Formális Nyelvek és Automaták

v1.9

Hernyák Zoltán

E másolat nem használható fel szabadon, a készülő jegyzet egy munkapéldánya.

A teljes jegyzetről, vagy annak bármely részéről bármely másolat készítéséhez a szerző előzetes írásbeli hozzájárulására van szükség. A másolatnak tartalmaznia kell a sokszorosításra vonatkozó korlátozó kitételt is. A jegyzet kizárólag főiskolai oktatási vagy tanulmányi célra használható!

A szerző hozzájárulását adja ahhoz, hogy az EKF számítástechnika tanári, programozó matematikus, programtervező informatikus szakos, a 2001/2002-es tanévtől kezdve a tárgyat az EKF TO által elfogadott módon felvett hallgatók bármelyike, kizárólag saját maga részére, tanulmányaihoz egyetlen egy példány másolatot készítsen a jegyzetből.

A jegyzet e változata még tartalmazhat mind gépelési, mind helyességi hibákat. Az állítások nem mindegyike lett tesztelve teljes körűen. Minden észrevételt, amely valamilyen hibára vonatkozik, örömmel fogadok.

Eger, 2006. február 4.

Hernyák Zoltán

1. Fejezet: ABC-k, szavak, nyelvek

1.1 ABC, szó, szó hossza, üres szó.

Def.: Egy (általában véges), nem üres halmazt **ABC**-nek nevezünk. Jele nagy betű, pl: *A,B*.

Def.: Egy A ABC-t, mint halmazt alkotó elemeket **jeleknek** (karaktereknek, betűknek) nevezzük. Jele kis betű, pl: *a,b*.

Def.: Egy *A* ABC jeleiből alkotott kifejezéseket az A ABC fölötti **szónak** (sztringnek) nevezzük. Jele kis görög betű. Pl: α. $\alpha << A$ egy *A* abc fölötti szót jelent.

Def.: Egy A ABC fölötti α **szó hosszán** az őt alkotó jelek számát értjük. Jele: $L(\alpha)$ vagy $d(\alpha)$.

Def.: Egy A ABC fölötti ε szót **üres szónak** nevezzük, ha $L(\varepsilon)=0$.

Megj.: Vagyis ha a szó hossza nulla. Az üres szó jele általában ε.
ε görög betű, epszilon.

1.2 Műveletek szavakkal

Def.: Egy *A* ABC fölötti α és β szavak konkatenációján azt a γ *A* ABC fölötti szót értjük, melyet úgy kapunk, hogy az α szót alkotó jelek mögé írjuk a β szót alkotó jeleket. A konkatenáció jele a **+**.

Megj.: Vagyis ha pl: α="alma" és β="fa" akkor $\alpha+\beta$ = "almafa".
A konkatenáció jelölése során a + jelet nem mindig írjuk ki, $\alpha+\beta = \alpha\beta$.

1.3 A konkatenáció tulajdonságai, szó hatványozása

asszociatív , vagyis $\alpha+(\beta+\gamma) = (\alpha+\beta)+\gamma = \alpha+\beta+\gamma$
nem kommutatív , vagyis $\alpha+\beta \neq \beta+\alpha$
van neutrális elem , vagyis $\alpha+\varepsilon = \varepsilon+\alpha = \alpha$

Legyen adva egy $\alpha << A$ (A ABC fölötti α szó).
Def.: $\alpha^0 = \varepsilon$ (Bármely szó nulladik hatványa az üres szó)
Def.: $\alpha^n = \alpha + \alpha^{n-1}$ ($n \geq 1$) (Bármely szó n. hatványa nem más, mint a szó n-szeres konkatenációja)

1.3 Prefixum, szuffixum

Def.: Ha $\alpha=\beta+\gamma$ akkor a β szót az α szó **prefixumának** (szókezdő részszó) nevezzük.
Def.: Ha $\alpha=\beta+\gamma$ és $\beta\neq\varepsilon$, akkor a β szót az α szó **valódi prefixumának** nevezzük.
Def.: Ha $\alpha=\beta+\gamma$ akkor a γ szót az α szó **szuffixumának** (szó záró részszó) nevezzük.
Def.: Ha $\alpha=\beta+\gamma$ és $\gamma\neq\varepsilon$ akkor a γ szót az α szó **valódi szuffixumának** nevezzük.

1.4 Műveletek ABC-kel

Def.: Ha *A* és *B* két ABC, akkor $A*B := \{ ab \mid a \in A, b \in B\}$. Ezt a műveletet **komplexus szorzásnak** hívják.

Megj.: Vagyis két ABC komplexus szorzatán egy olyan ABC-t értünk, melynek a karakterei kettős jelek, az egyik jel az első ABC-ből származik, a másik jel pedig a második ABC-ből.

Pl.: A:={ a,b }, és B:={0,1}. C:=A∗B:={a0,a1,b0,b1}. Ezek alapján a C ABC fölötti szó pl. az α="a0b0a1", és L(α)=3, ugyanis a fenti szót három C-beli karakter alkotja, az "a0", a "b0", és az "a1".
Ugyanakkor, pl. az "a0aba1" szó nem lehet "C" ABC fölötti szó, ugyanis azt nem lehet csak "C" ABC jeleiből összerakni.

Legyen adva egy "A" ABC.
Def.: $A^0 :=\{ \varepsilon \}$, vagyis minden ABC **nulladik hatványa** az a halmaz, amelynek egyetlen eleme van, az üres jel.
Def.: $A^n := A*A^{n-1}$ ahol $n\geq1$. Vagyis egy ABC **n-edik hatványán** az n-szeres komplexus szorzatát értjük.

Megj.: $A^0=\{\varepsilon\}$ szükséges, mivel $A^1=A*A^0$, és vissza kell kapni A-t!

Megj.: Ez alapján pl. az "A" ABC harmadik hatványa egy olyan ABC, amelynek minden eleme igazából három karakterből áll. Általánosan: az n-edik hatványa egy olyan ABC, melynek minden eleme n karakter hosszú.

1.5 Lezárt, pozitív lezárt

Pl.: ha A={a,b}, akkor pl. A^2={aa,ab,ba,bb}. Ezt a halmazt mint ABC is fel lehet fogni, de sokkal izgalmasabb, ha úgy fogjuk fel, mint az "A" ABC fölötti kettő hosszú szavak halmazát.

Hogy precízebbek legyünk:

Def.: V:={ α | α << A és L(α)=1 }. Vagyis legyen a "V" halmaz az "A" ABC fölötti **egy hosszú szavak halmaza**. Jele V^{*1} << A, vagy röviden V.

Def.: A V halmazon értelmezett kontextusos szorzás V⊗V:={ $\alpha\beta$ | $\alpha\in$ V és $\beta\in$V} műveletén azt a szavakból álló halmazt értjük, amelyeket a meglévő szavakból kapunk úgy, hogy mindegyiket mindegyikkel összefűzzük.

Megj.: A V halmazt valójában az 1 hosszú szavak alkotják. A V⊗V halmazt valójában a kettő hosszú szavak alkotják.

Def.: V^0 := { ε }, és
V^n := V⊗V^{n-1} , n≥1.

Def.: A V* := $\bigcup_{i=0}^{\infty} V^i = V^0 \cup V^1 \cup V^2 \cup ...$ halmazt a "V" **lezártjának** nevezzük.

Megj.: Vagyis elemei a nulla hosszú szó, az egy hosszú szavak, a kettő hosszú szavak, stb...

Def.: A V+ := $\bigcup_{i=1}^{\infty} V^i = V^1 \cup V^2 \cup V^3 \cup ...$ halmazt a "V" **pozitív lezártjának** nevezzük.

Megj.: vagyis $V_* = V+ \cup \{ \varepsilon \}$,

Megj.: a V+ elemei az egy hosszú szavak, a kettő hosszú szavak, stb., így V+-nak nem eleme az üres szó !

Pl.: ha V:={'a','b'}. Akkor V*={ε,'a','b','aa','ab','ba','bb','aaa','aab',...}.
és V+={'a','b','aa','ab','ba','bb','aaa','aab',...}.

Pl.: az $\alpha\in$ V* azt jelenti, hogy α egy tetszőleges hosszú szó, L(α)≥0.

Pl.: az $\alpha\in$ V+ azt jelenti, hogy α egy tetszőleges hosszú szó, de nem lehet üres szó, vagyis L(α)≥1.

Pl.: ha V={ 0, 1 }, ekkor a V* a bináris számok halmaza (és benne van az ε is !).

Pl.: ha V={ 0 }, W= { 1 }, ekkor a $(V \cup W)^* = \{ (01)^n \mid n \in N \}$.

1.6 Formális nyelv

Def.: Legyen V^{*1} << A. Egy $L \subseteq V_*$ halmazt az "A" ABC fölötti **formális nyelvnek** nevezzük.

Megj.: Vagyis a formális nyelv nem más, mint egy adott ABC jeleiből alkotott tetszőleges hosszú szavak halmazának részhalmaza, vagyis a formális nyelv egy adott ABC jeleiből alkotható, meghatározott szavak halmaza.

Megj.: A formális nyelv állhat véges sok szóból, állhat végtelen sok szóból és tartalmazhatja az üres szót is.

Pl.: A:={a,b}, V^{*1}<<A. Ekkor az L:={'a','ab','abb','abbb','abbbb',...} nyelv egy "A" ABC fölötti formális nyelv, mely végtelen sok szót tartalmaz ('a'-val kezdődő, tetszőleges sok 'b'-vel folytatódó szavakból alkotott nyelv).

Pl.: A:={a,b}, V^{*1} << A . Ekkor az L:={'ab','ba','abab','baab','aabb',...} nyelv egy "A" ABC fölötti formális nyelv, mely végtelen sok szót tartalmaz (olyan szavak halmaza, ahol minden szóban annyi 'a'-van, mint 'b').

Def.: Ha L_1,L_2 két formális nyelv, akkor az $L_1 * L_2 := \{ \alpha\beta \mid \alpha \in L_1$ és $\beta \in L_2\}$. Ezt a műveletet nyelvek közötti **kontextus szorzásnak** hívjuk.

Megj.: A kontextus szorzás disztributív: $L_1 * (L_2 \cup L_3) = L_1 * L_2 \cup L_1 * L_3$.

Megj.: A formális nyelv megadható
1. felsorolással (csak véges nyelvek!),
2. a szavakat alkotó szabály szöveges leírásával
3. a szavakat alkotó szabály matematikai leírásával
4. generatív grammatika segítségével

Pl.: Szöveges leírással: "*legyen L_1 a páros számok nyelve*". Vagyis L_1={2,4,6,8,10,12,14,16,...} szavakból álló nyelv. Legyen L_2 a páratlan számok nyelve. Matematikai szabállyal: legyen $L_3 := L_1 * L_2$ nyelv (azon számok nyelve, melyek páratlanok, de tartalmaznak legalább egy páros számjegyet).

Pl.: Matematikai szabállyal: $L := \{ 0^n 1 0^n \mid n \geq 1\}$, vagyis a szám közepén áll egy 1-es, és előtte, mögötte ugyanannyi 0.

Tétel: Ha L_1,L_2,L_3 formális nyelvek $\Rightarrow L_1 \cup (L_2 * L_3) \neq (L_1 \cup L_2) * (L_1 \cup L_3)$.

Tétel: Ha L_1,L_2,L_3 formális nyelvek $\Rightarrow L_1 * (L_2 \cap L_3) = (L_1 * L_2) \cap (L_1 * L_3)$.

2. Generatív grammatikák

2.1 Generatív grammatikák

Megj.: Az eddigiekben a nyelvek egyszerű definíciójával foglalkoztunk. Ennek segítségével egy formális nyelvet úgy definiálhatunk, hogy mint halmazt felfogva felsoroljuk az elemeit, a szavakat. Ezzel a módszerrel azonban kis elemszámú, véges nyelvek definiálhatók csak. A "nagyon sok", ill. végtelen sok szót tartalmazó nyelvek azonban felsorolással nem adhatók meg. A fentiekben is volt példa végtelen nyelv megadására, de a "generáló" szabályt egyszerű szöveges leírással adtuk meg. Nézzük ennek matematikailag helytálló módszerét.

Def: Egy G(V,W,S,P) formális négyest **generatív grammatikának** nevezzük, ahol:

V : a terminális jelek ABC-je,
W : a nemterminális jelek ABC-je,
$V \cap W = \{\}$, vagyis a két halmaz diszjunkt, nincs közös elemük,
$S : S \in W$ egy speciális nemterminális jel, a kezdőszimbólum,
P : helyettesítési szabályok halmaza, ha $A:=V \cup W$, akkor $P \subseteq A^* x A^*$

Pl.: Ha V:={ a, b } és W:={ S, B} akkor P:={ (S,aB), (B,SbSb), (S,aSa), (S, ε } akkor a G:=(V,W,S,P) négyes egy nyelvet ír le. Ennek a nyelvnek a szavai a "a" és "b" jelekből írhatók fel. Az "S" és "B" jelek nem szerepelnek a végső szavakban, csak a szavak létrehozása közben mintegy "segédváltozók", köztes állapotokban használt jelek. Hogy menet közben ne zavarodjunk össze, ezen segédváltozók jelei pl. csupa nagybetű (S,B), míg a nyelv abc-nek elemei csupa kisbetűvel vannak jelölve (a,b). Így ha pl. "aaSabSb" jelsorozatról azonnal látszik, hogy még "nincs befejezve", hiszen még tartalmaz köztes változókat. Ezek neve ezért "nem terminális", vagyis nem befejező jelek, hiszen a szó generálása még nincs kész, nincs befejezve. A "aaabb" jelsorozat viszont már csak "terminális" jeleket tartalmaz, melyek a "befejező", megállító karakterek. Mivel az előző jelsorozat már nem tartalmaz "nemterminális" jelet, csupa terminális jelet, ezért a szó generálása befejezettnek tekinthető.

Megj.: A fenti P halmaz egy Descartes szorzat, melynek minden eleme egy páros, pl. (S,aB) alakú. Az továbbiakban ezt inkább $S \rightarrow aB$ módon fogjuk jelölni. Ennek megfelelően a fenti P halmazt az alábbi módon fogjuk leírni: P:={ ($S \rightarrow aB$), ($B \rightarrow SbSb$), ($S \rightarrow aSa$), ($S \rightarrow \varepsilon$) }.

Megj.: A szavak generálását a kezdőszimbólumtól (startszimbólum) kezdjük el.
"S"
Mivel az "S" egy nemterminális szimbólum, itt nem hagyhatjuk abba, helyettesítsünk tovább.
$S \rightarrow aB$, így most "S"$\rightarrow$"aB" alapján a szavunk. "aB".
"B" egy nemterminális jel, így folytatnunk kell a szó generálását.

B→SbSb szabály figyelembevételével haladhatunk tovább: "aB"→"aSbSb". Ebben a köztes állapotban kétszer is szerepel az "S" szimbólum, mint nemterminális jel, folytatnunk kell a szó generálását. Helyettesítsük tovább az első "S" szimbólumot, mondjuk az S→aSa szabály segítségével. Ennek megfelelően "aSbSb"→"aaSabSb". Többféleképpen folytathatjuk. Így a szabályok segítségével több szót is leírhatunk. Ezért hívják ezt a rendszert generatív grammatikának, hisz segítségével szavakat generálhatunk, állíthatunk elő. Ezek a szavak írják le az általunk kívánt nyelvet.

Megj.: A rendszerben a legfontosabb elem az utolsó, a helyettesítési szabályok halmaza. A rendszer másik három eleme a szabályokat felépítő jelsorozat értelmezésében segít, megadja, hogy melyik a startszimbólum, mely jelek terminálisak, melyek nem.

2.2 Egy lépésben levezethető, több lépésben levezethető

Def.: Legyen adva G(V,W,S,P) generatív grammatika, és $X,Y \in (V \cup W)^*$. $X=\alpha\gamma\beta$, $Y=\alpha\omega\beta$ alakú szavak, ahol $\alpha,\beta,\gamma,\omega \in (V \cup W)^*$. Azt mondjuk, hogy X-ből Y **egy lépésben levezethető**, ha létezik $\gamma \to \omega \in P$. Jele: $X \Rightarrow Y$.

Megj.: X és Y két sztring, amelyben szerepelhetnek terminális és nemterminális karakterek is. X és Y a levezetés két szomszédos mozzanata (egy lépésben levezethető), vagyis X-állapotból Y-állapotba egy lépésen belül el tudunk jutni, ha az X-t felépítő γ jelsorozatot ki tudjuk cserélni a szükséges ω jelsorozatra. Az γ és ω jelsorozat is egy rész-string.

Megj.: Az előző példánál maradva "aSbSb"-ból az "aaSabSb" egy lépésben levezethető. X="aSbSb", vagyis α="a", γ="S" és β="bSb'. Folytatva Y="aaSabSb", vagyis ω=aSa. Mivel létezik $\gamma \to \omega$ szabály, ezért az X-ből egy lépésben az Y levezethető.

Megj: vegyük észre, hogy α és β, a cserélendő szakasz előtti és utáni részszó lehet üres szó is.

Def.: Legyen adva G(V,W,S,P) generatív grammatika, és $X,Y \in (V \cup W)^*$. X-ből Y **n lépésben levezethető** ($n \geq 1$), ha $\exists$ X_1, X_2, ..., $X_n \in (V \cup W)^*$, hogy $X \Rightarrow X_1 \Rightarrow X_2 \Rightarrow X_3 \Rightarrow \ldots \Rightarrow X_n = Y$. Jele: $X \Rightarrow^+ Y$.

Def.: Legyen adva G(V,W,S,P) generatív grammatika, és $X,Y \in (V \cup W)^*$. X-ből Y **levezethető**, ha $X \Rightarrow^+ Y$ vagy X=Y. Jele: $X \Rightarrow^* Y$.

Def.: Legyen adva G(V,W,S,P) generatív grammatika. A G egy $X \in (V \cup W)^*$ **szót generál**, ha $S \Rightarrow^* X$. Ekkor X-et **mondatformának** nevezzük.

