Számításelmélet

órai jegyzet

Előadó: Dr. Gazdag Zsolt

Utolsó módosítás: 2008. január 23.

Előszó

Ez a folyamatosan bővülő órai jegyzet az ELTE Informatikai Karán, a 2007-2008-as őszi szemeszterben tartott „Számításelmélet” című kurzus anyagát tartalmazza. Ezáltal segítséget nyújthat a kurzus jobb megértéséhez és a vizsgára való felkészüléshez.

Az előadás célja, hogy betekintést nyújtson az elméleti számítástudomány alábbi területeire:

- **Kiszámíthatóság elmélet**: Mely problémák oldhatók meg algoritmikusan?
- **Bonyolultságelmélet**: Az eldönthető problémák közül melyek a nehéz problémák, vagyis azok a problémák melyek az erőforrások (tár és idő) felhasználása szempontjából nem hatékonyak?
- **Automaták és formális nyelvek elmélete**: Alapul szolgál a fenti két területhez, de az informatika más területein is hasznos (fordítóprogramok, természetes nyelvek feldolgozása, logika, stb.).

Miért hasznos ezeket a dolgokat ismerni? Többek között azért mert

- nem kezdünk el megoldani egy problémát ha tudjuk, hogy megoldhatatlan.
- ha megértjük, hogy egy probléma miért nehezen megoldható, akkor módosítva a nehézséget okozó részt esetleg egy könnyebben megoldható (de a célnak még megfelelő) problémát kapunk.
- Ha tudjuk, hogy a pontos megoldás megtalálása nehéz feladat, akkor esetleg megelégszünk közelítő megoldások keresésével.

Tartalomjegyzék

1. fejezet

Bevezetés

Ebben a fejezetben megismerkedünk a tantárgy alapfogalmaival, valamint a kiszámíthatóság elmélet rövid történetével. Majd ejtünk pár szót arról, hogy mivel foglalkozik az előadás másik fő témaköre, a bonyolultságelmélet. A fejezet végén összefoglaljuk, hogy milyen témákat ölel fel az előadás anyaga.

1.1. Alapfogalmak

Számítási problémának nevezünk egy olyan, a matematika nyelvén megfogalmazott kérdést, amire számítógéppel szeretnénk megadni a választ. A gyakorlati élet szinte minden problémájához rendelhető, megfelelő absztrakciót használva, egy számítási probléma.

Példa Tekintsük a következő, valós életből vett problémát. Tegyük fel, hogy van több, azonos magasságú, de különböző méretű hordónk, melyeket el szeretnénk szállítani valahova. Adódik a kérdés: hogyan helyezzük el a teherautónkon a hordóinkat úgy, hogy minél nagyobb legyen a hordók együttes űrtartalma. Az ehhez a feladathoz rendelhető számítási probléma: hogyan helyezhetünk el egy téglalapban különböző sugarú köröket úgy, hogy a téglalapnak minél nagyobb részét lefedjük? □

Egy problémát a hozzá tartozó konkrét bementettel együtt a probléma egy példányának nevezzük. A fenti számítási probléma egy példánya az amikor megadjuk a téglalap és a körök konkrét méreteit.

Speciális számítási probléma az eldöntési probléma. Ilyenkor a problémával kapcsolatos kérdés egy eldöntendő kérdés, tehát a probléma egy példányára a válasz „igen” vagy „nem” lesz.

Ilyen eldöntési probléma az úgynevezett SAT probléma, amit a következőképpen definiálunk. Adott egy ϕ zérusrendű (ítéletkalkulusbeli) konjunktív normálfor-

ma. A kérdés az, hogy kielégíthető-e ϕ. Tehát a problémára a válasz „igen” ha ϕ kielégíthető és „nem” egyébként.

Egy számítási probléma reprezentálható egy $f : A \to B$ parciális függvénnyel. Az A halmaz tartalmazza a probléma egyes példányait, jellemzően egy megfelelő ábécé feletti szóban elkódolva, míg a B halmaz tartalmazza az egyes példányokra a függvény által adott értékeket, szintén valamely alkalmas ábécé feletti szóban elkódolva. Értelemszerűen, ha eldöntési problémáról van szó, akkor az f értékkészlete, vagyis a B egy két elemű halmaz: $\{\texttt{igen, nem}\}$, $\{1, 0\}$, stb. Az f azért parciális függvény, mert az f által reprezentált probléma lehet olyan, hogy a probléma egyes példányaira nem lehet algoritmikusan kiszámítani a választ.

Egy $f : A \to B$ függvényt kiszámíthatónak nevezünk, ha létezik olyan algoritmus amely minden $x \in A$ elemre véges sok lépésben kiszámítja az $f(x) \in B$ értéket (tehát f teljesen definiált, azaz totális függvény). Egy probléma pedig megoldható, ha az általa meghatározott függvény kiszámítható. Ha egy eldöntési probléma megoldható, akkor azt is mondjuk, hogy a probléma eldönthető.

A SAT probléma eldönthető, hisz könnyen adható egy algoritmus, ami eldönti azt, hogy egy ϕ formula kielégíthető-e. Ez az algoritmus nem csinál mást, mint a ϕ-ben szereplő változóknak logikai értéket ad az összes lehetséges módon, majd rendre kiértékeli a formulát.

Egy problémát még akkor is eldönthetőnek nevezünk, ha a probléma bizonyos példányaira az eldöntő algoritmus évszázadokig fut. A lényeg csak az, hogy az algoritmus véges sok lépés után megálljon. Tekintsük például újra az eldönthető SAT problémát. A probléma eldöntésére a fent leírt algoritmus a formula változószámának függvényében exponenciális lépésszámú. Ez pedig a gyakorlatban, legalábbis a sok változót tartalmazó formulákra, használhatatlan. Hogy ezt belássuk tekintsünk egy 100 változót tartalmazó ϕ konjunktív normálformát. Tudjuk, hogy ϕ kielégíthetőségének az eldöntéséhez általában 2^{100} kiértékelés szükséges. Ez akkora szám, hogy egy másodpercenként 10^{12} műveletre képes számítógép körül-belül $4 \cdot 10^{16}$ évig dolgozna a problémán, ami pedig több, mint az univerzum jelenlegi életkora.

Mindazonáltal a SAT és valójában a nála még sokkal bonyolultabb problémák zöme is eldönthető. Felmerül tehát a kérdés, hogy van-e egyáltalán olyan probléma ami nem dönthető el. Később látni fogjuk, hogy ilyen probléma létezik.

Egy eldönthető probléma tekinthető úgy is mint egy formális nyelv. A probléma példányait elkódoljuk egy megfelelő ábécé feletti szavakban. Ezek után magát a problémát azonosítjuk azzal a formális nyelvvel, mely azokat a szavakat tartalmazza, melyek a probléma „igen” példányait kódolják, vagyis azokat a példányokat melyekre a problémát eldöntő algoritmus „igen” választ ad.

Az így kapott formális nyelvet általában ugyanúgy nevezzük, mint magát a problémát. Tehát például a SAT jelentheti a fent definiált eldöntési problémát és azt a formális nyelvet is, amely szavai a kielégíthető zérusrendű formulákat kódolják.

Ahhoz, hogy egy eldöntési problémát algoritmikusan megoldjunk, elegendő az

algoritmus fogalmának egy intuitív definíciója is:

> Utasítások jóldefiniált, véges sorozata, melyeket végrehajtva megoldható egy adott feladat (probléma).

Ez az intuitív definíció elég lehet ahhoz, hogy megoldjunk egy konkrét problémát, de nem elég ahhoz, hogy megmutassuk egy problémáról azt, hogy algoritmikusan eldönthetetlen (sőt, pontos definíció nélkül azt sem lehet megmutatni, hogy egyáltalán létezik-e algoritmikusan eldönthetetlen probléma).

A következőkben röviden áttekintjük, hogy melyek voltak azok a főbb események, amelyek a kiszámíthatóság elmélet kialakulásához és ezzel együtt az algoritmus matematikailag is precíz definíciójához vezettek.

1.2. A kiszámíthatóság elmélet rövid története

1900-ban, a századforduló alkalmából, David Hilbert német matematikus 23 addig megválaszolatlan kérdést intézett a kor matematikusaihoz. Ezek közül néhány, mint ahogy az később kiderült, nagy hatással volt a huszadik századi matematika, és különösen a kiszámíthatóságelmélet, fejlődésére.

Ezen problémák közül a 10-ik a következőképpen szólt. Adott egy p egész együtthatós polinom. A kérdés az, hogy tudunk-e p változóiba olyan egész számokat helyettesíteni, hogy p értéke 0 legyen? Legyen például $p = 2x^2 - 3xy + 2z$. Akkor ha x, y és z helyébe rendre 2-t, 1-et és 1-et helyettesítünk, akkor p értéke 0 lesz. Tehát ennek a konkrét polinomnak az esetében „igen" a válasz a kérdésre. Hilbert olyan algoritmust keresett, ami tetszőleges polinom esetén „igen" vagy „nem" választ ad. Úgy gondolta, hogy nincs eldönthetetlen probléma és meg volt győződve róla, hogy a 10-ik probléma is eldönthető megfelelő algoritmussal. Ezt a problémát végül Matijasevič oldotta meg 1970-ben. Megmutatta, hogy Hilbert 10-ik problémája algoritmikusan eldönthetetlen.

Hilbert az 1920-as években meghirdette nagyra törő programját, melynek lényege az volt, hogy formalizálni kellene a matematika összes elméletét egy véges axiómarendszerrel, és megmutatni, hogy ez az axiómarendszer konzisztens (nem vezethető le belőle ellentmondás, azaz egy állítás és annak a tagadása is).

Hilbert programjának része volt az úgynevezett Entscheidungsproblem (magyarra Eldönthetőségi Problémaként fordítható) mely egy olyan algoritmus megadását tűzte ki célul ami a matematika tetszőleges állításáról eldönti, hogy az igaz vagy hamis.

1931-ben Kurt Gödel bebizonyította az ún. első nemteljességi tételét: Minden olyan mechanikusan kiszámítható elméletben ami tartalmazza az elemi aritmetikát van olyan állítás, hogy az adott elméletben sem az állítás, sem annak tagadása nem bizonyítható. Tehát minden ilyen elméletben van olyan állítás ami igaz de nem bizonyítható. Ebből a tételből már következik, hogy Hilbert programjának alapvető célkitűzései megvalósíthatatlanok. Azt viszont a tétel még

elvileg nem zárta ki, hogy létezik algoritmus, ami eldönti a matematika összes állítását, mivel az algoritmus pontos definíciója még nem létezett akkor. Az Eldönthetőségi Problémára a negatív választ Alonzo Church és Alan Turing adta meg egymástól függetlenül, de nagyjából egy időben, 1936-ban. Ehhez viszont az kellett, hogy bevezetésre kerüljenek olyan algoritmus modellek, amelyekről később kiderül, hogy egymással megegyező számítási erővel rendelkeznek:

Gödel: 1931-ben bevezeti a primitív rekurzív függvényeket. 1934-ben, egy előadáson, Herbrand javaslatára definiálja az általánosabb rekurzív függvényeket és megfogalmazza azt a nézetét, hogy ezek a függvények megfelelnek a „mechanikusan" kiszámítható függvényeknek.

Church: Az 1930-as évek elején tanítványaival (Kleene és Rosser) megalkotja a λ-kalkulust, egy formális rendszert, ami a függvény fogalmán és a függvényeknek a változók értékeire való alkalmazásán alapszik. Ezen belül megalkotják a λ-definiálható függvényeket. Később bebizonyítják, hogy ezek ekvivalensek rekurzív függvényekkel.

Turing: 1936-os cikkében definiálja a később róla elnevezett Turing-gépet és megfogalmazza azt a nézetét, hogy a Turing-géppel kiszámítható függvények megegyeznek az algoritmikusan kiszámítható függvényekkel. A cikke végén vázolja annak bizonyítását, hogy a λ-definiálható valamit a Turing-géppel kiszámítható függvények megegyeznek.

Church az Entscheidungsproblem-re úgy adott negatív választ, hogy megmutatta, nincs olyan kiszámítható függvény, ami két λ-kalkulusbeli kifejezésről eldönti, hogy ekvivalensek-e. Turing a következőképpen gondolkodott. Először megmutatta, hogy a Turing-gépek megállási problémája eldönthetetlen. Utána pedig megfogalmazta a problémát matematikai állításként. Ebből már következett, hogy nem létezhet olyan algoritmus ami eldönteni a matematikai állítások igazságértékét.

Később további modelleket is definiáltak (pl. RAM gépek, Post-gépek, Markov-algoritmusok), de mindről kiderült, hogy nem rendelkeznek a Turing-gépnél nagyobb számítási erővel. Ezek az eredmények is alátámasztják az ún. Church-Turing tézist:

> A kiszámíthatóság különböző matematikai modelljei mind az effektíven kiszámítható függvények osztályát definiálják.

1.3. A bonyolultságelméletről

Amíg a kiszámíthatóság elmélet alapvetően azzal foglalkozik, hogy egy probléma megoldható-e (eldönthető-e), addig a bonyolultságelmélet azt vizsgálja, hogy az

eldönthető problémák közül melyek milyen hatékonyan bánnak a legfontosabb erőforrásokkal, az idővel és a tárral.

Tekintsük például azokat a problémákat melyek eldöntésére ismert polinom időigényű algoritmus (itt most a Chuch-Turing tézis alapján mondhatnánk Turig gépet is). Ezek a problémák alkotják a **P** bonyolultsági osztályt. Vannak azonban olyan problémák melyek eldöntésére nem ismert polinom időigényű algoritmus (azaz Turing-gép), viszont a Turing-gép egy kiterjesztésével, a nemdeterminisztikus Turing-géppel, már polinom időben eldönthetők. Ilyen például a korábban látott SAT probléma is. Ezek a problémák alkotják az **NP** bonyolultsági osztályt.

A bonyolultságelmélet egyik legfontosabb témaköre a **P** és a **NP** nyelvosztályok közötti határ vizsgálata. Az, hogy $\mathbf{P} \subseteq \mathbf{NP}$ könnyen következik abból, hogy a determinisztikus Turing-gépek tulajdonképpen speciális nemdeterminisztikus Turing-gépek. A bonyolultságelmélet egyik legnagyobb kihívása (és egyben kudarca is), hogy eddig nem sikerült bizonyítani azt, hogy a fenti tartalmazás valódi. Az előadáson közelebbről megvizsgáljuk az előbb említett **P** és **NP** osztályok kapcsolatát és megnézünk néhány fontosabb tárbonyolultsági osztályt is.

1.4. Az előadás felépítése

Az előadás az alábbi főbb témakörökre bontható.

Formális nyelvi alapismeretek: formális nyelvek és véges reprezentációik, a Chomsky-hierarchia, reguláris nyelvek és környezetfüggetlen nyelvek, ezek eldönthető tulajdonságai.

Turing-gépek: definíciók, Turing-gépek által eldönthető (rekurzív) és felismerhető (rekurzívan felsorolható) nyelvek.

További algoritmus modellek és ezek ekvivalenciája.

A Turing-gépek megállási problémája és néhány algoritmikusan eldönthetetlen probléma.

Időbonyolultsági osztályok: P és **NP**.

A „nehezen" megoldható problémák: NP teljesség és visszavezetés.

Tárbonyolultsági osztályok.

2. fejezet

Formális nyelvi alapismeretek

2.1. Nyelvek és véges reprezentációik

Definíció Legyen Σ egy véges, nem üres halmaz. Σ-t ábécének, az elemeit pedig betűknek nevezzük.

Σ betűinek egy tetszőleges véges (akár üres) sorozatát Σ-feletti szónak nevezzük. Σ^* jelöli az összes Σ-feletti szót, Σ^+ a $\Sigma^* - \{\varepsilon\}$ halmazt, $l(u)$ az $u \in \Sigma^*$ szó hosszát, $l_a(u)$ pedig az u-beli a betűk számát. A 0 hosszú szót üres szónak nevezzük (jele: ε). Σ-feletti nyelven a Σ^* egy részhalmazát értjük. □

Példa Legyen $\Sigma = \{0, 1\}$. Akkor $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, \ldots\}$. Az alábbi nyelvek pedig mind Σ^* részhalmazai:

$$\varnothing, \quad \{0, 111, 0^5, 1^{10}\}, \quad \{\varepsilon\}, \quad \{\varepsilon, 0, 1, 10, 11\}$$

$$\{(01)^n : n \geq 0\}, \quad \{0^n 1^n : n \geq 0\}, \quad \{u \in \Sigma^* : l_0(u) = l_1(u)\}.$$ □

Mivel egy nyelv általában végtelen, kell találni egy véges reprezentációját a nyelvnek. Hogyan lehet megadni egy $L \subseteq \Sigma^*$ nyelvet?

- Algoritmussal, ami az inputjára adott $u \in \Sigma^*$ szóra „igen"-nel áll meg, ha $u \in L$ és „nem"-mel, ha $u \notin L$. Az ily módon megadható nyelveket nevezzük eldönthetőnek vagy rekurzívnak.
- Eljárással, ami az inputjára adott $u \in \Sigma^*$ szóra „igen"-nel áll meg, ha $u \in L$ és vagy nem áll meg, vagy „nem"-mel áll meg, ha $u \notin L$. Az ily módon megadható nyelveket nevezzük Turing felismerhetőnek vagy rekurzívan felsorolhatónak.

Megjegyzzük, hogy a Church-Turing tézis értelmében, a fenti definíciókban az „algoritmus" és az „eljárás" fogalmak helyett írhattunk volna Turing-gépet is.

Az a definíciókból látszik, hogy minden eldönthető nyelv felismerhető is. A fordított állításról viszont később látni fogjuk, hogy nem áll fenn.

A formális nyelvek további véges reprezentációi még többek között a nyelvtanok és a különböző véges sok állapottal rendelkező gépek. A következő részben ezeket az eszközöket ismertetjük.

2.1.1. Nyelvtanok

A nyelvtanok bevezetésének főbb motivációi:

- Természetes nyelvek szerkezetének modellezése (Chomsky, 1956)
- Programozási nyelvek szintaktikájának megadása (BNF - ALGOL specifikáció, 1960)

Definíció Nyelvtannak nevezünk egy olyan $G = (V, \Sigma, R, S)$ rendszert, ahol

- V véges nemüres halmaz, a nemterminálisok halmaza,
- Σ véges nemüres halmaz a terminálisok halmaza ($V \cap \Sigma = \varnothing$),
- S egy kitüntetett szimbólum a V-ből amit kezdőszimbólumnak nevezünk és
- R pedig $u \to v$ alakú szabályok véges halmaza, ahol $u, v \in (V \cup \Sigma)^*$ és u tartalmaz legalább egy nemterminálist. □

Most megnézzük, hogy hogyan lehet a nyelvtant szavak generálására felhasználni. Ha ezt tudjuk, akkor definiálni tudjuk, hogy mit értünk egy nyelvtan által generált nyelven.

Definíció Legyen $G = (V, \Sigma, R, S)$ egy nyelvtan. A G által meghatározott közvetlen levezetési relációt (jele $\Rightarrow$) a következőképpen definiáljuk. Legyen $u, v \in (V \cup \Sigma)^*$. $u \Rightarrow v$ pontosan akkor, ha létezik olyan $x, y, y', z \in (V \cup \Sigma)^*$, amelyre $u = xyz, v = xy'z$ és $y \to y' \in R$.

A G által meghatározott levezetési reláció (jele $\Rightarrow^*$) pedig a következőképpen adódik. $u \Rightarrow^* v$ pontosan akkor, ha u megkapható v-ből a közvetlen levezetési reláció véges sok (akár 0) számú alkalmazásával. Formálisan, $u \Rightarrow^* v$, ha létezik olyan $n \geq 0$ és $w_0, w_1, \ldots, w_n \in (V \cup \Sigma)^*$, hogy $u = w_0$, minden $0 \leq i \leq n-1$-re $w_i \Rightarrow w_{i+1}$ és $w_n = v$.

A G által generált nyelv (jele $L(G)$) azon Σ^*-beli szavak halmaza, melyek megkaphatók (levezethetők) a kezdőszimbólumból a levezetési reláció alkalmazásával: $L(G) = \{u \in \Sigma^* : S \Rightarrow^* u\}$. □

Példa

- Legyen $G = (V, \Sigma, R, S)$, ahol
  - $V = \{S\}$
  - $\Sigma = \{0, 1\}$
  - $R = \{S \to 01S, S \to \varepsilon\}$

  Például $S \Rightarrow 01S \Rightarrow 0101S \Rightarrow 0101$, vagyis $S \Rightarrow^* 0101$. Tehát $0101 \in L(G)$. A G által generált nyelv: $L(G) = \{(01)^n : n \geq 0\}$.

- Legyen $G = (V, \Sigma, R, S)$, ahol
  - $V = \{S\}$
  - $\Sigma = \{0, 1\}$
  - $R = \{S \to 0S1, S \to \varepsilon\}$

  Például $S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 0011$, vagyis $S \Rightarrow^* 0011$. Tehát $0011 \in L(G)$. A G által generált nyelv: $L(G) = \{0^n 1^n : n \geq 0\}$.

□

A nyelvtanoknak négy fő típusát különböztetjük meg. Legyen $G = (V, \Sigma, R, S)$ egy nyelvtan. Azt mondjuk, hogy G

Jobblineáris (3-as típusú) nyelvtan, ha R-ben minden szabály $A \to uB$ vagy $A \to u$ alakú, ahol $A, B \in V$ és $u \in \Sigma^*$.

Környezetfüggetlen (2-es típusú) nyelvtan, ha R-ben minden szabály $A \to u$ alakú, ahol $u \in (V \cup \Sigma)^*$.

Környezetfüggő (1-es típusú) nyelvtan, ha R-ben minden szabály $\alpha A \beta \to \alpha \gamma \beta$ alakú, ahol $\alpha, \beta, \gamma \in (V \cup \Sigma)^*$ és $\gamma \neq \varepsilon$ (kivéve az $S \to \varepsilon$ szabályt, de ez esetben S nem fordulhat elő szabály jobb oldalán).

Általános (0-ás típusú) nyelvtan, ha a szabályokra semmilyen megkötés nincsen.

