Nyelvek és automaták

Csima Judit Friedl Katalin

2013. augusztus

Ez a jegyzet a Budapesti Műszaki és Gazdaságtudományi Egyetem mérnökinformatikus hallgatói számára tartott *Nyelvek és Automaták* tantárgy alapján készült.

Köszönjük Horváth Ádámnak és Mészégető Balázsnak, hogy maguk is a tárgy hallgatóiként, a jegyzet első változatát elkészítették. Továbbá köszönjük Kövesdán Gábornak és a többi lelkes hallgatónak, hogy megjegyzéseikkel, észrevételeikkel segítették a javított változat megírását.

A jegyzetet időről-időre a továbbiakban is frissítjük, javítjuk. Az utolsó változat a cs.bme.hu/nya oldalról érhető el.

Tartalomjegyzék

1. Alapfogalmak, jelölések **5**

2. Véges automaták **9**
2.1. Véges automaták 9
2.2. Hiányos véges automaták 11
2.3. Nemdeterminisztikus véges automaták 13
2.4. Minimalizálás 18

3. Reguláris nyelvek, reguláris kifejezések **26**
3.1. Reguláris nyelvek 26
3.2. Reguláris kifejezések 35

4. Nyelvtanok, reguláris nyelvtanok **40**
4.1. Nyelvtanok 40
4.2. Chomsky-féle nyelvosztályok 44
4.3. Reguláris nyelvtanok 46

5. Reguláris nyelvek algoritmikus kérdései **50**
5.1. Beletartozás 50
5.2. Üresség 51
5.3. Egyenlőség 51
5.4. Diszjunktság 52
5.5. Végesség 52

6. Környezetfüggetlen nyelvek **54**
6.1. Levezetési fa 54
6.2. Egyértelműség 55
6.3. CF nyelvtanok átalakítása 58
6.3.1. ε-szabályok 58
6.3.2. Egyszeres szabályok 60
6.3.3. Felesleges szimbólumok 62

6.4. Normálformák . . . 64
6.4.1. Chomsky-normálforma . . . 64
6.4.2. Greibach-féle normálforma . . . 66
6.5. CF nyelvek tulajdonságai . . . 68

7. Veremautomaták **73**
7.1. Veremautomata-típusok . . . 73
7.2. Kapcsolat a környezetfüggetlen nyelvekkel . . . 79
7.3. Determinisztikus környezetfüggetlen nyelvek . . . 84

8. CF nyelvek algoritmikus kérdései **86**
8.1. Beletartozás . . . 86
8.2. Üresség . . . 88
8.3. Végesség . . . 89

9. Turing-gépek **90**
9.1. Egyszalagos, determinisztikus Turing-gép . . . 90
9.2. k-szalagos, determinisztikus Turing-gép . . . 94
9.3. Nemdeterminisztikus Turing-gép . . . 97
9.4. Felsorolós Turing-gépek . . . 98
9.5. Számoló Turing-gép . . . 101
9.6. A Turing-gépek számítási ereje . . . 105

10. Algoritmikus kiszámíthatóság, eldönthetőség **107**
10.1. Univerzális Turing-gép . . . 107
10.2. Nevezetes rekurzív és rekurzívan felsorolható nyelvek . . . 111
10.3. Műveletek rekurzív és rekurzívan felsorolható nyelvekkel . . . 115
10.4. Nyelvi tulajdonságok . . . 119
10.5. Dominó probléma . . . 124
10.6. Post megfeleltetési problémája . . . 127

11. A bonyolultságelmélet alapjai **129**
11.1. A számítás költsége . . . 129
11.2. Idő- és tárosztályok . . . 131
11.3. Az NP nyelvosztály . . . 138
11.4. NP-teljesség . . . 141

12. Nyelvtanok és Turing-gépek **155**
12.1. A 0. és az 1. Chomsky-féle nyelvosztály . . . 155
12.2. Turing-gépek és kapcsolatuk a CF nyelvekkel . . . 163
12.3. CF nyelvtanok algoritmikus kérdései – eldönthetetlenségi eredmények . . . 165

Irodalomjegyzék **167**

1. fejezet

Alapfogalmak, jelölések

1.1. Definíció *Egy tetszőleges nem üres, véges halmazt* ábécének *hívunk. Jelölése:* Σ.

Σ jelölheti például a magyar vagy az angol ábécé betűinek halmazát, a számjegyeket vagy egyéb véges halmazt. Ebben a jegyzetben legtöbbször a $\Sigma = \{ \mathtt{0}, \mathtt{1} \}$ halmazt fogjuk használni.

1.2. Definíció *A* Σ *ábécé elemeit* betűknek *vagy* karaktereknek *hívjuk. Egy* szó *a* Σ *elemeiből képzett tetszőleges véges hosszú sorozat. Az* y *szó* hosszát $|y|$ *jelöli.*
A Σ *ábécéből képezhető összes szó halmazának jelölése* Σ^*.

A nulla hosszú sorozat is szó, ennek jelölése ε. (Szokták rá a λ jelölést is használni.)

Így $\varepsilon \in \Sigma^*$ minden ábécére teljesül, és $\Sigma = \{ \mathtt{0}, \mathtt{1} \}$ esetén például $\mathtt{0} \in \Sigma^*$, $\mathtt{01100000} \in \Sigma^*$.

Minden $n \geq 0$ egész számra Σ^n jelöli a Σ elemeiből képezhető n hosszú sorozatok halmazát:

$$\Sigma^n = \{y \in \Sigma^* : |y| = n\}$$

A definíció szerint minden ábécé esetén $\Sigma^0 = \{ \varepsilon \}$ és például ha $\Sigma = \{ \mathtt{0}, \mathtt{1} \}$, akkor $\mathtt{01100000} \in \Sigma^8$.

Természetesen $\Sigma^n \subset \Sigma^*$. Ezzel a jelöléssel a Σ feletti összes szó halmaza

$$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$$

1.3. Definíció *A* Σ *ábécé feletti* nyelv*nek hívjuk a* Σ *elemeiből képezhető szavak egy tetszőleges (nem feltétlenül véges) részhalmazát, azaz* $L \subseteq \Sigma^*$

Például az üres halmaz $L = \emptyset$, illetve az üres szóból álló $L = \{ \varepsilon \}$ halmaz minden Σ esetén nyelv és maga $L = \Sigma^*$ is egy nyelv.

Ha $\mathtt{a} \in \Sigma$, akkor a két szóból álló $\{\mathtt{a}, \mathtt{aaa}\}$ halmaz egy nyelv. A csak $\mathtt{a}$ betűkből álló összes szó egy olyan $L \subseteq \Sigma^*$ nyelvet alkot, melynek végtelen sok eleme van.

Az így definiált nyelvfogalmat szokták formális nyelvnek is hívni, így megkülönböztetve a természetes nyelvektől (pl, magyar, angol).

A szavak Σ^* halmazán egy természetes művelet az alábbi:

1.4. Definíció *Két tetszőleges szó* $x = \mathtt{a}_1\mathtt{a}_2 \dots \mathtt{a}_n \in \Sigma^n \subset \Sigma^*$ *és* $y = \mathtt{b}_1\mathtt{b}_2 \dots \mathtt{b}_k \in \Sigma^k \subset \Sigma^*$ összefűzésén *vagy* konkatenációján *az* $\mathtt{a}_1\mathtt{a}_2 \dots \mathtt{a}_n\mathtt{b}_1\mathtt{b}_2 \dots \mathtt{b}_k \in \Sigma^{n+k} \subset \Sigma^*$ *szót értjük.*

Például $\Sigma = \{\, \mathtt{a}, \mathtt{b} \,\}$ esetén, legyen $x = \mathtt{ab}$ és $y = \mathtt{bbb}$. Ekkor $xy = \mathtt{abbbb}$ és $xxx = \mathtt{ababab}$.

A szokásos halmazműveletek (unió, metszet, komplementer) mellett a nyelveken is értelmezni lehet a konkatenációt:

1.5. Definíció *Az* $L_1, L_2 \subseteq \Sigma^*$ *nyelvek konkatenációján (összefűzésén) azt a nyelvet értjük, melynek szavai egy* L_1*-beli és egy* L_2*-beli szó összefűzéséből állnak, jelölése* L_1L_2*:*

$$L_1L_2 = \{w \in \Sigma^* : w = xy \text{ ahol } x \in L_1 \text{ és } y \in L_2\}$$

Például $\Sigma = \{\, \mathtt{a}, \mathtt{b} \,\}$ esetén legyen L_1 az $\mathtt{a}$-val kezdődő szavakból álló nyelv, L_2 pedig a $\mathtt{b}$-re végződőkből álló. Ekkor L_1L_2 az $\mathtt{a}$-val kezdődő és $\mathtt{b}$-re végződő szavakból áll, míg L_2L_1 az olyan szavakból áll, melyekben előfordul a $\mathtt{ba}$ részszó.

Minden ábécé felett teljesül, hogy

- ha $y = \varepsilon$, akkor $xy = yx = x$ tetszőleges x szóra;
- ha $L_2 = \{\varepsilon\}$, akkor $L_1L_2 = L_2L_1 = L_1$;
- ha $L_2 = \Sigma^*$ és $\varepsilon \in L_1$, akkor $L_1L_2 = L_2L_1 = \Sigma^*$;
- ha $L_2 = \emptyset$, akkor $L_1L_2 = L_2L_1 = \emptyset$.

1.6. Definíció *Az* $L \subseteq \Sigma^*$ *nyelv* tranzitív lezártja *az*

$$L^* = \{w \in \Sigma^* : w = x_1x_2 \dots x_k \text{ valamely } k \geq 0 \text{ számra, ahol } x_1, x_2, \dots, x_k \in L\}$$

Másként ez úgy is leírható, hogy ha $L^2 = LL$, $L^3 = L^2L$, általában $L^k = L^{k-1}L$, valamint $L^1 = L$ és $L^0 = \{\varepsilon\}$, akkor

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

Vegyük észre, hogy minden L nyelvre $\varepsilon \in L^*$ és $L \subseteq L^*$.

1.7. Példa *Legyen* $\Sigma = \{\,\mathtt{a}, \mathtt{b}\,\}$.

- *Ha* $L = \{\varepsilon\}$, *akkor* $L^* = L = \{\varepsilon\}$.
- *Ha* $L = \emptyset$, *akkor* $L^* = \{\varepsilon\}$.
- *Ha* $L = \Sigma^*$, *akkor* $L^* = L = \Sigma^*$
- *Ha* L *a páros sok* `a` *betűt tartalmazó szavakból áll, akkor* $L^* = L$.
- *Ha* L *a páratlan sok* `a` *betűt tartalmazó szavakból áll, akkor* L^* ε*-ból és azokból a nemüres szavakból áll, melyekben van* `a` *betű, formálisan* $L^* = (\Sigma^* \setminus \{b\}^*) \cup \{\varepsilon\}$.

Az összefűzés művelet az adott ábécé feletti szavakon és nyelveken is asszociatív, és ha az ábécé legalább két betűből áll, akkor nem kommutatív. Ezt úgy is fogalmazhatjuk, hogy a szavak halmaza, illetve a nyelvek halmaza ezzel a művelettel egységelemes félcsoportot alkot (egységelem az ε illetve az $\{\varepsilon\}$ nyelv). Ez lehetőséget ad a továbbiakban tárgyalt fogalmak algebrai vizsgálatára, de itt most nem ezt az irányt követjük, hanem elsősorban algoritmikus bonyolultsági szempontokra koncentrálunk. Például tekintsük a következő feladatot:

1.8. Példa *Egy szó betűit egymás után kapjuk és azt akarjuk, hogy a szó végére érve el tudjuk dönteni, a kapott szó beletartozik-e egy megadott nyelvbe. Kérdés, mekkora memóriára van ehhez szükségünk, azaz milyen (és mennyi) információt kell eltárolni a már látott betűkről (a szót csak egyszer olvashatjuk).*

- $\Sigma = \{\,\mathtt{0}, \mathtt{1}\,\}$ *és* L_1 = *nullával kezdődő szavak*
- $\Sigma = \{\,\mathtt{0}, \mathtt{1}\,\}$ *és* L_2 = *nullára végződő szavak*
- $\Sigma = \{\,\mathtt{0}, \mathtt{1}\,\}$ *és* L_3 = *azok a szavak, amelyekben az utolsó előtti karakter nulla*
- $\Sigma = \{\,\mathtt{0}, \mathtt{1}\,\}$ *és* L_4 = *páros sok egyest tartalmazó szavak*
- $\Sigma = \{\,(,)\,\}$ *és* L_5 = *a helyes zárójelezések*

Megoldás:

L_1 és L_2 esetén is elegendő 1 bit, mert csak az első, illetve az utolsó olvasott karaktert kell nyilvántartani.

L_3 esetén két bit elegendő: az utolsó és az utolsó előtti karakterre van szükségünk.

L_4 esetén ismét elég egyetlen bit, amellyel azt tartjuk nyilván, hogy eddig páros vagy páratlan sok nullát olvastunk.

L_5 esetén figyelnünk kell a zárójelezés mélységét. Egyrészt minden kinyitott zárójelet be kell zárnunk, másrészt a zárójelezés mélysége nem mehet nulla alá, azaz soha nem lehet

több csukó zárójel, mint nyitó. Itt tehát a pár nélküli nyitó zárójelek számát elegendő eltárolni, ehhez pedig egy n karakterből álló szó esetén $\lfloor \log(n+1) \rfloor$ bit elegendő. $\square$

Az utolsó példa elüt a többitől abban, hogy itt, legalább is ebben a megoldásban, logaritmikus sok bitet használunk a megelőző példák konstans bitjéhez képest. Később látni fogjuk, hogy nem a megoldásunk ügyetlen, hanem itt valóban nem elég a konstans bit.

2. fejezet

Véges automaták

Az egyik legegyszerűbb számítási modell a véges automata. A véges automaták több mindenre használhatók, mi most azzal a változattal foglalkozunk, ami egy nyelv felismerésére szolgál (az automatának nincs külön kimenete). Ez a típus is többféle módon definiálható, a kezdeti definíció után látni fogunk néhány további változatot, meg fogjuk mutatni, hogy számítási erejük ezeknek is ugyanaz, mint az első változatnak.

2.1. Véges automaták

2.1. Definíció *A* véges automata *egy* $M = (Q, \Sigma, \delta, q_0, F)$ *ötössel írható le, ahol:*

- Q *egy véges, nem üres halmaz. Ez az automata állapotainak halmaza.*
- Σ *egy véges, nem üres halmaz. Ez az automata ábécéje.*
- $\delta : Q \times \Sigma \to Q$*, az automata állapotátmeneti függvénye.*
- $q_0 \in Q$ *a kezdőállapot.*
- $F \subseteq Q$ *az elfogadó állapotok halmaza.*

A véges automata működése a következőképpen írható le egy adott $w = a_1 a_2 \dots a_n \in \Sigma^*$ szón:

2.2. Definíció *Az* $r_0, r_1, \dots, r_n (r_i \in Q)$ *állapotsorozat az* $a_1 a_2 \dots a_n$ *szóhoz tartozó* számítás*, ha* $r_0 = q_0$ *és* $r_i = \delta(r_{i-1}, a_i)$*, minden* $i = 1 \dots n$ *esetén.*

Az automatát tehát az $r_0 = q_0$ állapotból indítjuk, és minden olvasott karakternél az átmeneti függvény szerint meghatározzuk a következő állapotot. Az lesz számunkra az érdekes, hogy a számítás melyik állapotban ér véget. Pontosabban:

2.3. Definíció *Azt mondjuk, hogy az M véges automata* elfogadja *az n hosszú $w \in \Sigma^*$ szót, ha a w-hez tartozó számítás végén az utolsó r_n állapotra $r_n \in F$ teljesül.*

Egyébként M nem fogadja el *vagy* elutasítja *a $w \in \Sigma^*$ szót.*

2.4. Definíció *Az M véges automata által* elfogadott nyelv *azoknak a szavaknak a halmaza, amelyeket M elfogad. Jele: $L(M)$.*

A definícióból következik, hogy $L(M) \subseteq \Sigma^*$.

2.5. Feladat *Legyen $\Sigma = \{ \mathtt{0}, \mathtt{1} \}$. Adjunk meg egy olyan véges automatát, amely a nullára végződő szavakból álló nyelvet fogadja el!*

Megoldás: Adott, hogy $\Sigma = \{ \mathtt{0}, \mathtt{1} \}$. Legyen $Q = \{A, B\}$, a kezdőállapot A, az elfogadó állapotok halmaza $F = \{B\}$, az automata $M = (Q, \Sigma, \delta, A, F)$, ahol az állapotátmeneti függvény a következő:

$$\delta(A, \mathtt{0}) = B$$
$$\delta(A, \mathtt{1}) = A$$
$$\delta(B, \mathtt{0}) = B$$
$$\delta(B, \mathtt{1}) = A$$

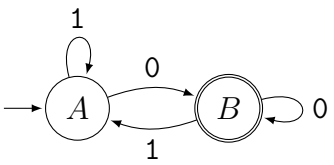
Vegyük észre, hogy ez a δ függvény olyan, hogy mindegy, hogy melyik állapotban volt előzőleg: ha `0` karaktert olvas az automata, akkor B állapotba, míg ha `1` karaktert olvas, akkor A állapotba kerül. Mivel B az egyetlen elfogadó állapot, ez az automata pontosan azokat a szavakat fogadja el, amelyek a `0` karakterre végződnek. □

Sokszor áttekinthetőbb, ha az állapotátmeneti függvényt táblázattal adjuk meg. A fenti esetben ez így néz ki:

	`0`	`1`
A	B	A
B	B	A

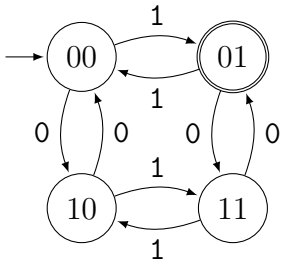
Az automatát irányított gráffal is ábrázolhatjuk. A gráf csúcsai az állapotoknak felelnek meg, az átmeneti függvényt az élek segítségével adjuk meg. A $\delta(q, \mathtt{a}) = q'$ átmenetnek egy, a q állapotnak megfelelő csúcsból a q' állapotnak megfelelő csúcsba menő irányított él felel meg, amihez az `a` címke tartozik. A kezdőállapotot egy bemenő nyíl jelöli, az elfogadó állapotokat dupla kör jelzi.

Az előző automata ebben a formában:



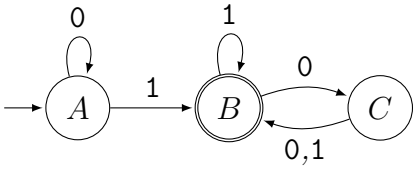
2.6. Feladat *Adjunk meg egy véges automatát, amely a páros sok nulla és páratlan sok egyest tartalmazó szavakat fogadja el! (*$\Sigma = \{\,\mathtt{0}, \mathtt{1}\,\}$*.*

Megoldás: Az automata az alábbi:



Könnyű látni, hogy az automata akkor kerül a 00 állapotba, ha páros sok `0` és páros sok `1` karaktert olvasott. A 01 állapot felel meg a páros sok `0` és páratlan `1` karakternek, stb. □

2.7. Feladat *Mely szavakat fogadja el az alábbi automata? (*$\Sigma = \{\,\mathtt{0}, \mathtt{1}\,\}$*)*



Megoldás: Az egyes állapotok jelentése:

- A: az eddig olvasott karakterek között nem volt egyes;
- B: volt már egyes, a karaktersorozat végén páros számú (esetleg nulla darab) nulla van
- C: volt már egyes, a karaktersorozat végén páratlan számú nulla van

Mivel itt most a B az elfogadó állapot, ezért az automata azokat a szavakat fogadja el, amelyekben van egyes, és a szó végén páros számú nulla van. □

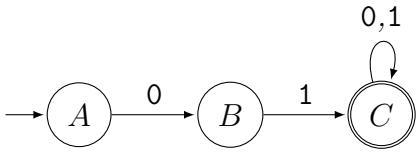
2.2. Hiányos véges automaták

Sokszor kényelmes lehet, ha az átmeneti függvényt nem kell mindenhol definiálni, a "hibás" átmeneteket elhagyva áttekinthetőbb lehet a kapott automata. Amikor hangsúlyozni akarjuk, hogy nem hiányos automatáról van szó, akkor a 2.1. definíció szerinti automatát *teljes véges automatának* hívjuk.

2.8. Definíció *A* hiányos véges automata *esetében a δ állapotátmeneti függvény nincs mindenhol definiálva.*

Egy hiányos automata, amennyiben számítása során egy q állapotban olyan $\mathtt{a}$ betűt olvas, amire $(q, \mathtt{a})$ helyen a δ nincs definiálva, akkor elakad. Egy ilyen számítás nem lehet elfogadó, mert az elfogadás feltétele továbbra is az, hogy a szó végére érve jusson elfogadó állapotba az automata. Az automata által elfogadott nyelv most is az elfogadott szavak összessége.

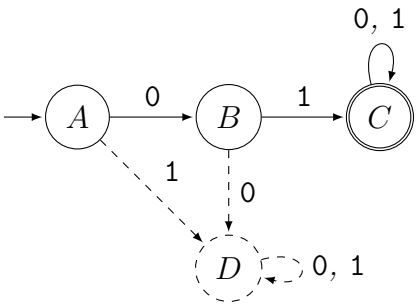
2.9. Példa *Az alábbi hiányos véges automata azokat a szavakat fogadja el, amelyek a* $\mathtt{01}$ *sorozattal kezdődnek (*$\Sigma = \{\, \mathtt{0}, \mathtt{1} \,\}$*).*



2.10. Tétel *Minden M hiányos véges automata kiegészíthető M' (teljes) véges automatává úgy, hogy az M és az M' által elfogadott nyelv ugyanaz legyen.*

Bizonyítás. Legyen $M = (Q, \Sigma, \delta, q_0, F)$. Vegyünk fel az M automatához egy új $q' \notin Q$ állapotot, azaz legyen $Q' = Q \cup \{\, q' \,\}$. Minden M-ben hiányzó átmenetet kössünk be ebbe az új q' állapotba, azaz legyen $\delta'(q, \mathtt{a}) = q'$, ha $q \in Q$ és ez az átmenet nem volt definiálva M-ben, egyébként legyen $\delta' = \delta$. Legyen továbbá $\delta'(q', \mathtt{a}) = q'$ minden $\mathtt{a} \in \Sigma$ betűre. A kezdőállapoton és az elfogadó állapotok halmazán ne változtassunk: $q_0' = q_0$ és $F' = F$. Az így kapott $M' = (Q', \Sigma', \delta', q_0', F')$ automata egy olyan teljes véges automata amely éppen az M által elfogadott $L(M)$ nyelvet fogadja el. (Amikor M számítása elakad, akkor M' a (nem elfogadó) q' állapotban csapdába kerül, onnan nem tud kijönni.) □

2.11. Feladat *Tegyük teljessé a 2.9. példa automatáját!*



2.3. Nemdeterminisztikus véges automaták

Nyelvek leírását sokszor megkönnyíti, ha nem csak elhagyhatunk állapotokat és átmeneteket, hanem az átmeneteknek nem is kell egyértelműeknek lenni, ugyanahhoz az állapot és karakter párhoz több következő állapot is lehetséges. Ez egy, a bonyolultságelméletben szokásos, nemdeterminisztikus számítási modellt eredményez. Bár ez az automatatípus nem ad közvetlenül algoritmust a nyelv szavainak felismerésére, de időnként sokkal kisebb automatát, rövidebb leírást biztosít.

2.12. Definíció *A* nemdeterminisztikus véges automatát *egy olyan* $M = (Q, \Sigma, \delta, q_0, F)$ *ötös adja meg, amelyben* Q, Σ, q_0 *és* F *jelentése ugyanaz, mint a 2.1. definícióban, azonban a* δ *átmeneti függvényre ezúttal az teljesül, hogy*

$$\delta(q, a) \subseteq Q,$$

ahol $q \in Q$ *és* $a \in \Sigma \cup \{\varepsilon\}$.

Vegyük észre, hogy ez a definíció a hiányos automatát is magában foglalja, hiszen lehet $\delta(q, a) = \emptyset$ is.

Vegyük észre továbbá azt is, hogy a nemdeterminisztikus véges automata nem csak arra ad lehetőséget, hogy egy (q, a) párra több átmeneti lehetőség legyen, hanem arra is, hogy olvasás nélkül, egy (q, ε) átmenettel átlépjünk egy újabb állapotba.

Megkülönböztetésül a 2.1. definíció szerinti automatát *determinisztikus* véges automatának is hívjuk. Ezek szerint a 2.1. definíció szerinti automata *teljes, determinisztikus* véges automata, ha meg akarjuk különböztetni a hiányos (2.8.) vagy nemdeterminisztikus változattól.

A nemdeterminisztikus véges automata működése a következőképpen írható le egy adott $w = a_1 a_2 \dots a_n \in \Sigma^*$ szón:

2.13. Definíció *Az* $r_0, r_1, \dots, r_m$ $(r_i \in Q)$ *állapotsorozat egy, az* $a_1 a_2 \dots a_n$ *szóhoz tartozó* számítás, *ha*

- $r_0 = q_0$,
- *amennyiben az automata az* $a_1 a_2 \dots a_{j-1}$ *szót elolvasva jutott* r_{i-1} *állapotba, akkor* $r_i \in \delta(r_{i-1}, \varepsilon) \cup \delta(r_{i-1}, a_j)$,
- *az utolsó* r_m *állapotba az egész szó elolvasása után jut az automata (az utolsó karakter elolvasása után esetleg még lehetnek* ε*-mozgások)*

Egy számítási lépés során tehát vagy a következő input karakter beolvasásával váltunk állapotot (azaz $r_i \in \delta(r_{i-1}, a_j)$) vagy az input olvasása nélkül, úgy nevezett ε-átmenettel (azaz $r_i \in \delta(r_{i-1}, \varepsilon)$).

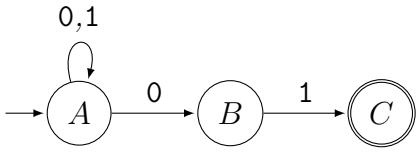
Vegyük észre, hogy az állapotok száma (amit itt m-mel jelöltünk) nem feltétlenül egyezik meg az olvasott karakterek számával (amit itt n-nel jelöltünk). Ez azért van így, mert az automata az input továbbolvasása nélkül is tud állapotot váltani az ε-mozgások segítségével.

2.14. Definíció *Azt mondjuk, hogy az M nemdeterminisztikus véges automata elfogadja az n hosszú $w \in \Sigma^*$ szót, ha van olyan w-hez tartozó számítás, aminek végén az utolsó r_m állapotra $r_m \in F$ teljesül.*

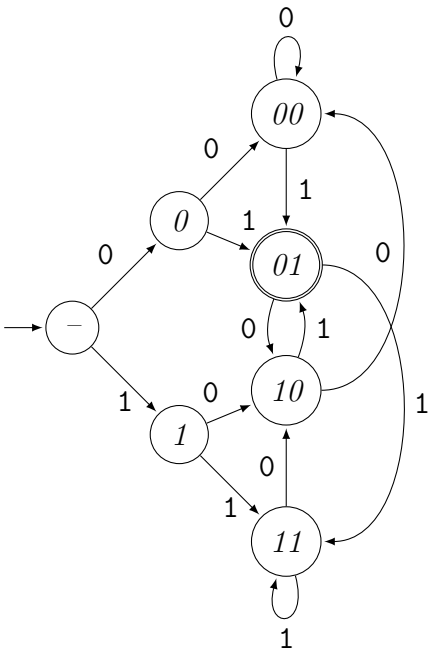
Egyébként M nem fogadja el *vagy* elutasítja *a $w \in \Sigma^*$ szót.*

Az M által elfogadott szavak halmazát itt is $L(M)$ jelöli. Természetesen $L(M) \subseteq \Sigma^*$.

2.15. Példa *A következő nemdeterminisztikus véges automata a* `01`*-re végződő szavakat fogadja el:*



Ebben a példában a nemdeterminizmust arra használjuk, hogy nem kell tudnunk, mikor jön az utolsó két karakter. Ha a számítás során korábban lépünk a B állapotba, akkor vagy a B vagy a C állapotban a számítás elakad (a szót nem tudjuk végigolvasni), azaz nem kapunk elfogadó számítást.
Ugyanehhez a nyelvhez (determinisztikus) véges automatát is készíthetünk. Ennek alapötlete, hogy az utolsó két karaktert tároljuk az állapotban, ezek között kell meghatározni az átmeneteket:



2.16. Tétel *Az L nyelvhez akkor és csak akkor létezik olyan M nemdeterminisztikus véges automata, melyre $L = L(M)$, ha van olyan M' determinisztikus véges automata is, amire $L = L(M')$.*

Bizonyítás. A visszafelé irány világos, hiszen egy véges automata egyben nemdeterminisztikus véges automata is, az M' véges automatához az $M = M'$ megfelelő lesz, mint nemdeterminisztikus véges automata.

A másik irányhoz megmutatjuk, hogyan lehet általában egy $M = (Q, \Sigma, \delta, q_0, F)$ nemdeterminisztikus véges automatából egy (determinisztikus) $M' = (Q', \Sigma', \delta', q_0', F')$ véges automatát készíteni, amire $L(M') = L(M)$. A konstrukció alapgondolata, hogy az új automata párhuzamosan követi M összes lehetséges számítását, és akkor fogad el, ha M-nek legalább az egyik számítása elfogadó.

Először tegyük fel, hogy M mindig olvas, azaz nincs $\delta(q, \varepsilon)$ típusú átmenet. Ekkor M'-ben legyen $\Sigma' = \Sigma$, az állapothalmaz pedig $Q' = \{R : R \subseteq Q\}$, azaz M' állapotai az M állapotainak részhalmazai lesznek, M' kezdőállapota legyen az M kezdőállapotából álló egy elemű halmaz, azaz $q_0' = \{q_0\}$, az elfogadó állapotok halmaza pedig azokból a részhalmazokból áll, melyekben van M-beli elfogadó állapot, azaz $F' = \{R : R \cap F \neq \emptyset\}$.

Az állapotátmenetet minden $R \subseteq Q$ és $a \in \Sigma$ esetén definiáljuk a következőképpen:

$$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$$

Most megmutatjuk, hogy valóban $L(M) = L(M')$. Ehhez először tegyük fel, hogy $w \in L(M)$, azaz w-t a nemdeterminisztikus véges automata elfogadja. Ekkor a 2.14.

definíció szerint van olyan állapotsorozat $r_0, r_1, \ldots, r_n$, ami egy elfogadó számításnak felel meg. Definíció szerint ekkor $r_0 = q_0 \in q_0' =: R_0$.

Mivel $r_0 = q_0 \in R_0$, majd $r_1 \in \delta(r_0, a_1) \subseteq \delta'(R_0, a_1) =: R_1$, illetve általában is $r_i \in \delta(r_{i-1}, a_i) \subseteq \delta'(R_{i-1}, a_i) =: R_i$, ezért az így kapott $R_1, R_2, \ldots, R_n \in Q'$ állapotsorozat a konstrukció szerint M' számítását írja le az adott szón.

A feltevés szerint M számítása elfogadó volt, azaz $r_n \in F$, ezért $R_n \cap F \neq \emptyset$, tehát R_n az M' véges automatának elfogadó állapota, vagyis M' elfogadja a w szót.

Másrészről, ha $w \in L(M')$, azaz a determinisztikus véges automata elfogadja a w szót, akkor a w szóhoz az M' automatának egy olyan $R_0, R_1, \ldots, R_n$ számítása tartozik, melyben $R_n \in F'$. Ekkor F' meghatározása szerint van olyan $r_n \in R_n$, hogy $r_n \in F$. Mivel $r_n \in R_n = \delta'(R_{n-1}, a_n) = \cup_{r \in R_{n-1}} \delta(r, a_n)$, van tehát olyan $r_{n-1} \in R_{n-1}$, melyre $r_n \in \delta(r_{n-1}, a_n)$. Általában pedig minden $i = n, n-1, \ldots, 1$ esetén vannak olyan $r_i \in R_i$ állapotok, melyekre $r_i \in \delta(r_{i-1}, a_i)$. A képzési szabály szerint ahhoz hogy az $r_0, r_1, \ldots, r_n \in Q$ állapotsorozat egy elfogadó számítása legyen az M nemdeterminisztikus véges automatának, már csak annyi hiányzik, hogy r_0 az M q_0-lal jelölt kezdőállapota legyen, ami pedig az $r_0 \in R_0 = \{q_0\}$ miatt teljesül.

Most nézzük meg hogyan módosul a konstrukció, ha az M nemdeterminisztikus véges automatának $\delta(q, \varepsilon)$ típusú átmenetei is vannak! Ehhez vezessük be az alábbi jelölést: tetszőleges $R \subseteq Q$ állapothalmazra jelölje $E(R) \subseteq Q$ azoknak az állapotoknak a halmazát, ahova újabb karakter olvasása nélkül el lehet jutni az R halmazból. Ezek szerint tehát $E(R)$ az R halmaz elemeit és még azokat a Q-beli elemeket tartalmazza, ahova ε címkéjű éleket használva (esetleg többet is egymás után) R-ből el lehet jutni.

Ekkor a fenti konstrukcióban az átmeneti függvényt a következőképpen kell módosítani:

$$\delta'(R, a) = \bigcup_{r \in R} E(\delta(r, a))$$

az új kezdőállapot pedig $q_0' = E(\{q_0\})$. Szemléletesen ez azt jelenti, hogy a determinisztikus automata átmeneteiben azt tároljuk, hogy a nemdeterminisztikus automata egy karakter beolvasásával (és még esetleg néhány ε lépéssel) hova tud eljutni.

Az előző konstrukcióhoz hasonlóan igazolható az új konstrukció helyessége is. □

2.17. Feladat *A fenti konstrukcióval készítsük el a 2.15. példában szereplő, a* 01 *végződésű szavakat elfogadó nemdeterminisztikus automatából a determinisztikus véges automatát!*

Megoldás: Az eredeti automata nem tartalmaz ε-átmenetet, ezért használhatjuk a konstrukció első változatát. Az új állapothalmaz az $\{A, B, C\}$ három elemű halmaz összes nemüres részhalmaza, de vegyük észre, hogy a determinisztikus automata kezdőállapotából nem elérhető állapotokat nyugodtan elhagyhatjuk, ezzel az elfogadott nyelvet nem befolyásoljuk (de az automatánkat egyszerűsítjük).

Az $\{A\}$ kezdőállapotból induló átmenetek:

$$\delta'(\{A\}, \mathtt{0}) = \{A, B\}$$
$$\delta'(\{A\}, \mathtt{1}) = \{A\}$$

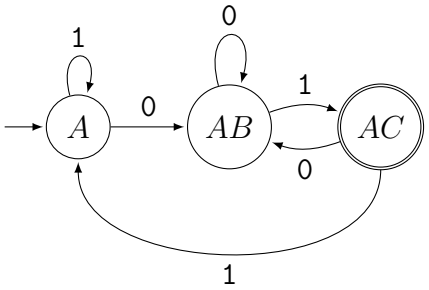
Ezért felvesszük az $\{A, B\}$ állapotot is:

$$\delta'(\{A, B\}, \mathtt{0}) = \{A, B\} \cup \emptyset = \{A, B\}$$
$$\delta'(\{A, B\}, \mathtt{1}) = \{A\} \cup \{C\} = \{A, C\}$$

és a kapott $\{A, C\}$ állapotból induló átmenetek:

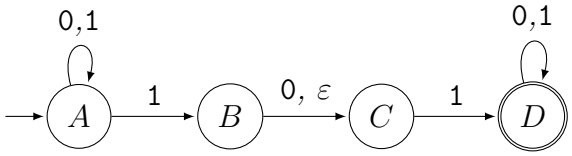
$$\delta'(\{A, C\}, \mathtt{0}) = \{A, B\} \cup \emptyset = \{A, B\}$$
$$\delta'(\{A, C\}, \mathtt{1}) = \{A\} \cup \emptyset = \{A\}$$

A kapott determinisztikus automata ábrája (az állapotokban a halmazjelet elhagytuk)

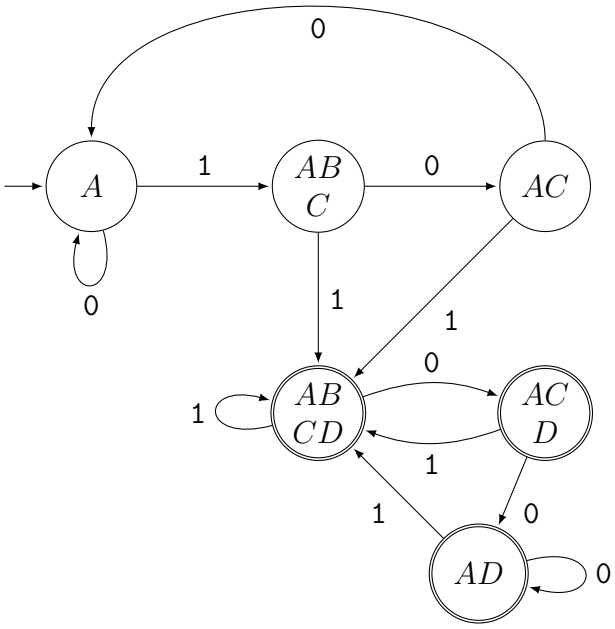


□

2.18. Feladat *Milyen nyelvet fogad el az alábbi automata? Készítsük el az automatából a determinisztikus véges automatát!* ($\Sigma = \{\, \mathtt{0}, \mathtt{1} \,\}$)



Megoldás: Az automata pontosan azokat a szavakat fogadja el, amelyekben előfordul az `101` vagy `11` részszó, mert csak ezekkel tudunk átjutni A-ból (a kezdőállapotból) D-be (az egyetlen elfogadó állapotba).
A determinizált automata:



□

Megjegyezzük, hogy míg a konstrukció alapján 2^4 állapot keletkezne, a kezdőállapotból csak a felrajzolt 6 állapot érhető el, tehát a többire nincs is szükség. Jelen esetben könnyen kaphatunk még kevesebb állapotú determinisztikus véges automatát ha észrevesszük, hogy az elfogadó állapotok egyetlen állapottá való összevonásával is determinisztikus véges automatát kapunk. Arról, hogy általában hogyan lehet az állapotok számát csökkenteni, a következő, a 2.4 fejezetben lesz szó.

2.4. Minimalizálás

Ebben a részben azzal foglalkozunk, hogyan lehet minél kevesebb állapotú véges automatát készíteni egy adott nyelvhez. Láttuk, hogy ugyanahhoz a nyelvhez készített determinisztikus és nemdeterminisztikus véges automaták állapotainak száma között nagy különbség lehet. Most a vizsgálódást a determinisztikus véges automaták körében végezzük el, ehhez szükségünk lesz néhány eszközre.

2.19. Definíció *Legyen $L \subseteq \Sigma^*$ egy nyelv. Tetszőleges $x \in \Sigma^*$ szóra vezessük be a következő jelölést:*

$$L/x = \{z \in \Sigma^*, \quad xz \in L\}$$

Szemléletesen ez azt jelenti, hogy L/x az összes olyan Σ^* szót tartalmazza, amit x után írva L-beli szót kapunk.

2.20. Példa *Az* $L_1 \subset \{0,1\}^*$ *nyelv álljon a* 01*-gyel kezdődő szavakból. Ekkor*

- $L_1/010 = \Sigma^*$
- $L_1/10 = \emptyset$,

mert akárhogy folytatjuk a 010 *sorozatot mindig* L_1*-beli lesz a szó, viszont akárhogy folytatjuk az* 10 *sorozatot, soha nem kapunk* L_1*-beli szót.*

Az $L_2 \subset \{0,1\}^*$ *nyelv álljon a páros sok egyest tartalmazó szavakból. Ekkor könnyen látszik, hogy*

- $L_2/011 = L_2$
- $L_2/01 = \overline{L_2}$.

2.21. Definíció *Azt mondjuk, hogy az* $x \in \Sigma^*$ *és az* $y \in \Sigma^*$ *szavak az* L *nyelvvel* megkülönböztethetetlenek*, ha* $L/x = L/y$*, különben megkülönböztethetőnek nevezzük őket.*

A definíció alapján az, hogy x és y az L nyelvvel megkülönböztethető pontosan azt jelenti, hogy van olyan $z \in \Sigma^*$ szó, melyre vagy $xz \in L$ és $yz \notin L$, vagy $xz \notin L$ és $yz \in L$, azaz van olyan közös folytatásuk, mellyel kiegészítve L-re nézve másként viselkednek.

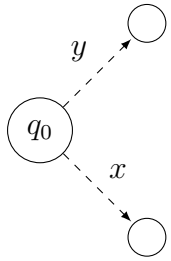
2.22. Példa *Legyen* $L \subseteq \{0,1\}$ *a* 01*-re végződő szavakból álló nyelv. (Már láttunk rá véges automatát a 2.15. és a 2.17. példában.) Tekintsük a következő három szót:* $x_1 = 0$*,* $x_2 = 1$ *és* $x_3 = 01$*. Az* x_1 *és* x_2 *szavakat az* L *megkülönbözteti, ez látható például a* $z_1 = 1$ *választással, hiszen* $x_1 z_1 = 01 \in L$*, míg* $x_2 z_1 = 11 \notin L$*. Hasonlóan a* $z_2 = \varepsilon$ *választás mutatja, hogy* x_1 *és* x_3 *is megkülönböztethető, mert* $x_1 z_2 = 0 \notin L$*, de* $x_3 z_2 = 01 \in L$*. Ugyanez a* z_2 *jó lesz* x_2 *és* x_3 *megkülönböztetésekor is.*

2.23. Megjegyzés *A definícióból következik, hogy minden* L *nyelvre* $L/\varepsilon = L$*, illetve hogy egy tetszőleges* $x \in \Sigma^*$ *szó saját magától megkülönböztethetetlen.*

A továbbiakban megpróbálunk kapcsolatot találni aközött, hogy egy reguláris nyelvben hány páronként megkülönböztethetetlen szó van illetve aközött, hogy egy, a nyelvet elfogadó véges automatának hány állapota lehet. Ez a vizsgálódás fog elvezetni minket ahhoz, hogy egy lehető legegyszerűbb véges automatát szerkesszünk egy reguláris nyelvhez.

2.24. Tétel *Legyen* M *egy (determinisztikus) véges automata és* $L = L(M) \subseteq \Sigma^*$*. Ha* Σ^* *elemei között van* t *darab, az* L *nyelvvel páronként megkülönböztethető szó, akkor* M*-nek legalább* t *állapota van.*

Bizonyítás. Ha x és y két olyan szó, ami megkülönböztethető az L nyelvvel, akkor az M automatában a kezdőállapotból kiindulva az x, illetve az y szó hatására különböző állapotokba kell jutnunk, különben xz és yz minden lehetséges z szóra pontosan ugyanakkor vezetne elfogadó állapotba.



Ha van t darab páronként megkülönböztethető szó, akkor tehát ezek páronként különböző állapotokba juttatják az M automatát, azaz kell, hogy legyen legalább ennyi különböző állapot. □

2.25. Példa *Legyen $L = \{ww : w \in \Sigma^*\}$, vagyis a „mindent kétszer mond" nyelv. Ekkor az L nyelvvel bármely két különböző k hosszú x_1 és x_2 szó megkülönböztethető egymástól, hiszen $x_1x_1 \in L$, de $x_2x_1 \notin L$. Tehát ha lenne az L nyelvhez véges automata, annak legalább 2^k állapota lenne. Mivel ennek minden k-ra teljesülnie kell, ezért ehhez a nyelvhez nem létezik véges automata.*

Figyeljük meg, hogy egy adott L nyelvvel való a megkülönböztethetetlenség egy olyan reláció a szavakon, amely:

- szimmetrikus, hiszen ha az egyik szó megkülönböztethetetlen a másiktól, akkor a másik is az egyiktől,
- reflexív, mivel minden szó megkülönböztethetetlen saját magától, és
- tranzitív, azaz ha x_1 és x_2, valamint x_2 és x_3 megkülönböztethetetlen, akkor x_1 és x_3 is ilyen.

A fenti tulajdonságok miatt az L nyelvvel való megkülönböztethetetlenség egy ekvivalenciareláció, amely a szavakat ekvivalenciaosztályokba osztja, úgy hogy az egyes osztályokba tartozó szavak mind páronként megkülönböztethetetlenek egymástól, a különböző osztályokba tartozó szavak pedig megkülönböztethetőek.

A továbbiakhoz egy M determinisztikus véges automata átmeneti függvényét rekurzívan kiterjesztjük a szavakra is. Egy q állapotra és egy n hosszú x szóra legyen

$$\overline{\delta}(q,x) = \begin{cases} \delta(q,x) & \text{ha} \quad n = 1 \\ \delta(r,x_n) & \text{ha} \quad x = yx_n \text{ ahol } x_n \in \Sigma, \text{ és } r = \overline{\delta}(q,y) \end{cases}$$

$\overline{\delta}$ tehát azt fejezi ki, hogy egy adott állapotból egy adott szót beolvasva melyik állapotba jut az automata.

A szavakon levő megkülönböztethetőség mintájára osztályozhatjuk az automata állapotait is:

2.26. Definíció *Az M véges automata p és q állapotai ekvivalensek, ha minden $y \in \Sigma^*$ esetén $\overline{\delta}(p, y)$ pontosan akkor elfogadó állapota az M automatának, ha $\overline{\delta}(q, y)$ elfogadó állapot.*

Ez könnyen láthatóan ekvivalenciareláció az állapotok halmazán.

Most megmutatjuk, hogy egy determinisztikus teljes véges automata esetén hogyan lehet meghatározni az ekvivalens állapotokat. Ez a felosztás fog elvezetni ahhoz a lehető legegyszerűbb automatához, amit az automata nyelvéhez készíteni lehet.

Egy adott véges automatánál az állapotok ekvivalenciaosztályai a következő egyszerű eljárással meghatározhatóak. Az ötlet az, hogy fokozatosan meghatározzuk a legfeljebb $0, 1, 2, \ldots$ hosszú y szavak alapján az állapothalmaz partícióját ekvivalens állapotokra.

2.4.1 Algoritmus (Ekvivalencia osztályok meghatározása)
Bemenet: *M véges automata (teljes, determinisztikus)*

1. *A kezdeti partíció legyen $A_1 = Q - F$ és $A_2 = F$ (ahol F az M elfogadó állapotainak halmaza). Ez a két állapothalmaz már a 0 hosszú szóval megkülönböztethető.*

2. *Egy meglevő $Q = A_1 \cup \cdots \cup A_k$ partíció esetén az A_i halmazokat bontsuk tovább úgy, hogy $p, q \in A_i$ pontosan akkor maradjon együtt, ha minden $\mathtt{a} \in \Sigma$ betűre $\delta(p, \mathtt{a})$ és $\delta(q, \mathtt{a})$ a Q partíció ugyanazon halmazában van.*

Amennyiben van olyan A_i, amit szétbontottunk, akkor a kapott új partícióra ismételjük a fenti eljárást 2. pontját, ha pedig nincs változás, akkor az eljárás leáll.

2.27. Állítás *A fenti eljárás véges és az eljárás végén a kapott partíció elemei éppen a keresett ekvivalenciaosztályok.*

Bizonyítás. Minden változás esetén a meglevő partícióban szereplő halmazok száma legalább eggyel nő. Mivel ezek a halmazok diszjunktak, legfeljebb $|Q|$ darab halmaz keletkezhet. A kiinduló partíció két elemű, tehát $|Q| - 1$ darab 2. típusú lépésen belül az eljárás biztosan véget ér.

A helyesség bizonyításához definiáljuk a j-ekvivalencia fogalmát. A p és q állapotok legyenek j-ekvivalensek, ha a legfeljebb j hosszú szavakra ekvivalensek (azaz minden $|y| \leq j$ esetén $\overline{\delta}(p, y)$ és $\overline{\delta}(q, y)$ egyformán elfogadó vagy nem).

Világos, hogy ha a p és q állapotok j-ekvivalensek, akkor ℓ-ekvivalensek is minden $\ell \leq j$ esetén. Másrészt, ha j-ekvivalensek, és minden $\mathtt{a}$ betűre $\delta(p, \mathtt{a})$ és $\delta(q, \mathtt{a})$ is j-ekvivalensek, akkor a p és q állapotok $(j+1)$-ekvivalensek is. Következésképp, ha a j-ekvivalenciával kapott felosztás megegyezik a $(j+1)$-ekvivalenciával kapott felosztással, akkor a j-ekvivalenciával kapott felosztás már az összes szóra vett ekvivalenciának megfelelő felosztással egyezik meg.

Vegyük észre, hogy a kezdeti partíció éppen a 0-ekvivalens osztályokból áll. Ezek után könnyen látszik, hogy a j-edik körben a j-ekvivalens osztályokat kapjuk meg. Akkor nincs már változás (az eljárás akkor áll le), ha a j-ekvivalenciával kapott felosztás

megegyezik a $(j+1)$-ekvivalenciával kapott felosztással, vagyis megkaptuk a keresett ekvivalenciaosztályokat. □

2.28. Definíció *Egy L nyelvhez tartozó* minimálautomata *egy olyan M véges (teljes, determinisztikus) automata, amelyre $L(M) = L$ és M az ilyen automaták közül a legkevesebb állapottal rendelkezik.*

2.29. Tétel *Ha az L nyelvhez van véges automata, akkor van minimálautomata is, és ez egyértelmű.*

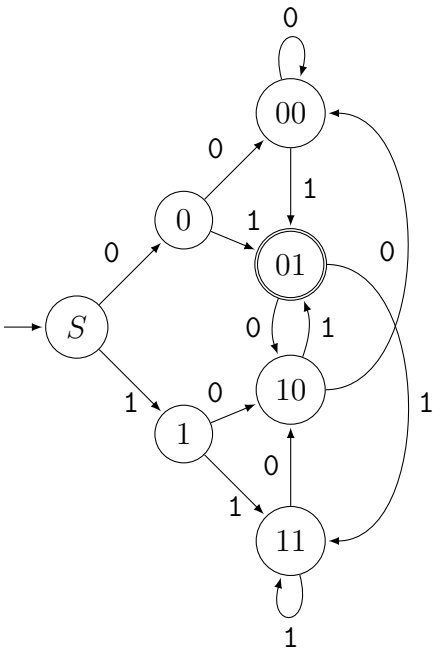
Bizonyítás. Ha L-hez van egy M determinisztikus véges automata, akkor az ebből az automatából az előbbi algoritmussal kapott osztályokon definiálhatunk egy véges automatát, melynek állapotai az előbb kapott ekvivalencia-osztályok, az állapotátmenetek pedig legyenek az állapothalmazok közötti átmenetek. (A konstrukció miatt egy betű egy ekvivalencia-osztály minden eleméből ugyanabba az osztályba visz, tehát az így definiált hozzárendelés tényleg tekinthető az osztályokon értelmezett átmeneti függvénynek.) Legyen a kezdőállapot az, ahol az eredeti M automata kezdőállapota van, elfogadó állapotok pedig az elfogadó állapotokat tartalmazó osztályok. (A konstrukcióból következik, hogy ha egy ekvivalencia osztályon belül van elfogadó állapot, akkor az összes állapot elfogadó.)

A konstrukcióból következik az is, hogy ezzel egy determinisztikus, teljes véges automatát kaptunk, amiből hagyjuk el azokat az állapotokat, amelyek a kezdőállapotból nem érhetők el. Az így keletkezett M' automatára teljesül, hogy $L(M') = L(M)$. Megmutatjuk, hogy M' egy minimálautomata.

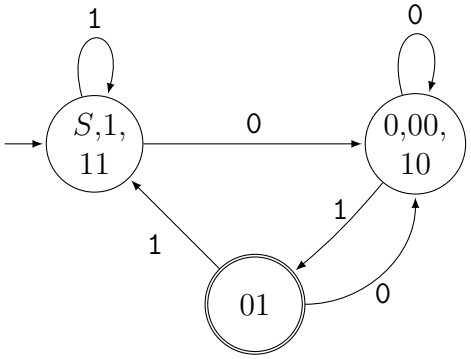
Tekintsünk két olyan $x, y \in \Sigma^*$ szót, hogy $L/x = L/y$. Tegyük fel, hogy $\overline{\delta}(q_0, x) = p$ és $\overline{\delta}(q_0, y) = q$. Ekkor az M automatában a p és q állapotok ekvivalensek (mert x és y minden közös folytatással ugyanúgy viselkedik), tehát biztos, hogy az M'-ben ezek egy állapotba kerülnek, azaz van olyan q' állapot M'-ben, hogy $\overline{\delta}(q_0, x) = \overline{\delta}(q_0, y) = q'$. Tehát M' állapotainak száma legfeljebb annyi, mint a nyelv szavain a megkülönböztethetetlenséggel kapott ekvivalenciaosztályok száma, ami a 2.24. állítással együtt garantálja, hogy M' állapotainak száma a lehető legkisebb.

A minimálautomata egyértelműsége onnan látszik, hogy állapotai megfelelnek a szavakon levő ekvivalencia-osztályoknak, így ezek egyértelműen meghatározottak, s így a közöttük levő átmenetek és a teljes véges automata is az. □

2.30. Feladat *A algoritmus használatával minimalizáljuk a korábbi (2.15. feladat) automatánkat, amely a* `01`*-re végződő szavakat tartalmazó nyelvet fogadja el.*



Megoldás: Először két állapothalmazunk lesz: $A_1 = \{S, 0, 00, 10, 11, 1\}$, és $A_2 = \{01\}$. Minden A_1-beli állapot esetén `0` hatására A_1-ben maradunk, míg `1` hatására S, 11 és 1 állapotokból A_1-be, de a 0, 00 és 10 állapotokból A_2-be jutunk. Ezért az A_1 állapotot felosztjuk. Az A_2 halmazt biztos nem kell tovább osztani, mivel eleve csak egy állapotot tartalmaz. Az új partíció: $A_1 = \{S, 11, 1\}$, $A_2 = \{01\}$, $A_3 = \{0, 00, 10\}$. Az így kialakult A_1, A_2 és A_3 állapotok az ábécé minden elemére egységesen viselkednek, így ezeket már nem osztjuk tovább, az algoritmus véget ért. A végeredményül kapott automata az alábbi:



□

A fenti eljárás, rajzolgatva, nem túl nagy automatákra könnyedén elvégezhető. Implementálásra alkalmasabb azonban az alábbi változat:

2.4.2 Algoritmus (Táblázatos módszer minimalizálásra)
Bemenet: *M véges automata (teljes, determinisztikus) leírása*

- *Vegyünk fel egy, az állapotokkal indexelt $|Q| \times |Q|$ méretű T tömböt, aminek elemei kezdetben üresek. (Valójában ennek csak az átló alatti részére van szükségünk.)*
- *Legyen $T[p, q] = 0$, ha p és q közül az egyik elfogadó állapot a másik nem.*
- *Az i-edik körben ($i \geq 1$):*
  ha $T[p, q]$ üres, de van olyan $\mathtt{a} \in \Sigma$, hogy a $p' = \delta(p, \mathtt{a})$ és $q' = \delta(q, \mathtt{a})$ állapotokra $T[p', q']$ nem üres, akkor legyen $T[p, q] = i$.
- *Az eljárás véget ér, ha T egy körben nem változik.*

Az eljárás helyessége abból következik, hogy $T[p, q] = i$ jelentése az, hogy a p és q állapotok nem i-ekvivalensek, de j-ekvivalensek minden $j < i$ értékre.

A módszerből következik, hogy az eljárás legkésőbb $i = n - 1$ esetén véget ér. Az eljárás végén üresen maradt mezők jelzik az ekvivalens állapotpárokat.

2.31. Példa *Nézzük meg a módszert az előző példán! Ahogy már megjegyeztük, a táblázatnak csak az átló alatti részével foglalkozunk.*

A kezdeti üres táblázat, a sorok elején és az oszlopok alján feltüntetve a megfelelő állapotok nevei

S							
0							
1							
00							
01							
10							
11							
	S	***0***	***1***	***00***	***01***	***10***	***11***

Beírjuk az elfogadó–nem elfogadó párokhoz a nullákat.

S							
0							
1							
00							
01	*0*	*0*	*0*	*0*			
10					*0*		
11					*0*		
	S	***0***	***1***	***00***	***01***	***10***	***11***

Az első lépésben például a 00 *és* S *pár esetén* `0` *hatására a* $(00, 0)$ *mezőbe, míg* `1` *hatására a* $(01, 1)$ *mezőbe jutunk, melyek közül az utóbbi nem üres, ezért a* $(00, S)$ *mezőbe egy egyest kell írnunk. A teljes első menet után a táblázat állapota:*

S							
0	*1*						
1		*1*					
00	*1*		*1*				
01	*0*	*0*	*0*	*0*			
10	*1*		*1*		*0*		
11		*1*		*1*	*0*	*1*	
	S	***0***	***1***	***00***	***01***	***10***	***11***

Újra végigmenve a táblázaton láthatjuk, hogy semmi sem változik, ezzel az algoritmus véget ér.

A táblázatból tetszőleges állapotpárra kiolvasható, hogy ekvivalensek-e. Például S *és* 1 *igen (mert a mező üres), de* S *és* 0 *nem.*

A táblázatos módszer segítségével az algoritmus lépésszáma is megbecsülhető:

2.32. Következmény *Minden véges automatából* $O(|Q|^3 \cdot |\Sigma|)$ *lépésben megkapható a minimálautomata, ami ugyanazt a nyelvet fogadja el.*

Bizonyítás. Egy $O(|Q|^2)$ méretű táblázatot töltünk ki. Kevesebb, mint $|Q|$ kör lehetséges, minden körben a táblázat minden elemét az összes betűvel meg kell vizsgálni, ebből következik az $O(|Q| \cdot |Q|^2 \cdot |\Sigma|)$ becslés. □

Megjegyezzük, hogy van olyan algoritmus, ami a minimalizálást $O(|Q|^2|\Sigma|)$ lépésben megoldja.

3. fejezet

Reguláris nyelvek, reguláris kifejezések

3.1. Reguláris nyelvek

A véges automatákkal felismerhető nyelvek fontos szerepet játszanak a programozási nyelvek elméletében és számos más területen. Ebben a fejezetben ezt az igen fontos nyelvosztályt fogjuk több szempontból megvizsgálni.

3.1. Definíció *Egy $L \subseteq \Sigma^*$ nyelvet* reguláris*nak hívunk, ha létezik olyan determinisztikus M véges automata, hogy $L(M) = L$.*

A determinisztikus és nemdeterminisztikus véges automaták egyenértékűsége miatt nem szükséges a definícióban kikötni, hogy determinisztikus véges automatának kell létezni.

3.2. Példa *Legyen $\Sigma = \{\,\mathtt{0}, \mathtt{1}\,\}$.*

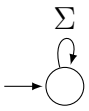
- *Ha L a páros sok nullát és páratlan sok egyest tartalmazó szóból álló nyelv, akkor L reguláris (2.6. feladat).*
- *Ha L a $\mathtt{01}$-re végződő szavakból álló nyelv, akkor L reguláris (2.15. példa).*
- *Ha $L = \{ww : w \in \Sigma^*\}$, akkor L nem reguláris (2.25. példa).*

3.3. Tétel *Tetszőleges Σ ábécé esetén az alábbi nyelvek mindegyike reguláris.*

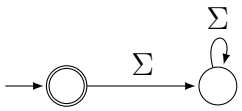
1. $L = \emptyset$
2. $L = \{\varepsilon\}$
3. $L = \Sigma^*$

Bizonyítás. Megadunk egy-egy lehetséges automatát a három nyelvre. Az állapotátmenetet jelképező nyilakra írt Σ azt jelzik, hogy minden betű esetén azt a nyilat kell követni.

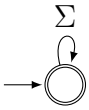
1. Az alábbi automata semmit sem fogad el, hisz nincs elfogadó állapota ($L = \emptyset$):



2. Az alábbi automata csak az üres szót fogadja el ($L = \{\varepsilon\}$):



3. Ez pedig minden szót elfogad ($L = \Sigma^*$):



□

Most megmutatjuk, hogy a szokásos halmazműveletekre a reguláris nyelvek halmaza zárt.

