellegű végtelen memóriával és egy fejjel rendelkezik. Működése során a szalagot képes olvasni és módosítani, továbbá a szalag mentén képes jobbra és balra mozogni. Az alábbi táblázat a nyelvek és az automaták közötti kapcsolatot tartalmazzák.

Chomsky típus	Nyelv	Automata
0	Rekurzívan felsorolható	Turing-gép
1	Környezetfüggő	Lineárisan korlátozott
2	Környezetfüggetlen	Veremautomata
3	Reguláris	Véges automata

Az automaták és a nyelvek kapcsolata

1.2. Reguláris nyelvek

A számítástechnika nyelvek közül a legegyszerűbb nyelvek a reguláris nyelvek. A reguláris nyelvek számos módon megadhatók. A jelen fejezetben három megadási módot mutatunk be: a grammatikával (G), az automatával (A), illetve a reguláris kifejezéssel (R) történő leírásokat. Ahogy ezt a 1.2. ábra mutatja, bármely leírási módból áttérhetünk bármely másikba, illetve egy nyelvet több különböző automatával, grammatikával és reguláris kifejezéssel is megadhatunk.



1.2. ábra. A reguláris nyelvek leírási módjai, valamint a leírási módok közötti lehetséges áttérések

A jelen fejezet két nagyobb részből áll. Az első regulásis nyelvek leírási, megadási lehetőségeivel, a második pedig a regulási nyelvek leírási formái közötti áttérésekkel foglalkozik.

1.2.1. A reguláris nyelvek leírása

A bevezető fejezetben láttuk, hogy egy nyelv, amely nem más mint a szövegek részhalmaza, többféle képpen is megadható. Egyik lehetséges módja a leírásnak, hogy megadjuk azokat a mondatokat, amelyek a nyelvhez tartoznak. Egy másik lehetőség, hogy megadjuk azt a szabályrendszert, amely segítségével a nyelv mondatai előállíthatók. Egy harmadik lehetőség pedig, hogy megadjuk azt az algoritmust, amely segítségével eldönthető egy tetszőleges szövegről, hogy eleme-e a nyelvnek. Következőkben ezt a három lehetőséget vizsgáljuk meg a reguláris nyelvekre korlátozva.

1.2.1.1. Grammatika

A reguláris nyelveket, mint ahogy azt már megismerhettük a Chomsky féle 3. osztályba tartozó grammatikák generálják:

- Jobbreguláris nyelvtan esetén:
  $A \to a \mid bB$
- Balreguláris nyelvtan esetén pedig:
  $A \to a \mid Bb$

Példa: Adjuk meg az

$a^i b^j c^k \quad i, j, k > 0$

mondatokat generáló jobbreguláris nyelvtant!

A grammatika:

$$G(N, \Sigma, P, S)$$

ahol

- nemterminálisok halmaza: $N = \{S, A, B, C\}$,
- karakterkészlet: $\Sigma = \{a, b, c\}$,
- A levezetési szabályok:
  $S \to aA$
  $A \to aA \mid bB$
  $B \to bB \mid cC \mid c$
  $C \to cC \mid c$
- a mondatszimbólum: S

1.2.1.2. Automata

A reguláris nyelveket elfogadó automataosztály a véges automaták osztálya. A véges automatát úgy kell elképzelni, hogy mozgása során a kiinduló állapotból indulva egyik állapotból megy a másikba. Egy-egy lépése során mindig először beolvas egy bemenő karaktert, majd pedig a pillanatnyi állapotától és a bemenő karaktertől függően veszi fel a következő állapotát.

A véges automatát matematikai eszközökkel az alábbiak szerint írhatjuk le:

$M(Q, \Sigma, \delta, q_0, F)$,

ahol

- Q - az automata lehetséges állapotainak véges halmaza,
- Σ - elemzendő jelsorozat karakterkészlete,
- δ - a mozgás szabályainak halmaza,
- $q_0 \in Q$ - induló állapot (az automata ebből az állapotból kezdi feldolgozni a szöveget) és
- $F \subseteq Q$ - elfogadó állapotok halmaza.

Az automata akkor fogadja el az elemzendő szöveget mondatnak, ha

- a teljes szöveget feldolgozta, és
- az utolsó karakter feldolgozása után egy az elfogadó állapotok halmazában lévő állapotba jutott.

A mozgási szabályok az automaták világában ugyanolyan szerepet játszanak, mint a helyettesítési szabályok a grammatikák világában. A mozgási szabályok mondják meg, hogy egy adott állapotban egy adott karakter beolvasásának hatására milyen új állapotot vesz fel az automata. Amennyiben egy állapotból ugyanannak a karakternek a hatására több állapotba is eljuthatunk, az automatánk nemdeterminisztikus. Ilyen esetben az automatánk minden lehetséges megengedett mozgását követni kell. Az elfogadás feltétele pedig, hogy az utolsó karakter feldolgozása után legyen olyan mozgás, amely eredményeként az automata az elfogadó állapotokhoz tartozó állapotba jut.

A mozgási szabályok megadási módjai

A. A mozgási szabályok megadása függvénnyel

A mozgási szabályok megadhatók függvénnyel. A $\delta(A, a) = B$ leképezés azt jelenti, hogy az A állapotból a karakter hatására B állapotba jut az automata. Másképpen megfogalmazva: a mozgási szabályok egy leképezést valósítanak meg; az automataállapotok és a karakterkészlet halmazát leképezik az automataállapotok halmazára:

$$Q \times \Sigma \Rightarrow Q.$$

Ha minden állapot-karakterpár esetén van mozgási szabály, akkor azt mondjuk, hogy az automata teljesen specifikált. Ha minden egyes állapot-karakterpárhoz

csak egy lehetséges mozgási szabály tartozik, akkor az automata determinisztikus automata. Ha van olyan állapot-karakterpáros, amelyhez egynél több leképezés tartozik, akkor az automata nemdeterminisztikus. A nemdeterminisztikus automaták egy speciális fajtája az epszilon mozgást is megengedi. Ebben az esetben a mozgási szabályok a

$$Q \times (\Sigma \cup \{\varepsilon\}) \Rightarrow Q.$$

leképezést valósítják meg.

Minden nemdeterminisztikus és epszilon mozgást is megengedő automata átalakítható vele ekvivalens determinisztikus automatává. A determinisztikus automata realizációk közül azt az automatát, amelyek a legkisebb állapotszámmal rendelkezik minimál automatának nevezik.

Példa Adjuk meg az

$a^i b^j c^k \quad i, j, k > 0$

mondatokat elfogadó automatát!

Az $M(Q, \Sigma, \delta, q_0, F)$ automata:

- az állapotok halmaza: $Q = \{S, A, B, C\}$
- a karakterkészlet: $\Sigma = \{a, b, c\}$
- a kezdeti állapot: $q_0 = S$
- az elfogadó állapot(ok) halmaza: $F = \{C\}$
- a mozgási szabályok:
  $\delta = \{\delta(S, a) = A,\ \delta(A, a) = A,\ \delta(A, b) = B,\ \delta(B, b) = B,\ \delta(B, c) = C,$
  $\delta(C, c) = C\}$

B. A mozgási szabályok megadása gráffal

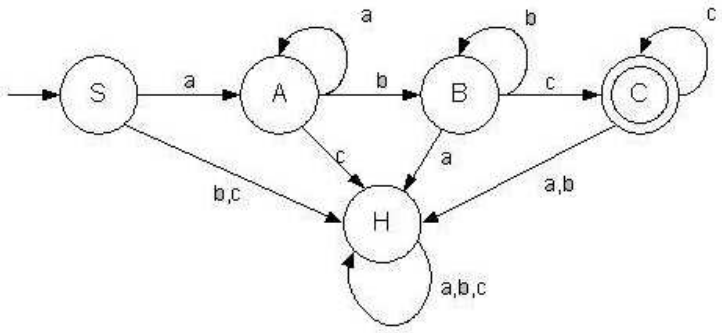
A véges automata leírható állapot-átmeneti gráffal is. A gráf csomópontjai az állapotok, a gráf élei pedig a lehetséges állapotátmenetek. Az állapotokat körökkel, a mozgási szabályokat két állapot közötti irányított nyíllal jelöljük. A lehetséges állapotátmenteknél megadjuk azt a karaktert, amelynek érkezése esetén mehetünk át az egyik állapotból a másikba. A kezdő állapotot egy az állapotba mutató nyíllal

adjuk meg, az elfogadó állapotot pedig kettős körbe helyezzük. Ha a mozgási szabályok között szerepel a $\delta(A, a) = B$ szabály, akkor az A állapotból a B állapotba egy a jelű él vezet.

Nézzük meg, hogyan adható meg

$a^i b^j c^k \quad i, j, k > 0$

mondatokat elfogadó automata! A fenti mondatokat elfogadó automata állapot-átmeneti gráfját a 1.3. ábra mutatja. Az ábra alapján látható, hogy az automata állapotainak a halmaza: $Q = \{S, A, B, C, H\}$, a karakterkészlete $\Sigma = \{a, b, c\}$, a kezdőállapot $q_0 = S$ és az elfogadó állapotok halmaza pedig: $F = \{C\}$.



1.3. ábra. Állapot-átmeneti gráf

A 1.3. ábrán látható állapot-átmeneti gráfban az összes lehetséges mozgási szabályt jelöltük. Az állapot-átmeneti gráf így teljesen specifikált. Ha az automata a H hibaállapotba jut, akkor a beolvasott karaktersorozat nem mondat. Innen nincsen kiút, és nem is elfogadó állapot.

C. A mozgási szabályok megadása táblázattal

Az állapotátmenetek könnyen leképezhetők egy táblázatba is. A táblázat első oszlopában az aktuális állapotok szerepelnek, az első sorában pedig a karakterkészlet. A táblázat elemei az automata következő állapotait tartalmazzák. A táblázatot úgy értelmezhetjük, hogy az automata adott karakter hatására, adott állapotból milyen állapotba kerül. Az automatát a táblázat teljesen definiálja, ha megadjuk a kezdő állapotot, valamint az elfogadó állapotok halmazát. A kezdő állapotot nyíllal, a végállapotokat pedig kövér szedésű karakterrel jelöltük. A fenti

automata táblázatos formában:

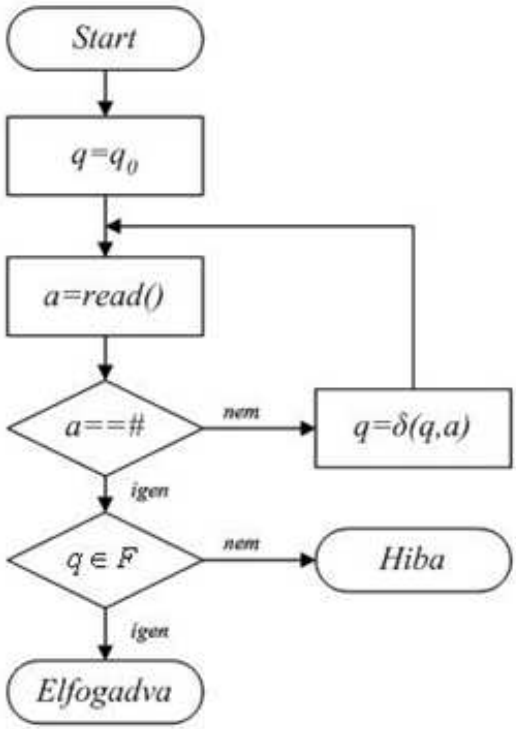
	a	b	c
→S	A	H	H
A	A	B	H
B	H	B	C
C	H	H	C
H	H	H	H

Az S a kiinduló, a C pedig az elfogadó állapot. A jelen példában a táblázat minden egyes cellájában csak egy állapot található, az automata determinisztikus. Nemdetermisztikus automata esetén egy adott állapotból egy adott karakter hatására több állapotátmenet is lehetséges. Ilyenkor egy cellába több állapot (állapotok halmaza) kerül.

D. Az automata leképezése programmal

A determinisztikus véges automata folyamatábrájának a működését a 1.4. ábrán követhetjük nyomon. Az ábrán # a sorvége karaktert jelöli.

A táblázatos leírásra épülő determinisztikus véges automatának könnyen elkészíthetjük a számítógépes programját. Pszeudo pascal programozási nyelven megadjuk az előző fejezetben bemutatott automatát megvalósító programot. A könynyebb olvashatóság érdekében ékezetes betűket is használunk, amit a pascal nyelv nem enged meg. Feltételezzük, hogy rendelkezésünkre áll az *olvas* függvény, amely a soronkövetkező karaktert olvassa be az inputról. A függvény visszaadott értéke hamis, ha a szöveg végére értünk, különben pedig igaz.



1.4. ábra. A determinisztikus véges automata folyamatábrája

```
type állapot_típus=(S,A,B,C,H);
     karakter_típus =('a','b','c');
const táblázat: array [állapot_típus,karakter_típus] of állapot_típus=
          ((A, H, H), (A, B, H), (H, B, C), (H, H, C), (H, H, H));
var  aktuális_állapot: állapot_típus;
     aktuális_karakter: karakter_típus;
begin
     aktuális_állapot:=S;
     while olvas(aktuális_karakter) do
          aktuális_állapot:=táblázat [aktuális_állapot,aktuális_karakter];
     if aktuális_állapot in [C] then writeln('OK!')
     else writeln('Nincs ilyen mondat!');
end.
```

A programot célszerű kiegészíteni további ellenőrzésekkel is annak érdekében, hogy a nyelvhez nem tartozó karakterek esetén ne eredményezzen futási hibát.

1.2.1.3. Reguláris kifejezés

Egy reguláris nyelvet egyértelműen megadhatunk reguláris kifejezéssel is. A reguláris kifejezésben az alábbi alaphalmazokat használjuk:

- $\varnothing$ $\{\}$ üres halmaz,
- ε $\{\varepsilon\}$ csak az ε -t tartalmazó halmaz,
- $a, b, ...$ $\{a\}, \{b\}, ...$ a karakterkészlet elemei.

Az alaphalmazok között műveleteket értelmezünk. A reguláris kifejezésekben megengedett műveletek:

- $x + y$ $\{x, y\}$ alternatíva,
- xy $\{xy\}$ folytatás,
- x^* $\{\varepsilon, x, xx, xxx, ...\}$ iterált.

Gyakran használják még a következő műveletet is

- x^+ $\{x, xx, xxx, ...\}$

Ez az előzőekből levezethető: $x^+ = xx^*$.

Reguláris kifejezést kapunk a fenti műveletek véges sokszori alkalmazásával. A műveletek felírásánál a zárójelek használata megengedett.

1. példa

Az $ab^*a + ba$ reguláris kifejezéssel leírt nyelv mondatai:

$ba, aa, aba, abba...$

2. példa

Az

$a^i b^j c^k \quad i, j, k > 0$

mondatokat tartalmazó nyelv reguláris kifejezéssel

$aa^*bb^*cc^*$

formában írható fel. Egy alternatív felírási módja: $a^+b^+c^+$.

A reguláris kifejezések átalakíthatók. Az átalakítást segítheti az alábbi azonosságok használata:

- $x + x = x$
- $x + y = y + x$ (kommutatív összeadásra)
- $xy \neq yx$ (nem kommutatív folytatásra)
- $(x + y) + z = x + (y + z)$ (asszociatív összeadásra)
- $(xy)z = x(yz)$ (asszociatív folytatásra)
- $x(y + z) = xy + xz$ (disztributív összeadásra)
- $(x^*)^* = x^*$
- $(x + y)^* = (x^* + y^*)^*$
- $(x + y)^* = (x^* y^*)^*$
- $\varnothing^* = \varepsilon$
- $xx^* + \varepsilon = x^*$

Példa: Határozzuk meg, hogy

$R{=}aabb^* + aa + bb^*aab^*$

reguláris kifejezés milyen mondatokat ír le! A fenti azonosságok felhasználásával:

$$\begin{aligned} R &= aabb^* + aa + bb^*aab^* \\ &= (aabb^* + aa) + bb^*aab^* \\ &= aa(bb^* + \varepsilon) + bb^*aab^* \\ &= aab^* + bb^*aab^* \\ &= (\varepsilon + bb^*)aab^* \\ &= b^*aab^* \end{aligned}$$

Az átalakítások után látható, hogy az R reguláris kifejezéssel olyan mondatok képezhetők, amelyekben két a karakter van egymás után, és előtte és utána akárhány b karakter áll.

A szoftverekben gyakran használják a reguláris kifejezéseket. Például a *grep* programot direkt arra készítették, hogy egy reguláris kifejezéssel leírt mintának az előfordulását megtalálják egy állományban. Az elnevezés rendkívül régről származik: a *global regular expression print* rövidítése. Mind a DOS, mind a WINDOWS, mind pedig a UNIX környezetben használható. Itt a UNIX alatti verziónak a leírását adjuk meg:

Az alaphalmazok megadása:

- c : c karakter (nem lehet speciális karakter)
- $\backslash c$: c karakter (most c speciális karakter is lehet pl. .,[, ...)
- $\hat{}$: sor eleje
- \$: sor vége
- . : tetszőleges karakter
- $[abc]$: karakterhalmaz megadása
- $[a\text{-}z]$: karakterhalmaz megadása intervallummal
- $[\hat{}abc]$: karakterhalmaz megadása kizárással (nem a, b vagy c karakter)

A műveletek megadása:

- $r*$: ismétlés 0,1,2,...-szor
- $r+$: ismétlés 1,2,...-szor
- $r?$: opció (ismétlés 0-szor vagy 1-szer)
- (r) : zárójelezés
- $r1 \mid r2$: alternatíva

1. példa

A fenti szintaktikával írjuk fel azt a kifejezést, amely megadja a betűvel kezdődő betű-szám kombinációval folytatódó HTML kiterjesztésű állományok nevét!

A megoldás:

$$[a-zA-Z][a-zA-Z0-9]*\backslash.HTML$$

2. példa

Adjuk meg azt a kifejezést, amely leírja azokat a sorokat, melyek a karakterrel kezdődnek és b karakterrel végződnek!

A megoldás:

$\hat{}a.^*b\$$

1.2.2. A reguláris nyelvek leírási formái közötti áttérések

Az előző fejezet alapján látható, hogy egy reguláris nyelv megadható nyelvtannal, automatával, valamint reguláris kifejezéssel is. Ebben a fejezetben arra keresünk

választ, hogy a reguláris nyelvek leírási formái közötti áttérések hogyan valósíthatók meg. Megnézzük, hogy a jobb- és balreguláris nyelvtanokhoz hogyan szerkeszthető automata, a reguláris kifejezések milyen automatával dolgozhatók fel, továbbá hogyan készíthető determinisztikus automata nemdeterminisztikus automatához, valamint hogyan állítható elő a minimális állapotszámmal rendelkező determinisztikus automata.

1.2.2.1. Átalakítás nyelvtan és automata között

Először nézzük, hogyan lehet meghatározni egy reguláris nyelvtanhoz tartozó automatát! *Jobbreguláris nyelvtan* esetén az alábbi nyelvtani szabályok lehetnek a grammatikában:

$A \to aB \mid b.$

Először nézzük a

$A \to aB$

szabályt! Az A nemterminális olyan szöveget jelent, amely a-val kezdődik és a B nemterminális által reprezentált szöveggel folytatódik. Ez az automaták világában a

$\delta(A, a) = B$

mozgási szabállyal valósítható meg. Azaz a leképezés úgy értelmezhető, hogy az A állapotból olyan szöveget tudunk feldolgozni, ami a karakterrel kezdődik és a B állapotból elfogadható szöveggel folytatódik. Az

$A \to b.$

szabály pedig azt jelenti, hogy az A nemterminális az egyetlen b karakterből álló szöveggel helyettesíthető. Az automata nyelvén ez úgy interpretálható, hogy az A állapotból olyan szöveget fogadunk el, amely egyetlen b karakterből áll. Másképpen fogalmazva ez azt jelenti, hogy az állapotáttérés után elfogadó állapotba jutottunk:

$\delta(A, b) = E,$

ahol E elfogadó állapot.

Példa

Nézzük meg, hogyan készíthető automata az alábbi nyelvtanhoz:

$S \to aA$

$A \to aA \mid bB$

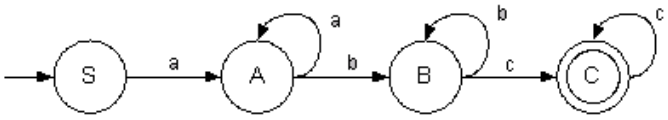
$B \to bB \mid cC \mid c$

$C \to cC \mid c$

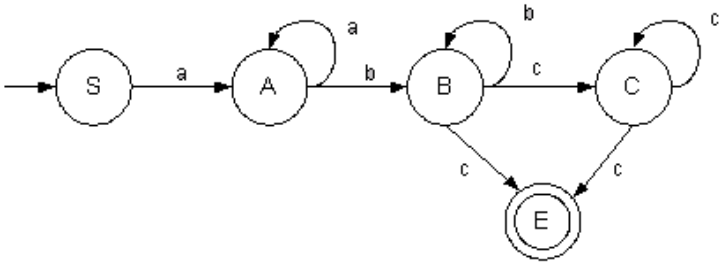
A nyelvtan nemterminálisainak az automatáknál állapotok feleltethetők meg. Az S mondatszimbólumnak például az automata S kezdőállapota. S-ből olyan szövegek dolgozhatók fel, amelyek a-val kezdődnek és A-ból feldolgozható szöveggel folytatódik. Azaz S-ből az a jelölésű nyíl mutat A-ba.

...

$B \to c$ azt jelenti, hogy B-ből olyan szöveget dolgozunk fel, amely egyetlen c karakterből áll, vagyis elfogadó állapotba jutunk. Ezek alapján a fenti nyelvtannak megfelelő automata ábrája:



A fenti automata alábbi alakra is hozható.



Egy külön E elfogadó állapotot generáltunk. Összesen egy ilyen állapot van. Az E elfogadó állapotba az utolsó karakter beolvasása után jutunk. Formálisan a nyelvünk grammatikája átalakítva:

$S \to aA$

$A \to aA \mid bB$

$B \to bB \mid cC \mid cE$

$C \to cC \mid cE$

$E \to \varepsilon$

A nyelvtanunk ebben az esetben csak $A \to aB$ jellegű szabályokat tartalmaz a kitüntettet $E \to \varepsilon$ szabálytól eltekintve, amelyből az E végállapot keletkezik. Automatánk ebben az esetben azonban nemdeterminisztikus, mivel mind a B,

mind pedig a C állapotban nem tudjuk előre, hogy az utolsó karaktert dolgoztuk-e fel.