Ha L egy i-típusú ($0 \leq i \leq 3$) nyelvtannal generálható nyelv, akkor L-et i-típusú nyelvnek nevezzük. Az összes i-típusú nyelv osztályát $\mathcal{L}_i$-vel jelöljük.

Az $\mathcal{L}_0, \mathcal{L}_1, \mathcal{L}_2$ és $\mathcal{L}_3$ nyelvosztályok alkotják a Chomsky-féle hierarchiát. Az világos, hogy $\mathcal{L}_3 \subseteq \mathcal{L}_2 \subseteq \mathcal{L}_0$ és $\mathcal{L}_1 \subseteq \mathcal{L}_0$. Az, hogy $\mathcal{L}_2 \subseteq \mathcal{L}_1$ is fennáll abból következik, hogy minden környezetfüggetlen nyelvtanhoz megadható egy vele ekvivalens (ugyanazt a nyelvet generáló) úgynevezett ε-mentes nyelvtan mely egyben környezetfüggő nyelvtan is. Igaz továbbá, hogy az összes fenti tartalmazás valódi, vagyis adódik a következő tétel.

Tétel (Chomsky-féle hierarchia) $\mathcal{L}_3 \subset \mathcal{L}_2 \subset \mathcal{L}_1 \subset \mathcal{L}_0$.

Ismert, hogy minden $\mathcal{L}_1$-beli nyelv eldönthető és $\mathcal{L}_0$ pedig megegyezik a felismerhető nyelvek osztályával.

2.2. Jobblineáris nyelvek

Jobblineáris nyelvnek nevezünk egy nyelvet, ha az eleme az $\mathcal{L}_3$ osztálynak. A továbbiakban megnézzük, hogy ezeket a nyelveket, a nyelvtanokon kívül, milyen véges eszközökkel lehet még megadni.

2.2.1. Véges automaták

Ebben a részben a véges automatákat vizsgáljuk meg közelebbről.

A véges automaták olyan egyszerű véges sok állapottal rendelkező gépek, melyek modelljei a mindennapjainkban elforduló egyszerű gépeknek: liftek, elektromos ajtók, elektronikus termosztátok vezérlőegységeinek. Ezekkel az eszközökkel pontosan az $\mathcal{L}_3$-beli nyelveket lehet felismerni (eldönteni).

Definíció Formálisan a véges automata (DFA) egy $(Q, \Sigma, \delta, q_0, F)$ rendszer, ahol

- Q az állapotok véges, nemüres halmaza,
- Σ : egy ábécé a bemenő jelek (betűk) véges, nemüres halmaza,
- $\delta : Q \times \Sigma \to Q$ az átmeneti függvény,
- $q_0 \in Q$: a kezdőállapot,
- $F \subseteq Q$ pedig a végállapotok halmaza. □

Egy $M = (Q, \Sigma, \delta, q_0, F)$ véges automata működését egy $u = a_1 \dots a_n \in \Sigma^*$ szón $(a_1, \dots, a_n \in \Sigma)$ a következőképpen írható le. M kezdetben a q_0 kezdőállapotban van, majd miután elolvasta az a_1 betűt - az átmenetfüggvénye alapján - átmegy egy q_1 állapotba. Ezután elolvassa az a_2 betűt és átmegy egy q_2 állapotba. Ezt addig folytatja, amíg el nem éri az utolsó betűt, amit elolvasva átmegy egy q_n állapotba. Ha a $q_n \in F$ akkor M elfogadja az u szót, egyébként pedig elutasítja. Az M által felismert nyelv azon Σ^*-beli szavak halmaza, melyeket M elfogad. Formálisan a fentiek a következőképpen írhatók le.

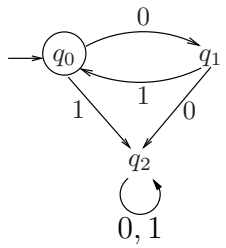
Definíció Legyen $M = (Q, \Sigma, \delta, q_0, F)$ egy véges automata és $u = a_1 \dots a_n \in \Sigma^*$ $(a_1, \dots, a_n \in \Sigma)$. Azt mondjuk, hogy M elfogadja u-t, ha van olyan $q_0, q_1, \dots, q_n$ állapotsorozat, melyre minden $1 \leq i \leq n$-re $q_i = \delta(q_{i-1}, a_i)$ és $q_n \in F$. Az M által felismert nyelv: $L(M) = \{u \in \Sigma^* : M \text{ elfogadja } u\text{-t}\}$. □

Példa Legyen $M_1 = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, \{q_0\})$, ahol

δ :

	0	1
q_0	q_1	q_2
q_1	q_2	q_0
q_2	q_2	q_2

M_1 :



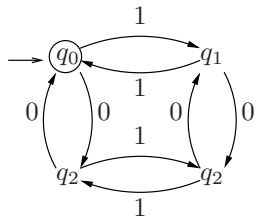
Könnyen látható, hogy $L(M_1) = \{(01)^n : n \geq 0\}$. □

Példa Legyen $M_2 = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, \{q_0\})$, ahol

δ :

	0	1
q_0	q_3	q_1
q_1	q_2	q_0
q_2	q_1	q_3
q_3	q_0	q_2

M_2 :



Látható, hogy $L(M_2)$ pontosan azokat az $u \in \Sigma^*$ szavakat tartalmazza, melyekben páros számú 1-es és páros sok 0 van. □

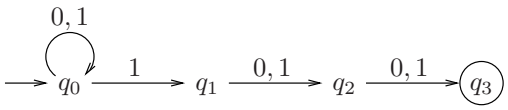
A véges automatával felismerhető nyelvek osztályát felismerhető nyelveknek nevezzük. Létezik a véges automatáknak egy általánosabb verziója is. Ezek a véges nemdeterminisztikus automaták. A nemdeterminizmus itt azt jelenti, hogy az automata a szó olvasása közben egy bizonyos ponton egynél több állapotba is átmehet, vagy éppen nem tud átmenni egy állapotba sem. Ily módon az automata számítási sorozata ezen a ponton több részre ágazik és minden ágon egy külön számítása indul az automatának a még el nem olvasott részszón. Ha viszont nem tud az automata az adott ponton átmenni egy másik állapotba, akkor a szóban forgó ágon az automata számítása befejeződik, vagyis azon az ágon a számítási sorozat „elhal". Egy véges nemdeterminisztikus automata akkor fogad el egy szót, ha a szón az összes lehetséges q_0-ból induló számítási sorozata közül legalább egy végállapotban végződik.

Definíció Formálisan a véges nemdeterminisztikus automata (NFA) egy $M = (Q, \Sigma, \delta, q_0, F)$ rendszer, ahol Q, Σ, q_0 és F ugyanazok mint a DFA esetében, δ viszont nem Q-ba, hanem Q részhalmazainak halmazába képez, azaz $\delta : Q \times \Sigma \to \mathcal{P}(Q)$.

Legyen $u = a_1 \dots a_n \in \Sigma^*$ $(a_1, \dots, a_n \in \Sigma)$ egy szó. Azt mondjuk, hogy M elfogadja u-t, ha van olyan $q_0, q_1, \dots, q_n$ állapotsorozat, melyre minden $1 \leq i \leq n$-re $q_i \in \delta(q_{i-1}, a_i)$ és $q_n \in F$. Az M által felismert nyelv: $L(M) = \{u \in \Sigma^* : M$ elfogadja u-t$\}$. □

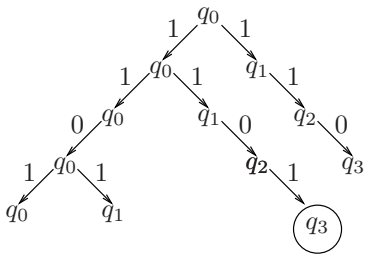
Példa Legyen $M_3 = (\{q_0, q_1, q_2, q_3\}, \{0,1\}, \delta, q_0\{q_3\})$, ahol

δ	0	1
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_2\}$	$\{q_2\}$
q_2	$\{q_3\}$	$\{q_3\}$
q_3	$\varnothing$	$\varnothing$



$L(M_3)$ pontosan azokat az $u \in \Sigma^*$ szavakat tartalmazza melyekben a szó végétől számolt harmadik pozíció 1-es.

M_3 különböző számítási sorozatai az $u = 1101$ szón a következő fával szemléltethetőek.



□

Az NFA definíciójából látszik, hogy az ilyen automata azt is megteheti, hogy egy bizonyos állapotban az adott betű olvasásakor nem megy át semmilyen új állapotba. Látszik továbbá, hogy minden DFA egyben speciális NFA is, tehát minden nyelv ami felismerhető DFA-val az felismerhető NFA-val is. A meglepő inkább az lehet, hogy a nemdeterminizmus nem nyújt plusz számítási erőt az automatának. Bizonyítás nélkül közöljük az alábbi tételt.

Tétel A véges nemdeterminisztikus automatával felismerhető nyelvek megegyeznek a véges (determinisztikus) automatával felismerhető nyelvekkel.

Most azt mutatjuk meg, hogy a véges automaták ugyanakkora számítási erővel rendelkeznek mint a 3-as típusú, azaz jobbllineáris nyelvtanok.

Tétel A felismerhető nyelvek osztálya megegyezik az $\mathcal{L}_3$ osztállyal.

Bizonyítás (vázlat) Csak az egyik irányt bizonyítjuk, nevezetesen, hogy minden jobblineáris nyelvtannal generálható L nyelvhez megadható olyan DFA ami L-et ismeri fel. A fordított irányú állítás hasonlóan bizonyítható.

Legyen $L \in \mathcal{L}_3$. Akkor van egy olyan $G = (V, \Sigma, R, S)$ jobblineáris nyelvtan, hogy $L = L(G)$. Ebből a nyelvtanból megkonstruálunk egy M NFA-t úgy, hogy $L(M) = L$ teljesüljön. Az előbbi tétel alapján ebből az NFA-ból már megkonstruálható egy DFA ami szintén L-et ismeri fel.

Az általánosság megsértése nélkül feltehetjük, hogy R-ben minden szabály $A \to aB$ illetve $A \to \varepsilon$ alakú, ahol $a \in \Sigma$ (eliminálni kell az ún. láncszabályokat és új nemterminálisok bevezetésével el kell „törni” a hosszú szavakat a szabályok

jobb oldalán).

Legyen $M = (Q, \Sigma, \delta, q_0, F)$ az az NFA, ahol $Q = V$, $q_0 = S$, $F = \{A \in V \mid A \to \varepsilon \in R\}$ és δ a következőképpen van definiálva. Minden $A \in V$-re és $a \in \Sigma$-ra $\delta(A, a) = \{B \in V \mid A \to aB \in R\}$.

Nem nehéz belátni, hogy ebben az esetben minden $u \in \Sigma^*$ szóra, $u \in L(G)$ akkor és csak akkor teljesül, ha $u \in L(M)$ is teljesül (indukcióval az u hossza szerint). Azt kaptuk tehát, hogy $L = L(M)$, amit bizonyítani akartunk. □

2.2.2. A felismerhető nyelvek zártsági tulajdonságai

Legyen L_1 és L_2 két nyelv. Ezek konkatenációját a következőképpen definiáljuk: $L_1 \cdot L_2 = \{uv : u \in L_1, v \in L_2\}$. $L_1 \cdot L_2$ helyett általában L_1L_2-t írunk. L_1 (Kleene) iteráltja a következőképpen adódik: $L_1^* = \{u_1 \dots u_n : n \geq 0, u_1, \dots, u_n \in L_1\}$. Ezek a műveletek az unióval együtt alkotják az úgynevezett reguláris műveleteket.

A felismerhet nyelvek egyik alapvető tulajdonsága, hogy zártak a Boole-féle műveletekre (unió, metszet és komplementer képzés) valamit a fent említett reguláris műveletekre. Először a Boole-műveletekre való zártságot bizonyítjuk.

Tétel A felismerhető nyelvek zártak a Boole-féle műveletekre.

Bizonyítás Legyen L_1 és L_2 két Σ feletti felismerhető nyelv. Akkor vannak olyan $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ és $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ véges automaták, hogy $L_1 = L(M_1)$ és $L_2 = L(M_2)$. Legyen $M = (Q, \Sigma, \delta, q_0, F)$, ahol $Q = Q_1 \times Q_2$, $q_0 = (q_1, q_2)$ és a $\delta : Q \times \Sigma \to Q$ függvény a következőképpen van definiálva. Minden $p_1 \in Q_1, p_2 \in Q_2$ állapotokra és $a \in \Sigma$-ra, $\delta((p_1, p_2), a) = (\delta_1(p_1, a), \delta_2(p_2, a))$.

Nem nehéz belátni, hogy ha F-et $F_1 \times Q_2 \cup Q_1 \times F_2$-nek definiáljuk, akkor egy olyan automatát kapunk, ami az $L_1 \cup L_2$ nyelvet ismeri fel. Másrészt, ha F-et $F_1 \times F_2$-nek definiáljuk, akor egy $L_1 \cap L_2$-t felismerő automatát kapunk.

Legyen továbbá $\bar{M} = (Q_1, \Sigma, \delta_1, q_1, \bar{F})$, ahol $\bar{F} = Q_1 - F$. Könnyű belátni, hogy $\bar{M}$ az $\bar{L}_1$ nyelvet ismeri fel. □

Most megmutatjuk, hogy a felismerhető nyelvek zártak a reguláris műveletekre.

Tétel A felismerhető nyelvek zártak a reguláris műveletekre.

Bizonyítás (vázlat) Legyen L_1 és L_2 két Σ feletti felismerhető nyelv. Azt, hogy $L_1 \cup L_2$ felismerhető már láttuk. Láttuk azt is, hogy a felismerhető nyelvek generálhatók jobblineáris nyelvtannal. Legyen tehát $G_1 = (V_1, \Sigma, R_1, S_1)$ és $G_2 = (V_2, \Sigma, R_2, S_2)$ két jobblineáris nyelvtan melyekre $L_1 = L(G_1)$ és $L_2 = L(G_2)$. Feltehetjük, hogy V_1 és V_2 diszjunktak. A két nyelv konkatenációját az a $G = (V, \Sigma, R, S_1)$ nyelvtan generálja, melyben $V = V_1 \cup V_2$ és R-et pedig a következőképpen kapjuk. R tartalmazza G_2 összes szabályát, valamint G_1 azon szabályait, melyek jobboldalai nemterminálissal végződnek. Továbbá G_1 minden olyan $u \to v$ szabályára melyre $v \in \Sigma^*$, G-be felvesszük az $u \to vS_2$

szabályt.

L_1 iteráltját az a $G' = (V_1 \cup \{S\}, \Sigma, R_1', S)$ nyelvtan generálja, ahol R_1'-et a következőképpen kapjuk. R_1' tartalmazza az R_1-beli szabályokat plusz az $S \to \varepsilon$ és az $S \to S_1$ szabályt. Továbbá G_1 minden olyan $u \to v$ szabályára melyre $v \in \Sigma^*$, G-be felvesszük az $u \to vS_1$ szabályt. □

Egy $\Sigma = \{a_1, \ldots, a_n\}$ $(n \geq 1)$ ábécé feletti nyelvet regulárisnak nevezzük ha megkapható az $\varnothing$ illetve az $\{a_1\}, \ldots, \{a_n\}$ nyelvekből a reguláris műveletek véges számú alkalmazásával. Később látni fogjuk, hogy a reguláris nyelvek megegyeznek a felismerhető nyelvekkel.

2.2.3. Reguláris kifejezések

Az alábbiakban röviden ismertetjük a reguláris kifejezéseket. Mint látni fogjuk ezek a kifejezések éppen a reguláris nyelvek jelölésére szolgálnak.

Definíció Legyen Σ egy ábécé. A (Σ feletti) reguláris kifejezések REG halmaza a legszűkebb olyan halmaz, melyre teljesülnek a következő állítások.

1. $\varnothing, \varepsilon \in REG$.
2. Minden $a \in \Sigma$-ra $a \in REG$.
3. Ha $R_1, R_2 \in REG$, akkor $(R_1 + R_2), (R_1 \cdot R_2), (R_1^*) \in REG$.
4. REG minden eleme megkapható a fenti szabályok véges számú alkalmazásával.

Az R reguláris kifejezés által jelölt nyelvet $|R|$-el jelöljük. Értelemszerűen,

1. ha $R = a$ valamely $a \in \Sigma$-ra, akkor $|R| = \{a\}$,
2. ha $R = \varepsilon$, akkor $|R| = \{\varepsilon\}$,
3. ha $R = \varnothing$, akkor $|R| = \varnothing$,
4. ha $R = (R_1 + R_2)$ valamely $R_1, R_2 \in REG$-re, akkor $|R| = |R_1| \cup |R_2|$,
5. ha $R = (R_1 \cdot R_2)$ valamely $R_1, R_2 \in REG$-re, akkor $|R| = |R_1| \cdot |R_2|$,
6. ha $R = (R_1^*)$ valamely $R_1 \in REG$-re, akkor $|R| = |R_1|^*$.

A reguláris kifejezések könnyebb olvashatósága érdekében el szeretnénk hagyni a kifejezésekből bizonyos zárójeleket, ezért megegyezünk abban, hogy * a legerősebb precedenciájú reguláris kifejezésbeli műveleti jel, utána jön a $\cdot$ és végül a $+$ jel. Továbbá a $\cdot$ jelet általában elhagyjuk.

Példa Legyen $\Sigma = \{0, 1\}$ és tekintsük az alábbi Σ feletti reguláris kifejezéseket és az általuk jelölt nyelveket.

1. $|(0+\varepsilon)(1+\varepsilon)| = \{\varepsilon, 0, 1, 01\}$,
2. $|(0+1)(0+1)(0+1)(0+1)^*| = \{u \in \Sigma | l(u) \geq 3\}$,
3. A $|(\varepsilon+1)(01)^*(\varepsilon+0)|$ nyelv azokat a Σ feletti szavakat tartalmazza, melyekben a 0-k és 1-k egymást váltogatják,
4. A $|((0+1)1)^*|$ nyelv azokat a szavakat tartalmazza, melyeknek minden második pozíciója 1-es. □

Könnyen belátható a következő tétel.

Tétel Legyen Σ egy ábécé. Egy $L \subseteq \Sigma^*$ nyelv akkor és csak akkor reguláris, ha van olyan R Σ feletti reguláris kifejezés, melyre $|R| = L$.

Kleene tételeként tartjuk számon az alábbi állítást.

Tétel Egy nyelv akkor és csak akkor reguláris, ha felismerhető véges automatával.

Bizonyítás (vázlat) Az, hogy egy Σ feletti reguláris kifejetéssel jelölt nyelv felismerhető, könnyen látható, ha meggondoljuk, hogy az $\varnothing$ és minden $a \in \Sigma$-ra az $\{a\}$ felismerhető nyelvek. Továbbá, mint azt már korábban láttuk, a felismerhető nyelvek zártak a reguláris műveletekre.

Másrészt, ha egy $L \subseteq \Sigma^*$ nyelv felismerhető, akkor az őt felismerő automatából megkonstruálható egy reguláris kifejezés, mely L-et jelöli (nem bizonyítjuk). □

Az eddigiek során megismertük a felismerhető nyelvek három különböző jellemzését. Kimondhatjuk tehát az alábbi következményt.

Következmény Legyen L egy tetszőleges Σ ábécé feletti nyelv. Akkor a következő három állítás ekvivalens.

1. L felismerhető.
2. L generálható 3-as típusú nyelvtannal.
3. L jelölhető valamely Σ feletti reguláris kifejezéssel.

2.2.4. A felismerhető nyelvekkel kapcsolatos eldönthetőségi kérdések

Az alábbiakban bemutatunk néhány a felismerhető nyelvekkel kapcsolatos eldönthető problémát. Az első ilyen probléma így szól. Adott egy L felismerhető nyelv. Tartalmaz-e az L legalább egy szót?

Első hallásra egy kicsit furcsa lehet ezt a kérdést feltenni, hisz egy nyelv pontosan akkor tartalmaz legalább egy szót, ha a nyelv nem maga az üres halmaz. De amint azt már korábban láttuk, a formális nyelvek általában végtelen elemszámú

nyelvek, és ily módon egy nyelv általában nem az elemei explicite felsorolásával, hanem inkább a nyelv valamilyen véges reprezentációjával adott. Mint azt korábban láttuk, a felismerhető nyelvek véges reprezentációi lehetnek például a jobblineáris nyelvtanok, a véges automaták vagy a reguláris kifejezések. Tehát amikor el akarunk dönteni egy formális nyelvvel kapcsolatos problémát, akkor feltesszük, hogy a nyelv valamely véges reprezentációjával adott.

Tétel Legyen L egy M felismerhető nyelv. Eldönthető, hogy L az üreshalmaz-e vagy sem.

Bizonyítás Tegyük fel, hogy az L egy M véges automatával adott. Számoljuk ki, hogy M mely állapotai érhetők el a kezdőállapotból (egy állapot elérhető, ha van olyan input szó, melynek az elolvasása után az automata az adott állapotba kerül). Legyen az így kapott halmaz D. Ezek után ha D nem tartalmaz egyetlen végállapotot sem, akkor $L = \varnothing$, egyébként pedig $L \neq \varnothing$.

Megjegyzés: A D halmaz kiszámolása az M automata méretének függvényében négyzetes időigényű. Tehát, ha M n darab állapottal rendelkezik, akkor D kiszámolása nagyságrendileg n^2 lépést vesz igénybe. □

Mint azt bizonyos esetekben láttuk is, a felismerhető nyelvek különböző reprezentációi egymásba alakíthatók. Így például ha a nyelvünk nem véges automatával adott, de mi csak véges automatára ismerjük az adott problémát eldöntő algoritmust, akkor megtehetjük, hogy átalakítjuk az adott reprezentációt (nemdeterminisztikus automatát, reguláris kifejezést vagy nyelvtant) véges automatává. Meg kell azonban jegyezni, hogy ebben az esetben a kérdés eldöntésének az időigényéhez hozzá kell venni az adott reprezentáció átalakításának időigényét is.

Persze nem feltétlenül szükséges egy nyelv reprezentációját átalakítani. A fenti problémát például akkor is el tudjuk dönteni, ha a nyelv egy reguláris kifejezéssel adott.

Feladat Hogyan lehetne közvetlenül eldönteni, hogy egy R reguláris kifejezéssel adott L nyelv az üres halmaz-e vagy sem?