3.4. Tétel *Ha L_1 és L_2 reguláris nyelv, akkor az alábbiak mindegyike reguláris*

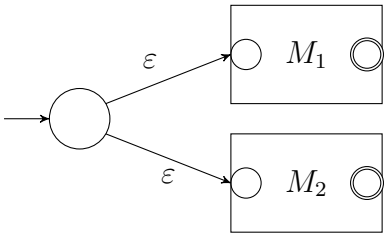
- $L = L_1 \cup L_2$,
- $L = L_1 \cap L_2$,
- $L = L_1 \setminus L_2$.

Bizonyítás. Mivel L_1 és L_2 reguláris, van hozzájuk M_1, illetve M_2 véges automata, melyekre $L(M_i) = L_i$. Legyen

$$\begin{aligned} M_1 &= (Q_1, \Sigma_1, \delta_1, q_1, F_1) \\ M_2 &= (Q_2, \Sigma_2, \delta_2, q_2, F_2) \end{aligned}$$

Megmutatjuk, hogyan lehet ezekből az L nyelvhez egy $M = (Q, \Sigma, \delta, q_0, F)$ véges automatát készíteni, ahol $\Sigma = \Sigma_1 \cup \Sigma_2$.

unió Először nézzünk egy nemdeterminisztikus automatát! Az ötlet az, hogy egyetlen automatában „egymás mellé” rakjuk M_1-et és M_2-t. Egy új kezdőállapotból M, még mielőtt bármit olvasna, az M_1 vagy az M_2 kezdőállapotába lép (nemdeterminisztikusan), ott lép az input alapján és végül elfogad, ha a megfelelő M_i elfogadó állapotában ér véget ($F = F_1 \cup F_2$).



Világos, hogy $L(M) = L_1 \cup L_2$, és mivel tudjuk, hogy egy nemdeterminisztikus véges automatából készíthető ugyanazt a nyelvet felismerő determinisztikus (2.16. tétel), ezzel az $L = L_1 \cup L_2$ nyelv regularitását beláttuk.

Lehetséges azonban rögtön egy determinisztikus véges automatát is készíteni $L_1 \cup L_2$-re. Ebben az esetben feltételezzük, hogy M_1 és M_2 determinisztikus és $\Sigma_1 = \Sigma_2$. Ha ez utóbbi tulajdonság nem teljesül, akkor M_i-t a Σ ábécére nézve hiányos véges automatának tekinthetjük, amit a konstrukció előtt a 2.10. tétel segítségével teljessé kell tenni. (Ezek a lépések az előző konstrukcióhoz nem kellettek.)

Az M véges automata „párhuzamosan” követi a két automata lépéseit és akkor fogad el, ha valamelyik M_i elfogad. Ehhez legyen

$$\begin{aligned} Q &= Q_1 \times Q_2, \\ q_0 &= (q_1, q_2) \\ F &= \{(q, q') : q \in F_1 \text{ vagy } q' \in F_2\} \\ \delta((q, q'), a) &= (\delta_1(q, a), \delta_2(q', a)), \end{aligned}$$

azaz az első „koordinátában” M_1, a másodikban M_2 szerint mozgunk, így egy w bemeneten valóban pontosan akkor érünk elfogadó állapotba, ha $w \in L = L_1 \cup L_2$.

metszet Az előző, determinisztikus konstrukció kis módosítással itt is jó lesz, elegendő az elfogadó állapotokat úgy megválasztani, hogy az új véges automata akkor fogadja el a w szót, ha ezt M_1 és M_2 is elfogadja

$$F = \{(q, q') : q \in F_1 \text{ és } q' \in F_2\}.$$

különbség Ez is egyszerűen megkapható az előzőből, csak most

$$F = \{(q, q') : q \in F_1 \text{ és } q' \notin F_2\}$$

a jó választás. □

3.5. Következmény *Ha az L nyelv reguláris, akkor az $\overline{L}$ nyelv is reguláris.*

Bizonyítás. A 3.3. és a 3.4. tételek egyszerű következménye, hogy $\overline{L} = \Sigma^* \setminus L$ reguláris.□

3.6. Megjegyzés *Közvetlenül is létrehozhatunk $\overline{L}$-t elfogadó automatát, ha L determinisztikus, teljes véges automatájában az elfogadó és nem elfogadó állapotokat felcseréljük, azaz ha $M = (Q, \Sigma, \delta, q_0, F)$ determinisztikus, teljes véges automata, melyre $L(M) = L$, akkor az $M' = (Q, \Sigma, \delta, q_0, Q \setminus F)$ véges automatára $L(M') = \overline{L}$ teljesül.*

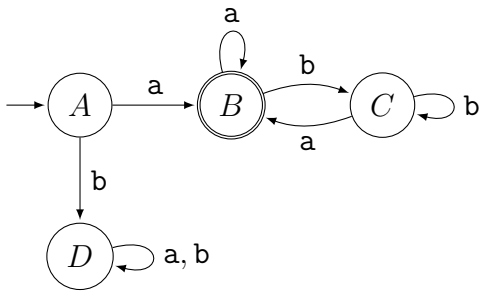
3.7. Feladat *Legyen $\Sigma = \{\,\mathtt{a}, \mathtt{b}\,\}$. Adjunk meg olyan véges automatát, amely az alábbi L nyelvet fogadja el!*

$$L = \{w \in \Sigma^* : w \text{ első és utolsó karaktere megegyezik és } |w| \geq 1\}$$

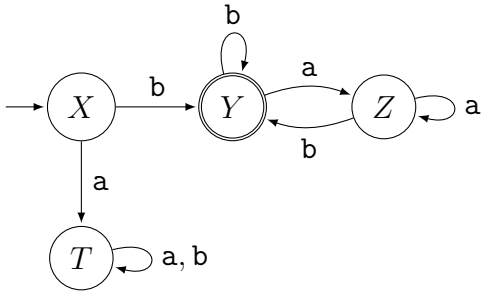
Megoldás: A kívánt nyelv felírható mint $L = L_a \cup L_b$, ahol

$$\begin{aligned} L_a &= \{w \in \Sigma^* : w \text{ első és utolsó karaktere } \mathtt{a} \text{ és } |w| \geq 1\} \\ L_b &= \{w \in \Sigma^* : w \text{ első és utolsó karaktereb } \mathtt{b} \text{ és } |w| \geq 1\} \end{aligned}$$

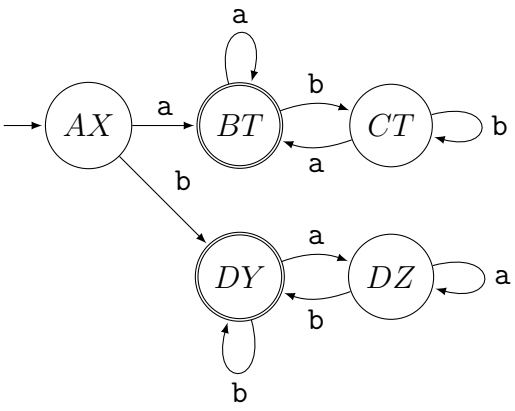
Egy-egy véges automata a két nyelvhez: L_a:



L_b:



A korábbi (determinisztikus) konstrukció alapján előállíthatjuk a két automata unióját, azonban sok olyan állapot lesz, amelyet a kezdőállapotból nem érhetünk el. Érdemes ezért a kiinduló állapotból végigkövetni az elérhető állapotokat és így létrehozni az uniónak megfelelő automatát. A végeredmény tehát:



□

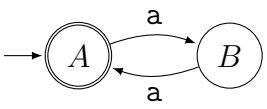
3.8. Feladat *Legyen* $\Sigma = \{\mathtt{a}\}$. *Adjunk meg egy olyan véges automatát, amely az alábbi nyelvek unióját fogadja el!*

L_1 : *páros sok* a*-ból álló szavak.*

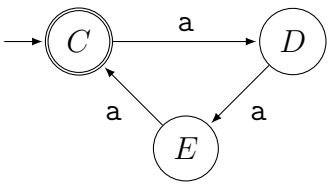
L_2 : *az* a*-k száma hárommal osztható.*

Megoldás: Most a nemdeterminisztikus konstrukció használatát mutatjuk be.

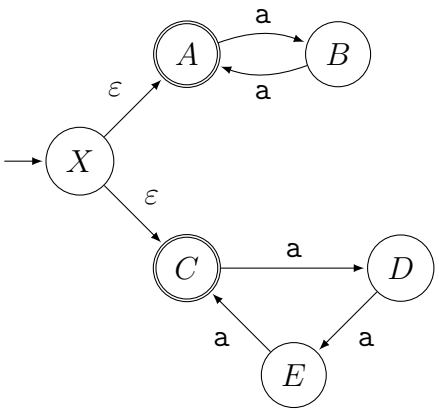
A két nyelvhez könnyű véges automatát csinálni, csak az a betűk számát kell figyelni modulo 2, illetve 3. Egy, az L_1 nyelvhez tartózó M_1 automata:



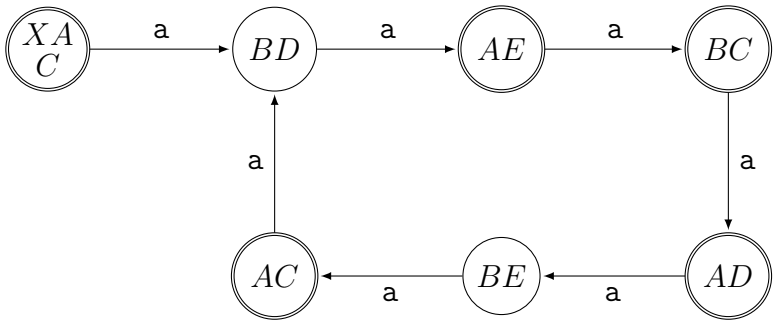
Az L_2 nyelvhez tartózó M_2 automata:



Az uniót elfogadó nemdeterminisztikus automata:



Ezt determinizálhatjuk a tanult eljárás szerint (2.16. tétel):



Vegyük észre, hogy az XAC állapot összevonható az AC-vel és akkor pont azt az automatát kapjuk, amit a determinisztikus konstrukció eredményezett volna. □

A halmazműveletek után nézzük a nyelveken értelmezett két további műveletünket, az összefűzést és a tranzitív lezárást!

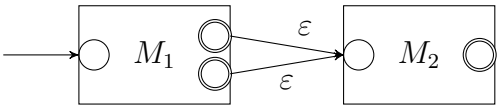
3.9. Tétel *Ha L_1 és L_2 reguláris, akkor $L = L_1L_2$ is reguláris.*

Bizonyítás. Mivel L_1 és L_2 reguláris, léteznie kell olyan M_1 és M_2 véges automatáknak, melyekre $L(M_1) = L_1$ és $L(M_2) = L_2$. Az unióval, metszettel ellentétben, ahol „párhuzamosan" kellett futtatni a két automatát, itt most egymás után kell kötni őket. Egy $w \in L = L_1L_2$ szó eleje L_1-beli, a vége L_2-beli, de nem tudjuk, hol kell elvágni, ezért az átlépés az M_1 automatából M_2-be nemdeterminisztikus lesz. A regularitás bizonyításához ez elegendő, hiszen a 2.16. tétel miatt akkor a nyelvhez determinisztikus véges automata is létezik

A nemdeterminisztikus véges automata konstrukciója:

- Az M automata kezdőállapota azonos M_1 kezdőállapotával.
- Az M_1 elfogadó állapotaiból induljon egy-egy ε átmenet M_2 kezdőállapotába.

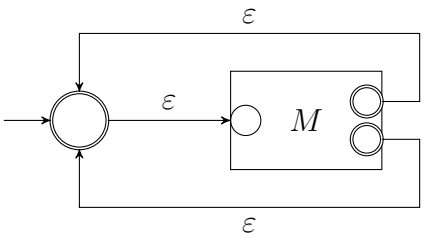
- M elfogadó állapotai legyenek M_2 elfogadó állapotai.



Világos, hogy az új automata éppen a konkatenált nyelv szavait fogja elfogadni. □

3.10. Tétel *Ha L reguláris, akkor L^* is reguláris.*

Bizonyítás. Az M véges automatából, melyre $L(M) = L$ megint egy nemdeterminisztikus véges automata konstrukcióját adjuk meg az L nyelv tranzitív lezártjához, amiből a véges automata létezése is következik.



Ebben az automatában az elfogadó állapotokból bármikor vissza lehet térni a kezdőállapotba, azaz újra lehet kezdeni a nyelv egy szavának felismerését. Ez (és a kezdőállapot elfogadó mivolta) biztosítja, hogy az automata éppen a tranzitív lezárt nyelvet fogadja el. □

Az L nyelvvel való megkülönböztethetetlenség (2.21. definíció) által a szavakon definiált *ekvivalencia osztályok*, mint már láttuk (2.25. példa), egy lehetséges eszközt szolgáltatnak arra, hogy az L nyelvről megmutassuk, hogy nem reguláris. Az alábbi eredmény egy másik, jól használható eszközt biztosít ugyanerre.

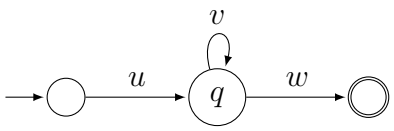
3.11. Lemma (Pumpálási lemma) *Tetszőleges L reguláris nyelvhez létezik olyan $p > 0$ egész szám, hogy minden legalább p hosszú $x \in L$ szónak van olyan $x = uvw$ felosztása, melyre*

- $|uv| \leq p$
- $|v| \geq 1$
- $uv^k w \in L$ *minden* $k \geq 0$ *esetén.*

Ez tehát azt jelenti, hogy a nyelvbeli elég hosszú szavak középső része „pumpálható”. Fontos azonban megjegyezni, hogy akár az is lehetséges, hogy az illető reguláris nyelv egyáltalán nem is tartalmaz legalább p hosszú szavakat, tehát nem minden reguláris nyelvben vannak „pumpálható” szavak.

A lemmában szereplő, csak az L nyelvtől függő p számot az L *pumpálási hosszának* is hívjuk.

Bizonyítás. Mivel L reguláris, biztosan létezik hozzá véges automata. Legyen p egy ilyen (teljes determinisztikus) véges automata állapotainak száma. Tekintsünk egy $|x| = n \geq p$ szóhoz tartozó $r_0, r_1, \ldots, r_n$ számítást. Mivel az automatának csak p különböző állapota van, az $r_0, r_1, \ldots, r_p$ sorozatban biztos van ismétlődés. Legyen q az első olyan állapot, ami megismétlődik, q első előfordulása legyen r_i, második előfordulása legyen r_j. Tekintsük azt az $x = uvw$ felosztást, ahol $|u| = i$, $|v| = j - i$, azaz az u szó végén érjük el először a q állapotot, másodszor meg akkor, amikor a v rész végére érünk.



A $\overline{\delta}$ függvény jelölést használva $\overline{\delta}(q, v) = q$, és így $\overline{\delta}(q, v^k) = q$ is teljesül minden $k \geq 0$ esetén. Azaz $\overline{\delta}(r_0, uv^k) = \overline{\delta}(r_0, uv) = q$, és így minden $k \geq 0$ esetén a kezdőállapotból indított számítás az uv^kw szón ugyanabban az állapotban ér véget. Tehát, ha $x = uvk \in L$, akkor $uv^kw \in L$ szintén igaz. A felosztás megválasztása miatt $|uv| = j \leq p$ és $|v| = j - i \geq 1$ is teljesül, azaz az állítást bebizonyítottuk. □

3.12. Megjegyzés *Fontos megjegyezni, hogy a felbontás középső, v részéről megköveteljük, hogy ne legyen üres (különben az állítás triviális lenne), de az u és w rész lehet üres.*

3.13. Feladat *Mutassuk meg, hogy az $L = \{\mathtt{0}^i\mathtt{1}^i, i \geq 0\}$ nyelv nem reguláris! (A nyelv tehát azokat a szavakat tartalmazza, amelyekben valahány darab* `0` *karaktert ugyanannyi* `1` *követ.)*

Megoldás: Indirekt módon bizonyítunk. Ha L reguláris, akkor igaz rá a pumpálási lemma, legyen p a pumpálási hossz. Tekintsük az $x = 0^p1^p$ szót. Ez nyilván benne van L-ben és $|x| = 2p$, tehát $|x| \geq p$, azaz x olyan szó, amiről a pumpálási lemma szól. Ennek a szónak azonban minden olyan uvw felbontásában, ahol $|uv| \leq p$ és $|v| \geq 1$, a v szó csak nullákból áll. Ekkor a v szót pumpálva biztosan kikerülünk a nyelvből, vagyis ebben a szóban nincs a lemmának megfelelő felosztás, tehát a nyelv nem lehet reguláris. □

3.14. Feladat *Álljon az $L \subseteq \{\,\mathtt{0},\mathtt{1}\,\}^*$ nyelv azokból az y szavakból, melyekben a* `0` *és* `1` *karakterek száma megegyezik. Mutassuk meg, hogy L nem reguláris!*

Megoldás: Az előző bizonyítás most is működik. Ha p a pumpálási hossz, megint választhatjuk a $\mathtt{0}^p\mathtt{1}^p$ szót. Az előbbi érvelés mutatja, hogy ennek akármilyen, a lemma által megengedett felosztását is vesszük, a pumpálás kivezet az L nyelvből.
Második megoldás: Legyen a 3.13. feladatban bemutatott nyelv L_1, valamint legyen

$$L_2 = \{\mathtt{0}^i\mathtt{1}^j, i \geq 0, j \geq 0\},$$

vagyis az a nyelv, amelynek szavaiban a nullák megelőzik az egyeseket (de számuk tetszőleges lehet). Könnyű látni, hogy L_2 reguláris (csináljunk hozzá véges automatát!) és hogy

$$L \cap L_2 = L_1$$

Ha L reguláris lenne, akkor a metszetre való zártság miatt L_1 is reguláris lenne. Az előző feladat szerint L_1 nem reguláris, tehát L sem az. □

3.15. Feladat *Mutassuk meg, hogy az* $\{\mathtt{a}, \mathtt{b}\}$ *ábécé feletti palindromokból álló nyelv nem reguláris. (Palindrom egy olyan* $w = c_1c_2 \ldots c_n$ *szó, ahol* $c_i \in \{\mathtt{a}, \mathtt{b}\}$*, és* $c_n \ldots c_2c_1 = c_1c_2 \ldots c_n$*.)*

Megoldás: Indirekt úton bizonyítunk most is. Tegyük fel, hogy L reguláris, és p a pumpálási hossz. Válasszuk az alábbi $x \in L$ szót:

$$x = \underbrace{\mathtt{a}\ldots\mathtt{a}}_{p}\mathtt{b}\underbrace{\mathtt{a}\ldots\mathtt{a}}_{p} \in L$$

Erre $|x| = 2p+1$, tehát az $|x| \geq p$ teljesül. Ennek a szónak bármilyen uvw felbontásában, ahol az uv hossza p-nél nem nagyobb és v nem üres, a v szó csakis az első, $\mathtt{a}$ betűkből álló blokkba eshet. Akármilyen $k \neq 1$ számra, a v szót k-szor ismételve a két $\mathtt{a}$ betűkből álló blokk hossza különböző lesz, így biztosan kikerülünk a nyelvből. Mivel ennek az $x \in L$ szónak nincs olyan felosztása, amilyet reguláris nyelvek esetén a pumpálási lemma garantál, ezért a nyelv nem reguláris. □

A pumpálási lemma a pumpálhatóságot szükséges feltételként mondja ki a regularitásra. A következő példa mutatja, hogy a pumpálhatóság nem elégséges feltétel, vannak olyan nyelvek, amik pumpálhatóak, de mégsem regulárisak.

3.16. Feladat *Legyen* $p > 0$ *tetszőleges egész szám. Mutassuk meg, hogy az alábbi nyelvnek minden legalább* p *hosszú* x *szavához van a pumpálási lemmában leírt olyan* $x = uvw$ *felbontás, melyre minden* $k \geq 0$ *esetén* $uv^kw \in L$*, de a nyelv nem reguláris!*

$$L = \{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^j : i \geq 1, j \geq 0\} \cup \{\mathtt{b}^j\mathtt{c}^k : j, k \geq 0\}$$

Megoldás: Legyen L_1 az első nyelv, L_2 a második, tehát

$$\begin{aligned} L_1 &= \{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^j : i \geq 1, j \geq 0\} \\ L_2 &= \{\mathtt{b}^j\mathtt{c}^k : j, k \geq 0\} \end{aligned}$$

Legyen $x \in L$, legalább p hosszú és vegyük azt a felosztást, melyben $|u| = 0$ és $|v| = 1$. Megmutatjuk, hogy ez mindig jó, azaz $uv^kw \in L$ minden $k \geq 0$ egészre teljesül. Nézzük, milyen esetek lehetségesek!

Ha $x \in L_1$, akkor $v = \mathtt{a}$. Ha a pumpálással kapott uv^kw szó továbbra is $\mathtt{a}$ betűvel kezdődik az L_1 nyelv eleme marad (a $\mathtt{b}$ és $\mathtt{c}$ betűk száma nem változott). Viszont az is előfordulhat, hogy az uv^kw szóban már nincs $\mathtt{a}$ betű (ha $i = 1$ volt és $k = 0$), de ilyenkor, világos módon, L_2-beli szót kapunk.

Amennyiben $x \in L_2$, akkor a pumpálás az első betűt vagy törli vagy sokszorosítja, de mindkét esetben a kapott szó is az L_2 nyelvben lesz. Tehát ez az L nyelv valóban pumpálható.

Másrészt a $\mathtt{ab}^i$ és $\mathtt{ab}^j$ szavak minden esetben megkülönböztethetőek a nyelvvel (ha $i \neq j$), ezért végtelen sok olyan szó van, amely az L nyelvvel páronként megkülönböztethető, tehát L nem reguláris. □

3.2. Reguláris kifejezések

A reguláris nyelveknek egy sok helyen előforduló megadási módja a reguláris kifejezés. Ezeket, mint mindjárt látható, rekurzív definícióval adhatjuk meg. A definíció ismeretében megmutatjuk, hogy a reguláris kifejezések pontosan a reguláris nyelvek megadására alkalmasak. A reguláris kifejezésekre is többféle jelölési rendszer van, itt ezekből egyet fogunk használni. Hasonlókkal találkozhatunk szövegszerkesztőkben, programozási nyelvekben és általában keresési kifejezések megadásánál is.

3.17. Definíció (Reguláris kifejezés) *Egy Σ ábécé felett reguláris kifejezések a következők:*

- $\emptyset$
- ε
- $\mathtt{a}$ *reguláris kifejezés, ha* $\mathtt{a} \in \Sigma$

Továbbá, ha r_1 és r_2 reguláris kifejezés, akkor

- $r_1 + r_2$
- $r_1 r_2$

- r_1^*

is reguláris kifejezés.

A reguláris kifejezésben szereplő jelöléseket, műveleteket természetes módon meg lehet feleltetni nyelveknek, és ezeken végzett műveleteknek.

3.18. Definíció *A Σ ábécé feletti r reguláris kifejezés által leírt $L(r)$ nyelv legyen*

- $L(\emptyset) = \emptyset$,
- $L(\varepsilon) = \{\varepsilon\}$.
- $L(\mathtt{a}) = \{\mathtt{a}\}$, *ha* $\mathtt{a} \in \Sigma$.

Továbbá, ha r_1 és r_2 reguláris kifejezés, akkor

- $L(r_1 + r_2) = L(r_1) \cup L(r_2)$,
- $L(r_1 r_2) = L(r_1)L(r_2)$,
- $L(r_1^*) = L(r_1)^*$

Egy összetettebb reguláris kifejezés megadásánál zárójeleket használunk annak feltüntetésére, hogyan keletkezett a kifejezés, pl. $(\mathtt{0}+\mathtt{1})^*(\mathtt{00})$. Általános szabály, hogy ha nincs zárójel, akkor a * műveletet kell először elvégezni, utána az összefűzést és végül a $+$ műveletet. Ha azonos műveletek vannak egymás után, akkor balról jobbra hajtjuk őket végre. Pl. az $\mathtt{a} + \mathtt{bc}^*\mathtt{a} + \mathtt{b}$ kifejezés zárójelekkel ellátva így néz ki: $(\mathtt{a} + ((\mathtt{b}(\mathtt{c})^*)\mathtt{a})) + \mathtt{b}$, azaz a definíció szerinti előállítása $r_1 = \mathtt{c}^*$, $r_2 = \mathtt{b}r_1$, $r_3 = r_2\mathtt{a}$, $r_4 = \mathtt{a} + r_3$, $r_5 = r_4 + \mathtt{b}$.

3.19. Példa *Reguláris kifejezések és az általuk leírt nyelvek:*

- *Ha* $r = (a + b)^*$, *akkor* $L(r) = \{\, \mathtt{a}, \mathtt{b} \,\}^*$.
- *Ha* $r = \mathtt{0}^*\mathtt{1}\mathtt{0}^*$, *akkor* $L(r) \subset \{\, \mathtt{0}, \mathtt{1} \,\}^*$ *a pontosan 1 darab* `1` *karaktert tartalmazó szavakból álló nyelv.*
- *Ha* $r = (\mathtt{0}+\mathtt{1})^*\mathtt{1}(\mathtt{0}+\mathtt{1})^*$, *akkor* $L(r)$ *a legalább egy darab egyest tartalmazó szavakból álló nyelv.*
- *Ha* $r = \varepsilon + 0(0+1)^*0 + 1(0+1)^*1 + 0 + 1$, *akkor* $L(r) \subset \{\, \mathtt{0}, \mathtt{1} \,\}^*$ *azokból a szavakból áll, amelyekben az első karakter azonos az utolsóval.*

3.20. Állítás *Ha r egy Σ ábécé feletti reguláris kifejezés, akkor $L(r) \subseteq \Sigma^*$ reguláris nyelv.*

Bizonyítás. Az állítás következik abból, hogy az $\emptyset$, $\{\varepsilon\}$ és $\{a\}$ nyelvek regulárisak, valamint, hogy reguláris nyelvek uniója, összefűzése, tranzitív lezártja is reguláris. □

Mielőtt megmutatjuk, hogy az előző állítás megfordítása is igaz, felsoroljuk a reguláris kifejezések néhány „műveleti tulajdonságát". Ezen tulajdonságok a kifejezések két oldalának megfelelő nyelvek azonosságán alapulnak.

Tulajdonság:

- $(r_1 + r_2) + r_3 = r_1 + (r_2 + r_3)$ (az összeadás asszociatív)
- $r_1 + r_2 = r_2 + r_1$ (az összeadás kommutatív)
- $r + \emptyset = r$ (az összeadásra az $\emptyset$ egységelem)
- $r + \varepsilon = r \Leftrightarrow \varepsilon \in L(r)$
- $(r_1 r_2) r_3 = r_1 (r_2 r_3)$ (az összefűzés asszociatív művelet)
- $r\varepsilon = \varepsilon r = r$ (az összefűzésre ε egységelem)
- $r\emptyset = \emptyset r = \emptyset$
- $\emptyset^* = \varepsilon$
- $(a + \varepsilon)^* = a^*$
- $(a + \varepsilon)(a + \varepsilon)^* = a^*$

Most megmutatjuk, hogy a 3.20. állítás megfordítása is igaz, azaz a reguláris nyelvekhez szerkeszthető reguláris kifejezés.

3.21. Tétel *Minden L reguláris nyelvhez van olyan r reguláris kifejezés, hogy $L = L(r)$.*

Bizonyítás. Az L-hez tartozó reguláris kifejezést több lépcsőben adjuk meg. Legyen $M = (Q, \Sigma, \delta, q_0, F)$ egy determinisztikus véges automata, melyre $L(M) = L$. A jelölések egyszerűbbé tételéhez tegyük fel, hogy $Q = \{1, 2, \ldots, n\}$ és $q_0 = 1$

Minden $p, q \in Q$ állapotpárra definiáljuk az

$$L(p, q) = \{x \in \Sigma^* : \overline{\delta}(p, x) = q\}$$

nyelvet, mint azon szavak halmazát, amikre az automata p állapotból q állapotba jut. Ekkor világos, hogy

$$L = \bigcup_{q \in F} L(1, q)$$

Elegendő tehát az $L(p,q)$ típusú nyelvekhez egy-egy $r(p,q)$ reguláris kifejezést találni. mert ezekből egy L-et leíró reguláris r kifejezés már

$$r = \sum_{q \in F} r(1,q)$$

alakban előállítható. Az $r(p,q)$ kifejezések meghatározása a dinamikus programozás módszerével történhet. Ehhez vezessük be az $L(p,q,t)$ nyelveket, ahol

$$\begin{aligned} L(p,q,t) &= \{x \in \Sigma^* : \overline{\delta}(p,x) = q, \text{ és a lépések során a közbenső állapotok az} \\ &\quad \{1,\ldots,t\} \text{ halmazból kerülnek ki.}\} \end{aligned}$$

Mivel összesen n állapot van, ha $t = n$, akkor mindegyik állapot használata megengedett, tehát $L(p,q) = L(p,q,n)$. Így tehát ha meghatározzuk az $L(p,q,n)$ nyelvekhez tartozó reguláris kifejezéseket, akkor készen vagyunk.

Vegyük most szemügyre az $L(p,q,0)$ nyelvet. Ekkor semmilyen közbenső állapot sem lehet, azaz, ha van M-ben p-ből q-ba él, akkor a p és q közötti éleken található betűk lesznek az $L(p,q,0)$ nyelv elemei, valamint ha $p = q$, akkor ε is. Abban az esetben, ha nincs él és $p \neq q$, akkor $L(p,q,0) = \emptyset$. Mindegyik esetben egy, az $L(p,q,0)$ nyelvet leíró reguláris kifejezés könnyen felírató.

Tegyük most fel, hogy az $L(p,q,t-1)$ nyelvekre már van reguláris kifejezésünk, jelölje ezeket $r(p,q,t-1)$. Most tekintsük az $L(p,q,t)$ nyelvet. Az $L(p,q,t-1)$ nyelvhez képest az a különbség hogy az automatában a lépések során a t állapotot is érinthetjük, akár többször is. A t állapot két szomszédos meglátogatása során viszont most is csak az $\{1,\ldots,t-1\}$ állapotokat használhatjuk. Tehát az egész számítást felbonthatjuk szakaszokra: $\{1,\ldots,t-1\}$-en keresztül eljutunk t-be, azután néhányszor (akár nullaszor) $\{1,\ldots,t-1\}$-en keresztül visszajövünk t-be, majd t-ből eljutunk a q állapotba. Az is lehet természetesen, hogy a t állapotot egyáltalán nem érintjük – ezeket az eseteket $r(p,q,t-1)$ már leírja.

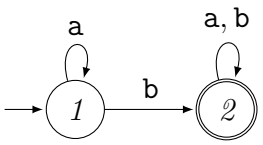
Mindezt összerakva kapjuk, hogy

$$r(p,q,t) = r(p,q,t-1) + r(p,t,t-1)r(t,t,t-1)^* r(t,q,t-1)$$

Ezzel az eljárással az $r(p,q,0)$ reguláris kifejezésekből kiindulva előállíthatjuk az $r(p,q) = r(p,q,n)$ kifejezéseket, amelyek közül a megfelelőeket összeadva megkapunk egy, az L nyelvet leíró reguláris kifejezést. □

3.22. Megjegyzés *Az eljárás (kis módosítással) abban az esetben is működik, ha az M automata nem determinisztikus.*

3.23. Feladat *Adjunk meg egy reguláris kifejezést az alábbi automata által elfogadott nyelvhez!*



Megoldás: Nézzük először az $r(p, q, 0)$ típusú kifejezéseket!

	$\boldsymbol{r(p,1,0)}$	$\boldsymbol{r(p,2,0)}$
$\boldsymbol{p=1}$	$\mathtt{a}+\varepsilon$	$\mathtt{b}$
$\boldsymbol{p=2}$	$\emptyset$	$\mathtt{a}+\mathtt{b}+\varepsilon$

Használva a bizonyításban leírt képletet, valamint a reguláris kifejezésekre alkalmazható egyszerűsítéseket, kapjuk, hogy

$$\begin{aligned} r(1,1,1) &= r(1,1,0) + r(1,1,0)r(1,1,0)^* r(1,1,0) = \\ &\quad (\mathtt{a}+\varepsilon) + (\mathtt{a}+\varepsilon)(\mathtt{a}+\varepsilon)^*(\mathtt{a}+\varepsilon) = \mathtt{a}^* \\ r(1,2,1) &= r(1,2,0) + r(1,1,0)r(1,1,0)^* r(1,2,0) = \\ &\quad \mathtt{b} + (\mathtt{a}+\varepsilon)(\mathtt{a}+\varepsilon)^*\mathtt{b} = \mathtt{b} + \mathtt{a}^*\mathtt{b} = a^*b \end{aligned}$$

Így folytatva, $t = 1$-re a következő reguláris kifejezések adódnak

	$\boldsymbol{r(p,1,1)}$	$\boldsymbol{r(p,2,1)}$
$\boldsymbol{p=1}$	$\mathtt{a}^*$	$\mathtt{a}^*\mathtt{b}$
$\boldsymbol{p=2}$	$\emptyset$	$\mathtt{a}+\mathtt{b}+\varepsilon$

Ebből $t = 2$-re a reguláris kifejezések:

	$\boldsymbol{r(p,1,2)}$	$\boldsymbol{r(p,2,2)}$
$\boldsymbol{p=1}$	$\mathtt{a}^*$	$\mathtt{a}^*\mathtt{b}(\mathtt{a}+\mathtt{b})^*$
$\boldsymbol{p=2}$	$\emptyset$	$(\mathtt{a}+\mathtt{b})^*$

Az L-hez tartozó reguláris kifejezés tehát $r(1, 2, 2)$, azaz: $\mathtt{a}^*\mathtt{b}(\mathtt{a}+\mathtt{b})^*$. □ Természetesen megtehettük volna, hogy közben nem egyszerűsítjük a kapott reguláris kifejezéseket, akkor egy ezzel ekvivalens, de lényegesen hosszabb, bonyolultabb kifejezést kaptunk volna. A megkapott egyszerű alakból az automata által elfogadott nyelv is könnyen kiolvasható: kell legyen a szóban $\mathtt{b}$ betű.

4. fejezet

Nyelvtanok, reguláris nyelvtanok

4.1. Nyelvtanok

Az itt tárgyalt *formális* nyelvtanok nem egészen olyanok, mint például a magyar nyelvtan. Inkább használhatóak mesterséges nyelvek, például programozási nyelvek pontos megadására, mint egy beszélt nyelv helyes mondatainak tökéletes leírására.

Jelentőségük abban áll, hogy véges eszközt nyújtanak egy (esetleg) végtelen szóhalmaz leírására.

4.1. Definíció Nyelvtan *(vagy* formális nyelvtan*) alatt egy olyan* $G = (V, \Sigma, S, P)$ *rendszert értünk, ahol*

- V *egy véges nem üres halmaz, a változók halmaza,*
- Σ *egy ábécé,* $V \cap \Sigma = \emptyset$*,*
- $S \in V$ *a kezdő változó,*
- P *egy véges halmaz, az ún. levezetési vagy produkciós szabályok halmaza.* P *elemei* $\alpha \to \beta$ *alakúak,* α *és* β *tetszőleges,* V *és* Σ *elemeiből képzett sorozat, az egyetlen megkötés, hogy* α *tartalmazzon legalább egy változót is (*β *lehet akár az üres szó is).*

A levezetési szabályokra adott feltétel tömören így is írható: $\alpha \in (V \cup \Sigma)^* V (V \cup \Sigma)^*$ és $\beta \in (V \cup \Sigma)^*$.

4.2. Megjegyzés V *elemeit szokás nemterminálisoknak,* a Σ *elemeit terminálisoknak,* P *elemeit pedig átírási szabályoknak is hívni. Az elnevezést az indokolja, hogy a* Σ *karakterei már nem változnak a levezetés során, míg* V *elemei biztosan átalakulnak még.*

4.3. Definíció *Egy G nyelvtanbeli* levezetés *alatt olyan*

$$\gamma_0 \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \cdots \Rightarrow \gamma_n$$

véges hosszú sorozatot értünk ($n \geq 0$), melyben $\gamma_0 = S$, továbbá $\gamma_i \in (V \cup \Sigma)^*$, és mindegyik γ_{i+1} megkapható γ_i-ből egy levezetési szabály alkalmazásával. Ez azt jelenti, hogy minden $0 \leq i < n$ esetén van olyan $\delta_1, \delta_2, \alpha, \beta \in (V \cup \Sigma)^*$, hogy γ_i és γ_{i+1} felbontható $\gamma_i = \delta_1\alpha\delta_2$, illetve $\gamma_{i+1} = \delta_1\beta\delta_2$ alakra és $(\alpha \to \beta) \in P$.

Egy levezetés során tehát amennyiben γ_i-ben több helyen is szerepel valamilyen szabály bal oldala, akkor akármelyik bal oldal akármelyik lehetséges helyettesítésével folytathatjuk a levezetést.

4.4. Definíció *Az olyan levezetést, amiben minden lépésben a lehetséges helyettesítések közül mindig egy, a γ_i elejéhez legközelebbi helyettesítést végzünk el* bal-levezetésnek *hívjuk.*

4.5. Példa *Tekintsük a $G = (\{S\}, \{\,\mathtt{a}, \mathtt{b}\,\}, S, P)$ nyelvtant, ahol*

$$P = \{S \to \mathtt{a}S, \quad S \to \mathtt{b}S, \quad S \to \varepsilon\}$$

Ekkor $S \Rightarrow \mathtt{a}S \Rightarrow \mathtt{ab}S$ vagy $S \Rightarrow \mathtt{a}S \Rightarrow \mathtt{ab}S \Rightarrow \mathtt{ab}$ is egy levezetés. Itt minden levezetés egyben bal-levezetés is, mivel a nyelvtan szabályai olyanok, hogy egyszerre csak egy változó szerepel a szavakban.

A nyelvtanok megadásánál sokszor nem írjuk ki az összes paramétert, csak a levezetési szabályokat soroljuk fel. Ha mást nem mondunk, akkor a szabályokban szereplő kisbetűk Σ elemeit jelölik, a nagybetűk a változókat, és az első szabály bal oldala a kezdőváltozó. Ebben a formában az előző nyelvtan így néz ki:

$$S \to \mathtt{a}S, \quad S \to \mathtt{b}S, \quad S \to \varepsilon$$

Tovább tömörítve, azok a szabályok, melyeknek a bal oldalán ugyanaz áll, összevonhatóak, függőleges vonallal elválasztva a különböző jobb oldalakat,

$$S \to \mathtt{a}S \mid \mathtt{b}S \mid \varepsilon$$

A levezetések közül azok lesznek érdekesek, amikor a kezdőváltozóból indulva végül egy olyan sorozathoz jutunk, amiben már nincsenek változók.

4.6. Definíció *A $G = (V, \Sigma, S, P)$ nyelvtan által* generált $L(G)$ nyelv *azokból a $w \in \Sigma^*$ szavakból áll, melyekhez van egy valahány lépésből álló $S \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \cdots \Rightarrow w$ levezetés.*

Vegyük észre, hogy ha egy levezetés során eljutunk egy $w \in \Sigma^*$ szóhoz, akkor a levezetés tovább már biztos nem folytatható, hiszen w nem tartalmaz változót, egyetlen szabály sem alkalmazható.

4.7. Feladat *Határozzuk meg az előző, 4.5. példában szereplő nyelvtan által generált nyelvet!*

Megoldás: Tetszőleges $x \in \{\mathtt{a}, \mathtt{b}\}^*$ szó levezethető, x következő betűjétől függően kell az $S \to \mathtt{a}S$ vagy az $S \to \mathtt{b}S$ szabályt használni a levezetés során, és amikor már megkaptuk a megfelelő betűket, akkor az $S \to \varepsilon$ szabály segítségével megszabadulunk az S változótól, azaz pl. $x = \mathtt{aab}$ esetén $S \Rightarrow \mathtt{a}S \Rightarrow \mathtt{aa}S \Rightarrow \mathtt{aab}S \Rightarrow \mathtt{aab}$ egy levezetés, amiben rendre az 1., 1., 2., 3. szabályt alkalmaztuk.

Tehát $L(G) = \{a, b\}^*$ □

4.8. Feladat *Álljon a nyelvtan az alábbi szabályokból:*

$$S \to aSb \mid \varepsilon$$

Határozzuk meg az általa generált nyelvet!

Megoldás: A korábban leírtak szerint a nyelvtannak egyetlen változója van (S), ami egyben a nyelvtan kezdőváltozója is, továbbá van két levezetési szabály. Ha ezek közül a másodikat ($S \Rightarrow \varepsilon$) alkalmazzuk, a levezetés véget ér. Ha előtte n lépés volt a levezetés során, akkor végül az $\mathtt{a}^n\mathtt{b}^n$ szót kaptuk meg. Tehát $L(G) = \{\mathtt{a}^n\mathtt{b}^n : n \geq 0\}$. (Az üres szó is eleme a nyelvnek, ezt az $S \Rightarrow \varepsilon$ levezetés adja, ekkor $n = 0$.) □

4.9. Példa *Tekintsük az alábbi szabályokkal megadott nyelvtant!*

$$S \to \mathtt{a}SBC \mid \mathtt{ab}C \tag{1-2}$$
$$CB \to BC \tag{3}$$
$$\mathtt{b}B \to \mathtt{bb} \tag{4}$$
$$\mathtt{b}C \to \mathtt{bc} \tag{5}$$
$$\mathtt{c}C \to \mathtt{cc} \tag{6}$$

Ekkor S a kezdőváltozó, $V = \{B, C, S\}$ és $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. Figyeljük meg a levezetési szabályok alkalmazásának az alábbi sorrendjét (az aláhúzott rész jelöli a következőként alkalmazott szabály bal oldalát, a nyíl feletti szám a szabály sorszámát):

$$S \overset{1}{\Rightarrow} \mathtt{a}\underline{S}BC \overset{1}{\Rightarrow} \mathtt{aa}\underline{S}BCBC \overset{2}{\Rightarrow} \mathtt{aaa}\underline{bC}BCBC \overset{5}{\Rightarrow} \mathtt{aaaabc}B\underline{CB}C \overset{3}{\Rightarrow} \mathtt{aaabc}BBCC$$

Látható, hogy „elakadtunk”, már egy szabály sem alkalmazható, tehát ezen a módon nem kapjuk meg $L(G)$ egy elemét. Viszont ha másként választjuk meg az egyes lépéseket, akkor egy Σ feletti szóhoz juthatunk.

Könnyen látható, hogy minden $\mathtt{a}^i\mathtt{b}^i\mathtt{c}^i$ típusú szó ($i \geq 1$) eleme az $L(G)$ nyelvnek, mert

$$
\begin{aligned}
S \quad & \underbrace{\overset{1}{\Rightarrow} \gamma_1 \overset{1}{\Rightarrow} \ldots \overset{1}{\Rightarrow} \gamma_n}_{n} \overset{2}{\Rightarrow} \mathtt{a}^{n+1}\mathtt{b}\underbrace{CBCB\ldots CB}_{n}C \overset{3}{\Rightarrow} \\
& \gamma_{n+2} \overset{3}{\Rightarrow} \gamma_{n+3} \overset{3}{\Rightarrow} \ldots \overset{3}{\Rightarrow} \mathtt{a}^{n+1}\mathtt{b}B^nC^{n+1} \overset{4}{\Rightarrow} \\
& \gamma_m \overset{4}{\Rightarrow} \gamma_{m+1} \overset{4}{\Rightarrow} \ldots \overset{4}{\Rightarrow} \mathtt{a}^{n+1}\mathtt{b}^{n+1}C^{n+1} \overset{5}{\Rightarrow} \\
& \mathtt{a}^{n+1}\mathtt{b}^{n+1}\mathtt{c}C^n \overset{6}{\Rightarrow} \gamma_k \overset{6}{\Rightarrow} \gamma_{k+1} \overset{6}{\Rightarrow} \ldots \overset{6}{\Rightarrow} \mathtt{a}^{n+1}\mathtt{b}^{n+1}\mathtt{c}^{n+1}
\end{aligned}
$$

Megmutatható, hogy csak az ilyen típusú szavak vezethetők le a nyelvtanból, azaz $L(G) = \{\mathtt{a}^i\mathtt{b}^i\mathtt{c}^i : i \geq 1\}$.

4.10. Feladat *Adjunk meg egy nyelvtant az $\{\mathtt{a}, \mathtt{b}\}$ ábécé feletti palindromokhoz!*

Megoldás: Egy lehetséges megoldás: $S \to \mathtt{a}S\mathtt{a} \mid \mathtt{b}S\mathtt{b} \mid \mathtt{a} \mid \mathtt{b} \mid \varepsilon$

Triviálisan igaz, hogy az így generált nyelvben nincs olyan szó, amely nem palindrom. Azt, hogy minden palindrom levezethető, a szó hossza szerinti indukcióval bizonyítjuk. A 0 és 1 hosszú palindromok egyetlen lépésben levezethetők. Tegyük fel, hogy már tudjuk, minden k-nál rövidebb palindrom levezethető a nyelvtanból. Mivel minden, legalább 2 hosszú palindrom olyan, hogy az első és utolsó karaktere azonos, a kettő között pedig egy rövidebb w' palindrom áll, a levezetés első lépéseként az 1. vagy 2. szabállyal előállíthatjuk az első és utolsó betűt, a keletkezett S változóból pedig az indukció miatt w' levezethető. Így tehát a generált nyelv minden palindromot tartalmaz. □

4.11. Feladat *Legyen $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ és L álljon azokból a szavakból, melyekben az $\mathtt{a}$ betűk száma megegyezik a $\mathtt{b}$ betűk számával. Adjunk olyan G nyelvtant, amire $L(G) = L$.*

Megoldás: Egy lehetséges megoldás: $S \to \mathtt{a}S\mathtt{b}S \mid \mathtt{b}S\mathtt{a}S \mid \varepsilon$

Ahhoz, hogy ez valóban jó nyelvtan, először is vegyük észre, hogy minden esetben, amikor valamelyik szabályt alkalmazzuk, ugyanannyi $\mathtt{a}$-t generálunk, mint ahány $\mathtt{b}$-t, és ezek a betűk meg is maradnak a továbbiakban, ezért $L(G) \subseteq L$.

Azt kell még megmutatni, hogy minden $w \in L$ szó levezethető. Ezt a w hossza szerinti indukcióval látjuk be. Nyilván ez igaz a 0 hosszú $w = \varepsilon$ szóra. A kettő hosszú szavakra is könnyű látni, mert vagy az első vagy a második szabály egyszeri alkalmazása után a harmadik szabályt kétszer használva megkaphatjuk a w szót.

Tegyük fel, hogy L legfeljebb k hosszú szavairól már tudjuk, hogy levezethetők és legyen $|w| = k + 1$. Több eset lehetséges: amennyiben $w = \mathtt{a}w'\mathtt{b}$, akkor $w' \in L$ és ebben

az esetben az $S \Rightarrow \mathtt{a}S\mathtt{b}S$ kezdés után S első előfordulásából, az indukciós feltevés szerint w' levezethető, miután a második S betűre az $S \to \varepsilon$ szabályt alkalmazva megkapjuk a w szót.

Hasonlóan járhatunk el, amennyiben $w = \mathtt{b}w'\mathtt{a}$.

Ha viszont w első és utolsó betűje megegyezik, akkor w felírható $\mathtt{a}w'\mathtt{b}w''$ vagy $\mathtt{b}w'\mathtt{a}w''$ alakban, ahol $w', w'' \in L$. (Például ha w az $\mathtt{a}$ betűvel kezdődik, akkor $\mathtt{a}w'\mathtt{b}$ lehet az első nem üres kezdőszelet, amiben az $\mathtt{a}$ és $\mathtt{b}$ betűk száma megegyezik.) A két eset hasonlóan kezelhető, pl. az elsőnél az $S \Rightarrow \mathtt{a}S\mathtt{b}S$ lépéssel indulva, az indukciós feltevés miatt S első előfordulásából levezethetjük a w' szót, a második előfordulásából pedig a w'' szót. □

4.2. Chomsky-féle nyelvosztályok

Noam Chomsky a nyelvtanok 4 típusát definiálta aszerint, hogy milyen megkötéseket teszünk a szabályok alakjára. Látni fogjuk, hogy ezek a megkötések jól vannak kitalálva: az így definiált nyelvosztályok szoros kapcsolatban állnak a nyelvek felismerésére szolgáló különféle automatákkal.

A nyelvtan-típusok 0-tól 3-ig vannak számozva.

4.12. Definíció *A G nyelvtan a* 3. osztályba *tartozik ha szabályai $A \to \mathtt{a}B$ illetve $A \to \mathtt{a}$ alakúak ($A, B \in V$ és $\mathtt{a} \in \Sigma$ tetszőleges).*

A fentieken kívül megengedett még a kezdőváltozóra az $S \to \varepsilon$ szabály, amennyiben S nem fordul elő egyetlen szabály jobb oldalán sem.

4.13. Definíció *A G nyelvtan a* 2. osztályba *tartozik ha szabályai $A \to \alpha$ alakúak, ahol $A \in V$ és $\alpha \in (V \cup \Sigma)^*$ tetszőleges, de legalább 1 hosszú.*

A fentieken kívül megengedett még a kezdőváltozóra az $S \to \varepsilon$ szabály, amennyiben S nem fordul elő egyetlen szabály jobb oldalán sem.

4.14. Definíció *A G nyelvtan az* 1. osztályba *tartozik, ha szabályai $\beta A\gamma \to \beta\alpha\gamma$ alakúak, ahol $A \in V$ és $\alpha, \beta, \gamma \in (V \cup \Sigma)^*$ tetszőleges, de α legalább 1 hosszú.*

A fentieken kívül megengedett még a kezdőváltozóra az $S \to \varepsilon$ szabály, amennyiben S nem fordul elő egyetlen szabály bal oldalán sem.

4.15. Definíció *A* 0. osztály *nyelvtanaira semmilyen megkötés nincs (a szokásos megkötésen kívül, miszerint a bal oldal legalább egy változót tartalmaz).*

A 3. osztályba tartozó nyelvtanokat nevezik *reguláris* nyelvtanoknak is, majd látni fogjuk (4.20. tétel), hogy ez az elnevezés megalapozott.

A 2. osztályba tartozó nyelvtanok másik elnevezése *környezetfüggetlen nyelvtanok*, vagy röviden *CF nyelvtanok* az angol context free elnevezésből. Ez az elnevezés arra utal, hogy az A változót bármely környezetben helyettesíthetjük az α jobb oldallal.

Ezzel szemben az 1. osztály nyelvtanai, ahol az A változót nem lehet akármikor az α karaktersorozattal helyettesíteni, csak ha a megfelelő környezetben van (előtte β, utána γ áll) a *környezetfüggő nyelvtanok*, vagy röviden *CS nyelvtanok* az angol context sensitive elnevezésből.

A 0. osztálynak nincs külön neve.

A definícióból következik, hogy egy i. osztálybeli nyelvtan egyben j. osztálybeli is, ha $j \leq i$.

4.16. Definíció *Az L nyelv az* i-edik osztályba *tartozik, ha van olyan i-edik osztályú G nyelvtan, amire $L(G) = L$.*

Az i-edik osztályba tartozó nyelvek halmazát jelölje $\mathcal{L}_i$.

A definíció következménye, hogy $\mathcal{L}_3 \subseteq \mathcal{L}_2 \subseteq \mathcal{L}_1 \subseteq \mathcal{L}_0$.

Majd látni fogjuk, hogy ezek mindegyike valódi tartalmazás, azaz minden esetben van olyan nyelv, amihez van i-edik osztályba tartozó nyelvtan, de $(i+1)$-edik osztályba tartozó már nincs.

Most azt mutatjuk meg, hogy van olyan nyelv, amire nem tudunk nyelvtant készíteni.

4.17. Tétel *Van olyan L nyelv, amihez nincs generáló nyelvtan.*

Bizonyítás. Az állítás számossági megfontolásból következik. Az ábécé egy véges halmaz, ezért az összes szó Σ^* halmaza megszámlálhatóan végtelen halmaz, a nyelvek pedig ezen megszámlálhatóan végtelen halmaz kontinuum számosságú hatványhalmazát alkotják.

Ezzel szemben minden nyelvtan leírható egy véges jelsorozattal valamilyen fix ábécé felett (mondjuk binárisan elkódolva), ezért a nyelvtanok számossága megszámlálható, így megszámlálható a nyelvtanokkal leírható nyelvek számossága is.

Később mutatni fogunk olyan L nyelvet is, amire $L \notin \mathcal{L}_0$. □

4.18. Feladat *Adjunk 3. osztálybeli nyelvtant az $\{\,\mathtt{a},\mathtt{b}\,\}^*$ nyelvre!*

Megoldás: A 4.5. példabeli nyelvtan nem megfelelő, hiszen az csak a 0. osztályba tartozik bele, de nem nehéz úgy módosítani, hogy már megfeleljen a 3. osztály követelményeinek, csak arról kell gondoskodni, hogy a kezdőváltozó ne szerepeljen szabály jobb oldalán.

$$\begin{aligned} S &\to \varepsilon \mid \mathtt{a}A \mid \mathtt{b}A \mid \mathtt{a} \mid \mathtt{b} \\ A &\to \mathtt{a}A \mid \mathtt{b}A \mid a \mid \mathtt{b} \end{aligned}$$

□

4.3. Reguláris nyelvtanok

Meg fogjuk mutatni, hogy a reguláris nyelvtanok pontosan a reguláris nyelveket, vagyis a véges automatákkal felismerhető nyelveket generálják. Ehhez szükségünk lesz a következő állításra.

4.19. Tétel *Minden $M = (Q, \Sigma, \delta, q_0, F)$ véges automatához készíthető egy olyan $M' = (Q', \Sigma, \delta', q_0', F')$ véges automata, melynek nincs olyan számítása, amiben a q_0' a kezdet után később is előfordul és amelyre $L(M) = L(M')$.*

Bizonyítás. Ha M-nek nincs olyan átmenete, ahol $\delta(p, a) = q_0$, akkor $M' = M$ megfelelő. Ellenkező esetben legyen q_0' egy új állapot, $Q' = Q \cup \{q_0'\}$, $F' = \{q_0'\}$ (q_0' lesz az az állapot, ahova q_0 helyett az átmenetek mutatnak majd). Ehhez az M' állapotátmeneteit minden $p \in Q$ és $a \in \Sigma$ esetén definiáljuk a következőképpen:

$$\begin{aligned} \delta'(p, a) &= \delta(p, a), \text{ ha } \delta(p, a) \neq q_0 \\ \delta'(p, a) &= q_0', \text{ ha } p \neq q_0' \text{ és } \delta(p, a) = q_0 \\ \delta'(q_0', a) &= \delta(q_0, a) \, \forall a \in \Sigma \text{ esetén} \end{aligned}$$

Könnyen látható, hogy az így kapott M' megfelel az állításnak. □

4.20. Tétel *Az L nyelvhez akkor és csak akkor létezik 3. osztálybeli nyelvtan, ha L reguláris.*

Bizonyítás. Először tegyük fel, hogy L reguláris, azaz L felismerhető véges automatával. Megmutatjuk, hogy ekkor generálható reguláris nyelvtannal. Legyen $M = (Q, \Sigma, \delta, q_0, F)$ egy az előző, 4.19. tételnek megfelelő, az L nyelvet elfogadó véges automata. Ebből definiálunk egy megfelelő nyelvtant. A nyelvtan változói feleljenek meg M állapotainak: minden $q \in Q$ állapothoz létrehozunk egy, a többitől különböző A_q változót, azaz $V = \{A_q : q \in Q\}$. Legyen a kezdőváltozó $S = A_{q_0}$. Az automata minden állapotátmenetéből egy vagy két levezetési szabály lesz. A keletkező szabályok

$$\begin{aligned} A_q &\to \mathtt{a}A_p, \text{ ha } \delta(q, \mathtt{a}) = p \\ A_q &\to \mathtt{a}, \text{ ha } \delta(q, \mathtt{a}) = p \in F \\ S &\to \varepsilon, \text{ ha } q_0 \in F \end{aligned}$$

Az így kapott nyelvtan valóban reguláris nyelvtan. Ehhez azt kell csak meggondolni, hogy az $S = A_{q_0}$ kezdőszimbólum nem szerepel nyelvtani szabály jobb oldalán, de ez teljesül, mert az automatánkban egyetlen állapotátmenet sem vezet a q_0 állapotba. Az, hogy $L(M) = L(G)$ az automata számításai és a nyelvtan levezetései közötti szoros kapcsolat következménye. Az $x = a_1 a_2 \dots a_n$ szón ($a_i \in \Sigma$, $n > 0$) az $r_0, r_1, \dots, r_{n-1}, r_n$ állapotsorozat pontosan akkor elfogadó számítás, ha $A_{q_0} \Rightarrow a_1 A_{r_1} \Rightarrow a_1 a_2 A_{r_2} \Rightarrow \dots \Rightarrow$

$a_1a_2\ldots a_{n-1}A_{r_{n-1}} \Rightarrow a_1a_2\ldots a_{n-1}a_n$ egy levezetés. Másrészt $\varepsilon \in L(G)$ csak úgy teljesülhet, ha van $S \to \varepsilon$ szabály a nyelvtanban, aminek feltétele, hogy $q_0 \in F$, azaz ha $\varepsilon \in L(M)$.

Visszafelé tegyük fel, hogy $G = (V, \Sigma, S, P)$ egy, az L nyelvet generáló reguláris nyelvtan. A nyelvtan alapján megadunk egy $M = (Q, \Sigma, \delta, S, F)$ véges automatát L-hez. Az állapotok a változóknak fognak megfelelni, egy, a levezetés végét jelképező állapottal kiegészítve, azaz $Q = V \cup \{E\}$, ahol $E \notin V$. A kezdőállapot az S kezdőváltozó. Amennyiben nincs a nyelvtanban $S \to \varepsilon$ szabály, akkor legyen $F = \{E\}$, különben pedig $F = \{E, S\}$. Az állapotátmeneteket definiáljuk a következőképpen

$$\begin{aligned} \delta(A, a) &= B, \quad \text{ha van} \quad A \to aB \text{ szabály} \\ \delta(A, a) &= E, \quad \text{ha van} \quad A \to a \text{ szabály} \end{aligned}$$

Ezzel megadtunk egy M nemdeterminisztikus véges automatát. Ebből következik, hogy $L(M)$ reguláris (2.16. tétel), már csak azt kell megmutatni, hogy $L(M) = L(G)$.