Az automatánk formailag egyszerűbb formát ölt, ha megengedjük az ε szabályok használatát. Nyelvtanunk ε szabály felhasználásával

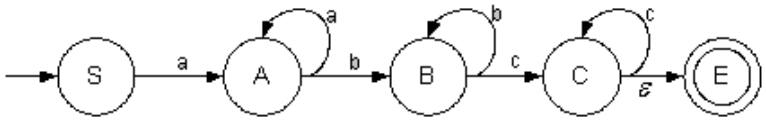
$S \to aA$

$A \to aA \mid bB$

$B \to bB \mid cC$

$C \to cC \mid \varepsilon$

alakra redukálódik. A hozzá tartozó automata az alábbi ábrán látható.



Balreguláris nyelvtan esetén az alábbi nyelvtani szabályok lehetnek a grammatikában:

$A \to Ba \mid b$

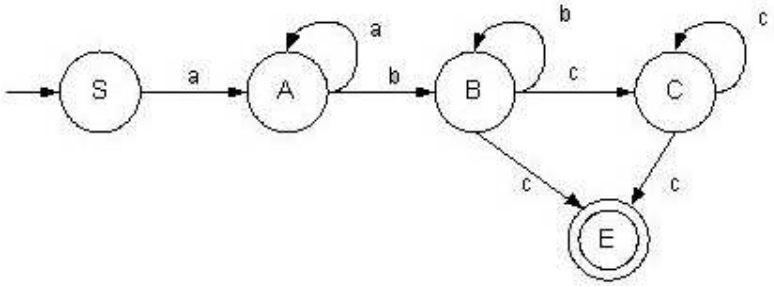
A fenti helyettesítési szabályok azt jelentik, hogy A olyan szövegek helyett áll, amely B által leképezhető szöveg folytatása egy a karakterrel, vagy egyetlen egy b karakter. Ez az alábbi mozgási szabályokkal realizálható:

$\delta(B, a) = A$, illetve $\delta(q_0, b) = A$.

A fenti szabályokat úgy értelmezhetjük, hogy A állapotba az automata a B állapotból a karakter érkezése esetén jut, illetve kiinduló állapotból b karakter érkezése után.

Az eredeti nyelvtan mondatszimbólumának az automata elfogadó állapota felel meg, míg az újonnan bevezetett q_0 állapot lesz az automata kiindulási állapota. Ha egy automatának több elfogadó állapota van, akkor a balreguláris nyelvtan felírása előtt ezeket egyesíteni célszerű.

Az alábbi automatából



a következő balreguláris nyelvtan generálható:

$E \to Bc \mid Cc$

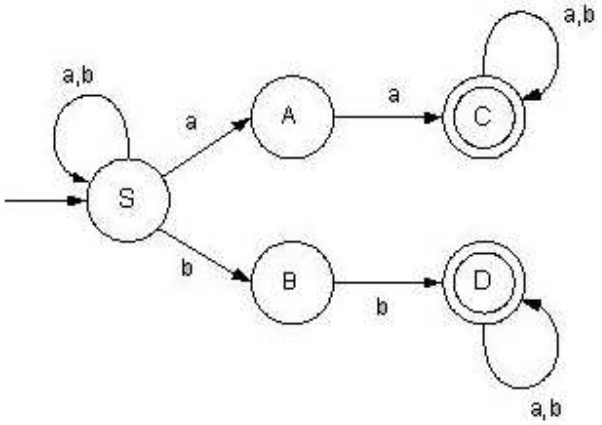
$C \to Bc \mid Cc$

$B \to Ab \mid Bb$

$A \to Aa \mid a$

A generálás során azt néztük, hogyan lehet egy adott állapotba jutni. Például E állapotba juthatunk B állapotból c karakter hatására, valamint ugyancsak E állapotba jutunk C állapotból c karakter érkezése után. Az $A \to a$ helyettesítési szabály annak felel meg, hogy kiindulási állapotból jutunk A állapotba a karakter hatására.

Most nézzünk egy kicsivel bonyolultabb példát! Határozzuk meg az alábbi automatához tartozó jobbreguláris nyelvtant!



Az ismertetett átírási szabályokat alkalmazva a jobbreguláris grammatika a következő:

$S \to aS \mid bS \mid aA \mid bB$

$A \to aC \mid a$

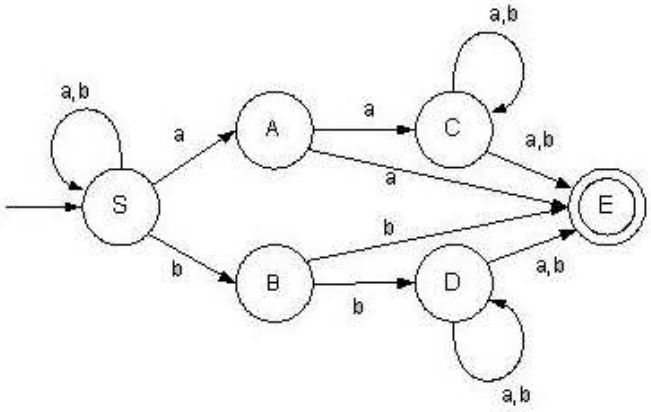
$B \to bD \mid b$

$C \to aC \mid bC \mid a \mid b$

$D \to aD \mid bD \mid a \mid b$

A szabályok írásánál azt követtük, hogyan lehet egy állapotból kijutni. Például a C állapotból a vagy b karakter olvasása után C állapotba jutunk, mely egyben végállapot is.

Most nézzük meg, hogyan kaphatunk ebből az automatából balreguláris nyelvtant! A balreguláris grammatika származtatására alakítsuk át az automatát úgy, hogy csak egyetlen elfogadó állapotot tartalmazzon.



Az átírási szabályok megfordításával most már könnyen megszerkeszthetjük a balreguláris nyelvtan helyettesítési szabályait. A szabályok felírásánál azt vizsgáljuk, hogy egy adott állapotba hogyan lehet eljutni a többi állapotból. A balreguláris nyelvtan:

$E \rightarrow Aa \mid Bb \mid Ca \mid Cb \mid Da \mid Db$

$C \rightarrow Aa \mid Ca \mid Cb$

$D \rightarrow Bb \mid Da \mid Db$

$A \rightarrow Sa \mid a$

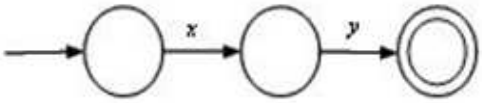
$B \rightarrow Sb \mid b$

$S \rightarrow Sa \mid Sb \mid a \mid b$

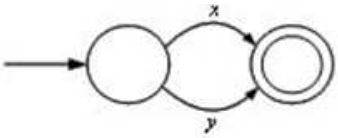
1.2.2.2. Reguláris kifejezés átalakítása automatába

Mint azt a korábbi fejezetekben láttuk, a reguláris kifejezések alaphalmazokból és a köztük értelmezett műveletekből állnak. Automatába való átrajzoláskor a reguláris kifejezést a benne szereplő műveleteknek megfelelően részekre szedjük, ezeket külön-külön ábrázoljuk, majd pedig ezeket összeillesztjük. Nézzük meg néhány elemi reguláris kifejezésnek az átalakítását:

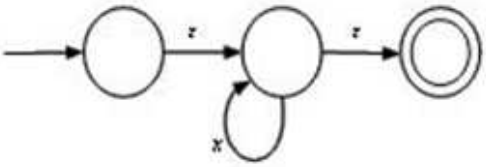
1. xy



2. $x + y$



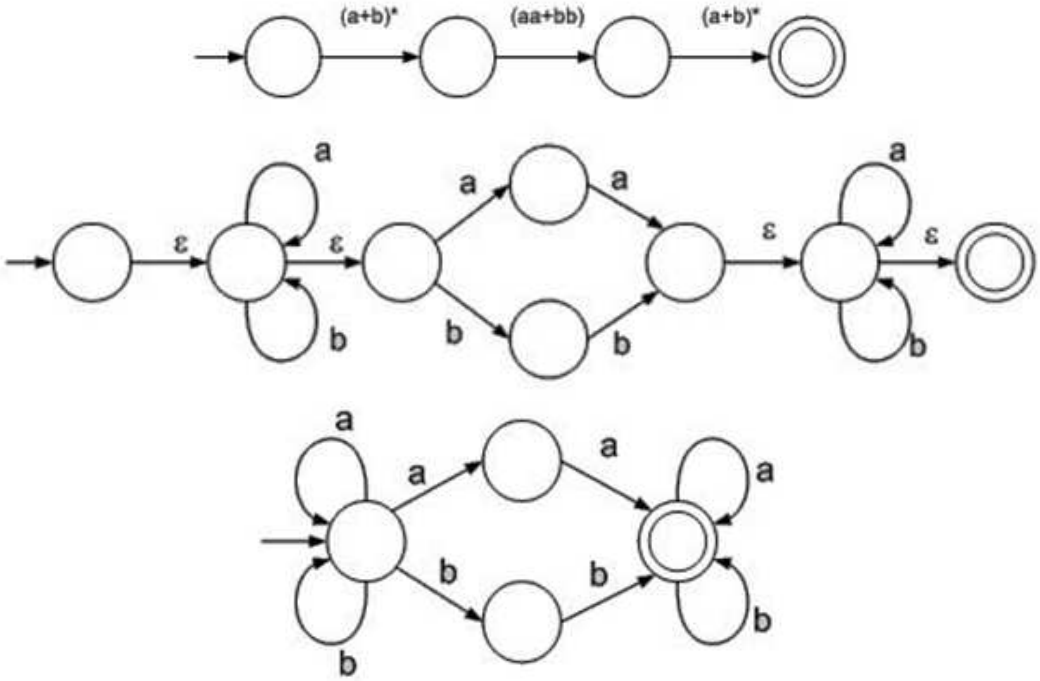
3. x^*



Példaként nézzük, hogyan képezhető az

$R = (a + b)^*(aa + bb)(a + b)^*$

reguláris kifejezésnek megfelelő automata! A kifejezést három részre bontjuk: $(a + b)^*, (aa + bb)$ és $(a + b)^*$. Ezeket külön-külön átalakítjuk, és ε szabályokkal összekötjük. Az átalakítás folyamatát a következő ábrán követhetjük nyomon:



Az automata két állapota összevonható, ha mindkettőből ugyanazokat a szövegeket fogadja el az automata. Az állapotok összevonásával a következő fejezetben részletesen foglalkozunk. Az ε szabályok, mint az ábrán is látszik, jelen esetben elhagyhatók, és az ε szabállyal összekötött állapotok összevonhatók. Az előállított

automata nemdeterminisztikus. Az induló állapotból mind az a, mind pedig a b karakter hatására két állapotba is átmehetünk.

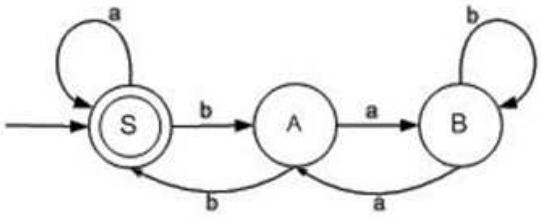
1.2.2.3. Az automaták átalakítása

Az előző fejezetekben láttuk, hogy egy reguláris nyelvhez számos automata készíthető. Ebben a fejezetben azzal foglalkozunk, hogy hogyan tudjuk eldönteni két automatáról, hogy ekvivalensek-e. Megnézzük, hogy nemdeterminisztikus automaták, valamint az epszilon mozgást is megengedő nemdeterminisztikus automatákhoz hogyan készíthetünk vele ekvivalens determinisztikus automatát, továbbá a determinisztikus automaták közül hogyan találhatjuk meg a legkevesebb állapottal rendelkező automata variánst.

Az automaták ekvivalenciája

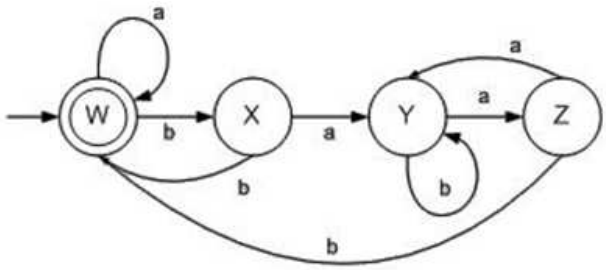
Két automata ekvivalens, ha ugyanazokat a szövegeket fogadja el mondatként. Következőkben egy példán keresztül megmutatjuk, hogyan lehet eldönteni az ekvivalencia kérdését. Kövessük az alábbi két automata mozgását! Az első automata legyen:

	a	b
→**S**	S	A
A	B	S
B	A	B



A második automata pedig:

	a	b
→**W**	W	X
X	Y	W
Y	Z	Y
Z	Y	W



Az első automatának három, a másodiknak négy állapota van.

Készítsünk olyan összehasonlító táblázatot, amely segítségével az automaták mozgása szinkron módon követhető! Az alábbi táblázatban a két automata állapotátmeneteit tüntettük fel a kezdő állapotokból kiindulva.

	a	b
→(**S**,**W**)	(S,W)	(A,X)
(A,X)	(B,Y)	(S,W)
(B,Y)	(A,Z)	(B,Y)
(A,Z)	(B,Y)	(S,W)

A két automata akkor ekvivalens, ha mozgásuk során mindig ugyanakkor jutnak elfogadó, illetve nem elfogadó állapotba. Az összehasonlító táblázat első oszlopa alapján látható, hogy ez a feltétel ebben az esetben teljesül, vagyis a két automata ekvivalens.

Nemdeterminisztikus automata átalakítása

Egy determinisztikus automata esetén az állapotok közötti mozgási szabályok egyértelműek. Egy adott állapotból egy adott karakter vételekor csak egy lehetséges állapotátmenet megengedett. Nemdeterminisztikus esetben a mozgási szabályok nem egyértelműek. Egy adott állapotból egy adott karakter érkezése esetén az automata több állapotba is átkerülhet.

Minden véges nemdeterminisztikus $M(Q, \Sigma, \delta, q_0, F)$ automatához készíthető vele ekvivalens $M(2^Q, \Sigma, \delta', p_0, F')$ determinisztikus automata. A két automata karakterkészlete azonos. A determinisztikus automata állapotainak halmaza a nemdeterminisztikus automata állapothalmazának hatványhalmaza. Legyen a Q

halmaz:

$Q = \{A, B, C\}$.

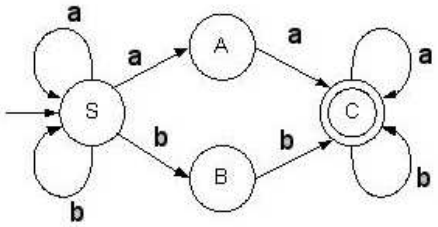
A 2^Q hatványhalmaz definíció szerint:

$2^Q = \{\{\}, \{A\}, \{B\}, \{C\}, \{A, B\}, \{A, C\}, \{B, C\}, \{A, B, C\}\}$.

A hatványhalmaz elemeinek számossága a fenti példában $2^3 = 8$.

A determinisztikus automata állapotait jelöljük p-vel ($p \in 2^Q$). Az automata induló állapota $p_0 = \{q_0\}$. Az elfogadó állapotok halmaza azon p_i állapotokat tartalmazza, amelyekben található olyan állapot, amely a nemdeterminisztikus automatában elfogadó állapot. A mozgási szabályok pedig oly módon szerkesztendők, hogy akkor létezik a $\delta'(p_i, a) = p_j$ átmenet, ha p_i állapotot alkotó állapotok legalább egyikéből átmehetünk a p_j állapotot alkotó állapotok legalább egyikébe $a \in \Sigma$ karakter hatására. Egy példán mutatjuk be a mozgási szabályok generálásának a lépéseit. A példában nem vizsgáljuk valamennyi állapot párosítását, csak azokat az állapotokat határozzuk meg, amelyek a kiinduló állapotból elérhetőek.

Példa Készítsük el az $(a + b)^*(aa + bb)(a + b)^*$ reguláris kifejezést értelmező automatát!



Az automata táblázatos formában:

	a	b
→S	S,A	S,B
A	C	-
B	-	C
C	C	C

A táblázat alapján látható, hogy az automata nemdeterminisztikus, hiszen a táblázatban található olyan mező, ahol egynél több állapot is szerepel.

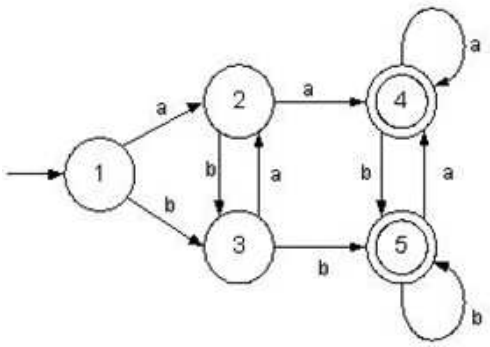
Képezzük a fenti táblázat alapján a vele ekvivalens determinisztikus automatát! Ezt úgy készíthetjük el, hogy kiindulunk az $\{S\}$ kezdő állapotból és megnézzük,

hogy ebből a halmazból a, valamint b karakterek hatására milyen állapotokba juthatunk el. A táblázat első sorában azt láthatjuk, hogy $\{S\}$-ből a hatására $\{S, A\}$-ba juthatunk, hasonlóan b-vel $\{S, B\}$-be. Így az állapotok új halmazait kapjuk, amelyeket a következő sorokba írunk, mint új állapothalmazokat. Ismét megnézzük, hogy ezekből hova juthatunk el. Ha az állapotbővítések során nem kapunk új állapothalmazt, akkor kész az elemző táblánk.

	a	b
$\rightarrow \{S\}$	$\{S, A\}$	$\{S, B\}$
$\{S, A\}$	$\{S, A, C\}$	$\{S, B\}$
$\{S, B\}$	$\{S, A\}$	$\{S, B, C\}$
{S,A,C}	$\{S, A, C\}$	$\{S, B, C\}$
{S,B,C}	$\{S, A, C\}$	$\{S, B, C\}$

Az automata állapothalmazait számmal jelölve az alábbi automata adódik:

	a	b
$\rightarrow$1	2	3
2	4	3
3	2	5
4	4	5
5	4	5



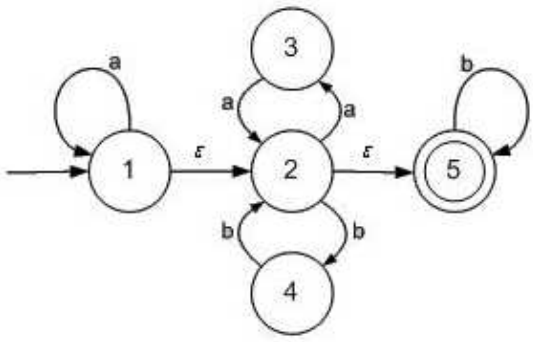
Ezzel az eljárással minden nemdeterminisztikus automata determinisztikussá tehető.

A következő példán keresztül arra keresünk választ, hogyan lehet az epszilon mozgást is megengedő nemdeterminisztikus automatával ekvivalens determinisztikus automatát előállítani. Készítsük el az

$R = a^*(aa + bb)^*b^*$

reguláris kifejezést feldolgozó automata állapot-átmeneti táblázatát! A reguláris

kifejezésből közvetlenül felrajzolható az állapot-átmeneti gráf.



Minden epszilon szabályokat tartalmazó automata mechanikus lépések sorozataként determinisztikus automatává alakítható. Először állítsuk elő a állapot-átmeneti gráfnak megfelelő állapot-átmeneti táblázatot! A fenti epszilon szabályokat is tartalmazó állapot-átmeneti gráfnak egy táblázatos leírása:

	a	b	ε
$\rightarrow 1$	1		2
2	3	4	5
3	2		
4		2	
5		5	

A korábban használt állapotátmeneti táblázatot kiegészítettük egy plusz oszloppal. Ez az oszlop tartalmazza azokat az állapotokat, amelyekbe az adott sornak megfelelő állapotból epszilon állapotátmeneti nyíl vezet a kiinduló állapot-átmeneti gráf alapján. Következő lépésként a táblázat utolsó oszlopa alapján határozzuk meg, hogy az egyes állapotokból epszilon mozgásokkal milyen állapotokba tudunk lépni! A táblázat egyes állapotait bővítsük az adott állapotból epszilon mozgással elérhető állapotokkal! Például az *1* állapotból epszilon mozgással a *2* állapotba, a *2* állapotól pedig az *5* állapotba is el tudunk jutni. Így a táblázatban előforduló *1* állapotot bővítsük a *2* és *5* állapotokkal! A *2* állapotból epszilon mozgással az *5* állapotba tudunk lépni. Így minden *2* előfordulás bővítendő az *5* állapottal. A módosított táblázat:

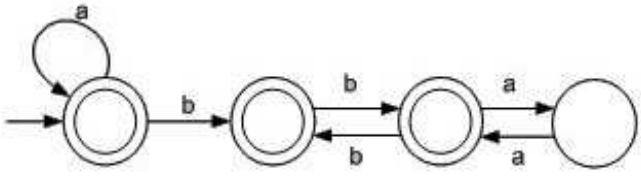
	a	b
1	1, 2, 5	
2	3	4
3	2, 5	
4		2, 5
5		5

Továbbá tekintsük kezdőállapotnak az eredeti automata kezdőállapotának kibővítését azokkal az állapotokkal, amelyek a kezdőállapotból epszilon mozgásokkal elérhetőek. Ez a fenti példánál: *{1,2,5}*.

Az új kezdőállapotból a fenti táblázat segítségével kövessük nyomon az automata lehetséges mozgásait! Például a *{1,2,5}* induló állapotból a táblázat *1*, *2* és *5* sora alapján az *a* karakter hatására a *{1,2,3,5}*, a *b* karakter hatására pedig a {4,5} hatványállapotba léphetünk.

	a	b
→ **{1,2,5}**	{1, 2, 3, 5}	{4, 5}
{1,2,3,5}	{1, 2, 3, 5}	{4, 5}
{4,5}	{}	{2, 5}
{2,5}	{3}	{4, 5}
{3}	{2, 5}	{}
{}	{}	{}

A táblázatban vastag szedéssel jelöltük az elfogadó állapotokat. A táblázat első két állapotának összevonásával az alábbi determinisztikus minimálautomatát kapjuk:



Az állapotok összevonásának kérdésével a következő fejezetben foglalkozunk.