Megj.: A mondatformát vegyesen terminális, és nemterminális jelek építhetik fel. Ő a levezetés köztes állapota, formája.

Def.: Legyen adva G(V,W,S,P) generatív grammatika. Ha G egy X szót generál, és $X \in V^*$ (már nincsenek benne nemterminális elemek), akkor X-t **mondatnak** nevezzük.

Def.: Legyen adva G(V,W,S,P) generatív grammatika. Az $L(G) = \{ \alpha \mid \alpha \in V^* \text{ és } S \Rightarrow^* \alpha \}$ nyelvet a G grammatika által **generált nyelvnek** nevezzük.

*Megj.:*A G startszimbólumából levezethető szavak alkotta nyelvet nevezzük a generált nyelvnek. Vagyis az L minden szavát le kell tudni vezetni S-ből, és az S-ből bármely levezethető szó eleme kell legyen a nyelvnek.

Megj.: Vizsgáljuk meg a $P_1 = \{ S \to 00, S \to 11, S \to 0S0, S \to 1S1, S \to \varepsilon \}$ szabályhalmazt, ahol 0,1 terminális szimbólumok. A szabályrendszer segítségével levezethető szavak az alábbiak: L(G)={ "", "00", "11", "0000", "010010", ... }.

A $P_2 = \{ S \to 0S0, S \to 1S1, S \to \varepsilon \}$ szabályrendszer által generált nyelv is ugyanezekből a szavakból áll, de a P_2 szabályrendszer egyszerűbb.

Megj.: Látható, hogy egy nyelvhez több generatív grammatika is tartozhat.

Def.: A $G_1(V,W_1,S_1,P_1)$ és a $G_2(V,W_2,S_2,P_2)$ generatív grammatikák **ekvivalensek**, ha $L(G_1) = L(G_2)$.

Megj.: Vagyis akkor ekvivalensek, ha az általuk generált szavak ugyanazok.

Megj.: Természetesen a két nyelv csak akkor lehet ugyanaz, ha a terminális szimbólumok ugyanazok (ezért nincs V_1 és V_2, csak V). De minden másban különbözhet a két grammatika (melyik milyen és hány terminális jelet alkalmaz, mi a startszimbólum, és milyen szabályokból építkezik).

Megj.: A szabályrendszert felépítő párokat megfigyelve bizonyos rendszert vihetünk a leírásba. Ennek segítségével osztályozhatjuk a leírás módját. Négy fokozatba osztályozhatjuk, a szabályostól a szabálytalanig. Mivel azonban a szabályrendszer jellemzi magát a generált nyelvet is, így a nyelveket is osztályozhatjuk. Természetesen annál jobb, minél szabályosabb a leíró nyelvtan, hisz annál szabályosabb a nyelv is.

A grammatika nem csak betűkből álló ABC alapokon nyugodhat (mely esetben szavakat állít elő), hanem ha az ABC 'elemei' szavak, akkor a grammatika generáló szabályai segítségével mondatokat állíthatunk elő.

Jelen példában az S szimbólumot a <mondat> nemterminális jelölő helyettesíti. A nemterminális jeleket <..> jelölők helyettesítik.

<mondat> ::= <alanyi rész> <állítmányi rész>
<alanyi rész> ::= <főnévi rész> <határozó>
<állítmányi rész> ::= <tárgyi rész> <igei rész:>
<főnévi rész> ::= <névelő> <jelzők> <főnév>
<névelő> ::= ε | a | az | egy
<jelzők> ::= <jelző> | <jelző> <jelzők>
<jelző> ::= ε | hideg | meleg | fehér | fekete | nagy | kis
<főnév> ::= kutya | macska | hús | egér | sajt | tej | víz
<határozó> ::= ε | nappal | éjjel | reggel | este
<tárgyi rész> ::= <főnévi rész>t
<igei rész> ::= eszik | iszik

A fenti generatív grammatika generálja az alábbi 'szót': ***a nagy fehér kutya reggel meleg húst eszik***. De sajnos generálja az alábbi szintaktikailag helyes, bár szemantikailag értelmetlen mondatot is: a ***fekete tej kutyat iszik*** , illetve a nem teljesen kidolgozott szintaktikai szabályok alapján az alábbi mondatot is: ***az fehér egér hideg sajtt eszik***.

2.2 Nyelvtanok Chomsky-féle osztályozása

Legyen adva egy G(V,W,S,P), és

$\alpha,\beta,\gamma \in (V \cup W)^*$	*mondatformák, lehetnek akár ε értékűek is,*
$\omega \in (V \cup W)^+$	*mondatforma, de nem lehet ε,*
$A,B \in W$,	*nemterminális jelek,*
$a,b \in V$.	*terminális jelek.*

Def.: **0-ás** típusú (phrase-structure, **mondatszerkezetű**) nyelvek
Minden szabály $\alpha A\beta \rightarrow \gamma$ alakú.

Def.: 1-es típusú (context sensitive, **környezetfüggő**) nyelvek.
Minden szabály $\alpha A\beta \rightarrow \alpha\omega\beta$, és megengedett az $S \rightarrow \varepsilon$ szabály.

Def.: 2-es típusú (context free, **környezetfüggetlen**) nyelvek.
Minden szabály $A \rightarrow \omega$ alakú, és megengedett az $S \rightarrow \varepsilon$ szabály.

Def.: 3-as típusú (reguláris, **szabályos**) nyelvek.
1. jobb reguláris: Minden szabály **A→a** vagy **A→Ba** alakú.
2. bal reguláris: Minden szabály **A→a** vagy **A→aB** alakú.

Megj.: 0-ás típusú nyelvek esetén lényegében nincs megkötés a szabályokra, ugyanis az, hogy a szabályrendszer bal oldalán állnia kell legalább egy nemterminális szimbólumnak, ez a szabályrendszer használatóságából fakadó megkötés.

Megj.: 1-es típusú nyelvek esetén fontos, hogy a levezetés során a nemterminális szimbólum helyettesítése után a szimbólum környezete (előtte és mögötte levő részszó) ne változzon meg (α és γ). Mivel ekkor a szó lényegében csak hosszabb lehet (lényegében $A \rightarrow \omega$, és ω nem lehet üres szó, így minden nemterminális jelet legalább egy hosszú jelsorozatra kell cserélni), ezért ezeket a nyelvtanokat hossz nem csökkentő nyelvtanoknak nevezzük. Ez alól kivételt képez az "S" startszimbólum, amelyet lehet üres szóra cserélni, ha az a szabály szerepel a szabályrendszerben. De a hossz nem csökkentés ekkor is fennáll, hiszen az S előtti és utáni résznek meg kell maradnia.

Megj.: Az ilyen nyelvtanokban, ha a szó eleje már kialakult (a szó elején már csak terminális szimbólumok szerepelnek), az már nem fog megváltozni.

Megj.: 2-es típusú nyelvek esetén fontos, hogy a szabályok bal oldalán csak egyetlen darab nemterminális jel állhat, amelyet valamilyen, legalább egy hosszú jelsorozatra kell cserélni. Itt nem kell figyelni a jel előtti és mögötti környezetre, csak a jelre. Ha végiggondoljuk, ezek a nyelvtanokat is hossz nem csökkentő nyelvtanoknak nevezhetjük.

Megj.: 3-as típusú nyelvek esetén fontos, hogy a szó felépítése egy irányban halad. Először a szó bal eleje, és az irány nem változik, hiszen a csere során újabb terminális szimbólumot teszünk a generált szóba, és egy újabb nemterminálisat a terminális jel jobb oldalára. Itt a köztes állapotokban nemterminális jelből egyszerre mindig csak egy lehet, és az mindig a köztes szó jobb végén található.

Megj.: Itt nincs kitétel az $\rightarrow\varepsilon$ szabályra, mert a generálás bármikor abba hagyható egy véghelyettesítéssel.

Megj: Fontos, hogy vagy bal reguláris, vagy jobb reguláris szabályok szerepeljenek csak. Ha mindkét típusból fordul elő egyszerre, akkor a nyelvtan nem reguláris.

Miért érdekes a Chomsky-féle osztályozás?

Az algoritmusok - mint ismeretes - problématípusok (problémaosztályok) megoldására szolgáló lépéssorozatok. A generatív grammatikákkal kapcsolatban több, számítógéppel (esetleg) megoldható probléma merül fel, melyekre algoritmusok készíthetőek. Nagyon fontos annak eldöntése, hogy ha egy konkrét generatív grammatika egy konkrét problémájának megoldására kifejlesztett algoritmus alkalmazható-e más grammatika ugyanazon problémájára, illetve mennyi módosítással alakítható át.

Amennyiben a két grammatika ugyanazon Chomsky-osztályba tartozik, úgy egy jól megfogalmazott algoritmus váza a paraméterektől eltekintve elvileg alkalmazható lesz.

Másik fontos indok, ami miatt fontos ismerni egy grammatika Chomsky-osztályát, hogy igazolható bizonyos problémakörökről, hogy nem készíthető hozzá általános érvényű megoldó algoritmus.

2.3 Állítások a Chomsky-osztályokkal kapcsolatban

Tétel: A G(V,W,S,P) egy bal reguláris nyelvtan $\Leftrightarrow$ ha $\exists$ G'(V,W',S',P') jobb reguláris nyelvtan, hogy L(G)=L(G').

Def.: Egy K nyelv *i* típusú ($i \in \{ 0, 1, 2, 3\}$), ha $\exists$ olyan G(V,W,S,P) generatív grammatika, amelyre L(G)=K, és G szabályrendszere *i*. Chomsky osztályú. Jele: K^i.

Megj.: Pl: K^2 értelmezése: K egy környezetfüggetlen nyelv, mert létezik olyan grammatika, amely 2. Chomsky osztályú, és a K nyelvet generálja.

Megj.: Mivel egy nyelvhez több generáló nyelvtan is tartozhat, azok elképzelhető, hogy különböző Chomsky osztályba tartoznak. Ezért a K^2 olvasata: K legalább környezetfüggetlen nyelv. De ha felfedezünk egy olyan nyelvtant, amely reguláris, és ugyanezt a nyelvet generálja, akkor K-ról be lehet bizonyítani, hogy reguláris nyelv. Minél nagyobb sorszámú osztályba tudunk sorolni egy nyelvet, annál egyszerűbb a nyelvtan, annál egyszerűbb a nyelv.

Megj.: Megfigyelhetjük, hogy a generáló nyelvtanok szabályaira egyre szigorúbb megkötések vannak, de a megkötések nem ütik egymást. Pl. ha egy szabályrendszer megfelel a 2. osztály megkötéseinek, akkor megfelel az 1. és a 0. osztály megkötéseinek is (azok megengedőbbek). Ezért $L^3 \subseteq L^2 \subseteq L^1 \subseteq L^0$ állítás igaz.

Tétel: Ha egy L nyelv $i \in \{0,1,2,3\}$ típusú $\Rightarrow L \in \{\varepsilon\}$ és $L \setminus \{\varepsilon\}$ is i típusú.

Megj.:Vagyis abban, hogy egy nyelv milyen osztályú, az ε-nak nincs semmilyen szerepe.

Tétel: Ha L_1 és L_2 reguláris nyelv $\Rightarrow L_1 \cup L_2$ is reguláris.

Biz: (vázlatosan) Ha L_1 reguláris, akkor tartozik hozzá egy $G_1(V,W_1,S_1,P_1)$ reguláris nyelvtan. Hasonlóképpen, L_2-hoz is tartozik valamely $G_2(V,W_2,S_2,P_2)$ reguláris nyelvtan. Az általánosság megsértése nélkül feltételezhetjük, hogy W1 és W2 diszjunkt halmazok, vagyis nincs közös elemük (ha lenne, akkor például indexekkel átjelölhetjük a közös elemeket. Szintén az általánosság megsértése nélkül feltételezhetjük, hogy G_1 és G_2 is jobb-reguláris.

$L_1 \cup L_2$ nyelvben olyan szavak fordulnak elő, amelyek vagy L_1-nek, vagy L_2-nek eleme. Ha L_1-nek eleme, akkor van olyan szabály, amely S_1->… alakúan elkezdi képezni a szabályt. A jobb-reguláris nyelvtani szabályok szerint ezen szabály vagy S_1 -> a, vagy S_1 -> B_1a alakú. Hasonlóan, az L_2-beli szavakra van S_2->a vagy S_2->B_2a alakú kezdő lépés.

Az állítás, mely szerint $L_1 \cup L_2$ is reguláris könnyen bizonyítható, ha sikerül egy olyan reguláris nyelvtant definiálni, amely épp az $L_1 \cup L_2$ nyelv szavait generálja.

$G_3(V,W_3,S_3,P_3)$ legyen egy olyan generatív grammatika, ahol

$W_3 := W_1 \cup W_2 \cup \{ S_3 \} / \{S_1, S_2\}$, vagyis legyenek benne a W1 és W2 terminális jelei, kivéve az S1 és S2 eredeti startszimbólumokat, és vegyünk bele egy új jelet, az S3-t, amely majd a G3 startszimbóluma lesz.

$S_3 \in W_3$

P_3 szabályait pedig építsük fel úgy, hogy vegyük az összes szabályt a P_1-ből, de minden S_1-t cseréljünk le S_3-ra, valamint hasonlóan a P_2 minden szabályát vegyük át, de S_2-t cseréljük le S_3-ra. Ez biztosítja, hogy a fent definiált W_3 nem terminális szimbólum-halmaz megfelelő lesz.

A fenti G_3 pontosan az $L_1 \cup L_2$ beli szavakat produkálja, hiszen a fentiek szerint elkészített P_3 szabályrendszer az S_3-ból kiindulva tudja generálni mind az L_1-beli, mind az L_2-beli szavakat. A P3-beli szabályrendszer alakja viszont értelemszerűen szintén jobb-reguláris, hiszen az S_1->S_3 és S_2->S_3-ra vonatkozó cseréktől eltekintve (amely ezt nem változtatja meg) ezek a szabályok mindegyike jobb-reguláris volt.

Tétel: Ha L_1 és L_2 reguláris nyelv $\Rightarrow L_1 \cap L_2$ is reguláris.

Tétel: Ha L_1 és L_2 reguláris nyelv $\Rightarrow L_1 * L_2$ is reguláris.

Tétel: Ha L reguláris nyelv $\Rightarrow L_{1*}$ is reguláris.

Biz: (vázlatos) Ha L reguláris nyelv, akkor létezik olyan G(V,W,S,P) grammatika, amely reguláris, és éppen az L nyelvet generálja. Az általánosság megsértése nélkül feltételezhetjük, hogy P jobb-reguláris.
Ezen P szabályrendszer szabályai S->a, S->Ba alakú szabályokat tartalmaz.

Legyen G'(V,W,S,P') olyan generatív szabályrendszer, ahol a V,W,S komponensek az eredeti G grammatikákból származnak. A P' szabályrendszert pedig képezzük oly módon, hogy vegyük az eredeti P szabályrendszer szabályait, és vegyünk még bele pluszban szabályokat: minden S->a alakú szabály esetén vegyük bele még S->aS szabályokat is.

Ezen kiegészítés mellett a G' grammatika marad jobb-reguláris, és generálja az $L1_*$ nyelv szavait. Ez utóbbiak ugyanis olyan szavak, amelyek L1-beli szavak n-szeres (n=1,2,…) konkatenációjából tevődnek össze.

Tétel: Minden véges nyelv 3-as típusú (reguláris).

Tétel: Létezik olyan 3-as típusú (reguláris) nyelv, amely nem véges.

2.4 Chomsky-féle normál alak

Def: Egy G(V,W,S,P) generatív grammatika Chomsky-féle normál alakban van, ha minden P-beli szabály **A→BC** vagy **A→a** alakú (ahol $A,B,C \in W$, $a \in V$).

Tétel: Minden, legalább 2-es típusú (környezetfüggetlen) Chomsky-osztályú nyelvtan átalakítható Chomsky-féle normál alakra.

2.5 Feladatok

1. feladat: A 0,1 jelekből alkossunk olyan szavakat, amelyek páros hosszúak, és szimmetrikusak.

Megoldás: P := { S→0S0, S→1S1, S→ε }

Megj.: a fenti megoldás 2-es típusú.

Megoldás: P := { S→0B, B→S0, S→1C, C→S1, S→ε }

Megj.: a fenti megoldás is 2-es típusú.

---

2. feladat: A 0,1,…,9 jelekből alkossunk pozitív egész számokat.

Megoldás: P := { S→0S, S→1S,…, S→9S, S→ε }

Megj. a fenti megoldás 2-es típusú.

Megoldás: P := { S→0S, S→1S,…, S→9S, S→0, S→1,…, S→9 }

Megj.: a fenti megoldás 3-as típusú.

---

3. feladat: A 0,1,…,9 jelekből alkossunk pozitív egész számokat, de azok ne kezdődjenek 0-al.

Megoldás: P := { S→1B, S→2B,…, S→9B,
S→1, S→2, …, S→9,
B→0, B→1,…, B→9,
B→0B, B→1B, …, B→9B }

Megj.: a fenti megoldás 3-as típusú.

---

4. feladat: A **x, *, +** és a **( )** segítségével alkossunk komplex matematikai kifejezéseket.

Megoldás: P := { S→A, S→S+A, A→A*B, A→B, B→X, B→(S) }

Megj.: a fenti megoldás 2-es típusú.

---

5. feladat: A 0,1 jelekből alkossunk olyan szavakat, amelyek tetszőleges (akár nulla) darab 0-al kezdődnek, és tetszőleges (akár nulla) darab 1-essel végződnek.

Megoldás: P := { S→0S, S→1A, A→1A, A→1, S→0, S→1 }

Megj.: a fenti megoldás 3-as típusú.

---

6. feladat: A 0,1 jelekből alkossunk olyan szavakat, amelyek tetszőleges (akár nulla) darab 0-al kezdődnek, és legalább ugyanennyi (vagy több) darab 1-essel végződnek.

Megoldás: P := { S→0S1, S→S1, S→ε }

Megj.: a fenti megoldás 2-es típusú.

Megoldás: P := { S→0B, B→S1, S→ε }

Megj.: a fenti megoldás 2-es típusú.