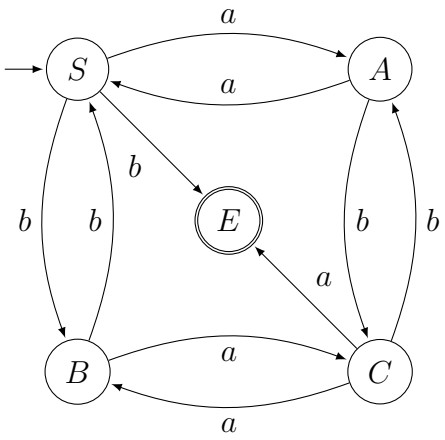
Ennek az automatának egy $x = a_1a_2\ldots a_n$ szóhoz $(n > 0)$ tartozó elfogadó számításai megfelelnek a nyelvtanban x levezetéseinek, mert $S \Rightarrow a_1A_1 \Rightarrow a_1a_2A_2 \Rightarrow \cdots \Rightarrow a_1a_2\ldots a_{n-1}A_{n-1} \Rightarrow a_1a_2\ldots a_{n-1}a_n$ pontosan akkor levezetés, ha az S, A_1, A_2, …, A_{n-1}, E az M véges automatának egy elfogadó számítása. A 0 hosszú szóra pedig teljesül, hogy $\varepsilon \in L(M)$ pontosan akkor, ha S elfogadó állapot, ez pedig pontosan akkor van, ha a nyelvtanban van $S \to \varepsilon$ szabály, azaz amikor $\varepsilon \in L(G)$. □

4.21. Megjegyzés *Ha $\varepsilon \notin L$, akkor felesleges megcsinálni a 4.19. szerinti átalakítást, hiszen nem lesz $S \to \varepsilon$ szabály a nyelvtanban, ami gondot okozhatna.*

4.22. Feladat *Az iménti bizonyításban használt konstrukció segítségével készítsünk automatát az alábbi nyelvtannal megadott nyelvhez:*

$$\begin{aligned} S &\to \mathtt{a}A \mid \mathtt{b}B \mid \mathtt{b} \\ A &\to \mathtt{a}S \mid \mathtt{b}C \\ B &\to \mathtt{a}C \mid \mathtt{b}S \\ C &\to \mathtt{a}B \mid \mathtt{b}A \mid \mathtt{a} \end{aligned}$$

Megoldás: A kapott nemdeterminisztikus véges automata



□

A korábban definiált reguláris nyelvtanokat szokták *jobb-reguláris nyelvtan*oknak is hívni, mivel a levezetések során minden lépésben a kapott karaktersorozat jobb oldalán van a változó. Ennek mintájára definiálhatunk bal-reguláris nyelvtanokat is.

4.23. Definíció Bal-reguláris *egy olyan nyelvtan, amelyben a levezetési szabályok* $A \to B\mathtt{a}$, *illetve* $A \to \mathtt{a}$ *alakúak. Továbbá megengedett az* $S \to \varepsilon$ *szabály is, amennyiben* S *nem fordul elő egyetlen szabály jobb oldalán sem.*

Egy bal-reguláris nyelvtan a 2. Chomsky-féle osztályban van, de nincs a 3. osztályban (ha van benne $A \to B\mathtt{a}$ típusú szabály). Azonban nem nehéz látni, hogy minden bal-reguláris nyelvtan „átfordítható" egy reguláris nyelvtanná a generált nyelv megőrzésével.

Tegyük fel, hogy van egy G bal-reguláris nyelvtanunk, ami az L nyelvet generálja. Ha a nyelvtan minden $A \to B\mathtt{a}$ szabályában megcseréljük a jobb oldalon álló két karaktert, azaz az előző szabály helyett az $A \to \mathtt{a}B$ szabályt tekintjük, akkor így egy jobb-reguláris nyelvtant kapunk, mely éppen az L nyelv fordítottját generálja. Az L fordítottját szokás L^{-1}-gyel jelölni, ez a nyelv éppen az L szavainak megfordítását tartalmazza, azaz $a_1a_2 \dots a_n$ pontosan akkor van L^{-1}-ben, ha $a_n \dots a_2a_1$ L-beli.

A fenti eljárással G-ből kapott jobb-reguláris nyelvtant G'-vel jelölve azt mondhatjuk tehát, hogy $L(G') = L^{-1}$. G'-höz a 4.20. tételben látott eljárással készítünk egy M' véges automatát, amire tehát igaz lesz, hogy $L(M') = L^{-1}$. Ezt az automatát könnyen átalakíthatjuk olyan automatává, melynek csak egyetlen elfogadó állapota van: ez megtehető például egy új elfogadó állapot bevezetésével, amibe minden korábbi elfogadó állapotból ε mozgással tudunk eljutni. Legyen ez az átalakított automata M''. Világos, hogy $L(M'') = L^{-1}$ is fennáll. Ha most M''-ben minden nyilat megfordítunk, a korábbi egyetlen elfogadó állapotot kezdőállapottá tesszük, a korábbi kezdőt pedig elfogadóvá, akkor az új M automata éppen $L(M'') = L^{-1}$ fordítottját, azaz L-et ismeri fel. M-hez a 4.20. tételben látott eljárással elkészíthetünk egy jobb-reguláris nyelvtant, ami ezek

szerint ekvivalens lesz a kiindulási bal-reguláris nyelvtanunkkal, azaz ugyanazt a nyelvet generálja.

A fordított irány (jobb-regulárisból bal-reguláris nyelvtan készítése) hasonlóan kapható: jobb-reguláris nyelvtan L-re $\rightarrow$ véges automata L-re a szokásos eljárással $\rightarrow$ véges automata L^{-1}-re a nyilak átforgatásával $\rightarrow$ jobb-reguláris nyelvtan L^{-1}-re a szokásos eljárással $\rightarrow$ bal-reguláris nyelvtan L-re a szabályok jobb oldalának felcserélésével.

5. fejezet

Reguláris nyelvek algoritmikus kérdései

Egy L reguláris nyelv az eddigi eredmények szerint megadható véges automatával, reguláris kifejezéssel vagy reguláris nyelvtannal. Felmerül, hogy egy így megadott nyelv néhány egyszerű tulajdonságát (üres-e, véges-e, két nyelv azonos-e), hogyan tudjuk eldönteni.

5.1. Beletartozás

Adott: $L \subseteq \Sigma^*$ reguláris nyelv, továbbá egy $x \in \Sigma^*$ szó.
Kérdés: $x \in L$?

Ha a nyelv determinisztikus véges automatával adott, akkor csak végig kell követni az állapotváltozásokat és kiderül, elfogadó állapotba jut-e.

Ha a nyelv egy M nemdeterminisztikus véges automatával van megadva, akkor az M automatának az x szón lehetséges összes számítását követni kell. Ezt megvalósíthatjuk úgy, hogy *determinizáljuk* az automatát (2.16. tétel), és az így kapott determinisztikus véges automatával dolgozunk az előzőek szerint. A determinizálás az M méretében exponenciális eljárás (de független az x szótól), utána már x hosszával arányos lépésszám elegendő.

Egy másik lehetőség, hogy determinizálás nélkül meghatározzuk, hogy az i-edik lépésben mely állapotokba kerülhet M. Ez lépésenként az M állapotainak számával arányos időben történhet. Ez az eljárás akkor gyorsabb az előzőnél, ha x hossza M méretéhez képest nem túl nagy.

Ha L reguláris nyelvtannal adott, akkor ki tudjuk próbálni az összes, legfeljebb $|x|$ lépésből álló levezetést, megkapjuk-e valamelyiknél az x szót. Ha minden változó legfeljebb k szabály bal oldalán áll, akkor ez legfeljebb $k^{|x|}$ lehetőség.

Hosszú x szavakra ennél jobb módszer, ha a nyelvtanból elkészítjük a véges automatát (4.20. tétel), és utána használjuk valamelyik előző módszert.

5.2. Üresség

Adott: $L \subseteq \Sigma^*$ reguláris nyelv.
Kérdés: $L = \emptyset$?

Ha az L nyelv (nemdeterminisztikus) automatával adott, akkor ez arra a gráfelméleti kérdésre vezet, hogy az automatában vezet-e út a kezdőállapotból legalább egy elfogadó állapotba. Ez például egy szélességi vagy mélységi bejárással eldönthető, a lépésszám az automata méretében lineáris lesz. Az utat alkotó élek címkéit sorban leolvasva megkapjuk a nyelv egy szavát is.

Azt is megtehettük volna, hogy a kezdeti, esetleg nemdeterminisztikus véges automatát *determinizáljuk* (2.16.), majd ezt *minimalizáljuk* (2.29.). Az eredménynek a 3.3. tételben mutatottnak kell lenni, amennyiben $L = \emptyset$. Hátránya ennek a módszernek, hogy a determinizálás az állapotok számában exponenciális idejű lehet, és végül a kapott, minimalizálandó véges automata mérete is lehet exponenciális.

Ha L reguláris nyelvtannal adott, akkor elkészíthetünk hozzá egy nemdeterminisztikus véges automatát (4.20. tétel) és ebben vizsgálhatjuk az elérhetőséget.

Egy reguláris kifejezés által leírt nyelv esetén vagy a 3.20. állítás alapján elkészítjük a reguláris kifejezésből a véges automatát, és megint a kezdőállapotból az elfogadó állapotok elérhetőségét vizsgáljuk, vagy lehet közvetlenül a reguláris kifejezésből is okoskodni. Ehhez azt kell használni, hogy $L(r_1+r_2)$ pontosan akkor lesz üres, ha $L(r_1) = L(r_2) = \emptyset$, míg $L(r_1r_2)$ pontosan akkor üres, ha az $L(r_i)$ nyelvek legalább egyike üres. Ezekből, és abból, hogy $L(r^*)$ soha sem üres, a teljes reguláris kifejezés felépítését követve meghatározható, hogy L az üres nyelv-e vagy nem.

5.3. Egyenlőség

Adott: $L_1, L_2 \subseteq \Sigma^*$ reguláris nyelvek.
Kérdés: $L_1 = L_2$?
(Ez az előző kérdés általánosítása, amely $L_2 = \emptyset$ speciális esetben kapható.)

Amennyiben a nyelvekhez van egy-egy, akár nemdeterminisztikus véges automatánk, akkor ezeket szükség esetén *determinizálva* (2.16.) tétel, majd *minimalizálva* (2.29., elegendő a kapott két automatát összehasonlítani. A kezdőállapotokat megfeleltetjük egymásnak, majd az állapotátmenetek segítségével a két automata további állapotait is megpróbáljuk azonosítani. Ha ez maradéktalanul sikerül, és az elfogadó állapotok is elfogadó állapotoknak felelnek meg, akkor a két automata izomorf, a nyelvek azonosak. Egyébként pedig a minimálautomata egyértelműsége miatt a nyelvek különböznek.

5.1. Megjegyzés *A két automata izomorfiája a vázolt módon lineáris időben megoldható. Ez egy sokkal egyszerűbb kérdés, mint általában két gráf izomorfiájának eldöntése.*

A determinisztikus véges automaták gráfjainál a címkék segítenek, mert egyértelműen

meghatározzák az állapothalmazok közötti egyetlen lehetséges megfeleltetést, amit már csak ellenőriznünk kell.

5.2. Megjegyzés *Bár az eljárás a minimálautomaták méretében lineáris, de ha eredetileg nemdeterminisztikus automatánk volt, a determinizálás során az automaták mérete exponenciálisan megnőhetett.*

Ha a nyelvek reguláris nyelvtannal vagy reguláris kifejezéssel adottak, akkor célszerű elkészíteni belőlük az automatákat és utána az előző módszert használni.

A kérdést visszavezethetjük az üresség eldöntésére is (amennyiben L_1 és L_2 determinisztikus teljes véges automatáját elkészítettük), mert $L_1 = L_2$ akkor és csak akkor teljesül, ha $(L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2) = \emptyset$, és a komplementer nyelvekhez, metszetekhez, unióhoz tudunk véges automatát készíteni (3.4. tétel).

5.4. Diszjunktság

Adott: $L_1, L_2 \subseteq \Sigma^*$ reguláris nyelvek.
Kérdés: $L_1 \cap L_2 = \emptyset$?

Itt is az a célszerű. hogy (még ha nem is úgy vannak megadva), egy-egy, akár nemdeterminisztikus véges automatát veszünk a nyelvekhez. Ezekből az $L_1 \cap L_2$ nyelvhez is tudunk egy véges automatát készíteni (3.4. tétel). Erről azt kell eldönteni, van-e olyan szó, amit elfogad. Ha nincs ilyen, akkor a nyelvek diszjunktak, különben nem. Így tehát visszavezettük a problémát a metszet nyelv ürességének eldöntésére.

5.5. Végesség

Adott: $L \subseteq \Sigma^*$ reguláris nyelv.
Kérdés: L elemszáma véges?

Ezt is az automaták alapján lehet egyszerűen eldönteni, de szükségünk van a pumpálási lemma (3.11. lemma) egy következményére is.

5.3. Tétel *Legyen L reguláris nyelv és n a hozzá tartozó minimálautomata állapotainak száma. Az L nyelvnek akkor és csak akkor van végtelen sok eleme, ha létezik olyan $w \in L$ szó, hogy $n \leq |w| \leq 2n$.*

Bizonyítás. $\Leftarrow$:

A pumpálási lemma bizonyításából tudjuk, hogy n lesz a pumpálási hossz, tehát ha létezik legalább n hosszú az L nyelvbe tartozó szó, akkor az pumpálható, amivel a nyelv végtelen sok különböző elemét kapjuk.

$\Rightarrow$:

A másik irányhoz azt kell megmutatnunk, hogy a nyelv végtelen sok szava között kell legyen olyan, melynek hossza az $[n, 2n]$ intervallumba esik. Legyen $x \in L$ egy legrövidebb szó, melyre $|x| \geq n$. Ilyen x szó a nyelv végtelensége miatt biztosan van. Belátjuk, hogy ez az x megfelelő. Ugyanis, ha $|x| > 2n$ lenne, akkor a pumpálási lemma szerinti felírható $x = uvw$ alakban úgy, hogy $uv^k w \in L$ minden $k \geq 0$ esetén. Ha ezt $k = 0$ választással alkalmazzuk, akkor az $uw \in L$ szót kapjuk. Ennek a szónak a hosszára igaz, hogy $|uw| = 2n - |v| \geq n$, mivel, a pumpálási lemma miatt v hossza legfeljebb n. Tehát találtunk egy olyan szót az L nyelvben, aminek hossza legalább n, de rövidebb, mint x hossza, ami ellentmond x választásának. □

A fenti tétel értelmében a nyelv végességének eldöntéséhez elegendő, ha kipróbáljuk, van-e olyan x szó, melynek hossza n és $2n$ közé esik, és amit az előállított minimálautomata elfogad.

Ez az eljárás a minimálautomata (esetleg exponenciálisan sok lépést igénylő) előállításán túl még exponenciálisan sok ($2^{2n} - 2^n - 1 = 2^n(2^n - 1) - 1$) szó végigkövetését is jelentheti az automatán.

5.4. Megjegyzés *Minden véges nyelv reguláris, mert például az a reguláris kifejezés jó hozzá, ami a nyelv szavainak összegéből áll. (Véges automatát és reguláris nyelvtant is egyszerű megadni egy véges nyelvhez.)*

6. fejezet

Környezetfüggetlen nyelvek

A 4.1 fejezetben leírtak szerint egy L nyelv akkor környezetfüggetlen (röviden CF), ha van hozzá 2. osztálybeli nyelvtan. Egy ilyen nyelvtan minden szabálya egy változó egy lehetséges helyettesítését adja meg.

6.1. Levezetési fa

Egy szó CF nyelvtannal történő levezetése sokszor jobban áttekinthető ha a lépéseket egy fába rendezzük.

6.1. Definíció *Legyen G egy 2. osztálybeli nyelvtan és x egy szó. Az x* levezetési fája *G-ben egy gyökeres fa, melyben*

- *a gyökér a kezdőváltozóval,*
- *minden nem levél csúcs egy-egy változóval,*
- *minden levél pedig Σ egy-egy elemével (vagy ε-nal) van címkézve.*
- *Ha egy A csúcs gyerekei balról jobbra olvasva $B_1, B_2, \ldots, B_k$, akkor a nyelvtannak van $A \to B_1 B_2 \ldots B_k$ szabálya. (Itt $B_i \in \Sigma \cup V \cup \{\varepsilon\}$.)*
- *A levelek balról jobbra olvasva éppen az x szót adják.*

A definícióból világos, hogy egy $x \in L(G)$ szó tetszőleges levezetéséből lehet levezetési fát készíteni, és a levezetési fából is kiolvasható legalább egy levezetés.

Fontos azonban megjegyezni, hogy míg a levezetés egyértelműen meghatározza a fát, visszafelé ez nem igaz, általában egy levezetési fából ugyanannak a szónak több levezetése is kiolvasható. Ha viszont ragaszkodunk a bal-levezetéshez, akkor ezt már a fa egyértelműen meghatározza.

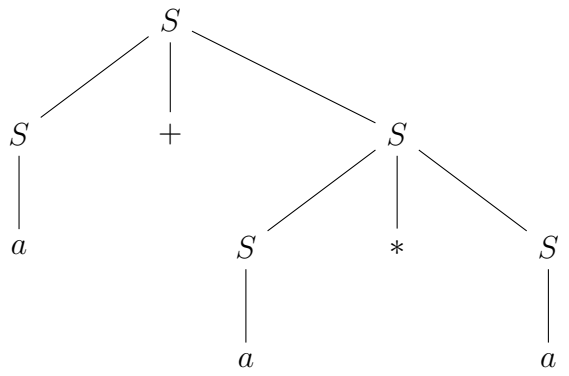
6.2. Példa *Az alábbi CF nyelvtan az egyszerű aritmetikai kifejezéseket írja le:*

$$S \to \underbrace{S+S}_{1} \mid \underbrace{S*S}_{2} \mid \underbrace{(S)}_{3} \mid \underbrace{a}_{4}$$

Itt S az egyetlen változó, az ábécé elemei pedig $+$,$*$, a, valamint a nyitó és csukó zárójel.
Egy lehetséges levezetés:

$$S \overset{1}{\Rightarrow} \underline{S}+S \overset{4}{\Rightarrow} a+\underline{S} \overset{2}{\Rightarrow} a+\underline{S}*S \overset{4}{\Rightarrow} a+a*\underline{S} \overset{4}{\Rightarrow} a+a*a$$

Az ehhez tartozó levezetési fa pedig



Ebből a levezetési fából másik levezetés is kiolvasható, például

$$S \overset{1}{\Rightarrow} S+\underline{S} \overset{2}{\Rightarrow} S+\underline{S}*S \overset{4}{\Rightarrow} S+a*\underline{S} \overset{4}{\Rightarrow} \underline{S}+a*a \overset{4}{\Rightarrow} a+a*a$$

vagy lehet ún. jobb-levezetést *is készíteni, amikor mindig a legutolsó változót helyettesítjük*

$$S \overset{1}{\Rightarrow} S+\underline{S} \overset{2}{\Rightarrow} S+S*\underline{S} \overset{4}{\Rightarrow} S+\underline{S}*a \overset{4}{\Rightarrow} \underline{S}+a*a \overset{4}{\Rightarrow} a+a*a$$

6.3. Megjegyzés *Figyeljük meg, hogy reguláris nyelvtanok esetén a levezetési fa egy olyan bináris fa lesz, amelynek minden levele a szülő csúcs bal oldalán van. Ilyenkor a levezetési fához egyetlen levezetés tartozik, minden levezetés bal-levezetés és jobb-levezetés is egyben.*

6.2. Egyértelműség

Egy nyelvtannal kapcsolatban az egyik fontos kérdés az, hogy mely szavakat generálja. Gyakran az is fontos kérdés azonban, hogy az egyes generált szavakat hányféleképp tudja a nyelvtan előállítani. Ebben a részben ezt a kérdést járjuk körül.

6.4. Definíció *Egy $w \in \Sigma^*$ szó* egyértelműen levezethető *a G nyelvtanból, ha G-ben csak egy levezetési fája van.*

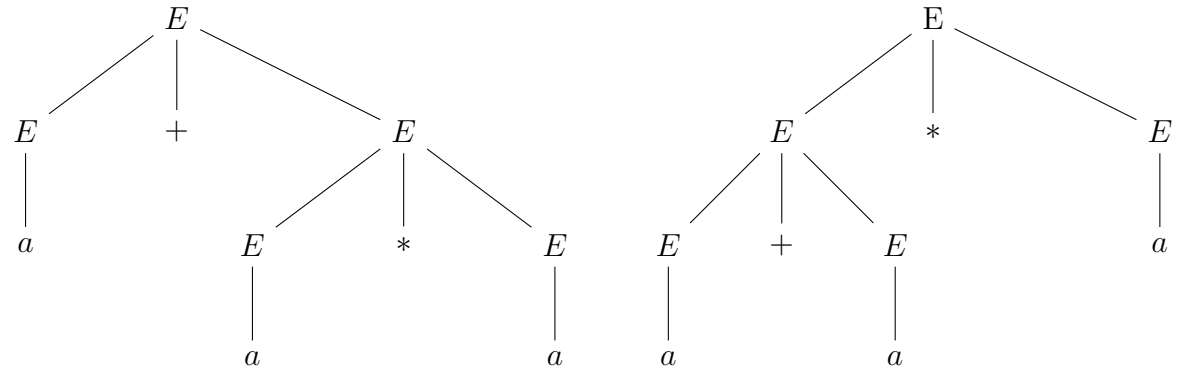
A G nyelvtan egyértelmű*, ha G-ből minden $w \in L(G)$ szó egyértelműen levezethető.*

Az L nyelv egyértelmű*, ha létezik egyértelmű nyelvtana.*

6.5. Megjegyzés *Az egyértelműen levezethetőség fenti definíciója ekvivalens azzal, hogy a szó bal-levezetése egyértelmű.*

6.6. Állítás *Az $E \to E + E \mid E * E \mid (E) \mid a$ nyelvtan nem egyértelmű, de a generált nyelve egyértelmű.*

Bizonyítás. Például az $a + a * a$ szó levezetése nem egyértelmű, mert két levezetési fa is tartozik hozzá.



A két megfelelő bal-levezetés:

$$E \Rightarrow E + E \Rightarrow a + E \Rightarrow a + E * E \Rightarrow a + a * E \Rightarrow a + a * a,$$

illetve

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

Tekintsük az alábbi G' nyelvtant:

$$\begin{aligned} E &\to E + T \mid T \\ T &\to T * F \mid F \\ F &\to (E) \mid a \end{aligned} \tag{6.1}$$

Világos, hogy a G' nyelvtannal levezethető aritmetikai kifejezések levezethetők az eredeti nyelvtanból is.

Lássuk be most azt, hogy ami az eredeti nyelvtannal levezethető, az egyértelműen levezethető G'-vel is.

Legyen w egy, az eredeti nyelvtanból levezethető szó. A w hossza szerinti teljes indukcióval bizonyítható, hogy w a G' nyelvtanban egyértelműen vezethető le.

Legyen ugyanis $|w| = 1$. Ekkor $w = a$, és ez csak a $E \Rightarrow T \Rightarrow F \Rightarrow a$ lépésekkel kapható meg.

Tegyük fel, hogy már minden n-nél rövidebb szóra ($n > 1$) igaz az állítás és legyen $|w| = n$. Ha van a w-ben zárójelen kívüli $+$, akkor a levezetés első lépésének ezek közül az utolsót létre kell hoznia, tehát a kezdés $E \Rightarrow E \underset{\substack{\uparrow \\ \text{utolsó}}}{+} T$. Az itt lévő $+$ a levezetésben az utolsó. Ekkor a megfelelő részszavakat kell a keletkezett E, illetve T változóból generálni, és ezek, az indukciós feltevés miatt, már egyértelműen tehetők meg.

Ha nincs zárójelen kívüli $+$, de van zárójelen kívüli $*$ a w szóban, akkor hasonló megfontolásból a levezetés első néhány lépése $E \Rightarrow T \Rightarrow E \underset{\substack{\uparrow \\ \text{utolsó}}}{*} T$ kell legyen, a folytatás pedig ismét az indukciós feltevés miatt egyértelmű.

Amennyiben sem $+$, sem $*$ nincs zárójelen kívül, akkor a kezdés $E \Rightarrow T \Rightarrow F \Rightarrow (E)$ lesz, és ismét az indukció miatt egyértelmű a folytatás. □

6.7. Megjegyzés *A levezetési fa az aritmetikai kifejezések esetén tekinthető a kiértékelési folyamat leírásának is. Ebben az értelemben az első fa azt jelenti, hogy előbb a szorzást végezzük el, utána adjuk hozzá az első paramétert, míg a második fa a két első paraméter összegét szorozza a harmadikkal. Ha a műveletek szokásos elvégzési sorrendjét akarjuk megvalósítani, akkor az első változat a helyes, és vegyük észre, hogy a fent megadott egyértelmű nyelvtan ezt a műveleti sorrendet valósítja meg.*

6.8. Megjegyzés *A programozási nyelvekben előforduló feltételes utasítás nyelvtana az alábbi, legegyszerűbb formában nem egyértelmű:*

if (felt) utasítás
if (felt) utasítás1 **else** utasítás2

Itt az **if** *és* **else** *utasításokat a Σ ábécé karaktereinek tekintjük, a* felt *és* utasítás *a feltételeket, illetve a feltételek teljesülése esetén végrehajtandó utasításokat jelképezik.*

Például az **if** (felt1) **if** (felt2) utasítás1 **else** utasítás2 *esetén az* **else** *ág bármelyik* **if** *része lehet, de szerencsére itt is van megfelelő egyértelmű nyelvtan, a programozási nyelvek természetesen ez utóbbit használják.*

A fenti nyelvtanhoz lehet készíteni egyértelmű verziót, de ez nincs mindig így, ezt mondja ki a következő tétel.

6.9. Tétel *Létezik nem egyértelmű nyelv.*

A bizonyítást lásd később (6.40. tétel). Megjegyezzük, hogy a nyelvtanok egyértelműségének eldöntése általában nem könnyű feladat, lásd a 12.16. tételt.

6.3. CF nyelvtanok átalakítása

Célunk, hogy egy környezetfüggetlen nyelvhez kényelmesebb, vagy számítógépes felhasználás szempontjából jobb nyelvtant találjunk.

6.3.1. ε-szabályok

Ebben a részben azt mutatjuk meg, hogy lehetséges kissé enyhíteni a korábban tett megkötéseinken a CF nyelvtanok szabályainak alakjára vonatkozóan: megengedhetünk ε jobb oldalú szabályokat korlátozás nélkül, ebben az esetben is a környezetfüggetlen nyelveket tudjuk generálni.

Legyen $G = (V, \Sigma, S, P)$ egy olyan nyelvtan, amiben minden szabály $A \to \alpha$ alakú. Ez az esetleg előforduló $A \to \varepsilon$ szabályok miatt nem feltétlenül tartozik a 2. osztályba, de megmutatjuk, hogy átalakítható 2. osztálybeli $G' = (V', \Sigma, S', P')$ nyelvtanná. Ezen állítás miatt, ha úgy kényelmesebb, egy nyelv környezetfüggetlen voltát egy ilyen nyelvtan megadásával is igazolhatjuk.

6.10. Definíció *Nevezzük* elenyésző *változóknak azokat az $A \in V$ változókat, amelyekből az ε szó levezethető.*

ε-mentesítés:

- Elenyésző változók meghatározása
  - $N_1 = \{A \in V : (A \to \varepsilon) \in P\}$
  - $i = 1, 2, \ldots,$ esetén legyen

$$N_{i+1} = N_i \cup \{A \in V : \text{ létezik } (A \to \alpha) \in P, \text{ hogy } \alpha \in N_i^*\}$$

  - Ha $N_{i+1} = N_i$, akkor az eljárás leáll, legyen N ez az utolsó halmaz.
- A nyelvtan átalakítása
  - ha $(A \to \alpha) \in P$ és $\alpha \neq \varepsilon$, akkor legyen $(A \to \beta) \in P'$ minden olyan $\beta \neq \varepsilon$ sorozatra, ami az α sorozatból úgy kapható, hogy annak valahány (akár nulla darab) elenyésző változóját elhagyjuk (ε-nal helyettesítjük).
  - Ha $S \in N$, akkor S' egy új változó lesz az $S' \to \varepsilon \mid S$ szabályokkal, különben $S' = S$.

Az eljárás helyességét három lemmán keresztül bizonyítjuk.

6.11. Lemma *Van olyan $i \leq |V|$, melyre $N_i = N_{i+1}$ és ekkor $N = N_i$ az elenyésző változókból áll.*

Bizonyítás. Mivel $N_1 \subseteq N_2 \subseteq \cdots \subseteq V$, ez a halmazsorozat legfeljebb $|V|$ lépésig nőhet. Ha $N_i = N_{i+1}$, akkor $N_i = N_j$ a $j \geq i+1$ esetekben N_{i+1} definíciója miatt.

Egyrészt világos, hogy N csak elenyésző változókat tartalmaz, másrészt N_1 éppen az egy lépésben elenyésző változókból áll. Így N_2 biztosan tartalmazza a legfeljebb 2 lépésben elenyészőket, általában N_j pedig tartalmazza azokat a változókat, amelyekből legfeljebb j lépésben levezethető az üres szó. Mivel $N_j \subseteq N$ teljesül, ezért N minden elenyésző változót tartalmaz. □

6.12. Lemma *Legyen $x \neq \varepsilon$ egy szó. Ha x az $A \in V$ változóból indulva levezethető G-ben, akkor az A változóból indulva G'-ben is levezethető.*

Bizonyítás. A G-beli levezetés álljon n lépésből. Az állítást n szerinti teljes indukcióval bizonyítjuk.

Ha $n = 1$, akkor G-ben van $A \to x$ szabály, és mivel $x \neq \varepsilon$, ez szabály G'-ben.

Legyen $n > 1$ és tegyük fel, hogy G minden változójára és minden abból n-nél rövidebben levezethető szóra már igaz az állítás. Az x levezetésének első lépése legyen $A \Rightarrow \alpha$ és $\alpha = X_1 \dots X_k$, ahol $X_i \in V \cup \Sigma$. Ha X_i egy változó, akkor jelölje $x_i \in \Sigma^*$ az x szónak azt a darabját, melyet a levezetés során X_i-ből kapunk. Amennyiben $X_i \in \Sigma$, akkor legyen $x_i = X_i$. Az α sorozatból minden olyan X_i-t hagyjunk el, amelyre $x_i = \varepsilon$, legyen ennek eredménye β. Ekkor az $A \to \beta$ a G' egy szabálya, hiszen az elhagyott változók elenyészők, és G'-ben van $A \Rightarrow \beta \Rightarrow \cdots \Rightarrow x$ levezetés, mert a meghagyott X_i változókból, az indukciós feltevés miatt a megfelelő x_i szavak levezethetők. □

6.13. Lemma *Legyen $x \neq \varepsilon$ egy szó. Ha x az $A \in V$ változóból indulva levezethető G'-ben, akkor az A változóból indulva G-ben is levezethető.*

Bizonyítás. Ehhez csak azt kell észrevenni, hogy egy G'-beli levezetésben használt minden $X \Rightarrow \beta$ helyettesítés a G nyelvtan egy $X \to \alpha = X_1 X_2 \dots X_k$ szabályából származik, néhány elenyésző X_i változó elhagyásával. Ezért $X \Rightarrow \alpha \Rightarrow \cdots \Rightarrow \beta$ egy levezetés G-ben, amikor a megfelelő elenyésző változókra egy $X_i \Rightarrow \cdots \Rightarrow \varepsilon$ levezetést alkalmaztunk. □

6.14. Tétel *Ha a G nyelvtan minden szabálya $A \to \alpha$ alakú, akkor az $L(G)$ nyelv környezetfüggetlen.*

Bizonyítás. A fent leírt eljárással kapott G' nyelvtan minden szabálya $A \to \alpha$ alakú, ahol α csak akkor lehet az üres szó, ha $A = S'$ az újonnan bevezetett kezdőváltozó, ami csak az $S' \to \varepsilon \mid S$ szabályokban szerepel, ezért a kapott nyelvtan valóban a 2. osztályba tartozik.

A két előző lemma $A = S$, illetve $A = S'$ választással garantálja, hogy minden $x \neq \varepsilon$ szóra $x \in L(G)$ akkor és csak akkor teljesül, ha $x \in L(G')$.

Még az $x = \varepsilon$ esetet kell ellenőrizni. A leírt eljárás szerint $\varepsilon \in L(G)$ akkor és csak akkor igaz, ha $S \in N$, és pontosan ekkor lesz $\varepsilon \in L(G')$, mert G' csak akkor generálja az üres szót, ha bevezettük az új S' kezdőváltozót az $S' \to \varepsilon$ szabállyal együtt, ezt pedig éppen $S \in N$ esetén tesszük. □

6.15. Feladat *Az alábbi nyelvtant alakítsuk át CF nyelvtanná!*

$$\begin{aligned}
&S \to ABCD\\
&A \to CD \mid AC\\
&B \to Cb\\
&C \to \mathtt{a} \mid \varepsilon\\
&D \to \mathtt{b}D \mid \varepsilon
\end{aligned}$$

Megoldás: Először határozzuk meg az elenyésző változók halmazát!

$$\begin{aligned}
N_0 &= \{C, D\}\\
N_1 &= \{C, D, A\}\\
N_2 &= \{C, D, A\}
\end{aligned}$$

Mivel $N_1 = N_2$, az algoritmus véget ér.

Az S kezdőváltozó nem elenyésző, ezért ε nincs benne a nyelvben, így nincs szükség új kezdőváltozóra sem.

A meglévő szabályokat egészítsük ki az újakkal, ahol az elenyésző változókat minden lehetséges módon helyettesítjük ε-nal. A kapott új (és már CF) nyelvtan:

$$\begin{aligned}
&S \to ABCD \mid BCD \mid ABD \mid ABC \mid BD \mid BC \mid AB \mid B\\
&A \to CD \mid C \mid D \mid AC\\
&B \to C\mathtt{b} \mid \mathtt{b}\\
&C \to \mathtt{a}\\
&D \to \mathtt{b}D \mid \mathtt{b}
\end{aligned}$$

□

6.16. Megjegyzés *Ha a G nyelvtanban minden szabály vagy $A \to aB$ vagy $A \to a$ vagy $A \to \varepsilon$ alakú, ahol $A, B \in V$, $a \in \Sigma$, akkor a fenti eljárás egy reguláris G' nyelvtant eredményez.*

6.3.2. Egyszeres szabályok

Az előző pontban kapott, ε-szabályt már nem tartalmazó nyelvtanunk még tartalmazhat olyan, ún. egyszeres szabályokat, melyeknek jobb oldalán egyetlen változó áll. Ez belefér a 2. típusú nyelvtan definíciójába, de néha könnyebb lenne olyan nyelvtannal dolgoznunk, amiben ilyen szabályok nincsenek. Most megmutatjuk, hogy ezek a szabályok is kiküszöbölhetőek a nyelvtanból.

6.17. Definíció Egyszeres szabály*nak hívjuk az $A \to B$ szabályokat, ahol $A, B \in V$.*

Legyen adott egy $G = (V, \Sigma, S, P)$ környezetfüggetlen nyelvtan.
Egyszeres szabályok kiküszöbölése:

- Vegyük az egyszeres szabályok H gráfját, azaz azt a gráfot, melynek csúcsai a változók, és az A csúcsból akkor megy él a B csúcsba, ha van $A \to B$ szabály a nyelvtanban.
- Minden $A \in V$ csúcsra határozzuk meg a H gráfban belőle elérhető változók $U(A)$ halmazát. ($A \in U(A) \subseteq V$).
- A G nyelvtanból hagyjuk el az egyszeres szabályokat
- Ha $B \in U(A)$, akkor minden $(B \to \beta) \in P$, $|\beta| \neq 1$ esetén vegyük hozzá a nyelvtanhoz az $A \to \beta$ szabályt.

6.18. Tétel *Az így kapott G' környezetfüggetlen nyelvtanban nincs egyszeres szabály és $L(G) = L(G')$.*

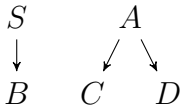
Bizonyítás. Világos, hogy az eljárással egy CF nyelvtant kapunk. Mivel először elhagytuk az egyszeres szabályokat, a G' nyelvtanban nincsenek egyszeres szabályok. Azt kell tehát még megmutatnunk, hogy $L(G) = L(G')$.

Egy, a G nyelvtanhoz tartozó $S \Rightarrow \cdots \Rightarrow x \in \Sigma^*$ levezetés, ha nem alkalmazunk benne egyszeres szabályt, akkor érvényes levezetés a G' nyelvtan alapján is. Amikor van benne egy $A_1 \Rightarrow A_2 \Rightarrow \ldots A_k \Rightarrow \alpha$ rész, ami az utolsó lépés kivételével egyszeres szabályokból áll, akkor $A_k \in U(A_1)$ és ezért a G' nyelvtan alapján $A_1 \Rightarrow \alpha$ egy levezetési lépés, azaz az egyszeres szabályok sorozata „kivágható” a levezetésekből.

A másik irányban, a G' egy levezetésében tetszőleges $A \Rightarrow \alpha$ lépés megvalósítható a G nyelvtanban egy $A \Rightarrow A_1 \Rightarrow \cdots \Rightarrow A_k \Rightarrow \alpha$ lépéssorozattal (ha $(A \to \alpha) \notin P$, akkor $A_k \in U(A)$, az utolsó lépés előtti szabályalkalmazások egyszeres szabályokat használnak). □

6.19. Feladat *Folytatva az előző feladatot, szüntessük meg a kapott nyelvtan egyszeres szabályait!*

Megoldás: Rajzoljuk fel a gráfot:



Ez alapján az átalakított nyelvtan

$$S \to ABCD \mid BCD \mid BD \mid BC \mid \overbrace{Cb \mid b}^{B \text{ helyett}} \mid ABD \mid ABC \mid AB$$

$$A \to CD \mid \overbrace{\texttt{a}}^{C \text{ helyett}} \mid \overbrace{\texttt{b}D \mid \texttt{b}}^{D \text{ helyett}} \mid AC$$

$$B \to C\texttt{b} \mid \texttt{b}$$

$$C \to \texttt{a}$$

$$D \to \texttt{b}D \mid \texttt{b}$$

□

6.3.3. Felesleges szimbólumok

Tekintsünk egy nem üres L környezetfüggetlen nyelvet. Egy $G = (V, \Sigma, S, P)$ nyelvtan hasznos szimbólumai azok amelyek az $L(G) = L$ nyelv valamely szavának legalább egy levezetésében előfordulnak. A többi, *felesleges* szimbólumra nincs igazán szükség, a nyelvtant (és a levezetések keresését) egyszerűsítjük, ha ezektől megszabadulunk.

Alapvetően kétféle felesleges szimbólum van. Az egyik esetben az a baj, hogy egy X változóból nem lehet Σ^* egyetlen elemét sem levezetni (nem terminálódik), a másik esetben pedig az a baj, hogy az $Y \in V \cup \Sigma$ szimbólum egyetlen S-ből induló levezetésben sem jelenik meg (nem elérhető).

Felesleges szimbólumok elhagyása

- Nem terminálódók elhagyása
  - $B_0 = \Sigma$
  - $i = 1, 2, \ldots$ esetén

$$B_i = B_{i-1} \cup \{A \in V : \text{ létezik } (A \to \alpha) \in P, \text{ amire } \alpha \in B_{i-1}^*\}$$

  - Ha $B_i = B_{i+1}$, akkor álljon V' a B_i-beli változókból, azaz $V' = B_i \cap V$.
  - Hagyjuk el azokat a szabályokat, amelyek tartalmaznak nem V'-beli változót (a szabály bármelyik oldalán).
- Nem elérhetők elhagyása
  - $T_0 = \{S\}$
  - $i = 1, 2, \ldots$ esetén

$$T_i = T_{i-1} \quad \cup \quad \{Z \in V' \cup \Sigma : \text{ létezik } (A \to \alpha) \in P, \text{ hogy } A \in T_{i-1} \text{ és } Z \text{ előfordul } \alpha\text{-ban}\}$$

– Ha $T_i = T_{i+1}$, akkor $V'' = T_i \cap V'$ és $\Sigma' = T_i \cap \Sigma$.

– Hagyjuk el azokat a szabályokat, amelyek tartalmaznak nem T-beli szimbólumot (a szabály bármelyik oldalán).

6.20. Tétel *Minden $L \neq \emptyset$ környezetfüggetlen nyelv generálható felesleges szimbólumok nélküli környezetfüggetlen nyelvtannal.*

Bizonyítás. Legyen $G = (V, \Sigma, S, P)$ egy CF nyelvtan amire $L(G) = L$. Alkalmazzuk G-re a fenti kétrészes eljárást. Az eljárás véget ér, mert $\Sigma \subseteq B_0 \subseteq B_1 \subseteq \cdots \subseteq \Sigma \cup V$ és $\{S\} \subseteq T_0 \subseteq T_1, \subseteq \cdots \subseteq \Sigma \cup V$, így legfeljebb $|\Sigma \cup V| - 1$ bővítés lehetséges. A végén kapott nyelvtan CF lesz, hiszen csak elhagytunk dolgokat egy CF nyelvtanból.

Világos, hogy az új nyelvtan ugyanazt generálja, amit az eredeti, hiszen csak olyan szabályokat dobtunk ki, amik soha nem vehettek részt terminálódó levezetésben. Azt kell tehát már csak megmutatnunk, hogy az új nyelvtan nem tartalmaz felesleges szimbólumokat.

Mivel a végső B_j-ben azok a változók vannak, amelyekből egy $w \in \Sigma^*$ szó levezethető, ezért V' a terminálódó változókból fog állni. A végső T_j halmaz a kezdőváltozóból elérhető szimbólumokat tartalmazza, tehát V'' változói elérhetők. Azt kell még látni, hogy a második részben nem keletkeznek nem terminálódó változók. Ha G-ben volt egy $A \Rightarrow \cdots \Rightarrow x \in \Sigma^*$ levezetés és $A \in V''$, akkor a levezetésben szereplő minden szimbólum elérhető a kezdőváltozóból (pl. A-n keresztül), ezért a szereplő változók mindegyike benne lesz a V'' halmazban és így $x \in \Sigma'^*$. □

6.21. Feladat *Hagyjuk el a felesleges szimbólumokat az alábbi nyelvtanból:*

$$S \to AB \mid \mathtt{a} \quad A \to B \mid \mathtt{b}$$

Megoldás: $B_0 = \{\mathtt{a}, \mathtt{b}\}$, $B_1 = \{S, A, a, b\}$, $B_2 = B_1$, ezért $V' = \{S, A\}$, a nyelvtan ezen a ponton

$$S \to \mathtt{a} \quad A \to \mathtt{b}$$

Tovább folytatva $T_0 = \{S\}$, $T_1 = \{S, a\}$, $T_2 = T_1$, a nyelvtan pedig végül ez lesz:

$$S \to \mathtt{a}$$

□

6.22. Megjegyzés *Ha a két részt fordított sorrendben végeznénk, akkor maradhatnak felesleges szimbólumok a nyelvtanban. Például, ha ez előző feladat nyelvtanából előbb a*

nem elérhetőeket szűrjük ki, akkor ebben a részben a nyelvtan nem változik, utána a nem terminálódók kiszűrésekor az

$$S \to \texttt{a} \quad A \to B \mid \texttt{b}$$

nyelvtan az eredmény, amiben az A változó és a `b` *terminális felesleges.*

6.23. Megjegyzés *Az eljárást egy reguláris nyelvtanra elvégezve az eredmény is reguláris lesz.*

A fenti három eljárás (ε-szabályok kiirtása, egyszeres szabályok eltüntetése, felesleges szimbólumok elhagyása) ebben a sorrendben végezve egy olyan nyelvtant eredményez, amiben a tárgyalt három jelenség egyike sem fordul elő. Ehhez azt kell csak meggondolnunk, hogy az egyszeres szabályok eltüntetése és a felesleges szimbólumok elhagyása nem hoz létre ε-szabályt illetve a felesleges szimbólumok elhagyása nem hoz létre sem ε, sem egyszeres szabályt.

6.4. Normálformák

Sokszor megkönnyíti a környezetfüggetlen nyelvekkel és nyelvtanokkal való munkát, ha feltételezhetjük, hogy a környezetfüggetlen nyelvtan (a szokásos megkötéseken felül) valami speciális alakban adott. A következőkben ilyen speciális alakokkal, úgy nevezett normálformákkal foglalkozunk.

6.4.1. Chomsky-normálforma

6.24. Definíció *Egy CF nyelvtan* Chomsky-normálformájú (CNF)*, ha a szabályai*

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

alakúak, ahol $a \in \Sigma$ és $A, B, C \in V$

Látszik, hogy ha a G nyelvtan CNF, akkor $\varepsilon \notin L(G)$. Most azt fogjuk megmutatni, hogy amennyiben elhagyjuk az esetlegesen előállítható ε szót egy környezetfüggetlen nyelvből, akkor az így kapott nyelvre már van CNF nyelvtan.

6.25. Tétel *Minden G CF nyelvtanhoz lehet készíteni olyan Chomsky-normálformájú G' nyelvtant, hogy $L(G') = L(G) \setminus \{\varepsilon\}$.*

Bizonyítás. A következő lépésekkel megkaphatunk egy kívánt G' nyelvtant:

- Legyen G_1 az a CF nyelvtan, amit G-ből az egyszeres szabályok kiküszöbölésével kapunk (lásd 6.3.2 fejezet).

- Ha G_1-ben van $S \to \varepsilon$ szabály, ezt hagyjuk el, így kapjuk a G_2 CF nyelvtant.
- Minden $a \in \Sigma$ karakterhez, ami valamelyik G_2-beli szabály legalább 2 hosszúságú jobb oldalán fordul elő
  - felveszünk egy új változót: X_a
  - felveszünk egy új szabályt: $X_a \to a$
  - G_2 minden szabályában, ahol a jobb oldalon a nem egyedüli karakter, X_a-val helyettesítjük a-t
- ezután minden $A \to B_1 B_2 \dots B_k$ szabály helyett (itt B_i-k már mind változók), ha $k > 2$ bevezetjük az

$$\begin{aligned} A &\to B_1 C_1 \\ C_1 &\to B_2 C_2 \\ &\vdots \\ C_{k-2} &\to B_{k-1} B_k \end{aligned}$$

  új szabályokat, ahol $C_1, \dots, C_{k-2}$ új változók.

Az előzőek miatt $L(G_1) = L(G)$, $L(G_2) = L(G) - \{\varepsilon\}$ és könnyű látni, hogy a további átalakítások már nem változtatnak ezen a nyelven.

Az eljárás eredményeként kapott G' nyelvtanban nincs egyszeres szabály (G_1-ben sincs már és később sem veszünk hozzá ilyet), minden szabály, amelynek jobb oldala egynél hosszabb csak változókat tartalmaz (erre szolgálnak az X_a-k), és nem lehet kettőnél hosszabb (hisz azokat széttördeltük). Tehát G' valóban a kívánt alakú nyelvtan. □

6.26. Feladat *Alakítsuk át a* $S \to \mathtt{a}S\mathtt{a} \mid \mathtt{b}S\mathtt{b} \mid \mathtt{c}$ *nyelvtant CNF formára!*

Megoldás: A nyelvtanban nincs egyszeres és $S \to \varepsilon$ szabály, ezért $G_2 = G$. Vezessük be az X_a változókat, ahol kell

$$\begin{array}{lcl} S & \to & X_a S X_a \mid X_b S X_b \mid \mathtt{c} \\ X_a & \to & \mathtt{a} \\ X_b & \to & \mathtt{b} \end{array}$$

A kettőnél hosszabb szabályok feldarabolásához itt két új változóra van szükségünk, legyenek ezek A és B. A kapott CNF alakú nyelvtan:

$$\begin{array}{lcl} S & \to & X_a A \mid X_b B \mid \mathtt{c} \\ X_a & \to & \mathtt{a} \\ X_b & \to & \mathtt{b} \\ A & \to & S X_a \\ B & \to & S X_b \end{array}$$

□

6.4.2. Greibach-féle normálforma

Amikor egy CF nyelvtanról el szeretnénk dönteni, hogy generál-e egy adott szót, akkor sokszor hasznos lesz, ha feltehetjük, hogy a nyelvtan nem tartalmaz úgy nevezett balrekurziót: semelyik A változóból sem lehet olyan szót előállítani, ami A-val kezdődik.

Most egy olyan normálformát mutatunk be, mely garantálja egy CF nyelvtan balrekurzió-mentességét.

6.27. Definíció *Egy CF nyelvtan* Greibach-normálformájú (GNF)*, ha a szabályai* $A \to a\alpha$ *alakúak, ahol* $a \in \Sigma$ *és* $\alpha \in V^*$.

A definíció azt mondja, hogy minden szabály jobb oldalának első karaktere terminális karakter.

Itt is igaz, hogy egy GNF nyelvtan nem tudja az üres szót generálni, de ettől eltekintve, ez is minden CF nyelvet elő tud állítani, Ehhez előbb megmutatjuk, hogy az $A \to A\alpha$ típusú szabályok (*közvetlen balrekurzió*) elkerülhetőek.

6.28. Lemma *A G CF nyelvtanban legyenek az* $A \in V$ *változóhoz tartozó szabályok*

$$A \to A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_k \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_m,$$

ahol egy β_j *sem kezdődik az* A *változóval és egyetlen* β_j *sem* ε*. Ha ezeket a szabályokat helyettesítjük az*

$$\begin{aligned} A &\to \beta_1 \mid \beta_2 \mid \cdots \mid \beta_m \\ A &\to \beta_1 B \mid \beta_2 B \mid \cdots \mid \beta_m B \\ B &\to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_k \\ B &\to \alpha_1 B \mid \alpha_2 B \mid \cdots \mid \alpha_k B \end{aligned}$$

szabályokkal, ahol B *egy új változó, akkor az így kapott* G' *nyelvtanra* $L(G') = L(G)$ *teljesül.*

Bizonyítás. Ha az eredeti nyelvtannál egy szó levezetésében megjelenik az A változó, akkor a bal-levezetésben néhány $A \to \alpha_j$ típusú szabályalkalmazást egy $A \to \beta_i$ követ. Egy ilyen $A \Rightarrow A\alpha_{j_1} \Rightarrow A\alpha_{j_2}\alpha_{j_1} \Rightarrow \ldots \beta_i\alpha_{j_t} \ldots \alpha_{j_2}\alpha_{j_1}$ lépéssorozat megfelel az új nyelvtanban az $A \Rightarrow \beta_i B \Rightarrow \beta_i\alpha_t B \Rightarrow \cdots \Rightarrow \beta_i\alpha_{j_t} \ldots \alpha_{j_2}\alpha_{j_1}$ jobb levezetésnek, és ez a megfeleltetés meg is fordítható. □

6.29. Tétel *Minden G CF nyelvtanhoz lehet készíteni olyan* G' *Greibach-normálformájú nyelvtant, melyre* $L(G') = L(G) - \{\varepsilon\}$.

Bizonyítás. Feltehetjük, hogy G-ben nincsenek egyszeres szabályok (lásd 6.3.2 fejezet), legyen $V = \{A_1, A_2, \ldots, A_n\}$, azaz a változók legyenek megszámozva 1-től n-ig.

Az átalakítás most is több lépésből áll. Először a Chomsky-normálformánál látott módon elérjük, hogy a jobb oldalakon a kezdőkarakter kivételével csak változók legyenek. Ehhez minden $a \in \Sigma$ betűhöz, amely valamelyik szabály jobb oldalának nem első szimbólumaként fordul elő felveszünk egy új X_a változót, és minden ilyen helyen ezt használjuk az a betű helyett, valamint bevezetjük az új $X_a \to a$ szabályokat.

Ezután a nem megfelelő formájú szabályok alakja $A_i \to A_j\beta$ és $\beta \in V^*$, a továbbiakban ezekkel foglalkozunk. Először elérjük, hogy minden ilyen fajta szabályban $j > i$ legyen. Az $i = 1$ esetben $j \geq 1$, vagyis ilyenkor csak a $j = 1$ eset lehet gond. Ekkor azonban a 6.28. lemma segítségével átalakíthatjuk a nyelvtant, ezután csak $j > 1$ típusú szabály fordul elő. Általában $i > 1$ esetén az $A_i \to A_j\beta$ szabályt, ha $j < i$, cseréljük le az $A_i \to \gamma_{jk}\beta$ szabályokra, amennyiben $A_j \to \gamma_{jk}$. Ezeket a lépéseket legfeljebb i-szer ismételve $A_i \to A_j\beta$ szabályokat kapunk, $j \geq i$. A $j = i$ esettől az előző, 6.28. lemma segítségével tudunk megszabadulni. Ha az új változókat sorban -1, -2, $\ldots$ indexekkel látjuk el, akkor az eljárás nem hoz létre újabb, nem megfelelő alakú szabályokat, az eljárás ezen része véget ér.

Most már a csak változókat tartalmazó jobboldalú szabályok mind $A_i \to A_j\beta$ alakúak $(j > i)$, ami azt jelenti, hogy A_n-re nincs ilyen szabály, az ő szabályai már megfelelnek a Greibach-féle normálforma követelményeinek. Így az $A_{n-1} \to A_n\beta$ szabályoknál A_n helyébe az A_n jobb oldalait helyettesítve az A_{n-1} szabályai is megfelelő formájúak lesznek. Ezt folytatva $(n-1, n-2, \ldots, 1, -1, -2, \ldots)$, a végén kapott G' nyelvtan Greibach-féle normálformájú lesz.

Az alkalmazott lépések az üres szó kivételével a levezethetőséget nem változtatják, ezért $L(G') = L(G) - \{\varepsilon\}$. □

6.30. Feladat *Alakítsuk át az*

$$\begin{aligned} A_1 &\to A_2\mathtt{a} \\ A_2 &\to A_3A_1 \mid \mathtt{b} \\ A_3 &\to A_1A_2 \mid \mathtt{a} \end{aligned}$$

nyelvtant Greibach-normálformájúvá!

Megoldás: Első lépésként az első szabály miatt kell egy új változó

$$\begin{aligned} A_1 &\to A_2X_a \\ A_2 &\to A_3A_1 \mid \mathtt{b} \\ A_3 &\to A_1A_2 \mid \mathtt{a} \\ X_a &\to \mathtt{a} \end{aligned}$$

Az $A_3 \to A_1A_2$ szabályt átalakítjuk először $A_3 \to A_2X_aA_2$ szabállyá, majd tovább folytatva $A_3 \to A_3A_1X_aA_2 \mid \mathtt{b}X_aA_2 \mid \mathtt{b}$, majd a 6.28. lemma alapján $A_3 \to \mathtt{b}X_aA_2 \mid \mathtt{b} \mid \mathtt{b}X_aA_2A_{-1} \mid \mathtt{b}a_{-1}$ és $A_{-1} \to A_1X_aA_2 \mid A_1X_aA_2A_{-1}$. Az A_2, majd az A_1 szabályaiba helyettesítve kapjuk a teljes nyelvtant:

$$
\begin{aligned}
A_1 &\to \mathtt{b}X_aA_2A_1X_a \mid \mathtt{b}A_1X_a \mid \mathtt{b}X_aA_2A_{-1}A_1X_a \mid \mathtt{b}A_{-1}A_1X_a \mid \mathtt{b}X_a \\
A_2 &\to \mathtt{b}X_aA_2A_1 \mid \mathtt{b}A_1 \mid \mathtt{b}X_aA_2A_{-1}A_1 \mid \mathtt{b}A_{-1}A_1 \mid \mathtt{b} \\
A_3 &\to \mathtt{b}X_aA_2 \mid \mathtt{b} \mid \mathtt{b}X_aA_2A_{-1} \mid \mathtt{b}A_{-1} \\
A_{-1} &\to \mathtt{b}X_aA_2A_1X_aX_aA_2 \mid \mathtt{b}A_1X_aX_aA_2 \mid \mathtt{b}X_aA_2A_{-1}A_1X_aX_aA_2 \mid \mathtt{b}A_{-1}A_1X_aX_aA_2 \mid \\
&\quad \mathtt{b}X_aX_aA_2 \mid \mathtt{b}X_aA_2A_1X_aX_aA_2A_{-1} \mid \mathtt{b}A_1X_aX_aA_2A_{-1} \mid \\
&\quad \mathtt{b}X_aA_2A_{-1}A_1X_aX_aA_2A_{-1} \mid \mathtt{b}A_{-1}A_1X_aX_aA_2A_{-1} \mid \mathtt{b}X_aX_aA_2A_{-1} \\
X_a &\to \mathtt{a}
\end{aligned}
$$

□

6.5. CF nyelvek tulajdonságai

A reguláris nyelvek osztályáról már láttuk, hogy zárt az alapvető műveletekre. Most a környezetfüggetlen nyelvek osztályát is megvizsgáljuk ebből a szempontból.

6.31. Tétel *Ha L_1 és L_2 CF nyelvek, akkor*

- $L_1 \cup L_2$
- L_1L_2
- L_1^*

is CF nyelv.

Bizonyítás. Az L_i nyelvhez van $G_i = (V_i, \Sigma, S_i, P_i)$ környezetfüggetlen nyelvtan, azaz $L_1 = L(G_1)$ és $L_2 = L(G_2)$. Feltehetjük, hogy $V_1 \cap V_2 = \emptyset$. (Ha nem így van, akkor nevezzük át a változókat.) Legyen S egy új változó ($S \notin V_1$ és $S \notin V_2$), a G nyelvtan változói legyenek G_1 és G_2 változói, kiegészítve az S kezdőváltozóval, azaz $V = V_1 \cup V_2 \cup \{S\}$.

unió Legyen $P = P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}$. Ekkor a $G = (V, \Sigma, S, P)$ nyelvtanra teljesül, hogy $L(G) = L(G_1) \cup L(G_2)$. A G levezetési szabályai $A \to \alpha$ alakúak. Ha a nyelvtanokban nincs $S_i \to \varepsilon$ szabály, akkor G egy CF nyelvtan, ellenkező esetben még kell egy ε-mentesítés (lásd a 6.3.1 részt), hogy CF nyelvtant kapjunk.

összefűzés Legyen $P = P_1 \cup P_2 \cup \{S \to S_1 S_2\}$. Világos, hogy ekkor a $G = (V, \Sigma, S, P)$ nyelvtanra $L(G) = L(G_1)L(G_2)$ teljesül. Itt is szükség lehet még ε-mentesítésre, hogy ebből CF nyelvtant kapjunk.

tranzitív lezárt Most legyen $P = P_1 \cup \{S \to S_1 S, S \to \varepsilon\}$. Ezzel elérjük, hogy a $G = (V, \Sigma, S, P)$ nyelvtanból levezethető szavak egy vagy több $L(G_1)$-beli szó összefűzései, vagy az üres szó, ami épp az $L(G_1)^*$ nyelv. Ahhoz, hogy CF nyelvtanunk legyen, itt is szükséges lehet az ε-mentesítés. □

A következő lemma a reguláris nyelvek pumpálhatóságához hasonló állítást fogalmaz meg CF nyelvekre.

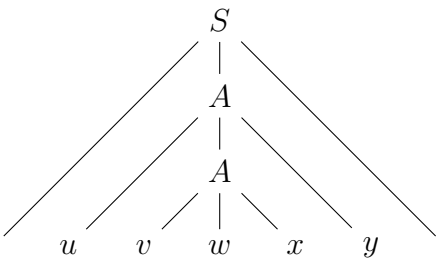
6.32. Lemma (Pumpálási lemma CF nyelvekre) *Tetszőleges L CF nyelvhez létezik olyan $p > 0$ egész szám (pumpálási hossz), hogy minden legalább p hosszú $z \in L$ szónak van $z = uvwxy$ felosztása, melyre*

- $|vwx| \leq p$
- $|vx| \geq 1$
- $uv^k wx^k y \in L$ *minden* $k \geq 0$ *esetén.*

Így tehát a reguláris nyelvekkel ellentétben (3.11. lemma) a környezetfüggetlen nyelvek elég hosszú szavaiban két rész „párhuzamosan” pumpálható.

Bizonyítás. Tekintsünk L-hez egy olyan környezetfüggetlen nyelvtant, amelyben nincsenek egyszeres szabályok. (A 6.18. tétel értelmében ilyen van.) Legyen $m = |V|$ a nyelvtan változóinak száma és legyen d a levezetési szabályok jobb oldalain álló kifejezések hosszának maximuma. Megmutatjuk, hogy a $p = d^{m+1}$ választás jó.

Tekintsünk egy $z \in L$ tetszőleges, legalább p hosszú szót és vegyük szemügyre ennek egy levezetési fáját (lásd a 6.1 fejezet). Ebben a fában minden csúcsnak legfeljebb d fia lehet, a levelek száma pedig nyilván $|z|$. Emiatt biztosan van olyan út a gyökér és valamelyik levél között, amelyik a gyökérrel és a levéllel együtt legalább $m+2$ csúcsot tartalmaz (különben a szó hossza legfeljebb d^m lehetne). Az úton a levélen kívül minden csúcshoz egy változó tartozik, ezért biztosan van két csúcs ugyanazzal a változóval. Legyen A egy olyan ismétlődő változó, amelyiknél az ismétlődés a gyökértől a legtávolabb van. Osszuk fel a z szót a levezetési fa alapján az alábbi módon:



A fához ezek szerint az $S \Rightarrow \ldots \Rightarrow uAy \Rightarrow \ldots \Rightarrow uvAxy \Rightarrow \ldots \Rightarrow uvwxy$ levezetés tartozhat.