Minimálautomata készítése

Minden állapotból az automata egy adott szöveghalmazt tud elfogadni. Két állapot összevonható, ha mindkettőből azonos szöveget fogad el az automata. Más

formában megfogalmazva, p és q állapotok összevonhatók, ha mindig ugyanakkor kerül az automata elfogadó vagy elutasító állapotba a p és q állapotokból kiinduló mozgások során. A két állapot összevonhatóságát az automaták ekvivalenciájánál bemutatott eljárással ellenőrizhetjük.

Legyen egy automata állapot-átmeneti táblázata az alábbi:

	a	b
→1	2	6
2	7	3
3	1	3
4	3	7
5	8	6
6	3	7
7	7	5
8	7	3

Döntsük el, hogy az automata 1 és 5 állapotai összevonhatók-e! Az állapotok ekvivalenciájánál látott módon járjuk el! Induljunk ki az 1 és 5 állapotokból és kövessük az automata mozgását:

	a	b
(1,5)	(2,8)	(6,6)
(2,8)	(7,7)	(3,3)
...	...	...

A táblázatnak csak azon részét készítettük el, amely alapján az összevonás eldönthető. A második lépéstől kezdve a táblázatban csak azonos állapotpárok szerepelnek. A táblázat alapján könnyen beláthatjuk, hogy a két állapot összevonható, mivel mindkét állapotból kiindulva, ugyanakkor jutunk elfogadó, valamint nem elfogadó állapotokba.

Most nézzük meg, hogy az 1 és 7 állapotok összevonhatók-e!

	a	b
(1,7)	(2,7)	(6,5)
(2,7)	(7,7)	(3,5)
(3,5)!!!	...	...

A fenti táblázattöredékből látszik, hogy az ab karaktersorozat hatására az 1 álla-

potból elfogadó, a 7 állapotból pedig nem elfogadó állapotba jutunk. Tehát a két állapot nem vonható össze.

Mivel az állapotpárok száma szignifikáns lehet, minden lehetséges párosítás megvizsgálására szisztematikus módszereket dolgoztak ki. Az állapotminimalizálás két szokásos eljárása az ún. partíciófinomítás és az állapotok egyesítésén alapuló (un. lépcsős tábla) eljárás. Most csak a partíciófinomítás módszerét ismertetjük.

Egy determinisztikus automata minimálautomatájának előállítási lépései az alábbiak:

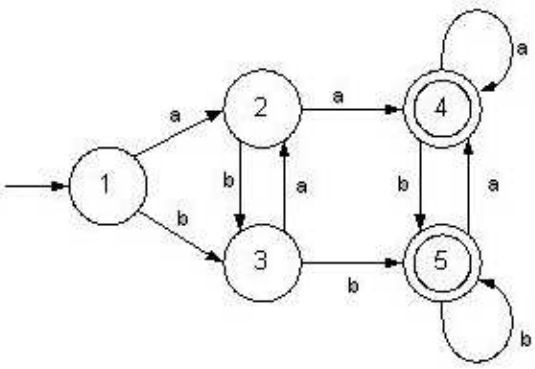
1. Töröljük azokat az állapotokat, ahová nem lehet eljutni a kezdőállapotból.
2. Az állapotok ekvivalenciaosztályokba sorolásának finomításával összevonjuk az ekvivalens állapotokat.

Az algoritmust néhány példán keresztül mutatjuk be.

1. példa

Az $(a+b)^*(aa+bb)(a+b)^*$ reguláris kifejezést feldolgozó automata állapot-átmeneti táblázata

	a	b
→1	2	3
2	4	3
3	2	5
4	4	5
5	4	5



Az automata *4* és *5* állapotai az elfogadó állapotok.

Az állapotokat kiinduláskor két csoportba sorolhatjuk. Az egyik az elfogadó állapotok halmaza, a másik pedig az összes többi állapot. Ezeket az állapotokat biztos nem lehet összevonni. Jelöljük ezeket A-val és B-vel:

$A=\{1\ 2\ 3\}\quad B=\{4\ 5\}$.

Az elemzőtáblát bontsuk fel két részre. Az egyik az A, a másik pedig B halmaznak a további elemzésére fog szolgálni.

Az A halmazhoz tartozó résztáblázat:

A	a	b
1	2	3
2	4	3
3	2	5

A B halmazhoz tartozó résztáblázat pedig:

B	a	b
4	4	5
5	4	5

Az elemző táblázatokat egészítsük ki egy új résszel! Ebben a részben az állapotátmeneteket a bevezetett új állapotosztályokkal írjuk le.

A	a	b	a	b
1	2	3	A	A
2	4	3	B	A
3	2	5	A	B

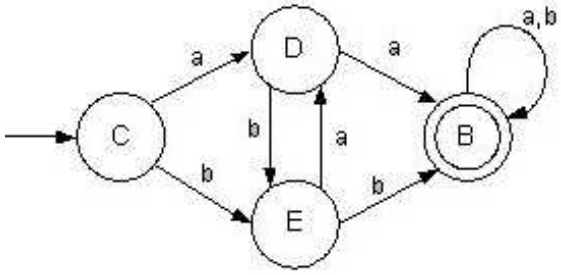
B	a	b	a	b
4	4	5	B	B
5	4	5	B	B

Az A halmaz táblázatában látható, hogy mind a három állapot megkülönböztethető, mivel azonos bemenet esetén más állapotokba mennek. Az új állapotokra vezessük be a $C=\{1\}$, $D=\{2\}$ valamint az $E=\{3\}$ jelölést. A B résztáblázat viszont azt mutatja, hogy a 4 és 5 állapotok nem különböztethetők meg, hiszen azonos bemenetekre azonos állapotba kerülnek. Így ezek az állapotok összevonhatók. Az összevont állapotot jelöljük B-vel, vagyis $B=\{4,5\}$. Az állapot finomításának folyamatának ezzel végére értünk, mivel az állapotok további szétbontása nem módosítja résztáblázatainkat. Így megkaptuk a minimalizált automata állapot-átmeneti táblázatát:

	a	b
→C	D	E
D	B	E
E	D	B
B	B	B

	a	b
→1	2	3
2	(4,5)	3
3	2	(4,5)
(4,5)	(4,5)	(4,5)

A minimálautomatának megfelelő állapot-átmeneti diagram pedig:



2. példa

Legyen egy automata állapot-átmeneti táblázata az alábbi:

	a	b
→1	2	6
2	7	3
3	1	3
4	3	7
5	8	6
6	3	7
7	7	5
8	7	3

Az induló állapot a táblázat szerint az *1*-es állapot. Az elfogadó állapotok halmaza pedig egy állapotból áll, mégpedig a *3* állapotból.

Először ellenőrizzük, hogy minden állapot elérhető-e a kiinduló *1*-es állapotból! Építsük fel az alábbi állapothalmaz szekvenciát! Az induló halmazba tegyük be a kiinduló állapotot:

$Q_0 = \{1\}$.

Az *1*-es állapotból a táblázat szerint elérhető a *2*-es és a *6*-os állapot. Bővítsük a kiinduló halmazt ezekkel az állapotokkal:

$Q_1 = \{1\ 2\ 6\}$.

A *2*-es állapotból a *7*-es és a *3*-as, a *6*-os állapotból a *3*-es és a *7*-es állapot érthető el. Így az elérhető állapotok halmazát bővíthetjük:

$Q_2 = \{1\ 2\ 3\ 6\ 7\}$.

A *3*-as és a *7*-es állapotból a halmazban még nem előforduló *5*-ös állapot is elérhető. Így:

$Q_3 = \{1\ 2\ 3\ 5\ 6\ 7\}$.

A következő lépésben a halmazt tovább kell bővíteni a *8*-as állapottal:

$Q_4 = \{1\ 2\ 3\ 5\ 6\ 7\ 8\}$.

A *8*-as állapottal való bővítés után már nem bővül a halmazunk. Ez azt jelenti, hogy a *4*-es állapotba nem juthatunk el a kezdőállapotból, így törölhető a táblázatból.

	a	b
→1	2	6
2	7	3
3	1	3
5	8	6
6	3	7
7	7	5
8	7	3

Ezek után foglalkozzunk a lehetséges állapotösszevonások felderítésével! Az állapotok két csoportba oszthatók. Vannak a nem elfogadó állapotok és az elfogadó állapotok. Jelöljük ezeket A-val és B-vel:

A={1 2 5 6 7 8} B={3}.

Ez alapján az elemzőtáblázat két részre bontható. Az A halmazhoz tartozó rész:

A	a	b	a	b
1	2	6	A	A
2	7	3	A	B
5	8	6	A	A
6	3	7	B	A
7	7	5	A	A
8	7	3	A	B

valamint a B halmazhoz tartozó rész:

B	a	b	a	b
3	1	3	A	B

Mivel a B halmaz egyetlen állapotból áll, így az elfogadó állapotok halmazában nincs lehetőség az állapotok összevonására. Nézzük az A táblázatrészt! Mivel az *1*, *5* és *7* állapotok ugyanazt az átmenetet tartalmazzák a táblázat jobb oldali részében, potenciálisan összevonható állapotok lehetnek. Hasonló indok alapján a *2* és a *8* állapotok is esetlegesen összevonhatóak. A *6* állapot különbözik a többitől. Ilyen átmenet nem található a táblázat más részében. Ezek alapján a táblázat három részre bontható az új állapotok bevezetése után:

C={1 5 7}

C	a	b	a	b
1	2	6	D	E
5	8	6	D	E
7	7	5	C	C

D={2 8}

D	a	b	a	b
2	7	3	C	B
8	7	3	C	B

E={6}

E	a	b	a	b
6	3	7	B	C

A táblázatok alapján látható, hogy az E halmaz tovább nem bontható, hiszen egyetlen állapotból áll. A D táblázat szerint lehetséges, hogy a *2* és a *8* állapotok összevonhatóak. A C táblázat szerint a C halmaz állapotai nem tekinthetők azonosnak. A C táblázat szerint az *1* és *5* állapotok megkülönböztethetőek a *7* állapottól. Ezért a C táblázatot továbbfinomítjuk, kétfelé bontjuk:

F={1 5}

F	a	b	a	b
1	2	6	D	E
5	8	6	D	E

G={7}

G	a	b	a	b
7	7	5	G	F

D={2 8}

D	a	b	a	b
2	7	3	G	B
8	7	3	G	B

Az F és a D halmazba tartozó állapotok a táblázatok szerint nem megkülönböztethetőek, így összevonhatóak.

Összegzésképpen megállapítható, hogy az eredetileg 8 állapotot tartalmazó automatában a *4*-es állapot az *1*-es állapotból nem érhető el, továbbá az *1* és az *5* állapotok, valamint a *2* és a *8* állapotok összevonhatók. Így a minimálautomatának öt állapota van. A minimálautomata elemző táblája pedig:

	a	b
→F	D	E
D	G	B
B	F	B
E	B	G
G	G	F

	a	b
→(1,5)	(2,8)	6
(2,8)	7	3
3	(1,5)	3
6	3	7
7	7	(1,5)

1.3. Környezetfüggetlen nyelvek

Mint azt a nyelvtanok Chomsky féle osztályozásánál már említettük, azokat a nyelveket nevezzük környezetfüggetlen nyelveknek, amelyeket a 2. nyelvosztályba tartozó grammatikák generálnak. Az ilyen nyelvtanokban csak $A \rightarrow \alpha$ alakú levezetési szabályok lehetnek, ahol α tetszőleges terminálisokból és nemterminálisokból álló sorozat lehet. A helyettesítési szabályok formája alapján látható, hogy a környezetfüggetlen nyelvek kevésbé kötöttek, mint a reguláris nyelvek. Minden reguláris nyelv környezetfüggetlen nyelv is.

1.3.1. A környezetfüggetlen nyelvek nyelvtana

Vizsgáljuk az alábbi környezetfüggetlen nyelvtant:

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid a$

A nemterminális szimbólumokat most is nagybetűvel jelöltük, a terminális szimbólumok közé azonban bekerültek az a szimbólum mellett a $+$, $*$, $($ és $)$ szimbólumok is. Annak oka, hogy a terminálisokat most nem csak latin kisbetűkkel jelöljük az, hogy a szimbólumoknak jelentése van. Méghozzá az összeadás, szorzás és zárójelezés operátorai. A mondatszimbólum jele most nem S, ennek is jelentése van. A nemterminális szimbólumok neve is jelentést hordoz. Ebben a példában E - expression, T - term, F - factor. Nyelvtanunk a $+$ és $*$ operátorokat tartalmazó és zárójelezést megengedő aritmetikai kifejezéseket generálja. Természetesen további operátorokkal is bővíthetünk egy ilyen típusú nyelvtant, de mi megelégszünk ennek az egyszerűsített nyelvtannak a vizsgálatával. A fenti nyelvtan segítségével a műveletek közötti precedencia is kezelhető.

Levezetési fa - egyértelmű nyelvtanok

Az előző fejezetben vizsgált nyelvnek eleme az

$a + a * a$

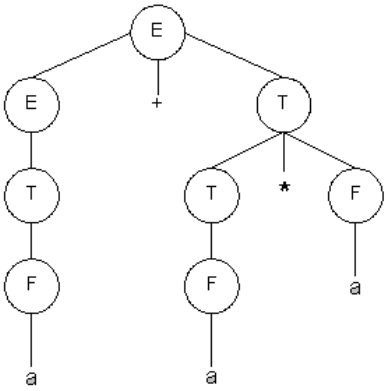
mondat, ami az alábbi levezetéssel igazolható:

$$E \Rightarrow E+T \Rightarrow T+T \Rightarrow F+T \Rightarrow a+T \Rightarrow a+T*F \Rightarrow a+F*F$$
$$\Rightarrow a+a*F \Rightarrow a+a*a$$

Ez tehát a mondatnak egy lehetséges levezetése. Más úton is levezethető ugyanez a mondat. Nézzünk most erre is egy példát:

$$E \Rightarrow E+T \Rightarrow E+T*F \Rightarrow E+T*a \Rightarrow E+F*a \Rightarrow E+a*a$$
$$\Rightarrow T+a*a \Rightarrow F+a*a \Rightarrow a+a*a$$

Az előző példa alapján is látható, hogy egy mondatnak több levezetése is lehet. Felmerülhet a kérdés, hogy okoz-e problémát az a tény, hogy egy mondatnak több levezetése is lehet. Erre a választ a levezetési fák egyértelműsége adja meg. A levezetési fa a levezetés ábrázolása egy rendezett irányított gráffal. Minden csomópontból annyi él és abban a sorrendben fut ki, amennyi az alkalmazott helyettesítési szabály jobb oldalán található szimbólumok száma. A kifutó élek végén az alkalmazott produkciós szabály jobb oldalán található szimbólumok helyezkednek el. A fenti példában mindkét levezetés ugyanarra a levezetési fára vezet:



Az olyan nyelvtanokat, ahol minden mondathoz egy és csakis egy levezetési fa tartozik, egyértelmű nyelvtanoknak nevezzük.

A levezetések között vannak nevezetesek. Ilyen például az a levezetés, amely során mindig a mondatszerű forma legbaloldalibb nemterminális szimbólumát helyettesítjük. Ezt baloldali levezetésnek nevezzük. Ugyanígy jobboldali levezetést is értelmezünk. A fenti példában épp ezt a két levezetést adtuk meg.

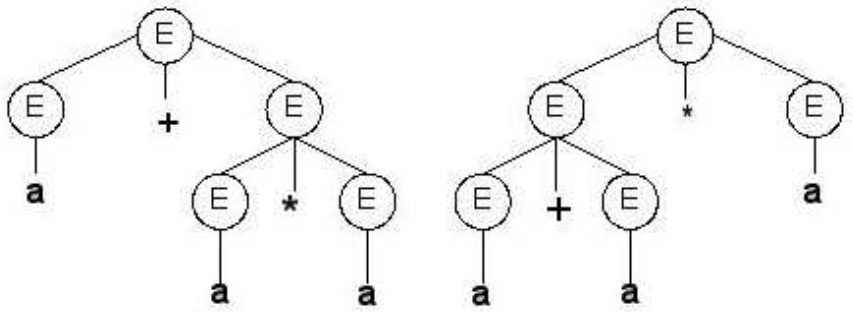
Nézzünk egy másik nyelvtant:

$E \to E + E \mid E * E \mid (E) \mid a$

Ez a nyelvtan ugyanazt a nyelvet generálja, mint a korábban ismertetett nyelvtanunk. Vezessük most le belőle az $a + a * a$ példamondatunkat két módon is:

1. $E \Rightarrow E + E \Rightarrow a + E \Rightarrow a + E * E \Rightarrow a + a * E \Rightarrow a + a * a$
2. $E \Rightarrow E * E \Rightarrow E * a \Rightarrow E + E * a \Rightarrow E + a * a \Rightarrow a + a * a$

Nézzük meg ezen levezetések levezetési fáit:



A két levezetési fa különbözik, tehát nem egyértelmű a nyelvtan. Noha ez a nyelvtan is ugyanazokat a mondatokat generálja mint a korábbi, számítástechnika szempontjából mégis használhatatlan.

A fentiek alapján megállapíthatjuk, hogy egy nyelv több nyelvtannal is generálható. Az egy-, illetve többértelműséget a fenti példában a nyelvtan okozta. Felmerülhet a kérdés, hogy lehet-e a többértelműség nyelvi tulajdonság, illetve hogy ki lehet-e mutatni algoritmikusan egy nyelvtanról, hogy az egyértelmű-e, és ha nem, akkor van-e a nyelvnek egyértelmű nyelvtana? Léteznek olyan nyelvek, amelyről bebizonyítható, hogy nem lehet egyértelmű nyelvtanuk, ebben az esetben a többértelműség nyelvi tulajdonság. Egy nyelvtanról mindig el tudjuk dönteni, hogy egyértelmű-e vagy sem. Azonban a nyelvhez megtalálni az egyértelmű nyelvtanát nehéz feladat, nem algoritmikus probléma.

Vizsgáljuk meg az alábbi nyelvtant az egyértelműség szempontjából! A nyelvtan:

$S \to \mathit{if}\ b\ \mathit{then}\ S\ \mathit{else}\ S$

$S \to \mathit{if}\ b\ \mathit{then}\ S$

$S \to a$

Nyelvtanunk egyetlen nemterminálist tartalmaz:

$N = \{S\}$

A terminálisok halmaza pedig:

$\Sigma = \{$ *if, then, else, a, b*$\}$

Legyen a vizsgálandó szöveg: *if b then if b then a else a*

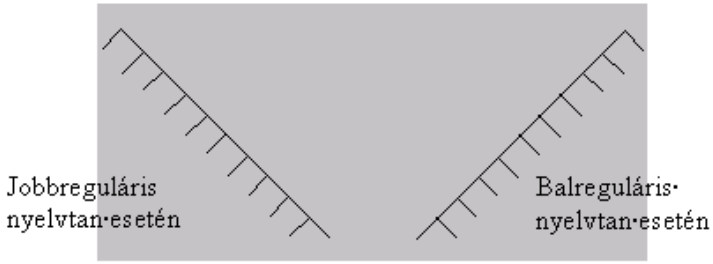
Könnyen belátható, hogy a fenti mondatnak két szignifikánsan különböző levezetése is lehetséges, amely eltérő levezetési fát eredményez:

$S \Rightarrow$ *if b then S else* $S \Rightarrow$ *if b then if b then S else S*
$\Rightarrow$ *if b then if b then a else* $S \Rightarrow$ *if b then if b then a else a*

$S \Rightarrow$ *if b then S* $\Rightarrow$ *if b then if b then S else S*
$\Rightarrow$ *if b then if b then a else* $S \Rightarrow$ *if b then if b then a else a*

Tehát a nyelvtanuk nem egyértelmű. A gyakorlatban a többértelműséget szóbeli kiegészítő korlátozásokkal egyértelművé tehetjük . Kiválasztjuk a lehetséges levezetések közül az egyiket. A fenti példában, ha kikötjük, hogy minden *else* a hozzá legközelebbi *if*-hez tartozik, akkor a két alternatíva közül csak az egyiket engedjük meg. Így a kiegészített nyelvtan alapján történő elemzés már csak egy lehetséges levezetési fához vezet, vagyis a nyelvtanunkat egyértelművé tettük.

A levezetési fát természetesen reguláris nyelveknél is megrajzolhatjuk. Reguláris nyelvtanok esetén a fa nagyon egyszerű lesz, csupán egyetlen szárból áll, amelynek csúcsán és jobb vagy bal oldalán vannak terminálisok, attól függően, hogy jobb- vagy balreguláris nyelvtanról van-e szó. Ez abból is látható, hogy itt minden mondatszerű formában csak egyetlen nemterminális szimbólum van, és így csak ezt bonthatjuk tovább.



Nyelvtanok átalakítása

Egy nyelv több nyelvtan segítségével is leírható. A környezetfüggetlen nyelvtanok $A \to \alpha$ produkciós szabályainak jobb oldalának formájára célszerűségi és esztétikai szempontból további megkötéseket is tehetünk. Két nyelvtan természetesen csak akkor cserélhető fel, ha ekvivalensek, azaz ugyanazokat a mondatokat generálják. A következőkben megnézzük, hogy mikor nevezzük egy nyelvtant jól fésülnek, mi a környezetfüggetlen nyelvek Chomsky, valamint a Greibach normál alakja.

Jól fésült (proper) nyelvtanok

Egy nyelvtant jól fésültnek, vagy másként propernek nevezzük, ha

- nincsenek benne felesleges terminálisok és nemterminálisok,
- nem tartalmaz $A \to \varepsilon$ alakú szabályt, valamint
- nincsenek benne egyszeres szabályok ($A \to B$).

Felesleges terminálisok és nemterminálisok szűrése

Keressük meg először az úgynevezett felesleges szimbólumokat! Egy terminális szimbólum akkor felesleges, ha nincs a nyelvnek olyan mondata, amelyben a szóban forgó szimbólum szerepelne. Egy nemterminális szimbólum akkor felesleges, ha nincs a nyelvnek olyan mondata, amelynek levezetésében előfordulna. A felesleges szimbólumok kiszűrésére algoritmus adható, amely összegyűjti a nem felesleges szimbólumokat. A szűrés két irányba végezhető.