A következő feladat annak eldöntése, hogy egy L nyelv tartalmaz-e egy adott u szót.

Tétel Legyen L egy Σ feletti felismerhető nyelv és legyen $u \in \Sigma^*$ egy szó. Eldönthető, hogy u eleme-e L-nek.

Bizonyítás Tegyük fel, hogy az L az M véges automatával adott. Egyszerűen olvastassuk el M-el u-t. Ha ezek után M végállapotba kerül, akkor u eleme L-nek, egyébként pedig nem. Könnyen látható, hogy ezen probléma eldöntésének az időigénye az u méretével lineárisan arányos.

Most azt mutatjuk meg, hogyan lehet eldönteni azt, hogy egy L felismerhető nyelv végtelen-e.

Tétel Legyen L egy Σ feletti felismerhető nyelv. Eldönthető, hogy L végtelen-e.

Bizonyítás Tegyük fel, hogy az L az M véges automatával adott. Legyen az automata állapotszáma n. Vizsgáljuk meg, hogy van-e olyan $u \in L$ szó, melyre $n \leq l(u) \leq 2n$ teljesül. Ezt könnyen meg tudjuk csinálni úgy, hogy egyszerűen felsoroljuk Σ^* legalább n hosszú de $2n$-nél rövidebb elemeit, és eldöntjük, hogy valamelyik szó ezek közül eleme-e L-nek (korábban láttuk, hogy ez eldönthető). Ha a vizsgálat pozitív eredménnyel zárul, akkor L végtelen elemszámú, egyébként pedig nem.

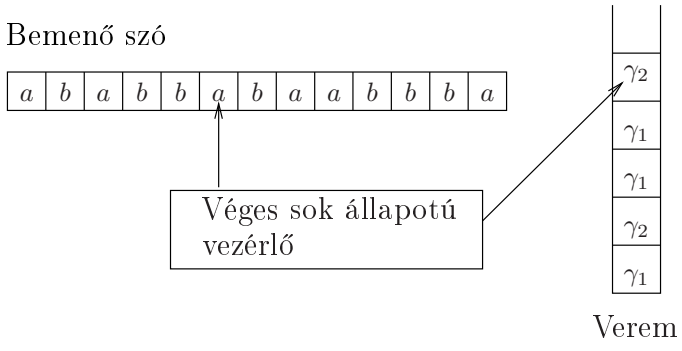
Két véges automatát ekvivalensnek nevezünk, hogy ha mindketten ugyanazt a nyelvet ismerik fel. Most megmutatjuk, hogy a véges automaták ekvivalencia-problémája eldönthető. Így az is eldönthető lesz, hogy két felismerhető nyelv megegyezik-e.

Tétel Legyen M_1 és M_2 két véges automata. Eldönthető, hogy $L(M_1) = L(M_2)$ fennáll-e.

Bizonyítás Legyen $M = (Q_1 \times Q_2, \Sigma, \delta, (q_1, q_2), F)$ az a véges automata amit a felismerhető nyelvek unióra és metszetre való zártságának bizonyításakor konstruáltunk. Nem nehéz belátni, hogy ha F-nek az $F_1 \times (Q_2 - F_2) \cup (Q_1 - F_1) \times F_2$ halmazt választjuk, akkor M az $(L(M_1) - L(M_2)) \cup (L(M_2) - L(M_1))$ halmazt ismeri fel. Világos az is, hogy $L(M_1) = L(M_2)$ akkor és csak akkor teljesül, ha $L(M) = \varnothing$. Azt viszont láttuk, hogy ez az utóbbi egyenlőség eldönthető, amiből következik, hogy $L(M_1) = L(M_2)$ is eldönthető.

2.3. Környezetfüggetlen nyelvek

Egy nyelv környezetfüggetlen (CF), ha van őt generáló környezetfüggetlen nyelvtan. Ebben a részben definiáljuk a veremautomatákat, melyek olyan, a véges automatáknál erősebb kiszámító erővel rendelkező gépek, melyek pontosan a környezetfüggetlen nyelveket ismerik fel. Megvizsgáljuk továbbá a CF nyelvek néhány zártsági tulajdonságát valamint néhány CF nyelvekkel kapcsolatos eldöntési problémát.



2.3.1. Veremautomaták

A veremautomaták annyival tudnak többet a véges automatáknál, hogy a számítások során felhasználhatnak egy potenciálisan végtelen vermet is.

Ezzel a képességgel felvértezve már alkalmasak olyan nyelvek felismerésére, melyeket a véges automaták képtelenek felismerni. Tekintsük például a korábban látott (15. oldal) $L = \{0^n 1^n \mid n \geq 0\}$ nyelvet és az őt generáló nyelvtant. Látható, hogy az L-et generáló nyelvtan környezetfüggetlen. Az is könnyen belátható viszont, hogy ez a nyelv nem felismerhető. Legyen ugyanis M egy automata amiről feltesszük, hogy felismeri L-et. Legyen N az automata állapotszáma. Nyilván akkor M elfogadja az $u = 0^N 1^N$ szót is. Továbbá, könnyen meggondolható, hogy az u felírható $xyz1^N$ alakban $(x, y, z \in \{a\}^*)$ úgy, hogy $l(y) \neq 0$ és az x valamint az xy részszó elolvasása után M ugyanabban az állapotban van. Nem nehéz belátni azt sem, hogy ez esetben M-nek el kell fogadnia az összes $xy^i z1^N$ $(i \geq 0)$ alakú szót is, ami ellentmondás. Tehát az L nyelv nem lehet felismerhető.

M kudarcának az oka az, hogy mivel neki csak véges számú állapota van, ezért ő csak véges számú 0-t tud megjegyezni (megszámolni) az állapotaiban (nevezetesen legfeljebb csak annyit ahány állapota van a kezdőállapoton kívül). Így viszont ha a vizsgálandó szóban túl sok az 0, akkor az automata nem tudja leellenőrizni azt, hogy a 0-k illetve 1-k száma megegyezik-e. Ezzel szemben a veremautomata képes arra hogy az L-beli szavakat felismerje úgy, hogy először az összes 0-t belerakja egy verembe, majd minden egyes 1-es elolvasásakor kivesz egy 0-t veremből. Ha egyszerre fogy el a bemenő szó még elolvasatlan része illetve ürül ki a verem akkor a veremautomata elfogadja a bemenő szót.

Definíció Formálisan a veremautomata egy $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ rendszer, ahol Q, Σ, q_0 és F ugyanazok mint a véges automata esetében, Γ a veremjelek ábécéje, Z_0 a verem kezdőszimbólum és $\delta : Q \times \Sigma_\varepsilon \times \Gamma \to \mathcal{P}(Q \times \Gamma^*)$ az átmenetfüggvény (itt Σ_ε a $\Sigma \cup \{\varepsilon\}$ halmazt jelöli).

P működésének pillanatfelvételét egy adott $u \in \Sigma^*$ szón konfigurációnak nevezzük. Egy konfiguráció három dologról ad információt: milyen állapotban van a gép, mi az amit még nem olvasott el a bemenő szóból és mi van a veremben. Formálisan egy konfiguráció egy (q, w, γ) hármas, ahol $q \in Q$, $w \in \Sigma^*$ és $\gamma \in \Gamma^*$.

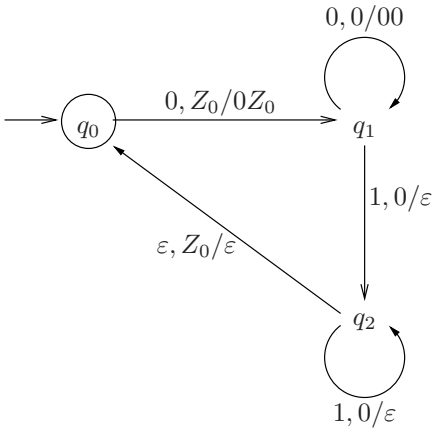
Nyilvánvaló, hogy P csak akkor mehet át egy konfigurációból egy másikba, ha ezt az átmenetfüggvénye megengedi. Ennek alapján P egy konfigurációátmenetét (jele $\vdash$) a következőképpen definiáljuk. Legyen $(q, vw, X\beta)$ és $(p, w, \alpha\beta)$ két konfigurációja P-nek, ahol $v \in \Sigma_\varepsilon$ és $X \in \Gamma$. Akkor $(q, vw, X\beta) \vdash (p, w, \alpha\beta)$, feltéve, hogy (p, α) eleme $\delta(q, v, X)$-nek.

Legyen C és C' a P két konfigurációja. Azt mondjuk, hogy a P véges sok lépésben eljut a C-ből a C'-be (jele $C_1 \vdash^* C_n$), ha van olyan $n \geq 0$ és $C_1, \ldots, C_n$ konfigurációsorozat, hogy $C_1 = C$, $C' = C_n$ és minden $1 \leq i \leq n$-re, $C_i \vdash C_{i+1}$.

A P által felismert nyelvet két módon is definiáljuk. A P által végállapottal

felismert nyelv: $L_f(P) = \{u \in \Sigma^* \mid (q_0, u, Z_0) \vdash^* (q, \varepsilon, \alpha)\}$, ahol $q \in F$. A P által üres veremmel felismert nyelv: $L_\varnothing(P) = \{u \in \Sigma^* \mid (q_0, u, Z_0) \vdash^* (q, \varepsilon, \varepsilon)\}$, ahol $q \in Q$, tehát az automatának nem kell feltétlenül végállapotba érkeznie, de a vermet ki kell ürítenie. Mivel üres veremmel történő felismeréskor az automata végállapothalmazának nincs szerepe, így ezt a halmazt ilyenkor nem kell megadni. □

Példa Tekintsük megint az $L = \{0^n 1^n \mid n \geq 0\}$ nyelvet. L-et a következő P veremautomata ismeri fel (végállapottal).



Megjegyezzük, hogy $L_\varnothing(P) = L - \{\varepsilon\}$, mivel az üres szón mint bemeneten P nem tudja kiüríteni a vermét, ezáltal üres veremmel nem tudja felismerni ε-t.

□

Bár a fenti definícióban két különböző módon definiáltuk a veremautomaták által felismert nyelvet, megmutatható, hogy ez nem jelent különbséget a felismert nyelvek osztályait tekintve. Bebizonyítható az alábbi tétel.

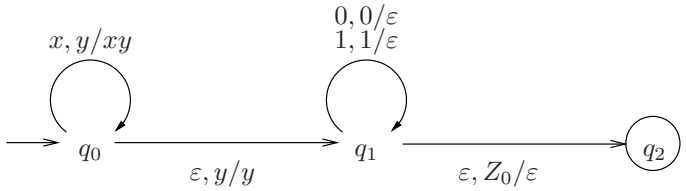
Tétel A veremautomatákkal végállapottal felismerhető illetve a veremautomatákkal üres veremmel felismerhető nyelvek osztálya megegyezik.

Amint az a fenti definícióból is látszik, a veremautomata egy nemdeterminisztikus modell csakúgy, mint a véges nemdeterminisztikus automata. Informálisan a determinisztikus veremautomata az aktuális konfigurációból mindig csak legfeljebb egy következő konfigurációba léphet. Formálisan egy $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ veremautomata determinisztikus ha az átmenetfüggvényére az alábbi két feltétel teljesül. Legyen $q \in Q$.

1. Minden $a \in \Sigma_\varepsilon, b \in \Gamma$ betűre, $|\delta(q, a, b)| \leq 1$.

2. Minden $a \in \Sigma, b \in \Gamma$ betűre, ha $\delta(q, a, b) \neq \varnothing$, akkor $\delta(q, \varepsilon, b) = \varnothing$.

Amíg a véges automaták esetében a determinisztikusság nem jelentett limitációt az automata felismerő erejére nézve, addig a determinisztikus veremautomaták határozottan kevesebb nyelvet képesek felismerni mint a nemdeterminisztikusak. Legyen például $L = \{uu^{-1} \mid u \in \{0,1\}\}$, ahol a w^{-1} szó a w megfordítását jelöli. Ez a nyelv felismerhető az alábbi (nemdeterminisztikus) veremautomatával, ahol az átmeneteken az x tetszőleges $\{0,1\}$, y pedig $\{0,1,Z_0\}$ halmazbeli elem.



Másrészt bebizonyítható, hogy L nem ismerhető fel determinisztikus veremautomatával. Adódik tehát a következő tétel.

Tétel A determinisztikus veremautomatával felismerhető nyelvek osztálya valódi részhalmaza a (nemdeterminisztikus) veremautomatával felismerhető nyelvek osztályának.

Most rátérünk a CF nyelvek és a veremautomaták közötti kapcsolat vizsgálatára. Nagyon fontos az alábbi tétel.

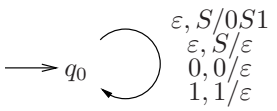
Tétel A környezetfüggetlen nyelvek osztálya megegyezik a veremautomatával felismerhető nyelvek osztályával.

Bizonyítás (vázlat) Csak az egyik irány bizonyítjuk, nevezetesen azt, hogy hogyan lehet egy L CF nyelvhez megadni egy olyan veremautomatatát ami pont L-et ismeri fel. Mivel L egy CF nyelv van olyan $G = (V, \Sigma, R, S)$ környezetfüggetlen nyelvtan melyre $L(G) = L$. Megadunk egy P üres veremmel felismerő veremautomatát, ami G olyan levezetéseit szimulálja, melyekben mindig a balról legelső nemterminális van helyettesítve (az ilyen levezetést hívjuk baloldali levezetésnek).

Formálisan P a következőképpen definiálható. Legyen $P = (\{q\}, \Sigma, V \cup \Sigma, \delta, q, S)$, ahol δ a következő függvény. Minden $A \in V$-re, $\delta(q, \varepsilon, A) = \{(q, \beta) \mid A \to \beta \in R\}$. Továbbá minden $a \in \Sigma$-ra, $\delta(q, a, a) = \{(q, \varepsilon)\}$.

Belátható, hogy egy $u \in \Sigma^*$ szót pontosan akkor ismer fel P, ha u levezethető S-ből. □

Példa Tekintsük ismét az $L = \{0^n 1^n : n \geq 0\}$ nyelvet és az őt generáló $G = (\{S\}, \{0,1\}, \{S \to 01S, S \to \varepsilon\}, S)$ nyelvtant. Ebből a nyelvtanból, a fenti tétel bizonyítása alapján, az alábbi L-et felismerő veremautomata konstruálható meg.



□

2.3.2. A környezetfüggetlen nyelvek zártsági tulajdonságai

Most bebizonyítunk néhány, a CF nyelvekkel kapcsolatos zártsági tulajdonságot.

Tétel A CF nyelvek zártak a reguláris műveletekre, de nem zártak a metszet és komplemens képzésre.

Bizonyítás A reguláris műveletekre való zártság a következőképpen bizonyítható. Legyen L_1 és L_2 két CF nyelv és $G_1 = (V_1, \Sigma, R_1, S_1)$ és $G_2 = (V_2, \Sigma, R_2, S_2)$ két CF nyelvtan melyekre $L_1 = L(G_1)$ és $L_2 = L(G_2)$. Feltehetjük, hogy $V_1 \cap V_2 = \varnothing$. G_1-ből és G_2-ből megkonstruáljuk az $L_1 \cup L_2$, $L_1 L_2$ illetve L_1^* nyelveket generáló nyelvtanokat az alábbi módon. Először vegyünk egy olyan S új nemterminálist ami nem fordul elő $V_1 \cup V_2$-ben. Ezután legyen

- $G_\cup = (V_1 \cup V_2, \Sigma, R_1 \cup R_2 \cup \{S \to S_1, S \to S_2\}, S)$,
- $G_{konk} = (V_1 \cup V_2, \Sigma, R_1 \cup R_2 \cup \{S \to S_1 S_2\}, S)$,
- $G_{iter} = (V_1, \Sigma, R_1 \cup \{S \to SS_1, S \to \varepsilon\}, S)$.

Az, hogy a CF nyelvek nem zártak a metszet képzésre következik abból, hogy az $L = \{a^n b^n c^n \mid n \geq 1\}$ nyelv nem környezetfüggetlen és L felírható két CF nyelv metszeteként: $L = L_1 \cap L_2$, ahol $L_1 = \{a^n b^n c^i \mid n \geq 1, i \geq 1\}$ és $L_1 = \{a^i b^n c^n \mid n \geq 1, i \geq 1\}$. Ezek után az, hogy a CF nyelvek nem zártak a komplemens képzésre már triviális.

2.3.3. A környezetfüggetlen nyelvekkel kapcsolatos eldönthetőségi kérdések

Ebben a részben megvizsgálunk néhány CF nyelvekkel kapcsolatos eldönthetőségi kérdést.

Tétel Eldönthető, hogy egy L CF nyelv üres-e.

Bizonyítás Legyen L egy környezetfüggetlen nyelv, és tegyük fel, hogy L egy $G = (V, \Sigma, R, S)$ CF nyelvtannal adott, azaz $L = L(G)$. Nevezzük G egy A nemterminálisát terminálónak, ha van olyan $u \in \Sigma^*$ szó, hogy $A \Rightarrow^* u$ teljesül. A következő algoritmussal kiszámítjuk G termináló nemterminálisainak V_t halmazát.

1. Legyen $V_1 = \{A \in V \mid$ van $A \to x$ szabály R-ben úgy, hogy $x \in \Sigma^*\}$; Legyen $i = 1$;
2. Legyen $V_{i+1} = V_i \cup \{A \in V \mid$ van $A \to \alpha$ szabály R-ben úgy, hogy $\alpha \in (V_i \cup \Sigma)^*\}$;

3. Ha $V_{i+1} = V_i$, akkor az algoritmus megáll, egyébként legyen $i = i + 1$ és ugrás 2-re;

Könnyen belátható, hogy van olyan $i_0 \geq 1$ szám, hogy $V_{i_0} = V_{i_0+1}$. Azt sem nehéz belátni, hogy ekkor $V_t = V_{i_0}$.

Ezek után $L = \varnothing$ akkor és csak akkor teljesül, ha $S \notin V_t$.

Megjegyezzük, hogy a fenti algoritmus futásának időigénye a G méretével négyzetesen arányos (itt most a méret alatt V számosságát értjük). □

Most azt mutatjuk meg, hogy hogyan dönthető el a tartalmazás probléma a CF nyelvek esetében. Ehhez először definiálunk egy a CF nyelvtanokon értelmezett normálformát, a Chomsky normálformát.

Deiníció Legyen $G = (V, \Sigma, R, S)$ egy CF nyelvtan. Azt mondjuk, hogy G Chomsky normálformában van, ha a következő feltételek teljesülnek. R-ben minden szabály $A \to BC$ vagy $A \to a$ alakú, ahol A, B, C nemterminálisok, a pedig egy terminális. Ezen feltétel alól csak az $S \to \varepsilon$ szabály lehet a kivétel, de ez esetben S nem fordulhat elő szabály jobb oldalán. □

Tétel Minden CF nyelvtanhoz megadható egy vele ekvivalens Chomsky normálformában lévő nyelvtan.

Nézzük, hogy ezek után hogyan lehet eldönteni a tartalmazás problémát egy CF nyelvtan esetében.

Tétel Legyen $L \subseteq \Sigma^*$ egy CF nyelvtan és $u \in \Sigma^*$ egy szó. Eldönthető, hogy $u \in L$ fennáll-e.

Bizonyítás Tegyük fel, hogy L egy $G = (V, \Sigma, R, S)$ Chomsky normálformában lévő CF nyelvtannal adott.

Ha $u = \varepsilon$, akkor $u \in L$ akkor és csak akkor, ha $S \to \varepsilon$ eleme az R-nek. Ez utóbbi eldönthető, tehát $u \in L$ is eldönthető. Egyébként pedig legyen $n > 0$ az u szó hossza. Nem nehéz belátni (n szerinti teljes indukcióval), hogy ha $S \Rightarrow^* u$ akkor a levezetés hossza $2n - 1$ (itt most a levezetés hosszán azt a számot értjük, ahányszor a közvetlen levezetési reláció alkalmazásra került a levezetésben). Ezek után ahhoz, hogy eldöntsük vajon $u \in L$ fennáll-e, elég megvizsgálni G S-ból induló $2n - 1$ hosszú levezetéseit. Ilyen levezetés pedig véges sok van.

Megjegyezzük, hogy ez az algoritmus n függvényében exponenciális, tehát a gyakorlatban nem igazán alkalmazható. Van azonban a kérdés eldöntésére polinom idejű algoritmus is, ilyen például az ún. CYK algoritmus, mely $\mathcal{O}(n^3)$ időben képes eldönteni ezt a problémát. □

A felismerhető nyelvekkel ellentétben a CF nyelvekkel kapcsolatban már meglepően kevés kérdés dönthető el. Az alább felsorolt problémák közül például egy sem eldönthető.

1. Üres-e két környezetfüggetlen nyelv metszete?

2. Megegyezik-e két környezetfüggetlen nyelv?
3. Megegyezik-e egy $L \subseteq \Sigma^*$ nyelv Σ^*-gal?

3. fejezet

A kiszámíthatóság elmélet alapjai

Most áttérünk annak vizsgálatára, hogy mely problémákat lehet algoritmukusan kiszámítani és melyeket nem. Ehhez bevezetünk egy eszközt, a Turing-gépet, mely az eddigi nyelveket felismerő eszközeink közül a legnagyobb kiszámítási erővel rendelkezik. Amint arról a bevezetésben szó volt, a Turing-gép az egyik legáltalánosabb algoritmusmodell. Ebben a részben megvizsgáljuk a Turing-gépek különböző változatait és a Turing-gép ekvivalenciáját néhány más algoritmusmodellel. Ezután rátérünk az eldönthető valamint a felismerhető nyelvek közötti kapcsolat vizsgálatára.

A későbbiekben majd azt is vizsgálni fogjuk, hogy egy Turing-gép mennyi lépésben illetve mekkora tár felhasználásával fogad el egy bemenő szót a szó hosszának függvényében. Igazából nem a pontos idő- és tárigényre lesz majd szükségünk, hanem ezek nagyságrendjére, ezért bevezetjük a következő fogalmakat.