Ekkor az A két előfordulása közötti részt akárhányszor megismételve továbbra is az L nyelv szabályai alapján érvényes levezetési fát kapunk, így $uw^kwx^ky \in L$.

Mivel nincs a nyelvben egyszeres szabály, v és x együttesen nem lehet üres. (Az A két előfordulása közötti részben biztosan generáltunk legalább egy levelet.)

Az is elmondható, hogy $|vwx| \leq p$, mert a legmélyebben lévő ismétlődést választottuk, így a felső A alatti változók száma minden úton legfeljebb m, így a levelek száma legfeljebb $d^{m+1} = p$. □

6.33. Állítás *Az* $L = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n : n \geq 0\}$ *nyelv nem környezetfüggetlen nyelv.*

Bizonyítás. Az állítás következik a pumpálási lemmából. Tegyük fel ugyanis, hogy L környezetfüggetlen és alkalmazzuk rá a pumplálási lemmát, a pumpálási hosszt jelöljük p-vel.

Válasszuk a $z = \overbrace{\mathtt{a}\ldots\mathtt{a}}^{p}\overbrace{\mathtt{b}\ldots\mathtt{b}}^{p}\overbrace{\mathtt{c}\ldots\mathtt{c}}^{p} = \mathtt{a}^p\mathtt{b}^p\mathtt{c}^p$ szót. Ennek a pumpálási lemma szerinti felosztásában, vwx legfeljebb kétféle betűből állhat, mert $|vwx| \leq p$.

Ez alapján v és x együttesen legfeljebb kétféle betűt tartalmaz, így ha ezeket többször vesszük, akkor a kapott szóban biztosan nem lesz mindhárom fajta betűből ugyanannyi, tehát a kapott szó nem lehet eleme az L nyelvnek. Ellentmondásra jutottunk, tehát L nem CF nyelv. □

6.34. Tétel *A CF nyelvek halmaza nem zárt a metszet műveletre, azaz van olyan* L_1, L_2 *környezetfüggetlen nyelv, hogy* $L_1 \cap L_2$ *nem környezetfüggetlen.*

Bizonyítás. Tekintsük a következő nyelveket

$$L_1 = \{a^nb^nc^m : n, m \geq 0\}$$
$$L_2 = \{a^mb^nc^n : n, m \geq 0\}$$
$$L = \{a^nb^nc^n : n \geq 0\}$$

Ezekre teljesül, hogy $L = L_1 \cap L_2$, és az előző állításból tudjuk, hogy az L nyelv nem CF. Már csak azt kell megmutatni, L_1 és L_2 viszont CF.

Az L_1 nyelv felfogható, mint $L_{ab}L_c$. Az $L_{ab} = \{a^nb^n : n \geq 0\}$ nyelvről már a 4.8. példában láttuk, hogy CF, az $L_c = \{c^m : m \geq 0\}$ nyelv könnyen láthatóan CF (sőt reguláris is). Ezért L_1 is CF.

Hasonlóan $L_2 = L_aL_{bc}$, ahol $L_a = \{a^m : m \geq 0\}$ és $L_{bc} = \{b^nc^n : n \geq 0\}$ az előzőek mintájára egyaránt CF. □

6.35. Következmény *Van olyan* L *környezetfüggetlen nyelv, hogy* $\overline{L}$ *nem környezetfüggetlen.*

Bizonyítás. Mivel a metszetet megkaphatjuk unió- és komplementerképzés segítségével ($\overline{\overline{L_1} \cup \overline{L_2}} = L_1 \cap L_2$), ezért ha a CF nyelvek halmaza zárt lenne a komplementerre, akkor 6.31. tétel miatt a metszetre is. □

6.36. Feladat *Mutassuk meg, hogy az* $L = \{ww \mid w \in \{a, b\}^*\}$ *nyelv nem környezetfüggetlen!*

Megoldás: Indirekt módon tegyük fel, hogy L egy CF nyelv és p a hozzá tartozó pumpálási hossz. Tekintsük a $z = a^p b^p a^p b^p$ szót, amely nyilván benne van a nyelvben és a hossza is megfelelő, ezért pumpálható. Mivel $|vwx| \leq p$ ezért vwx legfeljebb két egymást követő p hosszú blokkba lóghat bele. Ha v-t és x-et pumpáljuk, akkor az előzőek miatt vagy egyetlen blokkot tudunk növelni, vagy két egymás mellettit. Így biztos marad két blokk, melyeknek a hossza nem változik, tehát a kapott szó nem lehet a nyelvben. Ezzel ellentmondásra jutottunk, tehát L nem CF. □

A pumpálási lemma használatában nehézséget jelenthet, hogy nem tudjuk befolyásolni, hol helyezkednek el a szóban a pumpált szakaszok, csak annyit tudunk, hogy valahol léteznie kell. Ezen segít részben a következő általánosítás.

6.37. Lemma (Ogden-lemma) *Minden* L *környezetfüggetlen nyelvhez van olyan* $p > 0$ *egész szám, hogy minden legalább* p *hosszú* $z \in L$ *szó esetén, ha tetszőlegesen kijelölünk legalább* p *darab pozíciót (karaktert)* z*-ben, akkor létezik* z*-nek egy olyan* $z = uvwxy$ *felbontása, hogy*

- vwx *legfeljebb* p *darab kijelölt pozíciót tartalmaz,*
- v *és* x *együttesen legalább* 1 *kijelölt pozíciót tartalmaz,*
- *minden* $k \geq 0$ *esetén* $uv^k wx^k y \in L$

6.38. Megjegyzés *Az Ogden lemma speciális eseteként megkapható az eredeti pumpálási lemma úgy, hogy minden karaktert kijelölünk.*

A lemma bizonyítása a pumpálási lemmáéhoz hasonlóan történhet. Az alapötlet az, hogy egy levezetési fában legyen kijelölt az a belső csúcs, amelynek részfájában van legalább két kijelölt levél. Ekkor, ha a fának elég sok levele van, akkor található benne olyan gyökértől levélig menő út, aminek van ugyanahhoz a változóhoz tartozó két kijelölt csúcsa.

Nézzük, hogyan lehet alkalmazni az Ogden-lemmát arra, hogy megmutassuk egy nyelvről, hogy biztosan nem környezetfüggetlen.

6.39. Feladat *Az Ogden-lemma segítségével mutassuk meg, hogy az* $L = \{\mathtt{a}^i\mathtt{b}^i\mathtt{c}^j : i, j \geq 1, i \neq j\}$ *nyelv nem környezetfüggetlen!*

Megoldás: Indirekt módon tegyük fel, hogy a nyelv környezetfüggetlen és legyen az Ogden-lemma szerinti pumpálási hossz p. Tekintsük az $\mathtt{a}^p\mathtt{b}^p\mathtt{c}^{p+p!}$ szót, amely nyilván eleme a nyelvnek és megfelelő hosszúságú. Jelöljük ki az első p pozíciót, azaz a szó elején található $\mathtt{a}$ karaktereket. A lemma alapján v és x együttesen tartalmaz legalább egy $\mathtt{a}$ karaktert. Könnyen látható, hogy ekkor v biztosan csupa egyforma betűből áll, és ez igaz x-re is, különben a pumpálással kikerülnénk a nyelvből. Az előzőek alapján tehát valamelyikük csupa $\mathtt{a}$ karakterből áll. De ha az $\mathtt{a}$ karakterek számát változtatjuk, akkor a $\mathtt{b}$ karakterekét is változtatni kell, ha a nyelvben akarunk maradni, ezért az egyetlen lehetőség, hogy $v = \mathtt{a}^t$ és $x = \mathtt{b}^t$ valamilyen $t > 0$ számra. Ekkor persze $t \leq p$.

A pumpálással kapott szó $uv^kwx^ky = \mathtt{a}^{p+(k-1)t}\mathtt{b}^{p+(k-1)t}\mathtt{c}^{p+p!}$. Ha most $k = 1 + \frac{p!}{t}$ (ami egy pozitív egész, mert $t \leq p$), akkor az $\mathtt{a}$ és a $\mathtt{b}$ karakterek száma $p + (k-1)t = p + p!$, ami megegyezik a $\mathtt{c}$ karakterek számával, tehát a kapott szó nincs az L nyelvben. Ezzel ellentmondásra jutottunk, tehát L valóban nem CF. □

Az Ogden-lemma segítséget nyújt nem csak annak bizonyításában, hogy egy nyelv nem CF, hanem a már korábban említett (6.9. tétel) állítás bizonyításához is megfelelő eszköz.

6.40. Tétel *Az $L = \{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k : i = j$ vagy $j = k\}$ nyelv nem egyértelmű, azaz nincs olyan CF nyelvtana, amelyben minden szó levezetési fája egyértelmű.*

Bizonyítás. A bizonyítás vázlatát közöljük. Mivel $L = \{a^ib^ic^k\} \cup \{a^ib^kc^k\}$ két környezetfüggetlen nyelv uniója, ezért L egy CF nyelv és így igaz rá az Ogden-lemma. Legyen p a lemma által adott pumpálási hossz. Tekintsük a $z = a^pb^pc^{p+p!}$ szót és jelöljük ki benne az első p pozíciót. Az előző feladat megoldásában leírtak miatt z a lemmának megfelelő felbontásáról tudjuk, hogy $v = a^k$ és $x = b^k$. Jelen esetben tudjuk, hogy ilyenkor x és v valóban pumpálható lesz. Ez azt jelenti, hogy létezik olyan A változó, hogy A-ból levezethető az a^kAb^k sorozat. Tekintsük most az $s = a^{p+p!}b^{p+p!}c^{p+p!}$ szót. Világos, hogy $s \in L$, továbbá az s szót le tudjuk vezetni úgy, hogy a z szó levezetését az $A \Rightarrow \cdots \Rightarrow a^kAb^k$ levezetéssel megfelelően sokszor kiegészítjük. Hasonlóképpen elmondhatjuk (az Ogden lemmát alkalmazva az utolsó p karakter kijelölésével), hogy a $q = a^{p+p!}b^pc^p$ szó levezetése közben találhatunk egy olyan B változót amelyből levezethető egy b^tBc^t sorozat, így tehát s-et le tudjuk vezetni úgy is, hogy q levezetése közben az iménti részlevezetést alkalmazzuk megfelelően sokszor. Láttuk tehát, hogy az s szóra legalább két levezetés is található, amelyek levezetési fája is eltérő. Ez mutatja, hogy a nyelv nem egyértelmű. □

7. fejezet

Veremautomaták

A veremautomata a véges automatának egy olyan kiterjesztése, amikor az állapotokon kívül az automata, egy vermet (LIFO tulajdonságú tárolót) is használ memóriaként. Ez az automata is nyelvek felismerésére szolgál és mint látni fogjuk, erősebb eszköz, mint a korábban tárgyalt véges automata.

7.1. Veremautomata-típusok

A későbbi jelölések egyszerűsítésére, ha Σ egy ábécé, akkor vezessük be a

$$\Sigma_0 = \Sigma \cup \{\varepsilon\}$$

jelölést.

7.1. Definíció *A* veremautomata *egy* $M = (Q, \Sigma, \Gamma, q_0, Z_0, F, \delta)$ *hetessel írható le, ahol*

- Q *egy véges, nem üres halmaz. Ez az automata állapotainak halmaza.*
- Σ *egy véges, nem üres halmaz. Ez az automata (bemeneti) ábécéje.*
- Γ *egy véges, nem üres halmaz. Ez a verem ábécéje vagyis a lehetséges veremszimbólumok halmaza.*
- $q_0 \in Q$ *a kezdőállapot.*
- $Z_0 \in \Gamma$ *a verem kezdőszimbóluma.*
- $F \subseteq Q$ *az elfogadó állapotok halmaza.*
- δ *az állapotátmeneti függvény,* $\delta(q, a, A) \subseteq Q \times \Gamma^*$*, ahol* $q \in Q$*,* $a \in \Sigma_0$ *és* $A \in \Gamma_0$*.*

Az, hogy az állapotátmeneti függvény értéke egy halmaz az azt jelenti, hogy az itt definiált veremautomata nemdeterminisztikus. Hiányos is lehet, amennyiben $\delta(q, a, A) = \emptyset$ valamely (q, a, A) helyen.

A veremautomata működése a következőképpen írható le egy adott $w \in \Sigma^*$ bemeneten

7.2. Definíció *Legyen $w = a_1 a_2 \dots a_m$, ahol minden $a_i \in \Sigma_0$. Az M veremautomata* számítása *a w szón egy olyan $r_0, r_1, \dots, r_m \in Q$ állapotsorozat és $s_0, s_1, \dots, s_m \in \Gamma^*$ sorozata a veremtartalmaknak, melyekre teljesül, hogy*

- *$r_0 = q_0$ és $s_0 = Z_0$,*
- *minden $1 \leq i \leq m$ esetén, ha $s_{i-1} = A_{i-1}\tau_{i-1}$, és $A_{i-1} \in \Gamma_0$, valamint $\tau_{i-1} \in \Gamma^*$, akkor*
  - *$(r_i, t_i) \in \delta(r_{i-1}, a_i, A_{i-1})$ és*
  - *$s_i = t_i \tau_{i-1}$*

Ez azt jelenti, hogy kezdéskor a veremautomata a q_0 állapotban van, a vermében csak egyetlen Z_0 szimbólum található. Az i-edik lépésben r_i állapotban a bemenetről 1 vagy 0 karaktert olvas ($a_i \in \Sigma_0$), a verem tetejéről levesz 1 vagy 0 szimbólumot ($A_{i-1} \in \Gamma_0$), és az átmeneti függvény $\delta(r_{i-1}, a_i, A_{i-1})$ által megadott lehetőségeinek egyike alapján változik az állapota (az új állapot r_i) és 0, 1 vagy több karaktert ír a verem tetejére (A_{i-1} helyére a $t_i \in \Gamma^*$ szimbólumsorozat kerül). Ha az átmeneti függvény értéke valamelyik közbeeső lépésben az üres halmaz, akkor ott a számítás elakad.

Mivel a verembe írt karaktersorozat lehet az üres is, $\delta(q, a, A) = (q', \varepsilon)$ típusú átmenetekkel a verem kiüríthető. De vigyázni kell, hogy ne akarjunk a kiürült veremből $B \in \Gamma$ szimbólumot kiolvasni – ez hibát okoz (nincs automatikus teszt arra, hogy a verem üres-e).

Ha a veremautomata *konfigurációjára* bevezetjük a (q, x, s) jelölést, ahol q az aktuális állapot, x a bemenetből még hátra levő rész és s a verem tartalma, akkor az i-edik lépés leírható

$$(r_{i-1}, a_i a_{i+1} \dots a_m, A_{i-1}\tau_{i-1}) \Rightarrow (r_i, a_{i+1} \dots a_m, t_i \tau_{i-1})$$

alakban is (itt a_i lehet ε, amikor nem történik tényleges továbblépés a bemeneten, és ha $A_{i-1} = \varepsilon$, akkor a veremből nem veszünk ki semmit, legfeljebb írunk bele.)

7.3. Definíció *Azt mondjuk, hogy az M veremautomata* elfogadja *a $w \in \Sigma^*$ szót, ha van olyan számítása, hogy $r_m \in F$ teljesül.*

Az elfogadáshoz hozzátartozik, hogy a w szó feldolgozása közben a számítás nem akad el, azaz akkor van vége, ha a veremautomata az utolsó betűt is feldolgozta (és esetleg még néhány $\delta(q, \varepsilon, A)$ átmenetet is végzett).

A konfigurációk segítségével az elfogadás úgy írható le, hogy a kezdeti helyzetből véges sok lépésben eljuthatunk egy elfogadóba:

$$(q_0, w, Z_0) \Rightarrow \cdots \Rightarrow (q, \varepsilon, s) \text{ ahol } q \in F, s \in \Gamma^* \text{ tetszőleges}$$

7.4. Definíció *Az M veremautomata által* elfogadott nyelv *azoknak a Σ feletti szavaknak a halmaza, melyeket M elfogad. Jele $L(M)$.*

7.5. Példa *A következő automata az $L = \{\mathtt{0}^n\mathtt{1}^n : n \geq 0\}$ nyelvet fogadja el. Az elv elég egyszerű, amíg* `0` *karaktert olvasunk, berakunk egy-egy szimbólumot a verembe, az* `1` *karaktereknél pedig mindig kiveszünk egyet, az állapot jelzi melyik fázisban vagyunk.*

Legyen $\Gamma = \{Z, 0\}$, $Q = \{q_0, q_1, q_2, q_3\}$ és $M = (Q, \{\mathtt{0}, \mathtt{1}\}, \Gamma, q_0, Z, \{q_0, q_3\}, \delta)$, ahol

$$\begin{aligned}
\delta : (q_0, \mathtt{0}, Z) &\mapsto (q_1, 0Z) \\
(q_1, \mathtt{0}, \varepsilon) &\mapsto (q_1, 0) \\
(q_1, \mathtt{1}, 0) &\mapsto (q_2, \varepsilon) \\
(q_2, \mathtt{1}, 0) &\mapsto (q_2, \varepsilon) \\
(q_2, \varepsilon, Z) &\mapsto (q_3, Z)
\end{aligned}$$

a többi átmenet esetén δ értéke az üres halmaz.

Nézzük meg részletesen, miért is jó ez a veremautomata!

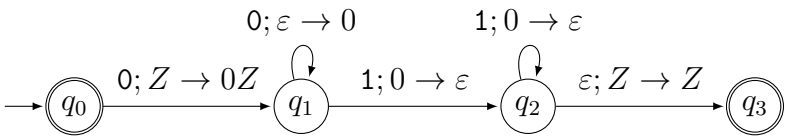
Látszik, hogy egy $\mathtt{0}^n\mathtt{1}^n$ alakú szón, ha $n \geq 1$, az utolsó karakter elolvasásakor a q_2 állapotban lesz, ugyanakkor a veremben csak a Z szimbólum marad, tehát az utolsó átmenetet használva eljut a q_3 elfogadó állapotba. Az $n = 0$ esetben a nulla hosszú számítás elfogadó, mert $q_0 \in F$.

Egy 0^n1^k alakú szó esetén, amikor $n > k$, akkor a q_2 állapotban nem lehet az összes 0 karaktert törölni a veremből, nincs mód átkerülni a q_3 állapotba. Ha viszont $n < k$, akkor, mivel a veremből a 0 szimbólumok elfogynak, a szó vége előtt átkerülünk a q_3 állapotba, ahol a számítás elakad, és ezért nem lesz elfogadó.

Ha a bemeneti szó első karaktere `1`*, akkor a veremautomata nem tud lépni, elakad anélkül, hogy a szó végére jutna, tehát nem elfogadó.*

Ha ugyan `0` *karakterrel kezdődik, de néhány* `1` *karakter után megint* `0` *jön, akkor q_2 (vagy q_3) állapotban találkozik a veremautomata* `0` *karakterrel, amikor is a számítása elakad.*

A veremautomatákat is lehet gráffal ábrázolni, a nyilakon, a szó megfelelő karakterétől vesszővel elválasztva jelezzük a veremben történő változást. Az előző példa ebben a formában



Természetesen lehet a determinisztikus változatot is definiálni, amikor minden helyzetben legfeljebb egy lépése lehetséges az automatának. (Pontosabban a véges automatánál használt terminológia szerint ez egy hiányos determinisztikus veremautomata lesz.)

7.6. Definíció *A* determinisztikus veremautomata *egy olyan* $M = (Q, \Sigma, \Gamma, q_0, Z_0, F, \delta)$ *veremautomata, melyre*

- *minden* $q \in Q$, $a \in \Sigma_0$ *és* $A \in \Gamma_0$ *esetén* $|\delta(q, a, A)| \leq 1$;
- *ha egy* $q \in Q$ *állapotra* $\delta(q, \varepsilon, \varepsilon) \neq \emptyset$, *akkor minden* $a \in \Sigma_0$ *és* $A \in \Gamma_0$ *esetén* $\delta(q, a, A) = \emptyset$, *amennyiben* a *és* A *nem egyszerre* ε;
- *ha egy* $q \in Q$ *és* $A \in \Gamma$ *esetén* $\delta(q, \varepsilon, A) \neq \emptyset$, *akkor minden* $a \in \Sigma$ *betűre* $\delta(q, a, A) = \emptyset$;
- *ha egy* $q \in Q$ *és* $a \in \Sigma$ *esetén* $\delta(q, a, \varepsilon) \neq \emptyset$, *akkor minden* $A \in \Gamma$ *szimbólumra* $\delta(q, a, A) = \emptyset$.

A definícióban az első utáni feltételek azt biztosítják, hogy a veremautomata egy tetszőleges helyzetében az is egyértelmű legyen, kell-e a bemeneten tovább haladni ($a \neq \varepsilon$?), illetve kell-e a verem tetejéről olvasni ($A \neq \varepsilon$?)

Vegyük észre, hogy az előző példa veremautomatája egy determinisztikus veremautomata.

A következő eredmény indokolja, hogy veremautomatának a nemdeterminisztikus változatot hívjuk.

7.7. Tétel *Van olyan* L *nyelv, amelyhez létezik* M *veremautomata, hogy* $L = L(M)$, *de nem létezik ilyen determinisztikus veremautomata.*

A palindromok nyelve például ilyen, de ezt itt nem bizonyítjuk.

A veremautomatának egy másik, sokszor kényelmesebben használható változata az olyan veremautomata, ahol nincsenek elfogadó állapotok, az elfogadás a verem tartalmán múlik.

7.8. Definíció *Az* üres veremmel elfogadó veremautomata *egy* $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$ *hatossal írható le, amiben a paraméterek ugyanazok, mint a veremautomatánál (kivéve, hogy nincsenek elfogadó állapotok).*

Ez az M *veremautomata akkor* fogad el *egy szót, ha van olyan számítása, melynek végén a veremben egyetlen szimbólum sem marad.*

A szó végigolvasása itt is feltétele az elfogadásnak, tehát konfigurációkkal leírva az elfogadás feltétele az, hogy

$$(q_0, w, Z_0) \Rightarrow \cdots \Rightarrow (q, \varepsilon, \varepsilon) \text{ ahol } q \in Q \text{ tetszőleges}$$

7.9. Tétel *Az L nyelvhez akkor és csak akkor létezik az L nyelvet elfogadó (állapottal elfogadó) M veremautomata, ha van az L nyelvet elfogadó M' üres veremmel elfogadó veremautomata.*

Bizonyítás. Legyen először $M = (Q, \Sigma, \Gamma, q_0, Z_0, F, \delta)$ egy olyan veremautomata, melyre $L(M) = L$. Hozzunk létre ebből egy, $M' = (Q', \Sigma, \Gamma', q_0', Z_0, \delta')$ üres veremmel elfogadó veremautomatát, ami az L-et fogadja el. Vegyünk fel két új állapotot, $q_0' \notin Q$ lesz az új kezdőállapot, $q_\varepsilon \notin Q$ pedig egy olyan állapot, amiben ki tudjuk üríteni a vermet. Felveszünk továbbá egy új $Z \notin \Gamma$ veremszimbólumot is, ez arra lesz jó, hogy ha M verme „véletlenül" kiürül egy nem elfogadó számítás végén, akkor M' verme ne legyen üres és így ne fogadja el (hibásan) az inputot. Ezek után legyen

- $Q' = Q \cup \{q_0', q_\varepsilon\}$
- $\Gamma' = \Gamma \cup \{Z\}$
- a kezdőállapot a q_0', a kezdőszimbólum a Z_0
- δ' a δ kiterjesztése az alábbi új állapotátmenetekkel:
  - $\delta'(q_0', \varepsilon, Z_0) = (q_0, Z_0 Z)$
  - $\delta'(q, \varepsilon, A) = (q_\varepsilon, \varepsilon)$, minden $q \in F$ elfogadó állapotra és $A \in \Gamma'$ szimbólumra
  - $\delta'(q_\varepsilon, \varepsilon, A) = (q_\varepsilon, \varepsilon)$, minden $A \in \Gamma'$ szimbólumra

A δ' első sorával betesszük Z-t a verem kezdőszimbóluma alá, a második sorral átmegyünk a veremkiürítő állapotba, a harmadikkal pedig teljesen kiürítjük a vermet. Látható, hogy $L(M) \subseteq L(M')$, mert M egy elfogadó számítása kiegészíthető M' elfogadó számításává a következőképpen: az állapotsorozat elejére kell egy q_0', végére annyi q_ε, ahány szimbólum a számítás végén a veremben maradt. A veremtartalom-sorozat elejére $Z_0 Z$-t kell tenni, a végén pedig egyesével elfogyasztani a veremtartalmat, amíg az utolsó szimbólumot is ki nem szedjük.

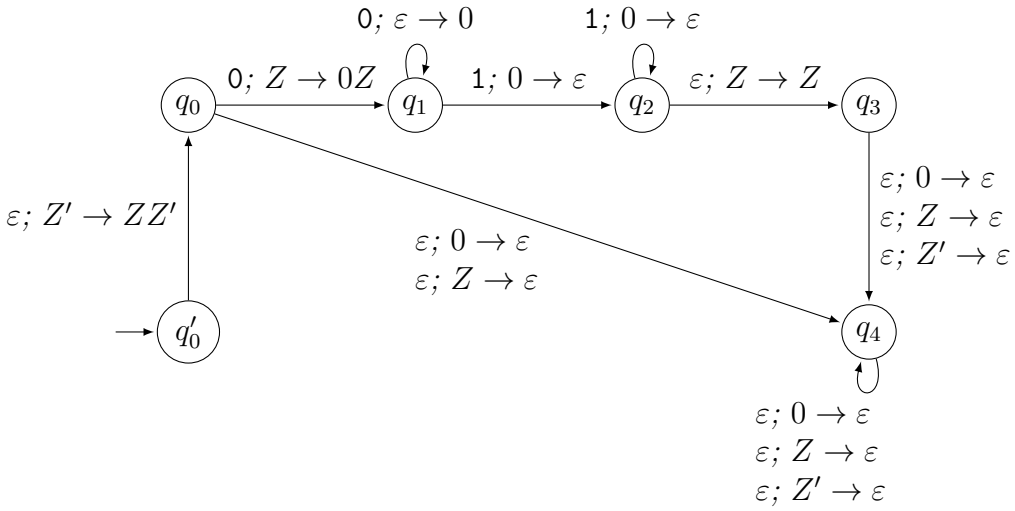
Az $L(M') \subseteq L(M)$ tartalmazás is egyszerűen látszik. Az M' elfogadó számításának ki kell venni a verem aljáról a nyitó lépésként odarakott Z szimbólumot. Ez nem lehetséges addig, amíg M lépéseit követjük (Z nincs M veremszimbólumai között), csak akkor ha M' már a q_ε állapotban van, ide viszont csak az eredeti automata elfogadó állapotából juthatunk.

A tétel másik irányához most tegyük fel, hogy $M' = (Q', \Sigma, \Gamma', q_0', Z_0', \delta')$ egy üres veremmel elfogadó veremautomata, melyre $L(M') = L$. Hozzunk létre ebből egy L-et elfogadó $M = (Q, \Sigma, \Gamma, q_0, Z, F, \delta)$ veremautomatát. A konstrukció gondolata az, hogy amikor M' kiüríti a vermét, akkor M kerüljön elfogadó állapotba. Mivel nincs arra beépített módszer, hogy a verem ürességét ellenőrizzük, ezért most is szükség van egy új $Z \notin \Gamma'$ szimbólumra, ami a verem alját jelzi M-nek. Ismét bevezetünk két új állapotot, $q_0 \notin Q'$ lesz az új kezdőállapot, $q_F \notin Q'$ pedig az egyetlen az elfogadó állapot.

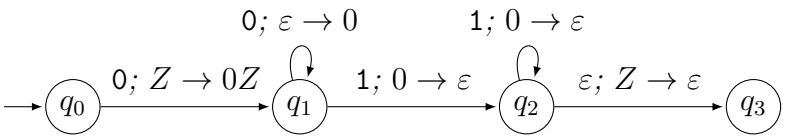
- $Q = Q' \cup \{q_0, q_F\}$
- $\Gamma = \Gamma \cup \{Z\}$
- a kezdőállapot q_0', az elfogadó állapotok halmaza $F = \{q_F\}$
- δ legyen δ' kiterjesztése az alábbi új állapotátmenetekkel:
  - $\delta(q_0, \varepsilon, Z_0{}') = (q_0', Z_0'Z)$
  - $\delta(q, \varepsilon, Z) = (q_F, Z)$ minden $q \in Q$ esetén

A konstrukció helyessége azon múlik, hogy amikor M' verme üres lesz, akkor M vermében csak a Z szimbólum marad, és ekkor az aktuális M'-beli állapottól függetlenül M átmegy az elfogadó állapotba, ahol a számítása véget ér. Másrészt M csak így tud elfogadni valamit, azaz szükséges, hogy M' kiürítse a vermét. □

7.10. Példa *A 7.5. példa veremautomatájából keletkező üres veremmel elfogadó veremautomata:*



Látszik, hogy ebben az esetben a kapott automatát egyszerűsíthetjük, ha q_3 állapotot kihagyjuk, és q_2-ből a Z törlésével egyből a q_4 állapotba lépünk. További megfontolásokkal az alábbi egyszerűsített változat ugyanerre a nyelvre:



7.11. Megjegyzés *A veremautomata fogalmát kiterjeszthetjük úgy, hogy egy lépésben ne csak egy szimbólumot olvashasson a veremből, hanem egy előre definiált mélységig belelásson a verembe. Megmutatható, hogy minden ilyen veremautomatához megadható egy olyan, az eredeti definíciónk szerinti veremautomata, ami ugyanazt a nyelvet fogadja el.*

7.2. Kapcsolat a környezetfüggetlen nyelvekkel

Ahogy a véges automaták pontosan a reguláris nyelvek elfogadására alkalmasak, a veremautomaták a következő, 2. Chomsky-féle nyelvosztályhoz kapcsolódnak. Ennek bizonyítása azonban lényegesen összetettebb, mint a korábbi esetben. Előbb azt mutatjuk meg (7.12. tétel), hogyan lehet egy, a 2. Chomsky-féle osztályba tartozó (*környezetfüggetlen*) nyelvtanból veremautomatát készíteni. Itt a természetes ötletet, hogy a levezetéseket a verem segítségével kövessük, nem nehéz megoldani. A másik irány bonyolultabb lesz (7.16. tétel).

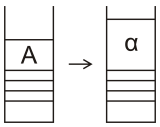
7.12. Tétel *Minden L környezetfüggetlen nyelvhez van olyan M veremautomata, melyre $L(M) = L$.*

Bizonyítás. Legyen $G = (V, \Sigma, S, P)$ egy 2. osztálybeli nyelvtan ami generálja az L nyelvet. A 7.9. tétel értelmében elegendő egy $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$ üres veremmel elfogadó veremautomatát megadni az L nyelvhez.

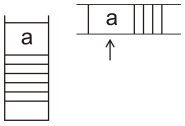
Legyen

- $Q = \{q\}$, azaz csak egy állapot lesz
- $\Gamma = V \cup \Sigma \cup \{Z_0, Z_1\}$, ahol Z_0, Z_1 két új szimbólum
- $q_0 = q$
- $\delta(q, \varepsilon, Z_0) = (q, SZ_1)$
- $\delta(q, \varepsilon, A) = \{(q, \alpha) : (A \to \alpha) \in P\}$ minden $A \in V$ változóra
- $\delta(q, a, a) = (q, \varepsilon)$ minden $a \in \Sigma$ karakterre
- $\delta(q, \varepsilon, Z_1) = \{(q, \varepsilon)\}$

Tehát amikor változó van a verem tetején, akkor ezt helyettesíteni lehet tetszőleges hozzá tartozó szabály jobb oldalával, sematikusan



Amikor pedig a verem tetején az a karakter van, ami a bemeneten épp következik, akkor a veremből ezt kiszedjük (és a bemeneten tovább lépünk).



Először megmutatjuk, hogy $L(G) \subseteq L(M)$. Ehhez vegyünk egy $x \in L(G)$ szót és tekintsük egy bal levezetését (4.4. definíció). A levezetés n-edik lépésében használt szabály legyen $A \to \gamma$,

$$S \Rightarrow \cdots \Rightarrow yA\alpha \Rightarrow y\gamma\alpha \Rightarrow \ldots \Rightarrow x \tag{7.1}$$

és legyen $y\gamma\alpha = yz\beta$, ahol β a $\gamma\alpha$ első változójával kezdődik (ha van α-ban változó, különben $\beta = \varepsilon$), azaz $z \in \Sigma^*$. Mivel ez egy bal levezetés, y nem tartalmaz változókat, és β választása miatt z sem. A 2. osztálybeli szabályok alkalmazása a továbbiakban nem változtatja az yz részt, az x szót felírhatjuk $x = yzw$ alakban ($w \in \Sigma^*$).

Megmutatjuk, hogy ha

$$S \Rightarrow \cdots \Rightarrow yA\alpha \Rightarrow y\gamma\alpha = yz\beta \tag{7.2}$$

a fenti jelölésekkel igaz, akkor a veremautomata konfigurációira igaz, hogy

$$(q, x, Z_0) \Rightarrow \cdots \Rightarrow (q, w, \beta Z_1), \tag{7.3}$$

Ha ezt a fenti állítást x teljes levezetésére alkalmazzuk (ekkor $\beta = \varepsilon$, mert már nincsenek változók), akkor eljutunk a (q, ε, Z_1) helyzethez, ahonnan egy lépésben kiüríthetjük a vermet, és ezzel egy elfogadó számítást kapunk.

A (7.3) tulajdonságot az n szerinti teljes indukcióval lehet megmutatni, ahol n a levezetés lépésszáma.

Ha $n = 1$, akkor $y = \varepsilon$, $A = S$, a vizsgált levezetési lépés $S \Rightarrow z\beta$ alakú. Az M veremautomata konfigurációinak megfelelő sorozata $(q, x, Z_0) \Rightarrow (q, x, SZ_1) \Rightarrow (q, x, z\beta Z_1) \Rightarrow \ldots \Rightarrow (q, w, \beta Z_1)$.

Legyen $N > 1$ és tegyük fel, hogy minden $n < N$ esetben (7.3) igaz. Ekkor az indukció miatt

$$(q, x, Z_0) \Rightarrow \cdots \Rightarrow (q, zw, A\alpha Z_1).$$

Ebben a helyzetben M helyettesítheti a verem tetején levő A változót a γ sorozattal, és ezért

$$(q, zw, A\alpha Z_1) \Rightarrow (q, zw, \gamma\alpha Z_1) = (q, zw, z\beta Z_1) \Rightarrow \cdots \Rightarrow (q, w, \beta Z_1) \qquad \square$$

Ezzel beláttuk, hogy $L(G) \subseteq L(M)$.

A másik irány bizonyítása is hasonlóan történhet. Legyen $x \in L(M)$ és tekintsük a konfigurációk egy elfogadó számításhoz tartozó

$$(q, x, Z_1) \Rightarrow \cdots \Rightarrow (q, az, A\beta Z_1) \Rightarrow (q, z, \gamma Z_1) \Rightarrow \cdots \Rightarrow (q, \varepsilon, \varepsilon) \tag{7.4}$$

sorozatát, ahol a $(q, az, A\beta Z_1) \Rightarrow (q, z, \gamma Z_1)$ az n-edik lépés. Ekkor persze $x = yaz$ valamilyen $y \in \Sigma^*$ szóra. A konstruált veremautomata olyan, hogy vagy $a = \varepsilon$ és van olyan $A \to \alpha$ szabály a nyelvtanban, hogy $\gamma = \alpha\beta$, vagy $A = \varepsilon$, amikor $\beta = a\gamma$.

Azt fogjuk megmutatni, hogy mindkét esetben a nyelvtanban az S kezdőváltozóból az $ya\gamma$ sorozat levezethető.

Ennek a bizonyítása is n szerinti teljes indukcióval történik. Ha $n = 2$, akkor $x = az$ (tehát $y = \varepsilon$), $A = S$, $\beta = \varepsilon$ és kell legyen egy olyan $S \to \alpha$ szabály a nyelvben, ami megadja a kívánt levezetést.

Legyen $N > 2$ és tegyük fel, hogy minden $n < N$ esetére tudjuk, hogy az állítás igaz. Ez azt jelenti, hogy $yA\beta$ levezethető az S kezdőváltozóból. Ha $A = \varepsilon$, akkor $y\beta = ya\gamma$, tehát készen vagyunk. Amikor a veremben történik valami, akkor $a = \varepsilon$, és ilyenkor a levezetést az $A \to \alpha$ szabállyal folytatva egy megfelelő levezetést kapunk.

7.13. Megjegyzés *Amennyiben a nyelvtan generálja az üres szót is, akkor nincs szükség a Z_0, Z_1 szimbólumokra, verem kezdőszimbólumként használhatjuk az S kezdőváltozót. Ezekre ugyanis csak azért volt szükség, nehogy $\varepsilon \in L$ legyen azonnal.*

7.14. Feladat *Az alábbi CF nyelvtanból konstruáljunk veremautomatát!*

$$A \to \underbrace{a}_{1} \mid \underbrace{aA}_{2} \mid \underbrace{bAA}_{3} \mid \underbrace{AAb}_{4} \mid \underbrace{AbA}_{5}$$

Megoldás: A keletkező nem üres átmenetek

$$\begin{aligned}
(q, \varepsilon, Z_0) &\to (q, AZ_1) \\
(q, \varepsilon, A) &\to \{(q, a), (q, aA), (q, bAA), (q, AAb), (q, AbA)\} \\
(q, a, a) &\to (q, \varepsilon) \\
(q, b, b) &\to (q, \varepsilon) \\
(q, \varepsilon, Z_1) &\to (q, \varepsilon)
\end{aligned}$$

$\square$

7.15. Példa *Vegyük a fenti nyelvtan esetén az abbaaa szó egy levezetését,*

$$A \overset{5}{\Rightarrow} \underline{A}bA \overset{1}{\Rightarrow} abA \overset{3}{\Rightarrow} abb\underline{A}A \overset{2}{\Rightarrow} abba\underline{A}A \overset{1}{\Rightarrow} abbaa\underline{A} \overset{1}{\Rightarrow} abbaaa$$

Nézzük az ennek megfelelő számítást az elkészített veremautomatában

$$\begin{aligned}(q, abbaaa, Z_0) \quad \Rightarrow \quad & (q, abbaaa, AZ_1) \overset{5}{\Rightarrow} (q, abbaaa, AbAZ_1) \overset{1}{\Rightarrow} (q, abbaaa, abAZ_1) \Rightarrow \\ & (q, bbaaa, bAZ_1) \Rightarrow (q, baaa, AZ_1) \overset{3}{\Rightarrow} (q, baaa, bAAZ_1) \Rightarrow \\ & (q, aaa, AAZ_1) \overset{2}{\Rightarrow} (q, aaa, aAAZ_1) \Rightarrow (q, aa, AAZ_1) \overset{1}{\Rightarrow} \\ & (q, aa, aAZ_1) \Rightarrow (q, a, AZ_1) \overset{1}{\Rightarrow} (q, a, aZ_1) \Rightarrow (q, \varepsilon, Z_1) \Rightarrow (q, \varepsilon, \varepsilon)\end{aligned}$$

A környezetfüggetlen nyelvtanok és veremautomaták közötti másik irányú kapcsolatot szintén egy konstrukcióval bizonyítjuk, de első ránézésre itt már az sem teljesen világos, hogyan is csináljunk egy veremautomatából nyelvtant. Az előző konstrukció „megfordítása" azért nem lehetséges, mert a nyelvtannak valahogy kezelnie kell azt, hogy a veremautomatának több állapota is lehet, nem csak egy.

7.16. Tétel *Minden M veremautomata által elfogadott $L = L(M)$ nyelvhez van az L-et generáló környezetfüggetlen nyelvtan.*

Bizonyítás. Tekintsünk egy üres veremmel elfogadó $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$ veremautomatát, ami az L nyelvet fogadja el (7.9. tétel). Ehhez fogunk egy megfelelő $G = (V, \Sigma, S, P)$ nyelvtant megadni.

Először alakítsuk át M-et úgy, hogy megszüntetjük azokat az átmeneteket, amelyekben nem veszünk le a verem tetejéről. Ezeket helyettesítsük azzal, hogy a verem tetején lévő elemet levesszük és utána visszatesszük. Formálisan minden $(q', \alpha) \in \delta(q, a, \varepsilon)$ átmenetet cseréljünk le az $(q', \alpha A) \in \delta(q, a, A)$ átmenetekre, ahol A végigfut Γ összes elemén

Világos, hogy az automata által meghatározott nyelv ezzel nem változik.

Készítsünk most el egy nyelvtant ehhez a veremautomatához.

A nyelvtan változóinak halmaza legyen

$$V = \{[qAp] : q, p \in Q, A \in \Gamma\} \cup \{S\},$$

S lesz a kezdőváltozó. Egy $[qAp]$ alakú változó annak fog megfelelni, hogy az A veremszimbólum a q állapotban került a verembe, és p állapot lesz az, amelyikben ez az A (minden, esetleg belőle származó, a helyére került karakterrel együtt) kikerül a veremből.

A nyelvtan levezetési szabályai a következők:

- minden $q \in Q$ állapothoz $S \to [q_0\, Z_0\, q]$
- egy $(q', \varepsilon) \in \delta(q, a, A)$ átmenethez (ahol $q, q' \in Q$, $A \in \Gamma, a \in \Sigma_0$)

$$[qAq'] \to a$$

- egy $(q', B_1 \dots B_k) \in \delta(q, a, A)$ átmenethez ($q, q' \in Q$, $A, B_1, \dots, B_k \in \Gamma$, $a \in \Sigma_0$)

$$[qAq''] \to a[q_1 B_1 q_2][q_2 B_2 q_3] \dots [q_k B_k q_{k+1}]$$

ahol $q_2, q_3, \dots, q_k, q_{k+1} = q'' \in Q$ végigfut az összes lehetséges állapoton és $q_1 = q'$.

Ezzel egy olyan nyelvtant adtunk meg, aminek szabályai $A \to \alpha$ alakúak, de előfordulhat, hogy $\alpha = \varepsilon$ (amikor $a = \varepsilon$). Ez a 6.14. tétel szerint átalakítható a definíció szerint 2. osztályba tartozó nyelvtanná anélkül, hogy a generált nyelv megváltozna. Így elegendő megmutatni, hogy a megadott G nyelvtan az $L(M)$ nyelvet generálja. Ez az alábbi állításon múlik:

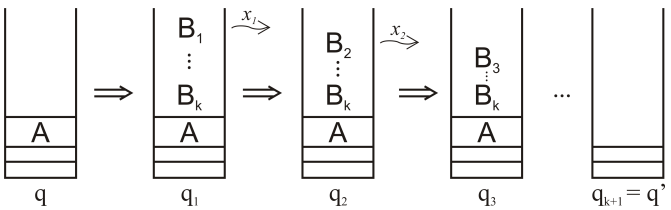
$$[qAq'] \Rightarrow \dots \Rightarrow x \in \Sigma^* \text{ akkor és csak akkor, ha } (q, x, A) \Rightarrow \dots \Rightarrow (q', \varepsilon, \varepsilon) \qquad (7.5)$$

azaz egy $[qAq']$ változóból a nyelvtanban pontosan akkor tudunk egy x szót levezetni, ha a veremautomatát az x bemenettel, A veremtartalommal q állapotban indítva végigolvassa a bemenetet és kiüríti a vermet.

Vegyük észre, hogy ha ez igaz, akkor mivel $x \in L(M)$ azt jelenti, hogy a (q_0, x, Z_0) helyzetből egy $(q, \varepsilon, \varepsilon)$ helyzet elérhető, ezért az elfogadás ekvivalens azzal, hogy a nyelvtanban a $[q_0 Z_0 q]$ változóból az x levezethető. Ez a változó pedig egyetlen szabály alkalmazásával megkapható az S kezdőváltozóból, tehát $L(M) = L(G)$.

A (7.5) állítás mindkét irányát, az előzőekhez hasonlóan, a lépésszám szerinti teljes indukcióval lehet igazolni.

Ha a $[qAq']$ változóból az x szó n lépésben levezethető, és $n = 1$, akkor ez egy $[qAq'] \to a$ szabállyal történik, ami azért része a nyelvtannak, mert ekkor $(q, a, A) \Rightarrow (q', \varepsilon, \varepsilon)$, és pont ez az, amit akartunk. Ha $n > 1$, akkor legyen a levezetés első lépése $[qAq'] \Rightarrow a[q_1 B_1 q_2][q_2 B_2 q_3] \dots [q_k B_k q']$. Ha a jobb oldalon álló i-edik változóból a továbbiakban levezetett szót $x_i \in \Sigma^*$ jelöli, akkor $x = a x_1 x_2 \dots x_k$. Az indukciós feltevés szerint $(q_i, x_i, B_i) \Rightarrow \dots \Rightarrow (q_{i+1}, \varepsilon, \varepsilon)$ teljesül, ha $1 \le i \le k$.



ami összerakva azt adja, hogy $(q, ax_1x_2 \dots x_k, A) \Rightarrow \dots \Rightarrow (q_1, x_1x_2 \dots c_k, B_1) \Rightarrow \dots \Rightarrow (q', \varepsilon, \varepsilon)$

A (7.5) másik irányának részleteit az olvasóra bízzuk. □

7.17. Következmény *Az L nyelv akkor és csak akkor környezetfüggetlen, ha van olyan M veremautomata, melyre $L(M) = L$.*

Bizonyítás. A 7.12. és 7.16. tételek közvetlen következménye. □

7.3. Determinisztikus környezetfüggetlen nyelvek

7.18. Definíció *Egy nyelv* determinisztikus CF nyelv (DCF)*, ha létezik hozzá determinisztikus veremautomata.*

7.19. Példa *Az alábbi két nyelv DCF:*

$$L_1 = \{a^n b^n c^m | n, m \geq 0\}$$
$$L_2 = \{a^m b^n c^n | n, m \geq 0\}$$

A két automata elkészítését az olvasóra bízzuk.

7.20. Állítás *A determinisztikus környezetfüggetlen nyelvek nem zártak a metszetképzésre.*

Bizonyítás. A 6.34. tétel szerint az $L_1 \cap L_2$ nyelv nem is CF. □

Megmutatható, hogy ennek a két nyelvnek az uniója sem DCF (de természetesen CF, hiszen CF nyelvek unója).

7.21. Tétel *A determinisztikus környezetfüggetlen nyelvek nem zártak az unióképzésre, de zártak a komplementer-képzésre.*

Az unióra vonatkozó állítást nem bizonyítjuk. A komplementeres állítás a véges automatáknál látott bizonyításhoz hasonlóan igazolható. Az jelent némi problémát, hogy egy veremautomata a bemeneti szó végigolvasása után még végezhet olvasás nélküli lépéseket, és e közben érinthet elfogadó és nem elfogadó állapotokat is – ennek kezelését itt nem részletezzük.

7.22. Következmény *Van olyan L környezetfüggetlen nyelv, ami nem determinisztikus.*

Bizonyítás. Az állítás következik abból, hogy míg a CF nyelvek zártak az unióra, de nem zártak a metszetre és komplementerre, addig a DCF nyelvek nem zártak az unióra és a metszetre, de zártak a komplementerre. Így a két halmaz (a DCF és a CF nyelvek osztálya) nem eshet egybe. □

7.23. Megjegyzés *Az* $L_1 \cup L_2 = \{a^n b^m c^k : n = m \text{ vagy } m = k\}$ *nyelv környezetfüggetlen, de nem determinisztikus. Ezt az állítást nem bizonyítjuk.*

Ezek szerint a reguláris nyelvekhez képest komoly eltérés, hogy a veremautomatákat általában nem lehet determinizálni. Bár az elméleti vizsgálatokhoz az általános CF nyelvek illeszkednek jobban (pl. a Chomsky-hierarchiában ezek jelennek meg), és ezért is definiáltuk a nemdeterminisztikus változatot, gyakorlati felhasználásra a determinisztikus veremautomaták és a determinisztikus CF nyelvek az inkább használhatóak.

8. fejezet

CF nyelvek algoritmikus kérdései

Ebben a fejezetben a reguláris nyelvekhez hasonlóan megvizsgáljuk, hogy a legegyszerűbb kérdéseket hogy tudjuk eldönteni CF nyelvek és nyelvtanok esetén. Itt most néhány algoritmust adunk meg ezen kérdések egy részére, sajnos azonban több olyan kérdés is van, amely környezetfüggetlen nyelvek esetén algoritmikusan nem megoldható, ezekre majd a 12. fejezetben térünk ki.

A környezetfüggetlen nyelv kétféleképp lehet adott: veremautomatával vagy CF nyelvtannal. A veremautomatából általában egyszerűen nem kapunk jó algoritmust, amennyiben az automata nem determinisztikus (és, mint már láttuk, itt nem tudunk determinizálni). Ezért azt tesszük fel, hogy a nyelv egy CF nyelvtannal adott.

8.1. Beletartozás

Adott: $L \subseteq \Sigma^*$ környezetfüggetlen nyelv, továbbá egy $x \in \Sigma^*$ szó.
Kérdés: $x \in L$?

Az $x = \varepsilon$ eset triviális. A továbbiakhoz vegyük észre, hogy ha a nyelvtanban nincsenek egyszeres szabályok, akkor egy levezetés minden lépésében vagy az előállított sorozat hossza nő, vagy a hossza ugyan nem változik, de a benne levő Σ-beli betűk száma nő. Tehát egy levezetésben mindkét típusú lépésből legfeljebb $|x|$ fordulhat elő. Azaz, ha x egy levezetését keressük, akkor elegendő az összes legfeljebb $2|x|$ mélységű levezetési fát megvizsgálni, valamelyik nem vezet-e el az x szóhoz.

Az eljárás tehát az, hogy előbb kiküszöböljük az egyszeres szabályokat (6.3.2 fejezet), majd végigvizsgáljuk adott mélységig a levezetési fákat (vagy az adott korlátig a ballevezetéseket). Ezzel egy, az x szó hosszában exponenciális algoritmust kapunk.

Egy másik lehetőség, ha az adott nyelvtant átalakítjuk Greibach-normálformájúvá (6.4.2 fejezet). A Greibach-normálforma előnye, hogy a bal levezetés minden egyes lépése előállítja az x szó következő betűjét, így most a levezetés pontosan $|x|$ lépésből áll, az alkalmazható szabályok körét pedig a következő karakter alapján lépésenként leszűkít-

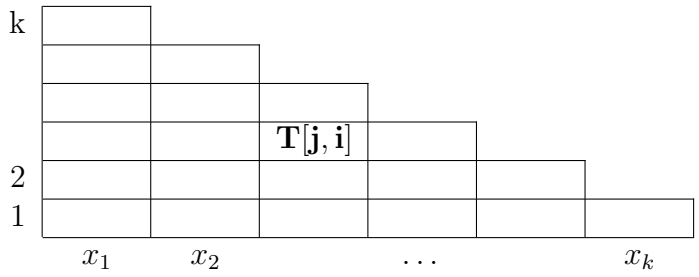
hetjük – ez egy tipikus elágazás-és-korlátozás módszerű algoritmus, amely azért továbbra is exponenciális hosszú lesz.

Bonyolultabbnak tűnő, de hatékonyabb a következő, dinamikus programozást használó eljárás.

Cocke-Younger-Kasami-algoritmus (CYK-algoritmus)

Az algoritmus egy CNF formára alakított G nyelvtannal indul és egy adott x szóhoz meghatározza, hogy x levezethető-e a nyelvtanból. Sőt azt is megkaphatjuk, hogy ha $x \in L(G)$, akkor a levezetése egyértelmű-e. A módszer egy módosításával maguk a levezetések is megkaphatók (ezt a módosítást itt nem tárgyaljuk).

Egy T táblázatot hozunk létre, ennek oszlopai a szó $x_1, x_2, \ldots, x_k \in \Sigma$ betűinek felelnek meg, sorai pedig a $j = 1, 2, \ldots, k$ indexeknek. Igazából csak a táblázatnak csak az átlóját, és az az alatti részt használjuk, a $T[j, i]$-be azok a változók kerülnek, melyekből az $x_i x_{i+1} \ldots x_{i+j-1}$ szórészlet levezethető.



A táblázat kitöltése soronként lentről felfelé történik:

- $j = 1$: az A változó bekerül a $T[1, i]$ cellába, ha $A \to x_i$ szabály G-ben
- $j > 1$: az A változó bekerül a $T[j, i]$ cellába, ha van olyan $A \to BC$ szabály G-ben, melyre $B \in T[\ell, i]$ és $C \in T[j - \ell, i + \ell]$, valamilyen $1 \leq \ell \leq j - 1$-re.

Amennyiben a kezdőváltozó megjelenik a $T[k, 1]$ (bal felső) cellában, akkor $x \in L(G)$, különben $x \notin L(G)$.

Világos, hogy ezzel a kitöltési eljárással a $T[i, j]$ pontosan azokat a változókat tartalmazza, amelyekből a megfelelő részszó levezethető. Ha az egyes celláknál nem csak a változók halmazát, hanem azt is számon tartjuk, hogy melyik változó „hányszorosan” került be az adott cellába, akkor azt is el tudjuk dönteni, hogy a levezetés egyértelmű-e.

Az algoritmus lépésszáma egy k hosszú szó esetén:

- cellánként: $\leq k$ esetet kell ellenőrizni
- a táblázat mérete: k^2

- összesen: $O(k^3)$

Megjegyezzük, hogy vannak a gyakorlatban ennél jobban használható eljárások is, amelyekre a fordítóprogramok is épülnek. Ezek az eljárások általában feltételeznek valami plusz tulajdonságot az elemzendő nyelvről, azon felül, hogy legyen környezetfüggetlen. A CYK-algoritmus egyik erénye, hogy tetszőleges CF nyelvre működik (egy megfelelő formájú nyelvtant feltételezve).

8.1. Példa *Legyen*

$$S \to AB \mid BC$$
$$A \to BA \mid a$$
$$B \to CC \mid b$$
$$C \to AB \mid a$$

a nyelvtan és $x = baaba$ *az elemzendő szó.*

A táblázat alsó sorának kitöltése egyszerű. Vegyük a $T[2,1]$ *elemet. Ehhez egy megvizsgálandó cellapár tartozik:* $T[1,1]$ *és* $T[1,2]$*. Keressük azokat a szabályokat, amik ezen cellapárok változóit tartalmazzák a jobb oldalon (az adott sorrendben), vagyis* $X \to BA$*, illetve* $X \to BC$ *alakúak. A nyelvtan alapján* $X = A$ *vagy* $X = S$*, a cella tartalma* A, S *lesz. A második sor többi eleme hasonlóan kapható. A* $T[3,1]$*-hez két cellapár tartozik:* $T[2,1]$*–*$T[1,3]$ *és* $T[1,1]$*–*$T[2,2]$*, azaz a* $\{A,S\}\{A,C\}$ *vagy* $\{B\}\{B\}$ *halmazokból kiolvasható* AA, AC, SA, SC, BB *párokhoz keresünk szabályt. Ilyen viszont nincs, ezt jelzi a kihúzás a táblázatban. A többi cella hasonló megfontolásokkal kitölthető.*

S, A, C				
–	S, A, C			
–	B	B		
A, S	B	C, S	A, S	
B	A, C	A, C	B	A, C
b	a	a	b	a

A táblázatból kiolvasható, hogy az x *szó a megadott nyelvtanból levezethető, mert a bal felső cellában szerepel a kezdőszimbólum.*

8.2. Üresség

Adott: $L \subseteq \Sigma^*$ reguláris nyelv.
Kérdés: $L = \emptyset$?

Okoskodhatunk a pumpálási lemmára alapozva: Ha p a pumpálási hossz, akkor a nyelv legrövidebb szava p-nél rövidebb (az 5.5 fejezetben látotthoz hasonlóan) tehát elegendő a p-nél rövidebb szavakra a beletartozást ellenőrizni, ami véges sok szó ellenőrzését jelenti.

Egy másik megközelítési mód, hogy meghatározzuk a nyelvtan felesleges szimbólumait (6.3.3 fejezet), ha a kezdőváltozó is ilyen, akkor a nyelv üres, egyébként meg nem.

8.3. Végesség

Adott: $L \subseteq \Sigma^*$ CF nyelv.
Kérdés: L elemszáma véges?

A reguláris nyelvekhez hasonlóan (5.3. tétel) használhatjuk a CF pumpálási lemmát, a bizonyítást itt nem ismételjük meg.

8.2. Tétel *Legyen L egy CF nyelv és p a pumpálási hossz. Az L nyelvnek akkor és csak akkor van végtelen sok eleme, ha létezik olyan $w \in L$ szó, hogy $p \leq |w| \leq 2p$.*

Ennek alapján elegendő az összes, p és $2p$ közötti hosszúságú szót megvizsgálni, hogy valamelyikük beletartozik-e a nyelvbe.

9. fejezet

Turing-gépek

Nyelvek felismerésére eddig kétféle automatát láttunk: a véges automatát és a veremautomatát. Láttuk azt is, hogy veremautomatával több nyelv esetén tudjuk megoldani a nyelvbe tartozás kérdését, mint véges automatával: a veremautomatákkal pontosan a környezetfüggetlen nyelveket lehet felismerni.

Természetesen adódik a kérdés, hogy lehetséges-e olyan automatát szerkeszteni, amivel nem környezetfüggetlen nyelvek esetén is eldönthető a nyelvbe tartozás. Ebben a fejezetben a Turing-gépet, egy olyan, (mint ki fog derülni, igen erős) automatát vizsgálunk, mely a veremautomaták egy valódi általánosítása.

9.1. Egyszalagos, determinisztikus Turing-gép

Először a legegyszerűbb változatát definiáljuk a Turing-gépnek, az egyszalagos Turing-gépet. Ekkor a gépnek egyetlen (egyik irányban végtelen) szalagja van, egy író-olvasó fejjel, mely mindkét irányba tud mozogni a szalagon. (Később tárgyalunk még további változatokat is.)

9.1. Definíció *Egy* Turing-gépet *a következő* $T = (Q, \Sigma, \Gamma, q_0, \_, F, \delta)$ *hetes ír le, ahol:*

- Q *egy véges, nem üres halmaz, ez a gép állapotainak halmaza*
- Σ *egy véges, nem üres halmaz, ez a bemeneti ábécé*
- Γ *egy véges, nem üres halmaz, ez a szalagábécé*
- $q_0 \in Q$ *a kezdőállapot*
- $\_ \in \Gamma \setminus \Sigma$, *a szalagon az üres jel*
- $F \subseteq Q$ *az elfogadó állapotok halmaza*

- δ *az átmeneti függvény,* $\delta : (q, a) \to (q', b, D)$*, ahol* $q, q' \in Q$*,* $a, b \in \Gamma$ *és* $D \in \{B, J, H\}$ *(azaz* ***B****alra,* ***J****obbra vagy* ***H****elyben).*

A gép kezdetben a q_0 állapotban van, a szalag elején a bemeneti szó található (ebben csak Σ-beli karakterek szerepelhetnek), a szalag többi része _ szimbólumokkal van feltöltve és a fej a szalag első (bal szélső) mezőjén áll. Ez a gép kezdőhelyzete.

A gép minden lépésben beolvassa a szalagon az aktuális karaktert, majd ennek, és az eddigi állapotnak a hatására változik az állapota, módosíthatja az író-olvasó fej alatt látott karaktert, majd lép a szalagon egyet jobbra vagy balra, esetleg helyben marad.

Az átmeneti függvény azt írja le, hogy egy lépés során (a gép belső állapotától és az olvasott karaktertől függően) mit tesz a gép: $\delta : (q, a) \to (q', b, D)$ azt adja meg, hogy ha a gép q állapotban van, a betűt olvas a szalagról akkor q' állapotba kerül, az olvasott a-t felülírja b-vel, majd a szalagot olvasó és író fej D irányba lép.