7. feladat: A 0,1 jelekből alkossunk olyan szavakat, amelyekben tetszőleges pozíción bár, de összesen páratlan darab 1-es szerepel.

Megoldás: P := { S→0S, S→1A, S→1, A→1S, A→0A, A→0 }

Megj.: a fenti megoldás 3-as típusú.

---

8. feladat: Az 1 jelből alkossunk páratlan darab betűből álló szavakat.

Megoldás: P := { S→1A, A→1S, S→1 }

Megj.: a fenti megoldás 3-as típusú.

---

9. feladat: Az 0,1 jelekből alkossunk legalább kettő darab 1-essel kezdődő szavakat.

Megoldás: P := { S→1A, A→1B, A→1, B→1B, B→0B, B→0, B→1 }

Megj.: a fenti megoldás 3-as típusú.

---

10. feladat: Az 0,1 jelekből alkossunk legalább kettő darab 1-essel végződő szavakat.

Megoldás: P := { S→0S, S→1S, S→1A, A→1 }

Megj.: a fenti megoldás 3-as típusú.

2.6 Nyelvek megadása más eszközökkel

2.6.1 Reguláris kifejezésekkel

A reguláris kifejezések használata széles körben elterjedt az informatikában. Reguláris kifejezések a reguláris nyelvek definiálásának eszköze, tehát nyelvleíró, szintaxisleíró eszköz.

Def: Legyen A egy abc, V << A. Reguláris halmaznak tekintjük az alábbiakat:

- ø üres halmaz
- { ε } az egyelemű, csakis az ε-t tartalmazó halmaz
- { a| a ∈ V } Egyetlen egy hosszú szót tartalmazó halmazt

Megj: A fenti halmazok formális nyelveknek tekinthetőek, hiszen szavak halmaza. Könnyű belátni, hogy a fentiek mindegyike 3-as típusú, vagyis reguláris nyelv.

Tétel: Ha P és Q egy-egy reguláris halmaz, akkor reguláris halmazok az alábbiak is:

a, **P ∪ Q** melyet P+Q –val fogunk jelölni
b, **{ ab | a∈P, b∈Q}** melyet P•Q-val fogunk jelölni
c, **P***

Biz: A fentiek is reguláris halmazok (reguláris nyelvek) maradnak, lásd a reguláris nyelvekre vonatkozó tételeket a 2.3-as fejezet végén.

Megj: a reguláris halmazokkal való halmazműveleteket reguláris kifejezések alkotásának nevezzük, melyet a szokásos halmazműveleti jelek helyett használt összeadás és szorzás szimbólumok is sejtetnek.

Pl: L := (**+** + **-** + ε)• (**0**+**1**+**2**+**3**+**4**+**5**+**6**+**7**+**8**+**9**) • (**0**+**1**+**2**+**3**+**4**+**5**+**6**+**7**+**8**+**9**)*
Ezen leírás szerint a szó első karaktere vagy + vagy – vagy üres jel, második karaktere valamilyen 10-es számrendszerbeli számjegy, és tetszőleges számú számjeggyel folytatódhat, vagyis ez nem más, mint a pozitív vagy negatív vagy előjel nélkül felírt 10-es számrendszerbeli számok nyelve.

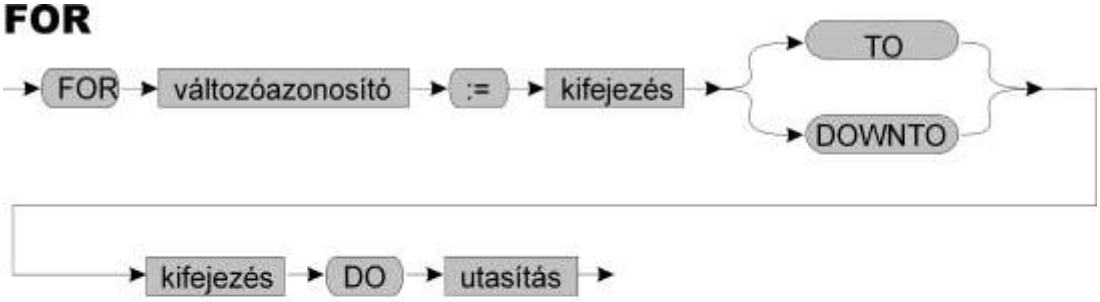
Megj: A fenti példában az egyszerűség kedvéért az egyelemű halmazokat a halmazképző jelek nélkül írtam fel. A forma teljes helyességgel az alábbi módon kezdődne: ({**+**} + {**-**} + {ε})• ({**0**}+{**1**}+{**2**}+{**3**}...
Vagy (minden véges nyelv reguláris):

A := { +,-,ε}
B := {0,1,2,3,4,5,6,7,8,9}
L := A•B•B*

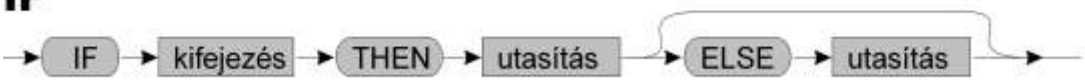
2.6.2 Szintaxisdiagrammal

A szintaxisdiagramm nem precíz matematikai eszköz a leírásra, de látványos és gyors eligazodást tesz lehetővé a nyelv szintaxisában. Jellemző felhasználási területe a programozási nyelvek szintaxisának leírása. A **gömbölyű sarkú** téglalapokban a terminális elemek megadása történik, a **szögletes sarkú** téglalapokban pedig a nemterminális elemek adhatóak meg. A teljs szintaxisleírás esetén természetesen a nemterminális elemek felépítését is meg kell adni egy későbbi szabályban mindaddig lebontva a leírást, amíg eljutunk a legegyszerűbb elemekig, amelyek lerajzolásához már csak terminális elemeket használunk fel.

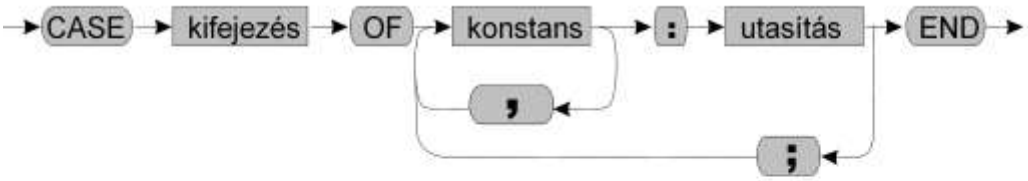
FOR



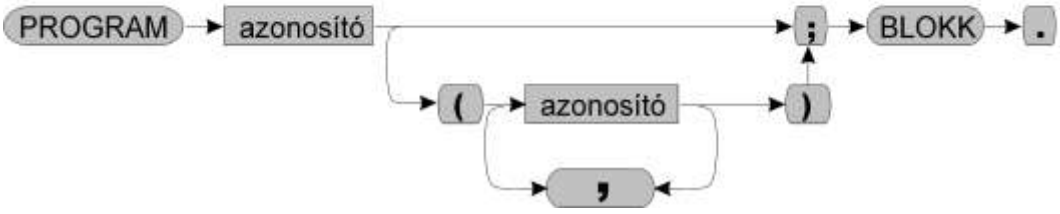
IF



CASE



PROGRAM



2.6.3 EBNF (Extended Bachus-Noir forma)

Az EBNF forma is szinatxisleíró eszköz, mely az előzőekben ismertetett szintaxisdiagrammhoz eltérően nem grafikus, hanem karakteres jelölésrendszert használ, így kedveltebb, szövegszerkesztővel könnyebben elkészíthető módszer. E miatt gyakrabban is találkozunk ezen leíró eszköz alkalmazásával mint az előzőek közül bármelyikkel.

Gyakran használt programozási nyelvek szintaxisának leírásával, de egyéb helyeken is használható, pl. parancssori (command-line) operációs rendszerbeli parancsok és kapcsolóinak leírására is használható.

Speciális szimbólumok:

::==	: szabály értékadás	
.	: szabály vége	
{ … }	: tetszőleges számban ismétlődhet	(iteráció)
[…]	: 1-szer vagy 0-szor ismétlődhet	(opció)
(…)	: egységbe foglalás	
|	: alternatív választó szimbólum	(alternatíva)
" … "	: terminális jelsorozat	

Példák:

AdditívMűveletiJel ::== " **+** " | " **-** " | " **or** ".
EgyszerűKifejezés ::== [Előjel] Tag { AdditívMűveletiJel Tag }.
Kifejezés ::== EgyszerűKifejezés [RelációsJel EgyszerűKifejezés]
ÉrtékadóUtasítás ::== (Változó | Függvényazonosító) " := " Kifejezés.
CaseUtasítás ::== " **CASE** " EsetIndex " **OF** " Eset { ";" Eset } [;] " **END** ".
IfUtasítás ::== "**IF**" LogikaiKifejezés "**THEN**" Utasítás ["**ELSE**" Utasítás].
ForUtasítás ::== "**FOR**" Ciklusváltozó " := " Kezdőérték ("**TO**" | "**DOWNTO**")
Végérték "**DO**" Utasítás.
FeltételesUtasítás ::== IfUtasítás | CaseUtasítás.

3. Fejezet: Automaták

Az eddigiekben azzal foglalkoztunk, hogy hogyan lehet azt megadni, hogy hogyan kell helyes szavakat, mondatokat generálni. A továbbiakban azzal foglalkozunk, hogy ha van egy szavunk (mondatunk), el kell dönteni, hogy helyes-e, vagy sem.

Ez a számítástechnikai nyelvészet alapvető problémája. A kérdéskör problémájának érdekességéhez említsük meg, hogy …

- ha az ABC elemei karakterek, és a nyelvtani szabályok a programozási nyelvünk kulcsszavainak előállítását írják le, úgy a forráskód elemeiről egyesével el lehet dönteni, hogy szintaktikailag helyesek-e.
- ha az ABC elemei a nyelvünk kulcsszavai, és a nyelvtani szabályok a kulcsszavakból felépíthető utasítások felépítését írják le, úgy a forráskód sorairól, komplex utasításairól el lehet dönteni, hogy szintaktikailag helyesek-e.
- ha az ABC elemei komplex utasítások, és a nyelvtani szabályok a programozási nyelvünk felső szintű szintaktikai szabályait írják le, úgy az utasításaink alapján eldönthető, hogy a programunk struktúrájában is helyes-e.

Az alábbiakban részletezett konstrukciók (automaták) tehát a programozási nyelvek szintaktikai elemzőinek is tekinthetőek. Összesen 4 fajta automatát tanulmányozunk majd, a 4 Chomsky osztálynak megfelelően. Minden bonyolultabb (megengedőbb) Chomsky osztályhoz egyre bonyolultabb automata tartozik majd. Amennyiben a nyelvtanunk egy programozási nyelvet ír le, úgy azt az automatát (szintaktikai elemző algoritmust) kell használni az adott programozási nyelv fordítójának – amely Chomsky osztályba az adott programozási nyelvünk a generáló szabályrendszere szempontjából tartozik.

Nyilván logikus észrevételnek tűnik, hogy amennyiben a nyelvünk szabályrendszere egyszerűbb (amely természetesen a programozónak is kedvező), úgy a fordítóprogramunk is egyszerűbb lesz.

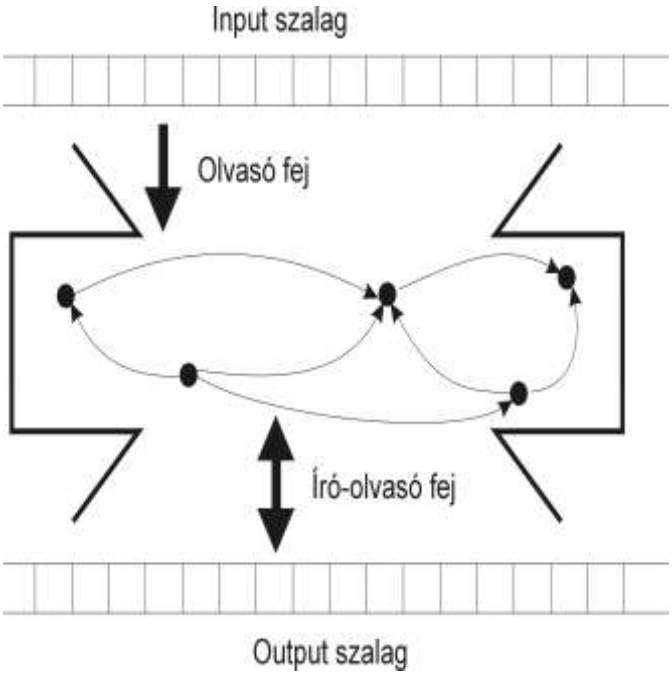
Általános esetre visszatérve ez a probléma (hogy egy adott szó eleme-e egy generáló szabályrendszerrel adott nyelvnek) nem egyszerű probléma. Legalábbis hatékony megoldása nem feltétlenül van a kérdéskörnek.

Az alábbiakban megoldó algoritmusokról, és az algoritmusok működéséhez szükséges típusokról, segédváltozókról lesz szó.

Egyes nyelvek esetén a felismerés egyszerű lépéssorozat, mely véges számú lépésben garantáltan véget ér, és garantált választ ad. Bonyolultabb nyelvek esetén ez a jó eset nem garantált, sőt, 0-s típusú nyelvek esetén nem is létezik algoritmus erre a feladatra, csak módszer (amely nem garantáltan vezet eredményre véges sok lépésben). Többek között ezért is kell törekedni a minél egyszerűbb nyelvmegadásra.

Az automata egy olyan konstrukció, amely egy input szóról el képes dönteni, hogy az helyes-e vagy sem.

1. Létezik egy **input szalag**ja, amely cellákra van osztva, és minden cellában egy-egy karakter van. Az input szalag általában véges, és épp olyan hosszú, amilyen hosszú az input szó.
2. Az input szalaghoz egy **olvasó fej** tartozik, amely mindig egy cella fölött áll, és ezen aktuális cellából képes kiolvasni a karaktert. Az olvasó fej lépked az input szalag cellái között egyesével balra-jobbra.
3. Létezik egy **output szalag**ja, amely cellákra van osztva, és minden cellában egy-egy karakter lehet. Az output szalag lehet véges, vagy végtelen. Azokban a cellákban, amelyekbe még nem írt semmit az automata, egy speciális karakter, a BLANK jel áll.
4. Az output szalaghoz egy író-olvasó fej tartozik. Ennek segítségével az automata egy jelet írhat az aktuális cellába, vagy olvashat belőle. Az író-olvasó fej léptethető a cellák fölött balra-jobbra.
5. Az automatának **belső állapota**i vannak, melyek között az automata lépkedhet egy megadott séma alapján. Az automata egyszerre mindig pontosan egy állapotban van (aktuális állapot).



3.1 Véges automaták

3.1.1 Véges automaták definíciója

A véges automaták a legegyszerűbb automata-csoport. Nincs output szalagjuk, sem író-olvasó fej. Az input szalag hossza véges, épp olyan hosszú, mint az input szó (ugyanannyi cellából áll, amennyi a szó hossza).

Def.: Egy G(K, V, δ, q_0, F) formális ötöst véges automatának nevezünk, ahol:
K: az automata belső állapotainak halmaza (véges!)
V: az input ABC (az input szalagon milyen jelek fordulhatnak elő)
δ: állapotátmeneti függvény, $\delta \subseteq KxV \rightarrow K$
$q_0 \in K$: speciális belső állapot, a kezdőállapot
$F \subseteq K$: befejező állapotok halmaza

Megj: Az automata működése az alábbi lépésekre osztható
Induláskor:
1. az automata q_0 kezdő állapotban van
2. az input szalagon az input szó jelei vannak felírva, balról-jobbra feltöltve, folytonosan.
3. az olvasó fej az input szalag legbaloldalibb cellája fölött áll

Működési ciklus:
4. az olvasó fej beolvassa az aktuális jelet az input szalagról
5. ezen jel, és az aktuális belső állapot ismeretében, a δ függvényben megfogalmazottak szerint újabb belső állapotba vált
6. az olvasó fejet lépteti egyel jobbra

Megállás:
7. az automata megáll, ha az olvasó fej lelép a szalagról (jobbra)

Megj.: Amennyiben az automata megáll, meg kell vizsgálni, hogy milyen az aktuális állapota. Amennyiben az F-beli (elfogadó) állapot, akkor az automata a szót elfogadja. *A megállás és elfogadás-t még később pontosítjuk.*

Megj.: Mivel az automata minden ciklusléptében olvas egy jelet az input szalagról, és mindig jobbra lép, ezért az automata biztosan megáll n lépés végrehajtása után.

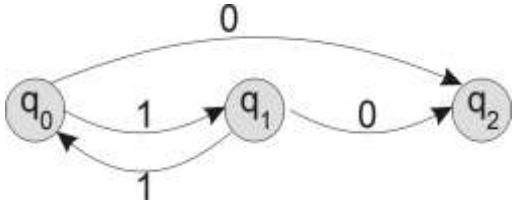
Megj.: A δ függvény egy kétváltozós függvény. Egy aktuális állapot (K) és egy input jel (V) alapján megadja, hogy milyen új állapotba (K) lépjen. $\delta \subseteq KxV \rightarrow K$.

A δ függvény megadható:

a. táblázattal

δ	K	V	K
	q_0	0	q_2
	q_0	1	q_1
	q_1	0	q_2

b. gráffal



3.1.2 Egy teljes és determinisztikus példa

Példa: (8. feladat): P := { S→1A, A→1S, S→1 } a nyelv grammatikája (páratlan számú 1-esből álló szavak). Feladat, olyan automatát konstruálni, amely a generálás szabályainak megfelelő szavakat elfogadja, a többit elutasítja.

K:={ q_0, q_1 }, V:={ 1 }, F := { q_1 }.

δ	K	V	K
	q_0	1	q_1
	q_1	1	q_0

Vizsgáljuk meg a fenti automatát működés közben, legyen az input szó **111.** Az automata q_0 kezdőállapotban kezd. Beolvasásra kerül az első **1**-es. Az automata átlép q_1 állapotba, és a fej jobbra lép. Beolvasásra kerül a második **1**-es. Az automata átlép q_0 állapotba, és a fej jobbra lép. Beolvasásra kerül a harmadik **1**-es. Az automata átlép q_1 állapotba, és a fej jobbra lép. Mivel a fej lelépett a szalagról, ezért az automata megáll. Mivel q_1 állapotban állt meg, és $q_1 \in F$, így az automata felismerte a szót.