Szűrés alulról felfelé

Egy adott nyelvtan esetén képezzünk az alábbi halmazsorozatot a

$B_0 = \Sigma$

terminális szimbólumokat tartalmazó kiinduló halmazból kiindulva:

$B_{i+1} = B_i \cup \{A \mid A \to \alpha, \alpha \in B_i^*\}$

A fenti szabályok szerint a következő halmazt úgy kapjuk az előző halmazból, hogy az előző halmazhoz hozzávesszük azokat a nemterminális szimbólumokat,

amelyekhez található olyan levezetési szabály, ahol a jobb oldalon minden szimbólum az előző halmaz eleme. Amennyiben egy lépés alkalmával nem találunk új elemet, akkor a sorozat képzését abbahagyhatjuk. Legyen B az így kapott legnagyobb halmaz. Így B csak azok a nemterminális szimbólumokat tartalmazza a karakterkészleten túl, amelyek a terminálisokból felépíthetők. Ezzel alulról felfelé fésültük végig a grammatikát. A kimaradó nemterminális szimbólumokat elhagyhatjuk.

Példa

Legyen a grammatika:

$S \to a \mid A$

$A \to Ab$

$B \to b$

A nyelvtant alulról felfelé megvizsgálva:

$B_0 = \{a, b\}$

$B_1 = \{a, b, S, B\}$

$B_2 = B_1$

A fenti halmazokból látható, hogy A nemterminális kiesett. Így ezt a szimbólumot és vele együtt minden olyan helyettesítési szabályt kihagyhatunk a nyelvtanból, amelyben az A nemterminális szerepel. Így az alábbi nyelvtant kapjuk:

$S \to a$

$B \to b$

Szűrés felülről lefelé

Vegyünk ismét egy halmazsorozatot a

$T_0 = \{S\}$

halmazból kiindulva az alábbi képzési szabállyal:

$T_{i+1} = T_i \cup \{X \mid A \to \alpha X \beta, A \in T_i\}$.

A képzési szabály szerint a T_{i+1} halmaz úgy képezhető, hogy a halmazhoz hozzá kell venni azon helyettesítési szabályok jobboldalán található szimbólumokat, amely szabályok baloldala az eredeti halmazhoz tartozó nemterminális szimbólum. A képzést ezúttal is abbahagyhatjuk, ha egy lépésben már nem találunk új elemet. Legyen a legnagyobb elemszámú halmaz T. T azokat és csakis azo-

kat a terminális és nemterminális szimbólumokat tartalmazza majd, amelyek egy, a mondatszimbólumból kiinduló levezetés mondatszerű formáiban szerepelhetnek. Ezzel az algoritmussal felülről lefelé vizsgáltuk át a grammatika szabályait.

Példa

Legyen a grammatika:

$S \rightarrow a \mid A$

$A \rightarrow Ab$

$B \rightarrow b$

A nyelvtant felülről lefelé megvizsgálva:

$T_0 = \{S\}$

$T_1 = \{S, a, A\}$

$T_2 = \{S, a, A, b\}$

$T_3 = T_2$

A szűrés azt mutatja, hogy a B nemterminális a levezetésekhez felesleges. Így a nyelvtan egyszerűsíthető:

$S \rightarrow a \mid A$

$A \rightarrow Ab$

A fenti szűrési példák azt mutatják, hogy egyik szűrés sem képes egy lépésben valamennyi felesleges szimbólumot a nyelvtanból törölni. A szűrt nyelvtanuk újraszűrésével már valamennyi felesleges terminális és nemterminális kiszedhető. Az egyszerűsített nyelvtan ebben az esetben:

$S \rightarrow a$

Végeredményben tehát az egyszerűsített nyelvtan egyetlen helyettesítési szabályt tartalmaz.

ε szabálymentesítés

A következőkben az $A \rightarrow \varepsilon$ alakú, úgynevezett ε szabályok kiküszöbölésére adunk eljárást. Először vizsgáljuk meg, hogy egy nyelvtanból az ε szabály egyáltalán kiiktatható-e! Ha a nyelvünknek eleme az üres jelsorozat, akkor a nyelvtanból

nem szedhető ki az ε szabály. Ha azonban a nyelvnek nem eleme az üres szöveg, akkor a nyelvtan ε mentesíthető.

Ha a nyelvünknek eleme ε, akkor a nyelv előállítható két nyelv uniójaként. Az egyik nyelv csak az üres szöveget tartalmazza, ugyanakkor a másik nyelv már előállítható ε szabálymentesen. Az első nyelv az $S' \to \varepsilon$ szabállyal generálható, a második pedig legyen S" szimbólumból származtatható, amely nem tartalmaz ε szabályokat. Az eredeti nyelv a fenti nyelvek uniója, az $S \to S' \mid S''$ szabályokkal állítható elő.

Nézzük meg, hogyan győződhetünk meg arról, hogy az üres szöveg része-e a nyelvünknek! A $B_0 = \{\varepsilon\}$ halmazból kiindulva alulról felfelé haladó fésüléssel határozzuk meg, hogy milyen nemterminálisok helyett állhat üres szöveg. Ha a halmazok származtatása során eljutunk a mondatszimbólumig, akkor a nyelvünknek része az ε. Ebben az esetben az ε szabályok nem küszöbölhetők ki a nyelvtanból. Ilyenkor új mondatszimbólumok bevezetésével a nyelvet két részre bontjuk. Az egyik résznyelv csak az üres szöveget tartalmazza, a másik pedig már előállítható ε szabálymentesen. Az ε -t tartalmazó szabályokból úgy származtatjuk az újakat, hogy minden lehetséges helyen az összes lehetséges kombinációban az üres szöveget is származtató nemterminális helyére az ε -t is behelyettesítjük. Nézzük egy példát!

Példa

Határozzuk meg az alábbi nyelvtan felbontását csak ε -t tartalmazó és ε szabálymentes nyelvtanok segítségével:

$S \to SaSb$

$S \to \varepsilon$

Fésüljük végig a nyelvtan a $B_0 = \{\varepsilon\}$ halmazból indulva alulról felfelé!

$B_0 = \{\varepsilon\}$

$B_1 = \{\varepsilon, S\}$

$B_2 = B_1$

A fésülés eredményeként látható, hogy a fenti nyelvtan által generált nyelvnek eleme az üres szöveg. Bontsuk fel nyelvünket két részre:

$S \to S' \mid S''$,

ahol

$S' \to \varepsilon$

és

$S'' \to S''aS''b \mid aS''b \mid S''ab \mid ab$.

Az $S \to SaSb$ szabályból az összes lehetséges ε helyettesítések eredményeként négy szabályt kaptunk.

Egyszeres szabályok szűrése

Végül az egyszeres szabályok kiküszöbölésének módját mutatjuk meg. Egyszeres szabálynak nevezzük az $A \to B$ alakú szabályokat. Számítástechnikai szempontból az egyszeres szabályok jelenléte csak akkor zavaró, ha a levezetés hurkot eredményez. Például:

$A \to B \qquad B \to C \qquad C \to A$

Ilyenkor egy mondat levezetése tetszőlegesen hosszúra nyújtható, ami nyilván programhibához vezethet. Ezt a veszélyt legegyszerűbben úgy lehet elhárítani, hogy eltűntetjük az egyszeres szabályokat. Kiküszöbölésük algoritmusa az alábbi.

Vegyük a nyelvtan egyszeres szabályait, és a továbbiakban azt a nyelvtant vizsgáljuk, amely ezekből a szabályokból áll. Vegyük a szabályokban szereplő összes nemterminálist. Alkalmazzuk a felülről lefelé irányuló fésülést úgy, hogy kiinduló halmazként rendre egy-egy ilyen nemterminálist választunk. Az egyes fésülések eredményeként kapott T záró halmazok azokból a nemterminálisokból állnak, amelyek az éppen kiindulásul választott nemterminálisból az egyszeres szabályokkal elérhetők.

Ennek megfelelően a nyelvtant olyan új szabályokkal kell kiegészíteni, amelyeknek a baloldala a kiindulásul választott nemterminális, jobb oldaluk pedig megegyezik olyan nem egyszeres szabályok jobboldalaival, amelyeknek a baloldalán a záró halmazban található nemterminálisok szerepelnek. A kiegészítéseknek köszönhetően az eredeti nyelvtanból az egyszeres szabályok már elhagyhatók.

Példa

Küszöböljük ki az alábbi nyelvtanból az egyszeres szabályokat:

$E \to E + T \mid T$

$T \to T * F \mid F$

$F \to (E) \mid a$

A nyelvtanban két egyszeres szabály található:

$E \to T \qquad T \to F$

A nemterminálisokból kiindulva felülről lefelé szűrünk. A kiinduló halmaz a vizsgált nemterminális. Az induló halmazt folyamatosan bővítjük a szabály jobb oldalán található egyszeres szabályt képező nemterminálisokkal. A nemterminálisok záróhalmazai ennél a példánál:

$T_E = \{E, T, F\}$

$T_T = \{T, F\}$

$T_F = \{F\}$

A záróhalmazok alapján a következő bővítéseket kell elvégezni:

$E \to T * F$

$E \to (E) \mid a$

$T \to (E) \mid a$

Így az egyszeres szabályokat már nem tartalmazó nyelvtan

$E \to E + T \mid T * F \mid (E) \mid a$

$T \to T * F \mid (E) \mid a$

$F \to (E) \mid a$

produkciós szabályokkal rendelkezik.

Az olyan nyelvtant, amely mentes a felesleges szimbólumoktól (terminálisoktól és nemterminálisoktól), nem vagy csak az ismert korlátozásokkal tartalmaz ε szabályt, végül nincs egyszeres szabálya *proper* vagy más néven *jól fésült nyelvtannak* nevezzük.

A felesleges szimbólumok elhagyása mindig, de az ε szabályok, illetve az egyszeres szabályok elhagyása nem mindig teszi átláthatóbbá és egyszerűbbé a nyelvtant, ezért is szoktuk megtűrni őket a nyelvtanainkban.

Ebben a fejezetben megvizsgáltuk, hogyan lehet egy nyelvtanból proper (jólfésült) nyelvtant formálni. A következő fejezetekben a helyettesítési szabályok formájára is megkötéseket teszünk. Ezeket az egységesített formájú nyelvtani szabályokat tartalmazó nyelvtanokat a nyelv normál alakjainak nevezzük.

Chomsky normál forma (CYK algoritmus)

A Chomsky normál alakban (Chomsky Normal Form - CNF) csak kétféle helyettesítési szabály engedélyezett:

$A \to BC \qquad A \to a$

Az alábbiakban megmutatjuk, hogy egy proper nyelvtannak mindig van CNF megfelelője, valamint azt is megnézzük, hogy az hogyan kapható meg.

Ha a nyelvtannak vannak olyan helyettesítési szabályai, amelyek eleget tesznek a CNF szabályoknak akkor ezeket véglegesnek tekintjük. A többi szabályra pedig az alábbi algoritmust használjuk.

A terminális szimbólumok helyett mindenütt vezessünk be nemterminálosokat, úgy hogy a terminálist egy $\hat{A}$ nemterminálissal helyettesítjük. Ezt használjuk mindenütt a helyett. Ezen kívül vezessünk még be egy $\hat{A} \to a$ szabályt, ami megfelel CNF kritériumainak. Így a Chomsky megkötéseknek eleget nem tevő helyettesítési szabályok alakja csakis a következő lehet:

$A \to A_1 A_2 ... A_n \qquad n > 2$

mert az eredeti nyelvtan proper volt tehát nem lehet benne egyszeres szabály, illetve ha $n = 2$ akkor már véglegesítettük volna. Bontsuk ezt fel két új szabályra:

$A \to A_1 \hat{A}_1 \qquad \hat{A}_1 \to A_2 ... A_n$

Ezzel az előző szabályt helyettesítettük egy olyan szabállyal, ami megfelel CNF kritériumainak ($A \to A_1 \hat{A}_1$), illetve egy az eredetivel azonos alakú, eggyel rövidebb szabállyal. Ha ezt az algoritmust tovább folytatjuk, akkor előbb-utóbb a jobb oldali szabály is két nemterminálisból fog állni. Megjegyezzük, hogy az $S \to \varepsilon$ szabályt a CNF is megengedi.

Példa: Hozzuk Chomsky féle normál alakra a következő nyelvtant:

$S \to aSb \quad S \to ab.$

A terminális szimbólumokat átírva az alábbi nyelvtant kapjuk:

$S \to \hat{A}S\hat{B} \quad S \to \hat{A}\hat{B}$

$\hat{A} \to a \qquad \hat{B} \to b$

Most már csak az első szabályt kell tovább alakítanunk, aminek eredményeként a következő szabályokat kapjuk:

$S \to \hat{A}C \quad C \to S\hat{B}$

Ezzel a nyelvtan Chomsky féle normál alakban:

$S \to \hat{A}\hat{B}$
$S \to \hat{A}C$
$C \to S\hat{B}$
$\hat{A} \to a$
$\hat{B} \to b$

Minden környezetfüggetlen nyelv elemzése a Chomsky féle normál alakból kiindulva elvégezhető. A **Cocke-Younger-Kasami (CYK) algoritmus** a CNF formában adott nyelvtan alapján elemez. A módszer alapötlete, hogy minden lehetséges kettes csoportosítást végezzük el a szöveget alkotó részszövegek között, s vizsgáljuk meg, hogy lentről felfelé haladva eljuthatunk-e a mondatszimbólumig.

A Cocke-Younger-Kasami módszer az alábbi algoritmikus lépesek sorozataként dönti el a tartalmazás kérdését:

- *A bemeneti szöveg karaktereit jelöljük a_i -vel. A szöveg legyen n karakter hosszúságú.*
- *A vizsgált nyelvtan tetszőleges terminális és nemterminális szimbólumát jelöljük R_i-vel. Számuk legyen r. A kezdő szimbólum pedig legyen R_1.*
- *Legyen P[n,n,r] egy Boole típusú tömb. Indulásnál valamennyi elem értéke legyen false.*
- *for i=1 to n*
  for minden $R_j \to a_i$ alakú produkciós szabály esetén
  P[i,1,j] = true.
- *for i=2 to n*
  for j=1 to n-i+1
  for k=1 to i-1
  for minden $R_A \to R_B R_C$ alakú produkciós szabály esetén
  if P[j,k,B] and P[j+k,i-k,C] then P[j,i,A] = true
- *if P[1,n,1] == true*
  then a szöveg a nyelv egy mondata
  else a szöveg nem eleme a nyelvnek

Az algoritmus komplexitása a sztring hosszának harmadik hatványával ará-

nyos.

Példa: Legyen nyelvtanunk:

$S \to aSa \mid bSb \mid aa \mid bb$

Hozzuk Chomsky normál formára:

$S \to AS_A \quad S \to BS_B \quad S \to AA \quad S \to BB$

$S_A \to SA \quad S_B \to SB$

$A \to a \qquad B \to b$

Elemezzük az

aabbaa

szöveget CYK algoritmussal:

S					
	S_A				
	S				
S_B		S_A			
S		S		S	
A	A	B	B	A	A
a	a	b	b	a	a

A fenti táblázat alapján látható, hogy a kiinduló szövegből a fenti példában csak egyféleképpen lehet eljutni a mondatszimbólumig. Általános esetben azonban a táblázat bármely négyzetében több nemterminális is előfordulhat.

Greibach normál forma (balrekurzió megszűntetése)

A Greibach normál alakban a helyettesítési szabályok engedélyezett alakja:

$A \to a\omega$

ahol ω tetszőleges hosszúságú terminálisokból és nemterminálisokból álló sorozat. Az $S \to \varepsilon$ helyettesítési szabályt a Greibach normál alak is megengedi.

Balrekurzívnak mondunk egy nyelvtant akkor, ha van olyan rekurzivitásért felelős nemterminális, amelyből kiinduló levezetés során újból a legbaloldalibb po-

zícióban jelenik meg ez a nemterminális. Azaz:

$A \Rightarrow ... \Rightarrow A\alpha$

levezetésre juthatunk. A balrekurzió a nyelvtan átalakításával megszűntethető. Az átalakított nyelvtan azonban továbbra is rekurzív marad. A balrekurzió megszűntetésére azért van szükség, mert bizonyos szintaktikus elemző módszerek nem alkalmazhatóak, ha a nyelvtan balrekurzív. A Greibach normál alak előállításával a balrekurzió feloldható.

Nézzük először a következő úgynevezett közvetlen balrekurzió megszűntetését! Legyen egy szabályrendszer az alábbi:

$A \rightarrow Ab \mid a$

Az A nyelvtani szimbólum a

$A = ab^*$

szöveget generálja, vagyis

$a \quad ab \quad abb \quad abbb \quad ...$

Keressünk olyan nem balrekurzív szabályokat, amelyek segítségével a fenti mondatok generálhatóak! Például a

$b \quad bb \quad bbb \quad bbbb \quad ...$

mondatok a

$A' \rightarrow b \mid bA'$

szabályokkal generálhatók. A fenti nemterminálisból

$A' = bb^*$

mondatok képezhetők. Ennek felhasználásával:

$a \quad ab \quad abb \quad abbb \quad ...$

mondatok

$A \rightarrow a \mid aA'$

szabályokkal előállíthatók.

A balrekurzió megszűntetésének egy másik alternatívája ε szabályok használatát is megengedi. Egy további lehetséges átalakítás:

$A \rightarrow aA''$

$A'' \rightarrow bA'' \mid \varepsilon$

ahol a bevezetett nemterminális a

$A'' = b^*$

szövegeket állítja elő. Itt jegyzem meg, hogy a Greibach normál forma csak a

mondatszimbólumnál engedi meg az epszilon produkciós szabályt. Így az ε szabály használatával megvalósított balrekurzió mentesített nyelvtan nem Greibach normál alakú.

Nézzük most a közvetlen balrekurzió megszűntetését egy kicsivel általánosabban! Vizsgáljuk az alábbi szabályrendszert:

$A \to A\beta_1 \mid A\beta_2 \mid ... \mid A\beta_n$

$A \to \alpha_1 \mid \alpha_2 \mid ... \mid \alpha_m \qquad m, n > 0$

A balrekurziót ebben az esetben is megoldhatjuk ε szabálymentesen

$A \to \alpha_1 \mid \alpha_2 \mid ... \mid \alpha_m \mid \alpha_1 A' \mid \alpha_2 A' \mid ... \mid \alpha_m A'$

$A' \to \beta_1 \mid \beta_2 \mid ... \mid \beta_n \mid \beta_1 A' \mid \beta_2 A' \mid ... \mid \beta_n A'$

vagy ε szabályt is tartalmazó formában:

$A \to \alpha_1 A'' \mid \alpha_2 A'' \mid ... \mid \alpha_m A''$

$A'' \to \beta_1 A'' \mid \beta_2 A'' \mid ... \mid \beta_n A'' \mid \varepsilon$

A balrekurzió nemcsak egy nemterminálisra korlátozva léphet fel. Lehet, hogy a szabályok

$A \to B\alpha \mid ...$

$B \to C\beta \mid ...$

$C \to A\gamma \mid ...$

jellegű rekurziót tartalmaznak. Ebben az esetben a balrekurzió szintén felszámolható.

Példa: Határozzuk meg az alábbi nyelvtan Greibach normál alakját:

$A \to BA \mid a$

$B \to AB \mid b$

A produkciós szabályok szerint A B-vel, B pedig A-val kezdődhet. Így a szabályrendszer balrekurziót tartalmaz. Az első szabályrendszert a másodikba helyettesítve

$B \to BAB \mid aB \mid b$

szabályrendszert kapjuk. A balrekurzió ebből a szabályrendszerből közvetlenül is látszik. Szüntessük meg a közvetlen balrekurziót! A korábban bemutatott eljárások most is használhatók. ε szabálymentes átalakítással

$B \to b \mid aB \mid bB' \mid aBB'$

$B' \to AB \mid ABB'$

összefüggésekre jutunk. B minden szabálya terminálissal kezdődik, így ez megfelel

a Greibach normál formának. Helyettesítsük B-t A-ba:

$A \to bA \mid aBA \mid bB'A \mid aBB'A \mid a$

A-t pedig B' -be:

$B' \to bAB \mid aBAB \mid bB'AB \mid aBB'AB \mid aB \mid$
$bABB' \mid aBABB' \mid bB'ABB' \mid aBB'ABB' \mid aBB'$

A fenti átalakítások eredményeként mindegyik produkciós szabály terminálissal kezdődik. Összefoglalva a

$A \to BA \mid a$

$B \to AB \mid b$

nyelvtan Greibach normál alakja:

$A \to bA \mid aBA \mid bB'A \mid aBB'A \mid a$

$B \to b \mid aB \mid bB' \mid aBB'$

$B' \to bAB \mid aBAB \mid bB'AB \mid aBB'AB \mid aB \mid$
$bABB' \mid aBABB' \mid bB'ABB' \mid aBB'ABB' \mid aBB'$

Mint látható, az eredeti, egyszerűnek tűnő nyelvtan Greibach normál formában kifejezetten bonyolult lett, de nem tartalmaz balrekurziót.

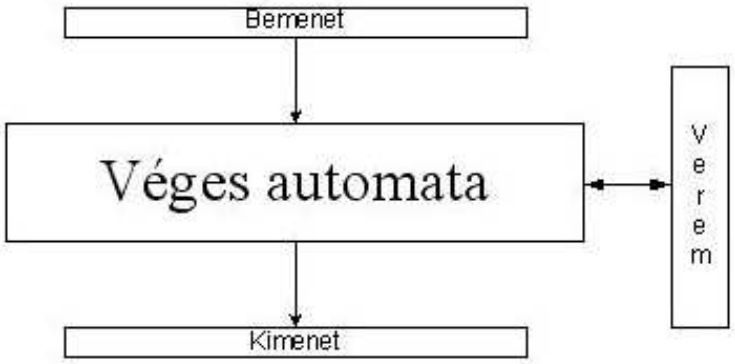
1.3.2. Veremautomata

Minden nyelvosztályhoz tartozik egy automataosztály, amely alkalmas a nyelvosztály bármely nyelvtana által generált nyelv mondatainak felismerésére. A környezetfüggetlen nyelvek esetében ez a veremautomaták osztálya.

A veremautomata és működése

A veremautomata szerkezeti felépítése abban különbözik a véges automatáktól, hogy az kiegészül egy memóriával, amit veremnek neveznek. A verembe beírni, és onnan kiolvasni is tudunk . De ezeket a műveleteket csak a verem tetején lehet elvégezni, azaz csakis a verem tetejére lehet szimbólumokat beírni, és csak a legfelső elemet lehet kiolvasni, kivenni.