A Turing-gép egy számítás során a kezdőhelyzetből indulva az átmeneti függvénynek megfelelő lépések sorozatát hajtja végre. Ha a szalag elején balra lépne (azaz "leesne" a szalagról az író-olvasó fej), akkor a gép hibával megáll. Egyébként a működés akkor ér véget, ha nem tud lépni, elakad, mert az adott belső állapot - olvasott karakter kombinációra nincs értelmezve az átmeneti függvény. A gép akkor fogadja el a bemeneti szót, ha ez az elakadás F-beli állapotban történik.

Fontos megjegyezni, hogy semmi nem garantálja, hogy egy adott bemenettel a Turing-gép valaha is le fog állni. A következő lehetőségek vannak:

- a gép előbb-utóbb leáll elfogadó állapotban, tehát a szót a gép elfogadja
- a gép előbb-utóbb leáll hibával ("leesik a szalagról") vagy leáll nem elfogadó állapotban
- a gép az adott bemenet hatására soha sem áll meg.

A második és harmadik esetben a gép a bemenetet nem fogadja el. Ha hangsúlyozni szeretnénk, hogy a második eset történt (megállt a gép, de nem elfogadóan), akkor azt mondjuk, hogy a gép a szót elutasítja.

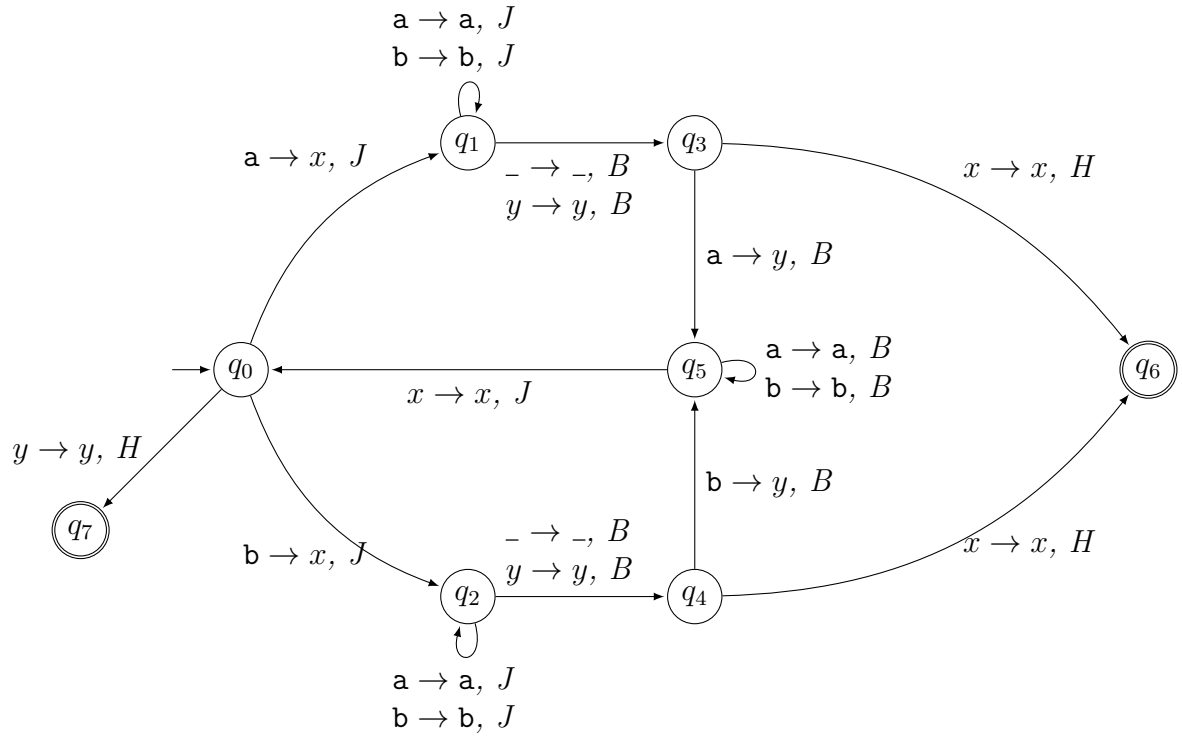
9.2. Megjegyzés *A definíció a determinisztikus, hiányos automatákra emlékeztet. Fontos formai megkötés, hogy most az elfogadás feltétele más: a véges automatáknál és veremautomatáknál megköveteltük, hogy elfogadó számításnál a bemeneti szót végig kell olvasni, nem szabad, hogy a számítás elakadjon. Most viszont a végigolvasás nem szükséges, és elfogadni csak elakadás esetén lehetséges. Lehetne az elfogadási feltételt a korábbiakhoz hasonlóan megadni, ezzel is egy ekvivalens modellt kapnánk, de a jelen forma az általánosan elterjedt, egyszerűbben használható.*

9.3. Definíció *Az* M *Turing-gép által elfogadott nyelv:*

$$L(M) = \{w \in \Sigma : M \text{ elfogadja } w \text{ szót}\}$$

A Turing-gépeket is lehet gráffal ábrázolni, ilyenkor az átmeneteket jelző nyilakon szerepel, hogy milyen olvasott karakter hatására milyen karaktert ír a gép, illetve milyen irányba lép tovább.

9.4. Példa *Az ábrán látható Turing-gép a* $\Sigma = \{\,\mathtt{a}, \mathtt{b}\,\}$ *feletti palindromokat fogadja el.*



A gép működésének lényege, hogy az első karaktertől kezdve sorban ellenőrzi, hogy a szó i-edik karaktere megegyezik-e a hátulról nézve i-edik karakterrel. Egy ellenőrzés során beolvassa az első, még nem ellenőrzött karaktert a szó elejéről, belső állapotában megjegyzi, hogy mit olvasott, majd elbaktat a szó végére a megfelelő karakterhez és ellenőrzi, hogy egyezik-e az állapotában megjegyzett karakterrel.
Kicsit pontosabban a gép működése a következő:

- q_0 állapotban olvassa be a következő ellenőrzendő karaktert a szó első feléből. Ha ez `a`, akkor a felső ágon (q_1, q_3, q_5) halad majd tovább, ha `b`, akkor az alsón (q_2, q_4, q_5), így jegyezve meg belső állapotában, hogy mit olvasott. Továbblépés előtt az `a` (vagy `b`) karaktert x-re cseréli, ezzel jelezve, hogy ezt a karaktert már ellenőrzés alá vonta. A továbbiakban az `a` esetet írjuk le, a `b` eset teljesen hasonló.

- q_1 állapotban olvas és lép jobbra, amíg el nem éri azt a karaktert, amivel az összehasonlítást el kell végeznie. Ez a karakter vagy a szó végén van (utána már _ karakter következik) vagy az első olyan karakter előtt, amit már egyezőnek találtunk (az

egyezett helyeket y-nal jelöli a gép). Ha tehát ezen két eset valamelyikét (_ vagy y) látja, akkor q_3 állapotba kerül és a szalagon visszalép balra egyet, így pontosan arra a karakterre érkezve, amivel az összehasonlítást el kell végeznie.

- A tényleges összehasonlítás q_3-ban történik: ha itt `a`-t talál, akkor ezt a karaktert is ellenőrzöttnek jelöli azzal, hogy átírja y-ra, majd q_5 állapotban visszasétál a szalagon balra, amíg x-et nem lát (ekkor elérte a szó elejének ellenőrzött részét). Ilyenkor q_0-ba kerül és jobbra lép, készen állva a következő ellenőrzésre. Ha q_3-ban `b`-t talál (nincs egyezés), akkor elakad, a szót nem fogadja el, hiszen q_3 nem elfogadó.

- A gép kétféleképpen juthat elfogadó állapotba. Ha páros hosszú palindromunk van, akkor az utolsó ellenőrzés (a szó közepén levő `aa` vagy `bb` megtalálása) után már nem talál a q_0 állapotban újabb ellenőrizendő karaktert, csak y-t, ekkor az elfogadó q_7 állapotba megy, ott elakad, tehát elfogad. Ha páratlan hosszú palindromunk van, akkor a szó közepi `a` vagy `b` x-re cserélése után rögtön egy y következik a szalagon (ennek felel meg az az eset, amikor q_3-ban x-et olvas a gép), ekkor a q_6 elfogadó állapotba lép és elakad.

A továbbiakban fontos szerepet kapnak azok a nyelvek, melyekhez létezik a nyelvet elfogadó Turing-gép. Különösen érdekesek lesznek azok a nyelvek, melyekhez olyan Turing-gép is szerkeszthető, ami nem csak elfogadja a nyelvet, hanem még az a szép tulajdonsága is megvan, hogy a gép mindig megáll. Ez fontos tulajdonsága a gépnek, hiszen ha egy nyelv esetén szeretnénk tetszőleges szóról eldönteni, hogy az eleme-e a nyelvnek vagy sem, akkor egy olyan gép nem nagy segítség, ami nem biztos, hogy valaha is megáll és választ ad.

A fenti két fogalmat ragadja meg a következő két definíció.

9.5. Definíció *Az L nyelv* rekurzívan felsorolható*, ha van olyan M Turing-gép, hogy $L(M) = L$. Azt is szoktuk mondani, hogy az L nyelv ekkor* felismerhető*. A rekurzívan felsorolható nyelvek halmazát* RE*-vel jelöljük.*

9.6. Definíció *Az L nyelv* rekurzív*, ha van olyan M Turing-gép, hogy $L(M) = L$ és M minden bemenet esetén véges sok lépésben megáll. Azt is szoktuk mondani, hogy az L nyelv ekkor* eldönthető*. A rekurzív nyelvek halmazát* R*-rel jelöljük.*

Ha egy nyelvre van őt eldöntő (a definíció szerint mindig megálló) Turing-gép, azaz a nyelv rekurzív, akkor ezen nyelv esetén a nyelvbe tartozás problémája tetszőleges szóra megoldható: a Turing-gépet lefuttatjuk a szóval és mivel mindig megáll, biztosan választ kapunk arra, hogy a szó eleme-e a nyelvnek.

Ha egy nyelvről csak azt tudjuk, hogy rekurzívan felsorolható, azaz csak olyan Turing-gépet ismerünk, ami elfogadja ugyan a nyelvet (a nyelv szavaira megáll elfogadóban), de

nem feltétlenül áll meg minden inputon (a nyelven kívüli szavakra vagy elutasítóban áll meg vagy sosem áll meg), akkor erre a nyelvre ezt a Turing-gépet nem tudjuk a nyelvbe tartozás problémájának megoldására használni: ha beadunk a gépnek egy szót és sokáig nem kapunk választ (a gép nem áll meg), akkor nem tudhatjuk, hogy ez azért van, mert ez egy kivételesen hosszú számítás, aminek nemsokára vége lesz vagy pedig a gép sosem fog leállni.

A definíciókból azonnal adódik, hogy R $\subseteq$ RE. Nemsokára látni fogjuk, hogy a tartalmazás valódi, mutatni fogunk rekurzívan felsorolható, de nem rekurzív nyelveket.

Látunk majd példát olyan nyelvre is, ami még csak nem is rekurzívan felsorolható, de azt már most is be tudjuk bizonyítani (konkrét nyelv megadása nélkül), hogy ilyennek léteznie kell.

9.7. Lemma *Tetszőleges Σ ábécé esetén van olyan $L \subseteq \Sigma^*$ nyelv, amelyre $L \notin$ RE.*

Bizonyítás. Az állítás, a 4.17. tételhez hasonló számossági megfontolásból következik. Egy Turing-gép leírás véges hosszú (például egy szónak tekinthető a gép formális leírása azzal a hetessel, ami definiálja), azt pedig tudjuk, hogy véges hosszú szóból egy véges ábécé felett megszámlálhatóan végtelen sok van. Ezért a Turing-gépek is megszámlálhatóan végtelen sokan vannak, így az általuk felismerhető nyelvek száma is megszámlálhatóan végtelen.

Az összes Σ feletti nyelv számossága azonban kontinuum, mert a nyelvek halmaza éppen a megszámlálhatóan végtelen Σ^* halmaz hatványhalmaza. Mivel a kontinuum nagyobb számosság, mint a megszámlálhatóan végtelen, ezért biztosan van olyan nyelv, amely nem ismerhető fel Turing-géppel. □

9.8. Megjegyzés *Vegyük észre, hogy a Turing-gép általánosítása mind a véges automatának, mind a veremautomatának: mindkét automatát könnyen lehet szimulálni Turing-géppel, az állapotokat a Turing-gép állapotaiban őrizzük, a veremautomata vermét pedig a szalagon tároljuk.*

A véges automatánál annyival van több lehetősége a Turing-gépnek, hogy a fej mindkét irányba mozoghat a szalagon és a fej írni is tud. Meg lehet mutatni, hogy ebből a két különbségből az utóbbi a lényeges, mert egy olyan véges automata is csak a reguláris nyelveket tudja elfogadni, ami mindkét irányban tud mozogni a szalagon.

9.2. k-szalagos, determinisztikus Turing-gép

Az előző részben a rekurzív és rekurzívan felsorolható nyelveket az egyszalagos, determinisztikusan működő Turing-gép segítségével definiáltuk. A fejezet hátralevő részében azt mutatjuk meg, hogy a Turing-gépnek számos olyan változata létezik, mely a 9.1. definícióban adott géppel egyenértékű.

Mielőtt megvizsgálnánk részletesen az első változatot, gondoljuk meg, hogy mennyivel könnyebb dolgunk lenne a palindrom nyelv felismerésével, ha gépünknek két szalagja lenne, mindkét szalagon egy-egy olvasó és író fejjel. Ekkor megtehetnénk azt, hogy az első szalagot elölről, a másodikat pedig hátulról olvassuk, lépésenként egy-egy karakterpárt leellenőrizve. Egy ilyen gép ötletét fogalmazza meg formálisan a következő definíció.

9.9. Definíció *A* k-szalagos Turing-gép *egy olyan Turing-gép változat, amely k darab szalagot használ ($k \geq 1$). A gépet ugyanúgy egy hetes adja meg, mint a korábbi definícióban: $T = (Q, \Sigma, \Gamma, q_0, \_, F, \delta)$, ahol δ kivételével minden ugyanaz, mint korábban, az átmeneti függvény pedig:*

$$\delta(q, a_1 a_2 \dots a_k) \to (q', b_2 b_3 \dots b_k, d_1 d_2 \dots d_k),$$

ahol $a_i, b_i \in \Gamma$, $q, q' \in Q$ és $d_i \in B, J, H$.

A gép kezdetben itt is a q_0 állapotban van, az első szalag elején a bemeneti szó található, a szalag többi része és a többi szalag teljes egészében a _ szimbólummal van feltöltve. Minden szalaghoz tartozik egy fej, ami a szalag első (bal szélső) mezőjén áll. Az első szalag csak olvasható, a többi szalagot munkaszalagnak hívjuk, ezeket írni és olvasni is lehet.

A gép egy lépésben minden szalagon elolvassa a fej alatti karaktert, ezek hatására állapotot vált, a munkaszalagokon módosíthatja az író-olvasó fej alatt látott karaktert, majd lép a szalagokon.

Az átmeneti függvény azt adja meg, ha a gép q állapotban van, $a_1, a_2, \dots, a_k$ betűket olvas a szalagokról, akkor q' állapotba kerül, a munkaszalagokra $b_2, \dots, b_k$ karaktereket ír, majd a fejekkel $d_1, d_2, \dots, d_k$ irányba lép (az input szalagon is mozoghat) .

Ha valamelyik szalag elején balra lépne (azaz „leesne" a szalagról az író-olvasó fej), akkor a gép hibával megáll, egyébként a működés akkor ér véget, ha már nem tud lépni a gép, elakad, azaz az adott belső állapot - olvasott karakterek kombinációra nincs értelmezve az átmeneti függvény. A gép akkor fogadja el a bemeneti szót, ha ez az elakadás F-beli állapotban történik.

9.10. Megjegyzés *Világos, hogy ha egy nyelvet el lehet fogadni egy egyszalagos Turing-géppel, akkor el lehet fogadni k-szalagossal ($k \geq 2$) is: a k-szalagos Turing-gép csinálja azt, hogy átmásolja a bemenetet a 2. szalagjára és ott szimulálja az eredeti Turing-gépet. Az új gép pontosan akkor áll meg, ha az eredeti megállt és pontosan akkor fogad el, ha az eredeti elfogadott.*

Az is igaz azonban, hogy a k-szalagos Turing-gépek nem tudnak többet az egyszalagosnál, ezt látjuk be a következő tételben.

9.11. Tétel *Adott egy k-szalagos M Turing-gép. Ekkor létezik olyan M' egyszalagos Turing-gép, hogy $L(M) = L(M')$.*

Bizonyítás. A konstrukció ötlete az, hogy úgy képzeljük, mintha az egyszalagos Turing-gép szalagja $2k$ sávra lenne osztva: a páratlan sorszámú sávok a k-szalagos Turing-gép megfelelő szalagjaira írt karaktereket tartalmazzák, a páros sorszámú sávok pedig M fejeinek helyzetét kódolják:

- M i-edik szalagjának tartalma megegyezik M' $(2i-1)$-edik sávjának tartalmával $(1 \leq i \leq k)$
- M i-edik szalagjához tartozó fej helyzetét M' $(2i)$-edik sávjában úgy kódoljuk, hogy az adott helyen 1-es karakter áll, minden más helyen pedig üres karakter van $(1 \leq i \leq k)$

A fenti ötletet formálisan a szalag ábécéjének megváltoztatásával, megnövelésével valósítjuk meg: $\Gamma' = (\Gamma \times \{1, \_\})^k$, a Γ'-beli karakterek megfelelő komponensei felelnek meg az egyes sávoknak.

Nézzük most azt, hogyan szimuláljuk M lépéseit M'-vel! Kezdetben M' egyetlen szalagjának első sávja az input szót tartalmazza, minden más páratlan sáv üres (csupa üres karakterből áll), a páros sávokon pedig az első karakter 1, minden máshol üres jel van, M' egyetlen feje a szalag elején található.

M' a belső állapotában fogja tárolni M állapotát, ez aszerint változik, ahogy M állapota változik.

A szimuláció során M egy lépését M' több lépésben fogja szimulálni, egy lépés szimulálását M' mindig úgy kezdi, hogy a fej a szalag elején található. Ezután M' jobbra gyalogol addig, míg minden párosadik sávon meg nem találja M fejeinek pozícióját, közben állapotában megjegyezve, hogy melyik sávban mi volt a fejpozíciókat jelző egyeseknek megfelelő helyen a megfelelő páratlanodik sávban. Mikor minden M-beli fej által olvasott karaktert ismer, kiszámolja M átmeneti függvénye szerint, hogy M mit lépne ebben a helyzetben és miközben visszagyalogol balra a szalag elejére, módosítja eszerint a fejpozíciókat jelző egyeseket és a hozzájuk tartozó karaktereket. □

9.12. Megjegyzés *A bizonyításból látszik, hogy az eredeti k-szalagos és a konstruált egyszalagos gép nemcsak hogy ugyanazt a nyelvet fogadja el, de pontosan ugyanakkor áll meg egy bemeneten az egyik gép, amikor a másik. Ha tehát egy nyelvről meg akarjuk mutatni, hogy rekurzív vagy rekurzívan felsorolható, akkor megtehetjük, hogy k-szalagos Turing-gépet konstruálunk rá, ami (mint ahogy a palindromok esetén láttuk) sokszor egyszerűsíti a dolgunkat.*

A tétel bizonyításából az is látszik, hogy az egyszalagos gépre való áttérésnél a lépések száma megnő, vagyis ha nem csak az a kérdés, hogy van-e Turing-gép egy adott nyelvhez, hanem az is kérdés, hogy a gép hány lépés alatt ad választ, akkor a k-szalagos és az egyszalagos verziók nem feltétlenül viselkednek hasonlóan. A lépésszám-növekedés kérdésével a 11. fejezetben fogunk foglalkozni.

9.3. Nemdeterminisztikus Turing-gép

A következő Turing-gép változat, amit megvizsgálunk, a nemdeterminisztikus Turing-gép. Természetesen adódik a kérdés, hogy nagyobb-e ennek a verziónak a számítási ereje, mint a determinisztikusnak, azaz több nyelvet lehet-e elfogadni vele. Definiáljuk most az egyszalagos nemdeterminisztikus Turing-gépet. (A nemdeterminisztikus Turing-gépek esetén is definiálható többszalagos gép, amely szimulálható egy egyszalagossal, ennek bizonyítása ugyanúgy megy, mint a determinisztikus esetben).

9.13. Definíció *A* nemdeterminisztikus Turing-gépeket *egy* $T = (Q, \Sigma, \Gamma, q_0, \_, F, \delta)$ *hetessel írhatjuk le, ahol* $Q, \Sigma, \Gamma, q_0, \_, F$ *szerepe azonos a Turing-gépek esetén definiálttal, valamint:*

$$\delta(q, a) \subseteq Q \times \Gamma \times \{J, B, H\}$$

A fő különbség a determinisztikus változathoz képest tehát az, hogy most egy állapot és egy olvasott karakter hatására többféle műveletet is végezhet a Turing-gép. Ellentétben a véges automatával és a veremautomatával, a nemdeterminisztikus Turing-gép esetén nem kell külön ε átmenetekkel bajlódnunk. Ha meg akarjuk engedni, hogy a gép léphessen úgy, hogy a fej alatt álló karaktert nem olvassa el, akkor ezt megtehetjük egy olyan szabálycsoport létrehozásával, melyben a fej bármit is olvas, ugyanazt írja vissza és a fej helyben marad.

Egy adott bemeneti szó esetén a gép lehetséges számításait egy úgy nevezett számítási fával is leírhatjuk, melynek csúcsai a gép konfigurációit reprezentálják (mi van a szalagon, milyen állapotban van a gép, hol áll a fej), elágazásai pedig az adott konfigurációban lehetséges továbblépéseknek felelnek meg. A fa gyökere a kezdőkonfiguráció.

9.14. Definíció *A nemdeterminisztikus Turing-gép akkor* fogadja el a bemenetét, *ha a számítási fában van elfogadó levél, azaz olyan konfiguráció, amiből nem lehetséges továbblépés és a hozzá tartozó belső állapot elfogadó.*

Ez úgy is mondható, hogy akkor fogad el a nemdeterminisztikus Turing-gép, ha van elfogadó számítása. Az előfordulhat, hogy a többi számítás között vannak elutasító vagy esetleg soha véget nem érő változatok is.

9.15. Definíció *Az* M nemdeterminisztikus Turing-gép által elfogadott nyelv *azon szavak halmaza, melyeket a gép elfogad, vagyis amely szavakra legalább egy elfogadó számítás létezik.*

9.16. Tétel *Bármely nemdeterminisztikus Turing-gép szimulálható determinisztikus Turing-géppel.*

Bizonyítás. Vázolni fogjuk, hogy hogyan lehet egy nemdeterminisztikus Turing-géphez olyan determinisztikus Turing-gépet készíteni, ami ugyanazt a nyelvet fogadja el.

Legyen M a nemdeterminisztikus Turing-gép, és tekintsük a számítási fáját egy tetszőleges bemeneten. Ebben a számítási fában minden csúcsnak (minden konfigurációnak) legfeljebb $3|Q||\Gamma|$ gyerek-konfigurációja van: $|Q|$ darab lehetséges új állapot van, Γ féle új karaktert írhat le a fej, majd három lehetséges irányba léphet. Az M' Turing-gép a következőképpen fog működni az adott bemeneten:

- Szélességi bejárással végigmegy M számítási fáján: ehhez minden csúcsban nyilvántartja (M átmeneti függvényének ismeretében), hogy mik a lehetséges továbblépések és ezek közül melyeket nézte már meg eddig.
- Ha valamelyik szinten elfogadó levelet talál, akkor megáll és elfogad.
- Ha végigment a teljes fán, és nem talált elfogadó levelet, akkor elutasít.
- Egyébként (azaz ha a fa végtelen és nincs elfogadó levele), nem áll meg. □

9.17. Megjegyzés *Vegyük észre, hogy a mélységi bejárása itt nem használható – ha van egy véget nem érő számítási ág, akkor a bejárás nem jutna el a további ágakba.*

Ha tehát egy nyelvről meg akarjuk mutatni, hogy rekurzívan felsorolható, akkor megtehetjük, hogy ennek belátására nemdeterminisztikus Turing-gépet konstruálunk rá, hiszen a determinisztikus és nemdeterminisztikus Turing-gépek ekvivalensek, amennyiben az általuk elfogadott nyelvek halmazát tekintjük. Fontos azonban látni azt, hogy a nemdeterminisztikus Turing-géppel ekvivalens determinisztikus változat sokkal több lépés után jut elfogadó állapotba, mint amekkora a legrövidebb nemdeterminisztikus elfogadó számítás hossza. Erről a különbségről részletesen fogunk majd beszélni a 11. fejezetben.

9.4. Felsorolós Turing-gépek

Ebben részben egy olyan Turing-gép változatot vizsgálunk meg, mely a rekurzív és rekurzívan felsorolható nyelvek egy újabb lehetséges jellemzését adja és egyben magyarázatul szolgál arra is, hogy miért hívunk egy Turing-géppel elfogadható nyelvet rekurzívan felsorolhatónak.

9.18. Definíció *A* felsorolós Turing-gép *egy olyan speciális Turing-gép, mely egy* $T = (Q, \Sigma, \Gamma, q_0, \_, \delta)$ *hatossal adott, ahol:*

- Q, Γ, q_0, $\_$ *és* δ *ugyanazt jelenti, mint a korábbi Turing-gép definíciókban.*
- *A gépnek nincs bemenete (ezért csak olvasható bemeneti szalagja sincsen és input ábécéje sincs), a működés kezdetén mindegyik szalagon csak üres jelek vannak.*

- *Van egyetlen, csak írható kimeneti szalagja, amin nem léphet vissza, ehhez tartozik egy kimeneti ábécé, most ezt jelöljük Σ-val.*
- *Van ezen kívül tetszőleges számú írható és olvasható munkaszalagja.*
- *Elfogadó állapota nincs.*
- *A gép kimeneti szalagján mindig $x_1\#x_2\#x_3\#\dots x_n$ alakú szó áll ($x_i \in \Sigma^*, \# \notin \Sigma, n \geq 0$).*

9.19. Megjegyzés *Egy felsorolós Turing-gép (input szó híján) csak egyféleképp tud működni: elindítva a csupa üres helyzetből szavakat kezd kiírni a kimeneti szalagjára, a # jellel elválasztva őket egymástól. Lehetséges, hogy ez a kimeneti szalagon levő sorozat folyamatosan nő, végtelen hosszú (ilyenkor biztosan nem áll meg a Turing-gép), de az is lehet, hogy a gép csak véges sok szót ír ki működése során (ez történik például akkor, ha a Turing-gép valamikor is megáll, azaz elakad, mert nem tud továbblépni).*

9.20. Definíció *A gép által* felsorolt *nyelv: $\{x_1, x_2, x_3 \dots\}$ ($x_i \in \Sigma^*$), azon szavak halmaza amiket a működés során a gép valamikor kiír, utána rakva egy # szimbólumot. Az elemek felsorolása közben lehet ismétlődés.*

A következő tétel, amellett, hogy a rekurzívan felsorolható nyelvek egy újabb jellemzését adja, részben magyarázatul szolgál a rekurzívan felsorolható elnevezésre is.

9.21. Tétel

$$L \in \mathrm{RE} \Leftrightarrow \textit{van olyan felsorolós Turing-gép, ami } L\textit{-et sorolja fel}$$

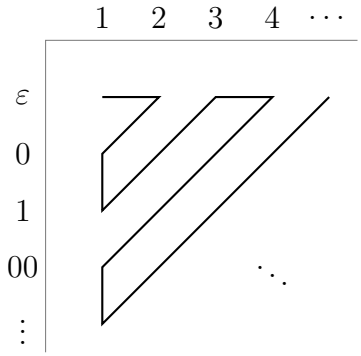
Bizonyítás. $\Leftarrow$ irány:

Tegyük fel, hogy van felsoroló M Turing-gépünk, ami éppen az L nyelv elemeit sorolja fel. Olyan M' Turing-gépet fogunk készíteni, ami minden L-beli szóra megáll elfogadóban és egyetlen L-en kívüli szóra sem áll meg, így tehát M' éppen az L nyelvet fogja elfogadni. Működjön egy x bemeneten M' a következőképpen: futtatja M-et és minden M által kiírt w szóra ellenőrzi, hogy $w \stackrel{?}{=} x$. Ha igen, akkor megáll és elfogadja x-et, egyébként tovább folytatja az ellenőrzést. Világos, hogy ha x eleme L-nek, akkor M' előbb-utóbb megáll és elfogad (akkor amikor M kiírja x-et) és az is világos, hogy ha $x \notin L$, akkor M' sosem fog megállni (mivel M sosem írja ki x-et).

$\Rightarrow$ irány:

Legyen most M egy L-et elfogadó Turing-gép, ebből fogunk egy L szavait felsoroló M' Turing-gépet konstruálni. Az első ötletünk az lehetne, hogy M' szimulálja M-et sorban Σ^* szavain és amiket M elfogad, azokat M' írja ki a kimenetére. Az a gond ezzel, hogy ha M egy x szóval elindítva nem áll meg (x nincs L-ben), akkor M' semelyik x

utáni szót nem fogja kiírni, el sem jut odáig, hogy ezt a szót egyáltalán megvizsgálja. Ezt a nehézséget a következő, diagonális eljárásnak nevezett trükkel lehet áthidalni. Rendezzük sorba Σ^* szavait: ha például $\Sigma = \{ \mathtt{0}, \mathtt{1} \}$, akkor egy lehetséges sorbarakás az, hogy $\varepsilon, \mathtt{0}, \mathtt{1}, \mathtt{00}, \mathtt{01}, \ldots$. Ezután M' egymás után futtatni fogja M-et a Σ^*-beli szavakon, egyre növekvő lépésszámig követve M működését. Ez pontosabban azt jelenti, hogy egy menetben M' valamely y_i szón futtatja M-et legfeljebb j lépésig, fokozatosan növelve j-t és i-t menetről menetre, az alábbi ábra szerint:



Vagyis először szimulálja M-et ε-nal indítva 1 lépésig, aztán ε-nal indítva 2 lépésig, aztán $\mathtt{0}$-val indítva 1 lépésig, aztán $\mathtt{1}$-gyel indítva 1 lépésig stb.

Ha valamely menetben a szimulálás során M elfogadja az éppen használt y_i szót az adott j lépésen belül, akkor M kiírja y_i-t és rátér a szimulálás következő menetére, ha pedig a j lépéses menet anélkül ér véget, hogy M az aktuális input szót elfogadná, akkor M' kiírás nélkül lép át a szimulálás következő menetébe.

Ha $x \in L$, akkor x-et M elfogadja mondjuk $j \geq 1$ lépésben. M' biztosan ki fogja írni x-et akkor, amikor az x szón futtatja M-et j lépésig. Ha $x \notin L$, akkor x-et M sosem fogja elfogadni, tehát M' sosem fogja kiírni, vagyis M' éppen az L nyelv szavait sorolja fel. □

Hasonló állítás igaz a rekurzív nyelvekre is.

9.22. Tétel $L \in \mathrm{R} \Leftrightarrow$ *van olyan felsorolós Turing-gép, ami L elemeit hosszúság szerint növekvő sorrendben sorolja fel.*

9.23. Megjegyzés *A felsorolós Turing-gép definíciójából csak az következik, hogy a gép a felsorolt nyelv szavait (és csak azokat) felsorolja, de a felsorolt szavak sorrendjére semmi megkötés nincsen. Ebben a tételben azt követeljük meg, hogy a gép a nyelv szavait úgy sorolja fel, hogy először jöjjön (ha van) a nulla hosszú szó, aztán az összes 1 hosszú nyelvbeli szó, aztán az összes 2 hosszú szó, stb. Az azonos hosszúságú szavak sorrendjére nincs megkötés. (Valójában ennél általánosabb sorrendek esetén is igaz ez a tétel, elég volna azt megkövetelnünk, hogy legyen egy olyan rögzített sorrendünk, melyben bármely szót véges sok szó előz csak meg.)*

Bizonyítás. A tétel állítása világos módon igaz, ha L véges nyelv. Könnyen látható ugyanis, hogy tetszőleges véges nyelvre van mindig megálló Turing-gép (hiszen már véges automata is van). Másrészt egy véges nyelvre van a nyelv szavait sorrendben felsoroló Turing-gép is: a gép véges átmeneti függvényébe bele tudjuk kódolni a véges sok szót, amit aztán egyesével kiírunk a kimenetre.

Tegyük most fel tehát azt, hogy L végtelen sok szót tartalmaz és lássuk be a tétel állítását ebben az esetben.

$\Leftarrow$ irányban:

Tegyük fel, hogy van egy L elemeit sorrendben felsoroló M Turing-gépünk, amelyből egy L-t elfogadó, minden bemeneten megálló M' Turing-gépet akarunk konstruálni. Működjön M' egy x bemeneten a következőképpen: futtassa M-et és figyelje, hogy M kiírja-e x-et. Ha M valamikor kiírja x-et, akkor M' megáll és elfogad. Ha pedig M kiír egy olyan y-t, ami x után jön a sorban, akkor M' megáll és elutasít. Világos, hogy ha $x \in L$, akkor x-et M egyszer felsorolja és ekkor M' x-et el fogja fogadni. Ha viszont $x \notin L$, akkor (mivel L végtelen) M előbb-utóbb ki fog írni egy x után következő szót. Ekkor M' észreveszi, hogy $x \notin L$ és elutasít.

$\Rightarrow$ irányban:

Most egy L-t elfogadó, minden bemenetre megálló M Turing-gépből fogunk egy L elemeit felsoroló M' Turing-gépet konstruálni. M' úgy működik, hogy sorban futtatja Σ^* szavain M-et (M minden szón megáll, ekkor M' tovább tud lépni a következő szimulációra) és közben M' kiír minden olyan szót, amit a szimulált M elfogadott. Világos, hogy M' pontosan azokat a szavakat írja ki, amelyeket M elfogad. □

9.5. Számoló Turing-gép

További lehetőség a Turing-gép definíciójának kibővítésére, ha a gépre nem nyelvfelismerő eszközként, hanem függvényt kiszámoló eljárásként tekintünk. Ezt az ötletet ragadja meg a következő definíció.

9.24. Definíció *A* számoló Turing-gép *olyan több szalagos Turing-gép, amelynek van egy csak olvasható bementi szalagja (első szalag), van egy csak írható kimeneti szalagja (utolsó szalag), és esetleg több munkaszalag, amelyeket olvasni és írni is tud.*

A gépet egy $T = (Q, \Sigma, \Gamma, q_0, \_, \delta)$ *hatos definiálja, ahol* $Q, \Sigma, \Gamma, q_0, \_$ *szerepe azonos az eddigi Turing-gépek esetén definiáltakkal, az átmeneti függvény pedig (ha* k *darab munkaszalag van):*

$$\delta(q, a_1 a_2 \dots a_{k+1}) \to (q', b_2 b_3 \dots b_{k+2}, d_1 d_2 \dots d_{k+2}),$$

ahol $a_i, b_i \in \Gamma$*,* $q, q' \in Q$ *és* $d_i \in B, J, H$*.*

9.25. Megjegyzés *Vegyük észre, hogy itt az elfogadó állapotok F halmaza hiányzik, mert erre most nincs szükség.*

A gép a szokásos kezdő konfigurációból indul, majd az átmeneti függvény szerint lép, amíg nem akad el. Egy $x \in \Sigma^*$ bemenet esetén a gép vagy soha nem áll meg, s ekkor azt mondjuk, hogy az x szóra nem számol ki semmit vagy egyszer csak megáll, s ha ekkor a kimeneti szalagon az $y \in \Gamma^*$ szó található, akkor azt mondjuk, hogy a gép az x szóhoz y-t számolja ki.

9.26. Definíció *Az M Turing-gép által kiszámított f_M (parciális) függvény a következő*

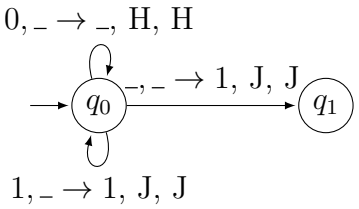
- *ha az x bemenet esetén a gép nem áll meg, akkor $f_M(x)$ nem értelmezett*
- *ha az x bemenethez a gép y-t számolja ki, akkor $f_M(x) = y$.*

A parciális szó itt azt jelzi, hogy a függvény nem feltétlenül definiált Σ^* összes elemére.

9.27. Megjegyzés *Az eredetileg (9.1. definíció) bevezetett Turing-gép lényegében egy olyan speciális számoló Turing-gép, ami egy nyelvbe tartozást reprezentáló $0-1$-függvényt számol ki (és a kiszámolt függvényt nem kiírással, hanem elfogadó állapotokkal jelzi). Ezt az észrevételt precízebb formában a 9.32. és 9.33. tételekben fogjuk kimondani és belátni.*

9.28. Feladat *Legyen f a következő, $\{\,\mathtt{0},\mathtt{1}\,\}$ ábécé feletti szavakon parciálisan definiált függvény: $f(1^n) = 1^{n+1}$ $(n \geq 1)$, a többi helyen f nincs definiálva. Például: $f(1111) = 11111$, $f(100)$ nem értelmezett. Készítsünk el egy, az f függvényt kiszámoló számoló Turing-gépet!*

Megoldás: Megoldásunkban egy kétszalagos számoló Turing-gépet konstruálunk, az ábrán mindkét fej által látott karaktert feltüntettük (munkaszalagok nincsenek).



□

Ha a függvény nem definiált (van a bemenetben `0`), akkor a gép nem áll meg. Ezt azzal érjük el, hogy a gép a bemenet olvasása közben végig q_0 kezdőállapotban marad, és ha itt `0`-t olvas, akkor egy helybenmaradós átmenettel végtelen ciklusba esik. Ha ez nem történik meg (nincs `0` a szóban), akkor minden `1`-et lemásol a kimeneti szalagra is és a szó végi első üres jel hatására leír még egy `1`-et, majd q_1-ben megáll.

A nyelvekhez hasonlóan a függvényeket is osztályozhatjuk aszerint hogy van-e őket kiszámoló (mindig megálló) számoló Turing-gép vagy nincsen.

9.29. Definíció *Az f függvényt* parciálisan rekurzív*nak hívjuk, ha létezik M számoló Turing-gép, hogy $f_M = f$. A függvény* rekurzív*, ha emellett az őt kiszámoló M Turing-gép minden bemeneten megáll. (Azaz az f minden x-re definiálva van.)*

9.30. Megjegyzés *Természetesen ha egy függvény rekurzív, akkor parciálisan rekurzív is.*

9.31. Megjegyzés *A 9.28. feladatban azt mutattuk meg, hogy az ott definiált függvény parciálisan rekurzív. Ennél többet ettől a függvénytől nem is remélhetünk, hiszen nincs mindenhol definiálva. Ebből a szempontból a rekurzív és parciálisan rekurzív függvény fogalma másként viselkedik, mint a rekurzivitás és rekurzívan felsorolhatóság a nyelvek esetén. Ha egy függvény parciálisan rekurzív, akkor azt, hogy rekurzív-e nagyon könnyű eldönteni: amennyiben minden Σ^*-beli szóra értelmezett, akkor rekurzív is, különben pedig nem az. Látni fogjuk a 10. fejezetben hogy nyelvek esetén ez máshogyan van: egy nyelvről még annak ismeretében sem könnyű eldönteni, hogy rekurzív-e, ha tudjuk róla, hogy rekurzívan felsorolható.*

A következő két tétel a rekurzív nyelvek és rekurzív függvények illetve a rekurzívan felsorolható nyelvek és a parciálisan rekurzív függvények között teremt kapcsolatot.

9.32. Tétel

$$L \in \mathrm{R} \Leftrightarrow \textit{Az alábbi módon definiált } f_L(x) \textit{ függvény rekurzív:}$$

$$f_L(x) = \begin{cases} 1 & \textit{ha } x \in L \\ 0 & \textit{ha } x \notin L \end{cases}$$

Bizonyítás. $\Rightarrow$ irányban:

Tegyük fel, hogy van egy L-t elfogadó, minden bemenetre megálló M egyszalagos Turing-gépünk (vagyis a gép L szavaira megáll elfogadó állapotban, az L-be nem tartozó szavakra pedig megáll elutasító állapotban). Olyan M' 3-szalagos számoló Turing-gépet fogunk létrehozni, amely szintén megáll minden bemeneten (ez biztosítja, hogy $f_{M'}$ mindenhol értelmezett) és az L-beli szavakra `1`-et ír a kimeneti szalagra, nem L-beli szavakra pedig `0`-t (ez biztosítja, hogy $f_{M'} = f_L$). M' ezt úgy teszi meg, hogy első szalagjáról átmásolja a második (munka)szalagra az inputot, itt szimulálja M-et annak megállásáig, majd harmadik, kimeneti szalagjára `1`-et ír, ha a megállás elfogadó állapotban történt és `0`-t ír, ha a megállás elutasító állapotban történt.

$\Leftarrow$ irányban:

Legyen M egy minden bemeneten megálló k-szalagos számoló Turing-gép, melyre $f_M = f_L$. Ilyen van, hiszen f_L rekurzív függvény. Ez azt jelenti, hogy M az L nyelv

szavaira megáll és `1`-et ír, a nyelven kívüli szavakra pedig megáll és `0`-t ír. Könnyen készíthetünk ez alapján egy olyan k-szalagos mindig megálló M' Turing-gépet, mely éppen L-et fogadja el. M' úgy működik, hogy először szimulálja az M gépet, majd annak megállásakor megnézi, hogy mi van írva M kimeneti szalagjára: ha `1`, akkor elfogadó állapotba lépve leáll, ha pedig `0`, akkor elutasító állapotba lépve áll meg.□

Hasonló állítás mondható ki a rekurzívan felsorolható nyelvekről is, ami az előzőhöz hasonlóan belátható.

9.33. Tétel

$L \in \mathrm{RE} \Leftrightarrow$ *Az alábbi módon definiált* $f(x)$ *függvény parciálisan rekurzív:*

$$f_L(x) = \begin{cases} 1 & \text{ha } x \in L \\ \text{nincs értelmezve} & \text{ha } x \notin L \end{cases}$$

A rekurzívan felsorolható nyelvekről az alábbi jellemzést is adhatjuk:

9.34. Tétel

$L \in \mathrm{RE} \Leftrightarrow$ *Van olyan* f *parciális rekurzív függvény, amelynek az értékkészlete* L

Bizonyítás. $\Rightarrow$: Tegyük fel, hogy van egy L-et elfogadó, M egyszalagos Turing-gépünk. Olyan M' 4-szalagos számoló Turing-gépet fogunk létrehozni, amely csak az L-beli szavakra fog megállni és ezeken a kimenete megegyezik a bemenettel, ez biztosítja, hogy $f_{M'}$ értékkészlete éppen L lesz. A konstrukció a következő: M' először első szalagjáról átmásolja a munkaszalagokra a bemenetet, az első munkaszalagon szimulálja M-et, egészen addig, amíg M meg nem áll. Ha ez sosem következik be, akkor M' sem áll meg. Ha M megáll elutasító állapotban, akkor M' végtelen ciklusba menekül: a fejek bármit olvasva ugyanazt visszaírják és helyben maradnak. Ha pedig M elfogadó állapotban áll meg, akkor M' a második munkaszalagról a kimeneti szalagra másolja annak tartalmát (a bemenetet), majd megáll.

$\Leftarrow$:

Legyen M az a Turing-gép, ami egy L értékkészletű f függvényt számol ki. Ez azt jelenti, hogy ha M-et lefuttatnánk az összes lehetséges bemenettel, akkor az így kapott kimenetek halmaza L lenne. Konstruálni fogunk egy olyan M' Turing-gépet, melyre $L(M') = L = \{f(x) \mid x \in \Sigma^*\}$. M' lényegében azt csinálja, hogy ha el kell döntenie, hogy az y szó benne van-e L-ben, akkor megnézi, hogy van-e olyan x, melyre $f(x) = y$. Ezt a 9.21. tétel bizonyításában használt szokásos diagonális eljárás szerint teszi meg: az $x_i \in \Sigma^*$ szavakon futtatja M-et j lépésig,

és ha eközben M megáll és y-t írja ki, akkor M' megáll és elfogadja y-t, egyébként pedig a diagonális bejárás szerinti következő szón futtatja M-et a diagonális bejárás szerint következő lépésig.

Világos, hogy M' pontosan akkor fog elfogadni egy szót, ha az előáll $f(x)$ alakban, vagyis ha y-t M' felsorolja. □

9.6. A Turing-gépek számítási ereje

A fejezet eddigi részében számos Turing-gép definíciót vizsgáltunk. A Turing-gépek legtöbb változata nyelvfelismerő automata, a számoló Turing-gépek pedig függvényeket számolnak ki. Azok a Turing-gépek, amelyek minden bemeneten megállnak, tekinthetők egy olyan mindig befejeződő eljárásnak, amit arra használhatunk, hogy egy nyelvbe tartozás illetve egy függvény kiszámításának problémáját megoldjuk.

A Turing-gép számítási erejével kapcsolatban természetesen adódnak a következő kérdések:

1. Nyelvfelismerő automataként tekintve erősebb eszköz-e a Turing-gép, mint az eddig látott véges automata és veremautomata? Azaz: igaz-e, hogy minden olyan nyelvet, amit ezekkel fel lehet ismerni, fel lehet ismerni Turing-géppel is illetve van-e olyan nyelv, amit csak Turing-géppel lehet felismerni, veremautomatával azonban nem? A Turing-gépek és nyelvtanok (és az ezeket felismerő egyszerűbb automaták) pontos kapcsolatát a 12. fejezetben fogjuk részletesen tárgyalni.

2. Nyelvfelismerő eszközként tekintve van-e erősebb automata, mint a Turing-gép? Van-e olyan gép, amivel nem rekurzív nyelvek esetén is el lehet dönteni a nyelvbe tartozást?

3. Kiszámító eszközként tekintve a Turing-gépre, azt mondhatjuk, hogy a Turing-gép egy lehetséges definíciója az algoritmus fogalmának. Kérdés, hogy más algoritmus definíciókkal (pl. egy adott programozási nyelven írható kódok, kvantumszámítógéppel elvégezhető számítások, biológiai számításokat használó algoritmusok) összehasonlítva a Turing-gépet mit mondhatunk: tud-e annyit a Turing-gép, mint ezek illetve tud-e esetleg többet? Vannak-e olyan számítási modellek, amik erősebbek, mint a Turing-gép?

A választ a fenti kérdésekre a Church–Turing-tézis összegzi:

9.35. Tézis *(**Church–Turing-tézis**)*

1. *Egy L nyelv rekurzívan felsorolható (L Turing-géppel felismerhető) $\Leftrightarrow$ van olyan (nem feltétlenül véges) algoritmus, ami pontosan L szavait fogadja el.*

2. *Egy f (parciális) függvény parciálisan rekurzív (f Turing-géppel kiszámítható) $\Leftrightarrow$ van olyan algoritmus, ami minden x bemenetre, ahol $f(x)$ értelmezve van, kiszámolja $f(x)$-et.*

3. *Egy L nyelv rekurzív (L mindig megálló Turing-géppel elfogadható) $\Leftrightarrow$ van olyan mindig megálló algoritmus, ami tetszőleges x bemenet esetén eldönti, hogy $x \stackrel{?}{\in} L$*

4. *Egy f (mindenhol értelmezett) függvény rekurzív (f Turing-géppel kiszámítható) $\Leftrightarrow$ van olyan algoritmus, ami minden x bemenetre kiszámolja $f(x)$-et.*

A tézisben szereplő algoritmus fogalomra nincsen definíciónk, ezért a fenti tézis tételnek nem tekinthető. Általában algoritmusnak tekintünk minden olyan eljárást, aminek leírása véges és futása, ami nem feltétlenül véges, egymás után következő lépések előre meghatározott sorozatából áll. Világos, hogy a Turing-gép algoritmusnak tekinthető, hiszen leírása véges, futása pedig előre definiált lépések egymásutánjából áll. Emiatt a fenti négy állítás $\Rightarrow$ iránya igaz. A Church–Turing-tézis másik iránya azt a tapasztalati tényt rögzíti, hogy eddig senki sem tudott olyan számítási modellt, olyan algoritmust kitalálni, amivel több nyelvet lehetett volna felismerni vagy több függvényt lehetett volna kiszámítani, mint Turing-gépek segítségével.

Röviden tehát azt mondja ki a Church–Turing-tézis, hogy számítási modellek terén a Turing-gép egy lehetséges legjobb, legerősebb eszköz, amivel rendelkezünk.

10. fejezet

Algoritmikus kiszámíthatóság, eldönthetőség

A Church–Turing-tézist igaznak feltételezve pontosan azokat a függvényeket gondoljuk kiszámíthatónak, amelyeket Turing-géppel ki tudunk számítani, és pontosan azon nyelvek esetén tekintjük úgy, hogy a nyelvbe tartozás problémája algoritmussal eldönthető, amelyekre tudunk Turing-gépet szerkeszteni. Ez azt jelenti, hogy különösen érdekes kérdés, hogy mely nyelvek azok, amelyekre van Turing-gép (azaz mely nyelvek rekurzívan felsorolhatóak), illetve melyekre van olyan Turing-gép, amely mindig megáll és a nyelvet fogadja el (vagyis mely nyelvek rekurzívak).

Ebben a fejezetben számos nyelvet megvizsgálunk abból a szempontból, hogy vajon rekurzívak illetve rekurzívan felsorolhatóak-e. Ehhez először egy újabb definícióra, az univerzális Turing-gép fogalmára lesz szükségünk.

10.1. Univerzális Turing-gép

Az univerzális Turing-gép egy olyan Turing-gép, mely más Turing-gépeket tud szimulálni: bemeneti szalagján egy Turing-gép leírását és egy szót kapva, az univerzális Turing-gép pontosan akkor áll meg, ha a leírás által meghatározott Turing-gép a megadott szón megáll és az univerzális gép pontosan akkor fogad el, ha a leírás által meghatározott Turing-gépet a megadott szóval, mint bemenettel indítva, az elfogadó állapotban áll meg.

Ahhoz, hogy egy ilyen gépet megvalósítsunk, szükséges, hogy a Turing-gépek leírásának módját rögzítsük. Egy Turing-gép egy véges jelsorozattal leírható, ezt a leírást fogjuk most szabványosítani. A cél, hogy végül egy Turing-gép megadása egy rögzített ábécé feletti szóval történjen. Ez lehetővé teszi majd, hogy egy Turing-gép leírását tekinthessük leírásnak, amiből ki tudjuk olvasni a gép működését, de tekinthessük egyszerűen egy szónak is, amit akár egy másik Turing-gép bemenetéül adhatunk.

A leíráshoz először „szabványosítjuk” a Turing-gépeket. Egyszerűen megmutatható, hogy minden Turing-gép átalakítható erre a formára, anélkül, hogy megváltozna az elfogadott nyelv, illetve azon szavak halmaza, amelyekre a gép megáll.

10.1. Definíció *Ha M egy Turing-gép és $s \in \Sigma^*$, akkor $M(s)$ jelöli az M gép számításának eredményét az s bemeneten, azaz: elfogad (és megáll), elutasít (és megáll), vagy nem áll meg.*

10.2. Állítás *Tetszőleges M (determinisztikus egyszalagos) Turing-géphez létezik olyan, egyetlen elfogadó állapottal rendelkező M' Turing-gép, hogy minden $s \in \Sigma^*$ szóra $M(s) = M'(s)$.*

Bizonyítás. Az M' gépet úgy kaphatjuk az M Turing-gépből, hogy felveszünk egy új q^+ állapotot, ez lesz M' egyetlen elfogadó állapota. Az M átmeneti függvényét kiterjesztjük oly módon, hogy minden olyan q állapotból, ami M-nek elfogadó állapota volt minden nem definiált átmenetet a q^+ állapotba irányítunk, azaz legyen $\delta'(q, a) = (q^+, a, H)$, ha $\delta(q, a)$ nem volt definiálva. Az q^+ állapotból egyetlen átmenetet sem definiálunk.

Az így megadott M' úgy működik, hogy követi M lépéseit, amíg az el nem akad. Ha az elakadás M egy elfogadó állapotában történik, akkor az újonnan definiált átmenetek egyikével M' átlép az új q^+ állapotba, ahol M' számítása is elakad, ezért M' elfogadta a bemenetet. Ha M számítása nem elfogadó állapotban akad el, akkor M' is ugyanott akad el, az ő számítása sem elfogadó. Természetesen, ha M nem áll meg, akkor M' sem, ilyenkor nem jut el az q^+ állapotba. □

Az előző állítás értelmében feltehető, hogy egy Turing-gép csak egy elfogadó állapottal rendelkezik és ebből nem tud tovább lépni. Hasonlóan az is feltehető, hogy a Turing-gépnek csak két olyan állapota van, amiben megáll (de ezekben mindig): az egyik elfogadó, a másik elutasító.

10.3. Állítás *Minden M (determinisztikus egyszalagos) Turing-géphez létezik olyan, két speciális „végső állapottal” (q^+, q^-) rendelkező M' Turing-gép, amely kizárólag ebben a két állapotban tud megállni, ezekből az egyik (q^+) az M' egyetlen elfogadó állapota és minden $s \in \Sigma^*$ szóra $M(s) = M'(s)$.*

Ezzel egy olyan Turing-gépet kapunk, ami egy bemeneten vagy nem áll meg vagy ha megáll, akkor ezt a kizárólag a q^+ elfogadó vagy a q^- elutasító állapotban teheti.

Egy *szabványos Turing-gép* a következő alakú:

- a gép determinisztikus (a 9.16. tétel alapján ez elérhető)
- a gépnek 1 szalagja van (a 9.11. tétel alapján ez elérhető)
- $\Sigma = \{0, 1\}$ (minden ábécét át tudunk kódolni binárissá)

- $\Gamma = \{0, 1, \ldots, t-2, \_\}$, azaz összesen t darab szalagszimbólum van és ezek közül az utolsó az üres _ karakter (a szalagjeleket kódolhatjuk számokkal)
- $Q = \{0, 1, \ldots, r\}$ és 0 a kezdőállapot (az állapotokat is jelölhetjük számokkal)
- $F = \{r\}$ (a 10.2. állítás és szerint ez elérhető)

Tehát egy ilyen szabványos Turing-gépnél nem kell külön megadni a kezdőállapotot és az elfogadó állapotokat, elegendő a szalagábécé méretét, az állapotok számát, valamint az átmeneti függvényt leírni. Ezt fogjuk most egységes formába foglalni.

10.4. Definíció *Az M* szabványos Turing-gépnek a kódja *legyen*

$$\underbrace{\#\#\#}_{\substack{\text{elválasztók}\\ \notin\{0,1\}}} \quad \underset{\substack{\uparrow\\ \text{szalagjelek}\\ \text{száma}}}{t} \quad \#\# \quad \underset{\substack{\uparrow\\ \text{az utolsó állapot}\\ \text{sorszáma}}}{r} \quad \#\# \underbrace{q\#a\#q'\#b\#d}_{\substack{\delta(q,a)=(q',b,d)\\ \text{átmenet leírása}}} \#\# q'' \ldots \#\#\#$$

ahol $\# \notin \{0, 1\}$ egy elválasztó szimbólum, a szereplő számokat pedig binárisan kódoljuk.

Ha még azt is megtesszük, hogy az így kapott teljes leírást tovább kódoljuk úgy, hogy 0 *helyett* 00*-t,* 1 *helyett* 11*-t, # helyett pedig* 01*-t írunk, akkor a Turing-gépnek egy $\{0, 1\}$ feletti bitsorozatot feleltetünk meg.*

A kódban (az eredeti alakban) három # jelöli a leírás elejét és végét, két-két # választja el a különböző komponenseket. Az elején szerepel a szalagábécé, illetve az állapothalmaz méretének megadása (binárisan), utána jön az átmeneti függvény. Ebben a részben az összes lehetséges átmenetet felsoroljuk, ezeket egymástól két # választja el. Egy $\delta(q, a) = (q', b, d)$ átmenetnek a kódban a $q\#a\#q'\#b\#d$ szakasz felel meg. Mivel ilyenekből véges sok van, az egész kód egy véges hosszú (végül bináris) sorozat lesz.

10.5. Definíció *Ha $w \in \{0, 1\}^*$ egy Turing-gép fent leírt kódolása, akkor a* hozzá tartozó Turing-gépet *jelöljük M_w-vel.*

10.6. Állítás *A Turing-gépek kódjai által alkotott $L = \{w : w$ egy Turing-gép kódja$\}$ nyelv rekurzív.*

Bizonyítás. Vázlat: Egy karaktersorozat akkor lesz egy Turing-gép kódja, ha az elválasztójelek jó helyen vannak és az átmeneti függvény leírásában előforduló állapotok, illetve karakterek kódja nem nagyobb, mint r illetve $t - 1$. Ez az ellenőrzés egy Turing-géppel véges időben megtehető, ezért a nyelv rekurzív. □

10.7. Megjegyzés *Fontos megjegyezni, hogy ez az ellenőrzés nem terjed ki arra, mit is csinál a kódhoz tartozó Turing-gép, itt kizárólag egy „szintaktikai" ellenőrzésről van szó.*

Eddigi Turing-gépeink lényegében egy-egy célra (egy adott nyelv elfogadására vagy egy adott függvény kiszámolására) készültek, nem felelnek meg a programozható számítógép fogalmának. A következő univerzális Turing-gép viszont felfogható úgy, hogy kap egy programot (egy Turing-gép kódját) valamint egy szót, és a programot futtatja a szón, tehát egy interpreterként működik.

10.8. Definíció *Egy U* univerzális Turing-gép *bemenete $w\#s$ alakú, ahol $w, s \in \{0,1\}^*$ és U a következőképp viselkedik:*

$$U(\underset{\substack{\uparrow\\ \text{gép}\\ \text{leírás}}}{w}\ \#\ \underset{\substack{\uparrow\\ \text{bemenet}}}{s}\ ) = \begin{cases} M_w(s) & \text{ha } w \text{ egy Turing-gép kódja} \\ \text{megáll, elutasít} & \text{ha } w \text{ nem Turing-gép kódja} \end{cases}$$

A fenti definíció tehát a fejezet elején informálisan megfogalmazott követelményt adja meg: az univerzális Turing-gép pontosan úgy viselkedik a $w\#s$ bemeneten, mint M_w az s bemeneten, amennyiben w Turing-gépet kódol.

Természetesen adódik a kérdés, hogy lehetséges-e ilyen univerzális Turing-gépet készíteni. Erre az igenlő választ a következő tétel szolgáltatja.

10.9. Tétel *Létezik univerzális Turing-gép.*

Bizonyítás. Vázoljuk egy U univerzális Turing-gép konstrukcióját. A gépnek legyen három szalagja. Az első szalagon van a bemenet, a másodikat úgy használja majd mint M_w a saját szalagját, a harmadik szalagon pedig U az M_w aktuális állapotát tárolja.

Először U ellenőrzi, hogy a bemenet $w\#s$ alakú-e ($w, s \in \Sigma^*$). Ha nem ilyen, akkor U megáll, elutasít. Azt is ellenőrzi, hogy w kódol-e Turing-gépet, és ha nem, akkor U szintén megáll, elutasít. Egyébként, azaz amikor w egy Turing-gép kódja, U a második szalagra átmásolja az s szót, a harmadikra 0-t ír (hiszen ez M_w kezdőállapota). Ezek után követi M_w lépéseit: a 2. szalagon a megfelelő karaktert olvassa, és ehhez, valamint a 3. szalagon levő állapothoz megkeresi a bemenet w részében a megfelelő átmenetet, majd elvégzi a megfelelő változtatásokat a 2. és 3. szalagon.