Amennyiben az input szó **1111** lenne, így az automata q_0 állapotban fejezi be a működését, amely nem elfogadó állapot, így az automata a szót elutasítja.

3.1.3 Parciális működés

Példa: (9. feladat): P := { S→1A, A→1B, A→1, B→1B, B→0B, B→0, B→1 } a nyelv grammatikája (legalább két 1-essel kezdődő szavak). Feladat, olyan automatát konstruálni, amely a generálás szabályainak megfelelő szavakat elfogadja, a többit elutasítja.

K:={ q_0, q_1, q_2 }, V:={ 0,1 }, F := { q_2 }.

δ	K	V	K
	q_0	1	q_1
	q_1	1	q_2
	q_2	1	q_2
	q_2	0	q_2

Ha megvizsgáljuk az automatát működés közben, láthatjuk, hogy az első két egyes beolvasása során q_0-ból q_1-be, q_1-ből q_2-be lép. Utána már akár nullát, akár egyest olvas a szalagról, mindenképpen q_2-ben marad, ami egyben az elfogadó állapot is.

Felmerül a kérdés, mi történik akkor, ha a szalagon levő szó 0-al kezdődik. Ekkor a q_0 állapotban 0-t olvasva az automata nem tud mit lépni, mert a δ függvény erre a lehetőségre nem tartalmazza a lépést. Ekkor az automata rögtön megáll.

Amennyiben a szó közepén áll meg az automata a fenti oknál fogva, úgy az automata nem fogadta el a szót.

A fentihez hasonló esetekben, amikor a δ nem minden eshetőségre ad egyértelmű lépést, úgy a δ leképezést **parciálisnak** (részlegesnek) nevezzük. Ellenkező esetben δ-t **teljesnek** nevezzük.

3.1.4 Nemdeterminisztikus működés

Példa: (10. feladat): P := { S→0S, S→1S, S→1A, A→1 } a nyelv grammatikája (legalább két 1-esre végződő szavak). Feladat, olyan automatát konstruálni, amely a generálás szabályainak megfelelő szavakat elfogadja, a többit elutasítja.

K:={ q_0, q_1, q_2 }, V:={ 0,1 }, F := { q_2 }.

δ	K	V	K
	q_0	0	q_0
	q_0	1	q_0
	q_0	1	q_1
	q_1	1	q_2

Ha megvizsgáljuk az automatát működés közben, láthatjuk, hogy q_0 állapotban 1-est olvasva nem egyértelmű, hogy mi a következő állapot. A δ értelmében ez a q_0, és q_1 is lehet. Ennek oka, hogy menet közben előfordulhat, hogy 1-es szerepel a szóban, amely nem biztos, hogy a szó végét jelzi még. Ha nem, akkor q_0-ban kell maradni. Az utolsó előtti 1-est olvasva kell átlépni q_1-be, majd onnan az újabb 1-est olvasva q_2-be. Mivel ekkor kell elfogyni a szónak, és q_2 egyben elfogadó állapot is.

Amennyiben a δ nem egyértelmű (mert adott állapothoz és input jelhez két (vagy több) különböző új belső állapotot is rendel, úgy azt mondjuk, hogy δ **nemdeterminisztikus**. Ellenkező esetben δ **determinisztikus**.

Nemdeterminisztikus esetben az automata maga dönti el (véletlenszerűen), hogy melyik lehetőséget választja a továbblépéshez. Ez azt is jelenti, hogy kétszer betáplálva ugyanazt a szót, az automata nem ugyanúgy reagál.

Pl. az 110011 input szóra az automata a q_0, q_0, q_0, q_0, q_1, q_2 sorozatot is választhatja,(és elfogadja a szót), de maradhat végig q_0-ban is (és nem fogadja el), de q_0, q_1, q_2 sorozatba is kezdhet, majd parciális okoknál fogva leáll (és nem fogadja el a szót). Mivel azonban a szó jó, megfelel a nyelv szabályainak, így az automata el kellene hogy fogadja a szót. Hogy a működés se változzon, az elfogadás is eredményes lehessen, ezért nemdeterminisztikus δ esetén azt mondjuk, hogy az automata akkor fogadja el a szót, ha létezik olyan eset (sorozat), amelyben elfogadja.

3.1.4 Automaták ekvivalenciája

Def.: Egy A(K,V, δ, q_0, F) véges automata egy L nyelvet felismer, ha minden $\alpha \in$L-re az automata megáll, és a szót felismeri, és minden $\beta \notin$L-re az automata a szót elutasítja.

Def.: Egy A(K,V, δ, q_0, F) véges automata által felismert szavakból alkotott L nyelvet az automata által felismert nyelvnek nevezzük.

Def.: Egy A(K,V, δ, q_0, F) véges automata egy A'(K',V, δ', q_0', F') automatával ekvivalens, ha ugyanazt a nyelvet ismeri fel.

Tétel: Egy A(K,V, δ, q_0, F) **parciális** véges automatához mindig konstruálható olyan A'(K',V, δ', q_0', F') **teljes** véges automata úgy, hogy a két automata **ekvivalens**.

Biz.: Azon múlik, hogy az automatát ki kell egészíteni egy új állapottal, és minden le nem kezelt esetet úgy kell lekezelni, hogy az automata átmenjen ebbe az új állapotba, és utána tetszőleges jel beolvasása esetén maradjon ebben az állapotban, és ez az állapot ne legyen elfogadó állapot.

Tétel: Egy A(K,V, δ, q_0, F) nemdeterminisztikus véges automatához mindig konstruálható olyan A'(K',V, δ', q_0', F') determinisztikus véges automata úgy, hogy a két automata **ekvivalens**.

Megj.: A fenti két tétel értelmében tehát minden automata visszavezethető teljes és determinisztikus működésre.

3.1.5 Automata konfigurációja

Def.: Egy A(K,V, δ, q_0, F) véges **automata konfigurációja** egy (α,q) formális kettes, ahol α az input szalagon még hátra levő szó, q pedig az aktuális belső állapot.

Megj.: Az automata konfigurációja felfogható úgy is, hogy amennyiben kikapcsoljuk az automatát működés közben, majd visszakapcsoljuk, és vissza akarjuk állítani a kikapcsoláskori állapotot, akkor az input szalagra vissza kell írni a még hátra levő szót, majd vissza kell állítani a kikapcsoláskori belső állapotot. Érthető, hogy az input szóból már beolvasott rész ezen szempontból már érdektelen, hiszen az olvasó fej csak jobbra mozoghat, így a már beolvasott részről többé nem olvas az automata.

Megj.: Az automata induláskori konfigurációja (ω, q_0), ahol ω a teljes input szó.
Az automata feldolgozás közbeni köztes konfigurációja valamely (α, q)
Normál működés végi (záró) konfigurációja (ε,q').
Egy záró konfiguráció **elfogadó**, ha q'$\in$F.

Az automata a működése gyakorlatilag nem más, mint konfigurációk sorozata. A kezdő konfigurációra alkalmazva a δ függvényt megkapjuk a következő konfigurációt, amelyre újra és újra alkalmazva kapjuk meg a konfiguráció-sorozatot.

Pl: *Legyen a véges automatánk a*: $K:=\{ q_0, q_1, q_2 \}$, $V:=\{ 0, 1\}$, $V:=\{ 0,1 \}$, $F := \{ q_2 \}$.

δ	K	V	K
	q_0	0	q_0
	q_0	1	q_0
	q_0	1	q_1
	q_1	1	q_2

A felismerendő input szó legyen ”**1011**”

1. lehetséges konfiguráció-sorozat:

($”1011”,q_0$) → ($”011”,q_0$) → ($”11”,q_0$) → ($”1”,q_1$) → (ε,q_2)

Mivel a záró konfigurációban a hátralévő input szó ε, és $q_2 \in F$, így ez esetben az input szót az automata felismerte, elfogadta.

2. Lehetséges konfiguráció-sorozat:

($”1011”,q_0$) → ($”011”,q_1$) → megállás

Ez esetben a $”0”,q_1$ párosra nincs a delta fv értelmezve, így az automata a feldolgozás közepette megáll. Mivel nem ε van hátra az input szóból, így érdektelen, hogy a megállási állapot eleme-e az F halmaznak! Az automata ezen menet az input szót elutasítja. Ugyanakkor mivel az automata nemdeterminisztikus, így ezen egy esetből nem szabad messzemenő következtetést levonni!

Pl: *Legyen a véges automatánk a*: $K:=\{ q_0, q_1, q_2 \}$, $V:=\{ 0, 1\}$, $V:=\{ 0,1 \}$, $F := \{ q_2 \}$.

δ	K	V	K
	q_0	0	q_0
	q_0	1	q_0
	q_0	1	q_1
	q_1	1	q_2
	q_2	0	q_0

A felismerendő input szó legyen ”**10110**”

1. lehetséges konfiguráció-sorozat:

($”10110”,q_0$) → ($”0110”,q_0$) → ($”110”,q_0$) → ($”10”,q_1$) → ($”0”,q_2$) → (ε,q_0)

Vegyük észre, hogy a feldolgozás közben az automata felvette a q_2 állapotot, amely elfogadó állapot! De ez nem jelenti azt, hogy az automatának meg is kell állnia! Az automata akkor áll meg, amikor a δ képtelen folytatni a feldolgozást (pl mert nincs már input jel az input szalagon, vagy input jel még van, de az adott input jel + belső állapot párosra nincs tárolt utasítása). Az automata ezen menete ezt az input szót elutasította, mert bár a feldolgozás végig tudott menni az input szalagon, de a végén nem elfogadó állapotban állt meg!

Megj: Ez a konfiguráció-sorozat pontosan n elemű, ha az automata sikeresen végigolvasta az n hosszú input szót. Ekkor a sorozat biztosan megszakad (az automata megáll), hiszen nem lehet következő elemet meghatározni.

Megj: A sorozat kevesebb, mint n elemű, ha az utolsó konfigurációra nem tudjuk alkalmazni a δ függvényt. Ekkor a δ függvény parciális, hiszen egy teljes δ függvény minden esetben alkalmazható, amíg van input jel az input szalagon.

Megj: A sorozat ugyanarra az input szóra más-más elemeket tartalmazhat, ha a δ függvény nemdeterminisztikus. Ekkor ugyanis létezhet olyan sorozat-elem, amelyre a δ függvény több következő konfiguráció-elemet is képes lenne megadni, így többször is 'lefuttatva' a δ függvényt legalább ezen a ponton (és ettől a ponttól kezdve) eltérő sorozat-elemeket generálhat. A generálás közben több ilyen pont is előkerülhet, így az komoly munka lehet, hogy térképezzük fel az összes lehetséges koniguráció-sorozatot, amely az adott szó esetén előfordulhat. Ugyanakkor a nemdeterminisztikus automata ezen input szót elfogadja, ha a lehetséges sorozatok közül van olyan, amelynek az utolsó eleme elfogadó konfiguráció!

Def.: Egy A(K,V, δ, q_0, F) véges automata egy ω input szót **elfogad** (felismer), ha létezik olyan lépéssorozat, amelynek során a δ leképezés véges sokszori alkalmazása révén az automata induló konfigurációja (ω,q_0) átvihető az (ε,q') záró konfigurációba, és q'$\in$F. Ellenkező esetben az automata az input szót **elutasítja**.

Megj.: A fenti definíció egyformán jó a determinisztikus, nemdeterminisztikus, teljes és parciális esetekre is. A parciális esetben nem létezik olyan sorozat, amelynek során az input szó ε-ra csökkenhetne, hiszen ekkor az automata menet közben meg fog állni, és a szalagon még lenni kell szó-résznek. Nemdeterminisztikus működés esetén a fent írtak szerint akkor jó a szó, ha van olyan sorozat, amely esetén az automata elfogyasztja a szót, és elfogadó állapotban áll meg.

3.1.6 A véges automata működésének elemzése

Megj.: Egy n hosszú input szó esetén a véges automata legfeljebb **n lépés** végrehajtása után biztosan **megáll** (korábban is megállhat parciális működés esetén).

Megj.: Az automata minden esetben **biztosan megáll**. Ennek oka, hogy minden lépésben olvas be jelet a szalagról, és mindig lépteti a fejet, és mindig jobbra. Ezért legfeljebb n lépés múlva a fej biztosan lelép a szalagról.

Megj.: **Ha egy helyes szót táplálunk az automatába:**

1. az automata pontosan n lépés után megáll, és elfogadó állapotba kerül (ha az automata jól választja meg a lépéssorozatot)
2. az automata pontosan n lépés után megáll, de nem elfogadó állapotban van (ha az automata nemdeterminisztikus, és rossz útvonalat választott)
3. az automata kevesebb, mint n lépésen belül megáll (ha az automata nemdeterminisztikus, és parciális, és rossz útvonalat választott)

Megj.: **Ha egy helytelen szót táplálunk az automatába:**

4. Az automata pontosan n lépés után megáll, de nem elfogadó állapotban van (nemdeterminisztikus és determinisztikus esetben is)
5. Az automata kevesebb, mint n lépésen belül megáll (parciális működés)

Megj: Az automata azért áll meg, mert nem tudja folytatni a feldolgozást a szalagról lelépve. Ugyanis a delta fv igényli minden lépésben egy jel beolvasását az input szalagról, mely a szalagról lelépés után nem kivitelezhető.

3.1.7 Minimálautomata

Mivel egyetlen nyelvhez több felismerő automata is tartozhat, próbáljuk meg megkeresni ezek közül a legjobbat. Ezen automata azért lesz legjobb, mert **állapotainak száma a legkisebb** mind közül. Ezen automatát **minimálautomatának** nevezzük.

A minimálautomata megszerkesztéséhez indulásképpen szükségünk van egy determinisztikus és teljes automatára. Ennek ismeretében módszer ismert a minimálautomata előállítására (a már létező automatánk állapotainak számának minimalizálására).

Bizonyítható, hogy minden determinisztikus véges automata esetén ezen módszer lépéssorozata eredményre vezet, és a minimálautomata a jelölésrendszerétől eltekintve egyértelműen létezik.

Véges automata leírás = Feldolgozó algoritmus

A **véges automata** leírása gyakorlatilag nem más, mint egy speciális (főként matematikai) jelölésrendszerrel leírt **algoritmus**. Megpróbáljuk leíró nyelvhez hasonló eszközökkel is leírni a fentiek tanulságát (feltételezvén, hogy a tömbök és a szalag celláinak indexelése 0-tól indul):

```
AktAllapot        := q0
FejIndex          := 0
NemDetMukodesVolt := HAMIS
CIKLUS AMÍG FejIndex<SzalagHossz
        Jel := Szalag[ FejIndex ]
        FvOutput := DeltaTablazat( Jel, AktAllapot )      // lista
        N := Darabszama(FvOutput )                        // hány elemű a lista
        HA N=0 AKKOR
                HA NemDetMukodesVolt AKKOR
                        Kiiras:"Nem fogadta el, de nem-determinisztikus feldolgozás volt!"
                        Kiiras:"Így újabb futtatás lehet más eredményt ad!"
                KULONBEN
                        Kiiras: "Az input szó helytelen (parciális működés)!"
                HVEGE
                Azonnali_Leallas          // automata program azonnal leáll
        HVEGE
        Valasztas := 0
        HA N>1 AKKOR
                // ha az UjAllapotok listája több elemű, akkor nem-determinisztikus
                // működés volt, választani kell a lehetőségek közül
                Valasztas := veletlenszam( 0 .. N-1 )
                NemDetMukodesVolt := IGAZ
        HVEGE
        AktAllapot := UjAllapotok[ Valasztas ].UjBelsoAllapot
        FejIndex++
CVÉGE
HA AktAllapot eleme ElfogadoAllapotok AKKOR
        Kiiras: "Az automata az input szót elfogadta"
KULONBEN
        HA NemDetMukodesVolt AKKOR
                Kiiras:"Nem fogadta el, de Nemdeterminisztikus feldolgozas volt!"
                Kiiras:"Újabb futtatás lehet más eredményt ad!"
        KULONBEN
                Kiiras: "Az input szó biztosan helytelen!"
        HVEGE
HVEGE
```

A fentiekből látszik, hogy az AktAllapot változót olyanra kell deklarálni, amelyben tárolhatóak lesznek az aktuális állapotok. Ez megoldható, ha az állapotokat besorszámozzuk, ekkor a változó lehet például egész típusú. Hogy az egészek milyen altípusa szükséges, az eldönthető a K halmaz számosságából, tekintve hogy ekkor sorszámok várhatóak az algoritmus futása közben.

A DeltaTablazat() függvény kétparaméteres, ahol az első paraméter típusát a szalagon felfedezhető jelek típusa (V) határozza meg. Ha ez leírható ASCII karakterekkel, akkor a típus char. Ha azonban a V halmaz számossága ennél nagyobb, akkor szintén megoldás lehet a V halmaz elemeinek sorszámozása, és valamilyen egész típus használata.

A NemDetMukodesVolt logikai változóra azért van szükség, mert ha a szó feldolgozása közben egyszer is választás elé lett állítva az automata, akkor ennek sajátossága miatt a válasz csak egyszeri működésre értelmezett, a végső válasz előállításához igazából a visszalépéses keresést kellene használni, és a feldolgozás végén visszatérni minden választáshoz, és kipróbálni a további választási útvonalakat is. E módosított algoritmus ezen változtatás miatt jóval bonyolultabb felírású lenne, de az automata válasza végleges lenne, vagy ELFOGADTAM vagy NEM_ELFOGADTAM típusú lenne.