Definíció Legyenek $f, g : N \to R_+$ függvények, ahol N a természetes számok, R_+ pedig a nemnegatív valós számok halmaza. Azt mondjuk, hogy f legfeljebb olyan gyorsan nő mint g (jelölése: $f(n) = \mathcal{O}(g(n))$) ha létezik olyan $c > 0$ szám és $n_0 \in N$, hogy $f(n) \leq c \cdot g(n)$ minden $n \geq n_0$ számra.

Példa $3n^3 + 5n^2 + 6 = \mathcal{O}(n^3)$, $123n^2 + 6235 = \mathcal{O}(n^2)$, $n^k = \mathcal{O}(2^n)$ minden $k \geq 0$-ra, $\log_2 n = \mathcal{O}(n)$, $\log \log n = \mathcal{O}(\log n)$.

3.1. Turing-gépek

Hasonlóan a véges automatához vagy a veremautomatához, a Turing-gép is egy véges sok állapottal rendelkező eszköz. A Turing-gép egy egy irányban végtelen szalagon dolgozik. A szalag tulajdonképpen a gép (korlátlan) memóriája. Kez-

detben a szalagon csak a bemenő szó van, mely a szalag bal végén helyezkedik el. A gép ún. író-olvasó feje a bemenő szó első betűjén áll és a gép a kezdőállapotában van. A szalag bemenő szón kívüli része csak üres ($\sqcup$) szimbólumokat tartalmaz.

A gép az író-olvasó fejet tetszőlegesen képes mozgatni a szalagon, a kikötés csak annyi, hogy a fej „nem eshet le" a szalag bal oldalán, azaz ha a fej a szalag legelején van, akkor a gép nem tudja a fejet balra léptetni. A gép képes továbbá a fej pozíciójában a szalag tartalmát kiolvasni és átírni. A gépnek van két kitüntetett állapota, a q_i és a q_n állapotok. Ha ezekbe az állapotokba kerül a gép, akkor rendre elfogadja illetve elutasítja a bemenő szót. Formálisan a Turing-gépet a következő módon definiáljuk.

Definíció A Turing-gép egy olyan $M = (Q, \Sigma, \Gamma, \delta, q_0, q_i, q_n)$ rendszer, ahol

- Q az állapotok véges, nemüres halmaza,
- $q_0, q_i, q_n \in Q$, q_0 a kezdő-, q_i az elfogadó és q_n az elutasító állapot,
- Σ és Γ ábécék, a bemenő jelek illetve a szalag szimbólumok ábécéje úgy, hogy $\Sigma \subseteq \Gamma$ és $\Gamma - \Sigma$ tartalmaz egy speciális $\sqcup$ szimbólumot,
- $\delta : (Q - \{q_i, q_n\}) \times \Gamma \to Q \times \Gamma \times \{L, R\}$ az átmenet függvény.

Úgy mint a veremautomaták esetében, a Turing-gép működésének fázisait is a gép konfigurációival írjuk le. A Turing-gép konfigurációja egy uqv szó, ahol $q \in Q$ és $u, v \in \Gamma^*$, $v \neq \varepsilon$. Ez a konfiguráció a gép azon állapotát tükrözi amikor a szalag tartalma uv (uv után szalagon már csak $\sqcup$ van), a gép a q állapotban van, és a gép író-olvasó feje a v első betűjére mutat. A gép kezdőkonfigurációja egy olyan $q_0u\sqcup$ szó, ahol u csak Σ beli betűket tartalmaz. Egy Turing-gép konfigurációátmenetét az alábbiak szerint definiáljuk.

Definíció Legyen $uqav$ egy konfiguráció, ahol $a \in \Gamma$ és $u, v \in \Gamma^*$. Ha $\delta(q, a) = (r, b, R)$, akkor $uqav \vdash ubrv'$, ahol $v' = v$, ha $v \neq \varepsilon$, különben $v' = \sqcup$. Most tegyük fel azt, hogy $\delta(q, a) = (r, b, L)$. Ebben az esetben ha $u \neq \varepsilon$, akkor $uqav \vdash u'rcbv'$, ahol $c \in \Gamma$ és $u'c = u$, egyébként pedig $uqav \vdash urbv$.

Azt mondjuk, hogy M véges sok lépésben eljut a C konfigurációból a C' konfigurációba (jele $C \vdash^* C'$), ha van olyan $n \geq 0$ és $C_1, \ldots, C_n$ konfigurációsorozat, hogy $C_1 = C$, $C' = C_n$ és minden $1 \leq i < n$-re, $C_i \vdash C_{i+1}$.

Ha $q \in \{q_i, q_n\}$, akkor azt mondjuk, hogy az uqv konfiguráció egy megállási konfiguráció. $q = q_i$ esetében elfogadó, míg $q = q_n$ esetében elutasító konfigurációról beszélünk.

Az M által felismert nyelv (amit $L(M)$-mel jelölünk) azoknak az $u \in \Sigma^*$ szavaknak a halmaza, melyekre igaz, hogy $q_0u\sqcup \vdash^* xq_iy$ valamely $x, y \in \Gamma^*, y \neq \varepsilon$ szavakra. □

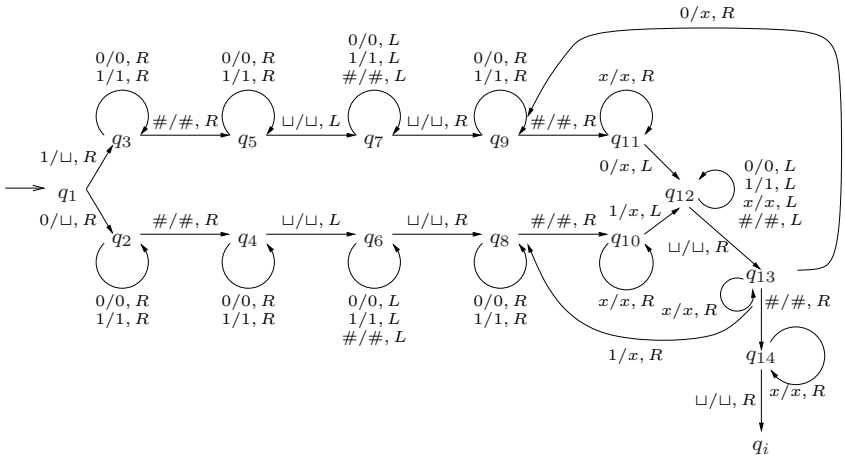
Példa Tekintsük az $L = \{u\#u \mid u \in \{0,1\}^+\}$ nyelvet. Az L felismerhető egy olyan Turing-géppel, mely a következő algoritmus szerint működik.

1. Végigolvasva az inputot ellenőrizzük le, hogy az pontosan egy #-t tartalmaz-e. Ha nem, akkor elutasítjuk a bemenetet. Figyelem, még az első lépésben meg kell jelölni az input első betűjét, különben a gép nem fogja tudni, mikor ért vissza a szó elejére. Esetünkben ez úgy történik, hogy átírjuk az első betűt ⊔-re, és egy állapotban megjegyezzük, hogy milyen szimbólumot töröltünk ki.

2. A szalagon oda-vissza haladva ellenőrizzük le, hogy a # jobb és bal oldalán, az egymásnak megfelelő pozíciókon ugyanazok a szimbólumok szerepelnek-e. Ez úgy történik, hogy először a # bal oldalán átírjuk a soron következő betűt x-re, a gép az állapotában megjegyzi, hogy milyen betűt írt át, és ilyen betűt fog keresni a # jobb oldalán. Ha talál ilyet, akkor átírja x-re, egyébként pedig elutasítja a bemenetet. Ezután a gép vissza megy a szó elejére, és kezdi újra a most vázolt műveletet.

3. Ha a # bal oldalán minden szimbólum átíródott x-re, akkor leellenőrizzük, hogy a # jobb oldalán van-e még feldolgozatlan betű (olyan ami még nincs átírva x-re). Ha igen, akkor elutasítjuk a bemenetet, egyébként pedig elfogadjuk.

Az alábbi ábrán az L-et felismerő Turing-gép δ átmenetfüggvénye látható (a gép állapothalmaza, bemenő- illetve szalagszimbólumai is leolvashatók az ábráról, a gép kezdőállapota a q_1 állapot). Az átmenetfüggvény egy gráffal van megadva, amit a következő módon kell értelmezni. Legyen q és p a gép két állapota, a, b szalagszimbólumok, D pedig egy $\{L, R\}$-beli irány. Akkor az ábrán egy q-ból p-be vezető él $a/b, D$ címkéje azt jelenti, hogy $\delta(q, a) = (p, b, R)$. A gép be nem rajzolt átmenetei (a δ definíció szerint totális függvény kell, hogy legyen) a q_n állapotba vezetnek.



A gép számítását a 01#01 szón a következő konfigurációátmenetekkel lehet leírni.

$q_101\#01 \vdash \sqcup q_21\#01 \vdash \sqcup 1q_2\#01 \vdash \sqcup 1\#q_401 \vdash^2 \sqcup 1\#01q_4\sqcup \vdash \sqcup 1\#0q_61 \vdash^4 q_6 \sqcup 1\#01 \vdash \sqcup q_81\#01 \vdash \sqcup 1q_8\#01 \vdash \sqcup 1\#q_{10}01 \vdash \sqcup 1q_{12}\#x1 \vdash^2 q_{12} \sqcup 1\#x1 \vdash \sqcup q_{13}1\#x1 \vdash \sqcup xq_9\#x1 \vdash^* \sqcup x\#xx \sqcup q_i \sqcup .$

Tehát a gép, helyesen, elfogadja a 01#01 szót. □

Megjegyzés Szokás a Turing-gépet úgy definiálni, hogy a gép író-olvasó feje képes egy konfigurációátmenet során helyben maradni. Nem nehéz megmutatni, hogy egy ilyen Turing-gép szimulálható egy olyannal ami csak jobbra vagy balra léphet.

Most a Turing-gép segítségével definiáljuk a Turing-felismerhető illetve az eldönthető nyelveket. Igaz, hogy korábban már definiáltuk ezeket a nyelvosztályokat (a 13-ik oldalon), de nem kerülünk ellentmondásba azzal a definícióval, ha meggondoljuk, hogy a Church-Turing tézis értelmében minden algoritmus megadható Turing-géppel.

Definíció Egy $L \in \Sigma^*$ nyelv Turing-felismerhető, ha $L = L(M)$ valamely M Turing-gépre. Továbbá, egy $L \subseteq \Sigma^*$ nyelv eldönthető, ha létezik olyan M Turing-gép, mely minden bemeneten megállási konfigurációba jut és felismeri az L-et. A Turing-felismerhető nyelveket szokás rekurzívan felsorolhatónak, az eldönthető nyelveket pedig rekurzívnak is nevezni. □

Megjegyezzük, hogy a fenti példában látható Turing-gép nem csak felismeri, hanem el is dönti az L nyelvet.

Most definiáljuk a Turing-gépek futásának időigényét.

Definíció Tekintsünk egy $M = (Q, \Sigma, \Gamma, \delta, q_0, q_i, q_n)$ Turing-gépet és annak egy $u \in \Sigma^*$ bemenő szavát. Azt mondjuk, hogy M futási ideje (időigénye) az u szón n ($n \geq 0$), ha M a $q_0u\sqcup$ kezdőkonfigurációból n lépésben el tud jutni egy megállási konfigurációba. Ha nincs ilyen szám, akkor M futási ideje az u-n végtelen.

Legyen $f : N \to N$ egy függvény. Azt mondjuk, hogy M időigénye $f(n)$ (vagy, hogy M egy $f(n)$ időkorlátos gép), ha minden $u \in \Sigma^*$ input szóra, M időigénye az u szón legfeljebb $f(l(u))$.

□

Példa Tekintsük újra a fenti példában látható L nyelvet, és az őt eldöntő Turing-gépet. Könnyen látható, hogy a gép időigénye egy $u\#u$ n hosszú szón a következőképpen alakul.

- Az input leellenőrzése: $2n$ lépés.
- Egy betű pár összehasonlítása minimum $2(\lfloor \frac{n}{2} \rfloor + 1)$ ($\lfloor \frac{n}{2} \rfloor$ az $\frac{n}{2}$ egész részét jelöli) és maximum $2n$ lépés (attól függően, hogy melyik pozíciónál tartunk az összehasonlításban).

- Az előbb leírt ellenőrzést pedig $\lfloor \frac{n}{2} \rfloor$ esetben kell végrehajtani.

Tehát a gép számítása az $u\#u$ szón nagyságrendileg $2n + \lfloor \frac{n}{2} \rfloor 2n$, azaz $n^2 + 2n$ lépésből áll. Ha a gép a szalagján olyan u szót kap bemenetként, amely nem eleme L-nek, akkor a betű párok leellenőrzése $\lfloor \frac{n}{2} \rfloor$-nél kevesebbszer hajtódik végre. Mindezekből következik, hogy az L nyelv eldönthető egy $\mathcal{O}(n^2)$ időkorlátos Turing-géppel. □

3.2. Különböző Turing-gép változatok

Ebben a fejezetben megvizsgálunk néhányat a Turing-gép különböző változatai közül. Ezek a gépek bár általánosabban vannak definiálva, mit a mi Turing-gép modellünk, de valójában mind ekvivalensek vele.

3.2.1. Többszalagos Turing-gépek

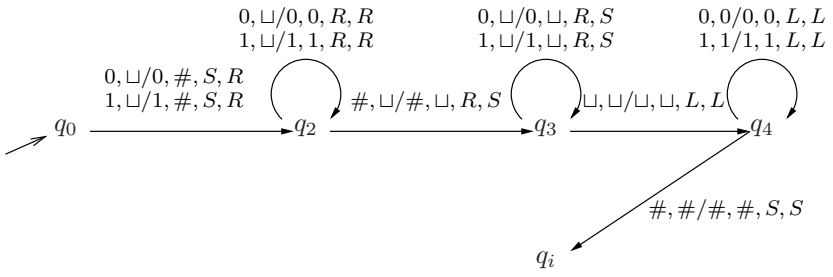
A többszalagos Turing-gépek, értelemszerűen, egynél több szalaggal rendelkeznek. Mindegyik szalaghoz tartozik egy író-olvasó fej, melyek egymástól függetlenül képesek mozogni jobbra és balra, valamint képesek arra is, hogy egy konfigurációátmenet során helyben maradjanak.

Definíció Legyen $k > 1$. Egy k-szalagos Turing-gép egy olyan $M = (Q, \Sigma, \Gamma, \delta, q_0, q_i, q_n)$ rendszer, ahol a komponensek a δ kivételével megegyeznek az egyszalagos Turing-gép komponenseivel, δ pedig a következőképpen adódik. $\delta : (Q - \{q_i, q_n\}) \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, S\}^k$. Itt az S szimbólum azt az esetet jelöli amikor az író-olvasó fej helyben marad. Legyenek q, p Q-beli állapotok, $a_1, \ldots, a_k, b_1, \ldots, b_k$ Γ-beli szimbólumok és $D_1, \ldots, D_k$ pedig $\{L, R, S\}$-beli irányok. Ha $\delta(q, a_1, \ldots, a_k) = (p, b_1, \ldots, b_k, D_1, \ldots, D_k)$, akkor a gép a q állapotból, ha a szalagjain rendre az $a_1, \ldots, a_k$ betűket olvassa, át tud menni a p állapotba, miközben az $a_1, \ldots, a_k$ betűket átírja a $b_1, \ldots, b_k$ betűkre és a szalagokon a fejeket a $D_1, \ldots, D_k$ irányokba mozgatja (ha valamely $1 \leq i \leq k$ esetén $D_i = S$, akkor az i-ik szalagon a fej helyben marad). A fejek, akár csak az egyszalagos esetben, nem mozdulnak balra, ha a szalag legelején állnak.

A többszalagos Turing-gép konfigurációi, a konfigurációátmenetek valamint a felismert illetve elöntött nyelv definíciója az egyszalagos eset értelemszerű általánosításai. A többszalagos Turing-gép modell időigényét is az egyszalagoshoz hasonlóan definiáljuk. □

A továbbiakban egy L nyelvet $f(n)$ időben eldönthetőnek nevezünk, ha eldönthető egy $f(n)$ időkorlátos (akár többszalagos) Turing-géppel.

Példa Tekintsük újra az $L = \{u\#u \mid u \in \{0,1\}^+\}$ nyelvet. Az alábbi kétszalagos gép szintén L-et dönti el. A gép δ átmenetfüggvénye a következő módon olvasható ki az ábrából. Legyen q és p a gép két állapota, a_1, a_2, b_1 és b_2 szalagszimbólumok, D_1 és D_2 pedig $\{L, R, S\}$-beli irányok. Ha az ábrán vezet



él q-ból p-be és az él címkéje $a_1, a_2/b_1, b_2, D_1, D_2$, akkor ez azt jelenti, hogy $\delta(q, a_1, a_2) = (p, b_1, b_2, D_1, D_2)$. A gép be nem rajzolt átmenetei itt is a q_n állapotba vezetnek.

A gép a következő módon működik. Először kiír egy # szimbólumot a második szalagjára, megjelölve ezzel a 2-ik szalagon a szó elejét. Ezután átmásolja a # baloldalán lévő szót a 2-ik szalagra. Majd elmegy az első szalagon a szó végére és mindkét szalagon az utolsó nem ⊔ szimbólumra áll. Végül mindkét szalagon balra lépkedve összehasonlítja az első szalagon a #-tól jobbra lévő szót a 2-ik szalagon lévő szóval. Ha a fejek a két szalagon egyszerre olvasnak #-t, akkor a gép elfogad, különben elutasít.

A gép időigényét egy n hosszú szón könnyű kiszámolni, az legfeljebb $\lfloor \frac{n}{2} \rfloor + n + 1$ lépés. Tehát L egy $\mathcal{O}(n)$ vagyis lineáris időben eldönthető nyelv. □

Bár a Turing-gép a több szalaggal egyszerűben képes felismerni egyes nyelveket, az általános kiszámítási ereje nem nő ezáltal, ahogy ezt az alábbi tétel bizonyításában is látni fogjuk. Két Turing-gépet ekvivalensnek nevezünk, ha ugyanazt a nyelvet ismerik fel.

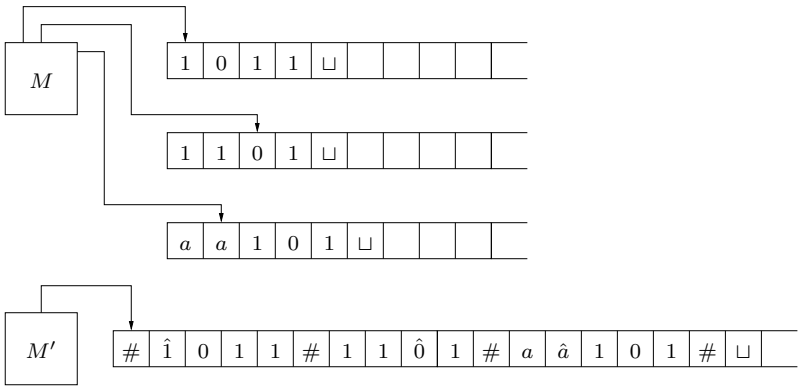
Tétel Minden k-szalagos, $f(n)$ időkorlátos Turing-géphez van vele ekvivalens egyszalagos, $\mathcal{O}(f(n)^2)$ időkorlátos Turing-gép.

Bizonyítás (Vázlat) Legyen M egy k-szalagos Turing-gép valamely $k \geq 2$-re. Megadunk egy M' egyszalagos Turing-gépet ami képes szimulálni M működését. A szimuláció ötlete az alábbi ábrán látható.

M' egymás után tárolja a szalagján M szalagjainak a tartalmát. A különböző szalagokat # jellel választja el egymástól. Azt, hogy M szalagjain a fejek mely szimbólumokra mutatnak M' úgy tartja számon, hogy a kérdéses szimbólumokat megjelöli egy ˆ jellel. Tehát M' szalagszimbólumai között, az M szalagszimbólumai mellett, ott van még a # szimbólum, valamint M szalagszimbólumainak egy ˆ-pal megjelölt változata.

A szimuláció lépései a következők. Legyen $w = a_1 \dots a_n$ M egy bemenő szava.

1. M' először előállítja a szalagján M kezdőkonfigurációját: $\#\hat{a}_1 a_2 \dots a_n \# \hat{\sqcup} \# \hat{\sqcup} \dots \#$.
2. M egy lépésének szimulálásához M' végigolvassa a szalagját az első #-tól



kezdve az utolsó (k+1-ik) #-ig. Eközben az állapotában eltárolja a ponttal megjelölt szimbólumok pont nélküli változatait (M' állapotai úgy vannak definiálva, hogy mindegyik képes tárolni M k darab szalagszimbólumát).

3. M' ezután az állapotában eltárolt adatok és M átmenetfüggvénye alapján végrehajtja a saját szalagján azokat a módosításokat, melyeket M végez a szalagjain.

4. Ha M valamelyik szalagján az utolsó nem ⊔ szimbólum olvasása után jobbra lép, akkor M' a megfelelő #-tól kezdve jobbra mozgatja egy pozícióval a szalagjának a tartalmát, és a felszabadult helyre beszúr egy $\hat{\sqcup}$ szimbólumot.

5. Ha M valamilyen (elfogadó vagy elutasító) megállási konfigurációba kerül, akkor M' is a megfelelő megállási konfigurációba lép. Egyébként M' folytatja M lépéseinek szimulálását a 2. ponttal.

Látható, hogy M' pontosan akkor lép elfogadó állapotba amikor M, tehát $L(M) = L(M')$. Továbbá nem nehéz megmutatni, hogy ha M $f(n)$ időkorlátos, akkor M' $\mathcal{O}(f(n)^2)$ időkorlátos. □

3.2.2. Turing-gép mindkét irányban végtelen szalaggal

A Turing-gép definiálható úgy is, hogy nem egy egy irányban végtelen szalagon, hanem egy mindkét irányban végtelenen dolgozik. Ekkor a Turing-gép korlátlanul léphet jobbra illetve balra a szalagján

Amint az várható, a mindkét irányban végtelen szalaggal működő Turing-gép kiszámítási ereje sem nagyobb mint az egy irányban végtelen szalagon működő Turing-gépé.

Tétel Minden két irányban végtelen szalagon működő Turing-géphez van vele ekvivalens Turing-gép.