A veremautomata (angol terminológiában pushdown automaton) formális leírására egy hetes szolgál:



1.5. ábra. A veremautomata felépítése

$P(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$
ahol

- Q az automata lehetséges állapotainak véges halmaza,
- Σ az elemzendő jelsorozatok karakterkészlete,
- Γ a verem karakterkészlete; azon szimbólumok halmaza, amelyek a veremmemóriában szerepelhetnek,
- δ az automata mozgási szabályainak halmaza,
- q_0 az induló állapot (az automata ebből az állapotból kezdi feldolgozni a szöveget),
- Z_0 a verem kezdeti tartalma és
- F az elfogadó állapotok halmaza.

A veremautomata mozgását az automata állapota, a bemenetről olvasott és a verem tetején található szimbólum alkotta hármas határozza meg. A mozgás hatására az automata új (lehet az eredetivel megegyező is) állapotba kerül, és egy jelsorozatot ír be a verem tetejére. A mozgási szabályok az alábbi formájúak:
$\delta(q, a, b) = (q', \alpha)$
ahol

$q \in Q$ - az automata eredeti állapota,
$a \in \Sigma$ - az inputról beolvasott karakter,
$b \in \Gamma$ - a verem tetején lévő elem, amely a kiolvasáskor el is tűnik onnan,

$q' \in Q$ - ebbe az állapotba kerül az automata,
$\alpha \in \Gamma^*$ - ezt a karaktersorozatot írja a verembe az automata (Lehet nulla hosszú is. A karaktersorozat legbaloldalibb szimbóluma kerül a verem tetejére).

A veremautomatának van egy másik mozgási lehetősége is, ez az ε mozgás. Itt a mozgást csak az automata aktuális állapota és a verem tetején lévő szimbólum szabja meg. Az ε mozgás az alábbi módon írható le:

$\delta(q, \varepsilon, b) = (q', \alpha)$

Egy szöveg akkor kerül mondatként elfogadásra, ha az automata beolvasta a szöveget, és elfogadó állapotba került, vagy további ε mozgásokkal elfogadó állapotba kerül. A veremautomatáknál nem feltétele az elfogadásnak a mozgásképtelenség. Az is elfogadható, ha az ε szabályok ciklikusságot idéznek elő, de közben elfogadó állapoton is áthalad az automata. Az elfogadó állapotban megkövetelhetjük, hogy a verem üres legyen.

Nézzünk egy példát a determinisztikus veremautomaták leírására és működésére! Legyen az automata állapotainak halmaza:

$Q = \{q_0, q_1, q_2, q_3\}$

a karakterkészlete:

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

a verem alfabetikája:

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

A # karakter a verem alját jelöli. Az automata kezdeti állapota: q_0. Induláskor a verem a # karaktert tartalmazza. Az elfogadó állapotok halmaza:

$F = \{q_3\}$

és az automata mozgási szabályai:

$\delta(\mathrm{q_0}, \mathrm{a}, \#) = (\mathrm{q_1}, \mathrm{a}\#)$ $\quad$ $\delta(\mathrm{q_0}, \mathrm{b}, \#) = (\mathrm{q_1}, \mathrm{b}\#)$
$\delta(\mathrm{q_1}, \mathrm{a}, \mathrm{a}) = (\mathrm{q_1}, \mathrm{aa})$ $\quad$ $\delta(\mathrm{q_1}, \mathrm{b}, \mathrm{a}) = (\mathrm{q_1}, \mathrm{ba})$
$\delta(\mathrm{q_1}, \mathrm{a}, \mathrm{b}) = (\mathrm{q_1}, \mathrm{ab})$ $\quad$ $\delta(\mathrm{q_1}, \mathrm{b}, \mathrm{b}) = (\mathrm{q_1}, \mathrm{bb})$
$\delta(\mathrm{q_1}, \mathrm{c}, \mathrm{a}) = (\mathrm{q_2}, \mathrm{a})$ $\quad$ $\delta(\mathrm{q_1}, \mathrm{c}, \mathrm{b}) = (\mathrm{q_2}, \mathrm{b})$
$\delta(\mathrm{q_2}, \mathrm{a}, \mathrm{a}) = (\mathrm{q_2}, \varepsilon)$ $\quad$ $\delta(\mathrm{q_2}, \mathrm{b}, \mathrm{b}) = (\mathrm{q_2}, \varepsilon)$
$\delta(\mathrm{q_2}, \varepsilon, \#) = (\mathrm{q_3}, \#)$

A fenti veremautomata a wcw^{-1} jelsorozatokat fogadja el, ahol w tetszőleges a és

b karakterből álló nem üres jelsorozat, és w^{-1} jelentése w fordított sorrendben. Egy mondat elemzését az automata úgy végzi, hogy a c karakter beolvasásáig a verembe menti a beolvasott karaktereket, majd ellenőrzi, hogy a szöveg c karakter után következő része megegyezik-e a szöveg első felének megfordítottjával.

Nézzük meg, hogyan dönthetjük el, hogy egy jelsorozatot az automata elfogade, vagy sem! A veremautomata működését az un. konfigurációk sorozatán keresztül követhetjük. Az automata konfigurációja az automata pillanatnyi működési állapotát jellemzi, amit egy hármassal adhatunk meg:

$\kappa = \{w, q, \chi\}$

ahol:

- w a még el nem olvasott jelsorozat
- q az automata állapota
- χ a verem tartalma

Döntsük el, hogy az *aabacabaa* mondat benne van-e a nyelvünkben! Az induló konfiguráció:

$\{aabacabaa, q_0, \#\}$

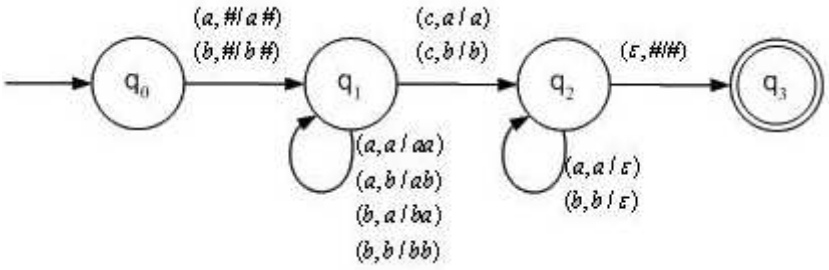
A konfigurációk sorozata a következő lesz:

$\{aabacabaa, q_0, \#\}$
$\{abacabaa, q_1, a\#\}$
$\{bacabaa, q_1, aa\#\}$
$\{acabaa, q_1, baa\#\}$
$\{cabaa, q_1, abaa\#\}$
$\{abaa, q_2, abaa\#\}$
$\{baa, q_2, baa\#\}$
$\{aa, q_2, aa\#\}$
$\{a, q_2, a\#\}$
$\{\varepsilon, q_2, \#\}$
$\{\varepsilon, q_3, \#\}$

Az automata a jelsorozatot elfogadta, mert minden szimbólumát feldolgozta, majd ezután egy ε szabállyal elfogadó állapotba került.

A mozgási szabályok megadhatók állapot-átmeneti gráf segítségével is. A gráf felépítése hasonlít a véges automatáknál használt gráfokéhoz. Az egyetlen különbség, hogy az állapotátmeneteknél figyelni kell nemcsak az inputról érkező karaktert,

hanem a verem tetején lévő karaktert is, valamint az átmenet során meg kell adni azt is, hogy a verem tetejére milyen jelsorozatot írjunk. A fenti példa állapotátmeneti gráfja:



A bemutatott példában minden állapotból adott feltételek esetén csak egy mozgás lehetséges, az automata determinisztikus. Az alábbi mozgási szabályrendszerrel egy nemdeterminisztikus veremautomatát adunk meg:

$\delta(\mathrm{q}_0, \mathrm{a}, \#) = (\mathrm{q}_1, \mathrm{a}\#)$

$\delta(\mathrm{q}_0, \mathrm{b}, \#) = (\mathrm{q}_1, \mathrm{b}\#)$

$\delta(\mathrm{q}_1, \mathrm{a}, \mathrm{a}) = (\mathrm{q}_1, \mathrm{aa}) \mid (\mathrm{q}_2, \varepsilon)$

$\delta(\mathrm{q}_1, \mathrm{b}, \mathrm{a}) = (\mathrm{q}_1, \mathrm{ba})$

$\delta(\mathrm{q}_1, \mathrm{a}, \mathrm{b}) = (\mathrm{q}_1, \mathrm{ab})$

$\delta(\mathrm{q}_1, \mathrm{b}, \mathrm{b}) = (\mathrm{q}_1, \mathrm{bb}) \mid (\mathrm{q}_2, \varepsilon)$

$\delta(\mathrm{q}_2, \mathrm{a}, \mathrm{a}) = (\mathrm{q}_2, \varepsilon)$

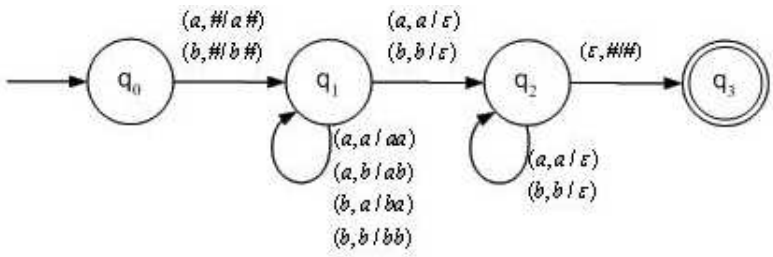
$\delta(\mathrm{q}_2, \mathrm{b}, \mathrm{b}) = (\mathrm{q}_2, \varepsilon)$

$\delta(\mathrm{q}_2, \varepsilon, \#) = (\mathrm{q}_3, \#)$

Az automata induló állapota q_0, az elfogadó állapotok halmaza pedig $F = \{q_3\}$.

Ennek az automatának két olyan szabálya is van, ahol egyetlen baloldalhoz két jobboldal tartozik. Ez azt jelenti, hogy ebben a szituációban két mozgás is lehetséges, az automata bármelyiket választhatja. Az automata *nemdeterminisztikus*.

Az automata megadható gráf segítségével is:



A most bemutatott automata a ww^{-1} mondatokat fogadja el, ahol w ismét tetszőleges a és b karakterekekből álló sorozat. A bemutatott determinisztikus és nemdeterminisztikus automata által elfogadó nyelv között az a különbség, hogy míg az első nyelv megadja, hol a mondat közepe a c karakter segítségével, addig a második nyelv esetében nekünk kell ezt megtalálnunk. Az elemzés során valahányszor két azonos szimbólum követi egymást, felvetődik a kérdés, hogy a mondat közepén állunk-e. Ennek köszönhető a kétféle mozgási lehetőség.

Elemezzük az *abbaabba* mondatot! A konfigurációk sorozata:

$\{abbaabba, q_0, \#\}$

$\{bbaabba, q_1, a\#\}$

$\{baabba, q_1, ba\#\}$???

Itt most válaszúthoz érkeztünk. Eldöntendő, hogy a mondat közepénél vagyunk-e? Válasszuk azt, hogy igen! Ekkor a levezetés az alábbiak szerint folytatódik:

$\{aabba, q_2, a\#\}$

$\{abba, q_2, \#\} \rightarrow hiba$

A levezetés elakadt, nem sikerült levezetnünk a mondatot. Menjünk vissza az elágazáshoz, és válasszuk a másik alternatívát! Ekkor az alábbi levezetést kapjuk:

$\{aabba, q_1, bba\#\}$

$\{abba, q_1, abba\#\}$???

Ismét választhatunk. Nézzük most azt az esetet, hogy megtaláltuk a mondat közepét:

$\{bba, q_2, bba\#\}$

$\{ba, q_2, ba\#\}$

$\{a, q_2, a\#\}$

$\{\varepsilon, q_2, \#\}$

$\{\varepsilon, q_3, \#\}$

Az automata tehát elfogadta a jelsorozatot, azaz az elemzett szöveg eleme az automata által feldolgozható nyelvnek. Nemdeterminisztikus automaták esetén egy jelsorozatot akkor tekintünk elfogadottnak, ha létezik olyan mozgás, amellyel az automata elfogadja a jelsorozatot.

A nemdeterminisztikus veremautomata nem minden esetben alakítható át determinisztikussá. A nemdeterminisztikus veremautomatával elemezhető nyelvek osztálya megegyezik a környezetfüggetlen nyelvek osztályával. Determinisztikus verem automatával csak a környezetfüggetlen nyelvek egy része elemezhető.

Nyelvtanból automata készítése

A környezetfüggetlen nyelvtanok éppen azokat a nyelveket generálják, amelyeket a veremautomaták elfogadnak. Most megadunk egy eljárást, hogy hogyan lehet elkészíteni egy környezetfüggetlen nyelvtan veremautomatáját:

- Az automata egyetlen állapottal rendelkezik. Így ez az induló és az elfogadó állapot is.
- A verem karakterkészlete legyen a nyelvtan terminális és nemterminális szimbólumainak uniója.
- A verem kiinduló tartalma legyen a mondatszimbólum.
- Két fajta mozgási szabályt értelmezünk:
  - Nyelvtantól független mozgási szabályok :
    $\delta(q, a, a) = (q, \varepsilon)$,
    ahol a a nyelv tetszőleges terminálisa. Ezek szerint, ha a verem tetején álló terminális szimbólum azonos a beolvasottal, akkor az a verem tetejéről leemelhető (ez a karakter elfogadásra került). Ha ez nem teljesül, akkor vissza kell mennünk egy elágazáshoz.
  - Nyelvtantól függő mozgási szabályok:
    $\delta(q, \varepsilon, A) = (q, \alpha)$,
    ahol A nemterminális, és létezik egy $A \to \alpha$ levezetési szabály. Tehát valahányszor egy nemterminális kerül a verem tetejére, azt a hozzá tartozó levezetési szabályok valamelyikének jobb oldalával helyettesítjük. Ezek lesznek az elágazások.

Ez az automata a mondatszimbólumból kiindulva a levezetési szabályok felhasználásával a nyelv minden mondatát eléri. Hiszen, ha végiggondoljuk, pontosan a mondat baloldali levezetését találja meg. Sajnos az itt bemutatott automata általában nemdeterminisztikus.

A mozgási szabályokat, mivel csak egy állapottal rendelkezik az automata, egyetlen egy táblázattal is meg lehetne adni. A táblázat a bemenetről érkező, valamint a verem tetején lévő karaktereknek megfelelően megadhatja a verem

tetejére helyezendő szöveget.

1. példa

A nyelvtan legyen az alábbi már jól ismert ww^{-1}-et leképező nyelv nyelvtana, ahol $w = (a + b)^+$:

$S \rightarrow aSa|bSb|aa|bb$

Határozzuk meg a hozzá tartozó automatát! Az automata mozgási szabályai:

- A nyelvtantól független mozgási szabályok:
  $\delta(q, a, a) = (q, \varepsilon)$
  $\delta(q, b, b) = (q, \varepsilon)$
- A nyelvtantól függő mozgási szabályok:
  $\delta(q, \varepsilon, S) = (q, aSa)|(q, bSb)|(q, aa)|(q, bb)$

2. példa

Határozzuk meg wcw^{-1} mondatokat leképező

$S \rightarrow aAa|bAb$

$A \rightarrow aAa|bAb|c$

nyelvtanhoz tartozó automatát, ahol $w = (a+b)^+$! Az automata mozgási szabályai:

- A nyelvtantól független mozgási szabályok:
  $\delta(q, a, a) = (q, \varepsilon)$
  $\delta(q, b, b) = (q, \varepsilon)$
  $\delta(q, c, c) = (q, \varepsilon)$
- A nyelvtantól függő mozgási szabályok:
  $\delta(q, \varepsilon, S) = (q, aAa)|(q, bAb)$
  $\delta(q, \varepsilon, A) = (q, aAa)|(q, bAb)|(q, c)$

Ez az automata is nemdeterminisztikus. Ha az automata előretekintéssel rendelkezne, akkor dönteni tudna, hogy a három szabály közül melyiket alkalmazza. Ha például az inputon az a karaktert látná, akkor választhatná a

$\delta(q, \varepsilon, S) = (q, aAa)$

mozgási szabályt; ha b-t, akkor

$\delta(q, \varepsilon, S) = (q, bAb)$

mozgási szabályt; ha pedig c-t, akkor

$\delta(q, \varepsilon, S) = (q, c)$

szabályt. Ezzel az automata nemdeterminisztikus jellegét megszüntethetnénk. Az első példa esetén azonban nincs remény az automata nemdeterminisztikus tulajdonságának az elkerülésére. A következő fejezetekben olyan környezetfüggetlen nyelvekkel foglalkozunk, amelyekhez determinisztikus veremautomata szerkeszthető.

1.4. LL(k) nyelvek

A determinisztikus veremautomatával elemezhető környezetfüggetlen nyelvek egyik csoportja az LL(k) nyelvek. A jelen fejezetből megismerhetjük az LL(k) nyelv fogalmát, az LL(k) elemezhetőség feltételeit, valamint az LL(k) automaták működését. Megnézzük, hogyan készíthető program LL(k) nyelvekhez szintaxis diagram felhasználásával. Végül pedig bemutatjuk az ANTLR integrált fejlesztőrendszert, aminek a segítségével EBNF leírás alapján automatikusan kód generálható.

1.4.1. Az LL(k) nyelvek fogalma és az LL(k) elemző működése

A fentről lefelé történő elemzésnél a mondatszimbólumból indulunk ki. A baloldali levezetés érdekében elemzés során mindig a mondatszerű forma legbaloldalibb nemterminálisát helyettesítjük. Általában a helyettesítendő nemterminálishoz több levezetési szabály is tartozik, és ezekből kell kiválasztanunk a megfelelőt, hogy megkapjuk az adott mondatot. Az elemzés akkor egyszerű, ha egyetlen döntési lépést sem kell visszavonni, vagyis a feldolgozás során a karakterek olvasásával egyidejűleg az alternatív produkciós szabályok közül ki tudjuk választani azt, amelyik levezetést eredményez.

Az LL(k) elemző úgy jár el, hogy az olvasás helyétől az elemző k szimbólumot előrenéz, és ezekből a terminálisokból dönti el, melyik alternatívát kell választania a lehetséges produkciós szabályok közül. Az elnevezés első L karaktere arra utal, hogy balról jobbra olvas az elemző, a második L pedig azt jelzi, hogy a levezetés során a legbaloldalibb levezetést állítja elő.

Az LL(k) nyelvek pontos leírásához vezessük be az alábbi jelölést! Legyen egy nyelvtan egyik produkciós szabálya $A \rightarrow \alpha$! $first_k^A(\alpha)$ jelölje azt a nyelvet, amit

úgy kapunk, hogy az α levezetési szabály által generált mondatokat csonkítjuk k hosszúságúra! Ha a mondat hosszabb mint k, akkor levágjuk a végét.

Nézzük meg, mi a feltétele annak, hogy egy nyelvtan LL(k) elemezhető legyen! Legyen egy nyelvtan A nemterminálisának összes lehetséges levezetési szabálya:

$A \to \alpha_1|\alpha_2|...|\alpha_n$

Egy nyelvtan LL(k) elemezhetőségének szükséges feltétele, hogy:

$first_k^A(\alpha_i) \cap first_k^A(\alpha_j) = \emptyset \quad (i,j = 1..n \quad i \neq j)$,

azaz a $first_k^A(\alpha_i)$ nyelvek diszjunkt volta. Ez a feltétel elégséges is abban az esetben, ha az A nemterminális levezetései legalább k hosszúságú szimbólumsorozatot generálnak.

A következő példa bemutatása előtt megemlítjük, hogy egy aritmetikai kifejezés megadható:

$a + b$	infix jelöléssel
$ab+$	postfix jelöléssel (fordított lengyel alak)
$+ab$	prefix jelöléssel (ilyen lesz a következő példa).

Nézzünk egy példát LL(k) nyelvekre! Nyelvtanként válasszuk az aritmetika kifejezések prefix megadását:

$E \to +EE$	1. szabály
$E \to *EE$	2. szabály
$E \to a$	3. szabály

A mondat végének és a verem aljának explicit jelölésére vezessük be a # karaktert. Ennek figyelembevételével

$S \to E\#$ 0. szabály

Ez a nyelvtan LL(1) nyelvtan, hiszen a szükséges és jelen esetben elégséges feltétel $k = 1$ mellett nyilvánvalóan teljesül:

$first_1^E(+EE) = \{+\}$

$first_1^E(*EE) = \{*\}$

$first_1^E(a) = \{a\}$

halmazok diszjunktak.

Az un. elemzőtáblát a nyelvtan alapján könnyen felírhatjuk. A vízszintes fejléc az előretekintés során kapható k hosszúságú (most *k=1*) jelsorozatokat tünteti fel. A függőleges fejléc a verem tetején lévő szimbólumokat mutatja. A táblázat

mezőibe a verem új tartalma kerül, illetve az elemzés jobb követhetősége érdekében megadjuk, hogy melyik szabállyal jutottunk ide.

	$+$	$*$	a	$\#$
E	*+EE,1*	**EE, 2*	*a, 3*	*hiba*
$+$	*pop*	*hiba*	*hiba*	*hiba*
$*$	*hiba*	*pop*	*hiba*	*hiba*
a	*hiba*	*hiba*	*pop*	*hiba*
$\#$	*hiba*	*hiba*	*hiba*	*Elfogad*

Például a *+EE,1* az adott mezőben azt jelenti, hogy ha a verem tetején E van, akkor egy $+$ szimbólum érkezésére a $+EE$ jelsorozat kerül a verem tetejére az E helyett, és az elemzés során az első szabályt használtuk.

A táblázat adott mezőiben a *pop* azt jelenti, hogy a beolvasott szimbólum megegyezik a verem legtetején lévővel, és így mindkettőt eltűntetjük. A *hiba* jelentése, hogy ilyen a levezetés során nem lehetséges. *Elfogad* esetén a teljes szöveget beolvastuk, és mondatként elfogadtuk, vagyis az inputnak a végére értünk, a verem is kiürült, így a szöveg elfogadásra került.

A táblázatnak az a része, ahol terminális volt a verem tetején, vagy a verem üres volt, a nyelvtan egyediségére jellemző információt nem tartalmaz. A táblázat ezen része minden automata esetén mechanikusan kitölthető. Ezért ezt a részt általában elhagyják, és csak azokat az eseteket nézik, ahol nemterminális van a verem tetején.