3.1.7 Baar-Hiller lemma

Tétel: Amennyiben van egy L nyelv, és egy A(K,V, δ, q_0, F) véges automata, amely az L nyelvet felismeri $\Rightarrow$ létezik egy n pozitív egész szám, hogy ha létezik olyan $\alpha \in L$ szó, hogy $L(\alpha) \geq n$, akkor igazak az alábbiak: az α szónak létezik olyan $\alpha = \beta\omega\gamma$ felbontása ($L(\beta\gamma) \leq n$, $L(\omega) \geq 1$), hogy $\forall i \geq 0$ esetén az $\beta\omega^i\gamma \in L$.

Biz: Mivel létezik ilyen véges automata, legyen az ő belső állapotainak száma n. Ebben az esetben az automatának n belső állapota van, de ha betápláljuk azt az α szót (amely a nyelvnek eleme, így az automata elfogadja), akkor a felismerés közben az automata egy belső állapotot legalább kétszer kell felvennie.

Tegyük fel, hogy ezen állapot, amelyet az automata legalább kétszer felvesz, q'. Ekkor a két q' állapot között legalább egy karaktert beolvas.

i. Jelöljük az α szó azon részét, amelyet az automata addig olvas be, amíg bbbelőször jut el a q' állapotába β-val.
ii. Azt a szakaszt, amíg q'-ből újra q'-be jut ω-al.
iii. Jelöljük az α maradék részét γ-val.

Ekkor gondoljuk végig. Ha a szó $\beta\omega\omega\gamma$ alakú lenne, mi történne. Az automata a szó felismerése közben β szakasz beolvasása után q' állapotba ér. Folytatva a beolvasást az első ω szakasz után újra q' állapotban lesz. Folytatva a beolvasást a második ω szakasz után újra q' állapotban lesz. Ebből a q'-ből a maradék γ beolvasása után elfogadó állapotba fog kerülni, mert tudjuk róla, hogy $\beta\omega\gamma$ esetén is oda került volna. Látható, hogy a középső ω szakaszt akárhányszor egymás után fűzhetjük, az automata legfeljebb annyiszor kerül (minden ω szakasz végére) q' állapotba. Az ω szakaszok száma tehát a szó elfogadását nem befolyásolja. Így a $\beta\gamma$, $\beta\omega\gamma$, $\beta\omega\omega\gamma$, $\beta\omega\omega\omega\gamma$, … alakú szavak is elfogadott szavak lesznek, így elemei az L nyelvnek.

Példa: az automatának 4 állapota van: q_0, q_1, q_2, q_3
Az input szó (amelyről tudjuk, hogy helyes): almáról

A szó beolvasása közben legalább egy állapot kétszer (sőt többször) kell szerepeljen, mivel a szó 7 hosszú, nekünk pedig 4 állapotunk van.

q_0 (a) q_1 (l) q_2 (m) q_3 (á) ??? mondjuk q_2 (r) q_1 (ó) q_3 (l) q_1

β = alm ω = áró γ = l

Mit tudunk elmondani az „alm áró áró l" szóról? „alm" végére q_3 lesz, innen indulva „áró" után újra q_3, innen indulva „áró" után megint csak q_3, innen indulva „l" után q_1, ami biztosan elfogadó állapot! Ezért ha „almáról" jó szó, akkor az „*almáróáról*" is jó szó!

De: „**alml**" is **jó szó** kell legyen, mert alm végén q_3 állapot, innen l után is elfogadó kell legyen az állapot!

Köv.: Amennyiben tehát van egy L nyelvünk, és ismert, hogy az őt felismerő véges automata pl. 4 belső állapottal rendelkezik, és mi találunk legalább egy olyan szót, amely 4-nél hosszabb, de eleme a nyelvnek, akkor végtelen sok szót találhatunk, így a nyelv végtelen.

Köv.: Amennyiben tehát van egy L nyelvünk, és ismert, hogy az őt felismerő véges automatának pl. 6 belső állapota van, úgy a nyelv akkor nem üres, ha létezik 6-nál rövidebb szó, amelyet a nyelv elfogad. Ha ugyanis létezik 6-nál hosszabb szó, akkor létezik 6-nál rövidebb is ($\beta\gamma$).

Megj.: Így a fenti tétel alapján egy algoritmus konstruálható, mely el tudja dönteni, hogy van-e olyan szó, amelyet egy automata felismer (végig kell próbálgatni az összes lehetséges összerakási kombinációt az n-nél rövidebb szavakra). Ha találunk ilyen szót, akkor a nyelv nem üres. Ha egyetlen n-nél rövidebb szót sem ismer fel az automata, akkor egyetlen szót sem fog! A végigpróbálgatásnál természetesen figyelni kell, hogy nemdeterminisztikus működés esetén az egyszeri próba sikertelensége nem jelenti azt, hogy egy újabb próba viszont nem lesz sikeres!

3.1.8 Számítási kapacitás

Def.: Egy adott (azonos) típusú automaták halmazát **absztrakt géposztálynak** nevezzük.

Def.: Egy absztrakt géposztály **számítási kapacitása** azon formális nyelvek halmaza, amelyet a géposztály valamely automatája felismer.

Tétel: A nemdeterminisztikus véges automataosztály és a determinisztikus véges automata osztály számítási kapacitása megegyezik.

Megj.: Ez első pillanatban meglepőnek tűnhet, hiszen a nemdeterminisztikus működés a determinisztikus működés kiterjesztése, és úgy tűnik, hogy a nemdet. automaták több lehetőséggel rendelkeznek, így az tűnik logikusnak, hogy az ő számítási kapacitásuk nagyobb. Az, hogy mégis egyenlő, annak természetesen az igazi oka az, hogy minden nemdet. véges automatához konstruálható vele ekvivalens det. véges változat.

Tétel: A véges automataosztály számítása kapacitása megegyezik a reguláris nyelvek osztályával.

Biz. $\Leftarrow$ **:** Adott egy véges automata, amely egy nyelvet felismer.

Vegyük a δ függvény definícióját. q_0 helyébe az S-t írjuk, q_1, q_2, …, q_n helyében pedig pl. S_1, S_2, ..., S_n nemterminális jeleket (így pont annyi nemterminális jelünk lesz, ahány belső állapotunk van). Ha a δ valamely sora $(q_n, a) \rightarrow q_m$ alakú, akkor ahhoz $S_n \rightarrow aS_m$ alakú helyettesítési szabályt kell felvenni (jobb reguláris nyelvtan). Bizonyítható, hogy a fenti szabályok alkalmazásával generált szavakat az automata felismeri, de egyéb szavakat nem.

Biz. $\Rightarrow$ **:** Adott egy reguláris nyelv.

A δ függvényt a leképezési szabályok alapján kell megkonstruálni. Az előző jelölésrendszert használva az $S_n \rightarrow aS_m$ alakú szabály esetén a δ függvénybe fel kell venni egy $(q_n, a) \rightarrow q_m$ sort. Persze, ha $S_n \rightarrow aS_k$ alakú szabály is lenne, úgy $(q_n, a) \rightarrow q_k$ is kell. Ez rögtön nemdeterminisztikus működést jelent, de ez nem probléma. A véghelyettesítések ($S_n \rightarrow a$) esetén esetleg fel kell venni egy újabb belső állapotot, mely egyben elfogadó állapot is legyen!

Megj.: Vagyis minden reguláris nyelvhez konstruálható olyan véges automata, amely az adott nyelvet felismeri, ill. ha van egy véges automatánk, akkor az általa felismert nyelv biztosan reguláris.

3.2 Verem automaták

3.2.1 Verem automaták definíciója

A verem automaták több dologban különböznek a véges automatáktól. Első, legszembetűnőbb különbség, hogy a verem automatának van output szalagja, méghozzá egy speciális tulajdonságú szalag. A szalaghoz tartozó író-olvasó fej ugyanis nem mozoghat szabadon a szalag fölött, hanem mindig a szalag legjobboldali részére írhat (és írás után jobbra mozdul), olvasáskor mindig csak a legjobboldali karaktert képes kiolvasni (előtte balra mozdul). Ez lényegében megegyezik a verem működésével (ahol is mindig a verem tetejére írunk, és olvasni is csak onnan tudunk). A veremautomata output szalagja elvileg végtelen (gyakorlatilag végtelen tárolókapacitású verem).

Def.: Egy G(K, V, W, δ, q_0, z_0, F) formális hetest verem automatának nevezünk, ahol:
K: az automata belső állapotainak halmaza (véges!)
V: az input ABC (az input szalagon milyen jelek fordulhatnak elő)
W: az output ABC (a verembe írható-olvasható jelek halmaza)
δ: állapotátmeneti függvény, $\delta \subseteq K \times (V \cup \{\varepsilon\}) \times W \rightarrow K \times W^*$
$q_0 \in K$: speciális belső állapot, a kezdőállapot
$z_0 \in W$: speciális veremABC-beli jel, „verem üres” szimbólum
$F \subseteq K$: befejező állapotok halmaza

Megj: Az automata működése az alábbi lépésekre osztható

Induláskor:
6. az automata q_0 kezdő állapotban van
7. a veremben csak egy jel van, a z_0
8. az input szalagon az input szó jelei vannak felírva, balról-jobbra feltöltve, folytonosan.
9. az olvasó fej az input szalag legbaloldalibb cellája fölött áll

Működési ciklus:
10. az olvasó fej vagy olvas az input szalagról, vagy nem
11. a veremből mindig olvassa a verem tetején levő jelet
12. ezen „input” jelek, és az aktuális belső állapot ismeretében, a δ függvényben megfogalmazottak szerint újabb belső állapotba vált, és a verembe egy szót ír
13. amennyiben olvasott az input szalagról, az olvasó fejet lépteti egyel jobbra

Megállás:
14. az automata megáll, ha az olvasó fej lelép a szalagról (jobbra)

Megj.: Amennyiben az automata megáll, meg kell vizsgálni, hogy milyen az aktuális állapota. Amennyiben az F-beli (elfogadó) állapot, akkor az automata a szót elfogadja. *A megállás és elfogadás-t még később pontosítjuk.*

Megj.: Mivel az automata nem minden ciklusléptésben olvas egy jelet az input szalagról, e miatt nem mindig lépteti az olvasó fejet, ezért az automata nem biztos, hogy megáll n lépés végrehajtása után.

Megj.: A δ függvény egy háromváltozós függvény. Egy aktuális állapot (K), egy input jel (V) vagy nem olvasás esetén ε, és egy input jel a veremből (W) alapján megadja, hogy milyen új állapotba (K) lépjen, és mit írjon vissza a veremve (W_*). $\delta \subseteq K \times (V \cup \{\varepsilon\}) \times W \rightarrow K \times W_*$

Megj.: A δ ezen esetben is lehet parciális, illetve teljes, valamint lehet nemdeterminisztikus ill. determinisztikus is. A fenti esetek kezelése lényegében megegyezik a véges automaták eseteivel.

Def.: Egy G(K, V, W, δ, q_0, z_0, F) verem automata egy L nyelvet felismer, ha minden $\alpha \in$L-re az automata megáll, és a szót felismeri, és minden $\beta \notin$L-re az automata a szót elutasítja.

Def.: Egy G(K, V, W, δ, q_0, z_0, F) verem automata által felismert szavakból alkotott L nyelvet az automata által felismert nyelvnek nevezzük.

Def.: Egy G(K, V, W, δ, q_0, z_0, F) verem automata egy G'(K', V, W', δ', q_0', z_0', F') automatával ekvivalens, ha ugyanazt a nyelvet ismeri fel.

Tétel: Egy G(K, V, W, δ, q_0, z_0, F) **parciális** verem automatához mindig konstruálható olyan G'(K', V, W', δ', q_0', z_0', F') **teljes** véges automata úgy, hogy a két automata **ekvivalens**.

Megj: A nemdeterminisztikus esettel nem ilyen egyértelmű a helyzet. Sajnos nem készíthető olyan algoritmus, amely tetszőleges nemdeterminisztikus veremautomata alapján elkészíti annak determinisztikus ekvivalens mását.

Megj: Az alábbi δ függvényű veremautomata nemdeterminisztikus! A q_0 állapotában, amikor a veremben z_0 van, és a szalagon '1'-es a következő karakter, akkor van választási lehetősége: vagy nem olvas a szalagról semmit (ε-t olvas), vagy beolvashatja az '1'-es karaktert is! Mindkét esetre van (különböző) outputja!

δ	K	$V \cup \{\varepsilon\}$	W	K	W_*
1.	q_0	ε	z_0	q_0	z_0
2.	q_0	1	z_0	q_1	zxy

Ezért a veremautomaták esetén a nemdeterminisztikus eset vizsgálatánál ügyelni kell az ε olvasási lehetőségekre is, ez esetben ugyanolyan választási lehetőséget kínálkozhat, mint egyszerűbb, látványosabb esetben, amikor az első három oszlop tökéletesen megegyezik!

3.2.2 Automata konfigurációja

Def.: Egy G(K, V, W, δ, q_0, z_0, F) verem **automata konfigurációja** egy (α, q, β) formális hármas, ahol α az input szalagon még hátra levő szó, q az aktuális belső állapot, β a veremben levő szó.

Megj.: Hogy miért ez az automata konfigurációja? Ugyanazon elv alapján mint a véges automatáknál. Amennyiben a fenti három információt ismerjük, úgy a verem automata adott időpillanatban levő teljes képét ismerjük.

Megj.: Az automata induláskori konfigurációja (ω, q_0, z_0), ahol ω a teljes input szó.
Az automata feldolgozás közbeni köztes konfigurációja valamely (α, q, β)
Normál működés végi (záró) konfigurációja (ε, q', β').

Def.: Egy G(K, V, W, δ, q_0, z_0, F) véges automata egy ω input szót **elfogad** (felismer), ha létezik olyan lépéssorozat, amelynek során a δ leképezés véges sokszori alkalmazása révén az automata induló konfigurációja (ω,q_0,z_0) átvihető az (ε,q',β') záró konfigurációba, és q'$\in$F. Ellenkező esetben az automata az input szót **elutasítja**.

Megj.: A fenti definíció megint csak megfelelő a δ minden működési módjára.

3.2.3 A δ leképezés elemzése

δ	K	V$\cup${ε}	W	K	W*
1.	q_0	ε	z_0	q_0	z_0
2.	q_0	1	z	q_1	zxy
3.	q_0	1	z	q_0	ε
4.	q_1	1	z	q_1	Z

1.sor: Az automata nem olvas az input szalagról, és olvasás előtt-után q_0 állapotba kerül, a veremből z_0-t olvas, és ír. Ezt a sort az automata önmagában végtelen sokszor ismételheti (végtelen ciklus).
2. sor: Az automata egy jelet vesz ki a veremből (z), és három jelet tesz vissza (zxy). Ezen sor végrehajtása után a verem aktuális mérete 2-vel növekszik.
3. sor: Az automata egy jelet vesz ki a veremből (z), és nem tesz vissza semmit (ε). Ezen sor végrehajtása után a verem aktuális 1-es csökken.
4. sor: Az automata egy jelet vesz ki a veremből (z), és egy jelet tesz vissza (z). Ezen sor végrehajtása után a verem aktuális mérete nem változik.

3.2.3 A verem automata működésének elemzése

Megj.: A verem automata **nem mindig áll meg.** Előfordulhat, hogy az input szalagról végtelen lépésen keresztül sem olvas, csak veremműveletetek végez (olvas a veremből, és ír). Ezen lépések közben csak a verem változik, meg a belső állapot. A veremautomata akkor esik „végtelen ciklusba", ha a lépéssorozatban előfordul az, hogy kétszer is ugyanazon belső állapotban van, és a verem tetején is ugyanaz a jel áll.

Megj.: **Ha egy helyes szót táplálunk az automatába:**

1. az automata legfeljebb n lépés után megáll, és elfogadó állapotba kerül (ha az automata jól választja meg a lépéssorozatot)
2. az automata legfeljebb n lépés után megáll, de nem elfogadó állapotban van (ha az automata nemdeterminisztikus, és rossz útvonalat választott)
3. az automata megáll (akár kevesebb, mint n lépésen belül, de akár több mint n lépés után is) parciális okokból (ha rossz útvonalat választott)
4. az automata nem áll meg (nemdeterminisztikus, végtelen ciklusba esik)

Megj.: **Ha egy helytelen szót táplálunk az automatába:**

5. Az automata legfeljebb n lépés után megáll, de nem elfogadó állapotban van (nemdeterminisztikus és determinisztikus esetben is)
6. Az automata legfeljebb n lépés után megáll parciális okokból (nemdeterminisztikus és determinisztikus esetben is)
7. Az automata nem áll meg (ekkor még determinisztikus is lehet)

3.2.4 Számítási kapacitás

Megj.: Létezik olyan verem automata konstrukció, amelyben nem szerepel az F halmaz. Ezen automata úgy jelzi, hogy felismer egy szót, hogy a működésének végén üres lesz a verem (vagyis a verem tetején levő jel z_0). Az ilyen automaták $G_{z0}(K, V, W, \delta, q_0, z_0)$ formális hatosok.

Tétel: Minden $G_{z0}(K, V, W, \delta, q_0, z_0)$ üres veremmel felismerő verem automatához konstruálható olyan $G(K, V, W, \delta, q_0, z_0, F)$ hagyományos verem automata, amelyek ekvivalensek, és viszont.

Biz.: Azon múlik, hogy a hagyományos veremautomata felismeréses befejezés megállása előtt ürítse ki a vermet egy veremkiürítő δ lépéssorozattal, melynek során az input szalagról ε-t olvas, a veremből egy jelet olvas, és ε-t tesz vissza.

Amennyiben pedig van egy üres veremmel felismerő verem automatánk, abból úgy tudunk hagyományosat csinálni, hogy a felismerés végén még megvizsgálja, hogy a verem tetején z_0 van-e. Ha igen, egy felismerő állapotba billen, ha nem, akkor egy nem felismerőbe, és csak ekkor áll meg.

Megj.: Az üres veremmel felismerő verem automaták automataosztályának számítása kapacitása megegyezik a hagyományos veremautomaták automataosztály számítási kapacitásával.

Tétel: A verem automaták automataosztályának számítása kapacitása megegyezik a környezetfüggetlen nyelvek osztályával.

Megj.: Vagyis minden 2-es típusú nyelvhez konstruálható olyan verem automata, amely az adott nyelvet felismeri, ill. ha van egy verem automatánk, akkor az általa felismert nyelv legalább 2-es típusú.