Ha $M_w(s)$ nem áll meg, akkor persze az azt szimuláló U sem. Amennyiben $M_w(s)$ gép megáll, akkor U is álljon meg, méghozzá pontosan akkor lépjen elfogadó állapotba, ha M_w elfogadó állapotban áll meg. □

10.10. Megjegyzés *A harmadik szalagra azért volt szükség, mert U állapotainak számát a definiálásakor rögzítjük, de olyan Turing-gépeket is tudnia kell szimulálni, melyeknek sokkal több állapotuk van mint U-nak.*

Természetesen a kapott univerzális Turing-gépet átalakíthatjuk olyanná is, aminek egyetlen szalagja van (9.11. tétel), sőt szabványos formátumúvá is – így neki is lesz kódja.

10.2. Nevezetes rekurzív és rekurzívan felsorolható nyelvek

Az univerzális Turing-gép segítségével számos nevezetes nyelv definiálható, először ezek közül tanulmányozunk néhányat.

A 9.7. tételben már láttuk, hogy biztosan van olyan nyelv, mely nem rekurzívan felsorolható. Most egy konkrét nyelvről fogjuk belátni, hogy ilyen tulajdonságú.

10.11. Definíció Diagonális nyelv*:*

$$L_d = \{w : w \text{ egy Turing-gép kódja és } w \notin L(M_w)\}$$

Tehát a diagonális nyelv olyan w szavakat tartalmaz, amelyek által leírt Turing-gépek a saját kódjukat nem fogadják el, azaz a saját kódjukat bemenetként megkapva nem állnak meg, vagy nem elfogadó állapotban állnak meg.

10.12. Tétel $L_d \notin \mathrm{RE}$*, azaz* L_d *olyan nyelv, ami nem rekurzívan felsorolható.*

Bizonyítás. Indirekt tegyük fel, hogy $L_d \in \mathrm{RE}$. Ekkor létezik olyan M Turing-gép, hogy $L(M) = L_d$. Legyen M kódja w, azaz ekkor $M = M_w$ és így $L(M_w) = L_d$. Két lehetőség van $w \overset{?}{\in} L_d$-re:

- Ha $w \in L_d$, akkor a diagonális nyelv definíciója szerint w olyan Turing-gépet ír le, amely nem fogadja el a saját kódját, azaz $w \notin L(M_w) = L_d$, ami ellentmond a feltételnek.
- Ha $w \notin L_d$, akkor a diagonális nyelv definíciója szerint $w \in L(M_w) = L_d$, ami szintén ellentmondás.

Mindkét esetben ellentmondásra jutottunk, tehát az L_d nyelv nem lehet rekurzívan felsorolható. □

A következő érdekes nyelv, amit megvizsgálunk az U univerzális Turing-gép által elfogadott nyelv.

10.13. Definíció Univerzális nyelv*:*

$$L_u = \{w\#s : w \text{ egy Turing-gép kódja és } M_w(s) = \text{elfogad}\}$$

Ebbe a nyelvbe tehát pontosan azok a $w\#s$ alakú szavak tartoznak, amelyek esetén a w egy Turing-gép leírása, és ez a Turing-gép elfogadja az s szót, azaz $s \in L(M_w)$.

10.14. Tétel (Turing) $L_u \in \mathrm{RE} \setminus \mathrm{R}$*, azaz az univerzális nyelv nem rekurzív, de rekurzívan felsorolható.*

Bizonyítás. A nyelv definíciójának közvetlen következménye, hogy L_u rekurzívan felsorolható, hiszen ez éppen az U univerzális Turing-gép által elfogadott nyelv, $L_u = L(U)$.

Indirekt tegyük fel, hogy $L_u \in \mathrm{R}$. Ekkor létezik olyan M Turing-gép, hogy $L(M) = L_u$ és M minden bemeneten megáll. M segítségével tehát el lehet dönteni tetszőleges w Turing-gép kód és s szó esetén, hogy M_w elfogadja-e s-et, azaz hogy $s \in L(M_w)$ fennáll-e. Ezt az M-et használhatjuk arra, hogy a diagonális nyelvhez készítsünk egy Turing-gépet, amiről pedig a 10.12. tétel értelmében tudjuk, hogy lehetetlen.

Definiáljuk a következő M' Turing-gépet. Ha w nem Turing-gép kód, akkor a w bemeneten M' megáll és elutasít. Egyébként M' futtassa az M gépet a $w\#w$ bemeneten. A feltevés szerint M biztosan meg fog állni.

- Ha M elfogadó állapotban áll meg, akkor M' lépjen nem elfogadó állapotba és álljon meg.
- Ha M nem elfogadó állapotban áll meg, akkor M' lépjen elfogadó állapotba és álljon meg.

Világos, hogy $w \in L(M')$ akkor és csak akkor teljesül, ha w egy Turing-gép kódja és $w\#w \notin L(M) = L_u$. Ez viszont pont azt jelenti, hogy $w \in L_d$. Tehát az M' Turing-gép éppen a diagonális nyelvet fogadja el (és még meg is áll minden bemeneten), ahonnan $L_d \in \mathrm{R} \subset \mathrm{RE}$ következne, ami ellentmondás, azaz az indirekt feltevés hibás volt, vagyis $L_u \notin \mathrm{R}$. □

Az elfogadás helyett nézzük most a *megállási problémát*. A következő nyelvbe azon (kód, bemenet) párok tartoznak, melyekre igaz, hogy az adott Turing-gép megáll, ha az adott bemenettel indítjuk.

10.15. Definíció Megállási nyelv*:*

$$L_h = \{w\#s : w \ \textit{egy Turing-gép kódja és} \ M_w \ \textit{az} \ s \ \textit{inputon megáll}\}$$

A definícióból adódik, hogy $L_u \subseteq L_h$.

10.16. Tétel $L_h \in \mathrm{RE} \setminus \mathrm{R}$*.*

Bizonyítás. A két részt külön bizonyítjuk.

- $L_h \in \mathrm{RE}$: Jelöljön U egy univerzális Turing-gépet. Ennek segítségével definiálunk egy M Turing-gépet, ami az L_h nyelvet fogadja el.

  M az $w\#s$ bemeneten a következőkép működik:

1. Ha a bemenet nem ilyen alakú vagy w nem egy Turing-gép kódja, akkor M megáll és elutasít.
2. Egyébként M futtatja U-t az $(w\#s)$ bemeneten és
   (a) ha $U(w\#s)$ megáll és elfogad, akkor M elfogad;
   (b) ha $U(w\#s)$ megáll és elutasít, akkor M elfogad;
   (c) ha $U(w\#s)$ nem áll meg, akkor M sem áll meg.

Világos, hogy $L(M) = L_h$, ezért L_h rekurzívan felsorolható.

- $L_h \notin$ R: Indirekt tegyük fel, hogy $L_h \in$ R. Ekkor létezik olyan M Turing-gép, hogy $L(M) = L_h$ és M megáll minden bemenetre, vagyis M tetszőleges w Turing-gép kódról és s szóról véges sok lépés után megmondja, hogy M_w s inputon megáll-e.

  M segítségével készíteni fogunk egy mindig megálló M' Turing-gépet az L_u univerzális nyelvre. Az M' gép a következőképpen működik egy bemeneten:

  1. ha ez a bemenet nem $w\#s$ alakú vagy w nem egy Turing-gép kódja, akkor M' megáll, elutasít;
  2. ha w egy Turing-gép kódja, akkor M' lefuttatja M-et a $w\#s$ bemeneten és
     (a) ha $M(w\#s) =$ megáll, elutasít, akkor M' is megáll és elutasít;
     (b) ha $M(w\#s) =$ megáll, elfogad, akkor M' lefuttatja az M_w gépet az s szón és
        i. ha $M_w(s) =$ elfogad, akkor M' megáll elfogadó állapotban;
        ii. ha $M_w(s) =$ elutasít, akkor M' megáll nem elfogadó állapotban.

  Az utolsó esetben nem kell azzal foglalkozni, hogy mi van, ha $M_w(s)$ nem áll meg, hiszen erre az ágra csak akkor kerülünk, ha $M(w\#s)$ elfogad, ami pontosan azt jelenti, hogy M_w az s bemeneten meg fog állni.

  Világos, hogy M' pontosan akkor fogadja el $w\#s$-et, ha M_w az s bemeneten megáll és elfogad, így $L(M') = L_u$. Az is látszik, hogy M' minden bemeneten megáll, így ellentmondásra jutottunk a 10.14. tétel állításával, vagyis $L_h \notin R$. □

A fenti tételt (a Church–Turing-tézis felhasználásával) úgy is megfogalmazhatjuk, hogy nincs olyan, mindig véges sok lépés után választ adó eljárás, ami tetszőleges algoritmusról és tetszőleges inputról eldöntené, hogy az adott algoritmus az adott inputon megáll-e. Ha az algoritmusok helyébe például egy adott programozási nyelven írott programokat képzelünk, akkor a tétel lényegében azt mondja, hogy nem lehetséges olyan programot írni, amely bármelyik szintaktikailag helyes programról véges időben meg tudja mondani, hogy az egy adott inputtal elindítva valaha le fog-e állni.

A következő tétel pedig lényegében azt mutatja, hogy olyan általános eljárást sem lehet írni, amely bármely szintaktikailag helyes programról meg tudja mondani, hogy az üres inputtal elindítva a program futása valaha le fog-e állni.

10.17. Definíció

$$L_\varepsilon = \{w : w \textit{ egy Turing-gép kódja és } M_w \textit{ az } \varepsilon \textit{ inputon megáll }\}$$

10.18. Tétel (Church)

$$L_\varepsilon \in \mathrm{RE} \setminus \mathrm{R}$$

Bizonyítás. Először igazoljuk, hogy az L_ε nyelv rekurzív felsorolható. Ehhez egy M' Turing-gépet kell mutatni, melyre $L(M') = L_\varepsilon$.

Legyen M egy, az L_h nyelvhez tartozó Turing-gép. Ilyen létezik, hiszen $L_h \in \mathrm{RE}$ (10.16. tétel).

Az M' Turing-gép egy w bemeneten szimulálja az M Turing-gép működését a $w\#\varepsilon$ bemeneten, M' pontosan akkor álljon meg és fogadja el a w szót, ha M megáll és elfogadja a $w\#\varepsilon$ szót. Ez utóbbi, $L(M) = L_h$ miatt pontosan akkor következik be, ha w egy Turing-gép kódja és M_w elfogadóan megáll az ε bemeneten. Ez azt jelenti, hogy a megadott M' Turing-gép pontosan az L_ε-beli szavakra áll meg elfogadó állapotban, vagyis $L(M') = L_\varepsilon$.

Most megmutatjuk, hogy L_ε nem rekurzív. Ehhez indirekt tegyük fel, hogy $L_\varepsilon \in \mathrm{R}$, vagyis tegyük fel, hogy létezik egy olyan M Turing-gép, amely éppen L_ε szavait fogadja el ($L(M) = L_\varepsilon$) és M minden bemenetre megáll.

Ezt az M Turing-gépet felhasználva egy olyan M' Turing-gépet hozunk létre, amely az L_h nyelvet fogadja el és minden bemenetre megáll. Ez ellentmond annak a ténynek, hogy L_h nem rekurzív (10.16. tétel).

Az M' egy adott bemeneten csinálja a következőt:

- Ha a bemenet nem $w\#s$ alakú vagy ebben w nem egy Turing-gép kódja, akkor M' megáll, elutasít.

- Ha w egy Turing-gép kódja, akkor M' előállítja a w kódú M_w Turing-gép egy olyan M'' változatát, amibe az s bemenet bele van építve. Ez azt jelenti, hogy az így elkészített M'' gép bármely bemenetnél (így az üres bemenetnél is) úgy működik, hogy az első néhány lépésével a szalagjára felírja s-et, majd szimulálja M_w-t ezen az s bemeneten. (Mivel adott w-ből az M'' konstruálása algoritmikusan lehetséges, ezért feltehető, hogy ezt az M' Turing-gép is meg tudja valósítani – de nem lenne nagyon nehéz a megfelelő Turing-gép átmeneteit, és így a kódját is közvetlenül megadni.)

  Az M' által készített M'' gép pontosan úgy viselkedik az üres bemeneten, mint M_w az s szón. Ha tehát azt akarjuk eldönteni, hogy M_w megáll-e az s bemeneten, akkor elegendő, ha M'-vel lefuttatjuk az M gépet, azon a bemeneten, ami M'' kódjának felel meg.

Az így elkészített M' Turing gép tetszőleges $w\#s$ bemenetről véges sok lépésben megmondja, hogy $w\#s \in L_h$ vagy sem, amivel ellentmondásra jutottunk, hiszen $L_h \notin \mathrm{R}$. □

10.3. Műveletek rekurzív és rekurzívan felsorolható nyelvekkel

Ebben a részben azt vizsgáljuk meg, hogyan viselkednek az R és RE nyelvosztályok a szokásos nyelvi műveletekre.

10.19. Tétel

1. *Ha* $L_1, L_2 \in \mathrm{R}$*, akkor* $L_1 \cup L_2 \in \mathrm{R}$.
2. *Ha* $L_1, L_2 \in \mathrm{R}$*, akkor* $L_1 \cap L_2 \in \mathrm{R}$.
3. *Ha* $L_1 \in \mathrm{R}$*, akkor* $\overline{L_1} \in \mathrm{R}$.
4. *Ha* $L_1, L_2 \in \mathrm{RE}$*, akkor* $L_1 \cup L_2 \in \mathrm{RE}$.
5. *Ha* $L_1, L_2 \in \mathrm{RE}$*, akkor* $L_1 \cap L_2 \in \mathrm{RE}$.

Bizonyítás. Legyen M_1 és M_2 olyan Turing-gép, amely az L_1, illetve az L_2 nyelvet fogadja el.

1.: Ebben az esetben feltehetjük, hogy M_1 és M_2 minden bemeneten megáll, hiszen a nyelvek a feltevés szerint rekurzívak. A két Turing-gépet „*sorosan*" kötve egymás után elkészíthetünk egy mindig megálló M Turing-gépet az $L_1 \cup L_2$ nyelvre. Ezt például a következőképp tehetjük meg. Az M gépnek legyen három szalagja, az elsőn levő bemenetet kezdetben átmásolja a másik két szalagra. Ezután M_1-et futtatja a második szalagján. Ha M_1 elfogad, akkor M megáll és elfogad. Ha M_1 elutasítja s-et, akkor M lefuttatja M_2-t is, a harmadik szalagot használva. Amikor ez megáll, M akkor fogad el, ha M_2 elfogadó állapotban állt meg. Mivel M_1 és M_2 is megáll minden bemeneten, az így leírt M is meg fog állni és azokat a szavakat fogadja el, amelyeket vagy M_1 vagy M_2 elfogadja, azaz $L(M) = L(M_1) \cup L(M_2)$.

Egy másik lehetséges módszer az unióhoz, ha a véges automatáknál látott megoldást alkalmazzuk kis módosítással. A 3.4. tétel bizonyításában használt ötlet az volt, hogy a két gépet *„párhuzamosan"* futtatjuk, és M elfogad, ha M_1 vagy M_2 elfogad. Amiért ez most itt egy az egyben nem működik az az, hogy M_1 és M_2 egymástól függetlenül mozgatja a szalagon a fejet, egymástól függetlenül írják felül a szalagot. Hogy ebből ne legyen baj, M-nek most is három szalagja lesz. Első lépéseivel a bemenetet átmásolja a második és a harmadik szalagra is, majd a 2. szalagon azt csinálja, amit M_1 a saját szalagján, a 3. szalagon pedig M_2 lépéseit szimulálja. Ezen a módon a két gépet egymástól függetlenül, párhuzamosan tudja futtatni. Azt kell még meggondolni, hogyan érjen véget M számítása. Az egyszerűség kedvéért feltehetjük, hogy M_1 és M_2 is a 10.3. következményben leírt alakú, vagyis két állapotban tudnak elakadni, az egyik elfogadó, a másik elutasító. Az M gép átmeneteit definiáljuk úgy, hogy egy számítás csak akkor érjen véget, ha mindkét

M_i gép megállt (végső állapotba került), M akkor fogadja el, ha legalább az egyik végső állapot elfogadó.

2.: Hasonló meggondolásokkal látható be. Most is feltehetjük, hogy M_1 és M_2 minden bemeneten megáll, hiszen a nyelvek rekurzívak. A két Turing-gépet „*sorosan*" kötve egymás után elkészíthetünk egy mindig megálló M Turing-gépet az $L_1 \cap L_2$ nyelvre. Annyit kell csak módosítani az előzőeken, hogy most, ha M_1 elutasítja s-et, akkor az új gép is ezt teszi, egyébként pedig lefuttatja az M_2 gépet az s szón (a harmadik szalagon), és pontosan akkor fogad el, amennyiben M_2 is ezt teszi. Világos, hogy M mindig megáll és éppen azokat a szavakat fogadja el, amiket mindkét M_i elfogadott.

A *„párhuzamos"* változat is működik: abban az esetben csak az elfogadás feltételét kell megváltoztatni: M pontosan akkor fogad el, ha mindkét M_i az elfogadó állapotba jutott.

3.: Mivel feltehető, hogy M_1 mindig megáll, alkalmazhatjuk a véges automatáknál bevált ötletet. Az M Turing-gép ugyanolyan, mint M_1, csak az elfogadó állapotok halmaza az eredeti F halmaz komplementere: $F' = Q \setminus F$. (Amennyiben M_1 a 10.3. következményben leírt típusú Turing-gép, akkor csak a két végső állapot szerepét kell felcserélni.)

4.: Itt a „soros" változat nem működik, mert ha M_1 nem áll meg egy adott s bemeneten, akkor nem kerül sor M_2 futtatására, pedig az lehet, hogy elfogadná s-et.

A „párhuzamos" konstrukció kis változtatással most is működik. Az M átmeneti függvényét úgy kell definiálni, hogy a számítás ne csak akkor álljon le, ha mindkét gép megállt, hanem akkor is, ha valamelyik M_i az elfogadó állapotába kerül, és ez könnyen megtehető.

5.: A 2. pontban leírt párhuzamos konstrukció itt is jó lesz, hiszen ha mindkét gép elfogad, akkor M is megáll elfogadó állapotban, egyébként pedig M nem fogad el (esetleg nem is áll meg). □

10.20. Megjegyzés *Az* RE *nyelvek komplementerére vonatkozó állítás nem véletlenül hiányzik. Könnyű látni, hogy a 3. pontban vázolt megoldás itt nem jó, hiszen ha az M_1 gép nem áll meg az s szón, akkor az M gép sem fog, tehát s-et sem M_1, sem M nem fogadja el. Ez a probléma nem oldható meg további ötletekkel sem, látni fogjuk, hogy a 10.27. tételből következik majd, hogy* RE *nem zárt a komplementerképzésre.*

Most nézzük meg azt, hogy mit mondhatunk az R és RE nyelvosztályoknak a konkatenációra és a tranzitív lezártra való zártságáról.

10.21. Tétel

1. *Ha $L_1, L_2 \in$ R, akkor $L_1 L_2 \in$ R.*
2. *Ha $L_1 \in$ R, akkor $L_1^* \in$ R.*
3. *Ha $L_1, L_2 \in$ RE, akkor $L_1 L_2 \in$ RE.*

4. Ha $L_1 \in \mathrm{RE}$*, akkor* $L_1^* \in \mathrm{RE}$.

Bizonyítás. Legyen M_1 és M_2 olyan Turing-gép, amely az L_1, illetve az L_2 nyelvet fogadja el. Mindegyik esetben egy M nemdeterminisztikus Turing-gépet fogunk készíteni a megfelelő nyelvhez M_1 és M_2 felhasználásával. Ebből szükség esetén lehet determinisztikus Turing-gépet is készíteni.

1.: Az M legyen egy 3 szalagos nemdeterminisztikus Turing-gép, ami először a bemenet első néhány karakterét átmásolja a második szalagra, a bemenet további részét pedig a harmadik szalagra. (Minden karakter első szalagról való átírásakor van egy nemdeterminisztikus választás: a következőt is a második szalagra írja, vagy innentől kezdve a harmadik szalagra másol.) Ezután a 2. és 3. szalagon visszaállítja a fejet a szalag elejére. Ezt követően a 2. szalagon, az eredeti bemenet oda írt első részével szimulálja az M_1 gépet. Ha ez megáll nem elfogadó állapotban, akkor M is így tesz. Ha viszont M_1 elfogad, akkor M a 3. szalagon az eredeti bemenet oda írt második részével az M_2 gépet szimulálja és akkor fogad el, ha M_2 így tesz.

Világos, hogy $s \in L(M)$ pontosan azokra a szavakra teljesül, melyek felbonthatóak $s = s_1 s_2$ alakban, ahol $s_i \in L_i$, azaz ha $s \in L_1 L_2$.

2.: Az előzőhöz hasonlóan most is nemdeterminisztikusan osztjuk fel a szót néhány részre. Viszont ezeket a részeket nem írhatjuk fel különböző szalagokra, hiszen a szalagok száma rögzített. Ezért legyen M egy olyan két szalagos nemdeterminisztikus Turing-gép, ami először felírja a bemenet első néhány karakterét a 2. szalagra, ezen szimulálja az M_1 gépet. Ha ez elutasít, akkor M is. Ha viszont M_1 elfogad, akkor M a bemenetéről a következő néhány karaktert átmásolja a 2. szalag elejére, ezen ismét szimulálja az M_1 gépet, és így tovább. M akkor fog elfogadni, ha a bemenet végére ért, és minden szimuláló fázisban M_1 elfogadó állapotban állt meg.

A 3. és 4. esetekben hasonlóan járhatunk el. Az nem jelent problémát, hogy esetleg egy szimulálás során valamelyik M_i nem áll meg, hiszen ez úgysem lehet egy elfogadó ág, nem baj, ha M sem áll meg. □

10.22. Megjegyzés *Lehet egyből determinisztikus Turing-gépet is definiálni az adott nyelvekhez. Ekkor a véletlen vágások helyett az összes lehetséges szétvágást ki kell próbálnunk. A szétvágások tesztelése az első két esetben ugyanúgy működik, mint a nemdeterminisztikus esetben. A két utóbbi esetben azonban nem lehet esetleg a végtelenségig szimulálni valamelyik* M_i *gépet, hiszen így nem jutnánk oda, hogy egy következő felosztási lehetőséget is kipróbáljunk. Ilyenkor a diagonális eljáráshoz hasonló módszer segít: a véges sok lehetséges felosztás mindegyikét először 1, majd 2, 3, ... lépéses szimulációkkal ellenőrizzük.*

A továbbiakban hasznos lesz, ha bevezetjük a komplementer nyelvosztály fogalmát. Nyelvosztálynak nevezzük nyelvek tetszőleges halmazát, egy nyelvosztály komplementere pedig a nyelvosztályban levő nyelvek komplementereiből áll.

10.23. Definíció *Az X* nyelvosztály komplementer nyelvosztálya*:*

$$\operatorname{co} X = \{L : \overline{L} \in X\}$$

Azaz X komplementer nyelvosztályát úgy képezzük, hogy minden X-beli nyelvnek külön-külön vesszük a komplementerét és az így kapott nyelveket összegyűjtjük, tehát ez a komplementer nyelvek osztálya. Fontos megjegyezni, hogy ez különbözhet az X-be nem tartozó nyelvek $\overline{X}$ osztályától.

10.24. Példa

- *Ha L egy tetszőleges nyelv és az X nyelvosztály csak az L nyelvből áll, azaz $X = \{L\}$, akkor* $\operatorname{co} X$ *egyetlen eleme az $\overline{L}$ nyelv, $\overline{X}$ pedig minden L-től különböző nyelvet tartalmaz.*

- *Ha X azokból a $\{\,\mathtt{0},\mathtt{1}\,\}$ feletti nyelvekből áll, melyekben van páros hosszú szó, akkor* $\operatorname{co} X$ *nyelvei azok, amelyek nem tartalmaznak minden páros hosszú szót (hiszen a komplementerükben van legalább egy páros hosszú), $\overline{X}$ pedig azokból a nyelvekből, melyek nem X-beliek, tehát minden szavuk páratlan hosszú. Így pl. $L_1 = \{\mathtt{00},\mathtt{01}\} \in X$, de ugyanakkor $L_1 \in \operatorname{co} X$ és persze $L_1 \notin \overline{X}$, másrészt $L_2 = \{\,\mathtt{0},\mathtt{1}\,\}^* \in X$, de $L_2 \notin \operatorname{co} X$ és $L_2 \notin \overline{X}$.*

- *Ha X a reguláris nyelvek halmaza, akkor* $\operatorname{co} X = X$*, mert a reguláris nyelvek zártak a komplementerképzésre, míg $\overline{X}$ a nem reguláris nyelvekből áll.*

- *A* co RE *osztály azokat a nyelveket tartalmazza, melyeknek a komplementere rekurzívan felsorolható, vagyis amelyeknek a komplementerére van Turing-gép. Tehát pontosan azok a nyelvek vannak* co RE*-ben, amelyeknél a nyelvbe nem-tartozást fel lehet ismerni Turing-géppel.*

A definíció közvetlen következményeit fogalmazza meg az alábbi lemma.

10.25. Lemma *A komplementer nyelvosztályok két alapvető tulajdonsága:*

1. *Ha $X \subseteq Y$, akkor* $\operatorname{co} X \subseteq \operatorname{co} Y$
2. $\operatorname{co}(\operatorname{co} X) = X$

Bizonyítás. Legyen $X \subseteq Y$. Az $L \in \operatorname{co} X$ a definíció szerint pontosan akkor teljesül, ha $\overline{L} \in X$. Mivel $X \subseteq Y$, ezért ilyenkor $\overline{L} \in Y$ is fennáll, vagyis $L \in \operatorname{co} Y$ is teljesül.

$L \in \operatorname{co}(\operatorname{co} X)$ a definíció szerint pontosan akkor teljesül, ha $\overline{L} \in \ coX$, ez pedig pontosan akkor igaz, ha $L = \overline{\overline{L}} \in X$ fennáll. □

Az eddigiek következményeként azonnal adódik a következő állítás:

10.26. Állítás $\operatorname{co}\mathrm{R} = \mathrm{R}$

Bizonyítás. A 10.19. tétel 3. pontja alapján ha $L \in \mathrm{R}$, akkor $\overline{L} \in \mathrm{R}$ vagyis $\operatorname{co}\mathrm{R} \subseteq \mathrm{R}$ fennáll. Ha erre alkalmazzuk a 10.25. lemma 1. pontját, akkor a $\operatorname{co}(\operatorname{co}\mathrm{R}) \subseteq \operatorname{co}\mathrm{R}$ összefüggéshez jutunk. Ezt a 2. összefüggésből kapott $\operatorname{co}(\operatorname{co}\mathrm{R})) = \mathrm{R}$ egyenlőséggel kombinálva kapjuk, hogy $\mathrm{R} \subseteq \operatorname{co}\mathrm{R}$ is fennáll. A két tartalmazást összevetve $\mathrm{R} = \operatorname{co}\mathrm{R}$ adódik. □

Az előző tétel szerint pontosan azokra a nyelvekre lehet algoritmussal eldönteni a nyelvbe tartozás kérését, amelyekre a nyelvbe nem-tartozás kérdését el lehet dönteni.

A következő tétel azt mondja ki, hogy egy nyelvre pontosan akkor van mindig megálló Turing gép (a nyelv akkor rekurzív), ha mind a nyelvbe tartozás, mind a nyelvbe nem-tartozás felismerhető Turing géppel (mind a nyelv, mind a nyelv komplementere rekurzívan felsorolható).

10.27. Tétel $\mathrm{R} = \mathrm{RE} \cap \operatorname{co}\mathrm{RE}$

Bizonyítás. A két irányú tartalmazást külön bizonyítjuk:

- $\subseteq$ irány: $\mathrm{R} \subseteq \mathrm{RE}$ a definíció alapján fennáll, innen pedig a 10.19. tétel 3. pontja alapján $\operatorname{co}\mathrm{R} \subseteq \operatorname{co}\mathrm{RE}$ adódik, amiből $\operatorname{co}\mathrm{R} = \mathrm{R}$ miatt kapjuk a kívánt $\mathrm{R} \subseteq \operatorname{co}\mathrm{RE}$ tartalmazást.

- $\supseteq$ irány: Tegyük most fel, hogy $L \in \mathrm{RE}$ és $L \in \operatorname{co}\mathrm{RE}$ egyaránt teljesül. Ekkor definíció szerint van olyan M_1 Turing-gép, amire $L(M_1) = L$ és van olyan M_2 Turing-gép, amire $L(M_2) = \overline{L}$. A két gép párhuzamos futtatásával egy olyan M Turing-gépet kaphatunk, ami pontosan az L-beli szavakat fogadja el és mindig megáll. Ehhez M működjön úgy hogy az s bemeneten az M_1 és M_2 gépeket párhuzamosan (külön szalagon) futtatja. Amennyiben az M_1 gép megáll és elfogad, akkor M álljon meg elfogadó állapotban, ha pedig az M_2 gép az, ami megáll és elfogad, akkor M álljon meg egy nem elfogadó állapotban. Mivel tetszőleges s szót az M_1 és M_2 közül pontosan az egyik fogadja el (és ekkor persze meg is áll), ezért M biztosan meg fog állni és világos, hogy éppen L szavait fogadja el. □

A fenti tételből következik, hogy $\operatorname{co}\mathrm{RE} \neq \mathrm{RE}$, hiszen $\operatorname{co}\mathrm{RE} = \mathrm{RE}$ esetén $\mathrm{R} = \mathrm{RE}$ is fennállna, ez pedig például $L_u \in \mathrm{RE} \setminus \mathrm{R}$ miatt (10.14. tétel) nem igaz, vagyis az RE nyelvosztály nem zárt a komplementer-képzésre.

10.4. Nyelvi tulajdonságok

Ebben a részben azt fogjuk vizsgálni, hogy melyek azok a nyelvi tulajdonságok, amelyek meglétét algoritmussal ellenőrizni lehet. Ki fog derülni, hogy (kevés triviális tulajdonságtól eltekintve) nincs olyan algoritmus, ami tetszőleges Turing-gép kódjáról el tudná

dönteni, hogy a kódnak megfelelő Turing-gép nyelve rendelkezik-e az adott tulajdonsággal.

Az első tételből az derül ki, hogy azt az egyszerűnek tűnő tulajdonságot sem tudjuk algoritmikusan ellenőrizni, hogy egy nyelv (amit egy kódjával adott Turing gép definiál) üres-e vagy sem.

10.28. Definíció

$$L_\emptyset = \{w : w \textit{ egy Turing-gép kódja és } L(M_w) = \emptyset\}$$

10.29. Tétel

$$L_\emptyset \in \operatorname{co}\mathrm{RE} \setminus \mathrm{R}$$

Bizonyítás. Először megmutatjuk, hogy $L_\emptyset \in \operatorname{co}\mathrm{RE}$. Ez a definíció szerint azt jelenti, hogy $\overline{L_\emptyset} \in \mathrm{RE}$, tehát ehhez a nyelvhez kell egy M Turing-gépet megadnunk. $\overline{L_\emptyset}$-ben kétféle típusú szó van: olyan, ami Turing-gép kódja, és olyan, ami nem:

$$\overline{L_\emptyset} = \{w : w \text{ nem Turing-gép kódja }\} \cup \{w : w \text{ egy Turing-gép kódja és } L(M_w) \neq \emptyset)\}$$

Az M Turing-gép egy w bemeneten a következő csinálja:

- Először ellenőrzi, hogy w egy Turing-gép kódja-e. Ha nem, akkor M megáll és elfogad.
- Ha w egy Turing-gép kódja, akkor M keres egy, az $L(M_w)$ nyelvben levő szót. Amennyiben talál egy ilyet, akkor megáll és elfogad. Az $L(M_w)$-beli szó keresését a már korábban is használt diagonális eljárással (lásd a 9.21. tétel bizonyítása) oldhatjuk meg: az M gép szimulálja M_w működését Σ^* szavain, egyre növekvő lépésszámmal, az adott bejárás szerint. Ha valamely esetben M_w az adott lépésszámon belül elfogadja a vizsgált szót, akkor M elfogadó állapotban megáll, és ezzel elfogadja a w szót. Ellenkező esetben (vagyis amikor M_w semmilyen inputra sem áll meg) M nem fog megállni.

Ha M megáll, akkor el is fogad, és az, hogy M nem áll meg, csak úgy következhet be, ha a w inputja Turing-gépet kódol, olyat, amire nincs olyan s szó és j lépéskorlát, hogy M_w az s bemeneten j lépésen belül elfogadó állapotban megáll, vagyis amikor az $L(M_w)$ nyelv üres. Az a helyzet tehát, hogy $L(M) = \overline{L_\emptyset}$, vagyis $\overline{L_\emptyset}$ rekurzívan felsorolható.

Most még belátjuk, hogy $L_\emptyset \notin \mathrm{R}$. Ehhez indirekt tegyük fel, hogy $L_\emptyset \in \mathrm{R}$, és legyen M egy minden bemeneten megálló Turing-gép, melyre $L(M) = L_\emptyset$. Az M segítségével olyan M' Turing-gépet adunk meg, amely minden bemeneten megáll, és $L(M') = L_\varepsilon$. Egy ilyen M' létezése ellentmondana a 10.18. tételnek.

Működjön M' egy w bemeneten a következőképpen:

- M' először ellenőrzi, hogy w egy Turing-gép kódja-e. Ha nem, akkor megáll és elutasítja a w bemenetet.
- Ha w egy Turing-gép kódja, akkor M' létrehozza azt az M'' Turing gépet, ami úgy működik, hogy tetszőleges s bemenet hatására M_w-t futtatja az üres szón (az s szót figyelmen kívül hagyva) és
  - ha M_w megáll, akkor M'' is megáll elfogadó állapotban (azon az inputon, amivel indítottuk),
  - ha M_w nem áll meg, akkor M'' sem áll meg.

  Mivel M'' nem foglalkozik a bemenetével, ezért minden s input szóra ugyanúgy viselkedik: vagy mindent elfogad vagy semmit sem. Ha M_w a ε inputon megáll, akkor $L(M'') = \Sigma^*$, ellenkező esetben pedig az M_w gépet szimuláló M'' gép sem áll meg, ilyenkor $L(M'') = \emptyset$. Összefoglalva: $L(M'')$ pontosan akkor nem üres, ha $w \in L_\varepsilon$.

  Az M' Turing-gép, miután előállította M'' kódját, (jelölje ezt a kódot w''), futtassa az M gépet a w'' bemeneten.

  Ha $M(w'')$ elfogad, akkor M' elutasít, ha pedig $M(w'')$ elutasít, akkor M' elfogad.

Az M' gép minden w inputon meg fog állni, mert M'' létrehozása véges sok lépésben megtehető (a w Turing-gép leírás ismeretében), majd M futtattása is véges folyamat, hiszen M mindig megáll.

M' konstrukciójából következik, hogy M' pontosan akkor fogadja el a w inputot, amikor M elutasítja w''-t, azaz amikor $L(M'')$ nem üres, vagyis amikor $w \in L_\varepsilon$.

Tehát M' egy olyan Turing-gép, ami mindig megáll és $L(M') = L_\varepsilon$, de ilyen nem létezhet, ellentmondásra jutottunk. □

A fenti tétel $L_\emptyset \notin \mathrm{R}$ részét úgy is fogalmazhatjuk, hogy az üresség, mint nyelvi tulajdonság olyan, hogy nem lehetséges algoritmikusan eldönteni egy leírásával adott Turing gépről, hogy az elfogadott nyelv rendelkezik-e ezzel a tulajdonsággal.

Természetesen merül fel a kérdés, hogy vannak-e más olyan tulajdonságok, amelyek hasonlóan viselkednek. Ennek tárgyalásához szükségünk van a következő definícióra.

10.30. Definíció *Legyen T nyelveknek egy tulajdonsága. A T egy* nem triviális nyelvi tulajdonság*, ha vannak olyan $L_1, L_2 \in \mathrm{RE}$ nyelvek, hogy L_1-re teljesül, L_2-re pedig nem teljesül T.*

A T tulajdonsággal rendelkező nyelvek halmazát is szokás T-vel jelölni, ezzel a jelöléssel a T tulajdonság akkor nem triviális, ha

$$T \cap \mathrm{RE} \neq \emptyset$$
$$T \cap \mathrm{RE} \neq \mathrm{RE}$$

Világos, hogy egy T triviális nyelvi tulajdonság esetén könnyű egy Turing-gép w kódjáról eldönteni, hogy $L(M_w)$ rendelkezik-e a T tulajdonsággal. Mivel T triviális tulajdonság, ezért vagy minden Turing-gép nyelve rendelkezik vele vagy egyik sem, vagyis a válasz a kód ismerete nélkül is megadható.

Az érdekes kérdés az, hogy melyek azok a nem-triviális nyelvi tulajdonságok melyek esetén el lehet dönteni azt, hogy egy leírásával adott Turing-gép nyelve rendelkezik-e ezzel a tulajdonsággal. A továbbiakban ezt fogjuk vizsgálni.

10.31. Definíció *Legyen T egy nyelvi tulajdonság. A T-hez* tartozó L_T nyelv *azon Turing-gépek kódjaiból áll, amiknek nyelve T tulajdonságú:*

$$L_T = \{w : w \text{ egy Turing-gép kódja és } L(M_w) \text{ } T \text{ tulajdonságú } \}.$$

10.32. Megjegyzés *Mivel maga L_T is egy nyelv, róla is megkérdezhetjük, hogy T tulajdonságú-e. A válasz az, hogy az L_T nyelv általában nem T tulajdonságú.*

Ha T triviális, akkor L_T vagy az üres nyelv vagy az összes Turing-gép kódjából álló nyelv, L_T mind a két esetben rekurzív (a 10.6. állítás, illetve amiatt, mert $\emptyset \in \mathrm{R}$).

Mielőtt rátérnénk annak vizsgálatára, hogy mely nem-triviális T tulajdonságok esetén lesz az L_T nyelv rekurzív, lássunk néhány lemmát, amik megkönnyítik majd a vizsgálódásunkat.

10.33. Lemma *Legyen T egy nyelvi tulajdonság és jelölje $\overline{T}$ a tulajdonság komplementerét. Ekkor*

$$L_{\overline{T}} = \overline{L_T} \cap L',$$

ahol L' az összes Turing-gép kódjából álló nyelv.

Bizonyítás. $\overline{L_T}$ azon szavakat tartalmazza, amik vagy nem kódjai Turing-gépnek vagy olyan Turing-gép kódjai, aminek nyelve nem T tulajdonságú. Ez utóbbi esetet úgy is fogalmazhatjuk, hogy ekkor olyan Turing-gépek kódjairól van szó, amiknek nyelve $\overline{T}$ tulajdonságú.

Így tehát $\overline{L_T}$ metszete az L' nyelvvel éppen az $L_{\overline{T}}$ halmaz lesz. □

10.34. Lemma *Egy T nyelvi tulajdonságra az L_T nyelv pontosan akkor rekurzív, ha az $L_{\overline{T}}$ nyelv rekurzív.*

Bizonyítás. Szimmetria okok miatt elég megmutatni, hogy ha L_T rekurzív, akkor az $L_{\overline{T}}$ is az. Az $L_{\overline{T}}$ nyelv az előző lemma szerint két nyelv metszete. Az első nyelv a feltétel és a rekurzív nyelvosztály komplementerképzésre való zártsága miatt maga is rekurzív, a második pedig a 10.6. állítás szerint az. Mivel rekurzív nyelvek metszete rekurzív (10.19. tétel), ezért $L_{\overline{T}}$ is rekurzív. □

A következő tétel azt mondja, hogy a triviális nyelvi tulajdonságokat leszámítva nem lehetséges algoritmust adni annak eldöntésére, hogy egy Turing-gép nyelve adott tulajdonságú-e vagy sem.

10.35. Tétel (Rice) *Legyen T egy nyelvi tulajdonság és tekintsük az ilyen tulajdonságú nyelvet elfogadó Turing-gépek kódjait, azaz legyen*

$$L_T = \{w : w \text{ egy Turing-gép kódja és } L(M_w) \text{ } T \text{ tulajdonságú } \}.$$

Ha T egy nem triviális nyelvi tulajdonság, akkor $L_T \notin \mathrm{R}$.

Bizonyítás. Az előző lemma miatt feltehetjük, hogy $\emptyset \notin T$, vagyis az üres nyelv nem T tulajdonságú. Ha nem így lenne, akkor T helyett a $\overline{T}$ komplementerre bizonyítunk.

Legyen tehát $\emptyset \notin T$ és $L \in \mathrm{RE}$ egy olyan nyelv, amelyre $L \in T$. (Ilyen L van, mert T nem triviális tulajdonság.) Legyen M egy olyan Turing-gép, amelyre $L(M) = L$.

Indirekt tegyük fel továbbá, hogy $L_T \in R$, azaz létezik egy olyan M_T, amely minden bemenetre megáll és $L(M_T) = L_T$. Ebből olyan M'-t fogunk konstruálni, amelyik minden bemenetre megáll és $L(M') = L_u$.

Működjön M' egy $w\#s$ bemeneten a következőképpen:

- Ha a bemenet nem $w\#s$ alakú vagy ebben a w nem egy Turing-gép kódja, akkor M' megáll és elutasítja a bemenetet.
- Ha w egy Turing-gép kódja, akkor M' konstruál a $w\#s$ szó felhasználásával egy olyan (w-től és s-től függő) M'' Turing-gépet (illetve ennek kódját), amely a következőképpen működik egy x bemeneten. M'' először az x inputtól függetlenül futtatja M_w-t s-en, majd ezen futás eredményétől függően esetleg futtatja M-et x-en:
  - ha M_w megáll és elutasítja s-et, akkor M'' megáll és elutasítja x-et
  - ha M_w megáll és elfogadja s-et, akkor M'' lefuttatja M-et x-en:
    - Ha $M(x)$ elfogad, akkor M'' megáll és elfogadja x-et
    - Ha $M(x)$ elutasít, akkor M'' megáll és elutasítja x-et
    - Ha $M(x)$ nem áll meg, akkor M'' sem áll meg
  - ha M_w nem áll meg s-en, akkor M'' sem áll meg.

Figyeljük meg, hogy az M'' gép $w\#s \in L_u$ esetén M-et futtatja, vagyis ekkor $L(M'') = L$. Amikor pedig $w\#s \notin L_u$, akkor $L(M'') = \emptyset$, hiszen ekkor nem tudunk M'' elfogadó ágára jutni. Az M'' gép az M_w gép és M segítségével véges sok lépésben megalkotható és az általa elfogadott nyelv pontosan akkor rendelkezik a T tulajdonsággal, ha $w\#s \in L_u$.

M' készítse el a fenti M'' Turing gép w'' kódját (véges sok lépésben), majd futtassa w''-re a mindig megálló M_T gépet. M' így minden inputra meg fog állni és pontosan

akkor fogadja el a $w\#s$ bemenetet, ha $M_T(w'')$ elfogad, vagyis amikor M'' nyelve T tulajdonságú. Az előbb láttuk, hogy ez pontosan akkor van, amikor $w\#s \in L_u$, vagyis M' egy mindig megálló Turing-gép lenne az L_u univerzális nyelvre, ami ellentmondás. □

A Rice-tétel következménye, hogy többek között a következő tulajdonságok egyikének megléte sem eldönthető. Nem létezik tehát olyan mindig választ adó algoritmus, amely egy Turing gép kódjáról el tudná dönteni, hogy például a gép fogad-e el bármit is, véges sok szót fogad-e el, van-e az elfogadott nyelvnek legalább 100 eleme, elfogadja-e a gép a rögzített s szót, reguláris-e a Turing gép nyelve, elfogad-e a gép minden szót illetve rekurzív-e a az elfogadott nyelv. A felsoroltakhoz a megfelelő tulajdonságok és hozzájuk tartozó nyelvek a következők:

T_1 : a nyelv nem üres $\Rightarrow L_{T_1} = \{w : \exists M_w,$ és $L(M_w) \neq \emptyset\} \notin R$

T_2 : a nyelv véges $\Rightarrow L_{T_2} = \{w : \exists M_w,$ és $L(M_w)$ véges $\} \notin R$

T_3 : a nyelvnek legalább 100 eleme van $\Rightarrow$

$L_{T_3} = \{w : \exists M_w$ és $L(M_w)$-nek legalább 100 eleme van $\} \notin R$

T_4 : a nyelvbe beletartozik egy rögzített s szó $\Rightarrow L_{T_4} = \{w : \exists M_w$ és $s \in L(M_w)\} \notin R$

T_5 : a nyelv reguláris $\Rightarrow L_{T_5} = \{w : \exists M_w$ és $L(M_w)$reguláris $\} \notin R$

T_6 : a nyelv $= \Sigma^*$ $\Rightarrow L_{T_6} = \{w : \exists M_w$ és $L(M_w) = \Sigma^*\} \notin R$

T_7 : a nyelv rekurzív $\Rightarrow L_{T_7} = \{w : \exists M_w$ és $L(M_w) =$ rekurzív$\} \notin R$

Ahhoz, hogy belássuk, hogy ezen L_{T_i} nyelvek nem rekurzívak, csak azt kell látnunk, hogy az itt felsorolt T_i tulajdonságok nem triviálisak, vagyis hogy van T_i tulajdonságú és nem T_i tulajdonságú rekurzívan felsorolható nyelv is.

- T_1, T_2, T_3, T_4, T_6: az $\emptyset$ és Σ^* nyelvek rekurzívan felsorolhatóak és pontosan egyikük rendelkezik a T_i tulajdonsággal
- T_5: $\emptyset$ és például $\{\mathtt{a}^n\mathtt{b}^n \mid n \geq 1\}$ nyelvek mutatják, hogy ez nem triviális tulajdonság, mivel könnyen belátható, hogy az $\{\mathtt{a}^n\mathtt{b}^n \mid n \geq 1\}$ nyelvre van Turing gép
- T_7: az $\emptyset$ és L_u nyelvek mutatják, hogy T_7 nem triviális tulajdonság

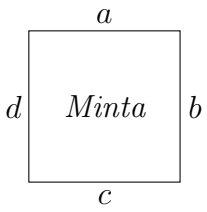
10.5. Dominó probléma

Eddig csupa olyan nyelvet vizsgáltunk és mutattuk meg róluk, hogy nem rekurzívak, melyek szorosan kötődtek definíciójukban is a Turing gépekhez. A fejezet hátralevő részében két további nyelvet vizsgálunk, melyek „hétköznapibbak”, kevésbé elvontak. Látni fogjuk, hogy ezek sem lesznek rekurzívak, ami azt mutatja, hogy nem csak az absztrakt módon definiált nyelvek között találunk algoritmikusan nem eldönthető kérdéseket.

A *dominó probléma* egy olyan sík lefedési feladat, ahol a teljes síkot kell megadott négyzet alakú építőelemekkel (dominókkal vagy inkább csempékkel) lefednünk.

10.36. Definíció *Egy* dominó *egy egységnyi oldalhosszúságú négyzet.*

A dominó típusa *egy* (a, b, c, d) *négyessel írható le, amely a dominó oldalainak típusát írja le a felső oldallal kezdve az óramutató járásának megfelelő körüljárási irányban.*



Egy dominókészlet *véges sok dominótípust tartalmaz, de mindegyik típusból megszámlálhatóan végtelen sok dominót.*

Azt a problémát fogjuk vizsgálni, hogy egy adott dominókészlettel le tudjuk-e fedni az egész síkot úgy, hogy a dominók egymás mellé kerülő oldalai azonos típusúak legyenek (a dominókat nem szabad elforgatni). Tehát például egy (a, b, c, d) típusú dominó mellé jobb oldalt csak olyat rakhatunk, melynek a bal oldala b típusú.

10.37. Példa *Nézzük meg, az alábbi készletek alkalmasak-e a sík lefedésére!*

- $\{(a, a, a, a)\}$
- $\{(a, b, b, b), (a, a, b, b)\}$

Az első készlettel le tudjuk fedni a síkot, hiszen bármely két dominó egymás mellé illeszthető bármilyen irányból, míg a másodikkal nem, mivel semelyik két dominót sem tudjuk egymás alá rakni (a teteje mindegyiknek a típusú, míg az alja b típusú).

Egy adott dominókészlet leírása véges hosszú karaktersorozat, ezért tekinthetjük egy szónak, és vizsgálhatjuk azt a nyelvet, amely a sík kirakására alkalmas készleteknek megfelelő szavakból áll. Az a kérdés, hogy ez a nyelv rekurzív-e lényegében azzal egyenértékű, hogy lehetséges-e olyan, mindig véget érő algoritmust adni, ami tetszőleges dominókészletről megmondja, hogy vele a sík kirakható-e.

A következőkben ezt a kérdést járjuk körül.

10.38. Definíció *A* dominó nyelv *azon dominókészletek halmaza, amelyekkel a dominó probléma megoldható.*

$$D = \{\ \textit{a sík kirakására alkalmas dominókészletek}\ \}$$

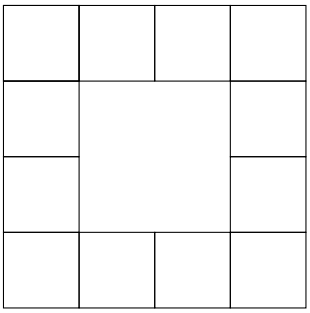
A lefedhetőségnek egy érdekes jellemzését adja a következő lemma.

10.39. Lemma *Egy dominókészlettel a sík akkor és csak akkor rakható ki, ha minden* $2k \times 2k$*-as méretű négyzet kirakható vele* ($k = 1, 2, \dots$).

Bizonyítás. Ha a sík kirakható, akkor nyilván minden négyzet is, ez az irány triviális.

A másik irányhoz azt kell megmutatni, hogy ha minden páros oldalhosszú négyzet kirakható, akkor az egész sík is. Ennek belátására definiálunk egy gyökeres fát. A gyökér fiai a 2×2-es négyzetek kirakásai. A feltétel szerint a gyökérnek legalább egy fia van, de azt is tudjuk, hogy a fiúk száma legfeljebb t^4, ha t jelöli a készletben a típusok számát.

Egy 2×2-es négyzetnek megfelelő csúcs fiai legyenek az ő 4×4-es lehetséges kiterjesztései, melyeket a 2×2-es kirakásból úgy kapunk, hogy köré rakunk egy réteg dominót.



A következő szinten egy 4×4-es csúcsnak a hasonló módon kapott 6×6-os négyzetekre való kiterjesztései (amikor is a szülőnek megfelelő négyzet van középen, e köré rakunk egy réteg dominót, amennyiben ez lehetséges) lesznek az ő fiai, és így tovább.

Ez egy gyökeres szintezett fa, melyben minden szinten véges sok csúcs van (mert egy adott méretű négyzetnek csak véges sok féle kirakása lehet), de minden szinten van legalább egy csúcs, tehát összesen végtelen sok (megszámlálhatóan végtelen sok) csúcsa van ennek a fának. Az is teljesül, hogy minden nem-gyökér elemnek van apja (a közepének megfelelő négyzet).

A sík egy kirakásához induljunk ki a gyökérből. Mivel véges sok fia van, biztosan van közöttük olyan, amelynek részfája végtelen sok csúcsot tartalmaz. Válasszunk egy ilyet, majd tekintsük az ő fiait. Mivel belőlük is csak véges sok van ezek között is biztosan van olyan, melynek a részfája végtelen sok elemet tartalmaz. Ezt az eljárást akármeddig folytathatjuk, a bejárás sosem akad el, és végül egy, a gyökérből induló, végtelen hosszú utat eredményez. Ez az út megadja az egész sík egy kirakását, egy megfelelő 2×2-es négyzetből kiindulva, mindig a választott fiúnak megfelelően újabb és újabb réteget rakhatunk a már meglevő négyzetünk köré. Az eljárással a sík minden pontjára megkaphatjuk, azt milyen dominó fedi. □

Most már készen állunk rá, hogy belássuk a következő tételt.

10.40. Tétel $D \in \operatorname{co}\mathrm{RE}$

Bizonyítás. A definíciók szerint

$$D \in \operatorname{co}\mathrm{RE} \Leftrightarrow \overline{D} \in \mathrm{RE} \Leftrightarrow \exists M \text{ Turing-gép } : L(M) = \overline{D}$$

Azt kell tehát megmutatnunk, hogy van olyan Turing-gép, ami pontosan azokat a szavakat fogadja el, melyek nem megfelelő készleteknek (vagy nem is készleteknek) felelnek meg. Működjön M a következőképpen:

- Ha a bemenet nem egy dominókészlet, akkor megáll és elfogad. (Ez a döntés véges sok lépésben meghozható a dominókészletek kódolásának ismeretében.)
- Ha a bemenet egy dominókészlet leírása, akkor M sorban az összes $k = 1, 2, \ldots$ esetre kipróbálja az összes lehetőséget a $2k \times 2k$-as négyzet lefedésére. Ha valamely k-ra ezeket végignézve nem talál megoldást, akkor M megáll és elfogadja a bemenetet.

Amennyiben az adott dominókészlettel a sík nem rakható ki, akkor, a fenti lemma értelmében biztosan van olyan k amire M nem talál jó kirakást. Mivel minden négyzet lefedésére csak véges sok lehetőséget kell kipróbálni, M el fog jutni a legkisebb ilyen k-hoz, és miután itt ellenőrizte az összes lehetőséget, megáll és elfogad.

Ezzel szemben, ha egy készlettel a sík kirakható, akkor M minden k-nál fog találni egy jó kirakást, és ezért sosem áll meg.

Így tehát M valóban a $\overline{D}$ nyelvet fogadja el. □

A D nyelvről többet is lehet mondani, de az alábbi állítást itt nem bizonyítjuk be.

10.41. Tétel $D \notin \mathrm{R}$

10.42. Következmény $D \notin \mathrm{RE}$

Bizonyítás. Mivel a 10.27. tétel alapján $\mathrm{R} = \mathrm{RE} \cap \mathrm{co\,RE}$ és az előzőek szerint $D \notin \mathrm{R}$, viszont $D \in \mathrm{co\,RE}$, ezért D nem lehet rekurzívan felsorolható. □

A dominó nyelv tehát egy újabb (a diagonális nyelvnél kevésbé absztrakt) példa nem rekurzívan felsorolható nyelvre.

10.6. Post megfeleltetési problémája

A most tárgyalandó probléma nagy hasznunkra lesz majd a 12. fejezetben, amikor környezetfüggetlen nyelvtanokkal kapcsolatos kérdések bonyolultságát vizsgáljuk.

10.43. Definíció *A* Post megfeleltetési problémában *adott véges sok,* Σ *ábécé feletti szópár:*

$$\{(s_i, t_i) : s_i \in \Sigma^*, t_i \in \Sigma^*, i = 1, 2 \ldots k\}.$$

Kérdés, hogy van-e olyan nem üres (esetleg ismétlődéseket is tartalmazó) $i_1, i_2, \ldots i_n$ *indexsorozat, amelyre*

$$s_{i_1} s_{i_2} \ldots s_{i_n} = t_{i_1} t_{i_2} \ldots t_{i_n}$$

10.44. Példa

- *A szópárok:* $\{(\texttt{ab}, \texttt{aba}), (\texttt{ab}, \texttt{ba}), (\texttt{aba}, \texttt{ba})\}$.
  Ekkor az 1, 3 *indexsorozat megfelelő, hiszen* $s_1 s_3 = \underbrace{ab}_{1}\underbrace{aba}_{3}$, *ami megegyezik a* $t_1 t_3 = \underbrace{aba}_{1}\underbrace{ba}_{3}$ *szóval.*

- $\{(1, 111), (10111, 10), (10, 0)\}$
  Könnyen látszik, hogy a 2, 1, 1, 3 *egy megoldás,* $s_2 s_1 s_1 s_3 = \underbrace{10111}_{2}\underbrace{1}_{1}\underbrace{1}_{1}\underbrace{10}_{3}$ *megegyezik a* $t_2 t_1 t_1 t_3 = \underbrace{10}_{2}\underbrace{111}_{1}\underbrace{111}_{1}\underbrace{0}_{3}$ *szóval.*

- $\{(\texttt{ab}, \texttt{a}), (\texttt{ab}, \texttt{ba}), (\texttt{b}, \texttt{ba})\}$
  Figyeljük meg, hogy ebben az esetben nincs jó indexsorozat, mert a szópárok első tagja mindig b*, míg a második tagok mindig* a *karakterre végződnek, ezért bárhogyan is próbálkozunk, az s-ekből és a t-kből képzett szavak vége nem lesz azonos (az indexsorozatnak nincs megfelelő utolsó eleme).*

10.45. Definíció *A* Post megfeleltetési probléma nyelve

$$\mathrm{PCP} = \{\ \textit{azon}\ \{(s_i, t_i) : i = 1, 2 \ldots k\}\ \textit{szópár-halmazok, melyekre van jó indexsorozat}\ \}$$

10.46. Tétel $\mathrm{PCP} \in \mathrm{RE} \setminus \mathrm{R}$

Bizonyítás. A tételnek itt csak a könnyebb részét vázoljuk, azt, hogy $\mathrm{PCP} \in \mathrm{RE}$. Ehhez elegendő egy M Turing-gépet mutatni, melyre $L(M) = \mathrm{PCP}$.

Az M Turing-gép működjön az alábbiak szerint:

- M először ellenőrzi a bemenetet, hogy az valóban néhány szópárból áll-e (ez egyszerűen megoldható). Ha a bemenet szintaktikailag nem megfelelő, akkor M megáll, elutasít.

- Ha a bemenet szópárok sorozata, akkor M előbb kipróbálja az összes lehetséges egy hosszú indexsorozatot (indexet), majd a kettő hosszú, három hosszú, stb. indexsorozatokat. Minden rögzített hossznál véges sok lehetséges indexsorozat van (n hossznál k^n). Ha ezek között talál megoldást, akkor M megáll és elfogad, ha pedig nem talál, akkor növeli a kipróbálandó indexsorozat hosszát.

Így, ha van megoldása a problémának, akkor M véges lépésben talál egyet, és elfogadja a bemenetet. Amennyiben viszont nincs megoldás, akkor egyre hosszabb sorozatokat ellenőriz, de soha nem fog megállni, azaz az általa elfogadott nyelv tényleg a PCP nyelv.□

11. fejezet

A bonyolultságelmélet alapjai

Az eddigi fejezetekben azzal a kérdéssel foglalkoztunk, hogy mely nyelveket lehet Turing-géppel felismerni illetve mely függvényeket lehet Turing-géppel kiszámolni. Ha egy nyelv illetve függvény algoritmikusan felismerhető illetve kiszámolható, akkor a következő, természetesen adódó kérdés az, hogy mennyi ideig tart és mennyi tárhelyet igényel a felismerés illetve kiszámolás.

A függvényeket és nyelveket osztályokba lehet sorolni az alapján, hogy mennyi idő alatt (vagy mekkora tár felhasználásával) lehet őket felismerni illetve kiszámolni. Ebben a fejezetben ezen osztályokat és a köztük levő kapcsolatokat fogjuk megvizsgálni.

11.1. A számítás költsége

Először definiáljuk, hogy mit értünk egy Turing-gép idő- és tárigényén. Az igényt mindkét esetben az input hosszának függvényében adjuk meg, az input méretét hagyományosan n-nel jelöljük.

Azt vizsgáljuk, hogy mekkora a lehetséges legtöbb lépés illetve legtöbb használt cella az adott hosszúságú inputokon vett futásokat tekintve (vagyis hogy a legrosszabb esetben mi történhet egy n méretű input esetén).