Nézzünk a fenti táblázat alapján egy elemzést! Az automata működését az alapján követhetjük nyomon, hogy az automata bemenetén mi az a jelsorozat, amit még nem olvastunk be, a veremnek mi az aktuális tartalma, valamint, hogy az automata milyen kimenetet generált. Indulásnál az elemzendő teljes jelsorozat még feldolgozásra vár, a verembe a mondatszimbólumot tesszük, az automata kimenetére még semmit sem írtunk. Kövessük nyomon az automata működését a $+a*aa$ szöveg elemzése esetén:

bemenet	*verem*	*kimenet*
$\{+a*aa\#,$	$E\#,$	$\varepsilon\}$
$\{+a*aa\#,$	$+EE\#,$	$1\}$ - az 1. szabályt használtuk
$\{a*aa\#,$	$EE\#,$	$1\}$ - a veremből és a mondat elejéről eltűntek a terminálisok
$\{a*aa\#,$	$aE\#,$	$13\}$ - ...
$\{*aa\#,$	$E\#,$	$13\}$
$\{*aa\#,$	$*EE\#,$	$132\}$
$\{aa\#,$	$EE\#,$	$132\}$
$\{aa\#,$	$aE\#,$	$1323\}$
$\{a\#,$	$E\#,$	$1323\}$
$\{a\#,$	$a\#,$	$13233\}$
$\{\#,$	$\#,$	$13233\}$
$Elfogadva$		

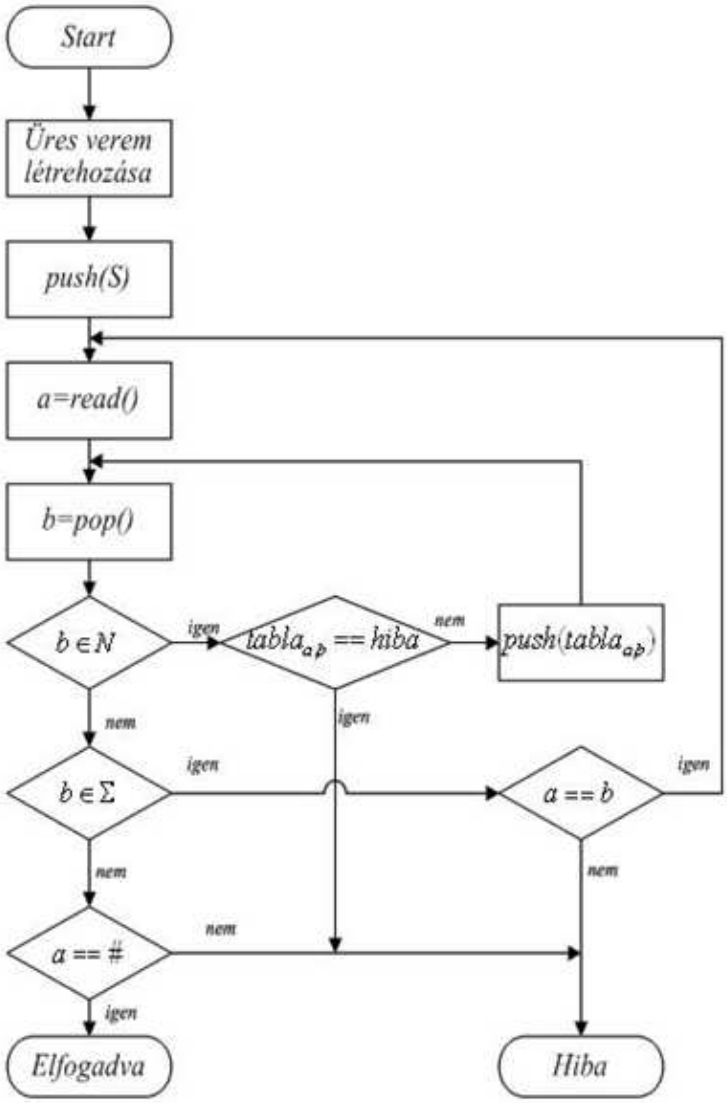
Egy LL(1) elemző működésének a folyamatábráját a 1.6. ábrán követhetjük nyomon.

Térjünk vissza a $first_k^A(\alpha_i) \cap first_k^A(\alpha_j) = \emptyset$ kifejezés szükséges, de nem elégséges voltára. Baj akkor adódhat, ha az A nemterminális utódai nem minden esetben alkotnak kellő hosszúságú, azaz az előretekintés k hosszával egyenlő vagy annál hosszabb jelsorozatot. Az elemző ugyanis ilyenkor nem tud a $first_k^A(\alpha_i)$ alapján dönteni. Ha a nemterminális derivátuma nem tölti ki a teljes hosszat, akkor az elemző hozzávesz a mögötte talált szimbólumokból annyit, hogy kiteljék a kívánt méret. Az alternatívák közötti választást ennek figyelembevételével kell megoldani.

A probléma szabatos tárgyalása érdekében vezessük be a *követő nyelv* fogalmát! Legyen A a nyelvtan egyik nemterminális szimbóluma. A $follow_k(A)$ jelentse azon legfeljebb k hosszúságú jelsorozatok halmazát, amelyet az adott grammatika az A nemterminális utódait követve generálhat. Ezt a nyelvet nevezzük az A nemterminális követő nyelvének. Amennyiben az A nemterminális nem ad ki elegendő (k hosszúságú) szimbólumot, akkor az elemző azt a követő nyelvből egészíti ki. Végülis az elemző által vizsgált jelsorozatok az alábbi összefüggéssel írhatók fel:

$first_k(first_k^A(\alpha_i) follow_k(A))$

Itt α_i ismét az A nemterminális egyik lehetséges felbontása.



1.6. ábra. Egy LL(1) elemző folyamatábrája

A $first_k(first_k^A(\alpha_i)follow_k(A))$ kifejezés magyarázata a következő. A külső zárójelen belül az A nemterminális α_i levezetési szabályával származtatható, valamint az A nemterminálist követő k hosszúságú jelsorozatok konkatenáltja áll. Ezen konkatenált jelsorozatoknak vesszük az első szimbólumait. Amennyiben az α_i egymagában generálja a megfelelő számú szimbólumot, akkor a követő nyelv jelenléte nincs hatással az eredményre. Ha viszont az α_i által generált jelsorozat rövidebb, akkor a hiányzó szimbólumokat a követő nyelv szolgáltatja. A fentiek-

ből nyilvánvaló, hogy a kifejezés éppen azokat a jelsorozatokat adja, amelyeket az elemző a vizsgálat tárgyává tesz.

Amennyiben a $first_k(first_k^A(\alpha_i)follow_k(A))$ szerinti kifejezések a különböző alternatívákra diszjunktak, akkor ez elegendő feltétel arra, hogy ennek alapján a helyes felbontás egyértelműen eldönthető legyen. Így a nyelvtan, illetve a nyelv LL(k) elemezhető.

Lássunk most egy olyan példát, ahol a $first_k^A(\alpha_i) \cap first_k^A(\alpha_j) = \emptyset$ összefüggéseken túlmenően a $first_k(first_k^A(\alpha_i)follow_k(A))$ típusú kifejezéseket is meg kell vizsgálnunk az alkalmazandó levezetési szabály kiválasztásához! Válasszuk a már jól ismert, az aritmetikai kifejezéseket generáló nyelvtant! Sajnos, ez a nyelvtan balrekurzív, és így nem balelemezhető. Az eredeti nyelvtan, mint ismeretes:

$E \to E + T|T$

$T \to T * F|F$

$F \to (E)|a$

A balrekurzivitás megszűntetésére használjuk a korábban látottaknak megfelelő ε szabályt is megengedő átírást:

1. $E \to TE''$
2. $E'' \to +TE''$
3. $E'' \to \varepsilon$
4. $T \to FT''$
5. $T'' \to *FT''$
6. $T'' \to \varepsilon$
7. $F \to (E)$
8. $F \to a$

Az új levezetési szabályokat sorszámmal is elláttuk. A mondat végének explicit jelzésére most is használhatjuk a # karaktert. Ennek figyelembevételével egy 0. szabályt is bevezethetünk:

0. $S \to E\#$

Tegyünk kísérletet egy LL(1) elemző készítésére! Határozzuk meg az egyes produkciós szabályok *first* halmazait!

1.	$E \to TE''$	$first_1^E(TE'')$	$= \{a, (\}$
2.	$E'' \to +TE''$	$first_1^{E''}(+TE'')$	$= \{+\}$
3.	$E'' \to \varepsilon$	$first_1^{E''}(\varepsilon)$	$= \{\varepsilon\}$
4.	$T \to FT''$	$first_1^T(FT'')$	$= \{a, (\}$
5.	$T'' \to *FT''$	$first_1^{T''}(*FT'')$	$= \{*\}$
6.	$T'' \to \varepsilon$	$first_1^{T''}(\varepsilon)$	$= \{\varepsilon\}$
7.	$F \to (E)$	$first_1^F\{\ (E)\ \}$	$= \{(\}$
8.	$F \to a$	$first_1^F(a)$	$= \{a\}$

Minthogy két olyan nemterminálisunk is van (lásd 3. és 6. szabályokat), amely ε-szabály alkalmazása során eltűnhet, és így adott esetben egyetlen szimbólumot sem generál, szükségünk lesz ennek a két nemterminálisnak a követő nyelvére.

Az E'' nemterminális az 1. produkciós szabály alkalmazásával kerülhet be a mondatszerű formába. Minthogy ilyenkor az E nemterminális helyébe lép, nyilván ugyanaz követheti, mint ami elődjét, az E szimbólumot követte. Az E szimbólum a mondatszimbólum, és így induláskor a mondatszerű forma egyedül ebből a szimbólumból áll. Ilyenkor semmi sem, pontosabban a szöveg vége jelsorozat, # követi. Az E nemterminális a 7. szabály révén kerülhet ismét a mondatszerű formába. Ekkor azt a végzárójel követi. Ezzel E'' követő nyelve:

$follow_1(E'') = \{\#,)\}$

Kissé bonyolultabb a másik eltűnő nemterminális, T'' követő nyelvének megállapítása. A T'' a 4. sorszámú szabály alkalmazásával lesz eleme a mondatszerű formának. Ilyenkor a T nemterminális helyébe lép, így ugyanaz követheti, ami a T nemterminálist követte. Ez utóbbit viszont, akár az 1. akár a 2. sorszámú szabály kapcsán tűnt fel a mondatszerű formában, az E'' szimbólum követi. Ez a nemterminális fogja generálni a T'' követő karaktereit. Amennyiben az E'' felbontására a 2. szabályt alkalmazzuk, akkor a helyére beírt jelsorozat első karaktere + lesz. Előfordulhat azonban, hogy a 3. szabály nyom nélkül eltünteti az E'' szimbólumot. Mi követi ilyenkor a T'' nemterminálist? Nos az E'' eltűnésével láthatóvá válik az, ami mögötte volt, ami eredetileg az E'' szimbólumot követte. Ezek alapján tehát a T'' követő nyelve:

$follow_1(T'') = \{+, \#\ ,)\}$

Ezek után már el tudjuk dönteni, hogy a szóbanforgó nyelvtan LL(1) elemezhető-e, vagyis diszjunktak-e a $first_k(first_k^A(\alpha_i)follow_k(A))$ szerinti kifejezések. Ezt a vizsgálatot csak azokra a szabályokra kell elvégezni, melyek alternatívát jelentenek, amelyek egymás konkurensei. Ilyenek a 2-3, az 5-6, és 7-8 szabálypárok. Ezekre a $first_k(first_k^A(\alpha_i)follow_k(A))$ szerinti kifejezések, amelyek az előretekintést szolgáltatják, a következők lesznek:

2. $E'' \to +TE''$ $\quad first_1^{E''}(+TE'') = \{+\}$
3. $E'' \to \varepsilon$ $\quad first_1^{E''}(\varepsilon\ follow_1(E'')) = \{\#,)\}$

5. $T'' \to *FT''$ $\quad first_1^{T''}(*FT'') = \{*\}$
6. $T'' \to \varepsilon$ $\quad first_1^{T''}(\varepsilon\ follow_1(T'')) = \{+, \#,)\}$

7. $F \to (E)$ $\quad first_1^F\{\ (E)\ \} = \{(\}$
8. $F \to a$ $\quad first_1^F(a)\ = \{a\}$

Mint látható az alternatívaként szóbajövő párokra nézve a halmazok mind diszjunktak, így a nyelvtan LL(1) elemezhető.

Írjuk fel ezek után az elemző táblát!

	(	*a*	*)*	*+*	***	*#*
E	*TE″, 1*	*TE″, 1*	*hiba*	*hiba*	*hiba*	*hiba*
E″	*hiba*	*hiba*	$\varepsilon, 3$	*+TE″, 2*	*hiba*	$\varepsilon, 3$
T	*FT″, 4*	*FT″, 4*	*hiba*	*hiba*	*hiba*	*hiba*
T″	*hiba*	*hiba*	$\varepsilon, 6$	$\varepsilon, 6$	**FT″, 5*	$\varepsilon, 6$
F	*(E), 7*	*a, 8*	*hiba*	*hiba*	*hiba*	*hiba*

A táblázat most csak a nyelvtantól függő szabályokat tartalmazza. A táblázat segítségével a nyelv egy mondatának szintaktikus elemzését könnyen elvégezhetjük.

Példaképpen legyen az elemzendő szöveg:

$a + a * a$

Az elemzés során előálló konfigurációk:

A bemeneti szöveg	*A verem tartalma*	*A kimeneti szöveg*
*a+a*a#*	*E#*	*ε*
*a+a*a#*	*TE"#*	*1*
*a+a*a#*	*FT"E"#*	*14*
*a+a*a#*	*aT"E"#*	*148*
*+a*a#*	*T"E"#*	*148*
*+a*a#*	*E"#*	*1486*
*+a*a#*	*+TE"#*	*14862*
*a*a#*	*TE"#*	*14862*
*a*a#*	*FT"E"#*	*148624*
*a*a#*	*aT"E"#*	*1486248*
**a#*	*T"E"#*	*1486248*
**a#*	**FT"E"#*	*14862485*
a#	*FT"E"#*	*14862485*
a#	*aT"E"#*	*148624858*
#	*T"E"#*	*148624858*
#	*E"#*	*1486248586*
#	*#*	*14862485863*
		Elfogadva

A követő nyelvek előállítása, mint ahogy ez a példából is látható, kellő körültekintést igényel. Itt megadjuk azt az algoritmust, amely segítségével ez mechanikussá tehető:

Az egy hosszúságú *follow* halmazok meghatározásának programja:

```
...
for A in nemterminálisok_halmaza do // minden nemterminálisra
    follow(A)={};
follow(Start_szimbólum)={ε};
repeat
    for A → xBy do begin
    // minden produkciós szabályra és
    // minden nemterminális előfordulásra a produkciós szabály jobb oldalán
    if (y == ε ) then // y üres
        follow(B)=follow(B) + follow(A);
    else if (y==terminális) then // y terminális
        follow(B)=follow(B) + y;
    else begin // y nemterminális
        follow(B)=follow(B) + (first(y)-ε);
        if (ε in first(y)) then
            follow(B)=follow(B)+follow(A);
        end
    end
until nem változik a ciklusban semelyik follow halmaz;
...
```

Példa LL(2) nyelvi elemzőre

Legyen a nyelvtanunk az alábbi:

$S \rightarrow aAbaS|bAabS|\varepsilon$ (1)(2)(3)

$A \rightarrow a|b|\varepsilon$ (4)(5)(6)

Az 1., a 2. és a 3. szabály közül egy karakter előreolvasásával egyértelműen

$first_1^S(aAbaS) = \{a\}$

választani tudunk. $first_1^S(bAabS) = \{b\}$ A problémát a 4., az 5. és

$first_1^S(\varepsilon) = \{\varepsilon\}$ - mondatvége

a 6. szabályok okozzák.

Vizsgáljuk meg, hogy *k=1*-re teljesülnek-e az LL(1) elemezhetőség feltételei!

$$first_1^A(a) = \{a\}$$
$$first_1^A(b) = \{b\}$$
$$first_1^A(\varepsilon) = \{\varepsilon\}$$
$$follow_1(A) = \{a, b\}$$

$$first_1(first_1^A(a)follow_1(A)) = \{a\}$$
$$first_1(first_1^A(b)follow_1(A)) = \{b\}$$
$$first_1(first_1^A(\varepsilon)follow_1(A)) = \{a, b\}$$

Ezek nem diszjunktak. Tehát a nyelvtan LL(1)-ként nem elemezhető.

Nézzük meg k=*2*-re!

$$first_2^A(a) = \{a\}$$
$$first_2^A(b) = \{b\}$$
$$first_2^A(\varepsilon) = \{\varepsilon\}$$
$$follow_2(A) = \{ba, ab\}$$

$$first_2(first_2^A(a)follow_2(A)) = \{aa, ab\}$$
$$first_2(first_2^A(b)follow_2(A)) = \{ba, bb\}$$
$$first_2(first_2^A(\varepsilon)follow_2(A)) = \{ba, ab\}$$

Ezek sem diszjunktak. Lehetne folytatni. Ezek a halmazok semmilyen k-ra nem diszjunktak.

Nyelvtanunk azonban átalakítható úgy, hogy LL(k) elemezhető legyen. A produkciós szabályokban a nemterminálisok különböző előfordulásait megkülönböztetve a szabályok átírhatók:

$S \to aA_1baS|bA_2abS|\varepsilon$ (1)(2)(3)
$A_1 \to a|b|\varepsilon$ (4)(5)(6)
$A_2 \to a|b|\varepsilon$ (4)(5)(6)

A mondat végének explicit jelzésére most is használhatjuk a # karaktert. Ennek figyelembevételével egy 0. szabályt is bevezetünk:

$S' \to S\#$ (0)

Vizsgáljuk meg a bevezetett új nemterminálisokkal az LL(k) elemezhetőség

feltételeit! Az A_1 nemterminális esetén:

$$first_2(first_2^{A_1}(a)follow_2(A_1)) = \{ab\}$$
$$first_2(first_2^{A_1}(b)follow_2(A_1)) = \{bb\$$$
$$first_2(first_2^{A_1}(\varepsilon)follow_2(A_1)) = \{ba\}$$

Ezek diszjunktak, tehát LL(2) elemezhető. Az A_2 nemterminális esetén:

$$first_2(first_2^{A_2}(a)follow_2(A_2)) = \{aa\}$$
$$first_2(first_2^{A_2}(b)follow_2(A_2)) = \{ba\}$$
$$first_2(first_2^{A_2}(\varepsilon)follow_2(A_2)) = \{ab\}$$

Ezek is diszjunktak. Tehát ez a nyelvtan LL(2) elemezhető.

A 4., 5., 6. szabály "verziói" közül két, az első három szabálynál pedig már egy karakter előreolvasásával egyértelműen tudunk választani. Az elemző tábla ebben az esetben is elkészíthető.

	aa	*ab*	*ba*	*bb*	*a#*	*b#*	*#*
S	aA_1baS,*1*	aA_1baS,*1*	bA_2abS,*2*	bA_2abS,*2*	*hiba*	*hiba*	$\varepsilon, 3$
A_1	*hiba*	*a,4*	$\varepsilon, 6$	*b,5*	*hiba*	*hiba*	*hiba*
A_2	*a,4*	$\varepsilon, 6$	*b,5*	*hiba*	*hiba*	*hiba*	*hiba*
a	*pop*	*pop*	*hiba*	*hiba*	*pop*	*hiba*	*hiba*
b	*hiba*	*hiba*	*pop*	*pop*	*hiba*	*pop*	*hiba*
#	*hiba*	*hiba*	*hiba*	*hiba*	*hiba*	*hiba*	*elfogad*

Függőleges szegélyen a verem tetején lévő terminálisok és nemterminálisok láthatók, vízszintes szegélyen pedig a beérkező terminálisok. Mivel két karaktert előreolvasásával döntünk, ezért fel kell tüntetni az összes lehetséges kettős karaktert, valamint az egyszeres karaktereket és a # -t is. Az egyszeres előfordulásokat azért kell feltüntetni, mivel a szöveg végén lehet, hogy már csak egyetlen karakter lesz az inputon. A táblázat celláiban a verem tetejére pakolt terminális, nemterminális sorozat és a kimeneten megjelenő karakter (az alkalmazott szabály sorszáma) található.

Kövessük egy mondat feldolgozását:

A bemeneti szöveg	A verem tartalma	A kimeneti szöveg
$\underline{aa}baaba\#$	$\underline{S}\#$	ε
$\underline{aa}baaba\#$	$\underline{a}A_1baS\#$	1
$\underline{ab}aaba\#$	$\underline{A_1}baS\#$	1
$\underline{ab}aaba\#$	$\underline{a}baS\#$	14
$\underline{ba}aba\#$	$\underline{b}aS\#$	14
$\underline{aa}ba\#$	$\underline{a}S\#$	14
$\underline{ab}a\#$	$\underline{S}\#$	14
$\underline{ab}a\#$	$\underline{a}A_1baS\#$	141
$\underline{ba}\#$	$\underline{A_1}baS\#$	141
$\underline{ba}\#$	$\underline{b}aS\#$	1416
$\underline{a\#}$	$\underline{a}S\#$	1416
$\underline{\#}$	$\underline{S}\#$	1416
$\underline{\#}$	$\underline{\#}$	14163
		$Elfogadva$

1.4.2. LL(k) elemző szintaxis diagram használatával

Az előző fejezetekben egy olyan fentről-lefelé, un. top-down felismerő algoritmust mutattunk be, amely bármilyen, az LL(k) nyelveknél ismertetett

1. $first_k^A(\alpha_i) \cap first_k^A(\alpha_j) = \emptyset$ és
2. $first_k(first_k^A(\alpha_i)follow_k(A)) \cap first_k(first_k^A(\alpha_j)follow_k(A)) = \emptyset$

szabályoknak megfelelő nyelvre alkalmazható. Az elemző program lelke az elemző tábla. Ebben az esetben az egyes nyelvtanokat a program által megkívánt módon, meghatározott adatstruktúrák formájában kell megadni. Az általános elemzőprogramot bizonyos értelemben az az elemző táblázatot leképező adatstruktúra irányítja, innen a táblázat vezérelt (table-driven) program elnevezés.