A bizonyítás vázlata a következő. Legyen M egy két irányban végtelen szalagon működő Turing-gép. Elég M-hez megkonstruálni egy ekvivalens M' kétszalagos Turing-gépet, mert azt már tudjuk, hogy M'-höz megadható egy vele ekvivalens egyszalagos Turing-gép.

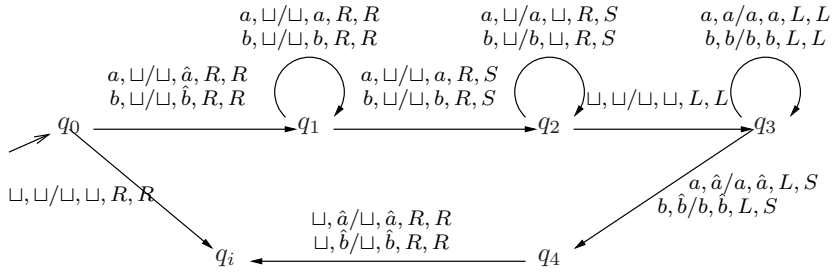
M' működésének az alapötlete a következő. M' két szalagjából rendre az első illetve a második szalag reprezentálja M szalagjának azon részét mely M kezdőkonfigurációjában a fejtől jobbra illetve balra szerepel. Amikor M a fej kezdőpozíciójához képest jobbra dolgozik, akkor M' az első szalagon lemásolja M működését. Amikor M a fej kezdőpozíciójától egyet balra lép, akkor M' a második szalagján kezd el dolgozni úgy, hogy M minden egyes olyan lépését mely a fej kezdőpozíciójától balra lévő szalagrészen dolgozik, egy a második szalagon történő ellentétes irányú lépéssel szimulálja. Ha a fej egyszer csak újra a kezdőpozíciójától jobbra lévő szalagrészen kezd el dolgozni, akkor M' újra az első szalagján kezdi el szimulálni M működését. Mindeközben M' állapotai tárolják M állapotait plusz még azt az információt, hogy M'-nek az első szalagon vagy a másodikon kell-e dolgoznia. □

3.2.3. Nemdeterminisztikus Turing-gépek

Természetesen a Turing-gépnek is létezik nemdeterminisztikus verziója. Ebben az alfejezetben ezekkel a gépekkel ismerkedünk meg. Egy M nemdeterminisztikus Turing-gép átmenetfüggvénye $\delta : (Q - \{q_i, q_n\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$ alakú. Tehát M minden konfigurációjából néhány (esetleg nulla) különböző konfigurációba mehet át. Ily módon M számítási sorozatai egy u szón egy fával reprezentálhatók. A fa csúcsa M kezdőkonfigurációja, a szögpontjai pedig M konfigurációi. A fa minden levele megfelel M egy számítási sorozatának az u-n. Végül M akkor fogadja el u-t, ha a fa valamelyik levele elfogadó konfiguráció. Nevezzük ezt a most leírt fát M nemdeterminisztikus számítási fájának az u-n. Az M által felismert nyelv a determinisztikus esethez hasonlóan definiálható, a gép által eldöntött nyelv pedig a következőképpen. Azt mondjuk, hogy egy nemdeterminisztikus Turing-gép eldönt egy $L \subseteq \Sigma^*$ nyelvet ha felismeri, és minden $u \in \Sigma$ szóra M számítási sorozatai végesek és elfogadási vagy elutasítási konfigurációba vezetnek. A nemdeterminisztikus Turing-gép definíciója értelemszerűen kiterjeszthető a többszalagos esetre is.

Példa Az alábbi Turing-gép az $L = \{ww \mid w \in \{a, b\}\}$ nyelvet dönti el (a be nem jelölt átmenetek a q_n állapotba vezetnek). A gép a q_0 állapotban elkezdi olvasni a bemenő szót, megjelöli a szó elejét, majd a q_1 állapotban tovább olvassa azt. Mindeközben átmásolja a szó már elolvasott részét a második szalagra. q_1-ben nemdeterminisztikusan dönthet úgy, hogy abbahagyja a szó másolását és átmegy a q_2 állapotba. Itt elmegy az első szalagon a szó végére, a második szalagon helyben marad. Ha a szó végére ért az első szalagon, akkor átmegy

q_3-ba, ahol elkezdi összehasonlítani az első és a második szalagon lévő szót. Ha egyformának találja őket, akkor elfogadja a bemenetet, egyébként pedig elutasítja. Ha a bemenő szó eleme a nyelvnek, akkor lesz a gépnek egy olyan számítási sorozata, mely pont a szó felénél hagyja abba a szó másolását, és ebben az esetben az első és a második szalagon ugyanaz a szó lesz. Tehát a gép elfogadja a bemenetet, tekintet nélkül arra, hogy a szón az összes többi számítási sorozat sikertelen lesz.



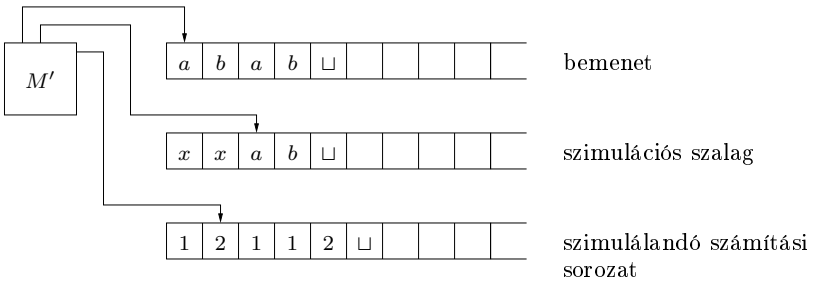
A nemdeterminisztikus Turing-gép időigényét a következő módon definiáljuk. Legyen $f : N \to N$ egy függvény és M egy nemdeterminisztikus Turing-gép. Azt mondjuk, hogy az M időigénye $f(n)$, ha egy n hosszú bemeneten nincsenek M-nek n-nél hosszabb számítási sorozatai. Az előbbi példában látható Turing-gép $\mathcal{O}(n)$ időben dönti el az L nyelvet.

Most belátjuk, hogy a nemdeterminizmus nem jelent plusz kiszámítási erőt a Turing-gépek esetében.

Tétel Miden nemdeterminisztikus Turing-gép ekvivalens egy determinisztikussal.

Bizonyítás Legyen M egy nemdeterminisztikus Turing-gép. Megadunk egy M' háromszalagos Turing-gépet ami ekvivalens M'-mel. Azt már korábban láttuk, hogy M'-höz megadható egy ekvivalens egyszalagos Turing-gép.

M' első szalagja tartalmazza a u bemenő szót. A második szalagon történik az M számítási sorozatainak a szimulációja. Ez a szalag tartalmazza M egy konkrét számítási sorozatának a lépésenkénti eredményét. A harmadik szalagon lévő szó alapján szimulálja M' az M egy számítási sorozatát. A szalagon a fej pozíciója mutatja azt, hogy melyik lépésnél tart M' a szimulációban. A harmadik szalagon tulajdonképpen az u-t elfogadó konfigurációk egy szélességi keresése történik a nemdeterminisztikus számítási fában. M' egy konfigurációja az alábbi ábrán látható.



Legyen b az M átmenetfüggvénye által megadott halmazok közül a legnagyobb elemszámúnak a számossága. Tegyük fel továbbá, hogy az átmenetfüggvény által megadott halmazokban az elemeknek van egy rögzített sorrendje. Így az u nemdeterminisztikus számítási fájának minden szögpontjához hozzárendelhetünk egy $\Sigma = \{1, 2, \ldots, b\}$ feletti szót. Persze nem feltétlenül minden Σ-feletti szó fog csúcsot reprezentálni a fában, de ez nem okoz majd gondot. A szimuláció a következőképpen történik.

1. Kezdetben az 1-es szalag tartalmazza a w bemenő szót, a 2-es és 3-as szalagok üresek.
2. M' az első szalag tartalmát rámásolja a második szalagra.
3. M' szimulálja a 2-ik szalagon M egy számítási sorozatát. Az, hogy melyiket kell szimulálnia a 3-ik szalagon lévő szótól függ. M' a szimuláció minden lépése előtt megnézi a 3-ik szalagján lévő szó következő betűjét, és e betű szerint (ami egy szám a Σ-ból) választ M átmenetfüggvényének a lehetőségei közül. Ha nincs megfelelő sorszámú választási lehetőség, vagy a 3-ik szalagon már nincs több betű, akkor a szimuláció véget ér, és ugrás a 4-ik lépésre.

   Ha a lépés szimulálása során M elutasító konfigurációba kerül, akkor szintén ugrás a 4-ik lépésre.

   Végül, ha elfogadó konfigurációba kerül M, akkor M' is elfogadó konfigurációba kerül és megáll.
4. M' kicseréli a 3-ik szalagján lévő szót az azt alfabetikusan követő szóra és a 2-ik pontra ugorva újrakezdi M működésének szimulálását ezen szó alapján.

Látható, hogy M' pontosan akkor kerül elfogadó illetve elutasító konfigurációba, amikor M, és ha M nem áll meg a bemenő szón, akkor M' sem áll meg. Következik tehát, hogy M és M' ekvivalensek.

Nem bizonyítjuk, de könnyen belátható, hogy ha M egy $f(n)$ időigényű Turing-gép, akkor M' $2^{\mathcal{O}(f(n))}$ időigényű. Speciálisan, ha M polinom időigényű, akkor M' exponenciális időigényű lesz.

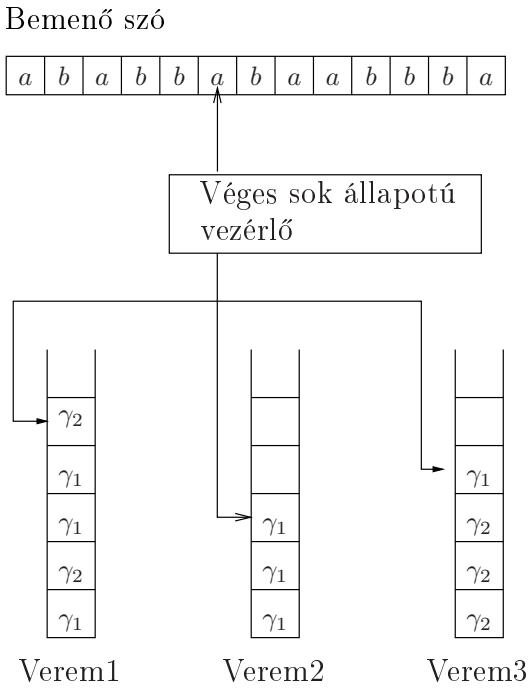
□

3.3. A Church-Turing tézis

Mint azt már korábban láttuk, a Church-Turing tézis azt mondja ki, hogy kiszámíthatóság különböző matematikai modelljei mind az effektíven kiszámítható függvények osztályát definiálják. A tézist persze formálisan nem lehet bizonyítani, hiszen az, hogy valami effektíven kiszámítható egy intuitív fogalom. A tézis bizonyítéka viszont többek között az, hogy minden algoritmus-modellről kiderült, hogy ekvivalens a Turing-géppel. Ebben a részben ismertetünk néhány, a Turing-géppel megegyező számítási erővel bíró eszközt.

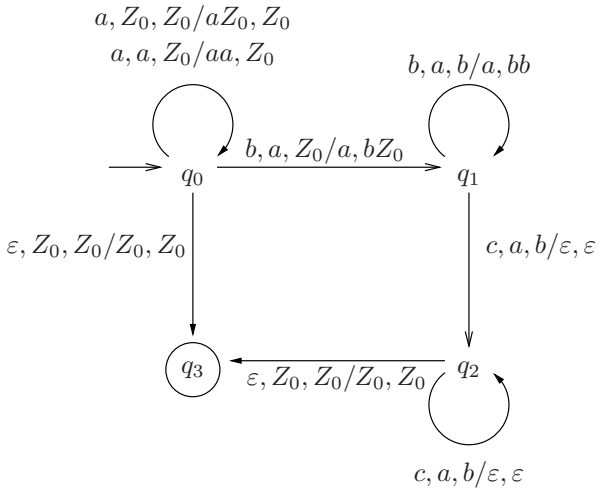
3.3.1. Többvermes automaták

A többvermes automata a veremautomata egyfajta általánosítása. Minden $k > 1$ számra, a k-vermes automata egy olyan végállapottal felismerő nemdeterminisztikus veremautomata, aminek, értelemszerűen, k darab verme van. Az alábbi ábrán egy 3-vermes automata vázlata látható:



A többvermes automata minden verme ugyanazt a veremábécét használja. Az automata konfiguráció-átmenete függ az éppen olvasott bemenő szimbólumtól (ez lehet ε is), a vermek legtetején álló szimbólumoktól, valamint az automata állapotától.

A többvermes automata akkor fogad el egy bemenő szót, ha van olyan számítási sorozata, mely során végigolvassa a szót és végállapotba kerül. Az alábbi ábrán szereplő kétvermes veremautomata például az $L = \{a^n b^n c^n \mid n \geq 0\}$ nyelvet ismeri fel.



Az talán nem annyira meglepő, hogy minden többvermes automatához van vele ekvivalens Turing-gép. Legyen ugyanis P egy k-vermes automata, valamely $k >$ 1-re. Akkor P szimulálható egy olyan k szalagos M Turing-géppel, ami a szalagjain tárolja P k darab vermének tartalmát, és a szalagokon lévő szavakat ugyanúgy manipulálja, ahogyan P a vermeit. Meglepőbb talán az alábbi tétel.

Tétel Minden Turing-felismerhető nyelv felismerhető egy kétvermes automatával.

Bizonyítás (Vázlat) Legyen L egy Turing-felismerhető nyelv és M egy Turing-gép, amire $L = L(M)$. Megadunk egy P kétvermes automatát, ami szimulálja M működését és szintén L-et ismeri fel. A szimuláció alapötlete a következő. P az első vermében tárolja azt a szót, ami M működése során az író-olvasó fejtől balra van, a második veremben pedig azt a szót ami fej pozíciójánál kezdődik. A szimuláció lépései a következők.

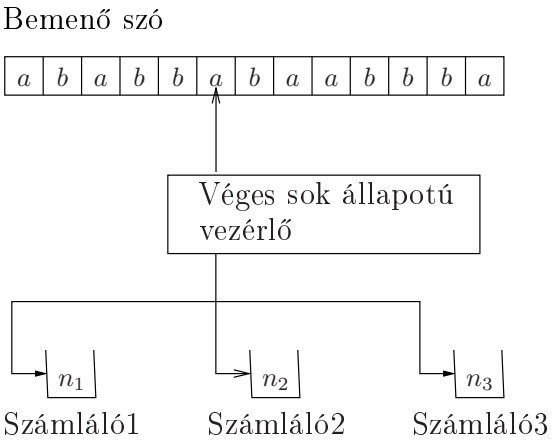
- P kezdetben csak a verem kezdőszimbólumokat tartalmazza mindkét vermében. A továbbiakban azt mondjuk, hogy egy verem üres, ha csak a verem kezdőszimbólumot tartalmazza.
- Tegyük fel, hogy a w szó van M bemenetén. P bemásolja w-t az első verembe.
- P betűnként átmásolja az első verem tartalmát a második verembe. Ha végez, akkor a konfigurációja M kezdőkonfigurációjának felel meg a w-n.

- P a következő módon szimulálja M egy átmenetét.
  - P ismeri M aktuális állapotát, legyen ez q, mert a saját állapotában tárolja M állapotait.
  - P tudja, hogy M milyen szimbólumot olvas éppen, hisz ez a szimbólum van a második verem tetején. Legyen ez a szimbólum X.
  - Ha M átírja X-et Y-ra és jobbra lép, akkor P beírja Y-t az első verembe X pedig kikerül a másodikból (ha a második verem üres volt, akkor nem változik, továbbá ha $Y = \sqcup$ és az első verem tartalma üres volt, akkor az első verem továbbra is üres marad).
  - Ha M átírja X-et Y-ra és balra lép, akkor P kiveszi az első verem legfelső elemét, legyen ez Z, és beírja ZY-t az második verembe (ha az első verem üres volt, akkor P csak Y-t ír a második verem tetejére).
  - Ha M új állapota q_i, akkor P végállapotba lép, egyébként pedig elkezdi szimulálni M egy újabb lépését a 4-ik pontra ugorva.

Nem nehéz belátni, hogy az így leírt veremautomata szintén L-et ismeri fel. □

3.3.2. Számlálós gépek

A többvermes automatákhoz nagyon hasonló felépítésűek a számlálós gépek, csak itt a gépeknek a vermek helyett számlálói vannak, melyek értéke egy természetes szám lehet. A gép egy lépésében beolvassa a bemenet egy betűjét (ez lehet ε is), állapotot vált és (egymástól függetlenül) növeli vagy csökkenti a számlálók értékét eggyel (ha egy számláló értéke 0 volt, akkor annak az értékét nem tudja csökkenteni). További fontos dolog még, hogy a gép nem tudja megmondani, hogy mekkora érték szerepel az egyes számlálókban, csak a 0 és nem 0 értékeket tudja megkülönböztetni egymástól. Az alábbi ábrán egy 3-számlálós automata vázlata látható:



A gép csak akkor fogad el egy bemenő szót, ha végig tudja olvasni azt és végállapotba jut. Tekintsük például az

$$L = \{u\#v\#w \mid u, v, w \in \{0,1\}^*, w \text{ az } u \text{ és } v \text{ bináris számok összege}\}.$$

Ezt a nyelvet az alábbi P (vázlatosan megadott) 4-számlálós automata ismeri fel. P a következőket teszi:

q_0-ban: Beírja az u számot (decimálisan) az 1-es számlálóba az alábbi módon:

- Ha 0-t olvas, akkor az 1-es számlálóban lévő számot megszorozza 2-vel a következőképpen. Először kimásolja az 1-ben lévő számot 4-be úgy, hogy egy ciklusban eggyel csökkenti az 1-es és növeli a 4-es értékét addíg, amig az 1-es értéke 0 nem lesz. Ezután egy ciklusban 4-et 0-ra csökkenti és minden lépésben 2-vel (1 plusz 1-gyel) növeli az 1-es számláló értékét).
- Ha 1-t olvas, akkor az 1-es számlálóban lévő számot a fenti módon megszorozza 2-vel és az eredményhez még hozzáad 1-et.
- Ha #-t olvas, akkor átmegy q_2-be.

q_2-ben: Beírja a v számot (decimálisan) a 2-es számlálóba az előbb vázolt módon. Ha végzett, akkor átmegy a q_3 állapotba.

q_3-ban: Beírja a w számot (decimálisan) a 3-es számlálóba és átmegy a q_4 állapotba.

q_4-ben: Átmásolja az 1-es számlálóban lévő számot a 2-esbe. Ha végez, akkor átmegy a q_5 állapotba.

q_5-ben: Egy ciklusban csökkenti a 2-es és a 3-as számláló értékét 1-el. Ha a két számláló értéke egyszerre lesz 0, akkor végállapotba megy. □

A számlálós gép számítási ereje is megegyezik a Turing-gépével. Azt, hogy a számlálós gépek nem tudnak felismerni több nyelvet, mint a Turing-gépek következik abból, hogy a számlálós gép tekinthető egyfajta többvermes automatának is, ami pedig ekvivalens a Turing-géppel. Nem nehéz meggondolni ugyanis azt, hogy egy számlálós gép ekvivalens egy olyan többvermes géppel, aminek ugyanannyi verme van, mint a számlálós gépnek, és ami a vermében a verem kezdőszimbólumon kívül csak egy veremszimbólumot használ. Ebből a szimbólumból egy veremben mindig annyi szerepel, mint amennyi a számlálós gép megfelelő számlálójában lévő szám.

Most belátjuk, hogy minden Turing-gép szimulálható egy számlálós géppel.

Tétel Minden Turing-géphez megadható egy vele ekvivalens három-számlálós gép.

Bizonyítás Korábbi tételünk szerint, nevezetesen, hogy minden Turing-gép ekvivalens egy kétvermes automatával, elegendő megmutatni azt, hogy egy kétvermes automata szimulálható egy három-számlálós géppel. Legyen tehát P egy kétvermes automata egy Γ veremábécével. Megadunk egy P-vel ekvivalens P' három-számlálós gépet. Az elv az, hogy P vermeinek a tartalmát egy számmá konvertáljuk és ezeket a számokat tárolja majd P' egy-egy számlálójában. P' harmadik számlálóját segéd-számlálóként használjuk. Először is rendeljünk Γ minden eleméhez egy sorszámot, azaz ha Γ $r-1$ elemű valamely $r > 1$ számra, akkor azonosítsuk Γ-t az $\{1, 2, \ldots, r-1\}$ halmazzal. Tegyük fel, hogy P egy verme a $\gamma = X_1 X_2 \ldots X_n \in \Gamma^*$ szót tartalmazza. Akkor γ-hoz hozzárendelhetünk egy l számot úgy, hogy l-ből egyértelműen meg lehessen határozni γ-t. Legyen $l = X_n r^{n-1} + X_{n-1} r^{n-2} + \ldots + X_2 r + X_1$. Ekkor l-ből visszakaphatjuk az $X_1, X_2, \ldots, X_n$ számokat úgy, hogy először osztjuk l-et r-el és a maradék lesz X_1, utána osztjuk $l - X_1$-et r^2-el és a maradék lesz X_2 és így tovább. Ezt addig csináljuk, amíg meg nem kapjuk X_n értékét is.

Mivel P' egyszerre csak eggyel tudja növelni vagy csökkenteni a számlálóinak az értékét, feltesszük, hogy P'-nek van r darab olyan állapota melyeket segítségével képes (r lépésben) egy számláló értékét r-rel növelni vagy csökkenteni. Tegyük fel, hogy P' egyik számlálójában, mondjuk az elsőben, egy $i > 0$ szám van. Akkor P képes kiszámolni az $i \cdot r$ értéket a következő módon. Egy ciklusban csökkenti az első számláló értékét eggyel amíg az 0 nem lesz, és minden lépésben növeli a harmadik számláló értékét r-el. Amikor P' végez a művelettel, a harmadik számlálóban $i \cdot r$ lesz.