11.1. Definíció *Az M determinisztikus Turing-gép* időigénye *a $T_M(n)$ nemnegatív függvény, ahol $T_M(n)$ az n hosszú bemeneteken vett futások leghosszabbikának lépésszáma. Ha a gép valamelyik n hosszú bemenetén nem áll meg, akkor ezen n-re $T_M(n) = \infty$.*

Hasonlóan, jelöljük az M gép tárigényét *$S_M(n)$-nel, ahol $S_M(n)$ azt adja meg, hogy az n hosszú bemeneteken M legfeljebb mennyi szalagcellát használ fel. Az egyszalagos Turing-gépnél ez alatt azt értjük, hogy az egyetlen szalagon hány olyan cella van, ahol vagy járt a gép (eljutott odáig) vagy az input valamely karaktere oda fel van írva. Többszalagos gép esetén a bemeneti (és az esetleges további csak olvasható) szalagok nem számítanak, csak a munkaszalagok: minden munkaszalagra megnézzük, hogy meddig jutott el a gép és ezeket az értékeket összeadjuk.*

Ha M nemdeterminisztikus Turing-gép, akkor $T_M(n)$, illetve $S_M(n)$ a maximális lépésszámot, illetve a legtöbb felhasznált cella számát adja meg az n hosszú bemenetek számítási fáinak összes ága közül.

11.2. Megjegyzés *$S_M(n)$ értéke is lehet ∞, amennyiben a gép valamelyik n hosszú bemenetre soha nem áll meg, hanem pl. folyamatosan gyalogol jobbra a munkaszalagon. Az is lehetséges azonban, hogy a gép úgy kerül végtelen ciklusba, hogy közben a felhasznált cellák száma véges, azaz abból, hogy $S_M(n)$ sehol sem végtelen, még nem következik, hogy a gép minden bemeneten megáll.*

11.3. Megjegyzés *Ha M nemdeterminisztikus gép, akkor $T_M(n)$ értéke olyankor is végtelen, ha csak egyetlen számítási ág is van valamelyik n hosszú inputhoz, ami nem ér véget.*

Általában $T_M(n)$ és $S_M(n)$ pontos meghatározása még könnyen átlátható működésű gépek esetén is nehéz, ezért pontos megadás helyett megelégszünk felső becsléssel is. Ez az eljárás hasonló ahhoz, amit korábban algoritmusok lépésszámának becslésekor követtünk: ott is nehéz volt meghatározni (és sokszor nem is igazán érdekes), hogy pontosan hány lépést tesz egy algoritmus. Az algoritmusok összehasonlításához elég volt az, hogy jó felső becsléseket tudtunk adni a lépésszámokra. Most ugyanezt a stratégiát követjük, ami nem is meglepő, hiszen a Turing-gépek, a Church–Turing-tézis alapján, maguk a lehetséges algoritmusok.

11.4. Definíció *Legyenek $t(n)$ és $s(n)$ a természetes számokon értelmezett nemnegatív függvények (melyek értéke sehol sem ∞). Az M Turing-gép*

- *$t(n)$* időkorlátos, *ha minden n-re fennáll, hogy: $T_M(n) \leq t(n)$.*
- *$s(n)$* tárkorlátos, *ha minden n-re fennáll, hogy: $S_M(n) \leq s(n)$.*

11.5. Megjegyzés *Általában olyan gépekkel foglalkozunk, amik végigolvassák a bemenetet, ezért legtöbbször $t(n) \geq n$. Azt is feltételezzük általában, hogy $s(n) \geq \log_2 n$ fennáll, mert $\log_2 n$ cella biztosan szükséges ahhoz, hogy az input szalagot címezni tudjuk (amit a kicsit is érdekesebb Turing-gépek biztosan tesznek).*

11.6. Megjegyzés *Ha az M Turing-gép $t(n)$ időkorlátos akármilyen $t(n)$ függvénnyel, akkor $L(M)$ biztosan rekurzív nyelv. A tárkorlátosságra ez nem világos, hiszen a gép akár úgy is kerülhet végtelen ciklusba, hogy csak egyetlen cellát használ fel a munkaszalagon.*

A 9.11. tételben láttuk, hogy lehetséges k szalagos Turing-gépet egyszalagos géppel szimulálni. Akkor említettük, hogy a két gépfajta ekvivalenciája csak akkor áll fenn, ha pusztán arra vagyunk kíváncsiak, hogy mely nyelveket lehet velük felismerni, a felismerés ideje nem számít. A következő tétel azt fogalmazza meg, hogy mit állíthatunk a szimuláló egyszalagos Turing-gép idő- és tárigényéről a szimulált géphez képest.

11.7. Tétel *Minden M k-szalagos Turing-géphez létezik olyan M' egyszalagos Turing-gép, hogy $L(M) = L(M')$, és*

- $T_{M'} = \mathrm{O}({T_M}^2(n))$
- $S_{M'} = \mathrm{O}(S_M(n) + n)$

Bizonyítás.

- A 9.11. tételben adott konstrukcióból világos, hogy M egy lépését M' úgy szimulálja, hogy elmegy a szalag „végéig" (ameddig M valaha is eljutott) és utána visszagyalogol a szalag elejére (közben pár helyen át is írhat értékeket). Mivel a szalag vége legfeljebb $T_M(n)$ messze van, ezért M egy lépését $\mathrm{O}(T_M(n))$ lépésben tudja szimulálni az M' gép. Az M legfeljebb $T_M(n)$ lépés után megáll, így M' lépésszáma $\mathrm{O}({T_M}^2(n))$.
- M' egyszalagos, ezért az input hossza biztosan beleszámít a felhasznált tárba (innen jön a $+n$ tag). Ezen felül M' legfeljebb olyan messzire jut el, amilyen messzire M eljutott valamely szalagján, ami $S_M(n)$-nel felülről becsülhető. □

A fenti tétel azt mutatja, hogy legfeljebb négyzetesen növekszik az időigény, amikor egyszalagos géppel szimulálunk. Ha nem egy-, hanem kétszalagos Turing-géppel szeretnénk szimulálni egy k-szalagos gépet, akkor kedvezőbb felső korlátot kapunk az új gép időigényére. (Ezt a tételt nem bizonyítjuk.)

11.8. Tétel *Minden M k-szalagos Turing-géphez létezik olyan M' kétszalagos Turing-gép, hogy $L(M) = L(M')$ és*

- $T_{M'} = \mathrm{O}(T_M(n) \cdot \log T_M(n))$
- $S_{M'} = \mathrm{O}(S_M(n))$

11.2. Idő- és tárosztályok

Eddig azzal foglalkoztunk, hogy az egyes Turing-gépeknek mennyi a tár- és időigényük. A tár- és időigényt azonban megközelíthetjük a nyelvek felől is: kérdezhetjük, hogy egy adott nyelv esetén milyen tár- és időigényű Turing-géppel lehetséges a felismerés (ha lehetséges egyáltalán). A következő tétel (amit nem bizonyítunk), azt mutatja, hogy konstans szorzókat nem érdemes figyelembe vennünk akkor, amikor egy nyelv időigényéről akarunk valamit mondani.

11.9. Tétel (Lineáris gyorsítás) *Ha az L nyelvhez létezik olyan M Turing-gép, hogy $L(M) = L$ és $T_M(n) \leq c \cdot n$, ahol $c \geq 1$ konstans, akkor minden $\varepsilon \geq 0$ esetén létezik olyan M' Turing-gép is, hogy $L(M') = L$ és $T_{M'}(n) \leq (1 + \varepsilon)n$.*

A bizonyítás lényege, hogy az M' gép egy lépésben az M gép m (ε-tól függő számú) lépését néhány (kicsi konstans számú) lépésben fogja szimulálni. Ezt úgy érhetjük el, hogy M szalag-ábécéjét nagyon felduzzasztjuk. A részletes bizonyításért lásd [7]-et.

Hasonló tétel igaz akkor is, ha az M Turing-gépről nem azt tudjuk, hogy cn időkorlátos, hanem általánosabban az igaz, hogy $f(n)$ időkorlátos valamilyen $f(n)$ függvénnyel. Az időbonyolultság ekkor tetszőleges $\varepsilon \geq 0$ esetén levihető $n + \varepsilon f(n)$-re.

A következő tétel azt a meglepő tényt mutatja, hogy bizonyos nyelvek esetén nem csak konstans szorzó erejéig lehetséges a nyelvet elfogadó Turing-gép gyorsítása, hanem bármely, az adott nyelvet elfogadó Turing-gép tetszőlegesen felgyorsítható.

11.10. Tétel (Gyorsítás) *Létezik olyan $L \in \mathrm{R}$ nyelv, hogy tetszőleges M Turing-géphez, amire $L(M) = L$ és $T_M(n)$ véges minden n-re, létezik olyan M' Turing-gép is, hogy $L(M') = L$ és $T_{M'}(n) = \mathrm{O}(\log T_M(n))$.*

Lássuk most azokat az alapvető definíciókat, melyek segítségével a nyelvek idő- és tárigényéről beszélni tudunk. A definícióban a nyelveket elfogadó Turing-gépek idő- és tárigényénél a konstans szorzótól úgy tekintünk el, hogy csak aszimptotikus (O) viselkedést írunk elő.

11.11. Definíció (Nyelvosztályok)

$$\begin{aligned}
\mathrm{TIME}(f(n)) = \{&L \ \textit{nyelv} : \exists M \ \textit{determinisztikus,} \ \mathrm{O}(f(n)) \ \textit{időkorlátos} \\
&\textit{Turing-gép, hogy } L(M) = L\} \\
\mathrm{SPACE}(f(n)) = \{&L \ \textit{nyelv} : \exists M \ \textit{determinisztikus,} \ \mathrm{O}(f(n)) \ \textit{tárkorlátos} \\
&\textit{Turing-gép, hogy } L(M) = L\} \\
\mathrm{NTIME}(f(n)) = \{&L \ \textit{nyelv} : \exists M \ \textit{nemdeterminisztikus,} \ \mathrm{O}(f(n)) \ \textit{időkorlátos} \\
&\textit{Turing-gép, hogy } L(M) = L\} \\
\mathrm{NSPACE}(f(n)) = \{&L \ \textit{nyelv} : \exists M \ \textit{nemdeterminisztikus,} \ \mathrm{O}(f(n)) \ \textit{tárkorlátos} \\
&\textit{Turing-gép, hogy } L(M) = L\}
\end{aligned}$$

A definíció közvetlen következményeként adódnak az alábbi összefüggések.

11.12. Állítás *Tetszőleges $f(n)$ és $g(n)$ függvény esetén igaz, hogy:*

1. $\mathrm{TIME}(f(n)) \subseteq \mathrm{NTIME}(f(n))$
2. $\mathrm{SPACE}(f(n)) \subseteq \mathrm{NSPACE}(f(n))$
3. $\mathrm{TIME}(f(n)) \subseteq \mathrm{R}$
4. $\mathrm{SPACE}(f(n)) \subseteq \mathrm{RE}$

5. *ha* $f(n) \leq g(n)$ $\forall n$*-re, akkor* $\mathrm{TIME}(f(n)) \subseteq \mathrm{TIME}(g(n))$ *és* $\mathrm{SPACE}(f(n)) \subseteq \mathrm{SPACE}(g(n))$

Bizonyítás. 1.-2. A determinisztikus gépek tekinthetők speciális nemdeterminisztikus gépeknek.

3. Ha a nyelvet elfogadó Turing-gép valamely $f(n)$ függvénnyel $f(n)$ időkorlátos, akkor biztosan megáll, így az elfogadott nyelv rekurzív.

4. Mivel ekkor a nyelvhez van őt elfogadó Turing-gép, ezért a nyelv biztosan rekurzívan felsorolható. A rekurzivitás a definícióból nem következik közvetlenül, de látni fogjuk később, hogy ez is fennáll.

5. A definíció közvetlen következménye. □

Az 5. pont tartalmazását látva természetesen adódik a kérdés, hogy mely esetekben kapunk valódi tartalmazást. A kérdés például az, hogy milyen $f(n)$ és $g(n)$ függvények esetén áll fenn a $\mathrm{TIME}(f(n)) \subset \mathrm{TIME}(g(n))$ valódi tartalmazás, azaz mikor igaz az, hogy nagyobb időigényt megengedve a Turing-gépek számára, több nyelvet tudunk elfogadni. A lineáris gyorsítási tételnél lényegében azt láttuk, hogy konstans szorzó különbség esetén nem kapunk más nyelvosztályt. A következő tétel azt mutatja, hogy bizonyos jól megválasztott (nem-konstans) szorzójú növekedés esetén a gépek által felismerhető nyelvek osztálya bővül. A hierarchia-tételeket meg lehet fogalmazni általánosabban is, mint ahogy mi tesszük, itt csak olyan speciális esetre mondjuk ki ezeket, amihez nincs szükség további jelölések és fogalmak bevezetésére. Mind a tár-, mind az idő-hierarchia tétel bizonyítását mellőzzük, a bizonyítást és az általánosabb alakot lásd például [5, 4] könyvekben.

11.13. Tétel (Idő-hierarchia) *Legyen az f függvény olyan, amihez létezik M kiszámolós Turing-gép, melyre $f_M(n) = f(n)$ és $T_M(n) = f(n)$. (Azaz az M Turing-gép legfeljebb $f(n)$ időben kiszámolja az n inputból $f(n)$ értékét. Ezt úgy nevezzük, hogy az $f(n)$ függvény időkonstruálható.)*
Ekkor $\mathrm{TIME}(f(n)) \subset \mathrm{TIME}(f(n) \cdot \log^2 f(n))$

11.14. Tétel (Tár-hierarchia) *Legyen az f függvény időkonstruálható.*
Ekkor $\mathrm{SPACE}(f(n)) \subset \mathrm{SPACE}(f(n) \cdot \log f(n))$

A hierarchia-tételekben lényeges feltevés, hogy $f(n)$ időkonstruálható. Amennyiben ezt nem tesszük fel, akkor sok csúfság megeshet, igaz például a következő tétel, mely szerint egészen durva idő-igény növekedés esetén sem kapunk újabb elfogadható nyelveket.

11.15. Tétel *Létezik olyan $f(n)$ függvény, melyre* $\mathrm{TIME}(f(n)) = \mathrm{TIME}(2^{f(n)})$.

Ha a tár- és időbonyolultsági osztályok definíciójában szereplő függvényeket speciálisan polinomoknak illetve exponenciális függvényeknek választjuk, akkor nevezetes nyelvosztályokhoz jutunk.

11.16. Definíció

$$\mathrm{P} = \bigcup_{k=1}^{\infty} \mathrm{TIME}(n^k)$$
$$\mathrm{NP} = \bigcup_{k=1}^{\infty} \mathrm{NTIME}(n^k)$$
$$\mathrm{PSPACE} = \bigcup_{k=1}^{\infty} \mathrm{SPACE}(n^k)$$
$$\mathrm{EXPTIME} = \bigcup_{k=1}^{\infty} \mathrm{TIME}(2^{n^k})$$

Az így kapott nyelvosztályok azért is érdekesek, mert robusztusak abban az értelemben, hogy a nyelvosztályba tartozó nyelvek halmaza független attól, hogy milyen gépmodellt használunk az osztály definiálására. Az eddigi (eredeti) definícióban tetszőleges determinisztikus illetve nemdeterminisztikus Turing-gépet megengedtünk, de ha ehelyett pl. csak az egyszalagos gépeket tekintenénk vagy valami más, a Turing-gépektől különböző elméleti modell időigényével definiálnánk a nyelvosztályokat (pl. RAM gép, lásd [7]), akkor is ugyanezekhez a nyelvhalmazokhoz jutnánk.

Mostantól ezeknek a most definiált nyelvosztályoknak a kapcsolatával fogunk foglalkozni. A definícióból illetve a korábban már látott tételekből az alábbi összefüggések következnek:

11.17. Állítás *1.* $\mathrm{P} \subseteq \mathrm{NP}$.

2. $\mathrm{NP} \subseteq \mathrm{EXPTIME}$

3. $\mathrm{NP} \subseteq \mathrm{PSPACE}$

4. $\mathrm{P} \subset \mathrm{EXPTIME}$

5. $\mathrm{P} = \mathrm{co\,P}$

6. $\mathrm{P} \subseteq \mathrm{co\,NP}$

Bizonyítás. 1. $\mathrm{TIME}(n^k) \subseteq \mathrm{NTIME}(n^k)$ fennáll 11.12. alapján, amiből az állítás következik. (Azzal, hogy ez a tartalmazás valódi-e, részletesen fogunk a fejezet második részében foglalkozni. Ezen tartalmazás valódisága a számítástudomány talán legfontosabb nyitott kérdése.)

2. - 3. Amikor a 9.16. tételben megmutattuk, hogy a nemdeterminisztikus Turing-gépek szimulálhatók determinisztikus gépekkel, akkor egy olyan konstrukciót adtunk, melyben a determinisztikus Turing-gép lényegében a nemdeterminisztikus számítási fát járja be. A szimuláció részleteinek pontos kidolgozásával látható (alaposan végig kell gondolni a részleteket), hogy ennek a bejárásnak az időigénye exponenciális, tárigénye pedig lineáris a fa mélységében, ami nem más, mint a szimulált nemdeterminisztikus gép időkorlátja. Ezen két tartalmazás valódiságát nem tudjuk.

4. A tartalmazás a definícióból következik, a valódisága pedig az idő-hierarchia tétel következményeként adódik, itt nem részletezett érveléssel.

5. Az $R = \operatorname{co} R$ összefüggés igazolásánál használt érveléssel látható, hogy tetszőleges $f(n)$ függvény esetén $\mathrm{TIME}(f(n)) = \operatorname{co}\mathrm{TIME}(f(n))$, ahonnan P definíciója alapján azonnal adódik az állítás.

6. Láttuk, hogy $\mathrm{P} \subseteq NP$, amiből $\operatorname{co}\mathrm{P} \subseteq \operatorname{co}\mathrm{NP}$ adódik a komplementer nyelvosztály képzésének definíciója alapján. Felhasználva, hogy $\mathrm{P} = \operatorname{co}\mathrm{P}$, megkapjuk az állítást. □

Kevésbé triviális az itt nem bizonyított alábbi kapcsolat a nemdeterminisztikus és determinisztikus tárosztályok között

11.18. Tétel (Savitch) *Ha* $s(n) \geq \log(n)$*, akkor* $NSPACE(s(n)) \subseteq SPACE(s^2(n))$

A következőkben a tár- és időosztályok közti kapcsolatot fogjuk megvizsgálni.

11.19. Lemma *Legyen* $f(n)$ *tetszőleges függvény. Ekkor* $\mathrm{TIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$.

Bizonyítás. Ha egy nyelv eleme $\mathrm{TIME}(f(n))$-nek, akkor létezik őt felismerő $\mathrm{O}(f(n))$ időkorlátos Turing-gép. Ez a Turing-gép egyben $\mathrm{O}(f(n))$ tárkorlátos is lesz, mert $\mathrm{O}(f(n))$ lépés alatt a gép legfeljebb $\mathrm{O}(f(n))$ tárat tud használni. □

A fenti tétel megfordítása nem igaz, sőt a definíció alapján még az sem látszik, hogy egy $\mathrm{O}(f(n))$ tárkorlátos gép által felismert nyelv rekurzív-e egyáltalán. A következő tételből azonban kiderül, hogy egy nyelv tárbonyolultságát ismerve lehetséges felső becslést adnunk a nyelv időbonyolultságára is.

11.20. Tétel (Tár-idő) *Ha* $L \in \mathrm{SPACE}(s(n))$*, ahol* $s(n) \geq \log n$*, akkor létezik olyan, az* L *nyelvtől függő* $c_L > 0$ *konstans, hogy* $L \in \mathrm{TIME}({c_L}^{s(n)})$.

Bizonyítás. Legyen M egy k-szalagos Turing-gép (feltehetjük, hogy $k > 1$), melynek tárkorlátjára valamely $d > 0$ konstanssal $S_M(n) \leq d \cdot s(n)$ teljesül.

A bizonyítás lényege, hogy M-hez készítünk egy olyan másik Turing-gépet, mely lényegében M-et szimulálja az olyan bemeneteken, amelyekre M megáll, azoknál a bemeneteknél pedig, melyekre M nem áll meg, észreveszi azt, hogy M végtelen ciklusban van és ekkor elutasítva megáll.

A fő kérdés tehát az, hogy honnan tudhatjuk, hogy M végtelen ciklusban van. Ehhez vegyük észre, hogy az M gép futása során M aktuális konfigurációját teljes mértékben leírjuk azzal, ha megadjuk, hogy a gép melyik állapotban van, mi szerepel a munkaszalagok nemüres mezőin, a munkaszalagokon hol állnak a fejek, és hogy hol van a fej az (esetleg csak olvasható) bemeneti szalagon. Ha ezt tudjuk egy adott pillanatban M egy számítása során, akkor (M átmeneti függvényének ismeretében) azt is meg tudjuk mondani, hogy ezután mi fog M-mel történni, vagyis egy ilyen leírás teljes mértékben meghatározza M további működését. Az is igaz tehát, hogy ha egy ilyen leírás ismétlődik, akkor újra és újra ismétlődni fog, vagyis a gép ekkor biztosan végtelen ciklusban van. A bizonyítás alapgondolata az esetleges ismétlődés felismerése.

Vegyük most számításba, hogy hányféle különböző konfigurációja lehet a gépnek egy n hosszú bemenet esetén.

Jellemző	**Lehetséges helyzetek száma**
aktuális állapot	$\lvert Q\rvert$
munkaszalagok tartalma	$\lvert\Gamma\rvert^{S_M(n)} \leq \lvert\Gamma\rvert^{d\cdot s(n)}$
fejek a munkaszalagokon	$S_M(n)^k \leq 2^{d\cdot s(n)}$
fej a bemeneten	$n+1 \leq 2^{2s(n)}$

A becslésekben felhasználtuk, hogy $S_M(n) \leq 2^{S_M(n)}$ és hogy $s(n) \leq \log n$.

Az összes lehetséges konfiguráció számára adódó felső becslés:

$$\lvert Q\rvert \cdot \lvert\Gamma\rvert^{d\cdot s(n)} \cdot 2^{(d\cdot s(n)k} \cdot 2^{2s(n)} \leq \underbrace{\lvert Q\rvert \cdot \left(\lvert\Gamma\rvert^d \cdot 2^{kd+2}\right)^{s(n)}}_{t} = \mathrm{O}(c^{s(n)})$$

A végül kapott becslésben a t-vel jelölt tényezőben szereplő hatvány alapjában minden tag csak az M géptől függ: k a gép szalagjainak, Γ pedig a szalagjeleinek száma, a d konstans pedig a tárbonyolultságból adódik. Ezt az alapot c-vel jelölve azt állíthatjuk tehát, hogy a gépnek legfeljebb $\mathrm{O}(c^{s(n)})$ különféle helyzete lehetséges, azaz legfeljebb $\mathrm{O}(c^{s(n)})$ lépésig tud úgy működni, hogy nem esik végtelen ciklusba. Ha figyeljük a gépet és több lépést tett meg már, mint $\mathrm{O}(c^{s(n)})$, akkor biztosan állíthatjuk, hogy sose fog megállni.

Első ötletünk az lehetne, hogy futtassuk M-et $t+1$ lépésig. Ha eközben valamikor megáll, akkor a szimuláló gép is álljon meg és fogadjon el pontosan akkor, amikor M elfogad. Ha pedig M ennyi lépés alatt nem áll meg, akkor biztosak lehetünk benne, hogy M végtelen ciklusba esett, így a szimuláló gépet állítsuk meg és elutasíthatunk.

Az a gond ezzel a megoldással, hogy szükség van hozzá t kiszámolására, azaz n ismeretében meg kell tudnunk határozni t (n-től természetesen függő) értékét, ez azonban nem mindig lehetséges, hiszen például vannak olyan függvények is, amik nem rekurzívak. Ha t nem ilyen egzotikus függvénye n-nek, akkor a fenti ötlet használható, de ha t nem rekurzív, akkor ez a megoldás nem működik.

Szerencsére a következő megoldás a nem-rekurzív esetben is (tehát mindig) működik. Vegyük M két példányát: M_1-et és M_2-t. M_1-et lépésenként futtatjuk, azaz először egy lépésig futtatjuk, azután csinálunk valamit M_2-vel, majd továbbléptetjük M_1-t a második lépésre, aztán újra csinálunk valamit M_2-vel, majd továbbléptetjük M_1-t a harmadik lépésre, stb.

Ha M_1 az ℓ-edik lépésnél tart, akkor M_2-t a számítás elejéről elindítjuk és $\ell-1$ lépésig hagyjuk futni. (Az ℓ aktuális értékét egy külön szalagon tároljuk és minden iteráció után növeljük eggyel.) M_2 minden lépése után összehasonlítjuk M_1-nek az aktuális (az ℓ. lépés utáni) konfigurációját) és M_2 aktuális helyzetét. Ha ezek megegyeznek, akkor biztos, hogy M_1 már végtelen ciklusban van, ezért elutasítunk. Ha M_1 valamikor megáll, akkor mi is megállunk és aszerint fogadunk el, hogy M_1 elfogadott-e.

Határozzuk meg, hogy legfeljebb hány lépés után jutunk el odáig, hogy biztosan megállunk. Eddig M_1 legfeljebb $t+1$ lépést tehetett, míg M_2 összesen legfeljebb $1+2+\cdots+t = O(t^2)$ lépést. Igaz tehát, hogy összesen legfeljebb $\mathrm{O}(t^2) = \mathrm{O}(c^{2s(n)})$ lépés után biztosan megállunk és pontosan akkor fogadunk el, ha M elfogadott, vagyis $c_L = c^2$ jelöléssel beláttuk, hogy $L \in \mathrm{TIME}({c_L}^{s(n)})$. Vegyük észre azt is, hogy a tárigény nagyságrendileg nem változott: M_1 és M_2 párhuzamos futtatása továbbra is $O(s(n))$ tárat igényel a munkaszalagokon, mert mindkét gép legfeljebb $S_M(n)$ tárat használ, az ℓ lépésszám nyilvántartása pedig egy további szalagon legfeljebb $\log t = O(\log c^{s(n)}) = O(s(n))$ tárban megoldható. □

11.21. Következmény

1. *Tetszőleges $s(n)$ függvény esetén* $\mathrm{SPACE}(s(n)) \subseteq \mathrm{R}$
2. *Tetszőleges $s(n)$ függvény esetén* $\mathrm{SPACE}(s(n)) = \mathrm{co}\,\mathrm{SPACE}(s(n))$
3. $\mathrm{PSPACE} = \mathrm{co}\,\mathrm{PSPACE}$
4. $\mathrm{PSPACE} \subseteq \mathrm{EXPTIME}$

Bizonyítás.

1. A tár-idő tétel bizonyításában konstruált Turing-gép mindig megáll.
2. A tár-idő tétel bizonyításában konstruált (mindig megálló) Turing-gépet módosítsuk úgy, hogy pontosan akkor fogadjon el, ha a szimulált gép elutasított vagy végtelen ciklusba került. Így éppen az eredeti nyelv komplementerét elfogadó gépet kapunk.

3. Az előző pontból és PSPACE definíciójából azonnal adódik.

4. Tegyük fel, hogy $L \in \text{PSPACE}$. Ekkor $L \in \text{SPACE}(n^k)$ fennáll valamilyen k egészre, ahonnan a tár-idő tétel miatt $L \in \text{TIME}(c^{n^k})$ következik. Felhasználva, hogy $c \leq 2^c$ adódik, hogy $L \in \text{TIME}(2^{n^{k+1}}) \subseteq \text{EXPTIME}$. □

11.22. Következmény $\text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXPTIME}$

Ebben a tartalmazási láncban érdekes az, hogy míg a két végén álló osztályról tudjuk, hogy nem egyenlők (11.17. állítás), az nem ismert, a közbenső tartalmazások közül melyik nem valódi.

11.3. Az NP nyelvosztály

Az eddig vizsgált nyelvosztályok közül különösen érdekes az NP nyelvosztály. Ez az osztály az algoritmikus kiszámíthatóság gyorsaságával kapcsolatos alapvető kérdések vizsgálatában játszik jelentős szerepet. A fejezet hátralevő részében először azzal foglalkozunk, hogy milyen (az eredeti definíciótól látszólag különböző) alternatív jellemzését adhatjuk ennek az osztálynak, majd az NP és P osztályok kapcsolatát tárgyaljuk.

A következő tétel lehetővé teszi, hogy ne kelljen Turing-gépekben gondolkodni, amikor az NP nyelvosztályba tartozásról van szó. Emiatt több helyen a most megismerendő jellemzést tekintik az NP nyelvosztály definíciójának, például olyankor, ha a Turing-gépek megkerülésével akarunk erről a fontos nyelvosztályról beszélni. Egy ilyen lehetséges bevezetésért lásd pl. [2]-et.

11.23. Tétel (Tanú tétel) *$L \in \text{NP}$ akkor és csak akkor teljesül, ha található egy olyan $c > 0$ konstans és egy L_1 szópárokból álló nyelv, melyre $L_1 \in \text{P}$ és*

$$L = \{x : \exists \underset{\substack{\uparrow \\ \text{tanú}}}{y}\ , |y| \leq |x|^c, (x, y) \in L_1\}$$

Mielőtt nekilátnánk a tétel bizonyításának, nézzük meg, hogy pontosan miben is áll ez az alternatív jellemzés. A jobb oldalon álló L_1 nyelv szópárokból áll és polinom időben felismerhető. Ez azt jelenti, hogy polinom időben el tudjuk dönteni két szóról, hogy ők jó párt alkotnak-e (az általuk alkotott pár benne van-e L_1-ben). Ezt a kérdést olyan párokat vizsgálva fogjuk majd feltenni, ahol a pár első tagjáról, x-ről el akarjuk dönteni, hogy L-beli-e. A tanú tétel azt mondja, hogy ha az L nyelv NP-beli, akkor tudunk olyan L_1-et találni, hogy egy x pontosan akkor van L-ben, ha van neki nem túl hosszú ($|y| \leq |x|^c$), jó párja, azaz van hozzá olyan y, amivel együtt az (x, y) pár L_1-ben van. Ezt a jó párt szokás x tanújának is nevezni, hiszen a létezése tanúsítja, hogy $x \in L$.

Röviden tehát azt mondja a tanú tétel, hogy L pontosan akkor NP-beli, azaz polinomidőben felismerhető egy nemdeterminisztikus Turing-géppel, ha van olyan polinomidőben eldönthető, szópárokból álló L_1 nyelv, melynek a segítségével egy x szóról eldönthető, hogy L-beli-e: ha létezik hozzá rövid tanú, akkor L-beli, különben pedig nincs L-ben.

Ha nem akarunk Turing-gépeket bevezetni, akkor a Church–Turing-tézis értelmében az $L_1 \in P$ feltételt helyettesíthetjük azzal, hogy polinomidejű algoritmussal L_1 eldönthető, így előállítván egy olyan jellemzését az NP nyelvosztálynak melyben Turing-gépekről szó sem esik.

Most lássuk a tétel bizonyítását.

Bizonyítás. A bizonyítás lényege, hogy x pontosan akkor van L-ben, ha az L-et felismerő nemdeterminisztikus Turing-gépnek van x-et elfogadó számítási ága. Ebben az esetben tekinthetjük ezt a számítási ágat x tanújának. Kissé pontosabban a következő mondható:

$\Rightarrow$ irányban: Ha $L \in \mathrm{NP}$, akkor van olyan M nemdeterminisztikus Turing-gép, amely éppen L-et ismeri fel, azaz pontosan akkor van legalább egy elfogadó számítási ága M-nek az x input esetén, ha x benne van L-ben. A számítási ág lényegében annak a leírása, hogy egy adott helyzetből melyik lehetséges továbblépést választva jut végül elfogadó állapotba a nemdeterminisztikus gép. Az M gép leírásából kiszámolható egy felső korlát arra, hogy egy adott helyzetből hányféle lehetséges továbblépés van (hány lehetséges következő állapot van, hány szalagon hányféle új szalagjel írása történhet meg illetve hány szalagon hányféle irányba mozoghat a fej). Jelöljük ezt a konstanst d-vel. Ha valamilyen sorrendet definiálunk a lehetséges továbblépések között (pl. először jönnek azok a továbblépések, amiknél a gép az első állapotba kerül, ezen belül először azok, amelyeknél az első szalagon az első szalagjelet írjuk, ezeken belül azok amiknél az első szalagon a fej az írás után balra mozog, stb.), akkor minden egyes csomópontban elég megadnunk egy legfeljebb d nagyságú egész számot, hogy tudjuk, melyik továbblépést választotta a gép. Így egy tetszőleges x-hez tartozó számítási út átírható egy legfeljebb $T_M(|x|)$ hosszú, számsorozattá, amiben minden tag 1 és d közti egész szám. Mivel az M gép $O(n^k)$ időkorlátos, ezért egy számítási út megadása legfeljebb $\log d \cdot c_1 \cdot |x|^k \leq |x|^c$ hosszú, alkalmasan nagy c konstanst választva.

Legyen tehát az y tanú egy elfogadó számítási út leírása és L_1 legyen az olyan (x, y) párok halmaza, ahol y egy, az x bemenethez tartozó elfogadó számítási út leírása. L_1 eldöntése lehetséges polinomidőben, hiszen az M gép és a lehetséges továbblépések rendezésére adott szabály ismeretében egyszerűen csak el kell dönteni, hogy y egy lehetséges számítási utat ír-e le, és ha igen, akkor ennek az útnak a végén M valóban elfogad-e.

Az is világos, hogy x-hez pontosan akkor tudunk jó párt, jó tanút találni, ha van elfogadó számítási út, azaz amikor $x \in L$.

$\Leftarrow$ irányban: Most egy olyan M' nemdeterminisztikus Turing-gépet kell L-hez mutatnunk, amely polinom időigényű.

Mivel $x \in L$ pontosan akkor teljesül, ha van hozzá legfeljebb $|x|^c$ hosszú y, amivel $(x, y) \in L_1$ és az L_1-be tartozás könnyen ellenőrizhető, ezért egy jó stratégia $x \in L$ eldöntésére a következő: az összes potenciális (legfeljebb $|x|^c$ hosszú) tanúval összepárosítjuk x-et és mindegyik párról megkérdezzük az L_1 nyelvet polinomidőben eldöntő Turing-gépet. Ez így egy determinisztikus (a potenciális tanúk száma miatt exponenciális idejű) algoritmus lenne, de mi most nemdeterminisztikus polinomidejű gépet akarunk szerkeszteni.

Ezt úgy tehetjük meg, hogy a nemdeterminisztikus M' gép x ismeretében először nemdeterminisztikusan generál egy legfeljebb $|x|^c$ hosszú y tanút (ez y hossza miatt polinomidőben megtehető), aztán az így kapott (x, y) párra lefuttatja az L_1 nyelvet polinomidőben elfogadó Turing-gépet. (Vagyis a számítási ágak a számítási fa tetején tartalmaznak csak elágazásokat, aztán minden ág vége egy elágazás nélküli út.) Mivel az L_1-et elfogadó gép futási ideje $O((|x| + |y|)^k)$ valami k egésszel és $|y| \leq |x|^c$, ezért az L_1-et elfogadó gép futtatása polinomidejű $|x|$-ben is, vagyis M' egész futása legfeljebb polinom sok lépésben véget ér.

Világos a konstrukcióból, hogy M'-nek éppen L szavaira lesz elfogadó számítása ága, vagyis M' az L nyelvet ismeri fel. □

Most nézzünk néhány példát NP-beli nyelvekre a tanú tétel felhasználásával. A példákhoz szükségünk lesz arra, hogy gráfokat bináris szavakként kódoljunk. A kódolásnak nem nagyon van jelentősége, lényegében bármi értelmes, nem túlságosan pazarló leírás jó. Tegyük most ezt úgy, hogy egy n csúcsú egyszerű gráf $n \times n$-es négyzetes adjacenciamátrixát sor-folytonosan leírjuk, így a gráfot egy n^2 hosszú bináris szóvá alakítjuk.

11.24. Példa *Legyen* H *a Hamilton-kört tartalmazó gráfokat kódoló szavak nyelve.* H$\in$ NP*, mert L_1-nek választható az olyan szópárok halmaza, ahol a pár első tagja egy gráf leírása, a második tag pedig a gráf csúcsainak egy olyan permutációja, ami Hamilton-kört alkot. (A csúcssorozat leírása történhet a* $\log n$ *hosszú bitsorozatok egymás után írásával.)*

Világos, hogy $L_1 \in$ P*, mert azt könnyű eldönteni egy adott gráfról és adott egy permutációról, hogy ez a permutáció valóban egy Hamilton-kör-e: le kell ellenőrizni, hogy minden csúcs pontosan egyszer szerepelt és hogy a szomszédos csúcsok közt valóban fut él a gráfban, ez egy n csúcsú gráf esetén $O(n)$ lépéses algoritmussal (és Turing-géppel is polinom időben) megtehető.*

Az is világos, hogy csak akkor van egy x gráfleíráshoz jó pár, ha a gráfban van Hamilton-kör. Azt kell még belátnunk, hogy a tanú mérete polinomiális a gráf méretéhez képest, de ez is teljesül, hiszen a gráf leírása n^2, a tanú leírása pedig $O(n \cdot \log n)$ méretű.

11.25. Példa *Legyen* 3SZÍN *a három színnel színezhető gráfokat kódoló szavak nyelve.* 3SZÍN$\in$ NP*, mert L_1-nek választható az olyan szópárok halmaza, ahol a pár első tagja*

egy gráf leírása, a második tag pedig a gráf egy jó színezésének megadása például úgy, hogy sorban felsoroljuk, hogy melyik csúcs melyik színt kapta (pl. 00 =*piros* , 01 = *sárga,* 11 = *kék).*

$L_1 \in$ P, *mert könnyű eldönteni egy adott gráfról és egy adott színezésről, hogy ez a gráfnak egy jó színezése-e: miután megbizonyosodtunk róla, hogy a pár második tagja egy színezés kódja, csak azt kell eldöntenünk minden élre, hogy a végpontjai különböző színt kapnak-e, ez pedig polinom időben (pl. egy* $O(n^2)$ *lépésű algoritmussal) megtehető.*

Az is világos, hogy csak akkor van egy x gráfhoz jó pár, ha a gráfnak van jó színezése. A tanú mérete polinomiális a gráf méretéhez képest, hiszen n-szer konstans bittel a színezés megadható.

11.26. Megjegyzés *Korábban (a nemdeterminisztikus Turing-gépek determinisztikussal való szimulációját felhasználva) láttuk már, hogy* NP $\subseteq$ EXPTIME. *A tanú tétel bizonyításának elvét követve egy újabb bizonyítást adhatunk erre a tényre. Ennek lényege a következő: egy x szó pontosan akkor van benne az L* NP*-beli nyelvben, ha van hozzá* $\leq |x|^c$ *hosszú tanú. Mivel egy tanú ellenőrzése polinomidőben lehetséges és exponenciális sok potenciális tanút kell ellenőriznünk, összesen exponenciális sok lépésben eldönthető egy x szóról, hogy van-e jó pár hozzá, azaz hogy x L-beli-e.*

Nem ismert, hogy az NP $\subseteq$ EXPTIME tartalmazás valódi tartalmazás-e vagy esetleg a két osztály egybeesik. Hasonlóan nem ismert, hogy a P $\subseteq$ NP tartalmazás valódi-e, ez a számítástudomány egyik legnagyobb nyitott kérdése, ezzel foglalkozunk a következő részben.

11.4. NP-teljesség

A P-beli nyelvek azok, amelyeknél van gyors (polinomidejű) algoritmus annak eldöntésére, hogy egy x szó a nyelvhez tartozik-e. Ez azt jelenti, hogy megkapva egy x szót, minden további segítség nélkül el tudjuk dönteni, hogy benne van-e a nyelvben vagy sem.

A tanú tétel alapján NP azon nyelvek osztálya, amiknél a nyelvbe tartozó x szavakhoz létezik a nyelvbe tartozásra rövid (polinomidejű), gyorsan (polinomidőben) ellenőrizhető tanú. Ez egy sokkal gyengébb követelménynek tűnik, mint az, amit a P-be tartozásnál megköveteltünk: az NP esetén nem kell tudnunk gyorsan (polinomidőben) megtalálni a létező rövid, gyorsan leellenőrizhető tanút, csak annyi a követelmény, hogy ha x benne van az L-ben, akkor létezzen ilyen.

Szemléletesen, bár kissé pongyolán úgy is mondhatjuk, hogy egy nyelv akkor van P-ben, ha tetszőleges x esetén a nyelvbe tartozást gyorsan el tudjuk dönteni, míg egy nyelv akkor van NP-ben, ha megsejtve (ajándékba kapva, megálmodva, valami manó megsúgja, stb.) egy bizonyítékot a nyelvbe tartozásra, a bizonyíték gyorsan leellenőrizhető.

Ha P = NP igaz lenne, ez azt jelentené, hogy ugyanolyan nehéz kitalálni egy jó bizonyítékot, mint leellenőrizni azt. Ez hihetetlennek tűnik, de nem ismert bizonyítás

arra, hogy P $\neq$ NP (ahogyan arra sem, hogy P = NP igaz lenne). Az általánosabban elfogadott sejtés szerint P $\neq$ NP, de nincs teljes egyetértés ebben a kérdésben.

A fejezet hátralevő részében azt vizsgáljuk, hogy min múlik ennek a két híres nyelvosztálynak a kapcsolata. Ez az elméleti érdekességen kívül azzal a haszonnal is jár majd, hogy osztályozni tudjuk az NP-beli nyelveket nehézségük szerint és képesek leszünk meghatározni NP-beli szupernehéz nyelvek egy olyan osztályát, amelyekre sejtésünk szerint nem léteznek polinomidejű algoritmusok. Ez azért is fontos, mert ha egy ilyen nyelvre kellene a nyelvbe tartozást eldöntő algoritmust szerkesztenünk, akkor legalább tudjuk majd, hogy mivel állunk szemben.

Az NP-beli nyelvek vizsgálatakor hasznos lesz a következő fogalom. A definíció nem csak NP-beli nyelvekre alkalmazható, bár mi főleg ebben a körben fogjuk használni.

11.27. Definíció (Karp-redukció) *Legyen $L_1, L_2 \subseteq \Sigma^*$ két nyelv. Az L_1 nyelv Karp-redukálható az L_2-nyelvre, ha létezik olyan $f : \Sigma^* \to \Sigma^*$ polinom időben számolható függvény, melyre*

$$x \in L_1 \Leftrightarrow f(x) \in L_2$$

A továbbiakban ezt $L_1 \prec L_2$-vel jelöljük és azt mondjuk, hogy L_1 visszavezethető vagy Karp-redukálható L_2-re. Magát az f függvényt az L_1 nyelv L_2 nyelvre történő Karp-redukciójának nevezzük.

Lássunk most egy példát Karp-redukcióra.

11.28. Példa *A* 3SZÍN*nyelvhez hasonlóan definiálhatjuk a* 4SZÍN *nyelvet is: ez azon irányítatlan gráfok kódjából áll, amiknek csúcsait ki lehet négy színnel színezni úgy, hogy a szomszédos csúcsok különböző színt kapjanak. Most egy Karp-redukciót fogunk mutatni a* 3SZÍN *nyelvről a* 4SZÍN *nyelvre, azaz azt fogjuk megmutatni, hogy* 3SZÍN$\prec$ 4SZÍN*. A Karp-redukció megadása azt jelenti, hogy olyan f függvényt kell mutatnunk, ami minden Σ^*-beli szóhoz egy másik Σ^*-beli szót rendel, mégpedig úgy, hogy*

1. *létezik olyan k konstans, hogy $f(x)$ kiszámolása $O(|x|^k)$ időben megtehető,*
2. *ha x egy 3 színnel színezhető gráf kódja, akkor $f(x)$ egy 4 színnel színezhető gráf kódja,*
3. *ha $f(x)$ egy 4 színnel színezhető gráf kódja, akkor x egy 3 színnel színezhető gráf kódja.*

Fontos, hogy f-nek minden Σ^*-beli szóhoz rendelnie kell valamit, akkor is, ha x egyáltalán nem kódol gráfot. Ez szerencsére nem okoz gondot, a kódolás ismeretében el tudjuk dönteni egy tetszőleges x-ről, hogy az kódol-e gráfot (megpróbáljuk visszafejteni a kódolást) és ha nem, akkor hozzárendelünk valami olyan Σ^*-beli szót, ami szintén nem kódol gráfot, pl. x-et saját magát.

Azt is fontos látni, hogy az $x \mapsto f(x)$ hozzárendelés nem függhet attól, hogy amennyiben x gráfot kódol, akkor a gráfot ki lehet-e színezni 3 színnel vagy sem. $f(x)$ kiszámításának anélkül kell megtörténnie, hogy tudnánk, az x által leírt gráf benne van-e a 3SZÍN *nyelvben vagy sem.*

Legyen f a következő függvény. Ha x nem kódol gráfot, akkor rendeljük hozzá saját magát, azaz legyen $f(x) = x$. Ha x gráfot kódol, akkor tekintsük a neki megfelelő G gráfot. Vegyünk hozzá ehhez a gráfhoz egy új csúcsot és kössük össze ezt az új csúcsot minden régivel. Az így kapott gráfot jelöljük G'-vel, G'-nek tehát $n+1$ csúcsa és $e+n$ éle lesz, ha G-nek n csúcsa és e éle volt.

Legyen $f(x)$ ezen G' gráf kódja. Megmutatjuk, hogy ez az f függvény rendelkezik a fenti 3 tulajdonsággal.

1. *A G' adjacenciamátrixa a G mátrixából egy, az átló kivételével csupa egyest tartalmazó sor és oszlop hozzávételével kapható meg. Az ennek megfelelő kód x-ből $O(n^2)$ időben előállítható.*

2. *Ha G 3 színnel színezhető, akkor G' színezhető 4 színnel: a régi csúcsok megkapják a G-beli színüket, az új csúcs pedig a 4. színt kapja.*

3. *Ha G' színezhető 4 színnel, akkor biztos, hogy egy jó 4 színnel való színezésben az új csúcs színe sehol máshol nem szerepel (hiszen mindenkivel szomszédos). Ez azt jelenti, hogy G csúcsait 3 színnel színeztük jól.*

A definíció a nyelvbe tartozási problémák egymásra való visszavezetését formalizálja. Ha $L_1 \prec L_2$ fennáll, akkor egy L_1 nyelvre vonatkozó nyelvbe tartozási kérdést vissza tudunk vezetni egy L_2 nyelvbe tartozási kérdésre a következőképpen: ha tudni akarjuk, hogy $x \in L_1$ fennáll-e, akkor kiszámoljuk $f(x)$-et (ez megtehető, mert $f(x)$ az x ismeretében polinomidőben kiszámolható), aztán kiderítjük, hogy $f(x)$ benne van-e L_2-ben. A feltétel miatt pontosan akkor kapunk *igen* választ itt, amikor $x \in L_1$.

Vagyis ha van egy algoritmusunk az L_2 nyelv eldöntésére és $L_1 \prec L_2$, akkor van algoritmusunk L_1 eldöntésére is. Ennél még több is igaz, amennyiben L_2-ről tudjuk, hogy P-beli vagy NP-beli, akkor ugyanezt állíthatjuk L_1-ről is.

11.29. Állítás

1. *Ha $L_1 \prec L_2$ és $L_2 \in \mathrm{P}$, akkor $L_1 \in \mathrm{P}$*

2. *Ha $L_1 \prec L_2$ és $L_2 \in \mathrm{NP}$, akkor $L_1 \in \mathrm{NP}$*

3. *Ha $L_1 \prec L_2$, akkor $\overline{L_1} \prec \overline{L_2}$*

4. *Ha $L_1 \prec L_2$ és $L_2 \in \mathrm{co\,NP}$, akkor $L_1 \in \mathrm{co\,NP}$*

5. *Ha $L_1 \prec L_2$ és $L_2 \prec L_3$, akkor $L_1 \prec L_3$*

Bizonyítás. 1. $L_1 \prec L_2$ a definíció szerint azt jelenti, hogy létezik olyan $f : \Sigma^* \to \Sigma^*$ polinom időben számolható függvény, melyre $x \in L_1 \Leftrightarrow f(x) \in L_2$. Legyen c olyan konstans, amire az $f(x)$ függvény $O(|x|^c)$ időben számolható.

Tegyük most fel ($L_2 \in$ P alapján), hogy létezik olyan M determinisztikus Turing-gép, ami éppen L_2-t ismeri fel és $O(n^k)$ időkorlátos valamilyen k egész konstanssal. Konstruálni fogunk egy M' Turing-gépet L_1-re, ami $O(n^\ell)$ időkorlátos valamilyen ℓ konstanssal.

M' csinálja a következőt egy x bemeneten: kiszámolja $f(x)$-et, lefuttatja (szimulálja) M-et $f(x)$-en, pontosan akkor fogad el, ha M elfogadott. Mivel $x \in L_1 \Leftrightarrow f(x) \in L_2$, ezért M' biztosan jó választ ad.

Nézzük most meg, hogy miért lesz M' polinomidejű. $f(x)$ kiszámolása megtehető $O(|x|^c)$ lépésben f definíciója alapján, ebből az is következik, hogy maga $f(x)$ sem lehet ennél hosszabb, azaz $|f(x)| = O(|x|^c)$. Az M Turing-gép $O(n^k)$ időkorlátos, azaz M az $f(x)$-en $O(|f(x)|^k) = O((|x|^c)^k) = O(|x|^{k \cdot c})$ ideig fut. Összesen tehát M teljes működése $O(|x|^c) + O(|x|^{k \cdot c}) = O(|x|^{k \cdot c})$ lépésben véget ér, vagyis az $\ell = k \cdot c$ jó választás lesz.

2. A tanú tételt felhasználva lényegében azt fogjuk megmutatni, hogy ha az L_2 nyelvbe tartozásra van tanú, akkor az L_1-be tartozásra is van.

Tegyük fel tehát, hogy L_2-höz létezik olyan $c_2 > 0$ konstans és ${L_2}' \in$ P nyelv, hogy $x \in L_2$ pontosan akkor teljesül, ha létezik olyan y szó, melyre $|y| \leq |x|^{c_2}$ és $(x, y) \in {L_2}'$ is fennáll.

Meg fogjuk mutatni, hogy ekkor L_1-hez létezik olyan c_1 konstans, és ${L_1}' \in$ P nyelv, hogy $x \in L_1$ pontosan akkor teljesül, ha létezik olyan y szó, melyre $|y| \leq |x|^{c_1}$ és $(x, y) \in {L_1}'$ fennáll.

Vegyük észre, hogy ha $x \in L_1$, akkor $f(x) \in L_2$ és ekkor az $f(x)$-hez a tanú tétel miatt létező y tanúja jó lesz x tanújának is; ${L_1}'$ éppen azokból az (x, y) szópárokból álló nyelv, amelyekre $(f(x), y) \in {L_2}'$.

Világos, hogy $x \in L_1$ pontosan akkor áll fenn, ha $f(x) \in L_2$, ez utóbbi pedig pontosan akkor igaz, ha létezik olyan y szó, melyre $|y| \leq |f(x)|^{c_2}$ és $(f(x), y) \in {L_2}'$ is fennáll, vagyis amikor $(x, y) \in {L_1}'$.

Tekintve, hogy $|f(x)| = O(|x|^c)$, mert f polinomidőben számolható, ezért $|y| \leq |x|^{c \cdot c_2}$, azaz x tanúja valóban polinom méretű x-hez képest.

Azt kell még belátnunk, hogy ${L_1}'$ is polinomidőben eldönthető. Legyen M_2 az az $O(n^k)$ időkorlátos Turing-gép, ami eldönti ${L_2}'$-t. Az ${L_1}'$-et eldöntő M_1 Turing-gép csinálja a következőt az (x, y) bemeneten: kiszámolja $f(x)$-et, majd lefuttatja M_2-t az $(f(x), y)$ párra és pontosan akkor fogad el, ha M_2 elfogad.

Világos, hogy M_1 éppen az ${L_1}'$ nyelvet ismeri fel, lépésszáma pedig $O(|x|^c) + O((|x|^c + |y|)^k) = O((|x|^c + (|x|^c)^{c_1})^k)$, ahol az első tag $f(x)$ kiszámolásából adódik, a második tag pedig M_2 futásából az $(f(x), y)$ bemeneten, ezért ${L_1}' \in \mathrm{P}$.

3. Az $L_1 \prec L_2$ Karp-redukciót megvalósító f függvény egyben Karp-redukció $\overline{L_1}$-ről $\overline{L_2}$-re is, mivel $x \in L_1 \Leftrightarrow f(x) \in L_2$ pontosan akkor áll fenn, ha $x \notin L_1 \Leftrightarrow f(x) \notin L_2$.

4. Adódik, felhasználva az előző két állítást és co NP definícióját.

5. Legyen f az L_1 Karp-redukciója L_2-re és jelölje g az L_2 Karp-redukcióját L_3-ra. Ez azt jelenti, hogy f egy $O(n^k)$ időben számolható függvény, melyre $x \in L_1 \Leftrightarrow f(x) \in L_2$, g pedig olyan $O(n^l)$ időben számolható függvény, melyre $y \in L_2 \Leftrightarrow g(y) \in L_3$.

   Megmutatjuk, hogy a $g(f(x))$ kompozíciófüggvény Karp-redukció L_1-ről L_3-ra.

   Kombinálva f és g tulajdonságait kapjuk, hogy $x \in L_1 \Leftrightarrow f(x) \in L_2 \Leftrightarrow g(f(x)) \in L_3$.

   Másrészt $f(x)$ kiszámítása $O(|x|^k)$ lépés (és így $f(x)$ hossza is $O(|x|^k)$), ezután pedig $g(f(x))$ kiszámolása $O(|f(x)|^\ell) = O(|x|^{k \cdot \ell})$, azaz $g(f(x))$ kiszámolása polinomidejű. □

Mint már említettük, $L_1 \prec L_2$ tulajdonképpen azt jelenti, hogy (a Karp-redukciót megvalósító $f(x)$ függvény felhasználásával) egy L_2-t eldöntő algoritmus használható L_1 eldöntésére is. Úgy is mondhatjuk, hogy L_1-et legfeljebb olyan nehéz eldönteni, mint L_2-t (ha eltekintünk az $f(x)$ értékek kiszámolásától). A fenti tétel utolsó pontja, mely a Karp-redukció tranzitivitását fogalmazza meg, szintén összhangban van ezzel az interpretációval: ha L_1 legfeljebb olyan nehéz, mint L_2 és L_2 legfeljebb olyan nehéz, mint L_3, akkor L_1 legfeljebb olyan nehéz, mint L_3.

Természetesen adódik a kérdés ezek után, hogy egy adott nyelvosztályon belül van-e (vannak-e) olyan nyelv(ek), amik legalább olyan nehezek, mint bármelyik, az adott nyelvosztályba tartozó másik nyelv. Ez a kérdés különösen érdekes az NP nyelvosztály esetén, ezt formalizálja a következő definíció.

11.30. Definíció *Az L nyelv* NP-teljes, *ha $L \in \mathrm{NP}$ és minden $L' \in \mathrm{NP}$ esetén $L' \prec L$.*

11.31. Megjegyzés *Ha egy L nyelvről csak azt tudjuk (vagy csak azt akarjuk hangsúlyozni), hogy minden $L' \in \mathrm{NP}$ esetén $L' \prec L$, akkor L-et* NP-nehéznek *nevezzük.*

Az első kérdés, ami a definíció láttán felmerül az az, hogy van-e egyáltalán NP-teljes nyelv. Nemsokára látni fogjuk, hogy léteznek NP-teljes nyelvek. Az is kérdés, hogy hogyan lehet egy L nyelvről megmutatni az NP-teljességet. A definíció szerint ehhez (az NP-beliségen kívül) azt kellene megmutatnunk, hogy minden NP-beli nyelv visszavezethető

L-re, ami első ránézésre nehéznek tűnik, hiszen nem ismerjük az NP-teljes nyelvek egy kimerítő felsorolását. Nemsokára azt is látjuk majd, hogy hogyan lehet ezen a látszólagos nehézségen átlépni.

A jó hír az, hogy ha egy L nyelvről valahogyan belátjuk, hogy NP-teljes, akkor ezen L nyelv segítségével már könnyebben tudjuk más nyelvek NP-teljességét igazolni. Ennek módszerét fogalmazza meg a következő állítás.

11.32. Állítás *Ha az L nyelv* NP*-teljes és L_1 olyan, hogy*

1. $L_1 \in \mathrm{NP}$*, és*
2. $L \prec L_1$

akkor L_1 is NP*-teljes.*

Bizonyítás. Mivel $L_1 \in \mathrm{NP}$, ezért az NP-teljesség első követelménye teljesül. Azt kell tehát csak belátnunk, hogy minden $L' \in \mathrm{NP}$ nyelvre fennáll $L' \prec L_1$. Ez abból következik, hogy L NP-teljessége miatt minden $L' \in \mathrm{NP}$ nyelvre fennáll $L' \prec L$, a feltétel miatt pedig $L \prec L_1$ és a Karp-redukció tranzitív. □

Fontos észrevenni, hogy ha ezt az állítást akarjuk használni, akkor mindig az ismerten NP-teljes nyelvet kell visszavezetnünk arra a nyelvre, aminek NP-teljességét igazolni akarjuk. Ez egyrészt világos a bizonyításból, másrészt a Karp-redukció „legfeljebb olyan nehéz, mint" interpretációjából is adódik. Ha fordítva csinálnánk a visszavezetést, azaz az NP-teljes nyelvre vezetnénk vissza az ismeretlen nyelvünket, az csak azt jelentené, hogy az ismeretlen bonyolultságú nyelvet visszavezettük egy nehéz kérdésre, amiből természetesen semmi nem következik az ismeretlen bonyolultságú nyelv nehézségéről (könnyű kérdéseket is el lehet bonyolítani).

Összefoglalva tehát a következő tennivalóink vannak:

- Valahogyan megmutatni a definíció alapján egy L nyelvről, hogy az NP-teljes.
- Ezen L és a fenti 11.32. állítás felhasználásával újabb nyelvről (nyelvekről) megmutatni az NP-teljességet.
- Minél több nyelvről tudjuk már, hogy NP-teljes, annál könnyebb lesz újabbakról ezt belátni, hiszen annál több már ismerten NP-teljes nyelv játszhatja L szerepét a 11.32.. állításban.

Lássunk neki az első NP-teljes nyelv megismerésének. Ehhez szükségünk lesz néhány definícióra.

11.33. Definíció *Egy* Boole-formula *0 és 1 logikai konstansokból (jelentésük „hamis" és „igaz"), $x_1, \ldots, x_n$ logikai, 0-1 értékű változókból és ezek $\overline{x_1}, \ldots, \overline{x_n}$ negáltjaiból, illetve az $\wedge$ („és") és a $\vee$ („vagy") műveleti jelekkel és zárójelekkel elkészített formula.*

A Boole-formula változóinak valahogyan $0-1$ értéket adva a formula egészére is adódik egy $0-1$ érték. Ilyenkor azt mondjuk, hogy a formulát a változók adott értékeivel kiértékeljük. Ha van olyan kiértékelése (behelyettesítése) a formulának, aminek eredménye „igaz" , akkor a formulát kielégíthetőnek nevezzük.

11.34. Példa *Például* $(x_1 \vee 1) \wedge \overline{x_2}$ *egy Boole-formula. Ennek egy lehetséges jó kiértékelése az* $x_1 = 1, x_2 = 0$ *értékadás.*

Tekintsük most a kielégíthető Boole-formulák nyelvét, erről fogjuk belátni a definíció alapján, hogy NP-teljes. Ennek formális megadásához szükségünk van egy formula például a $\{\,\mathtt{0},\mathtt{1}\,\}$ ábécé feletti szóvá kódolására. Ez lehetséges például úgy, hogy minden, a formulákban szereplő elemet a $\{\,\mathtt{0},\mathtt{1}\,\}$ ábécé feletti szavakkal kódolunk és ezeket a formulának megfelelő sorrendben egymás után fűzzük. A továbbiakban a kódolástól eltekintünk, úgy gondolunk a vizsgált nyelvre, mintha maguk a formulák lennének az elemei.