Egy másik technika olyan speciális top-down elemzőprogram kialakítása, amely az adott szintaxist egy utasítássorozatba viszi át, azaz egy programmá képezi le. Ebben az esetben a megadott szintaxist az ún. felismerő vagy *szintaxis diagram* formájában érdemes ábrázolni. Ez a gráf jól tükrözi egy mondatelemzési eljárásban a vezérlési folyamatot.

A top-down megközelítés fő jellemzője, hogy már induláskor ismert az elemző eljárás célja. Ez nem más, mint egy mondat felismerése, azaz egy szimbólumsorozatnak a startszimbólumból való generálhatóságának eldöntése. Egy produkció alkalmazása, vagyis egy szimbólumnak egy szimbólumsorozattal való helyettesítése annak felel meg, hogy a célt bizonyos számú részre, speciális sorrendben kielégítendő részcélra bontjuk. A top-down módszert ezért szokás célorientált elemzésnek is nevezni. Egy elemző (parser) szerkesztésében igen jól kihasználható a nemterminális szimbólumok és a célok nyilvánvaló megfeleltetése: minden nemterminális szimbólum számára egy részelemzőt (subparser) szerkeszthetünk. Minden részelemzőnek az a célja, hogy felismerje azt a részmondatot, amely a megfelelő nemterminális szimbólumból generálható.

Minthogy az egész elemzőhöz egyetlen reprezentáló gráfot kívánunk szerkeszteni, a nemterminális szimbólumokat részgráfokra fogjuk leképezni. Így jutunk a felismerő gráfok alábbi szerkesztési szabályaihoz.

A szintaxis diagram szerkesztési szabályai

Nézzük meg, hogy

$A \to a$

$A \to B$

$A \to \alpha_1\alpha_2...\alpha_n$

$A \to \alpha_1|\alpha_2|...|\alpha_n$

$A \to \alpha A|\varepsilon$

nyelvtani szabályokból hogyan szerkeszthető szintaxis diagram!

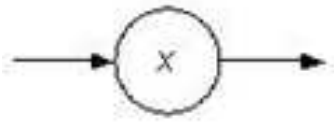
A1. Minden A nemteminális szimbólumot, amelynek képzési szabályainak halmaza:

$A \to \alpha_1|\alpha_2|\cdots|\alpha_n$

leképezünk egy A-t felismerő gráfba. A gráf struktúráját a képzési szabály jobb oldala fogja meghatározni, az $A2 \cdots A6$. szabályoknak megfelelően.

A2. $A \to a$

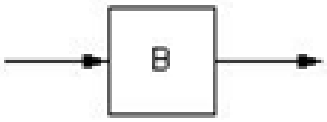
Bármelyik helyen levő a terminális szimbólumnak egy felismerő utasítás felel meg, miközben az input mondaton az olvasópozíció a következő szimbólumra helyezkedik. A gráfban ezt egy bekarikázott a címkéjű él ábrázolja:



???? ÁBRA x->a

A3. $A \to B$

Bármelyik α_i helyen egy B nemterminális szimbólumnak a B-t célzó felismerés elindítása felel meg. A gráfban ezt egy bekeretezett B címkéjű él ábrázolja:



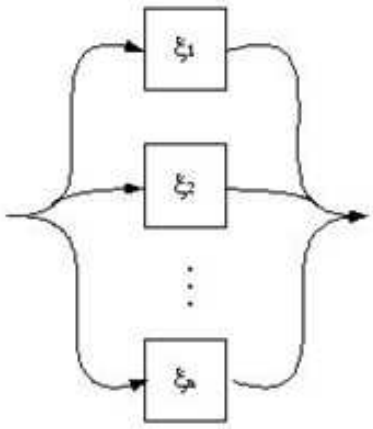
A4. Egy

$A \to \alpha_1\alpha_2 \ldots \alpha_n$

formájú produkciós szabály képe a következő:

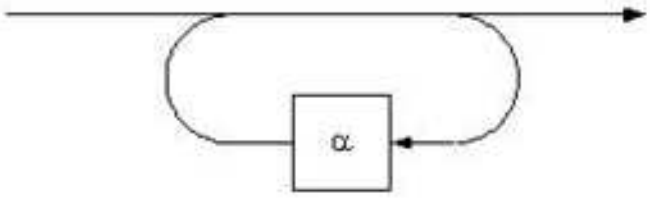
ahol minden α_i az $A2 \ldots A6$. szerkesztési szabályok α_i-re történő alkalmazásával kapható.

A5. $A \to \alpha_1|\alpha_2|\cdots|\alpha_n$
formájú képzési szabály képe az alábbi gráf:



???? ÁBRACSERE – alfa
amelyben minden α_i az $A2 \cdots A6$. szerkesztési szabályok α_i-re történő alkalmazásával nyerhető.

A6. Egy
$A \to \alpha A|\varepsilon$
képe pedig



ahol α az $A2 \cdots A6$. szerkesztési szabályok α-ra történő alkalmazásával kapható.

1. példa

Szerkesszünk szintaxis diagramot az alábbi nyelvtanhoz:

$A \to x|(B)$

$B \to AC$

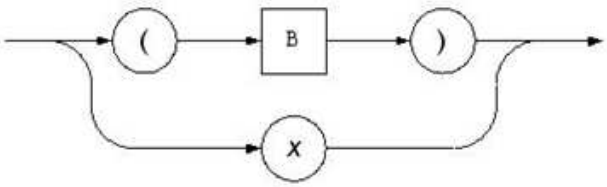
$C \to +AC|\varepsilon$

Itt *x*, *+*, *(* és *)* terminális szimbólumok. Az A-ból generálható nyelv az x operandusból, a $+$ műveletből és zárójelekből álló kifejezéseket tartalmazza, például

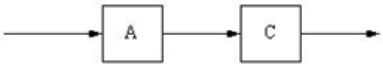
x
(x)
(x+x)
((x))
...

A szerkesztési szabályok alkalmazásával adódó diagramokat a következő ábra mutatja:

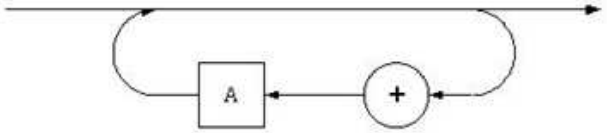
A:



B:

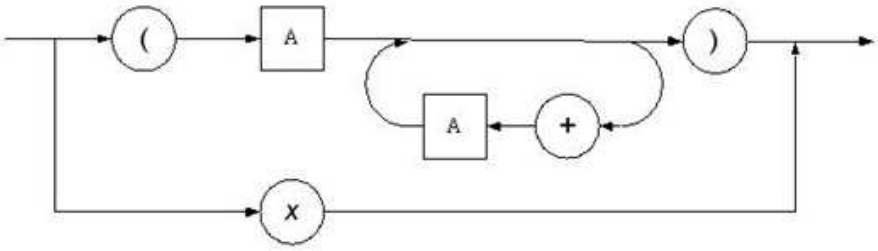


C:



Megjegyezzük, hogy a diagramokat egyetlen diagrammá alakíthatjuk, C-nek B-be és B-nek A-ba való alkalmas helyettesítésével.

A:



A felismerő gráf, amely a nyelvet leíró nyelvtan ekvivalens ábrázolása, a BNF (vagy EBNF) képzési szabályok halmaza helyett használható. Ez igen kényelmes forma, és sok esetben (sőt legtöbbször) előnyösebb a BNF-nél. Világos és tömör képet ad a nyelv struktúrájáról, az elemzőeljárás megértését is elősegíti. Egy nyelv tervezőjének a gráfos forma kiindulási lehetőség lehet.

Térjünk vissza az LL(1) elemezhetőséghez! Az 1. és 2. szabály előírásait azért vezettük be, hogy determinisztikus elemzést alkalmazhassunk csupán egy szimbólummal való előretekintéssel. Hogyan jelennek meg ezek a szabályok a szintaxis diagramban? Ebből a szempontból a diagramok egyszerűsége szembetűnő:

1. Az 1. szabálynak megfelelő követelmény az, hogy minden elágazásnál kizárólag az ágakhoz tartozó következő szimbólum alapján lehessen eldönteni, hogy melyik ágon folytatódjék a vizsgálat. Ebből következik, hogy ezekhez az ágakhoz különböző szimbólumoknak kell tartozniuk.
2. A 2. szabálynak megfelelő követelmény az, hogy ha egy A diagram bejárható "üres ágon" is, tehát úgy, hogy input szimbólumot egyáltalán nem kell leolvasni, akkor a folytató ágak címkéi között szerepeljen az összes olyan szimbólum, amely követheti A-t. (Ugyanis csak így dönthető el, hogy rálépjünk-e az üres ágra.)

Egyszerűen ellenőrizhető, hogy a diagrammok egy adott rendszere megfelel-e az alkalmazott szabályoknak, és nem kell-e visszatérnünk a nyelvtan (E)BNF reprezentációjára. Segéd-lépésként a *first(A)* és a *follow(A)* halmazokat kell minden A diagramban megállapítani. Ezek után 1. és 2. szabályok alkalmazása nyilvánvaló. Az olyan diagramrendszereket, amelyek kielégítik ezt a két szabályt, determinisztikus szintaxis diagramoknak nevezzük.

A determinisztikus szintaxis diagramból automatikusan generálható az elemzőprogram. Az elemző program készítésének lépései az alábbiak.

Legyen adott egy *"ch=getchar()"* és egy *"hiba()"* függvényünk!

0. lépés: A lehetséges összevonás elvégzése.
1. lépés: A fenti szabályok átírása
1.1.

$A \rightarrow a$

if (ch=='a') ch=getchar(); else hiba();

1.2.

$A \rightarrow B$

B(); //meghívjuk a "B" eljárást (függvényt)

1.3.

$A \rightarrow \alpha_1\alpha_2...\alpha_n$

$\{\alpha_1; \alpha_2; ...; \alpha_n\}$ *//meghívjuk egymás után a megfelelő eljárásokat*

1.4.

$A \rightarrow \alpha_1 \mid \alpha_2 \mid ... \mid \alpha_n$

switch (ch) {

case $L_1 : \alpha_1$; *break;*

case $L_2 : \alpha_2$; *break;*

...

case $L_n : \alpha_n$; *break;*

default: hiba();

}

/ meghívjuk a beolvasott karakternek megfelelő függvényt, ahol* L_i *az* α_i *, valamint az* A *nemterminális által meghatározott induló karakterhalmaz */*

1.5.

$A \rightarrow \alpha A \mid \varepsilon$

while (ch in L) α *;*

/ amíg a beolvasott karakter az* α *terminális, nemterminális sorozatnak megfelelő L halmazba esik, addig meghívja az "*α*" eljárást */*

A kódgenerálás szabályainak felhasználásával az előző példa elemző programja:

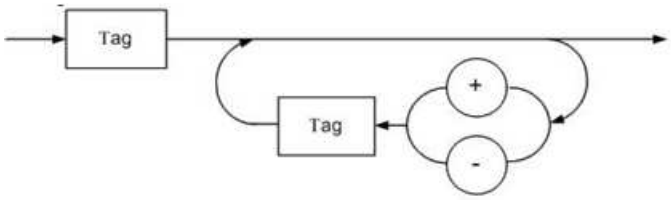
```
#include <stdlib.h>
#include <stdio.h>
void hiba(){
     puts("\r\n HIBA\r\n");
     exit(1);
}
int ch;
void A(){
     if (ch=='x')
          ch=getchar();
     else if (ch=='(') {
          ch=getchar();
          A();
          while (ch=='+') {
               ch=getchar();
               A();
          }
          if (ch==')')
               ch=getchar();
          else hiba();
     }
     else hiba();
}
int main(){
     puts("\r\n KIFEJEZES: ");
     ch=getchar();
     A();
     puts("\r\n O.K.\r\n");
     exit(0);
}
```

Az elemző program nemcsak a formai ellenőrzést tudja elvégezni, hanem a tényleges fordításnak is alapja lehet. Néhány járulékos utasítás segítségével a fordítás szemantikus részét is képes megvalósítani.

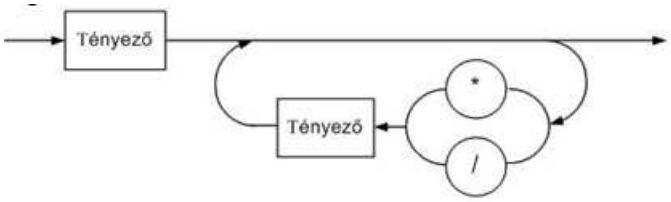
2. példa

Készítsünk az alábbi szintaxis diagramok alapján egy aritmetikai kifejezést elemző programot! Legyen a szintaxis diagramok rendszere:

Kifejezés:

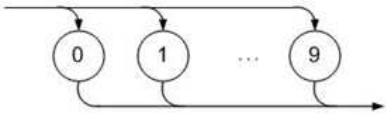


Tag:



Tényező:

Szám:



A szintaxis diagramok alapján az elemző program:

```
#include <stdio.h>
#include <stdlib.h>
extern void hiba();
extern void kifejezes();
int ch;
```

```
int main(){
      puts("\r\n KIFEJEZES: ");
      ch=getchar();
      kifejezes();
      puts("\r\n OK\r\n");
      exit(0);
}
```

```
void tenyezo() {
    if (ch=='(' ) {
        ch=getchar();
        kifejezes();
        if (ch==')') ch=getchar(); else hiba();
    }
    else if ( (ch>='0') && (ch<='9') ) ch=getchar();
    else hiba();
}
```

```
void tag(){
    tenyezo();
    while ( (ch=='*') || (ch=='/') ) {
        ch=getchar(); tenyezo();
    }
}
```

```
void kifejezes(){
    tag();
    while ( (ch=='+') || (ch=='-') ) {
        ch=getchar(); tag();
    }
}
```

```
void hiba(){
    puts("\r\n HIBA\r\n"); exit(1);
}
```

Bővítsük programunkat oly módon, hogy a produkciós szabályok mögött rejlő szemantikát is vegyük figyelembe! Készítsünk egy aritmetikai kifejezést kiértékelő programot!

```
#include <stdlib.h>
#include <stdio.h>
extern void hiba();
extern double kifejezes();
int ch;
```

```
int main(){
    puts("\r\n KIFEJEZES: ");
    ch=getchar();
    printf("\r\n EREDMENY: %10.2f\r\n",kifejezes());
    exit(0);
}
```

```
double tenyezo() {
    double a;
    if (ch=='(' ) {
        ch=getchar();
        a=kifejezes();
        if (ch==')') ch=getchar(); else hiba();
    }
    else if ( (ch>='0') && (ch<='9') ) { a=ch-'0'; ch=getchar(); }
    else hiba();
    return a;
}
```

```
double tag(){
    double a,b; int op;
    a=tenyezo();
    while ( (ch=='*') || (ch=='/') ) {
        op=ch;
        ch=getchar();
        b=tenyezo();
        a=(op=='*')?a*b:a/b;
    }
    return a;
}
```

```
double kifejezes(){
    double a,b; int op;
    a=tag();
    while ( (ch=='+') || (ch=='-') ) {
        op=ch;
        ch=getchar();
        b=tag();
        a=(op=='+')?a+b:a-b;
    }
    return a;
}
```

```
void hiba(){
    puts("\r\n HIBA\r\n"); exit(1);
}
```

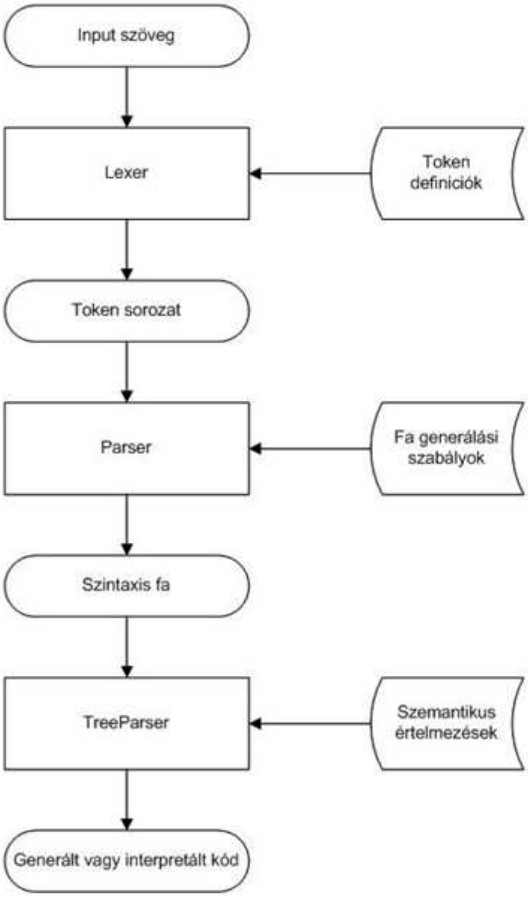
A fenti példa jól mutatja, hogy a szintaktikus elemző program a szemantikus feldolgozó program vázaként használható.

1.4.3. Az ANTLR fejlesztői eszköz

Az előző pontban láthattuk, hogy a fordítás folyamatának formai részei automatizálhatók. Így nem meglepő, hogy készültek olyan segédeszközök, amelyek segítsé-

gével a mechanikus tevékenységek elvégzése megkönnyíthető. Ilyen eszköz például az ANTLR szoftvercsomag. A programcsomag és a hozzá tartozó dokumentáció a *http://www.antlr.org* címről letölthető.

A program automatikusan elvégzi a program szövegének a tokenizálását, alapegységekre bontását, valamint a tokenek felhasználásával generálja a szintaxis fát. A szintaxis fa alapján pedig könnyen elkészíthető a szöveg szemantikus értelmezése. Az ANTLR program egy előfeldolgozó program, amely a nyelvi eszközökkel leírt programot képes - többek között - C++ kódú programmá fordítani. A szövegfeldolgozás folyamatábráját a 1.7. ábrán láthatjuk.



1.7. ábra. Az ANTLR logikai működése

A fordításhoz az alábbi információt kell megadni:

1. a tokenek leírását,
2. a szintaxis fa generálásának szabályait, valamint
3. a szintaxis fa szemantikus értelmezését.

Az ANTRL működését egy egyszerű példán szemléltetjük. A példa egy aritmetikai kifejezés kiértékelését megvalósító interpretert mutat be az egész számok körében. A bemutatott verzióban az aritmetikai kifejezés összeadást, kivonást, szorzást és osztást tartalmazhat. A kifejezésben zárójelek használata megengedett. A kiértékelendő kifejezést pontosvessző zárja. Az interpreter ANTRL programja az alábbi:

```
// A tokenek definíciói
class CalcLexer extends Lexer;
LPAREN  :       '('
        ;
RPAREN  :       ')'
        ;
STAR    :       '*'
        ;
SLASH   :       '/'
        ;
PLUS    :       '+'
        ;
MINUS   :       '-'
        ;
SEMI    :       ';'
        ;
WS_     :       (' '
        |       '\t'
        |       '\n'
        |       '\r')
                { _ttype = ANTLR_USE_NAMESPACE(antlr)Token::SKIP; }
        ;
INT     :       ('0'..'9')+
        ;
```

```
// A szintaxis fa generálásának szabályai
class CalcParser extends Parser;
... // ide fordítási opció kerül
statement   :       expr SEMI!
            ;
expr        :       mexpr ((PLUS^ | MINUS^)mexpr)*
            ;
mexpr       :       atom ((STAR^ | SLASH^)atom)*
            ;
atom        :       INT
            |       LPARAM! expr RPARAM!
            ;
```

```
// A szintaxis fa értelmezése
class CalcTreeWalker extends TreeParser;
expr returns [int r]
{
    int a,b;
    r=0;
}
    :       #(PLUS      a=expr b=expr)   {r = a+b;}
    |       #(MINUS     a=expr b=expr)   {r = a-b;}
    |       #(STAR      a=expr b=expr)   {r = a*b;}
    |       #(SLASH     a=expr b=expr)   {r = a/b;}
    |       i:INT                        {r = atoi(i→ getText().c_str());}
    ;
```

????

A program három részből áll, a Lexer-ből, a Parser-ből és a TreeParser-ből. Ahogy azt a fenti példa is mutatja, az ANTLR környezetben a tokeneknek és a szintaxis fának a nyelvtanát, valamint a szintaxis fának az értelmezését EBNF szabályok segítségével lehet megadni, amit a nyelvtani szabályok megadásába beágyazott C++ utasításokkal, valamint szintaktikus és szemantikus feltételekkel egészíthetünk ki. Az EBNF jellegű leírásban használható szimbólumok jelentését az alábbi táblázat tartalmazza.

Szimbólum	Jelentés
(...)	Alszabály (a zárójelen belüli elemek csoportként viselkednek)
(...)*	Ismétlődés 0,1,2,3...-szor
(...)+	Ismétlődés 1,2,3...- szor
(...)?	Opció (előfordulás 0-szor vagy 1-szer)
{...}	Szemantikai parancs (pl. szintaxis fa építése)
[...]	Szabály argumentum (pl. visszatérési érték)
|	Alternatívák elválasztása
..	Tartomány (intervallum)
~	Nem operátor (pl. nem szerepel egy bizonyos karakter)
.	Joker karakter
=	Hozzárendelés, legyen egyenlő
:	Címke operátor, szabály kezdete
;	Szabály vége

Az ANTRL-ben használt szimbólumok jelentése

A Lexer a beérkező szöveget a tokenek leírásának megfelelően résszövegekre vágja. Az így feldarabolt szöveg lesz a Parser bemenete. Az ANTLR-ben a nyelvtani szabályok formátumának alakja:

szabály: *alternatíva_1*
| *alternatíva_2*
...
| *alternatíva_n*

Minden alternatíva elemek sorozatából épül fel. A szabályok feldolgozása során a tokenek sorozatából a Parser a 1.8. ábrán látható szintaxis fát generálja, amelynek írott formája
(*gyökér_elem* *gyerek_elem_1* *gyerek_elem_2* ... *gyerek_elem_m*)
ahol minden *gyerek_elem* is lehet részfa.

A fa generálását speciális szimbólumokkal lehet vezérelni. Például a token után írt ! karakter hatására a tokenből nem képződik faelem. A ^ karakter segítségével pedig gyökérelem generálható a szöveg kiértékelési sorrendjétől függetlenül. A

1.8. ábra. A szintaxis fa

fenti program például a

$1 * 2 + 3$

bemenetre a

 # (PLUS # (STAR 1 2) 3)

fát generálja. A Parser által generált fát végül a TreeParser szemantikusan értelmezi.