Másrészt, P' képes az $\frac{i}{r}$ érték egész részét ($\lfloor \frac{i}{r} \rfloor$-t) valamint az $X = i - \lfloor \frac{i}{r} \rfloor r$ értéket (tehát az egész osztás maradékát) is kiszámolni. Egy ciklusban csökkenti az első számláló értékét r-rel addig, amíg az 0 nem lesz, és minden lépésben növeli a harmadik számláló értékét 1-el. Amikor P' végez a művelettel, a harmadik számlálóban $\lfloor \frac{i}{r} \rfloor$ lesz. Továbbá, a ciklus utolsó lépésében, amikor az első számláló értéke eléri a 0-t, P' az állapotából ki tudja olvasni azt az értéket, amivel még csökkenteni kellene a számláló értékét ahhoz, hogy az r kivonása befejezett legyen. Könnyen ellenőrizhető, hogy ez az érték pontosan az X.

Az alábbiakban megmutatjuk, hogy hogyan kell P'-nek módosítania a számlálói tartalmát ahhoz, hogy a számlálók folyamatosan P vermeit reprezentálják. Csak az első veremtartalmat reprezentáló első számlálóval foglalkozunk, a második tartalmát az elsővel analóg módon kell P'-nek módosítania.

Ahhoz, hogy P' szimulálni tudja P egy konfiguráció átmenetét, ismernie kell azokat a szimbólumokat, melyek P vermeinek a tetején vannak. Nem nehéz belátni azonban, hogy az $X = i - \lfloor \frac{i}{r} \rfloor r$ szám (i az első számlálóban lévő szám) éppen a P első vermének a tetején álló szimbólum. Fentebb pedig láttuk, hogy P' ezt az értéket ki tudja számolni és azt sem nehéz meggondolni, hogy P', miután kiszámolta X-et, vissza tudja állítani i értékét az első számlálóban.

Nem nehéz belátni, hogy P egy lépése során a vermen elvégzett művelet felbontható olyan műveletek sorozatára, melyek mindegyike az alábbiak közül való: a

verem tetején lévő betű kivétele a veremből; a verem tetején lévő betű átírása egy másikra; valamint egy betű írása a verem tetejére. Az a művelet például, ami kiveszi a verem tetejéről az X szimbólumot és kicseréli azt egy egynél hosszabb szóra felbontható azon műveletek sorozatára, melyek rendre kicserélik az X-et egy betűre és utána betűnként kiírnak egy szót a verem tetejére.

Feltehetjük tehát, hogy P minden lépésekor a következők egyikét teszi.

- Leveszi a verem tetejéről az X szimbólumot és nem ír a helyére semmit. Ebben az esetben P'-nek a fentebb látott módon ki kell cserélnie az első számláló tartalmát i-ről $\lfloor \frac{i}{r} \rfloor$-re.
- Leveszi a verem tetejéről X-et és a helyére irja az Y-t. Ekkor ha $Y > X$ akkor P' növeli a számláló értékét $Y - X$-el, egyébként pedig csökkenti azt $X - Y$-al.
- Kiírja a verem tetejére az X szimbólumot. Ekkor P'-nek növelnie kell a korábban látott módon az első számláló tartalmát i-ről $i \cdot r + X$-re. □

3.3.3. Közvetlen hozzáférésű gépek

A közvetlen hozzáférésű gép vagy röviden RAM (az angol Random Access Machine kifejezésből) olyan véges utasításkészlettel rendelkező kiszámítási eszköz, ami egy regiszterekből álló tömbön dolgozik. A gép regisztereit rendre ellátjuk a $0, 1, 2, \ldots$ sorszámokkal és a j-ik regiszter értékét r_j-vel jelöljük. Egy regiszter értéke egy tetszőleges egész szám lehet. A RAM számára potenciálisan végtelen sok regiszter áll rendelkezésre, de ezekből mindig csak véges sok van használva. Ez azt is jelenti, hogy a gép számítása során, mivel kezdetben a regiszterek értéke 0, a gépnek mindig csak véges sok olyan regisztere lehet, melynek az értéke nem 0.

A gép a bemenetét az úgynevezett bemeneti regiszterekből kapja, melyek egy $I = (i_1, \ldots, i_n)$ véges hosszúságú regisztertömbben vannak. A gépnek van egy kitüntetett belső regisztere, a 0-ik, amit akkumulátornak nevezünk. A gép csak ezen a regiszteren tud aritmetikai műveleteket végezni. A többi regiszter értékét csak úgy tudja módosítani, hogy az akkumulátor értékét betölti ezekbe a regiszterekbe. A regiszterek módosítását a gép programja végzi. RAM-programon egy $P = (\pi_1, \pi_2, \ldots, \pi_m)$ utasítássorozatot értünk, ahol minden $1 \leq i \leq m$-re a π_i az alább felsorolt utasítások egyike lehet.

Utasítás	**Operandusz**	**Jelentés**
READ	x	$r_0 := x$
STORE	x	$x := r_0$
LOAD	x	$r_0 := x$
ADD	x	$r_0 := r_0 + x$
SUB	x	$r_0 := r_0 - x$
HALF		$r_0 := \lfloor \frac{r_0}{2} \rfloor$

Utasítás	**Operandusz**	**Jelentés**
JUMP	j	$\kappa := j$
JPOS	j	ha $r_0 > 0$, akkor $\kappa := j$
JNEG	j	ha $r_0 < 0$, akkor $\kappa := j$
JZERO	j	ha $r_0 = 0$, akkor $\kappa := j$
HALT		$\kappa = 0$

Az x operandusz egy j, $\uparrow j$ illetve $=j$ alakú kifejezés lehet, ahol j egy egész szám (kivéve a READ és STORE utasításokat, melyek esetében x nem lehet $=j$ alakú). Ha x egy j alakú kifejezés, akkor x értéke r_j, vagyis a j-ik regiszter értéke. Ha x egy $\uparrow j$ alakú kifejezés, akkor az értéke r_{r_j}, vagyis azon regiszter értéke, amit a j-ik regiszterbe lévő szám címez meg. Végül, ha x egy $=j$ alakú kifejezés, akkor x értéke maga a j szám lesz. A HALF az akkumulátorba írja az $\lfloor \frac{r_0}{2} \rfloor$ értéket, tehát megfelezi az akkumulátorban lévő számot.

Az utasítások, kivéve a vezérlésmódosító utasításokat (JUMP, JPOS, JNEG, JZERO), 1-el növelik a κ utasításszámláló értékét, ami a program indulásakor 1. A JUMP j egy feltétel nélküli ugrás a j-ik utasításra. A JPOS j, JNEG j és JZERO j pedig egy-egy, az akkumulátor értékétől függő, feltételes ugrás a j-ik utasításra.

Egy RAM program az első utasítás végrehajtásával kezdődik, és akkor van vége, ha végrehajtódik egy HALT utasítás vagy a program egy szabálytalan utasítást próbál végrehajtani (olyan utasítást például ami negatív sorszámú regiszterre hivatkozik). A program megállása után a kimenet értéke az akkumulátorban van.

Példa Az alábbi példa egy $I = (i_1, i_2)$ bemenet esetén kiszámolja az $\lfloor \frac{i_1}{i_2} \rfloor$ értéket, ha $i_1 \geq 0, i_2 > 0$, és -1-et ad vissza egyébként.

```
1   READ 2          11  LOAD 3
2   JNEG 16         12  ADD 1
3   JZERO 16        13  STORE 3
4   STORE 2         14  LOAD 1
5   READ 1          15  JUMP 8
6   JNEG 16         16  LOAD =0
7   STORE 1         17  SUB 1
8   SUB 2           18  HALT
9   JNEG 19         19  LOAD 3
10  STORE 1         20  HALT
```

□

Egy P RAM program kezdőkonfigurációja az az állapot, amikor κ értéke 1 és minden (belső) regiszter értéke 0. A P megállási konfigurációja pedig az az állapot, amikor a $\kappa = 0$. Legyen most D az egész számok véges sorozatainak a halmaza és ϕ a D-ből az egész számok halmazába képező függvény. Azt mondjuk, hogy P kiszámítja a ϕ-t, ha minden $I \in D$ számsorozatra, P az I

bemenettel indítva el tud jutni a kezdőkonfigurációból egy megállási konfigurációba úgy, hogy az akkumulátor értéke $\phi(I)$ lesz.

Végezetül kimondjuk ennek a résznek a legfőbb eredményét.

Tétel A közvetlen hozzáférésű gépek és a Turing-gépek számítási ereje megegyezik.

A tételt nem bizonyítjuk, csak megjegyezzük a következőket. Legyen $\phi : D \to Z$ (Z az egész számok halmaza), egy RAM programmal kiszámítható függvény. Akkor megadható olyan M Turing-gép amely minden $I \in D$ számsorozatra, az I egy reprezentációjával a szalagján indítva megáll úgy, hogy a szalagon a $\phi(I)$ szám (egy reprezentációja) lesz. Másrészt, legyen $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ $(n > 0)$ egy ábécé és $L \subseteq \Sigma^*$ egy Turing-felismerhető nyelv. Akkor megadható egy olyan P RAM program, mely azt a ϕ függvényt számolja ki, amit a következőképpen definiálunk. Tetszőleges $I = (i_1, \ldots, i_m, 0)$ bemenetre $\phi(I) = 1$, ha $\sigma_{i_1} \ldots \sigma_{i_m} \in L$, és $\phi(I) = 0$ egyébként. (Az I bemeneti tömb végére azért kell odaírni a 0-t, hogy a RAM program képes legyen felismerni a bemenő szó végét.)

3.4. Eldönthetetlen problémák

Ebben a fejezetben megmutatjuk, hogy bár a Turing-gép a lehető legáltalánosabb algoritmus modell, mégis vannak olyan problémák, melyek nem számíthatók ki Turing-géppel.

Először felidézzük azt, hogy egy $L \subseteq \Sigma^*$ nyelvet Turing-felismerhetőnek (vagy rekurzívan felsorolhatónak) nevezünk, ha van olyan M Turing-gép ami az összes L-beli szóra q_i-ben áll meg, a többi szóra pedig q_n-ben áll meg vagy esetleg nem áll meg. A rekurzívan felsorolható nyelvek osztályát RE-vel jelöljük. Továbbá, L-et eldönthetőnek (vagy rekurzívnak) nevezzük, ha M minden bemeneten meg is áll. A rekurzív nyelvek osztályát R-rel jelöljük. Világos, hogy fennáll az $R \subseteq RE$ tartalmazás. A célunk az, hogy megmutassuk az R valódi részhalmaza az RE-nek, azaz van olyan nyelv (probléma) ami Turing-felismerhető, de nem eldönthető, azaz nincs olyan algoritmus, ami a problémát eldöntené.

A fenti célnak megfelelő nyelv a következő lesz: azon (M, w) párok halmaza (egy megfelelő bináris szóban elkódolva), ahol M egy $\{0,1\}$ bemenő ábécé feletti Turing-gép, w pedig egy $\{0,1\}$-feletti szó úgy, hogy $w \in L(M)$, azaz M elfogadja w-t. Jelöljük ezt a nyelvet L_{u}-val.

Ahhoz, hogy célunkat elérjük, először megmutatjuk, hogy a $\{0,1\}$ ábécé feletti Turing-gépek egyértelműen elkódolhatóak egy bináris szóval. Ez után pedig megmutatjuk, hogy az $L_{\text{átló}}$ nyelv, mely azon $\{0,1\}$-feletti Turing-gépek bináris kódjait tartalmazza, melyek nem fogadják el önmaguk kódját, mint bemenő szót, egy olyan nyelv, ami nem rekurzívan felsorolható.

3.4.1. Turing-gépek kódolása

Először megjegyezzük, hogy a $\{0,1\}$-feletti szavak felsorolhatóak (vagyis megszámlálhatóak). Valóban, tekintsük azt a felsorolást, amelyben a szavak a hosszuk szerint követik egymást, és két egyforma hosszú szó közül pedig az van előbb, amelyik az alfabetikus rendezés szerint megelőzi a másikat. Ily módon a $\{0,1\}^*$ halmaz elemeinek egy felsorolása a következőképpen alakul: $w_1 = \varepsilon$, $w_2 = 0$, $w_3 = 1$, $w_4 = 00$, $w_5 = 01$, és így tovább. Ezután a $\{0,1\}^*$ halmaz i-ik eleme alatt a w_i szót értjük.

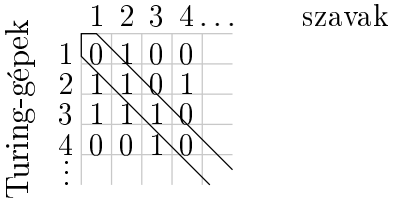
Legyen a továbbiakban $M = (Q, \{0,1\}, \Gamma, \delta, q_0, q_i, q_n)$ egy Turing-gép. Akkor van olyan $k > 0$ szám, hogy Q-t felírhatjuk $Q = \{q_0, q_1, \ldots, q_k\}$ alakban, ahol $q_{k-1} = q_i$ és $q_k = q_n$. Továbbá, van olyan $m > 0$ szám, hogy Γ felírható $\Gamma = \{X_1, X_2, \ldots, X_m\}$ alakban, ahol $X_1 = 0$, $X_2 = 1$, $X_3 = \sqcup$ és $X_4, \ldots, X_m$ az M további szalagszimbólumai. Nevezzük végül az L, R és S szimbólumokat, azaz a Turing-gép átmenetfüggvényében szereplő irányokat, rendre a D_1-es, D_2-es és D_3-as iránynak. Ezek után M egy $\delta(q_i, X_j) = (q_r, X_s, D_t)$ átmenete ($0 \leq i, r \leq k$, $1 \leq j, s \leq m$ és $1 \leq t \leq 3$) elkódolható a $0^{i+1}10^j10^{r+1}10^s10^t$ szóban. Valóban, ha az első és harmadik 0-s blokkban szereplő 0-k számából kivonunk egyet, akkor megkapjuk az átmenetben szereplő állapotok indexeit. A második, negyedik és ötödik 0-s blokkban szereplő 0-k száma pedig rendre az átmenetben szereplő szalagszimbólumok és az irány indexeit adják. Továbbá, mivel miden 0-s blokk hossza legalább 1, az átmenetet kódoló szóban nem szerepel az 11 részszó. Tehát az M összes átmenetét kódoló szavakat összefűzhetjük egy olyan szóvá, melyben az átmeneteket az 11 részszó választja el egymástól. Az így kapott szó pedig magát M-et kódolja.

A továbbiakban minden $i \geq$ 1-re M_i-vel jelöljük azt a Turing-gépet, amit a w_i bináris szó kódol (emlékeztetőül, w_i az i-ik elem a $\{0,1\}^*$ elemeit felsoroló listában). Megegyezünk továbbá abban, hogy ha valamely i-re w_i nem a fent leírt kódolása egy Turing-gépnek, akkor M_i-t azon Turing-gépnek tekintjük, ami minden inputon azonnal a q_n állapotba megy, vagyis $L(M_i) = \varnothing$. A későbbiekben szükségünk lesz arra, hogy elkódoljunk egy (M, w) Turing-gép és bemenő szó párost egy $\{0,1\}$-feletti szóban. Ehhez felhasználhatjuk azt, hogy a Turing-gépek fenti kódolása nem tartalmazhat három 1-est egymás mellett. Tehát az (M, w) párt úgy kódoljuk el, hogy M kódja után írjuk az 111 szót, ezután pedig w-t. A későbbiekben egy (M_i, w) párost kódoló w_i111w szó helyett, a könnyebb olvashatóság kedvéért, gyakran írjuk majd az $\langle M, w\rangle$ kifejezést.

3.4.2. Egy nem rekurzívan felsorolható nyelv

Felhasználva a Turing-gépek előbb látott elkódolását, most már pontosan is definiálni tudjuk, hogy mit értünk a fent említett L_{u} és $L_{\text{átló}}$ nyelvek alatt: $L_{\text{u}} = \{w_i111w_j \mid i, j \geq 1, w_j \in L(M_i)\}$ és $L_{\text{átló}} = \{w_i \mid i \geq 1, w_i \notin L(M_i)\}$. Erről az utóbbi nyelvről mutatjuk meg, hogy nem rekurzívan felsorolható.

Ezt a nyelvet azért nevezzük $L_{\text{átló}}$-nak, mert a karakterisztikus függvénye megkapható az alábbi módon. Tekintsük egy végtelen táblázatot, melynek oszlopai és sorai rendre az $1, 2, 3, \ldots$ számokkal vannak címkézve, az elemei pedig a 0 és 1 lehetnek az alábbiak szerint. Minden $i, j \geq 1$-re, az i-ik sorban a j-ik elem értéke pontosan akkor 1, ha $w_j \in L(M_i)$ és 0 egyébként. Tegyük fel, hogy az így kitöltött táblázatunk az alábbi módon néz ki.



Ebben a táblázatban például a második sor második eleme 1, ami azt jelenti, hogy M_2 elfogadja a w_2 szót (ez persze a valóságban nem igaz, hisz w_2 nem kódol legális Turing-gépet, így M_i nem is ismerhet fel semmilyen szót, de az összefüggések szemléltetéséhez feltesszük, hogy a fenti táblázat helyes).

Minden $i \geq 1$-re, a fenti táblázat i-ik sora tekinthető úgy, mint az $L(M_i)$ nyelv karakterisztikus függvénye. Mégpedig azért, mert minden $j \geq 1$-re, a w_j szó pontosan akkor eleme $L(M_i)$-nek, ha az i-ik sor j-ik eleme 1. Ebben az értelemben a táblázat átlójában szereplő bitsorozat komplementere pedig nem más, mint az $L_{\text{átló}}$ karakterisztikus függvénye. Ezért nyilvánvaló, hogy minden $i \geq 1$-re az $L_{\text{átló}}$ karakterisztikus függvénye különbözik $L(M_i)$ karakterisztikus függvényétől. Ezt használjuk fel az alábbi tétel bizonyításában.

Tétel Az $L_{\text{átló}}$ nem rekurzívan felsorolható.

Bizonyítás Indirekt bizonyítást adunk. Tegyük fel, hogy $L_{\text{átló}}$ rekurzívan felsorolható. Megmutatjuk, hogy ez ellentmondáshoz vezet. Ha $L_{\text{átló}}$ felismerhető, akkor van olyan M $\{0,1\}$-feletti Turing-gép, amire $L_{\text{átló}} = L(M)$. Mivel M egy $\{0,1\}$-feletti Turing-gép, van olyan $i \geq 1$, hogy M ekvivalens M_i-vel, vagyis $L_{\text{átló}} = L(M_i)$. Vajon eleme-e w_i az $L(M_i)$ nyelvnek? M és M_i ekvivalenciája miatt $w_i \in L(M_i)$ pontosan akkor teljesül, ha $w_i \in L_{\text{átló}}$ teljesül. Ez utóbbi viszont, $L_{\text{átló}}$ definíciója szerint, pontosan akkor igaz, ha $w_i \notin L(M_i)$. Tehát ellentmondást kaptunk, ami azt bizonyítja, hogy $L_{\text{átló}}$ nem rekurzívan felsorolható. □

3.4.3. Egy rekurzívan felsorolható, de nem eldönthető nyelv

Az előző alfejezet eredményét felhasználva szeretnénk megmutatni, hogy az L_{u} egy olyan rekurzívan felsorolható nyelv ami nem rekurzív. Ehhez először definiálnunk kell, hogy mit értünk egy nyelv komplementerén.

Legyen L egy Σ-feletti tetszőleges nyelv. Az L nyelv komplementerét $\bar{L}$-el jelöljük és az alábbi módon definiáljuk: $\bar{L} = \{w \mid w \in \Sigma^*, w \notin L\}$.

Most bebizonyítunk két, a nyelvek és komplementereik kapcsolatára vonatkozó állítást.

Tétel Ha L egy rekurzív nyelv, akkor a komplementere is rekurzív.

Bizonyítás Ha L rekurzív, akkor van olyan M Turing-gép, ami eldönti L-et. Tehát M olyan, hogy a nyelv elemein indítva q_i-ben áll meg, az összes többi szón pedig q_n-ben áll meg, vagyis minden szón megáll. Legyen M' az a Turing-gép, amit úgy kapunk M-ből, hogy M összes olyan átmenetét, ami q_i-be megy q_n-be irányítjuk, a q_n-be menő átmeneteket pedig q_i-be. Nem nehéz belátni, hogy az így definiált M' az $\bar{L}$ nyelvet dönti el. □

Most azt mutatjuk meg, hogy ha egy L nyelv és az ő komplementere is rekurzívan felsorolható, akkor L rekurzív is.

Tétel Legyen L egy nyelv. Ha $L, \bar{L} \in RE$, akkor $L \in R$.

Bizonyítás Ha L és $\bar{L}$ is rekurzívan felsorolható, akkor vanak olyan M_1 és M_2 Turing-gépek, melyek rendre L-et és $\bar{L}$-et ismerik fel. M_1 és M_2 felhasználásával megkonstruálunk egy olyan M turing-gépet, ami eldönti L-et.

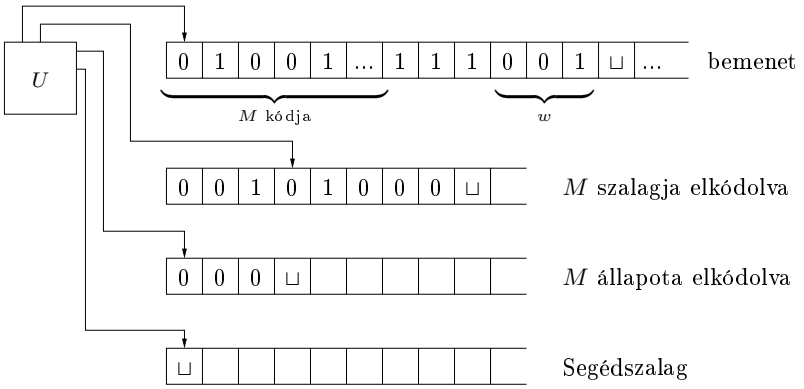
M egy olyan kétszalagos gép, ami egy w bemeneten az alábbiakat teszi. Első lépésként a második szalagra másolja w-t. Ezután egy ciklusban szimulálja az M_1 egy lépését az első szalagon és az M_2 egy lépését a másodikon. A ciklus addig fut, amíg M_1 és M_2 közül valamelyik elfogadó állapotba nem lép. Mivel w vagy az $L(M_1)$-nek vagy az $L(M_2)$-nek az eleme, véges számú szimulációs lépés után, vagy M_1 vagy M_2 elfogadja w-t, tehát valamelyik előbb-utóbb biztosan q_i-be lép. Ha M_1 lép q_i-be, akkor M elfogadja a bemenetet, ha M_2, akkor pedig elutasítja azt. Könnyen látható, hogy M is L-et ismeri fel ráadásul minden bemeneten megáll, tehát el is dönti L-et. Következésképpen $L \in R$. □

Most már készen állunk arra, hogy megmutassuk azt, hogy L_{u} rekurzívan felsorolható, de nem rekurzív. Először azt mutatjuk meg, hogy rekurzívan felsorolható.