Megj.: A veremautomaták felülről kompatíbilisek a véges automatákkal, ugyanis ha a veremautomata δ sorai mindig z_0-t olvas és ír a verembe, és minden lépésnél olvas a szalagról, akkor visszakapjuk a véges automatákat. Viszont ha egy veremautomata felismer egy nyelvet, arról lehet tudni, hogy legalább 2-es típusú, de lehet akár 3-as is!

3.2.5 Példa veremautomata

A nyelv: $S \rightarrow 1S1$, $S \rightarrow 0S0$, $S \rightarrow \varepsilon$, az automata elfogadó állapota a (B).
Az automata egyúttal üres veremmel is felismerő automata.

K	$V \cup \{\varepsilon\}$	W	K	W^*
q_0	1	z_0	q_0	1
q_0	0	z_0	q_0	0
q_0	0	1	q_0	10
q_0	0	0	q_0	00
q_0	1	1	q_0	11
q_0	1	0	q_0	01
q_0	1	1	A	ε
q_0	0	0	A	ε
A	1	1	A	ε
A	0	0	A	ε
A	ε	z_0	B	z_0

Veremautomata leírás = Feldolgozó algoritmus

```
AktAllapot        := q0
VeremBerak( Z0 )
FejIndex          := 0
NemDetMukodesVolt := HAMIS
CIKLUS VÉGTELEN
        VeremTeteje := VeremKiolvas()          // innen mindig olvasunk jelet
        Jel1 := ''                             // szalagról vagy epszilont (üres jel)
        Jel2 := Szalag[ FejIndex ]             // vagy 1 db karaktert
        FvOutput1 := DeltaTablazat( Jel1,VeremTeteje, AktAllapot )      // lista-1
        FvOutput2 := DeltaTablazat( Jel1,VeremTeteje, AktAllapot )      // lista-2
        N1 := Darabszama( FvOutput1 ) // epszilon esetén hány elemű a lista
        N2 := Darabszama( FvOutput2 ) // olvasás esetén hány elemű a lista
        // ---------------------------------------------------------------------------------------------
        // tudunk-e működni, ha epszilont olvastunk vagy ha karaktert
        // ---------------------------------------------------------------------------------------------
        ELÁGAZÁS                               // esetválasztás
                HA N1=0  ÉS N2=0 AKKOR         // nem lehet továbblépni
                        Ciklus_Befejez
                HVEGE
                HA N1>0 ÉS N2>0 AKKOR          // mindkét eset működne
                        NemDetMukodesVolt := IGAZ
                        Valasztas := VeletlenSzam( 0..1 )
                        HA Valasztas=0 AKKOR           // első esetet választottunk
                                JEL := Jel1            // amikor epszilon-t olvastunk
                                FvOutput := FvOutput1
                        KULONBEN                       // második esetet választjuk
                                JEL := Jel2            // amikor olvastunk elemet a szalagról
                                FvOutput := FvOutput2
                        HVÉGE
                HVÉGE
                HA N1=0 ÉS N2>0 AKKOR          // csak a 2. eset lehetséges
                                JEL := Jel2
                                FvOutput := FvOutput2
                HVÉGE
                HA N1>0 ÉS N2=0 AKKOR          // csak az első eset lehetséges
                                JEL := Jel1
                                FvOutput := FvOutput1
                HVÉGE
        EVÉGE
        // ---------------------------------------------------------------------------------------------
        // van működési esetünk - működtessük
        // ---------------------------------------------------------------------------------------------
        N := Darabszama( FvOutput )            // hány elemű a lista
        Valasztas := 0                         // 0. elem választása alapértelmezettként
        HA N>1 AKKOR
                // ha az UjAllapotok listája több elemű, akkor nem-determinisztikus
                // működés volt, választani kell a lehetőségek közül
                Valasztas := veletlenszam( 0 .. N-1 )
                NemDetMukodesVolt := IGAZ
        HVEGE
        VerembeKiirSzo(FvOutput[ Valasztas ].VeremSzo )
        AktAllapot := FvOutput[ Valasztas ].UjBelsoAllapot
        HA JEL > '' AKKOR      // ha olvastunk be jelet, akkor lépünk is a szalagon
                FejIndex++
        HVÉGE
CVÉGE
```

```
HA FejIndex == SzoHossza AKKOR                  // végigolvastuk a szót
        HA AktAllapot eleme ElfogadoAllapotok AKKOR
                Kiiras: ”Az automata az input szót elfogadta”
        KULONBEN
                HA NemDetMukodesVolt AKKOR
                        Kiiras:”Nem fogadta el, de Nemdeterminisztikus feldolgozas volt!”
                        Kiiras:”Újabb futtatás lehet más eredményt ad!”
                KULONBEN
                        Kiiras: ”Az input szó biztosan helytelen!”
                HVEGE
        HVEGE
KULONBEN
                HA NemDetMukodesVolt AKKOR
                        Kiiras:”Nem fogadta el, de Nemdeterminisztikus feldolgozas volt!”
                        Kiiras:”Újabb futtatás lehet más eredményt ad!”
                KULONBEN
                        Kiiras: ”Az input szó biztosan helytelen (parciállis leállás)!”
                HVEGE
HVÉGE
```

A működés során ki kell próbálni, hogy vannak-e a delta függvényben output sorok arra az esetre, ha nem olvasnánk jelet az input szalagról, valamint azt is meg kell vizsgálnunk, hogy ha beolvassuk a következő jelet, akkor van-e a delta függvényben erre vonatkozó output. Amennyiben mindkettőre van lehetőség, mert vannak output sorok, akkor a delta függvény máris nem-determinisztikus.

A ciklus akkor áll le, ha adott helyzetben egyáltalán nincs delta output. Ez előfordulhat azért, mert elfogyott a szó, de előfordulhat ’menet közben’ is. Ez utóbbi esetben biztosan nem fogadjuk el jelen futás alapján a szót. Ha a szó azonban elfogyott, akkor még meg kell vizsgálni, hogy elfogadó állapotban vagyunk-e.

A választott output tartalmazza a verembe írandó jelsorozatot is, és az új választott állapotot is. A szalagon a fejet csak akkor kell léptetni, ha olvastunk a szalagról jelet.

A VeremKiolvas() függvénynek vissza kell adnia a veremből a legfelső jelet. Amennyiben a verem üres lenne, úgy a verem-üres szimbólumot (z_0) kell visszaadnia.

3.3 Környezetfüggetlen nyelvek és a szintaxis-fa

Nyelvtan: $S \rightarrow 1S0S1$, $S \rightarrow 1$, $S \rightarrow 1S1$, $S \rightarrow S1$
A keresett szó: „1111101101111111”.

A szó generálható-e az adott nyelvtanból? Konstruáljunk hozzá levezetési fát. A levezetési fát a végén balról jobbra, felülről lefele kell kiolvasni (csak a levél elemeket, vagyis csak a terminális jeleket kell kiolvasni).

Tétel: Egy $X \in V^*$ szó pontosan akkor levezethető S-ből, ha konstruálható hozzá levezetési fa (szintaxis fa).

Def.: Egy 2-es típusú **generatív grammatika nem egyértelmű**, ha $\exists\ \alpha \in L(G)$, amelyhez két különböző (nem izomorf) levezetési fa is konstruálható.

Def.: Egy 2-es típusú **nyelv nem egyértelmű**, ha csak nem egyértelmű generatív grammatika segítségével definiálható.

Megj.: A levezetési fán egy helyettesítéssel egyszerre több új nemterminális jel is bekerülhet. A továbbhaladás szempontjából választani kell, hogy melyiket helyettesítsük tovább.

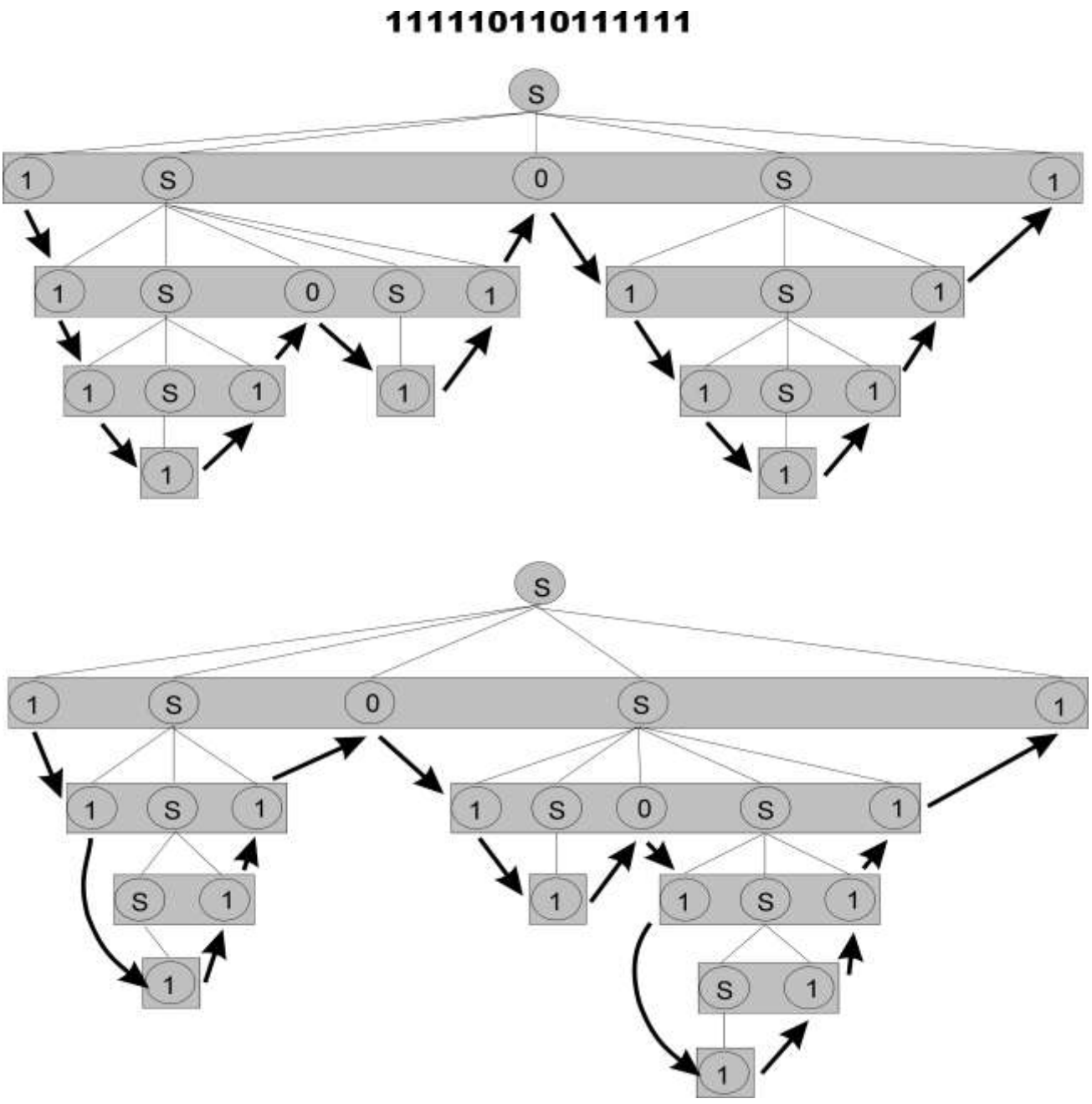
Def.: Amennyiben a levezetési fa konstruálása közben mindig a legbaloldalibb nemterminális szimbólumot helyettesítjük tovább, úgy a levezetést **kanonikusnak** nevezzük.

Tétel: Egy generatív grammatika nem egyértelmű, ha $\exists\ \alpha \in L(G)$, amelyhez legalább két kanonikus levezetés is tartozik.

Def.: Amennyiben a levezetési fát felülről lefelé (S-ből kiindulva) konstruáljuk, úgy a konstruálást **top-down stratégiának** nevezzük.

Def.: Amennyiben a levezetési fát alulról felfelé konstruáljuk, úgy a konstruálást **bottom-up stratégiának** nevezzük.

Megj.: A top-down hossznemcsökkentő stratégia, a kapott string folyamatosan bővül. Ha egy konkrét szóhoz konstruálunk levezetési fát, s amennyiben a levélelemek között szereplő terminális jelek száma már több, mint a szó hossza, úgy abbahagyhatjuk a konstruálást, mert az már biztosan nem fog sikeresen befejeződni. Vagy pedig nem töröljük az egész fát, hanem visszalépünk egy szintet, és próbálunk más helyettesítést használni. Ez is egy módszer annak ellenőrzésére, hogy mikor kell visszalépni, bár nem túl hatékony módszer.



Megj.: A másik módszert akkor használhatjuk, ha kanonikus levezetést alkalmazunk. Ekkor ugyanis a szó eleje (bal része) folyamatosan kezd kialakulni. Amennyiben a fa leolvasásakor kapott szó eleje (a terminális jelek szempontjából) már eltér a kívánt szótól, akkor vissza kell lépni, és más utat kell keresni. Ez az **egyszerű zsákutcás elemző** módszer.

Tétel: Ha L egy környezetfüggetlen nyelv $\Rightarrow \exists\, n \in N$ csak a nyelvtől függő konstans, hogy ha $\omega \in L$, $L(\omega) > n$ akkor az $\omega = uvwxy$ öt részre bontható, ahol $L(w) \geq 1$, $L(vx) \geq 1$, $L(vwx) \leq n$, és uv^iwx^iy alakú szavak is elemei a nyelvnek ($i=0,1,2,\ldots$).

Biz.: A bizonyítás a szintaxisfán nyugszik, n a nemterminális szimbólumok száma.

Tétel: Ha L_1, L_2 környezetfüggetlen nyelv $\Rightarrow$ $L_1 \cap L_2$ nem biztos, hogy környezetfüggetlen.

Biz: Triviális. Két halmaz metszete lehet véges is, és minden véges nyelv reguláris.

Tétel: Ha L_1, L_2 környezetfüggetlen nyelvek, $\Rightarrow$ $L_1 \cup L_2$ is környezetfüggetlen nyelv.

Tétel: Ha L környezetfüggetlen nyelv $\Rightarrow$ L^* is környezetfüggetlen nyelv.

Megj: Vegyük az S → aAbX , S→YbCc, A → aAb , A→ab, C → bCc , C→bc, X → cX , X→c, Y → aY , Y→ a szabályokkal megadott nyelvet.
Ez a nyelvtan az L = aibicj ∪ ajbici alakú szavakat generálja, és bizonyítható, hogy ezen nyelv nem definiálható egyértelmű nyelvtannal.

Megj: algoritmikusan eldönthetetlen, hogy egy nyelv vagy egy nyelvtan egyértelmű-e vagy sem (nincs erre általánosan használható algoritmus).

Tétel: minden reguláris nyelv és nyelvtan egyértelmű.

3.4 Szintaktikus elemzők

Szintaktikus elemzők feladata valamely jelsorozat levezetési fájának előállítása.

Early-algoritmus: lényegében megegyezik az egyszerű zsákutcás elemző algoritmusával! Kiindulunk a startszimbólumból, majd minden lehetséges továbbhelyettesítést megvizsgálunk. Amennyiben valamely továbbhelyettesítés során egy terminális szimbólum helyeződik a levezetett mondatforma elejére, úgy ellenőrízzük, hogy az megfelelő-e. Amennyiben nem, ezt a helyettesítést eldobjuk. Minden megtartott első lépésű továbbhelyettesítést külön-külön megvizsgálunk, és megpróbáljuk belőle kiindulva minden lehetséges továbbhelyettesít kipróbálni. Hasonlóan szűrjük ki a hibás útvonalakat. Amennyiben célhoz érünk, úgy a szó előállítható, és ismert a helyettesítési sorozat, amellyel célhoz érünk.

Coke-Younger-Kasami (CYK) algoritmus: ez az algoritmus csak akkor használható, ha a nyelvtanunk Chomsky-normál alakban van felírva. Mivel minden környezetfüggetlen nyelv felírható ilyen alakban, ezért ez az algoritmus használható ezen nyelvosztály problémájának kezelésében, bár az algoritmus azt fogja megadni, hogy a normál alakú szabályrendszer mely szabályainak sorozatával vezethető le a szó, és nem azt, hogy az eredeti nyelv szabályainak mely sorozatával vezethető le.

Az elemzés alulról-felfelé halad. Egy alsó háromszögmátrixot fogunk előállítani, mely celláiba a normál forma nemterminális szimbólumai kerülnek.

Példa: W={S,A,M,K,P,R,X,Z} V={a, (, +,) }

S→SA	*S→SM*	*S→KP*		
A→RS	*M→XS*	*P→SZ*		
R→+	*X→**	*K→(*	*Z→)*	*S→a*

Megj: a nyelvtan primitív numerikus kifejezéseket generál, pl *a+a*(a+a)*

Legyen az előállítandó szó az ”**a+a*a+a**”

A mátrixot a legalsó sorának kitöltésével kezdjük. A k. cellába kerül be az a nemterminális jel, amelyből a generálandó szó k. terminális jele előállítható (1 hosszú rész-szavak).

	1.	2.	3.	4.	5.	6.	7.
1.							
2.							
3.							
4.							
5.							
6.							
7.	S	R	S	X	S	R	S
	a	+	a	*	a	+	a

A következő lépésben a mátrix 6. sora kerül kitöltésre: a cellákba bekerül az a nemterminális jel, melyből kiindulva generálható a cél-szó adott pozíciójától kiinduló 2 hosszú rész-szava. Pl. a mátrix [6,2] cellájába azért került be az A szimbólum, mert képes a cél szó 2. karakterétől kiinduló 2 hosszú rész-szó (***+a***) generálására (*A→RS, R→+, S→a szabályok alapján).*

	1.	2.	3.	4.	5.	6.	7.
1.							
2.							
3.							
4.							
5.							
6.		A		M		A	
7.	S	R	S	X	S	R	S
	A	+	a	*	a	+	a

A következő lépésben az 5. sorba olyan nemterminálisok kerülnek be, amelyekből kiindulva 3 hosszú rész-szavak generálhatóak:

	1.	2.	3.	4.	5.	6.	7.
1.							
2.							
3.							
4.							
5.	S		S		S		
6.		A		M		A	
7.	S	R	S	X	S	R	S

Tovább folytatva az eljárást az alábbi mátrixot kapjuk:

	1.	2.	3.	4.	5.	6.	7.
1.	**S**						
2.		A					
3.	S		S				
4.		A		M			
5.	S		S		S		
6.		A		M		A	
7.	S	R	S	X	S	R	S

Az [1,1] cellába lévő S szimbólum azt mutatja, hogy belőle kiindulva előállítható a szó azon rész-szava, amely az első karaktertől kezdődik, és 7-1+1 hosszú. Ez gyakorlatilag megegyezik a teljes rész-szóval, így a kívánt szó előállítható az S szimbólumból kiindulva.