1.5. LR nyelvek

Az előző fejezetben olyan automatát (LL(k) automatát) készítettünk, amely a mondatszimbólumból kiindulva próbálja eldönteni egy adott szövegről, hogy az adott nyelvhez tartozik-e. Ezt az elemzést fentről lefelé irányuló (top-down) elemzésnek nevezik. Készíthető olyan elemzői is, amely fordított irányban működik: a kérdéses szövegből kiindulva próbál meg eljutni a mondatszimbólumig. Az ilyen elemző lentről felfelé irányú un. bottom-up elemzést végez. A Cocke-Younger-Kasami eljárás is ezt az irányt járva működik. A következőkben egy olyan elemzőt ismertetünk, amely lentről felfelé elemez, de ugyanakkor számítási komplexitása a CYK módszernél lényegesen kisebb. A fentről lefelé, illetve a lentről felfelé irányú elemzők közötti lényegi különbség a verem használatában van. Az egyik a felismerendő, a másik pedig a redukálandó szimbólumokat tartja a veremben. A következőkben bemutatásra kerülő un. LR módszer műveletigénye a feldolgozandó szöveg hosszával egyenesen arányos. A módszer a szöveget balról jobbra olvassa,

és a jobboldali levezetést fordított sorrendben állítja elő.

1.5.1. Az LR elemző fogalma és működése

A bemutatásra kerülő módszert az irodalomban shift-reduce elemzőnek nevezik. Az elemzés célja, hogy a nyelvtani szabályok segítségével a feldolgozandó szöveget egyetlen nemterminálisra redukáljuk. A módszer lényegét egy egyszerű példán mutatjuk be. Legyen nyelvtanunk az alábbi produkciós szabályokkal adott:

1. $E \to E + T$
2. $E \to T$
3. $T \to T * F$
4. $T \to F$
5. $F \to a$
6. $F \to (E)$

A feldolgozandó szöveg pedig:

$a + a * a$.

Feldolgozás során az automata két fajta műveletet végez. A beolvasott karaktert a verembe teszi, illetve ha létezik olyan produkciós szabály, amelynek jobboldali része megegyezik a verem tetején lévő terminálisokkal és nemterminálisokkal, akkor a verem tetején lévő jelsorozatot a produkciós szabály baloldali nemterminálisára redukálja. A két művelet angol elnevezése: shift, illetve reduce. Innen származik az automata elnevezése is. Az $a + a * a$ szöveg feldolgozását a 1.1. táblázatban követhetjük nyomon. A táblázat tartalmazza a verem tartalmát, a bemeneten még fel nem dolgozott szöveget, valamint az elemzés során elvégzendő műveleteket. A táblázatban a bemeneti szövegnek a végét és a veremnek az alját külön nem jelöltük.

Indulásnál a verem üres. Az első lépésben a beolvasott karaktert a verembe tesszük. Ha a verem tetejének tartalma megegyezik valamelyik produkciós szabály jobboldalával, akkor a verem tetején lévő szimbólumsorozatot a produkciós szabály baloldalával helyettesítjük. Az input karakterek verembe tételét és a redukciót addig folytatjuk, míg minden input karaktert fel nem dolgoztunk,

Sorszám	*A verem tartalma fordított sorrendben*	*A feldolgozandó szöveg*	*Az elvégzendő művelet*
1		*a+a*a*	*shift*
2	*a*	*+a*a*	*redukció (5.szabály)*
3	*F*	*+a*a*	*redukció (4.szabály)*
4	*T*	*+a*a*	*redukció (2.szabály)*
5	*E*	*+a*a*	*shift*
6	*E+*	*a*a*	*shift*
7	*E+a*	**a*	*redukció (5.szabály)*
8	*E+F*	**a*	*redukció (4.szabály)*
9	*E+T*	**a*	*shift*
10	*E+T**	*a*	*shift*
11	*E+T*a*		*redukció (5.szabály)*
12	*E+T*F*		*redukció (3.szabály)*
13	*E+T*		*redukció (1.szabály)*
14	*E*		*Elfogadva*

1.1. táblázat. Az $a+a*a$ szöveg feldolgozása

valamint csak egy nemterminális marad a vermen. A feldolgozás során a shift és a redukálás műveleti sorrendjében konfliktus lehet. A konfliktust a jelen példában egy karakterrel való előretekintéssel feloldhatjuk. A veremkezelés megkönnyítése érdekében a veremre nem terminálisokat és nemterminálisokat tesznek, hanem állapotokat. Így minden esetben elég csak a veremnek a tetejét vizsgálni. Az elemzéshez un. LR táblát használnak. A táblázat tartalmazza a shift és redukciós tevékenységek előírását. A táblázat oszlopai közül a verem tetején lévő állapotok, sorai közül pedig az előretekintésnél használt szimbólumok alapján választhatunk. Az LR elemzők készítésének egyik fő kérdése az elemző táblázat méretének csökkenése, valamint a nyelvtan alapján a táblázat automatikus előállítása. Példaként megadjuk a fenti nyelvtanhoz tartozó LR táblázatot.

A táblázatban az üres helyek hibát jelentenek. Az S a shift művelet jelöli. Az elemzés során a verembe az S indexében szereplő állapot kerül, valamint az input következő karaktere beolvasásra kerül. Az R a redukciót jelöli. Az R indexe határozza meg a redukciós szabály sorszámát. A redukció eredményeként a szabály jobb oldalán álló szimbólumok számának megfelelő elem a verem tetejéről törlő-

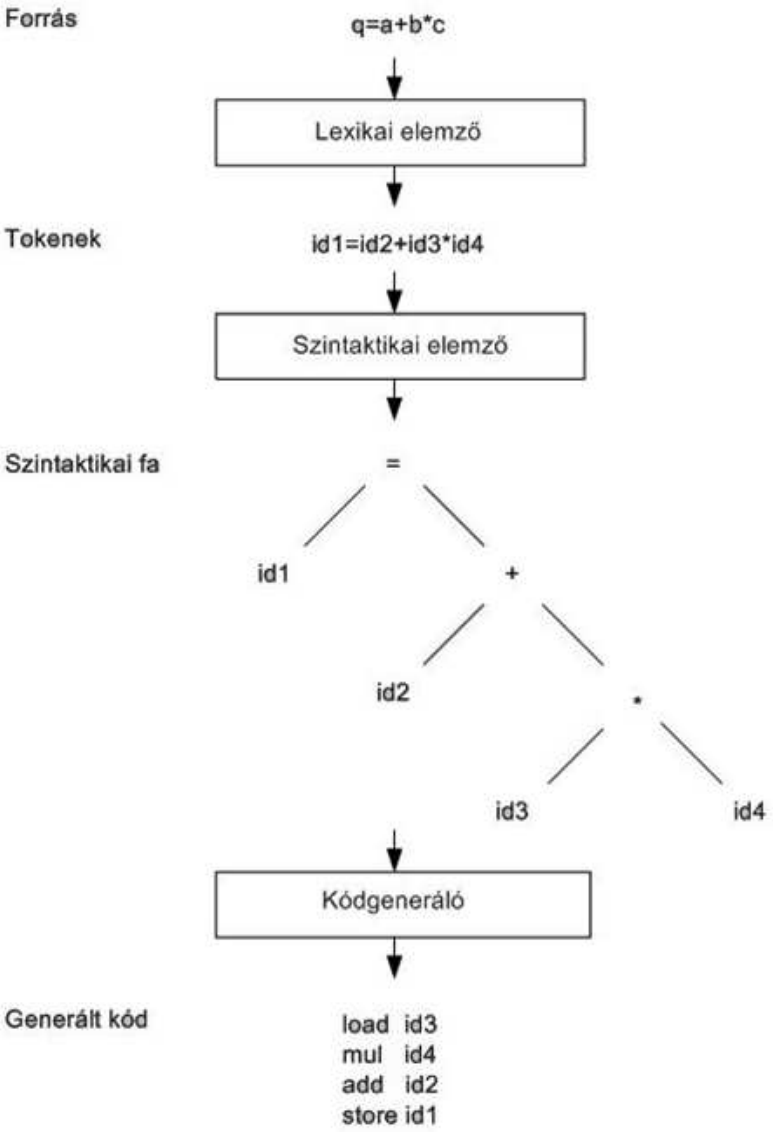
	E	T	F	a	$($	$+$	$*$	$)$	$\#$
S_0	S_1	S_7	S_4	S_5	S_6				
S_1						S_3			R_0
S_2									
S_3		S_{11}	S_4	S_5	S_6				
S_4						R_4	R_4	R_4	R_4
S_5						R_5	R_5	R_5	R_5
S_6	S_{12}	S_7	S_4	S_5	S_6				
S_7						R_2	S_8	R_2	R_2
S_8			S_9	S_5	S_6				
S_9						R_3	R_3	R_3	R_3
S_{10}						R_6	R_6	R_6	R_6
S_{11}						R_1	S_8	R_1	R_1
S_{12}						S_3		S_{10}	

dik, valamint a szabály baloldalán lévő nemterminális alapján az LR táblázatból kiolvassuk, hogy milyen állapotot tegyünk a verembe. R_0 esetén az automata a feldolgozás végére ér, és elfogadja a szöveget mondatként.

1.5.2. A LEX és a YACC elemző

A fordítóprogramok írásában sok automatizálható elem van. A szövegek szintaktikai elemzése a szabályok megadása után - mint, ahogy ezt az előző fejezetekben láttuk - mechanikus jellegű feladat. A szövegek szemantikus értelmezéséről azonban ugyanez nem mondható el. A fordítóprogram-készítés automatizálható részének támogatására szoftver eszközök állnak rendelkezésre. A reguláris kifejezések feldolgozását Unix környezetben a LEX (illetve Windows környezetben a FLEX), a BNF alapú leírásban adott környezet független nyelvek feldolgozását pedig a YACC (illetve BISON) program segíti. A LEX és a YACC a Unix rendszer része. A FLEX és a BISON a Free Software Foundation terméke. Mindkét program C nyelvű eljárást generál a nyelv specifikációja alapján. A generált C nyelvű eljárások közvetlenül beépíthetők az alkalmazásokba. A lexikális elemző (scanner), valamint a szintaktikai elemző (parser) elvileg két független program, de a két program tevékenysége összehangolható. A fordítás elemi lépéseit a 1.9. ábrán láthatjuk. A

lexikális elemző a szöveget alapvető szintaktikai egységekre tokenekre bontja. A szintaktikai elemző pedig elkészíti a tokenizált szöveg szintaktikai fáját.



1.9. ábra. A fordítás folyamata

LEX (Lexical Analyser Generator)

Nézzük először a LEX program használatát! A program a reguláris kifeje-

zésekhez automatikusan elkészíti azokat a determinisztikus véges automatákat, amelyek a szöveg felbontásához szükségesek.

A LEX-ben a reguláris kifejezések a 1.2. táblázatban bemutatott metakarakterekkel adhatók meg. A karakterosztályban további két speciális karakter használható. A két karakter között használt " - " jel a kezdő és a végkarakter közötti karakterhalmazt jelöli. Az első karakterként használt " $\widehat{}$ " karakter negálást jelöl. A 1.3. táblázat néhány példát mutat a metakarakterekből felépített reguláris kifejezésekre.

Metakarakter	Illesztési szabály
.	Tetszőleges karakter újsor jel kivételével
$\backslash n$	Újsor jel
$\widehat{}$	Sor eleje
$	Sor vége
$''a + b''$	Szövegkonstans
[]	Karakterosztály
$*$	Az előtte álló kifejezés ismétlése nullaszor vagy többször
$+$	Az előtte álló kifejezés ismétlése egyszer vagy többször
?	Az előtte álló kifejezés ismétlése nullaszor vagy egyszer
$a\|b$	Alternatíva: a vagy b
ab	Folytatás (nincs jele): a után b
()	Csoportosítás

1.2. táblázat. A LEX reguláris kifejezés metakarakterei

Kifejezés	Illesztés
abc	abc
$ab*$	$aababbabbb \ldots$
$ab+$	$ababbabbb \ldots$
$[abc]$	a vagy b vagy c karakter
$[a-zA-Z0-9]$	Egy alfanumerikus karater
$[\widehat{}abc]$	Tetszőleges karakter kivéve a vagy b vagy c karaktert
$a(bc)?$	$aabc$

1.3. táblázat. Példák reguláris kifejezésekre

A LEX által feldolgozható *.l* kiterjesztésű állomány három részből épül fel, melyeket %% jel választ el:

```
... definiciók ...
%%
... szabályok ...
%%
... függvények ...
```

A szabályok két részből tevődnek össze:

- egy reguláris kifejezésből és
- egy C nyelvű utasításból, ami lehet összetett utasítás is.

A reguláris kifejezést a sor legelső pozícióján kell kezdeni. A két részt legalább egy szóköz vagy egy tabulátor választja el.

A LEX program az *yylex* függvényt tartalmazó C nyelvű modult generálja. Az *yylex* függvény az *yyin* inputról folyamatosan olvassa a karaktereket az állomány végéig, és közben a szabályok által előírt illesztéseket, tokenizálást végzi. Amennyiben egy illesztés sikerült, akkor a reguláris kifejezés után álló utasítást hajtja végre az *yylex* függvény, a felismert token az *yytext* szöveg pointerrel érhető el. Az *yylex* függvény visszaadott értéke a token típusa. File vége olvasás után a függvény nulla értéket ad vissza.

Amennyiben a szövegre egyidejűleg a *.l* kiterjesztésű állomány szabály részében megadott több reguláris kifejezés is illeszthető, akkor a program ezek közül a leghosszabbat választja. Azonos hossz esetén pedig a felsorolásban előbb szereplő szabályt illeszti.

A LEX program bemenő állományának definíciós részében is elhelyezhető C nyelvű programrészlet. Ebben a részben elhelyezett C kódot %{ %} bevezető és lezáró karakterek közé kell zárni. A LEX által feldolgozható állomány harmadik része egy az egyben átmásodik a generált állományba.

Az alábbiakban egy egyszerű példát mutatunk be a lexikális elemző használatára. A program a parancssorban argumentumként megadott állományban található szöveget darabolja fel számokra, azonosítókra, elválasztókra és speciális karakterekre.

demo.l fájl

```
%{
#include <stdlib.h>
%}
%%
[0-9]+                          printf("integer: %s\r\n",yytext);
[a-zA-Z][a-zA-Z0-9]*            printf("identification: %s\r\n",yytext);
[\t\n]+                         printf("white space\r\n");
.                               printf("spec.char: %c\r\n",*yytext);
%%
int yywrap(void) {
        return 1;
}
int main (int argc, char* argv[]){
        yyin=fopen(argv[1],"r");
        yylex();
        fclose(yyin);
        return 0;
}
```

A fenti programban használtuk az *yywrap* függvényt, amit az *yylex* függvény a file vége olvasása után hív meg. Amennyiben a visszaadott értéke 1, akkor további állományt nem kívánunk az *yylex* függvénnyel elemezni. Végül itt jegyezzük meg, hogy a később bemutatásra kerülő programokban még használjuk az *yylval* változót, ami az illesztett tokenhez rendelt értéket tartalmazza. A szabályoknál megadott C összetett utasításban általában *return <token típus>* jellegű utasítás is szerepelhet, ami értelemszerűen az *yylex* függvényből való kilépést eredményezi. Az *yylex* függvény újabb meghívásával a felismert token utáni karaktertől folytathatjuk az illesztést.

YACC (Yet Another Compiler Compiler)

A YACC program a szintaktikai szabályok BNF alapú leírása alapján készít LR(1) alapú elemzőt. Pontosabban LALR(1) elemző táblát generál. A generáláshoz használt *.y* kiterjesztésű file felépítése hasonló a *.l* kiterjesztésű file-ok felépítéséhez:

```
... definiciók ...
%%
... szabályok ...
%%
... függvények ...
```

A LEX és a YACC által használt források közötti lényegi különbség, hogy a LEX reguláris nyelvek, a YACC pedig környezet független nyelvek leírására alkalmas szabályokat használ. A YACC által előállított *yyparse* függvény meghívja a LEX által előállított *yylex* függvényt, amelynek visszaadott értéke a felismert token típusa. A YACC-nek ezeket a típusokat ismernie kell. A két forrásállományban használt típusok egyezését az *y_ tab.h* header file teszi lehetővé, amit a YACC program kimenetként szintén generál.

A YACC program működését egy egyszerű kalkulátor programon keresztül ismertetjük. A kalkulátor alkalmas a négy alapművelet elvégzésére. A kalkulátornak vannak regiszterei, amelyeket az abc egy karakterével azonosítunk. A kifejezésekben a zárójelezés megengedett. A kalkulátor a standard inputról olvas és a standard outputra írja az eredményt. A generálandó program az alábbi mintabemenetekre az alábbi válaszokat generálja:

```
Input:   c=(2+3)*3
Output:  15
Input:   z=c-4
Output:  11
Input:   (z+c)/2
Output:  13
```

A szintaktikai elemzéshez az alábbi lexikális elemzőt használjuk:

calculator.l fájl

```
%{
        #include "y_tab.h"
        #include <stdlib.h>
        void yyerror(char *);
%}
%%
[a − z]                 yylval = *yytext - 'a'; return VARIABLE;
[0 − 9]+                yylval = atoi(yytext) ; return INTEGER;
[− + / ∗ () = \n]       return *yytext;
[\t]                    ;       /* skip whitespace */
.                       yyerror("Unknown character");
%%
int yywrap(void) {
        return 1;
}
```

A fenti lexikális elemző által generált tokenek:

- a VARIABLE egy kis betűből álló karakter,
- az INTEGER számjegyek sorozata,
- a speciális karakterek (- + / * ( ) = $\backslash n$).

A szóköz és a tabulátor karaktereket átugorjuk. Minden egyéb karakterre hibajelzést adunk. Mindegyik speciális karakternek, mint tokennek a típusa a fenti példában megegyezik a karakter kódjával. A karaktertípusú kódtól eltérő token azonosításról a YACC program gondoskodik. A tokenek nevét az *.y* fájl definíciós részében meg kell adni.

A levezetések során többféle levezetési fa is kiadódhat. Ezeknek az egyértelműsítése érdekében a YACC az alábbi megkötéseket teszi. Ha az illesztésnél eltolást és redukciót is lehetne választani, akkor az eltolást választja. Ha pedig többféle redukálást is végezhetnék a levezetés során, akkor a lehetséges variánsok közül a felírási sorban az elsőt választja.

A YACC lehetőséget ad a műveleti jelek kötési irányának, valamint precedenciájának az előírására is. Kalkulátor példákban a négy alapművelet mindegyike bal asszociatív. (Ez azt jelenti, hogy az $x\ op\ y\ op\ z$ műveletsor $(x\ op\ y)\ op\ z$ sorrendben értékelődik ki.) Ennek az előírását láthatjuk a programban a token definíciók utáni sorokban. A precedenciára vonatkozólag a szorzás és az osztás művelete magasabb precedenciájú, mint az összeadás és a kivonás. A felsorolásban a műveletek

precedenciája a kisebb precedenciától halad a magasabb precedenciák felé. Egy sorban pedig azonos precedenciájú műveletek vannak.

A kalkulátor program forráskódja a 1.5.2. ábrán látható. A program szabályrészében a produkciós szabályokat írjuk le. A produkció szabály megadásának formalizmusa:

$<$ *nemterminális*$>$: $<$*szabály jobb oldala*$>$ { $<$*akció*$>$};

A produkció szabály baloldalát (vagyis a nemterminálisokat) a sor elején kezdjük, és kettősponttal folytatjuk. Ezután következik a produkciós szabály jobb oldala: a produkciós szabálynak megfelelő sorrendben terminálisok (vagyis a LEX által felismert tokenek) és nemterminálisok. A felismerés után végrehajtandó akció (utasítássorozat) a produkciós szabály után írandó kapcsos zárójelbe. A szabály megadását pontosvessző zárja.

A szabályok feldolgozásához a YACC egy un. parser stack-et használ. Az akciók végrehajtásához pedig egy un. value stack-et kezel. A parser stack és a value stack mindig szinkron működik.

Nézzük az alábbi produkciós szabály feldolgozását:

$expression : expression' +' expression\{\$\$ = \$1 + \$3; \}$;

Ha a parser stack tetején megtalálható a produkciós szabály jobb oldala, akkor a parser stack legfelső három eleme törlődik, és a parser stack tetejére kerül a produkciós szabály baloldalán lévő nemterminális. A szabály akciós részében a value stacken lévő objektumokra $ előtaggal hivatkozhatunk. A $1 a produkciós szabály jobboldalának első tagjához tartozó value stacken lévő értékét jelöli, a $2 a másodikat és így tovább. A $$ a stack tetejét jelenti a redukció végrehajtása után. A jelen példánál maradva, a value stack-ről három értéket veszünk le, és az első és a harmadik érték összegét tesszük vissza a verem tetejére.
A kalkulátor YACC programja:

calculator.y fájl

```
%{
    #include <stdio.h>
    void yyerror(char *);
    int yylex(void);
    int sym[26];
%}
```

```
%token INTEGER VARIABLE
%left '+' '-'
%left '*' '/'
%%
```

```
program:
    program statement '\n'
    |
    ;
```

```
statement:
    expression                  { printf("%d\ n ", $1); }
    | VARIABLE '=' expression   { printf("%d\ n", $3); sym[$1] = $3; }
;
```

```
expression:
    INTEGER
    | VARIABLE                  { $$ = sym[$1]; }
    | expression '+' expression { $$ = $1 + $3; }
    | expression '-' expression { $$ = $1 - $3; }
    | expression '*' expression { $$ = $1 * $3; }
    | expression '/' expression { $$ = $1 / $3; }
    | '(' expression ')'        { $$ = $2; }
    ;
%%
```

```
int main(void) {
    yyparse();
}
```

1.10. ábra. A kalkulator YACC programja

A bemutatott program Windows-os környezetben a

```
bison -y -d calculator.y
flex calculator.l
gcc -c y_tab.c lex.yy.c
gcc y_tab.o lex.yy.o -o calculator.exe
```

batch programmal fordítható és szerkeszthető.

A fenti példák csak a LEX-YACC, illetve a FLEX-BISON programrendszer elemi használatát próbálták illusztrálni. Csak ízelítőt kívántak adni az eszköz hatékony felhasználhatóságából. A FLEX és BISON programok teljes leírása megtalálható például a

http://www.gnu.org/software/flex/manual/

http://www.gnu.org/software/bison/manual/

internetes címen.