Tétel $L_{\mathrm{u}} \in RE$.

Bizonyítás Megadunk egy olyan U Turing-gépet, ami L_{u}-t ismeri fel. Ezt a Turing-gépet nevezzük az Univerzális Turing-gépnek, mert mint látni fogjuk U szimulálja a bemenetén kapott $\langle M, w\rangle$ szó által kódolt M Turing-gép működését a w szón.

U-nak négy szalagja van. Először leellenőrzi, hogy a bemenete egy $\langle M, w\rangle$ alakú szó-e. Ha igen, akkor elkezdi szimulálni M működését. U a második szalagján tárolja az M szalagját, mégpedig ugyanolyan a kódolással, mint amilyen az M elkódolásánál is használatos. Vagyis például az M X_j szalagszimbóluma a 0^j szóval van reprezentálva és a szalagszimbólumok pedig egy 1-es szimbólummal vannak elválasztva. A harmadik szalagon tárolja U az M aktuális állapotát a szokásos módon kódolva, a q_i állapot például a 0^{i+1} szóval van reprezentálva. U felépítése az alábbi ábrán látható.



U működése egy v bemeneten az alábbi módon foglalható össze.

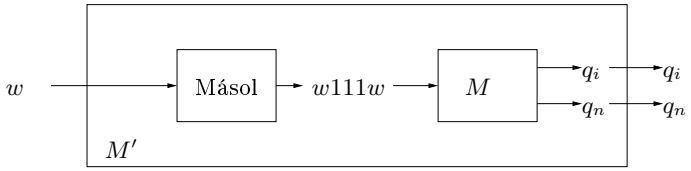
1. U először megvizsgálja, hogy v $\langle M, w\rangle$ alakú-e. Ha nem akkor elutasítja a bemenetet. Ezt helyesen teszi, hisz ebben az esetben, megállapodásunk szerint v egy olyan Turing-gépet kódol, ami nem fogad el egyetlen bemenő szót sem. Ekkor viszont v nem lehet eleme L_{u}-nak.
2. U rámásolja w-t a második szalagra a kódolt formában.
3. U 0-t ír a harmadik szalagjára, ezzel reprezentálva M kezdőállapotát.
4. U szimulálja M egy lépését az alábbi módon. Keres egy $0^i10^j10^r10^s10^t$ alakú részszót M kódjában az első szalagján úgy, hogy 0^i a harmadik szalagon lévő állapot legyen, 0^j pedig az a 0-s blokk, ami a második szalagon a fej pozíciójában kezdődik. Ezek után U elvégzi a fent kódolt átmenet alapján M egy lépését:
   - Kitörli a harmadik szalagon 0^i-t és a másodikról átmásolja a harmadikra 0^r-t (annak az állapotnak a kódját, amibe M-lépne).
   - Kicseréli a második szalagon 0^j-t 0^s-re, azaz átírja M szalagszimbólumát az átmenetnek megfelelően. Ehhez ha kell felhasználja a negyedik szalagot, ugyanis ha $j \neq s$, akkor rámásolja a negyedik szalagra a második szalagról a 0^i mögötti részt, kitörli a második szalagon 0^i-t, a helyére írja 0^s-t, végül visszamásolja a negyedik szalagon lévő szót a második szalagra és a fejet a 0^s blokk elejére állítja.
   - U megkeresi a második szalagon a fej pozícióján kezdődő 0-s blokktól balra vagy jobbra lévő 0-s blokk kezdetét, vagy helyben marad attól függően, hogy t értéke 1, 2, vagy 3.
5. Ha U azt találja, hogy M a 4-ik pontban q_i vagy q_n állapotba lépett, akkor U is q_i-be vagy q_n-be lép. Egyébként pedig folytatja M következő lépésének szimulálását a 4-ik ponttal.

Világos, hogy U pontosan akkor fogad el egy $\langle M, w\rangle$ alakú bemenetet, ha $w \in L(M)$, vagyis ha $\langle M, w\rangle \in L_{\mathrm{u}}$. Tehát U felismeri L_{u}-t, és ez az amit bizonyítani akartunk. □

Most megmutatjuk, hogy L_{u} nem eldönthető.

Tétel $L_{\mathrm{u}} \notin R$.

Bizonyítás Az állítást indirekt módon bizonyítjuk. Tegyük fel, hogy L_{u} eldönthető. Akkor egy korábbi tételünk alapján $\bar{L}_{\mathrm{u}}$ is eldönthető. Legyen akkor M az a Turing-gép, ami eldönti $\bar{L}_{\mathrm{u}}$-t. M-ből megkonstruálunk egy olyan M' Turing-gépet, ami $L_{\text{átló}}$-t dönti el, ellentmondva azon korábbi tételünknek, mely szerint $L_{\text{átló}} \notin RE$. M' felépítése az alábbi ábrán látható.



Adott w bementere M' a következőket csinálja:

1. Előállítja w-ből a $w' = w111w$ szót.
2. w'-n szimulálja M-et
3. Ha M elfogadja w'-t, akkor M' elfogadja w-t.
4. Ha M elutasítja w'-t, akkor M' is elutasítja w-t.

Vagyis w akkor és csak akkor eleme $L(M')$-nek, ha $w111w \in \bar{L}_{\mathrm{u}}$, azaz $w \in L_{\text{átló}}$. Azt kaptuk tehát, hogy $L(M') = L_{\text{átló}}$, vagyis $L_{\text{átló}}$ rekurzívan felsorolható. Ez viszont ellentmond annak a korábbi tételünknek, mely szerint $L_{\text{átló}}$ nem rekurzívan felsorolható. Ezzel a tétel bizonyítását befejeztük. □

3.5. További eldönthetetlen problémák

Tegyük fel, hogy van két eldöntési problémánk, legyenek ezek L_1 és L_2. Előfordulhat, hogy csak a L_2 probléma eldöntésére van egy A_2 algoritmusunk, a L_1-ére nincs. Ha viszont a L_1 probléma minden w példányához meg tudjuk konstruálni a L_2 probléma egy w' példányát úgy, hogy a w pontosan akkor „igen” példánya L_1-nek, ha w' „igen” példánya L_2-nek, akkor már a L_1 problémát is el tudjuk dönteni az alábbi módon. A L_1-et eldöntő A_1 algoritmus egy w bementből először elkészíti a L_2 fentebb leírt w' példányát. Ezután A_1

szimulálja A_2 működését a w' bemeneten. Ha A_2 „igen" választ ad a w' bemenetre, akkor A_1 is „igen" választ ad a w bemenetre, míg ha A_2 „nem" választ ad, akkor A_1 is „nem" választ ad a bemenetre. Könnyen látható, hogy az így leírt A_1 algoritmus valóban a L_1 problémát dönti el. A most vázolt módszert nevezzük visszavezetésnek, mely során a L_1 probléma eldöntését visszavezetjük egy másik probléma eldöntésére.

A módszert használhatjuk arra is, hogy egy problémáról megmutassuk, hogy eldönthetetlen. Tudjuk például, hogy az előző fejezetben vázolt L_{u} probléma eldönthetetlen. Ha L_{u}-t sikerülne visszavezetni egy másik problémára, akkor ezáltal bizonyítani tudnánk, hogy ez az újabb probléma is eldönthetetlen (gondoljuk csak meg, ha az újabb problémánk mégis eldönthető lenne, akkor a visszavezetést felhasználva meg tudnánk adni egy L_u-t eldöntő algoritmust, ami ellentmondáshoz vezet).

A visszavezetést formálisan az alábbi módon definiáljuk.

Definíció Legyen Σ és Δ két ábécé és f egy Σ^*-ból Δ^*-ba képező függvény. Azt mondjuk, hogy f kiszámítható, ha van olyan M Turing-gép, hogy M-et egy $w \in \Sigma^*$ szóval a bemenetén indítva, M úgy áll meg, hogy a szalagján az $f(w)$ szó van.

Legyen $L_1 \in \Sigma^*$ és $L_2 \in \Delta^*$ két nyelv (azaz eldöntési probléma). Azt mondjuk, hogy L_1 visszavezethető L_2-re, ha van olyan $f : \Sigma^* \to \Delta^*$ kiszámítható függvény, hogy minden $w \in \Sigma^*$ szóra, $w \in L_1$ akkor és csak akkor teljesül, ha $f(w) \in L_2$ is teljesül. □

Tétel Legyen L_1 és L_2 két eldöntési probléma és tegyük fel, hogy L_1 visszavezethető L_2-re. Akkor igazak az alábbi állítások:

1. Ha L_1 eldönthetetlen, akkor L_2 is az.
2. Ha $L_1 \notin RE$, azaz nem rekurzívan felsorolható, akkor $L_2 \notin RE$ szintén teljesül.

Bizonyítás Csak a második állítást bizonyítjuk, az első bizonyítása hasonló. Indirekt módon tegyük fel, hogy L_2 rekurzívan felsorolható. Akkor van olyan M_2 Turing-gép, hogy $L_2 = L(M_2)$. Továbbá, van olyan M' Turing-gép, ami kiszámolja az L_1 nyelv L_2-re való visszavezetését. Ezen gépek segítségével megadunk egy olyan M_1 Turing-gépet, ami L_1-et ismeri fel. M_1 egy w bemeneten először szimulálja az M' gépet. Amikor végez, akkor a szalagján lévő $f(w)$ szón szimulálja M_2 működését. Ha M_2 elfogadja az $f(w)$ szót, akkor M_1 is elfogadja a w-t. Ha M_2 elutasítja $f(w)$-t, akkor M_1 is elutasítja a w-t. Ha M_2 nem áll meg az $f(w)$ bemeneten, akkor M_1 sem áll meg w-n. Könnyen belátható, hogy az így vázolt M_1 gép pontosan az L_1 nyelvet ismeri fel. Feltevésünk szerint azonban az $L_1 \notin RE$, vagyis L_1 nem felismerhető, tehát ellentmondáshoz jutottunk. Következésképpen L_2 nem lehet rekurzívan felsorolható.

Most definiáljuk azt a problémát, mely a Turing-gépek megállási problémája-

ként ismert, és szintén egy eldönthetetlen probléma. Ennek a problémának az eldönthetetlenségét úgy bizonyítjuk, hogy visszavezetjük rá az L_{u} problémát.

Legyen $L_{\mathrm{halt}} = \{\langle M, w\rangle \mid \text{M megáll a } w \text{ bemeneten}\}$, azaz L_{halt} azon (M, w) Turing-gép és bemenet párosokat tartalmazza megfelelően elkódolva, hogy az M gép megáll a w bemenetet.

Kezdeti sikerként először bebizonyítjuk, hogy L_{halt} rekurzívan felsorolható.

Tétel $L_{\mathrm{halt}} \in RE$.

Bizonyítás Vegyük azt az U Turing-gépet, ami az L_{u} nyelvet ismeri fel. Ezt a gépet fogjuk módosítani úgy, hogy L_{halt}-ot ismerje fel. U minden olyan átmenetét, ami q_n-be megy irányítsuk q_i-be, és jelöljük a kapott Turing-gépet M'-vel. Nem nehéz belátni, hogy egy M Turing-gépre és annak w bemenő szavára az $\langle M, w\rangle$ szó pontosan akkor eleme $L(M')$-nek, ha M megáll a w bemeneten (q_i-ben vagy q_n-ben). Ez pedig azt jelenti, hogy $L_{\mathrm{halt}} = L(M')$, vagyis M' az L_{halt} nyelvet ismeri fel. □

Nyilvánvaló, hogy ha lenne egy Turing-gép, ami képes lenne eldönteni is az L_{halt} nyelvet, akkor tetszőleges Turing-gépről és így a Church-Turing tézis értelmében tetszőleges algoritmusról, C++, Pascal programról, stb.) el tudnánk dönteni, hogy megáll-e a bemenetén vagy sem. Ez a Turing-gép egy igen hasznos eszköz lenne a számunkra, sajnálatos módon azonban ilyen gép nem létezik.

Tétel Az L_{halt} nyelv eldönthetetlen.

Bizonyítás Visszavezetjük az L_{halt} problémára az L_{u} problémát, ami egy korábbi tételünk szerint, mivel L_{u} eldönthetetlen, bizonyítja azt, hogy L_{halt} is eldönthetetlen. A visszavezetés a következő módon történik. Egy tetszőleges M Turing-géphez legyen M' a következő Turing-gép.

M' egy w bemeneten:

1. Futtatja M-et a w szón (tulajdonképpen meghívja az U univerzális Turing gépet a $\langle M, w\rangle$ szóra).

2. Ha M elfogadja a w bemenetet, akkor M' is elfogadja w-t.

3. Ha M elutasítja w-t, akkor M' egy olyan állapotba megy, ahol egy végtelen ciklusban lépteti a fejet jobbra, tehát M' ebben az esetben soha nem áll meg.

Könnyű belátni, hogy $\langle M, w\rangle \in L_{\mathrm{u}}$ akkor és csak akkor teljesül, ha $\langle M', w\rangle \in L_{\mathrm{halt}}$ teljesül. Továbbá, M' megkonstruálható M-ből egy Turing-géppel. Azt kaptuk tehát, hogy L_{u} visszavezethető L_{halt}-ra, amiből következik, hogy L_{halt} eldönthetetlen. □

3.5.1. Rice tétele

Ebben az alfejezetben azt mutatjuk meg, hogy a Turing-gépekkel, pontosabban az általuk felismert nyelvekkel kapcsolatos összes nem triviális kérdés algoritmikusan eldönthetetlen. Először tisztázni kell, hogy mit nevezünk a rekurzívan felsorolható nyelvek egy nem triviális tulajdonságának.

Definíció Ha $\mathcal{P}$ a rekurzívan felsorolható nyelvek egy halmaza, akkor $\mathcal{P}$-t a rekurzívan felsorolható nyelvek egy tulajdonságának nevezzük. Továbbá, $\mathcal{P}$ egy nem triviális tulajdonság, ha $\mathcal{P} \neq \varnothing$ és $\mathcal{P} \neq RE$. Azt mondjuk, hogy egy $L \in RE$ nyelv rendelkezik a $\mathcal{P}$ tulajdonsággal, ha $L \in \mathcal{P}$. □

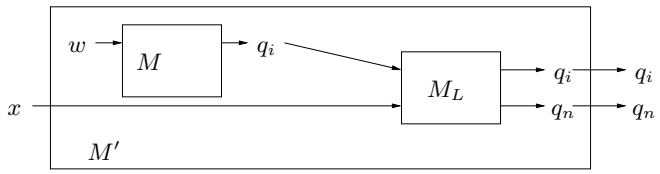
Például ha $\mathcal{P}$ az üres halmaz, akkor egyetlen L rekurzívan felsorolható nyelv sem rendelkezik a $\mathcal{P}$ tulajdonsággal. Viszont ha $\mathcal{P} = \{\varnothing\}$, azaz $\mathcal{P}$ az üres nyelvet tartalmazza (ami nyilván egy rekurzívan felsorolható nyelv), akkor kizárólag a $\varnothing$, vagyis az üres nyelv rendelkezik a $\mathcal{P}$ tulajdonsággal.

Ebben az alfejezetben megmutatjuk, hogy ha $\mathcal{P}$ egy nem triviális tulajdonság, akkor nincs olyan Turing-gép, ami tetszőleges $L \in RE$ nyelvre eldöntené, hogy L rendelkezik-e a $\mathcal{P}$ tulajdonsággal, azaz $L \in \mathcal{P}$ vagy sem (megjegyezzük, hogy ha $\mathcal{P}$ egy triviális tulajdonság lenne, akkor a probléma eldöntése is triviális lenne). De hogyan adjunk a Turing-gép bemenetére egy nyelvet, hisz a rekurzívan felsorolható nyelvek általában végtelen sok szót tartalmaznak? Ahogy azt már korábban láttuk, ilyenkor nem maga a nyelv lesz a bemenet, hanem egy a nyelvet reprezentáló véges eszköz, esetünkben egy Turing-gép. Tehát igazából nem azt akarjuk eldönteni, hogy egy adott L nyelv rendelkezik-e a $\mathcal{P}$ tulajdonsággal, hanem azt, hogy az L-et felismerő M Turing-gép kódja eleme-e az $L_{\mathcal{P}}$ nyelvnek, ahol $L_{\mathcal{P}}$ azon Turing-gépek kódjait tartalmazza, melyek $\mathcal{P}$ tulajdonsággal rendelkező nyelveket ismernek fel. Az alábbi tételben azt mondja ki, hogy az $L_{\mathcal{P}}$ nyelv eldönthetetlen, amennyiben $\mathcal{P}$ egy nem triviális tulajdonság.

Tétel Legyen $\mathcal{P}$ a rekurzívan felsorolható nyelvek egy nem triviális tulajdonsága. Akkor az $L_{\mathcal{P}}$ nyelv eldönthetetlen.

Bizonyítás Az állítás bizonyításához legyen $\mathcal{P}$ a rekurzívan felsorolható nyelvek egy nem triviális tulajdonsága. Tegyük fel, hogy $\varnothing \notin \mathcal{P}$ (a bizonyítás végén majd visszatérünk ahhoz az esethez, amikor $\varnothing \in \mathcal{P}$). Legyen továbbá L egy tetszőleges nyelv $\mathcal{P}$-ből (mivel $\mathcal{P}$ nem triviális és $\varnothing \notin \mathcal{P}$, L nem lehet az üres nyelv). Legyen M_L az L-et felismerő Turing-gép.

A következőkben visszavezetjük az L_u nyelvet az $L_{\mathcal{P}}$-re. Legyen $\langle M, w\rangle$ egy tetszőleges Turing-gép és annak egy w bemenete. Megmutatjuk, hogy megkonstruálható egy M' Turing-gép úgy, hogy $L(M') = \varnothing$, ha $w \notin L(M)$ és $L(M') = L$, ha $w \in L(M)$. A megkonstruált M' egy kétszalagos Turing-gép lesz, a gép vázlata az alábbi ábrán látható.



M' egy tetszőleges x bemeneten a következőket csinálja:

1. M', függetlenül attól, hogy mi a bemenete, szimulálja az egyik szalagján M működését a w szón (M és w kódja be van építve M' kódjába; M' felmásolja a szalagjára a w-t és meghívja az univerzális Turing-gépet, melynek segítségével szimulálja M működését w-n).

2. Ha M' azt találja, hogy M nem fogadja el w-t, akkor M' nem csinál semmit, vagyis nem fogadja el x bemenetet. Ebben az esetben $L(M') = \varnothing$, vagyis $\langle M' \rangle \notin L_{\mathcal{P}}$.

3. Ha M' azt találja, hogy M elfogadja w-t, akkor elkezdi szimulálni M_L működését az x bemeneten. Ebben az esetben pedig $L(M) = L$, vagyis $\langle M' \rangle \in L_{\mathcal{P}}$.

Látható, hogy $\langle M, w \rangle \in L_{\mathrm{u}}$ akkor és csak akkor teljesül, ha $\langle M' \rangle \in L_{\mathcal{P}}$, vagyis a fenti eljárás L_{u} visszavezetése $L_{\mathcal{P}}$-re. Mivel L_{u} eldönthetetlen kapjuk, hogy $L_{\mathcal{P}}$ is eldönthetetlen.

Meg kell még vizsgálni azt az esetet, amikor $\varnothing \in \mathcal{P}$. Ebben az esetben ismételjük meg a fenti bizonyítást a $\bar{\mathcal{P}} = RE - \mathcal{P}$ tulajdonságra. Azt kapjuk, hogy $L_{\bar{\mathcal{P}}}$ nem eldönthető. Tegyük fel most, hogy az $L_{\mathcal{P}}$ nyelv viszont eldönthető. Korábbi tételünk alapján tudjuk, hogy ebben az esetben $\bar{L}_{\mathcal{P}}$ is eldönthető. Másrészt nem nehéz belátni, hogy $L_{\bar{\mathcal{P}}} = \bar{L}_{\mathcal{P}}$, ami azt jelenti, hogy $L_{\bar{\mathcal{P}}}$ is eldönthető, ez pedig ellentmondás, amiből következik, hogy $L_{\mathcal{P}}$ eldönthetetlen, amit bizonyítani akartunk. □

A fentiek alapján az alábbi problémák mindegyike eldönthetetlen.

1. Egy tetszőleges M Turing-gép az üres nyelvet ismeri-e fel. (Ebben az esetben $\mathcal{P} = \{\varnothing\}$.)

2. Egy M Turing-gép véges nyelvet ismer-e fel ($\mathcal{P} = \{L \mid L \text{ véges}\}$).

3. Egy M Turing-gép reguláris nyelvet ismer-e fel ($\mathcal{P} = \{L \mid L \text{ reguláris}\}$).

4. Egy M Turing-gép környezetfüggetlen nyelvet ismer-e fel ($\mathcal{P} = \{L \mid L \text{ környezetfüggetlen}\}$).

4. fejezet

Bevezetés a bonyolultságelméletbe

Amint azt az előadás elején említettük, a bonyolultságelmélet célja a megoldható (és ezen belül az eldönthető) problémák osztályozása a megoldáshoz szükséges erőforrások (jellemzően az idő és a tár) mennyisége szerint. Szükségünk lesz az alábbi definíciókra.