11.35. Definíció

$$\mathrm{SAT} = \{\varphi(x_1, \ldots, x_n) : \exists b_1, \ldots, b_n \;\; \textit{behelyettesítés, hogy} \;\; \varphi(b_1, \ldots, b_n) = \textit{igaz}\}$$

11.36. Tétel (Cook, Levin) *A* SAT *nyelv NP-teljes.*

Bizonyítás. 1. SAT $\in$ NP, mert egy megfelelő $b_1, \ldots, b_n$ behelyettesítés megadása jó tanú. Ez polinom hosszú (hiszen pontosan olyan hosszú, ahány változó szerepel φ-ben) és polinom időben ellenőrizhető a formula értékének kiszámolásával. Ez utóbbi számolás a formula hosszával arányos.

2. Vázlatos bizonyítást adunk arra, hogy miért lehetséges minden NP-beli nyelvet visszavezetni a SAT-ra.

   Legyen $L \in$ NP nyelv és legyen M olyan egyszalagos, nemdeterminisztikus, polinom időkorlátos Turing-gép, hogy $L(M) = L$ és M az elfogadó állapotból soha nem tud kilépni (Ilyen gép a 10.2. tétel miatt létezik.)

   Az $L \prec$ SAT belátásához meg fogjuk mutatni, hogy M-hez és egy tetszőleges x-hez lehet olyan φ-t készíteni $O(|x|^k)$ időben (ez lesz $f(x)$), hogy $x \in L$ pontosan akkor teljesül, amikor $\varphi \in$ SAT. A Boole-formulának lesznek olyan részei, amelyek csak az M gép működésétől függenek és lesznek olyan részei is, amit az x szótól függően fogunk megkonstruálni.

   A gondolatmenet lényege az, hogy x-hez egy olyan Boole-formulát készítünk el, ami az M gép működését írja le az x inputon és a formula pontosan akkor lesz kielégíthető, ha van M-nek olyan számítása, ami mentén M x-et elfogadja.

   Ehhez tekintsük M-et és vezessünk be a következő logikai változókat:

- x_{ija}: az i-edik lépés után a j-edik szalagcellában a karakter van
- y_{ij}: az i-edik lépés után a fej a j-edik cellán áll
- z_{iq}: az i-edik lépés után az állapot q

Hány ilyen változóra van szükségünk? Az M gép polinom időkorlátos, ezért mind a lépésszámra, mind a felhasznált cellák sorszámára felső korlátunk van, ami polinomiális $|x|$ függvényében. A gépnek véges sok állapota van, ez pedig egy konstans, vagyis összesen legfeljebb $|x|$ hosszához képest polinom sok ilyen változót kell felvennünk, az ezek által felvett értékek együtt meghatározzák a Turing-gép pillanatnyi helyzetét.

Írjuk most fel azt, hogy milyen megkötések, milyen összefüggések megléte szükséges ahhoz, hogy egy érvényes, x inputtal induló, elfogadó számítást kapjunk. Külön formulákat fogunk felírni az egyes követelményekhez és ezeket a formulákat fogjuk egy nagy végső formulába összeéselni.

A részkövetelményeket leíró Boole-formulák azt biztosítják, hogy:

- Adott i-re a z_{iq} változók pontosan egyike lehet igaz
- Adott i-re az y_{ij} változók pontosan egyike lehet igaz
- Adott i-re és j-re az x_{ija} változók ($a \in \Gamma$) közül pontosan egy igaz
- Bekapcsoláskor a kezdőállapotban van a gép, a szalag elején van a fej és a szalagon az x bemenet található
- Egy lépés során az átmeneti függvény szerint változik a gép konfigurációja és ennek megfelelően a változók értékei
- A számítás végén elfogadó állapotban kell lennie a gépnek és meg kell állnia

Nem fogjuk végignézni részletesen, hogy ezek az egyes követelmények hogyan írhatók le Boole-formulákkal, csak néhány példával illusztráljuk az egyes pontokat, ami érzékelteti, hogy mindezen fenti követelmények leírhatók Boole-formulákkal.

- Azt, hogy tetszőleges i-re a z_{iq} változók közül legalább egy igaz, úgy írhatjuk le, hogy minden i-re (polinom sok ilyen van) tekintjük a $z_{iq_0} \vee z_{iq_1} \vee \cdots \vee z_{iq_n}$ formulát, ahol $q_0, \ldots, q_n$ az M állapotai. Azt, hogy legfeljebb egy z_{iq} változó lehet igaz úgy írhatjuk le, hogy i minden értékére és minden q, q' állapotpárra felvesszük a $\overline{z_{iq}} \vee \overline{z_{iq'}}$ kifejezéseket.
- Az y_{ij}, illetve az x_{ija} változókra tett kikötések hasonlóan írhatóak le.
- A fej kezdeti helyzetét és a kezdőállapotot az $y_{01} = 1$ és $z_{0q_0} = 1$ beállításokat leíró $y_{01} \vee 0$ illetve $z_{0q_0} \vee 0$ kifejezésekkel tudjuk megadni. (Az előző pontokban vázolt formulák ezek után már biztosítják, hogy a többi, a kezdőhelyzethez

tartozó állapotot kódoló és fej helyzetet leíró változó értéke csak 0 lehet egy jó kiértékeléskor).

Azt, hogy x van a szalagra írva, úgy adhatjuk meg, hogy az x_{0ja} a bemenet bitjei szerint beállítjuk a megfelelő j pozíciókon (azaz ahol $j \leq |x|$), ugyanazon trükkel élve, amit a kezdőállapot beállításánál is használtunk. A $j \geq |x|$ értékek esetén pedig az $x_{0j\_} = 1$ beállítás szerint adjuk meg a változók értékét.

- Az átmeneti függvény egy lehetséges alkalmazását úgy tudjuk leírni, hogy minden i lépésre és j fejhelyzetre olyan $\varphi_0 \to \varphi_1 \vee \ldots \vee \varphi_m$ formulákat szerkesztünk, ahol φ_0 a lépés előtti konfiguráció leírását tartalmazza (melyik lépés után hol áll a fej, milyen állapotban van a gép, mi van a szalagon), míg a φ_i függvények a különféle lehetséges továbblépéseknek megfelelő helyzeteket írják le. Ilyen típusú részformulából polinom sok lesz, hiszen i és j is legfeljebb polinom-méretű $|x|$-hez képest.
- Azt kell megfogalmaznunk Boole-formulával, hogy az eljárás végén a gép elfogadó állapotban van, ez megint csak egy változó értékének beállítását jelenti.

Ezen logikai kifejezések összeésélésével leírjuk az M Turing-gép működését, és látható, hogy pontosan akkor létezik az x szóhoz olyan számítási ág, mely elfogadó levélben ért véget, amikor létezik egy b_x behelyettesítés (ami lényegében a nemdeterminisztikus választásoknak felel meg), amely kielégíti a fenti logikai kifejezést. Az is világos, hogy a részformulák mérete és darabszáma polinomiális x hosszához képest (ennek végiggondolását az olvasóra bízzuk, az egyes részkifejezésfajták végignézésével ez könnyen látható). □

A SAT nyelv után azt mutatjuk meg, hogy bizonyos speciális alakú kielégíthető Boole-formulák nyelve is NP-teljes. Azokat a Boole-formulákat fogjuk most tekinteni, amelyek véges sok részkifejezés „és” művelettel való összekapcsolásából állnak, ahol az egyes részkifejezések változók, negáltjaik illetve a 0 és 1 logikai konstansok „vagy” művelettel összekapcsolt formulái.

11.37. Definíció *Egy Boole-formula* konjunktív normál formájú*, ha az alábbi alakú:*

$$(x_{i_1} \vee x_{i_2} \vee x_{i_3} \ldots) \wedge (x_{i_j} \vee x_{i_{j+1}} \vee x_{i_{j+2}} \ldots) \wedge \ldots$$

A fenti formulában az x_{i_k}-k változókat, azok negáltjait vagy a 0 *és* 1 *logikai konstansokat reprezentálják.*

11.38. Lemma A *konjunktív normál formájú kielégíthető Boole-formulák nyelve NP-teljes.*

Bizonyítás. A Cook–Levin-tétel fenti bizonyítását kicsit módosítva megmutatjuk, hogy minden NP-beli nyelv visszavezethető a kielégíthető konjunktív normál formájú Boole-formulák nyelvére.

A Cook–Levin-tétel bizonyításában előállított Boole-formula majdnem jó lesz most is: az sok részformula összeéselésével keletkezett, ezen részformulák többsége megfelel a konjunktív normálformára tett feltételnek (azaz csak „vagy" művelet szerepel benne). Az egyetlen kivételt az átmeneti függvény leírására alkotott $\varphi_0 \to \varphi_1 \vee \ldots \vee \varphi_m$ részformulák jelentik, de ezek a De Morgan azonosságok segítségével a kívánt alakra hozhatók, csak azt kell meggondolnunk, hogy közben megmarad a polinomidőben számolhatóság, azaz x hosszához képest polinomidőben ki tudjuk számolni az új formulát. Ez azért lesz így, mert a $\varphi_0 \to \varphi_1 \vee \ldots \vee \varphi_m$ alakú részformulák átírása független x-től (csak az M gép leírása számít), ezért ez $|x|$-től független konstans időben megtehető. □

Most, hogy két nyelvről már megmutattuk az NP-teljességet, sokkal könnyebb dolgunk lesz NP-teljességek igazolásánál, használhatjuk a 11.32. állításban megfogalmazott módszert.

Első alkalmazásként azt mutatjuk meg, hogy a kielégíthető konjunktív normálformájú Boole-formulák nyelvének az a részhalmaza is NP-teljes, amiben olyan formulák vannak csak, ahol minden zárójel legfeljebb három tagú.

11.39. Definíció *A* 3SAT *nyelv a* SAT *nyelv azon részhalmaza, amelyben azok a konjunktív normálformájú Boole-formulák vannak, ahol az „és"-ekkel összekapcsolt zárójelek mindegyikében legfeljebb három tag van.*

11.40. Megjegyzés *Például kielégíthető, konjunktív normál formájú Boole-formula a* $(x_1 \vee x_5) \wedge (\bar{x_2} \vee x_1 \vee x_2 \vee x_3) \wedge x_4 \wedge \overline{x_5}$ *kifejezés, de nincs benne* 3SAT*-ban, mert a második zárójelben négy tag szerepel.*

11.41. Tétel *A* 3SAT *nyelv NP-teljes.*

Bizonyítás. A megfelelő változóbehelyettesítés gyorsan ellenőrizhető, rövid, jó tanú lesz itt is, tehát a 3SAT nyelv NP-beli.

Most megmutatjuk, hogy SAT $\prec$ 3SAT. Ehhez mutatunk egy $\varphi \to \psi$ megfeleltetést, mely polinomidőben (azaz φ hosszához képest polinom sok lépésben) előállít egy olyan ψ formulát, melyre

$$\varphi \in \text{SAT} \Leftrightarrow \psi \in \text{3SAT}$$

A visszavezetés lényegi lépése, hogy az „és" művelettel összekapcsolt zárójeles tagokat át kell alakítanunk, ha bennük háromnál több tag szerepel. Egy példán mutatjuk meg, hogy ezt hogyan lehet megtenni.

Például ha egy zárójeles tag $(a \vee b \vee c \vee d \vee e)$ alakú, ahol a, b, c, d, e változók, ezek negáltjai vagy pedig konstansok, akkor vegyük ezen zárójel helyett a következő kifejezést: $(a \vee r_1) \wedge (\overline{r_1} \vee b \vee r_2) \wedge (\overline{r_2} \vee c \vee r_3) \wedge (\overline{r_3} \vee d \vee r_4) \wedge (\overline{r_4} \vee e)$.

Ebben az új kifejezésben az r_i változók most bevezetett új változók, minden eredeti zárójel feloldásakor újakat fogunk használni belőlük. Világos, hogy egy n hosszú nagy zárójel feloldásakor $n-1$ új változó és n rövid zárójel keletkezik, vagyis az átalakítás polinomidejű, azt kell csak megmutatnunk tehát, hogy az új kifejezés pontosan akkor kielégíthető, ha a régi az volt.

$\Rightarrow$: Ha van egy jó kiértékelése az a, b, c, d, e tagoknak, akkor a következőképpen kaphatunk jó kiértékelést az új formulához. Az eredeti a, b, c, d, e változók értéke legyen ugyanaz, mint az eredeti formulában, az r_i változók pedig a következőképp kapjanak értéket. Mivel a, b, c, d, e közül legalább egy igaz értéket kap tekintsük azt a rövid tagot az új kifejezésben, amiben ez az „igaz" értéket kapó tag szerepel.

Ha ez éppen az $(i+1)$. tag, azaz ebben a tagban $\overline{r_i}$ és r_{i+1} szerepel, akkor a ψ formula egy jó kiértékelését kapjuk, ha i-ig bezárólag minden r_j változót igazzá, $i+1$-től kezdve pedig hamissá teszünk. Ezzel a kiértékeléssel az első $i-1$ tag az r_j változók miatt lesz igaz, az i. tag az ott szereplő nem r_j típusú változótól, az $i+1$. tagtól kezdve pedig a negált r_j-k biztosítják a háromtagú zárójeles kifejezések „igaz" értékét.

$\Leftarrow$: Ha a háromtagú kifejezések mindegyike „igaz" értéket vesz fel, az csak úgy lehet, ha legalább egyikőjük nem az r_j vagy $\overline{r_j}$ típusú tagok miatt lesz „igaz". Ez azért van így, mert egy r_i csak egy zárójeles kifejezést tud igazzá tenni (vagy azt amiben r_j vagy azt amiben $\overline{r_j}$ alakban szerepel), viszont eggyel több zárójeles kifejezés van, mint ahány r_j változó, így legalább az egyik ilyen zárójeles kifejezés „igaz" értéke a középen álló tagból következik. Ez pedig éppen azt jelenti, hogy az eredeti kifejezés legalább egy tagja igaz.

A példa ötletét felhasználva, az összes túl hosszú tagot rövidebbek összeéselt verziójára alakíthatjuk át, így megkapjuk a kívánt ψ formulát. □

Érdekes észrevétel, hogy ha tovább egyszerűsítjük a nyelvet, három helyett csak 2 tagot engedünk meg a zárójeleken belül, akkor már P-beli nyelvet kapunk. Ezt az állítást itt nem bizonyítjuk.

11.42. Definíció *A* 2SAT *nyelv az a részhalmaza a* SAT *nyelvnek, amelyben olyan konjunktív normálformájú Boole-formulák vannak, ahol minden zárójelben legfeljebb két változó van.*

11.43. Tétel $2\text{SAT} \in \text{P}$

A 3SAT nyelv segítségével most ki fogunk lépni a logikai kifejezések világából és egy gráfokkal kapcsolatos nyelvről, a már korábban vizsgált 3SZÍN nyelvről mutatjuk meg, hogy NP-teljes.

11.44. Tétel *A* 3SZÍN *nyelv NP-teljes.*

Bizonyítás. A 3SZÍN nyelv NP-beliségét már beláttuk.

Az NP-teljesség belátásához megmutatjuk, hogy 3SAT $\prec$ 3SZÍN. Ehhez egy tetszőleges ψ formulához konstruálunk egy olyan G gráfot, hogy:

$$\psi \in \text{3SAT} \Leftrightarrow G \in \text{3SZÍN}$$

Olyan, polinom időben számolható leképezést kell tehát alkotnunk, ami minden konjunktív normálformájú, egy zárójelben legfeljebb három tagot tartalmazó formulához egy gráfot rendel, mégpedig úgy, hogy a formula pontosan akkor kielégíthető, ha a hozzárendelt gráf 3 színnel színezhető.

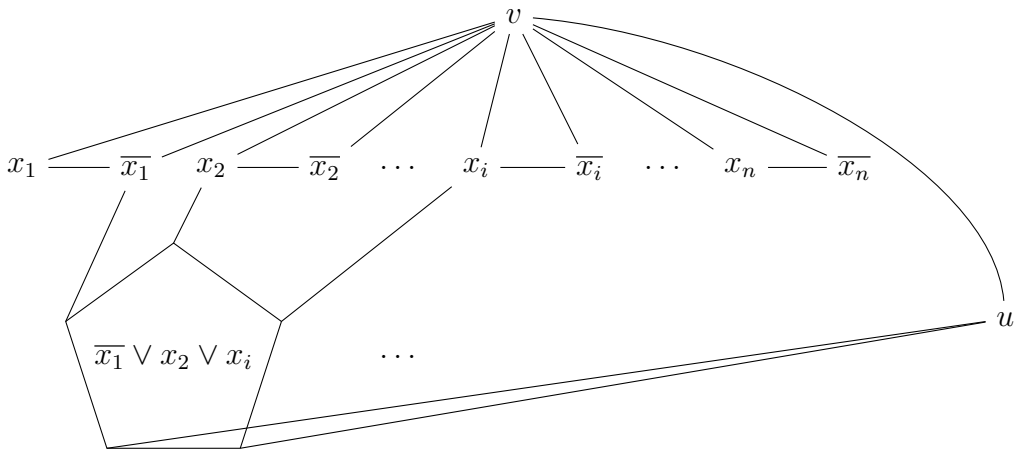
Valójában a nyelvekben nem formulák és gráfok vannak, hanem ezeket kódoló szavak és a leképezésünknek minden szóra valamit ki kell számolnia, akkor is, ha az nem egy konjunktív normálformájú, egy zárójelben legfeljebb három tagot tartalmazó formula kódja. A feladat ezen részével könnyű elbánni: ha egy szó nem kódol formulát, akkor rendeljünk hozzá egy olyan szót, ami nem kódol gráfot. A továbbiakban csak azzal foglalkozunk, hogy mit kell tennünk akkor, ha egy olyan szóhoz kell hozzárendelnünk valamit, ami egy megfelelő alakú formulának felel meg.

Legyenek ψ változói $x_1, x_2 \dots x_n$. Erről a formuláról feltehető, hogy nincsenek benne logikai konstansok, mert ha egy zárójelben szerepel az 1, akkor az biztosan igaz, ezért ez a zárójeles kifejezés elhagyható, ha pedig egy 0 szerepel valahol, akkor ezt a 0-t elhagyhatjuk. Feltehető tehát, hogy a kiindulási formulánkban csak változók és ezek negáltjai szerepelnek. Az is feltehető, hogy minden zárójelben pontosan három tag szerepel. Ha egy zárójelben nincs három tag, akkor valamelyik tagot ismételjük meg, hogy ezt a feltételt teljesítse a ψ formula.

Nézzük, hogy mik lesznek a ψ-hez rendelt gráf csúcsai. Először is felveszünk egy-egy csúcsot $x_1, \overline{x_1} \dots x_n, \overline{x_n}$ számára (azaz minden változónak és a negáltaknak is megfelel egy-egy csúcs), továbbá felveszünk még két csúcsot, u-t és v-t. Rendeljünk ezeken kívül még minden zárójelhez 5-5 új csúcsot.

A G gráf élei a következőképp alakulnak. Minden x_i és $\overline{x_i}$ pontot kössünk össze éllel, továbbá minden $x_1, \dots, x_n$ és $\overline{x_1}, \dots, \overline{x_n}$ pontot kössünk össze v-vel is. Legyen továbbá él u és v között is, valamint az egyes zárójelekhez tartozó öt pontot kössük össze úgy, hogy egy öt hosszú kört alkossanak.

Ezek után minden olyan ötszög esetén, amely egy zárójeles kifejezésnek felel meg, kössük össze az ötszög (tetszőleges) három pontját a megfelelő tagokhoz (változók vagy ezek negáltjaihoz) tartozó csúcsokkal, a másik két pontját pedig u-val.



Világos, hogy G konstruálása polinomidejű a ψ formula méretéhez képest, hiszen, ha ψ-ben n változó és m zárójeles kifejezés van, akkor G-ben $2n + 2 + 5m$ csúcs lesz és $n + 2n + 5m + 5m + 1$ él.

Azt kell még megmutatni, hogy ha ψ kielégíthető, akkor G 3 színnel színezhető és ha G 3 színnel színezhető, akkor ψ kielégíthető.

$\Rightarrow$: ψ egy jó kiértékelése alapján ki fogjuk színezni G-t piros, zöld és lila színekkel. Legyen v lila, az x_i és $\overline{x_i}$ csúcsok közül pedig az legyen zöld, amelyik a ψ jó kiértékelésekor „igaz” értéket kap, a negáltja pedig legyen piros. (Tehát, ha pl. a jó kiértékelésben x_1 igaz, akkor x_1 zöld, $\overline{x_1}$ pedig piros, ha azonban x_1 „hamis”, akkor x_1 piros lesz és $\overline{x_1}$ lesz zöld.

Az u csúcs színe legyen piros. Ki kell még színeznünk az ötszögeket. Egy ötszög színezéséhez 3 színt használhatunk, de úgy, hogy minden csúcsra van egy tiltott szín (az u-val szomszédosokra a piros, a többire pedig vagy a zöld, vagy a piros, aszerint hogy az ötszög megfelelő csúcsához bekötött tag „igaz” vagy „hamis” értéket kapott a ψ jó kiértékelésénél.

Azért tudjuk megvalósítani az ötszögek színezését, mert biztos, hogy nem mind az öt csúcsnál a piros szín lesz letiltva (hiszen ekkor a megfelelő zárójeles kifejezés nem lenne „igaz”). Könnyű végignézni (ezt az olvasóra bízzuk), hogy minden ilyen esetben meg lehet valósítani egy jó színezést.

$\Leftarrow$: Tekintsük most G egy jó piros, zöld és lila színnel való színezését. Legyen ebben v színe lila. Ekkor, mivel minden $x_1 \dots x_n$ és $\overline{x_1} \dots \overline{x_n}$ össze van kötve v-vel és a megfelelő párok is egymással, bármely x_i és $\overline{x_i}$ pár esetén az egyikük zöld, a másikuk pedig piros. Az u csúcs nyilván szintén vagy zöld vagy piros, mert össze van kötve v-vel. Feltehetjük, hogy u piros.

Ekkor ψ egy jó behelyettesítése legyen az, hogy egy x_i akkor kap „igaz” értéket, ha az ő színe zöld és akkor lesz „hamis”, ha az ő színe piros.

Az ötszögek kiszínezhetőségéből következik, hogy az ötszög u-hoz nem kötött pontjai közül legalább az egyik zöld ponthoz van kötve, azaz a „vagy” kapcsolatban lévő változók

közül legalább az egyik igaz. Ez amiatt van, mert ellenkező esetben az ötszög semelyik pontját sem színezhetnénk pirosra, vagyis az ötszög pontjaira csak két szín jutna, ami lehetetlen. □

A 3SZÍN nyelv segítségével számos más, gráfokkal kapcsolatos nyelvről látható be, hogy NP-teljesek. További NP-teljességi bizonyításokért lásd [7]-et.

12. fejezet

Nyelvtanok és Turing-gépek

Az eddigi kiszámíthatósági és bonyolultságelméleti kérdések után visszatérünk a nyelvtanokkal kapcsolatos kérdésekre. Még kétféle dologgal adósok vagyunk. Az egyik, hogy a Turing-gépek által elfogadott nyelvekhez is tartozik-e a nyelvtanoknak egy típusa, amivel pont ezeket a nyelveket lehet generálni. A másik pedig, hogy mi a helyzet azokkal az algoritmikus kérdésekkel, amelyeket a reguláris nyelveknél tárgyaltunk (5. fejezet), de a környezetfüggetleneknél (8. fejezet) nem.

12.1. A 0. és az 1. Chomsky-féle nyelvosztály

Korábban már tisztáztuk, hogy szoros kapcsolat van a 3. Chomsky-féle nyelvosztály és a véges automaták (4.3. fejezet), illetve a 2. nyelvosztály és a veremautomaták (7.2. fejezet) között. Megmutatjuk, hogy a Turing-gépek által elfogadott (tehát rekurzívan felsorolható) nyelvek éppen a 0. nyelvosztálynak (lásd 4.2. fejezet), tehát a nyelvtannal generálható nyelveknek felelnek meg.

A konstruktív bizonyítás alapgondolata hasonló a veremautomatáknál látotthoz, csak a megvalósítás annál még egy kicsit bonyolultabb.

12.1. Tétel *Minden* $G = (V, \Sigma, S, P)$ *nyelvtanhoz létezik olyan* M *Turing-gép, hogy* $L(M) = L(G)$.

Bizonyítás. Mivel már láttuk, hogy a nemdeterminisztikus Turing-gépek determinizálhatók (9.16. tétel), elegendő egy nemdeterminisztikus M Turing-gépet mutatnunk, amely megoldja a feladatot: az M gép pontosan azokat a szavakat fogja elfogadni, amiket G generálni tud.

Az M gépnek legyen 4 szalagja. Ezek közül az első a bemeneti szalag, kezdetben azon van a szó, amiről M döntést fog hozni, a többi szalag üres. Az alapgondolat az, hogy a bemenet elolvasása nélkül M a nyelvtan alapján (nemdeterminisztikusan) levezet egy szót, és amennyiben ez megegyezik a bemenettel, akkor M elfogadó állapotban megáll. A

2. szalagon végezzük a levezetést és ehhez a 3, és 4. szalagon tartjuk az aktuális szabály bal, illetve jobb oldalát.

Az első lépések:

- A nyelvtan S kezdőváltozóját M felírja a 2. és a 3. szalagra is, a fej az S karakteren marad.
- Egy olyan $\beta \in V \cup \Sigma^*$ szimbólumsorozatot ír a 4. szalagra, amelyre a nyelvtanban van $S \to \beta$ szabály (nemdeterminisztikus lépés).
- A 2. szalagon S-et (és a következő néhány üres karaktert) felülírja a β szimbólumsorozattal.

Általában pedig

1. A 2. szalagon a fejet a szalag egy nem üres helyére viszi (nemdeterminisztikusan határozza meg a pozíciót).
2. A 3. szalagra ír egy olyan $\alpha \in V \cup \Sigma^*$ sorozatot, ami valamely nyelvtani szabály bal oldala (nemdeterminisztikus választás).
3. Egy olyan $\beta \in V \cup \Sigma^*$ szimbólumsorozatot ír a 4. szalagra, amelyre a nyelvtanban van $\alpha \to \beta$ szabály (nemdeterminisztikus választás).
4. Ellenőrzi, hogy a 2. szalagon a fejtől kezdve éppen az α jelenik-e meg (utána még lehetnek további karakterek). Amennyiben valóban α áll ott, akkor ezt lecseréli a β sorozatra. (Mivel α lehet hosszabb vagy rövidebb is, mint β, a lecserélés együtt járhat azzal, hogy az α utáni részt jobbra vagy balra kell tolni a szalagon néhány hellyel.)

Ezt ismételi, amíg már nem lesz a 2. szalagon változó. Ekkor összehasonlítja a 2. szalag tartalmát az első szalagéval, és ha pontosan azt a szót sikerül előállítani a 2. szalagon, akkor M elfogadó állapotban megáll.

Ha M elfogad, az azt jelenti, hogy a bemeneti szóhoz van a nyelvtanban egy jó levezetés. A másik irányhoz azt kell meggondolni, hogy $L(G)$ minden szavát el fogja fogadni az M gép. Ez részben azért igaz, mert ilyenkor van jó levezetés, részben pedig azért, mert a felsorolt lépések mindegyike véges idő alatt megvalósítható egy Turing-géppel, ezért, ha van jó levezetés, akkor annak M a végére is ér (amikor éppen azokat a nemdeterminisztikus választásokat teszi meg, amik a levezetéshez tartoznak), és az összehasonlítás után el fogja fogadni a szót, tehát lesz egy elfogadó számítási út.

A precíz (teljes indukciós) bizonyítást az olvasóra bízzuk. □

Itt is igaz, hogy nem csak nyelvtanból lehet automatát készíteni, hanem Turing-gépből is definiálható nyelvtan. A nyelvtan levezetési szabályai a Turing-gép helyzetének változásait írják majd le. Ehhez előbb definiáljuk a Turing-gép egy konfigurációjának tömör leírását.

12.2. Definíció *Egy olyan Turing-gépnél, aminek csak egy szalagja van, a konfigurációt az aktuális állapot, a szalag tartalma és a fej helyzete adja meg. Ennek egy lehetséges felírása: xqy, ahol*

- *xy van a szalag elején, utána már csak az üres jel van a szalagon (feltehető, hogy y utolsó karaktere nem az üres szalagjel).*
- *x az olvasófej előtt lévő szalagrész tartalma.*
- *y a szalag további része, a fej y első karakterére mutat.*
- *q a pillanatnyi állapot.*

Az xy karaktersorozat véges hosszú, hiszen kezdetben a véges hosszú bemenet kivételével a szalag az üres jellel van feltöltve, véges sok lépés alatt ez a kezdeti nem üres rész véges hosszú marad. Ezért az xqy mindig egy véges hosszú leírása a konfigurációnak.

12.3. Tétel *Minden L rekurzívan felsorolható nyelvhez van olyan $G = (V, \Sigma, S, P)$ nyelvtan, hogy $L(G) = L$.*

Bizonyítás. Tekintsünk egy, az L nyelvet elfogadó (egyszalagos) $M = (Q, \Sigma, \Gamma, q_0, \_, F, \delta)$ Turing-gépet, amelynek egyetlen elfogadó állapota van ($F = \{q^+\}$), ahonnan nem tud kilépni (lásd 10.2. állítás). A cél, hogy a nyelvtan levezetési szabályai lényegében az M Turing-gép konfigurációjának lehetséges változásait írják le. A levezetésből akkor kapjuk meg a kívánt szót (akkor terminálódik a levezetés), ha a konfigurációba belekerül az egyetlen elfogadó állapot, tehát a Turing-gép számítása elfogadóan véget ér.

Most is át kell hidalnunk azt, hogy míg a Turing-gép kiindulásakor kap egy szót, és azután a szalagján ezt felül is írhatja, így a számítás végére a szó eltűnhet, addig a nyelvtannak a levezetés végén produkálnia kell a kívánt szót. Ezért igazából úgy nézünk a Turing-gép számítására, mintha 2 szalagja (de egy feje) lenne. Kezdetben a két szalag tartalma azonos, később a Turing-gép működése kizárólag a második szalag alapján történik: csak ezt olvassa, csak erre ír. Ha a végén elfogadja a bemenetet, akkor az első szalag tartalma alapján a nyelvtan elő tudja majd állítani az eredeti bemeneti szót.

A nyelvtan változói 3 típusúak:

- S, T_1, T_2 a kezdeti lépésekhez,
- $[t, x]$ alakúak, ahol $t \in \Sigma_0 = \Sigma \cup \{\varepsilon\}$ és $x \in \Gamma$,
- q, ahol $q \in Q$.

A kezdőkonfigurációt előállító szabályok:

- $S \to q_0 T_1$

- $T_1 \to [a, a]T_1$ minden $a \in \Sigma$ esetén
- $T_1 \to T_2$
- $T_2 \to [\varepsilon, \_]T_2 \mid \varepsilon$

Ezekkel az S kezdőváltozóból levezethető sorozatok

$$q_0[a_1, a_1][a_2, a_2] \dots [a_n, a_n][\varepsilon, \_]^m$$

alakúak, $a_i \in \Sigma$, $n, m \geq 0$. Ez a kezdő levezetés annak az állapotnak felel meg, amikor a Turing-gép mindkét szalagján ugyanaz a szó van a szalag elején, a második szalagon csupa _ jel van a szó után, az első szalagon pedig ε-ok, egészen addig, ameddig a gép az első szalagon el fog jutni működése során. (Hogy ez a távolság mekkora, azt azzal lehet beállítani, hogy hányszor használjuk a T_2 változó első szabályát.)

Ez után M átmeneti függvényének megfelelően készítünk levezetési szabályokat. Ezek formája természetesen függ attól, hogy a fej melyik irányban mozdul a szalagon.

Legyen $x, y \in \Gamma$, $q, p \in Q$ és $\delta(q, x) = (p, y, D)$ egy állapotátmenet.

- Ha $D = J$, akkor minden $a \in \Sigma_0$ esetén legyen

  $$q[a, x] \to [a, y]p$$

  Ezen szabály jelentése, hogy a fej egyet mozog jobbra, x helyett y lesz a 2. szalagon, miközben az 1. szalag nem változik.

- Ha $D = B$, akkor minden $a, b \in \Sigma_0$ és $z \in \Gamma$ esetén legyen

  $$[b, z]q[a, x] \to p[b, z][a, y]$$

- Ha $D = H$, akkor minden $a \in \Sigma_0$ esetén legyen

  $$q[a, x] \to p[a, y]$$

Végül vegyük még fel a következő szabályokat a q^+ elfogadó állapotra, amik egy elfogadó számításnak megfelelő levezetés végén biztosítják, hogy a nyelvtan az elfogadott szót generálni tudja a Turing-gép első szalagjának információiból.

- $q^+[a, x] \to q^+aq^+$ minden $a \in \Sigma$ esetén
- $[a, x]q^+ \to q^+aq^+$ minden $a \in \Sigma$ esetén
- $q^+ \to \varepsilon$

Ezzel készen vagyunk a nyelvtannal.

A konstrukció helyességének bizonyítását csak vázoljuk.

Ha $w = a_1 a_2 \dots a_n \in L$, akkor legyen m a Turing-gép számításának tárigénye. Az S kezdőváltozóból először létrehozzuk a

$$q_0[a_1, a_1][a_2, a_2] \dots [a_n, a_n][\varepsilon, \_]^m$$

sorozatot. Ebből a Turing-gép lépéseit követve eljutunk egy

$$[a_1, x_1][a_2, x_2] \dots [a_k, x_k] q^+ [a_{k+1}, x_{k+1}] \dots [a_m, x_m]$$

sorozathoz, ahol $a_j = \varepsilon$, ha $j > n$. Az utolsó csokor szabállyal pedig, előbb m lépésben megkaphatjuk a

$$q^+ a_1 q^+ a_2 q^+ \dots q^+ a_k q^+ a_{k+1} q^+ \dots q^+ a_m q^+$$

sorozatot, és erre a legutolsó szabályt $(m+1)$-szer alkalmazva kapjuk a kívánt

$$a_1 a_2 \dots a_m = a_1 a_2 \dots a_n = w$$

szót, tehát $w \in L(G)$.

A másik irányhoz induljunk ki egy $w \in L(G)$ szóból. A w szó levezetésének elején, amikor a T_2 változó eltűnik, akkor legyen egy

$$q_0[a_1, a_1][a_2, a_2] \dots [a_n, a_n][\varepsilon, \_]^m$$

alakú sorozatunk. Innentől kezdve csak a Turing-gép átmeneti függvényének megfelelő szabályokat lehet alkalmazni egészen addig, amíg meg nem jelenik az elfogadó állapotnak megfelelő q^+ változó. Ez viszont csak akkor történik, ha a Turing-gép elfogadja az $a_1 a_2 \dots a_n$ szót. Már csak azt kell észrevenni, hogy ez éppen a w szó kell legyen, hiszen az $[a, x]$ alakú változók első koordinátái alkotják a szót. Tehát $w \in L(M)$. □

12.4. Következmény *Egy L nyelvre $L \in$ RE akkor és csak akkor, ha van olyan G nyelvtan, melyre $L(G) = L$.*

Bizonyítás. 12.1. és 12.3. tételek adják a két irányt. □

Az a tisztázatlan kérdés maradt csupán, hogy melyik automata-típusnak van kapcsolata az 1. Chomsky-féle osztállyal, a környezetfüggő nyelvtannal (4.14. definíció) generálható nyelvek osztályával.

Ahhoz, hogy ezt a kérdést megválaszoljuk, ismerjük meg alaposabban ezt a nyelvosztályt. Itt a megengedett szabályokban mindig egyetlen változót helyettesítünk valamivel, de hogy a helyettesítés elvégezhető-e, azon múlik, milyen környezetben jelenik meg a változó. Általában itt sem engedélyezett, hogy egy változót az üres szóval helyettesítsünk

(kivéve, amikor a szabályok jobb oldalán nem szereplő kezdőváltozót ε-ra írjuk át egy lépésben).

Az ilyen környezetfüggő (vagy CS) nyelvtan fontos tulajdonsága, hogy egy levezetési lépés során az adott szimbólumsorozat hossza nem tud csökkenni, ez alól az egyetlen kivétel az $S \Rightarrow \varepsilon$ levezetés.

Ebből az észrevételből egy másik jellemzését is megkaphatjuk a CS nyelvtanoknak

12.5. Definíció *Egy $G = (V, \Sigma, S, P)$ nyelvtant* nem csökkentőnek *hívunk, ha a szabályai $\alpha \to \beta$ alakúak, ahol az α-ban van változó és $|\beta| \geq |\alpha| > 0$*

12.6. Állítás *Legyen $\varepsilon \notin L$. Az L nyelv akkor és csak akkor környezetfüggő, ha van az L-et generáló nem csökkentő nyelvtan.*

Bizonyítás. Az egyik irány világos, hiszen egy CS nyelvtan (amennyiben nincs $S \to \varepsilon$ szabálya) egyben nem csökkentő is.

Tegyük fel most, hogy $G = (V, \Sigma, S, P)$ egy nem csökkentő nyelvtan. Megmutatjuk, hogyan lehet ebből egy $G' = (V', \Sigma, S', P')$ CS nyelvtant kapni. Az lehet a baj, hogy a nem csökkentő nyelvtan egy szabálya nem egyetlen változót helyettesít valamivel, hanem szimbólumok egy sorozatát. Egy ilyen helyettesítést felbontunk majd több részre, hogy egyszerre mindig csak egy változó helyébe kerüljön valami.

- Minden $a \in \Sigma$ karakterhez legyen X_a egy új változó és vegyük fel az $X_a \to a$ szabályt.
- G minden szabályában az a karakter összes előfordulását cseréljük le az X_a változóra. (Ezután minden kettőnél hosszabb jobboldalú szabály mindkét oldalán csak változók szerepelnek.)
- Egy $A_1A_2 \ldots A_{k-1}A_k \to B_1B_2 \ldots B_m$ szabályhoz (ezen a ponton minden eredeti szabályból ilyet kaptunk) vegyünk fel k új változót, legyenek ezek $Y_1, Y_2, \ldots, Y_k$. Az eredeti szabályt helyettesítsük a következőkkel:

$$\begin{aligned}
A_1A_2 \ldots A_{k-1}A_k &\to Y_1A_2 \ldots A_{k-1}A_k \\
Y_1A_2 \ldots A_{k-1}A_k &\to Y_1Y_2 \ldots A_{k-1}A_k \\
&\ldots \\
Y_1Y_2 \ldots Y_{k-1}A_k &\to Y_1Y_2 \ldots Y_{k-1}Y_k \\
Y_1Y_2 \ldots Y_{k-1}Y_k &\to B_1Y_2 \ldots Y_{k-1}X_k \\
B_1Y_2 \ldots Y_{k-1}Y_k &\to B_1B_2 \ldots Y_{k-1}X_k \\
&\ldots \\
B_1B_2 \ldots B_{k-1}Y_k &\to B_1B_2 \ldots B_{k-1}B_k \ldots B_m
\end{aligned}$$

Mivel $m \geq k$, így valóban egy CS nyelvtant kapunk. A lépések száma szerinti teljes indukcióval belátható, hogy $L(G') = L(G)$. □

12.7. Példa *Legyen* $L = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n : n \geq 1\}$. *Tudjuk, hogy ez a nyelv nem környezetfüggetlen (6.33. állítás). Az alábbi viszont egy nem csökkentő nyelvtan hozzá, tehát, az előzőek szerint, környezetfüggő.*

$$\begin{aligned} S &\to \mathtt{a}BS\mathtt{c} \mid \mathtt{abc} \\ B\mathtt{a} &\to \mathtt{a}B \\ B\mathtt{b} &\to \mathtt{bb} \end{aligned}$$

Most már készen állunk arra, hogy megmutassuk melyik automatával ekvivalens a CS nyelvek osztálya.

12.8. Definíció *A* lineárisan korlátozott automata *a nemdeterminisztikus egy szalagos Turing-gépnek egy olyan változata, aminek a szalagján kezdetben a bemenet előtt és után van egy-egy speciális* szalag-eleje, *illetve* szalag-vége *jel. Ezek a jelek a későbbiekben nem írhatók felül, és nem lehet a szalag elejéről balra, illetve a szalag végéről jobbra lépni.*

Egy lineárisan korlátozott automata tehát egy olyan Turing-gép, amelynek munkaterülete olyan hosszú, mint a bemenete.

12.9. Tétel *Az L nyelv akkor és csak akkor környezetfüggő, ha van olyan lineárisan korlátozott M automata, melyre $L(M) = L$*

Bizonyítás. A nyelvtanból automata irányhoz azt kell meggondolni, hogy a 12.1. bizonyítás kis módosításokkal erre az esetre is átvihető. Miért kell módosítanunk? Először is, ott több szalagot használtunk, amit most nem lehet. De a 9.11. tétel (k-szalagosból egyszalagos Turing-gép) bizonyításában látott módszer szerint megoldhatjuk ezt egy szalagon és több sávon. Másodszor, el kell érni, hogy ha egy levezetési lépés során a szalag-vége jelre írni szeretnénk, akkor a számítás álljon le, utasítson el. Ez könnyen megvalósítható. Mivel a nyelvtan nem csökkentő, pontosan akkor van az adott szalag-darabon megkapható levezetése a szónak, ha a szó $L(G)$-nek eleme, ezért a lineárisan korlátozott automatára $L(M) = L(G)$ fennáll.

A másik irány a 12.3. tételhez hasonlóan bizonyítható. Egy nehézség van most: a $q^+ \to \varepsilon$ szabály nem alkalmazható, ezért az állapotokat a rákövetkező karakterrel együtt belefoglaljuk egy összetett változóba. Ezen túl még a két speciális szalagjelet (szalag-eleje és szalag-vége) is kezelni kell, amelyek a szalagon definíció szerint szerepelnek, de a nyelvtanból generált szóban nem jelenhetnek meg. Ezeket együtt kezeljük az első, illetve utolsó karakterrel, egyetlen változóba beolvasztva a két karaktert.

A részletekért lásd [4, 8]-at. □

12.10. Következmény *Ha L egy CS nyelv, akkor $L \in \mathrm{SPACE}(n^2)$.*

Bizonyítás. Ha L egy CS nyelv, akkor van hozzá lineárisan korlátozott automata, ezért $L \in \mathrm{NSPACE}(n)$. Az állítás Savitch tételéből (11.18. tétel) azonnal következik. □

A tár-idő tétel (11.20. tétel) segítségével kapjuk innen, hogy:

12.11. Következmény *Ha L egy CS nyelv, akkor $L \in \mathrm{R}$.*

Korábban láttunk már példát CF, de nem reguláris, illetve CS, de nem CF nyelvekre. A fentiek után már könnyű olyan nyelvre is példát adni, ami nem CS, de nyelvtannal generálható, és olyanra is, amihez egyáltalán nincs nyelvtan.

12.12. Következmény

- *Az L_u univerzális és az L_h megállási nyelv nem környezetfüggő, de generálható nyelvtannal.*
- *Az L_d diagonális nyelvhez nincs a nyelvet generáló nyelvtan*

Bizonyítás. Az állítás következik abból, hogy L_u és L_h nem rekurzív de rekurzívan felsorolható (10.14. és 10.16. tételek), míg L_d nem is rekurzívan felsorolható (10.12. tétel). □

Megmutatható, hogy olyan nyelv is van, ami rekurzív ugyan, de nem CS, de az „értelmes" rekurzív nyelvek nagy része egyben környezetfüggő is.

Nézzük meg most azt, hogy mit állíthatunk a 0. és az 1. nyelvosztályról a konkatenációra és a tranzitív lezártra való zártsággal kapcsolatban.

12.13. Állítás *Ha az L_1 és L_2 nyelv mindegyike 0. osztályú (1. osztályú), akkor*

- $L_1 \cup L_2$
- $L_1 L_2$
- L_1^*

is olyan.

Bizonyítás. A CF nyelvek hasonló 6.31. tételének bizonyítása itt is működik, hiszen csak környezetfüggetlen típusú szabályokkal bővítettük a nyelvtanokat. □

Ha L egy CS nyelv, akkor a $\overline{L}$ komplementere is CS: a CS nyelvet elfogadó mindig megálló lineárisan korlátozott automatáról is feltehető, hogy két állapotban áll csak meg, amik közül pontosan az egyik elfogadó. A két megállós állapot felcserélésével éppen a komplementer nyelvet fogadhatjuk el. Ebből az is következik, hogy a CS nyelvek zártak a metszetre is (mert unióra és komplementerre zártak).

Ha azonban L a 0. osztályba tartozik, akkor már nem következik, hogy a komplementere is ilyen (mert RE nem zárt a komplementerképzésre), de a 0. osztály zárt a metszetre (mert már láttuk korábban, hogy lehet a metszethez Turing-gépet konstruálni).

A Chomsky-féle nyelvosztályok zártsági tulajdonságai összefoglalva:

	unió	metszet	komplementer	összefűzés	tranzitív lezárt
reguláris (3. oszt)	zárt	zárt	zárt	zárt	zárt
CF (2. oszt)	zárt	nem zárt	nem zárt	zárt	zárt
determ. CF	nem zárt	nem zárt	zárt	zárt	zárt
CS (1. oszt)	zárt	zárt	zárt	zárt	zárt
általános (0. oszt)	zárt	zárt	nem zárt	zárt	zárt

12.2. Turing-gépek és kapcsolatuk a CF nyelvekkel

A Turing-gépeket nem csak úgy lehet a nyelvtanokkal kapcsolatba hozni, hogy a gép által elfogadott nyelvhez nyelvtant készítünk. Látni fogjuk, hogy egy Turing-gép elfogadó számításai is leírhatók nyelvtannal.

Tegyük fel, hogy az $M = (Q, \Sigma, \Gamma, q_0, \_, F, \delta)$ Turing-gép nemdeterminisztikus, egyszalagos, egyetlen elfogadó állapota van ($F = \{q^+\}$), amiből nem tud kilépni. Tudjuk, hogy egy ilyen Turing-gép konfigurációja leírható xqy alakban (12.2. definíció).

12.14. Definíció *Az M Turing-gép elfogadó számításainak nyelve legyen*

$$\text{elfogad}(M) = \{w_1 \# w_2^{-1} \# w_3 \# w_4^{-1} \dots w_{2k+1}\} \cup \{w_1 \# w_2^{-1} \# w_3 \# w_4^{-1} \dots w_{2k}^{-1}\},$$

ahol

- *w_i az M gép egy konfigurációja,*
- *$\# \notin \Gamma \cup Q$, a konfigurációkan nem szereplő elválasztó karakter,*
- *$w_1 = q_0 y$ a kezdőkonfiguráció,*
- *a w_i-ből egy lépésben előáll w_{i+1},*
- *w^{-1} a w konfiguráció visszafelé írva,*
- *az utolsó konfigurációban az elfogadó q^+ állapot szerepel,*

Tehát lényegében egy elfogadó számítás egymás utáni konfigurációinak sorozatából áll, ahol minden másodikat technikai okokból megfordítva írunk le.

12.15. Tétel *Van olyan algoritmus, amely egy adott M Turing-géphez meghatároz olyan G_1, G_2 és G_3 környezetfüggetlen nyelvtanokat, hogy a következők teljesüljenek.*

- $\text{elfogad}(M) = L(G_1) \cap L(G_2)$
- $\overline{\text{elfogad}(M)} = L(G_3)$

Bizonyítás. Nyelvtanok helyett elegendő veremautomatákat megadni (7.16. tétel) a nyelvekhez. Vázoljuk a három veremautomatát, M_i lesz az, amelyikből a G_i nyelvtan készülhet.

Az első állítás bizonyításához két olyan veremautomatát készítünk, melyek közül az egyik a leírt konfigurációsorozatról azt ellenőrzi, hogy a páratlan sorszámú lépések helyesek-e, a másik pedig azt, hogy a páros sorszámúak helyesek-e. A két gép nyelvének metszete ennek megfelelően a jó konfigurációsorozatokból fog állni.

Tegyük fel, hogy az M Turing-gép az xqy konfigurációból át tud lépni a zrs konfigurációba. Ha például ez a lépés egy $(q, y_1) \to (r, a, J)$ átmenetet valósít meg, akkor $z = xa$, és s az y szóból az első karakter elhagyásával kapható. (A fej másik két mozgásánál is hasonlóan, csak az állapot közvetlen környékén változik a konfiguráció.) Ezért ha egy veremautomata bemenete az $xqy\#s^{-1}r^{-1}z^{-1}$ szó, akkor ellenőrizni tudja, hogy ez valóban az M Turing-gépnek egy legális lépése:

- A beolvasott karaktereket beírja a verembe addig, amíg meg nem jelenik a Turing-gép egy állapota a bemeneten.
- Az állapotot is berakja a verembe.
- A verem tetején levő szimbólum (q) és a következő bemeneti karakter y_1 függvényében változtatja a verem tartalmát (a fenti példában q helyébe a verembe berakja az y_1 majd az r karaktereket).
- A további beolvasott karaktereket az elválasztó # jelig berakja a verembe.
- A # jel után összeveti a bemenetet a verem tartalmával, azaz minden beolvasott karakterrel kivesz egy karaktert a veremből, és ha ezek nem egyeznek meg, akkor megáll.
- Ha a bemenet végére érve a veremben csak a verem alját jelző szimbólum marad, akkor a veremautomata elfogadó állapotba lép.

Vegyük észre, hogy mivel a veremből először a legutoljára berakott karaktert tudjuk kivenni, ezért indokolt, hogy a zrs konfiguráció visszafelé szerepeljen a bemeneten – így lehet a veremtartalommal érdemben összehasonlítani.

M_1 működése:

- Sorban ellenőrzi, hogy w_{2i-1} alakja xqy és hogy a $w_{2i-1} \to w_{2i}$ egy lehetséges lépés $(i = 1, 2, \dots)$.
- Ellenőrzi, hogy az utolsó konfiguráció elfogadó.

M_2 működése:

- Sorban ellenőrzi, hogy w_{2i} alakja xqy és hogy a $w_{2i} \to w_{2i+1}$ egy lehetséges lépés $(i = 1, 2, \dots)$.
- Ellenőrzi, hogy w_1 valóban a kezdőkonfiguráció.

Ekkor $L(M_1) \cap L(M_2)$ épp azokat a sorozatokat tartalmazza, amelyek páros és páratlan lépései is az M egy legális lépésének felelnek meg, a kezdőkonfigurációból indulnak és egy elfogadó konfigurációval érnek véget, azaz $L(M_1) \cap L(M_2) = L(M)$.

Az elfogad(M) nyelv komplementere többféle szóból áll. Azokból, amelyek nem konfigurációk sorozatai, és azokból, amelyek ugyan konfiguráció-sorozatok, de nem írnak le számítást vagy nem elfogadó számítást írnak le.

Ennek megfelelően M_3 nemdeterminisztikusan választ, melyik lehetőséget ellenőrzi (melyik módon akarja lebuktatni a rossz konfiguráció-sorozatot):

- a bemenet megfelelő alakú-e
- w_1 a kezdőkonfiguráció-e
- egy $w_i \to w_{i+1}$ átmenet helyes-e
- az utolsó w konfiguráció elfogadó-e

Ha bármelyik esetben kiderül, hogy hiba van, akkor M_3 elfogadja a szót, különben elutasít.

A 3. eset kivételével az ellenőrzés véges automatával is elvégezhető. A 3. esetben csak egyetlen i indexre történik az ellenőrzés (a korábban vázolt módon). Az, hogy melyik i-re legyen, ismét nemdeterminisztikus választás eredménye. Ha egy konfiguráció-sorozat hibás, akkor van olyan nemdeterminisztikus választása az M_3 gépnek, aminél ez kiderül, ilyenkor M_3 elfogad, vagyis M_3 pontosan elfogad(M)-t ismeri fel. □

12.3. CF nyelvtanok algoritmikus kérdései – eldönthetetlenségi eredmények

Kezdjük egy korábban már említett (6.2 fejezet) eredménnyel.

12.16. Tétel *Az, hogy egy adott CF nyelvtan egyértelmű-e, algoritmikusan nem dönthető el, azaz az egyértelmű CF nyelvtanokból álló nyelv nem rekurzív.*

Bizonyítás. A bizonyításhoz a Post megfeleltetési problémát (10.6. fejezet) használjuk fel. Tetszőleges, szópárokból álló $\{(s_i, t_i) : i = 1, 2 \dots k\}$ halmazhoz mutatunk olyan G környezetfüggetlen nyelvtant, hogy

$$\{(s_i, t_i) : i = 1, 2 \dots k\} \notin \text{PCP} \Leftrightarrow G \text{ egyértelmű}$$

A szópárok Σ ábécéjét kiegészítjük k darab új karakterrel, legyenek ezek $c_1, c_2, \ldots, c_k$. Legyenek G szabályai az alábbiak:

- $S \to A \mid B$
- $A \to s_i A c_i \mid s_i c_i$, minden $i = 1, 2 \ldots k$-ra
- $B \to t_i B c_i \mid t_i c_i$, minden $i = 1, 2 \ldots k$-ra

Tehát a szavak indexének jelölésére a $c_1, c_2, \ldots, c_k$ karaktereket használjuk.

Ekkor ha a PCP problémának megoldása az $i_1, i_2, \ldots i_n$ indexsorozat, akkor fennáll, hogy

$$s_{i_1} s_{i_2} \ldots s_{i_n} = t_{i_1} t_{i_2} \ldots t_{i_n}$$

Ez viszont azt jelenti, hogy az alábbi két bal levezetés

$$\begin{aligned} S &\Rightarrow A \Rightarrow s_{i_1} A c_{i_1} \Rightarrow \cdots \Rightarrow s_{i_1} s_{i_2} \ldots s_{i_n} c_{i_n} c_{i_{n-1}} \ldots c_{i_1} \\ S &\Rightarrow B \Rightarrow t_{i_1} B c_{i_1} \Rightarrow \cdots \Rightarrow t_{i_1} t_{i_2} \ldots t_{i_n} c_{i_n} c_{i_{n-1}} \ldots c_{i_1} \end{aligned}$$

ugyanazt a szót eredményezi, tehát a nyelvtan nem egyértelmű.

Másrészt viszont az A illetve a B változóból minden szó egyértelműen vezethető le (ezt biztosítják az indexet jelző c_i karakterek). Tehát, ha a nyelvtan nem egyértelmű, akkor van olyan szó, amihez tartozik $S \Rightarrow A$ és $S \Rightarrow B$ kezdetű levezetés is. Ahhoz viszont, hogy a végeredmény ugyanaz legyen, olyan $i_1, i_2, \ldots, i_n$ indexsorozat kell, ami megoldása a Post megfeleltetési feladatnak. Tehát a megadott nyelvtan nem egyértelműsége ekvivalens a Post megfeleltetési probléma megoldhatóságával, amiről tudjuk, hogy nem rekurzív. □

Láttuk, hogy véges automatáknál az alapvető algoritmikus kérdésekre (beletartozás, üresség, végesség, egyenlőség, diszjunktság) vannak véges, bár nem feltétlenül gyors eljárások. A CF nyelvek esetében eddig az első háromra láttunk eljárást a 8. fejezetben. Most azt fogjuk megmutatni, hogy a másik kettőre nem létezik algoritmus. Ebből az is következik, hogy ha általánosabb nyelvtant (CS vagy akár tetszőleges) engedünk meg, ezek a problémák akkor is algoritmikusan eldönthetetlenek lesznek.

12.17. Tétel *Nincs olyan algoritmus, amely minden G_1 és G_2 környezetfüggetlen nyelvtanpár esetén eldönti, hogy*

- $L(G_1) \cap L(G_2) \stackrel{?}{=} \emptyset$
- $L(G_1) \stackrel{?}{=} \Sigma^*$
- $L(G_1) \stackrel{?}{=} L(G_2)$

Bizonyítás. Indirekt módon okoskodunk. Először tegyük fel, hogy van olyan $\mathcal{A}$ algoritmus, amely két tetszőleges CF nyelvtanról eldönti, hogy generált nyelveik diszjunktak-e. Tekintsünk egy tetszőleges M Turing-gépet. Ekkor a 12.15. tétel alapján M leírásából elő tudunk állítani olyan G_1 és G_2 CF nyelvtanokat, melyekre $L(G_1) \cap L(G_2) = \text{elfogad}(M)$. Ezen G_1 és G_2 nyelvtanokon futtatva az $\mathcal{A}$ algoritmust véges sok lépés után megtudjuk, hogy nyelveik metszete üres-e, vagyis hogy az M Turing-gép elfogadó számításainak elfogad(M) nyelve üres-e.

Ez tehát azt jelentené, hogy az a nyelvi tulajdonság, hogy a nyelv üres, algoritmikusan eldönthető lenne, ami ellentmond Rice tételének (10.35. tétel), hiszen az üresség egy nem triviális nyelvi tulajdonság.

A második állításhoz tegyük fel, hogy van egy $\mathcal{B}$ algoritmus, amely tetszőleges G CF nyelvtanról eldönti, hogy a nyelve az összes szót tartalmazza-e. Megint használhatjuk a 12.15. tételt, de most azt a részét, hogy M-ből kaphatunk egy G_3 CF nyelvtant, melyre $L(G_3) = \overline{\text{elfogad}(M)}$. Ha ezt a G_3 nyelvtant adjuk a $\mathcal{B}$ algoritmusnak, akkor véges sok lépés után megtudjuk, $\overline{\text{elfogad}(M)}$ tartalmaz-e minden szót, ami azzal ekvivalens, hogy elfogad(M) $= \emptyset$ igaz-e, azaz $L(M)$ üres-e. Mint már az előbb is, ezzel ellentmondásra jutottunk.

A harmadik állítást egyszerűen kapjuk az előzőből. Legyen ugyanis G_2 egy olyan CF nyelvtan, amire $L(G_2) = \Sigma^*$ teljesül. Ilyen nyelvtan van, mivel akár reguláris nyelvtan is adható. Ekkor, ha el tudnánk dönteni két tetszőleges CF nyelvtan esetén, hogy a generált nyelvük megegyezik-e, akkor az eljárást erre a G_2-re (és tetszőleges G_1-re) használva az $L(G_1) \stackrel{?}{=} \Sigma^*$ kérdést is eldöntenénk, ami az előző pont szerint lehetetlen. □

12.18. Megjegyzés *A* beletartozás *kérdése 0. osztályú nyelvtanok esetén algoritmikusan eldönthetetlen (mert az L_u nyelv nem rekurzív). Az 1. osztályú nyelvtanokra viszont eldönthető lineáris tárban, és így véges időben (mert $CS \subseteq R$).*

Az üresség és végesség kérdése 0. osztályú nyelvtanokra szintén algoritmikusan nem eldönthető (a Rice-tétel következménye), de ezek már az 1. osztályú nyelvtanokra sem eldönthetőek (ezt itt nem részletezzük).

A Chomsky-féle nyelvosztályok algoritmikus kérdéseinek eldönthetősége összefoglalva

	beletartozás	üresség	egyenlőség	diszjunktság	végesség
reguláris (3. oszt)	igen	igen	igen	igen	igen
CF (2. oszt)	igen	igen	nem	nem	igen
CS (1. oszt)	igen	nem	nem	nem	nem
általános (0. oszt)	nem	nem	nem	nem	nem

Irodalomjegyzék

[1] Bach Iván: *Formális nyelvek*, Typotex, 2002.

[2] Friedl Katalin: *Kiegészítő anyag az Algoritmuselmélet tárgyhoz III, A bonyolultságelmélet alapjai: a P és az NP osztály*, www.cs.bme.hu/~friedl/alg/bonyelm.pdf, 2010.

[3] Michael R. Garey, David S. Johnson: *Computers and Intractability: A Guide to the Theory of NP Completeness* W.H. Freeman, 1979.

[4] John E. Hopcroft, Jeffrey D. Ullman: *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

[5] Lovász László: *Algoritmusok bonyolultsága*, ELTE egyetemi jegyzet, 2013.

[6] Christos Papadimitriou: *Számítási bonyolultság* Novadat Bt, 1999.

[7] Rónyai Lajos, Ivanyos Gábor, Szabó Réka: *Algoritmusok*, Typotex, 1998.

[8] Jeffrey Shallit: *A Second Course in Formal Languages and Automata Theory*, Cambridge, 2009.

[9] Michael Sipser: *Introduction to the Theory of Computation*, Thomson, 1996.