4. Turing gépek

4.1 Turing gépek definíciója

A Turing gépek olyan automaták, amelyeknek nincs külön output szalagjuk, de az „input” szalagról nem csak olvasni képesek, de írni is. Ezen túl az input szalag mindkét irányban végtelen. Az input szalagon induláskor az ellenőrzendő szó szerepel balról jobbra folytonosan felírva. A többi cellában egy speciális jel, a BLANK jel áll.

A Turing gépek esetén a megállással probléma van, ugyanis az eddig megismert automaták azért álltak meg, mert elfogyott az input szalagjuk (a fej lelép róla). Illetve még azért is megállhattak, mert δ parciális. A Turing gépeknél az első ok nem következhet be, hiszen a szalag mindkét irányban végtelen. Ezért csak a második indok jöhet szóba a megállásra. Ezért a Turing gépek δ függvénye mindig parciális kell legyen.

Def.: A A(K, V, W, δ, q_0, B, F) formális hetest Turing automatának nevezzük, ahol

K: az automata belső állapotainak halmaza
V: az input ABC (a nyelv ABC-je)
W: output ABC (ilyen jeleket írhat vissza a szalagra). $V \subseteq W$.
δ: állapotátmeneti függvény, $\delta \subseteq KxW \rightarrow KxWx\{\leftarrow,\rightarrow\}$
q_0: kezdőállapot, $q_0 \in K$
B: blank jel, $B \in W$
F: elfogadó állapotok halmaza $F \subseteq K$

Megj: Az automata működése az alábbi lépésekre osztható

Induláskor:

8. az automata q_0 kezdő állapotban van
9. az input szalagon az input szó jelei vannak felírva, balról-jobbra feltöltve, folytonosan.
10. az író/olvasó fej az input szalag legbaloldalibb cellája fölött áll

Működési ciklus:

11. az olvasó fej az input szalagról beolvas egy jelet
12. ezen „input” jel, és az aktuális belső állapot ismeretében, a δ függvényben megfogalmazottak szerint újabb belső állapotba vált
13. egy jelet ír vissza a szalag aktuális cellájába
14. lépteti a fejet vagy balra, vagy jobbra

Megállás:

15. az automata megáll, ha δ függvény (parciális) nincs értelmezve az aktuális input jel és belső állapot esetén

Megj.: Bár az automata minden ciklusléptésben olvas egy jelet az input szalagról, de lelépni képtelen a szalagról, így az automata nem biztos, hogy megáll n lépés végrehajtása után. Sőt. Az automata egyáltalán nem biztos, hogy megáll.

4.2 Turing gépek konfigurációja, számítási folyamat, felismerés

Def.: Egy G(K, V, W, δ, q_0, B, F) Turing gép **konfigurációja** egy (α, q, β) formális hármas, ahol α az input szalagon az író/olvasó fejtől balra levő szó, q az aktuális belső állapot, β az író/olvasó fej alatti, és tőle jobbra levő szó. A fej a β szó első karakterén áll. *Az α, β szavak nem tartalmazzák a végtelen sok BLANK jelet. Úgy tekintjük, hogy az α előtt, és a β után már csupa BLANK jel áll a szalagon.*

Megj.: Az automata konfigurációja alapján lehet tudni minden lényeges információt a működés közbeni állapotról, és ennek ismeretében, a konfiguráció visszatöltése után az automata képes folytatni a folyamatot.

Megj.: A Turing gép kezdő konfigurációja (ε, q_0, ω) hármas, ω az input szó.
A működés közbeni konfigurációja valamely (α', q', β') hármas
A befejező konfigurációja valamely (α, q, β) hármas.

Megj.: A δ függvény egy (q,a) páron van értelmezve ($q \in K$, $a \in W$). δ(q,a) = (q', a', $\leftarrow$) alakú sorokból áll, ahol a (q,a) párhoz hozzárendel egy új állapotot (q'), egy jelet visszaír a szalagra (a'), és megadja, hogy a fejnek balra ($\leftarrow$), vagy jobbra kell lépnie.

Def.: A Turing gép valamely (α,q,β) konfigurációja (ahol β=aβ' alakú) **megállító konfiguráció**, ha a δ függvény a (q,a) párra nincs értelmezve.

Def.: A konfigurációknak egy olyan sorozatát, melynek első eleme a kezdő konfiguráció, minden további eleme az előzőből kapható a δ függvény alkalmazásával, **számítási folyamatnak** nevezzük.

Def.: Egy számítási folyamatot **teljes számítási folyamatnak** nevezzük, ha a sorozat utolsó konfigurációja megállító konfiguráció.

Def.: Egy Turing gép egy $\omega \in V^*$ szót **felismer** (elfogad), ha az (ε,q_0,ω) kezdő konfiguráció egy teljes számítási folyamat során átvihető valamely (α,q,β) megállító konfigurációba, és $q \in F$.

Megj.: A fenti felismerés definíció minden esetre jó (nemdeterminisztikus, determinisztikus). A parciális-nem parciális esetet nincs értelme külön venni, mert a Turing gép nem lehet teljes!

4.3 A Turing gépek működésének elemzése

Megj.: A Turing gép **nem mindig áll meg.** Előfordulhat, hogy az input szalag két cellája között alternáló mozgást végez a végtelenségig. De akár az is, hogy minden jel beolvasása után folyamatosan jobbra (vagy mindig balra) lép a végtelenségig. Ha a δ függvény nem parciális, úgy a Turing gép soha nem fog megállni (hibás gép).

Megj.: A Turing gép a BLANK cellákat felülírhatja valamely más jellel, így a szalag minden celláját felhasználhatja.

Megj.: A Turing gép által a végtelen szalagra írt szó nem feltétlenül folytonos. A felírt szó-szakaszok között BLANK cellák állhatnak.

Megj.: **Ha egy helyes szót táplálunk az automatába:**

16. az automata valahány (akár kevesenn mint n) lépés után megáll, és elfogadó állapotba kerül.
17. az automata valahány lépés után megáll, de nem elfogadó állapotban van (nemdeterminisztikus, és rossz útvonalat választott)
18. az automata nem áll meg (nemdeterminisztikus eset, rossz útvonal)

Megj.: **Ha egy helytelen szót táplálunk az automatába:**

19. Az automata valahány lépés után megáll, de nem elfogadó állapotban van (nemdeterminisztikus és determinisztikus esetben is)
20. Az automata nem áll meg.

4.4 Egy példa Turing gép

V := {0, 1}, K:={a, b, c, d, e}, W:={0, 1, B}, q_0=a

δ	K	V	K	W	Irány
	a	0	b	B	$\leftarrow$
	a	1	c	B	$\leftarrow$
	a	B	-	-	-
	b	0	b	0	$\leftarrow$
	b	1	b	1	$\leftarrow$
	b	B	d	B	$\rightarrow$
	c	0	c	0	$\leftarrow$
	c	1	c	1	$\leftarrow$
	c	B	e	B	$\rightarrow$
	d	0	d	0	$\rightarrow$
	d	1	e	0	$\rightarrow$
	d	B	a	0	$\rightarrow$
	e	0	d	1	$\rightarrow$
	e	1	e	1	$\rightarrow$
	e	B	a	1	$\rightarrow$

A fenti automata nem felismer egy szót, hanem gondos munkával megfordítja (tükrözi) azt. Pl. input szó: 01011

Másik mód, amivel a δ fv. megadható:

δ	a	b	c	d	E
0	B b $\leftarrow$	0 b $\leftarrow$	0 c $\leftarrow$	0 d $\rightarrow$	1 d $\rightarrow$
1	B c $\leftarrow$	1 b $\leftarrow$	1 c $\leftarrow$	0 e $\rightarrow$	1 e $\rightarrow$
B		B d $\rightarrow$	B e $\rightarrow$	0 a $\rightarrow$	1 a $\rightarrow$

4.5 Turing gépek által felismer nyelvek

Def.: Egy nyelvről azt mondjuk, hogy **rekurzívan felsorolható**, ha van olyan Turing gép, amely az adott nyelvet felismeri.

Def.: Egy nyelv **rekurzív**, ha létezik olyan Turing gép, amely az adott nyelvet felismeri, és mindig megáll.

Megj.: Ez azt jelenti, hogy egyéb szavakra is mindig megáll, nem csak a jó szavakra (amelyek elemei a nyelvnek).

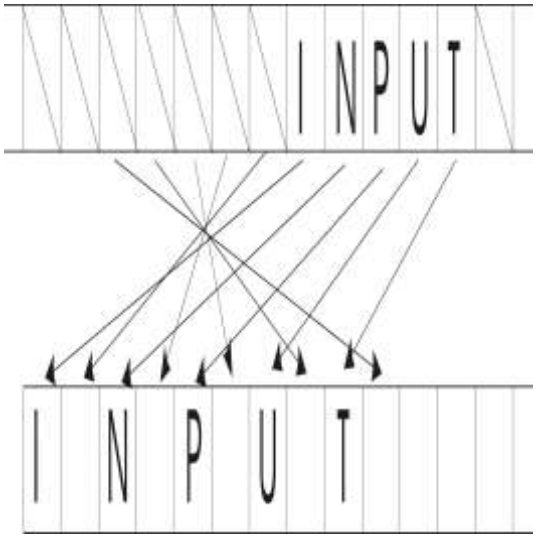
Tétel: A mondatszerkezetű nyelvek osztálya megegyezik a rekurzívan felsorolható nyelvek osztályával.

Megj.: Vagyis a Turing gépek a 0-s típusú nyelvek felismerő automatái, vagyis minden 0-s típusú nyelvhez konstruálható őt felismerő Turing gép, és viszont.

Megj.: Az 1-es típusú nyelvek felismerő automatái speciális Turing gépek.

4.6 Turing gépek változatai

1. **Egyirányú végtelen szalaggal rendelkező Turing gépek:**
   A szalag csak egyik irányban végtelen, a másik irányban le lehet róla lépni. Be lehet bizonyítani, hogy ezen automaták ekvivalensek a mindkét irányban végtelen szalagúakkal. Ugyanis ha az kétirányban végtelen szalagú Turing gép celláihoz hozzárendeljük az egyirányban végtelen szalag celláit (oly módon, hogy a páros cellákba kerülnek a jobb oldali cellák, a páratlan cellákba a bal oldali cellák), és átalakítjuk az eredeti, mindkét irányban végtelen szalaggal dolgozó Turing gép δ függvényét, hogy a megfelelő cellára lépés módját az új technikával végezze, akkor ugyanolyan felismerési képességet nyerünk, nem többet, és nem kevesebbet.



2. **Több szalagú Turing gépek:**
   Ezeknél több, esetleg minkét irányban végtelen szalagú output szalag van. Ekkor a δ függvény „input” adatai nem csak egy szalagról származnak, hanem mindegyikről, és mindegyikre ír minden lépésben, és minden fejre külön megmondja, hogy merre lépjen. Szintén belátható, hogy ezen automata osztály ekvivalens az egyszalagú Turing gépekkel.

Turing-gép leírás = Feldolgozó algoritmus

```
AktAllapot        := q0
FejIndex          := 0      // a szó 0. karakterén állunk, de balra is vannak BLANK cellák
NemDetMukodesVolt := HAMIS
CIKLUS VÉGTELEN
        Jel := Szalag[ FejIndex ]                       // mindig olvasunk karaktert
        FvOutput := DeltaTablazat( Jel, AktAllapot )    // lista vagy 1 db elem
        N := Darabszama( FvOutput )                     // hány elemű a lista
        // ------------------------------------------------------------------------------------------
        ELÁGAZÁS                        // esetválasztás, tudunk-e működni?
                HA N=0  AKKOR           // nem lehet továbblépni
                        Ciklus_Befejez
                HVEGE
                HA N=1 AKKOR            // a lista csak 1 elemeű
                        Valasztas := 0  // 0. listaelem választása
                HVEGE
                HA N>1 AKKOR
                        NemDetMukodesVolt := IGAZ
                        Valasztas := VeletlenSzam( 0..N-1 )
                HVÉGE
        EVÉGE
        // ------------------------------------------------------------------------------------------
        Szalag[ FejIndex ] := FvOutput[ Valasztas ].OutputJel
        AktAllapot         := FvOutput[ Valasztas ].UjBelsoAllapot
        HA FvOutput[ Valasztas ].FejIrany == JOBBRA AKKOR
                FejIndex++
        KÜLÖNBEN
                FejIndex--
        HVÉGE
CVÉGE

HA AktAllapot eleme ElfogadoAllapotok AKKOR
        Kiiras: ”Az automata az input szót elfogadta”
KULONBEN
        HA NemDetMukodesVolt AKKOR
                Kiiras:”Nem fogadta el, de Nemdeterminisztikus feldolgozas volt!”
                Kiiras:”Újabb futtatás lehet más eredményt ad!”
        KULONBEN
                Kiiras: ”Az input szó biztosan helytelen!”
        HVEGE
HVÉGE
```

Előfordulhat, hogy a Turing gép végtelen ciklusba esett. Ezt a lehetőséget nagyon nehéz detektálni, bár vannak rá módszerek (bár azok nagyon specifikusak). Általános esetben megjósolni, hogy egy Turing gép egy konkrét szót már nem fog felismerni – lehetetlen.

5. Lineárisan korlátolt Turing gépek

A lineárisan korlátolt Turing gépek jellemzői, hogy az input/output szalag nem végtelen, hanem mindkét irányban véges.

Megj.: Az input szalag hossza attól függ, hogy milyen hosszú az input szó. Léteznek olyan Turing gépek, amelyek a szó felismerése közben pont olyan hosszú szalaggal dolgoznak, mint maga az input szó hossza. Mások mindig kétszer, háromszor, négyszer, stb.. hosszabb szalagokat használnak. Minden ilyen típusú Turing gépe egy-egy osztályba sorolhatunk, ahol az osztály jellemzője, hogy hányszoros hosszú szalaggal dolgoznak.

Tétel: Minden fent említett automataosztály ekvivalens az egyszeres szalaggal dolgozó automataosztállyal.

Megj.: Mivel a véges szalagú Turing gépeknél a gép akkor is megállhat, ha lelép a szalagról, így a megállás, és elfogadás definíciója más, mint a végtelen szalagú Turing gépeknél.

Megj.: Egészítsük ki az ω input szót balról egy ➔ jellel, jobbról pedig egy ← jellel. Az egyik jelzi a szó bal végét, a másik a jobb végét.

Def.: Ha egy lineárisan korlátolt Turing gép a ➔ jelből indulva, a δ függvény véges sokszori alkalmával a ← jelbe kerül, és az automata elfogadó állapotban van, akkor az automata a szót **elfogadja** (felismeri).

Tétel: A lineárisan korlátolt Turing gépek által felismert nyelvek környezetfüggőek, és viszont.

6. Folytatás…

Def: Legyen $M = \langle K, V, W, \delta, s_0, B, F \rangle$ tetszőleges Turing-gép. Tekintsük az alábbi egyenlőséggel meghatározott függvényt:

$$f_M(\omega) = \begin{cases} \psi, \text{ ha létezik olyan } \langle \varepsilon, s_0, a\omega' \rangle \xrightarrow{M} \langle \psi', q, b\psi'' \rangle \text{ teljes számítási folyamat, hogy } a\omega' = \omega, \text{ valamint } \psi' b \psi'' = \psi; \\ \text{nincs meghatározva, ha ilyen teljes folyamat nem létezik.} \end{cases}$$

Az ilyen f_M függvényeket *Turing-értelemben előállítható (Turing-előállítható) függvényeknek* nevezzük, s azt mondjuk, hogy az M Turing-gép az f_M függvényt állítja elő.

A Turing-előállítható függvények általában csak részlegesen vannak értelmezve, mivel egyes bemeneti szavaknál a számítási folyamat végtelen hosszú is lehet. Ezeknek a függvényeknek a segítségével a Turing-gépekre is úgy tekinthetünk, mint formális nyelvek átalakítóira, melyek a szalagra írt bemeneti szót valamilyen kimeneti szóvá alakítják át a szalagon.

A formális nyelvek elméletének egyik jellegzetes problémája annak eldöntése, hogy egy adott ω szó beletartozik-e valamely adott nyelvbe. Láttuk, hogy ez a kérdés a környezetfüggő nyelvek körében mindig eldönthető, azaz megadható olyan általános módszer, melynek segítségével tetszőleges ω szóról és tetszőleges 1-típusú G grammatikáról eldönthető, hogy $\omega \in L(G)$ tartalmazás fennáll-e vagy sem. Egy ilyen módszert ***eljárásnak*** nevezünk, ha a változók megadása után teljesen gépiesen lehet végrehajtani, azaz kizárólag olyan műveleteknek az elvégzését igényli, amelyek matematikailag egyértelműen vannak definiálva, és e műveletek végrehajtási sorrendje is egyértelműen adva van. Egy eljárást ***algoritmusnak*** nevezünk, ha a változók tetszőleges választása mellett mindig véges sok lépésben befejeződik.

Eldöntési eljárásnak nevezünk egy eljárást, ha olyan kérdésre keresi a választ, ahol csak igen vagy nem választ várunk. ***Eldöntési algoritmusnak*** nevezünk ezek szerint minden olyan eljárást, amely a feltett kérdésre véges sok lépésben igen vagy nem választ ad. Egy problémát ***algoritmikusan eldönthetőnek*** nevezünk, ha megadható hozzá egy eldöntési algoritmus. Az $\omega \in L(G)$ probléma környezetfüggő grammatikák esetén algoritmikusan eldönthető, ebből következik, hogy egy szűkebb grammatikaosztály esetében – mint pl. a környezetfüggetlen vagy a reguláris grammatikák esetében – az $\omega \in L(G)$ probléma ugyancsak eldönthető.

Függőben hagytuk viszont azt a kérdést, hogy a 0-típusú nyelvekre nézve általában eldönthető-e a tartalmazási probléma.

Def (alternatív): Egy L nyelvet *rekurzívnak* nevezünk, ha az $\omega \in L$ tartalmazási probléma algoritmikusan eldönthető.

Def (alternatív): Egy L nyelvet *rekurzíve felsorolhatónak* nevezünk, ha van egy olyan (véges vagy végtelen) eljárás, amely az összes $\omega \in L$ szót valamilyen sorrendben előállítja, azaz felsorolja.

Minden rekurzív nyelv nyilván rekurzíve felsorolható. Nem ui. mást tennünk, mint rendre előállítani az összes $\omega \in V^*$ szót, miközben minden egyes új szó előállítása után alkalmazzuk rá az eldöntési algoritmust, és belevesszük a felsorolásba, ha igen választ kapunk, ellenkező esetben elhagyjuk.

Tétel: Egy L nyelv akkor és csakis akkor rekurzív, ha mind az *L* mind az $\overline{L}$ (L komplemense) rekurzíve felsorolható.

Tétel: Egy mindössze egyetlen betűt tartalmazó V_1 = {a} ábécében létezik olyan rekurzíve felsorolható $L \subseteq V_1^*$ nyelv, amelynek a komplemense nem rekurzíve felsorolható.

Tétel: Minden 1-típusú nyelv rekurzív, és minden 0-típusú nyelv rekurzíve felsorolható.

Tétel: Van olyan rekurzív nyelv, amely nem 1-típusú.

Church-féle tézis: Minden eljárás egyenértékű valamilyen Turing-géppel, azaz minden pontosan definiált eljáráshoz megadható egy Turing-gép, amely ezt az eljárást végrehajtja.

Matematikai értelemben ennek csak a cáfolatát lehetne bizonyítani, ha megadnánk egy konkrét eljárást, amelyhez nem lehet a megfelelő Turing-gépet megkonstruálni. Ilyen ellenpéldát még nem találtak, és minden jel arra vall, hogy ez a tézis kellően megalapozott. A Church-féle tézist legalább munkahipotézisként mindaddig elfogadjuk, amíg ellenpéldát nem találunk.

Tétel: A rekurzíve felsorolható nyelvek osztálya megegyezik a 0-típusú nyelvek osztályával.

Alan Turing 1936-ban konstruálta meg annak a gépnek a modelljét, amellyel bármely Turing-gép működését szimulálni lehet. Jelöljük a továbbiakban ezt az ***univerzális Turing-gépet*** U-val. Az U univerzalitása alatt a következőket értjük. Legyen adva egy tetszőleges T Turing-gép. Amennyiben T átmenetfüggvényét, valamint egy ω bemeneti szavát megfelelő módon kódoljuk, akkor U erre a kódolt bemenetre pontosan akkor áll meg végállapotban, ha T is megáll az ω szóra mint bemenetre. Az is igaz, hogy ha az U megáll, akkor a szalagján a megállás pillanatában épp a T befejező konfigurációja van valamilyen formában kódolva.

Tétel: Nincs olyan Turing-gép, amely tetszőleges ω szóról és tetszőleges T Turing-gépről véges sok lépésben eldöntené, hogy $\omega \in L(T)$ fennáll-e.

Tétel: Van olyan rekurzívan felsorolható nyelv, amely nem rekurzív.

Tétel: Legyen adva a rekurzívan felsorolható nyelveknek valamilyen nem triviális tulajdonsága. Ekkor annak eldöntése, hogy egy nyelv rendelkezik-e az adott tulajdonsággal, algoritmikusan megoldhatatlan probléma.

Következmény: Annak felismerése, hogy egy rekurzívan felsorolható nyelv mikor

a) üres;
b) véges;
c) rekurzív;
d) reguláris;
e) környezetfüggetlen;
f) környezetfüggő stb.,

algoritmikusan megoldhatatlan problémát jelent.

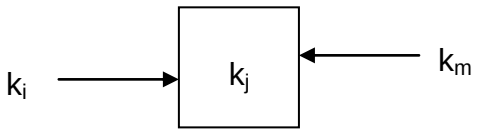
Neumann-automaták:

A Turing-gép működésének lényeges jellemzője, hogy az információ átalakítása minden ütemben csak egy négyzetben történik meg, a többi négyzet ezalatt várja, hogy a gép feje odaérjen. Természetesen vannak olyan esetek, amikor ezt a várakozást a megoldandó feladat természete indokolja. Ez a helyzet például akkor, amikor mindaddig nem világos, hogyan kell az adott négyzetben lévő jelet átalakítani, amíg egy másik, tőle távolabbi négyzeten nem kapunk eredményt. Könnyű azonban más szituációt is elképzelni, amikor nincs ilyen összefüggés. Ekkor annak egyetlen oka, hogy a gép nem egyszerre valósítja meg a két négyzetben lévő jel átalakítását, hogy nem képes rá.

Ezzel kapcsolatban vetődik fel az a gondolat, hogy lehetne olyan gépet is konstruálni, amelyet az információ-átalakítás párhuzamossága jellemez, azaz az információ átalakítása sok tárrekeszben egyszerre történik.
A legegyszerűbb ötlet, ami ezzel kapcsolatban kínálkozik, olyan Turing-gépek rendszerének a vizsgálata, amelyeknek közös a szalagjuk. Nem mindig lehet azonban kiszámítani két vagy több Turing-gép szerencsés együttműködését. Előfordulhat, hogy mindkét fej ugyanarra a jelre akar lépni. Ilyen esetekben olyan kiegészítő utasításokra van szükség, amelyeket az egyes Turing-gépeken nem lehetett előre látni. Célszerű ezért szabályozni több közös külső tárú Turing-gép együttműködését. Eszerint két (vagy több) fej azonos négyzetre lépését ne engedjük meg. Ezt pl. így érhetjük el: ha két fej annyira közel kerül egymáshoz, hogy az a veszély fenyeget, hogy azonos négyzetre lépnek, akkor olyan előírt utasításokat végez el a gép, amelyek kiküszöbölik ezt a veszélyt. Ilyen típusú utasításokkal lehet pontosabbá tenni a következő fogalmat: „ *n* számú Turing-gépből álló rendszer közös külső tárral” (*n* = 2, 3, …).Egy ilyen rendszert egyetlen új típusú gépnek tekinthetünk, komponenseinek rendszerint ugyanannak az M Turing-gépnek *n* példányát választjuk. A pontos definíciót nem részletezzük, csak azt jegyezzük meg, hogy egy *n* számú Turing-gépből álló rendszer semmilyen *n*-re sem tudja megoldani teljesen a kiszámítási folyamat párhuzamos szervezését, hiszen minden ütemben a négyzetek zöme eleve tétlenségre kényszerül.

A Neumann- automatában a párhuzamosság elve a lehetőségek határáig megvalósul. Ennek megfelelően a Neumann- automatában korlátlanul nőhet azoknak a négyzeteknek a száma, amelyekben egyszerre történik meg az információ átalakítása, bár ezek száma minden időpontban véges. Ebben az értelemben a Neumann- automata fölülmúl bármely olyan rendszert, amely rögzített számú Turing-gépből áll. A Neumann- automata nagyjából olyan rendszernek tekinthető, amely új és új Turing-gépet tud bekapcsolni a munkába, és képes arra, hogy együttes működésüket ugyanazon a külső táron megszervezze.

Definíció: A *Neumann-elem* olyan egység, amelyik a t = 1, 2, 3, … ütemek mindegyikében a $K = k_1, k_2, \ldots, k_n$ véges állapothalmaz valamelyik elemével jelölt állapotban van. Az elemnek két bemenő csatornája van: jobb és bal. Ezek mindegyikén a t-edik ütemben szintén egy állapotjel, azaz K egy eleme lép le. Az elem állapota a (t+1)-edik ütemben kizárólag attól függ, milyen állapotban volt az előző, t-edik ütemben és a t-edik ütemben milyen bemenő jeleket kapott a bemenő csatornákon. Másképpen ez a következőt jelenti: egy elem egy $\psi(p, r, q)$ háromváltozós függvényt realizál, a függvény argumentumai és a függvény értékei is K elemei lehetnek. A $\psi(k_i, k_j, k_m) = k_n$ egyenlőség azt mondja ki, hogy ha az elem k_j állapotban van, a bal oldali csatornán k_i, a jobb oldalin k_m jelet kap, akkor a következő ütemben k_n állapotba kerül. Egy ilyen egyenlőséget egy ún. *Neumann-utasítás* alakjában írunk fel, vagyis így: $k_i k_j k_m \rightarrow k_n$.

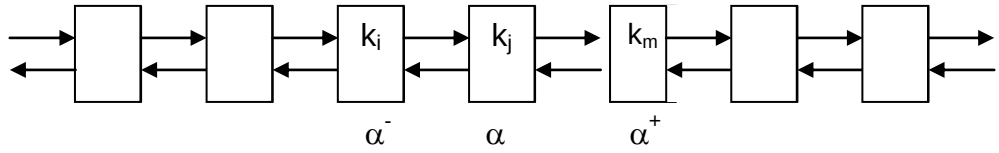


$$k_i k_j k_m \rightarrow k_n$$

Egy elem működését tehát a ψ függvénnyel, vagy ami ugyanaz, egy ν^3 Neumann-utasításból álló halmazzal adhatjuk meg, ez utóbbit *Neumann-programnak* nevezzük. A k állapotot nyugalmi állapotnak nevezzük, ha teljesül rá, hogy $\psi(k, k, k) = k$, ellenkező esetben az állapotot aktívnak nevezzük. Ez azt jelenti, hogy egy k

nyugalmi állapotban lévő elemet csak akkor lehet az állapotából kimozdítani, ha legalább az egyik bemenő csatornáján k-tól különböző jel lép be. A következőkben feltesszük, hogy a K állapothalmazban van egy speciális ϕ nyugalmi állapot. Erre tehát $\phi\phi\phi \rightarrow \phi$. Rögzítsünk egy tetszőleges N Neumann-elemet, ekkor (az adott elem feletti) *Neumann- automatát* az adott elem példányaiból álló, mindkét oldalon végtelen szalagként szerkeszthetjük meg.

Ha egy t időpontban valahogy rögzítjük az elemek állapotát, akkor ezzel egy R(t) *Neumann-konfigurációt* adunk meg. Jelöljük a szalag α elemének állapotát a t időpontban, azaz az R(t) konfigurációban α (t)-vel és a vele balról ill. jobbról szomszédos elemek állapotát α^-(t)-vel, ill. α^+(t)-vel.



Jegyezzük meg, hogy éppen az $\alpha^-(t)$, illetve $\alpha^+(t)$ állapotok lépnek be az α elembe ennek bal ill. jobb oldali csatornáján. Ezeknek megfelelően az elem a Neumann-automata programja alapján a következő időpontra $\alpha(t+1) = \psi(\alpha^-(t), \alpha(t), \alpha^+(t))$ állapotba kerül. Ez a szalag összes elemére érvényes, így egy ütem alatt az összes elemben egyidejűleg lezajló működés eredményeképpen alakul ki a következő R(t+1) konfiguráció. Pontosan így alakul ki az R(t+2), R(t+3) stb. konfiguráció is. Ebben áll a Neumann-automata működése: lényeges jellemzője éppen az, hogy a Neumann-konfiguráció formájában kódolt információt mindenütt egyidejűleg alakítja át.

A következőkben csak véges Neumann-konfigurációkat vizsgáljunk, azaz olyan konfigurációkat, amelyekben véges sok (de nem rögzített számú) elem kivételével az összes többi a ϕ nyugalmi állapotban van. A szalagnak ezt a legkisebb darabját, amelyen kívül az összes elem nyugalmi állapotban van, az automata *aktív zónájának* fogjuk nevezni. Ha egy Neumann-automatát véges konfigurációból indítunk, akkor később is csak véges konfigurációkba juthat, mivel az aktív zóna minden ütemben legfeljebb két elemmel (a két szélén) bővülhet. Ezért, bár a Neumann-automatát végtelen objektumként definiáltuk, ennek ellenére lényegileg minden időpontban csak véges objektum, bár korlátlanul nőhet. Ebben a Turing-gép szalagjához hasonlít, amelyik formálisan szintén végtelen, de lényegében végesnek tekinthetjük, megengedve, hogy növelhető akkor, amikor a rendelkezésre álló tár nem elég az eljárás folytatására. Ejtsünk néhány szó a Neumann- és Turing- eljárásokról!

Milyen feladatosztályok oldhatók meg Neumann-automatákkal, és a megfelelő megoldási folyamatoknak mik a specifikus jellemzői az általunk már ismert Turing-gépeken végzett megoldásokkal összehasonlítva? Ugyanazon okok miatt a Neumann-automatákat is függvények kiszámítására szolgáló eszközként kezeljük.

Definíció: Az N Neumann-automatáról akkor mondjuk, hogy kiszámítja az *f* függvényt, ha teljesül rá a következő: az *f* értelmezési tartományába tartozó tetszőleges ω szót ábrázoló (vagy ahogy még mondják, az ω szót kódoló) konfigurációt N az $R = f(\omega)$ szót kódoló konfigurációba viszi.

Azt kell pontosan megfogalmazni, hogy a szavaknak milyen kódolását választjuk. Az elfogadásra kerülő kódolási módszernek nemcsak a szóra vonatkozó információt kell tartalmaznia, hanem azt is mutatnia kell, hogy a szó kiinduló adatként vagy pedig eredményként szerepel. Sőt az is természetes követelmény, hogy az eredményszót ábrázoló konfiguráció stabil legyen, ami a következőnek felel meg: az eredmény megtalálása után a Neumann-automata álljon meg. Ezeket a követelményeket természetesen több kódolás is kielégíti.

Tétel: Minden Turing-kiszámítható f függvény Neumann-kiszámítható is, sőt f bármelyik Turing-kiszámításához meg lehet adni egy ugyanilyen gyors Neumann-kiszámítást is.

Tudjuk azonban, hogy egy Neumann-automata működhet olyan módon is, ami lényegesen különbözik egy Turing-gép, vagy akár egy közös külső tárú Turing-gép-rendszer egyszerű utánzásától. Ezzel kapcsolatban egy sor kérdés merül fel arra vonatkozóan, milyen további kiszámítási lehetőségek adódnak, ha a Neumann-automatát részesítjük előnyben a Turing-géppel szemben. Az első és alapvető kérdés, van-e olyan Neumann-kiszámítható függvény, amelyik nem Turing-kiszámítható? A tagadó választ, amit Church tézise is sugall, valóban igazolni lehet.

Tétel: Minden Neumann-kiszámítható függvény Turing-kiszámítható.

Bár a Neumann-automata és a Turing-gép ugyanazt képes elvégezni, de különböző módon. Ezért az algoritmusok hatékonysága szempontjából előfordulhat, hogy bizonyos típusú feladatok megoldását előnyösebb Neumann-automatával végezni, mint Turing-gépen. Bizonyítható, hogy Neumann-automaták alkalmazása árán legfeljebb olyan időösszevonást lehet elérni a számítási eljárásban, mint nagyságrendben annak az időnek a négyzetgyökét, amivel egy „gazdaságosan” működő Turing-gép dolgozik.

A Turing-gép a számítógép matematikai modellje. A Turing-automata δ függvénye a számítógép egyfajta programozási nyelvével egyenértékű, ahol a cellákban lévő értékek beolvasása (memória-olvasás), feldolgozása, és a cellák megváltoztatása (memória-írás) során egy program futtatása történik meg. A δ függvény felépítése révén megvalósíthatók ciklusok, elágazások, és gyakorlatilag minden sora egy-egy értékadó utasításnak felel meg. A Turing-gépek (és a többi automata) δ függvénye gyakorlatilag algoritmus-leíró eszköznek, primitív számítógépes programozási nyelvnek is tekinthető.

Felhívjuk a figyelmet arra, hogy minden fizikailag megvalósítható számítógép csak a Turing-gép vagy a Neumann-automata közelítő modelljének tekinthető. Hiszen ezekben a gépekben a külső tár terjedelme korlátos, míg a Turing-gépben és a Neumann- automatában ezt a szerepet egy végtelen szalag játssza. Korlátlan kapacitású tár technikai megvalósítása természetesen lehetetlen, de a tárkapacitás jelentős növelése nem csak kívánatos, hanem valóban lehetséges is. A technikai haladás mellett azonban alapvető jelentősége van az olyan tisztán matematikai kutatásoknak is, amelyek megvilágítják, milyen típusú gépek és algoritmusok a legalkalmasabbak egy adott típusú feladat gyorsabb megoldására.

Irodalomjegyzék:

Bach István:*Formális nyelvek*, Typotex, 2001, ISBN 9639132 92 6
http://www.typotex.hu/download/formalisnyelvek.pdf

Révész György: *Bevezetés a formális nyelvek elméletébe,* Akadémiai Kiadó, Budapest, 1979.

Demetrovics János – Jordan Denev – Radiszlav Pavlov: *A számítástudomány matematikai alapjai,* Nemzeti Tankönyvkiadó, Budapest, 1994.

B. A. Trahtenbrot: *Algoritmusok és absztrakt automaták,* Műszaki Könyvkiadó, Budapest, 1978.

Ádám András – Katona Gyula – Bagyinszki János: *Véges automaták,* MTA jegyzet, Budapest, 1972.

Varga László: *Rendszerprogramozás II.***,** Nemzeti Tankönyvkiadó, Budapest, 1994.

Fazekas Attila: *Nyelvek és automaták***,** KLTE jegyzet, Debrecen, 1998.