Definíció Legyen $f(n) : N \to N$ egy függvény. Akkor

$$\mathbf{TIME}(f(n)) = \{L \mid L \text{ eldönthető } \mathcal{O}(f(n)) \text{ időigényű Turing-géppel}\}.$$

Továbbá, $\mathbf{P} = \bigcup_{k \geq 1} \mathbf{TIME}(n^k)$. □

Tehát **P** azon nyelveket tartalmazza, melyek eldönthetőek polinom időkorlátos determinisztikus Turing-géppel. Ilyen például a jól ismert ELÉRHETŐSÉG probléma, melynek bemenete egy G gráf és annak két kitüntetett csúcsa (s és t). A kérdés az, hogy van-e a G-ben út s-ből t-be. Ha az ELÉRHETŐSÉG problémára nyelvként tekintünk, akkor írhatjuk azt, hogy ELÉRHETŐSÉG$= \{\langle G, s, t\rangle \mid$ G-ben van út s-ből t-be$\}$, ahol a $\langle G, s, t\rangle$ jelölés, amint azt már megszokhattuk, a G, s, t egy megfelelő elkódolása valamilyen ábécé feletti szóban. Könnyen megadható az ELÉRHETŐSÉG problémát polinom időben eldöntő Turing-gép, tehát ELÉRHETŐSÉG $\in \mathbf{P}$. Most definiáljuk a nemdeterminisztikus Turing-gépek analóg nyelvosztályait.

Definíció Ha $f(n) : N \to N$ egy függvény, akkor

$$\mathbf{NTIME}(f(n)) = \{L \mid L \text{ eldönthető } \mathcal{O}(f(n)) \text{ időigényű } \mathbf{nemdeterminisztikus} \text{ Turing-géppel}\}.$$

Továbbá, $\mathbf{NP} = \bigcup_{k \geq 1} \mathbf{NTIME}(n^k)$. □

Az **NP**-beli problémák rendelkeznek egy közös tulajdonsággal az alábbi értelemben. Ha tekintjük egy **NP**-beli probléma egy példányát és egy lehetséges „bizo-

nyítékot" arra nézve, hogy ez a példány „igen" példánya az adott problémának, akkor ezen bizonyíték helyességének leellenőrzése polinom időben elvégezhető. Ennek megfelelően egy **NP**-beli problémát eldöntő nemdeterminisztikus Turing-gép általában úgy működik, hogy „megsejti" a probléma bemenetének egy lehetséges megoldását, és polinom időben leellenőrzi, hogy a megoldás helyes-e.

Tekintsük például a SAT problémát. Tudjuk, hogy SAT igen példányai azon konjunktív normálformák, melyek kielégíthetőek. Tekintsünk egy tetszőleges ϕ konjunktív normálformát. Annak a bizonyítéka, hogy a ϕ kielégíthető egy olyan változóhozzárendelés, ami mellett kiértékelve a ϕ-t igaz értéket kapunk. Egy tetszőleges változóhozzárendelés tehát a ϕ kielégíthetőségének egy lehetséges bizonyítéka. Annak leellenőrzése pedig, hogy ez a hozzárendelés tényleg igazzá teszi-e ϕ-t polinom időben elvégezhető. Ennek megfelelően, a SAT eldönthető egy olyan nemdeterminisztikus Turing-géppel, mely megsejt egy változóhozzárendelést, és polinom időben leellenőrzi, hogy az kielégíti-e a bemenetet. Következésképpen a SAT egy **NP**-beli probléma.

A SAT azonban speciális is az **NP**-beli problémák között, mert az eldöntésére ismert összes algoritmus olyan, hogy a bemenet méretének függvényében exponenciális a lépésszáma. Ezek az algoritmusok alapvetően úgy működnek, hogy egy ϕ bemenetre szisztematikusan megvizsgálják a lehetséges megoldások terét, azaz addig állítják elő a ϕ változóinak értéket adó változóhozzárendeléseket, amíg nem találnak egy olyat ami kielégíti a ϕ-t. Az összes lehetséges hozzárendelés száma viszont a ϕ-ben szereplő változók számának a méretében exponenciális nagyságrendű, tehát a SAT-ot eldöntő algoritmus is exponenciális időigényű lesz. Amint arról már volt szó, a determinisztikus Turing-gép egy algoritmus modell, következik tehát, hogy a SAT-ot, jelenlegi ismereteink szerint, csak exponenciális időigényű determinisztikus Turing-géppel tudjuk eldönteni. A nemdeterminisztikus Turing-gép ezzel szemben rendelkezik azzal a nem realisztikus tulajdonsággal, hogy képes ráhibázni a megfelelő megoldásra, és nem kell szisztematikusan átvizsgálnia a lehetséges megoldások terét.

Az a definíciókból következik, hogy fennáll a $\mathbf{P} \subseteq \mathbf{NP}$ tartalmazás. Az a sejtés (azaz még nem bizonyított), hogy a fenti tartalmazás valódi. Éppen a SAT az egyik olyan probléma, ami valószínűleg nem eleme a **P**-nek, viszont **NP**-beli. Másrészt igaz, hogy ha sikerülne a SAT eldöntésére polinomiális időigényű algoritmust találni, akkor ez azt jelentené, hogy az **NP** osztály megegyezik a **P** osztállyal. A SAT-ot és a vele megegyező nehézségű problémákat nevezzük majd később **NP**-teljes problémáknak.

4.1. NP-teljes problémák

Ahhoz, hogy az **NP** osztály szerkezetét vizsgálni tudjuk, szükségünk lesz a visszavezetések egy speciális osztályára. Ezek a polinom idejű visszavezetések.

Definíció Legyen Σ és Δ két ábécé és f egy Σ^*-ból Δ^*-ba képező függvény. Azt mondjuk, hogy f polinom időben kiszámítható, ha kiszámítható egy polinom időigényű Turing-géppel.

Legyen $L_1 \in \Sigma^*$ és $L_2 \in \Delta^*$ két nyelv. Azt mondjuk, hogy L_1 polinom időben visszavezethető L_2-re (jele: $L_1 \leq_p L_2$), ha van olyan $f : \Sigma^* \to \Delta^*$ polinom időben kiszámítható függvény, hogy minden $w \in \Sigma^*$ szóra, $w \in L_1$ akkor és csak akkor teljesül, ha $f(w) \in L_2$ is teljesül. □

Két egymásra polinom időben visszavezethető probléma között az alábbi kapcsolatot figyelhetjük meg.

Tétel Legyen L_1 és L_2 két probléma úgy, hogy $L_1 \leq_p L_2$. Ha L_2

1. **P**-beli, akkor L_1 is **P**-beli.
2. **NP**-beli, akkor L_1 is **NP**-beli.

Bizonyítás Csak a második esetet vizsgáljuk (az első eset bizonyítása hasonló). Tegyük fel tehát, hogy $L_2 \in \mathbf{NP}$. Megmutatjuk, hogy $L_1 \in \mathbf{NP}$ is teljesül. Legyen ugyanis M_1 az a Turing-gép, ami polinom időben kiszámolja az $L_1 \leq_p L_2$ visszavezetést, M_2 pedig az a nemdeterminisztikus gép, ami polinom időben eldönti L_2-őt. Konstruáljunk meg M_1-ből és M_2-ből egy M Turing-gépet a következő módon. M egy w bemenetre kiszámolja az $f(w)$ szót (szimulálja M_1-et), majd a kapott $f(w)$ szóra meghívja M_2-t. Ha M_2 elfogadja $f(w)$-t, akkor M is elfogadja w-t, ha M_2 elutasítja $f(w)$-t, akkor M is elutasítja w-t. Könnyű belátni, hogy az így vázolt M is egy polinom idejű nemdeterminisztikus Turing-gép, ami viszont L_1-et dönti el. Vagyis $L_1 \in \mathbf{NP}$ is teljesül, amit bizonyítani akartunk. □

Most definiáljuk a **NP** egy fontos részosztályát.

Definíció Legyen L egy probléma. Azt mondjuk, hogy L **NP**-teljes, ha

1. **NP**-beli és
2. minden további **NP**-beli probléma polinom időben visszavezethető L-re.

□

Most megmutatjuk, hogy ha egy **NP**-teljes probléma eleme **P**-nek, akkor $\mathbf{P} = \mathbf{NP}$.

Tétel Legyen L egy **NP**-teljes probléma. Ha $L \in \mathbf{P}$, akkor $\mathbf{P} = \mathbf{NP}$.

Bizonyítás Tegyük fel, hogy $L \in \mathbf{P}$. Mivel $\mathbf{P} \subseteq \mathbf{NP}$, elég megmutatni, hogy $\mathbf{NP} \subseteq \mathbf{P}$ is teljesül, azaz minden $L' \in \mathbf{NP}$-re $L' \in \mathbf{P}$. Ez viszont igaz, hiszen mivel L **NP**-teljes, $L' \leq_p L$ és ebben az esetben az előző tételünk alapján $L' \in \mathbf{P}$ is fennáll. □

Ahogy azt az **NP**-teljes problémák definíciójában láttuk, ha meg akarjuk mutatni, hogy egy **NP**-beli probléma **NP**-teljes, akkor meg kell mutatni, hogy minden

NP-beli probléma visszavezethető rá. Ez, mint ahogy azt a SAT esetében hamarosan látni is fogjuk, általában nehéz feladat. Az alábbi tételt alkalmazva viszont egy **NP**-teljes probléma segítségével további **NP**-beli problémákról láthatjuk be, hogy **NP**-teljesek.

Tétel Legyen L_1 egy **NP**-teljes és L_2 egy **NP**-beli probléma. Ha $L_1 \leq_p L_2$, akkor L_2 is **NP**-teljes.

Bizonyítás Legyen L egy tetszőleges **NP**-beli probléma. Mivel L_1 **NP**-teljes kapjuk, hogy $L \leq_p L_1$. Legyenek M_1 az $L \leq_p L_1$ visszavezetést és M_2 az $L_1 \leq_p L_2$ visszavezetést kiszámító polinom időigényű Turing-gép. M_1-ből és M_2-ből könnyen megadható egy olyan M polinom időigényű Turing-gép, ami az $L \leq_p L_2$ visszavezetést számolja ki. Mivel L egy tetszőleges **NP**-beli nyelv volt kapjuk, hogy minden **NP**-beli nyelv visszavezethető polinom időben L_2-re. Mivel L_2 **NP**-beli következik, hogy L_2 is **NP**-teljes. □

4.1.1. SAT NP-teljes

Ebben a részben megmutatjuk a bonyolultságelmélet egyik fontos eredményét, nevezetesen, hogy a SAT egy **NP**-teljes probléma. Ehhez először definiáljuk a SAT problémát, most mint nyelvet, valamint néhány további, a logikában használatos alapfogalmat.

Definíció Literálnak nevezünk egy ítéletváltozót (melynek értéke `igaz` illetve `hamis` lehet) illetve annak negáltját. Tagnak nevezzük a literálok diszjunkcióját („vagy" kapcsolatát) és konjunktív normálformának (knf) a tagok konjunkcióját („és" kapcsolatát). Ezek után a SAT problémát a következőképpen definiáljuk.

SAT=$\{\langle\phi\rangle \mid \phi$ kielégíthető konjunktív normálformában adott logikai formula$\}$

Tehát SAT azon konjunktív normálformákat tartalmazza, egy megfelelő ábécé felett elkődolva, melyek kielégíthetőek. □

Példa $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \in$ SAT.

Tétel (Cook tétele) SAT **NP**-teljes.

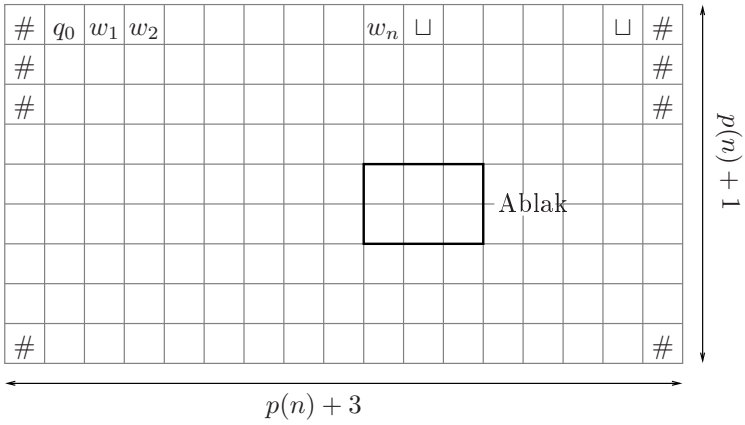
Bizonyítás Azt, hogy SAT **NP**-beli, a fejezet elején már láttuk. Amit be kell még bizonyítani az az, hogy minden **NP**-beli nyelv polinom időben visszavezethető SAT-ra. Legyen L egy tetszőleges **NP**-beli nyelv és legyen $M = (Q, \Sigma, \Gamma, \delta, q_0, q_i, q_n)$ egy $p(n)$ időigényű nemdeterminisztikus Turing-gép valamely $p : N \to N$ polinomra úgy, hogy $L = L(M)$. Feltehető, hogy minden $n \in N$-re, $p(n) \geq n$.

A feladatunk az, hogy M minden w bemenő szavához elkészítsünk egy ϕ formulát úgy, hogy:

- $w \in L$ akkor és csak akkor, ha ϕ kielégíthető,
- a $w \mapsto \phi$ hozzárendelés megvalósítható polinom időkorlátos determinisz-

tikus Turing-géppel.

M működése w szón leírható egy olyan táblázattal, melynek első sora M kezdőkonfigurációjának, két egymást követő sora pedig M két egymást követő konfigurációjának felel meg a w-n. Egy ilyen táblázat az alábbi ábrán látható.



Tegyük fel, hogy $w = w_1 w_2 \dots w_n$ valamely $n \in N$ számra. Mivel M $p(n)$ időkorlátos gép, azaz minden számítási sorozata maximum $p(n)$ lépésben véget ér a w-n, a fenti táblázat $(p(n)+3) \cdot (p(n)+1)$ darab cellából áll. A táblázatnak tehát van egy olyan i-ik sora $(1 \leq i \leq p(n)+1)$ mely M egy megállási konfigurációját reprezentálja. Megegyezünk abban, hogy a táblázat összes i-nél nagyobb sorszámú sora megegyezik az i-ik sorral.

A táblázat minden cellája a $C = Q \cup \Gamma \cup \{\#\}$ halmaz egy elemét tartalmazhatja. Minden cellához és C-beli szimbólumhoz bevezetünk egy ítéletváltozót, azaz minden $1 \leq i \leq p(n)+1$-re, $1 \leq j \leq p(n)+3$-ra és $s \in C$-re, az $x_{i,j,s}$ egy ítéletváltozó lesz ϕ-ben. Az $x_{i,j,s}$ változó igaz értéke a ϕ-ben azt fogja jelenteni, hogy a táblázat i-ik sorának j-ik oszlopában az s szimbólum van. A táblázat elfogadó, vagyis $w \in L$, ha az utolsó sora egy elfogadó konfiguráció, azaz az utolsó sorban szereplő állapot a q_i.

A megkonstruált ϕ-nek azt kell leírnia, hogy a táblázat egy elfogadó táblázat. Ennek megfelelően ϕ négy részformulából áll: $\phi = \phi_0 \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}$.

ϕ_0 azt fejezik ki, hogy a táblázat minden egyes mezőjében pontosan egy karakter van:

$$\phi_0 = \bigwedge_{\substack{1 \leq i \leq p(n)+1 \\ 1 \leq j \leq p(n)+3}} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \bigwedge_{s,t, \in C, s \neq t} (\neg x_{i,j,s} \vee \neg x_{i,j,t}) \right].$$

ϕ_{start} azt fejezi ki, hogy a táblázat első sorában a w-hez tartozó kezdőkonfi-

guráció van:

$$\phi_{start} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge x_{1,3,w_1} \wedge \ldots \\ \wedge x_{1,n+2,w_n} \wedge x_{1,n+3,\sqcup} \wedge \ldots \wedge x_{1,p(n)+2,\sqcup} \wedge x_{1,p(n)+3,\#}.$$

ϕ_{accept} azt fejezi ki, hogy a táblázat utolsó sorában elfogadó konfiguráció van:

$$\phi_{accept} = \bigvee_{j=2}^{p(n)+2} x_{p(n)+1,j,q_i}.$$

Végezetül, a ϕ_{move} formulának azt kell kifejeznie, hogy a táblázat két egymást követő sora M két egymást követő konfigurációjának felel meg a w-n. Ahhoz, hogy ezt formalizálni tudjuk, szükségünk lesz némi előkészületre.

Nevezzük a táblázat egy $\begin{array}{|c|c|c|}\hline a_1 & a_2 & a_3 \\ \hline a_4 & a_5 & a_6 \\ \hline \end{array}$ részét a táblázat egy ablakának. Azt mondjuk, hogy a táblázat egy ablaka legális, ha az megengedett az M átmeneti relációja által (ha a felső és az alsó sor megegyezik, akkor szintén legális az ablak).

Példaként tegyük fel, hogy $\delta(q_1, a) = \{(q_1, b, R)\}$ és $\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$ ($q_1, q_2 \in Q$ és $a, b, c \in \Gamma$). Akkor a következő két ablak legális: $\begin{array}{|c|c|c|}\hline a & a & q_1 \\ \hline a & a & b \\ \hline \end{array}$, $\begin{array}{|c|c|c|}\hline a & q_1 & b \\ \hline q_2 & a & c \\ \hline \end{array}$. Másrészt, bármilyen további átmeneteket engedjen is meg δ, az $\begin{array}{|c|c|c|}\hline a & a & a \\ \hline a & b & a \\ \hline \end{array}$ ablak nem legális.

Ezek után az általunk megadott ϕ_{move} azt fogja kifejezni, hogy a táblázat minden ablaka legális. Az, hogy ϕ_{move} így azt fejezi ki, amit elvárunk tőle következik az alábbi állításból, melyet nem bizonyítunk:

Ha a táblázat első sora az M kezdőkonfigurációja w-n és a táblázat minden ablaka legális, akkor a táblázat minden sora egy olyan konfigurációja M-nek, ami legálisan követi M-nek a táblázat megelőző sorában lévő konfigurációját.

Legyen tehát

$$\phi_{move} = \bigwedge_{\substack{1 \le i \le p(n) \\ 2 \le j \le p(n)+2}} \psi_{i,j},$$

ahol adott i-re és j-re $\psi_{i,j}$ az a formula, aminek azt fejezi ki, hogy a táblázat i-sorában és annak $j-1$-ik pozíciójában kezdődő ablak legális. Ezt az állítást a

$$\bigvee_{\substack{(a_1,\ldots,a_6) \\ \text{legális}}} x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6}$$

formulával lehet leírni, de ez a formula sajnos nem knf. Ezért $\psi_{i,j}$-vel egy ekvivalens állítást fogunk formalizálni: nem igaz az, hogy a táblázat i-sorában és annak $j-1$-ik pozíciójában kezdődő ablak nem legális. Ezt az állítást a

$$\neg\Big(\bigvee_{\substack{(a_1,\ldots,a_6)\\ \text{nem legális}}} x_{i,j-1,a_1}\wedge x_{i,j,a_2}\wedge x_{i,j+1,a_3}\wedge x_{i+1,j-1,a_4}\wedge x_{i+1,j,a_5}\wedge x_{i+1,j+1,a_6}\Big)$$

formula írja le. Ebből a formulából a De Morgan azonosságokat felhasználva kapjuk a keresett $\psi_{i,j}$ formulát:

$$\psi_{i,j}=\bigwedge_{\substack{(a_1,\ldots,a_6)\\ \text{nem legális}}} \neg x_{i,j-1,a_1}\vee\neg x_{i,j,a_2}\vee\neg x_{i,j+1,a_3}\vee \\ \neg x_{i+1,j-1,a_4}\vee\neg x_{i+1,j,a_5}\vee\neg x_{i+1,j+1,a_6}.$$

Mivel így $\psi_{i,j}$ knf, kapjuk, hogy ϕ_{move} is az. Másrészt a ϕ_0, ϕ_{start}, valamint a ϕ_{accept} részformulák mindegyike knf. Ezért az egész ϕ formula maga is knf. Felhasználva továbbá azt, hogy ezen részformulák mérete n függvényében polinomiális nagyságrendű, megmutatható, hogy ϕ konstrukciója n függvényében polinom idő alatt elvégezhető.

Másrészt az is könnyen látható, hogy $w\in L$ akkor és csak akkor igaz, ha ϕ kielégíthető. Tehát ϕ konstrukciója az L nyelv polinom idejű visszavezetése SAT-ra. Mivel L tetszőleges **NP**-beli nyelv volt kapjuk, hogy SAT **NP**-teljes. □

4.1.2. További NP-teljes problémák

Érdekes módon a SAT probléma akkor is **NP**-teljes marad, ha a probléma példányainak azokat a knf-kat tekintjük, melyek minden tagja pontosan három literált tartalmaz. A továbbiakban ezt az így kapott problémát vizsgáljuk.

Definíció Legyen $k\geq 1$. kSAT $=\{\langle\phi\rangle : \langle\phi\rangle\in$ SAT, ϕ minden tagjában k literál van $\}$. □

Most megmutatjuk, hogy a 3SAT probléma is **NP**-teljes.

Tétel 3SAT **NP**-teljes.

Bizonyítás Mivel SAT $\in$ **Np**, nyilvánvaló, hogy 3SAT $\in$ **NP** is teljesül. Megmutatjuk, hogy SAT $\leq_p$ 3SAT, ami egy korábbi tételünk alapján implikálja 3SAT **NP**-teljességét.

Legyen ϕ a SAT egy példánya. ϕ-hez megkonstruálunk egy olyan ψ formulát, melyben minden tag pontosan három literált tartalmaz, és $\phi\in$ SAT akkor és csak akkor áll fenn, ha $\psi\in$ 3SAT teljesül.

ϕ minden c tagját, az alábbi táblázat szerint, átalakítjuk ψ egy c' knf-ban lévő részformulájává:

c tag	c' knf
l	$l \vee l \vee l$
$l_1 \vee l_2$	$l_1 \vee l_2 \vee l_2$
$l_1 \vee l_2 \vee l_3$	$l_1 \vee l_2 \vee l_3$
$l_1 \vee l_2 \vee l_3 \vee l_4$	$(l_1 \vee l_2 \vee x) \wedge (\neg x \vee l_3 \vee l_4)$ $(x$ új változó$)$
$l_1 \vee l_2 \vee l_3 \vee l_4 \vee \ldots \vee l_n$ $(n > 4)$	$(l_1 \vee l_2 \vee x_1) \wedge (\neg x_1 \vee l_3 \vee x_2) \wedge$ $(\neg x_2 \vee l_4 \vee x_3) \wedge \ldots \wedge (\neg x_{n-3} \vee l_{n-1} \vee l_n)$ $(x_1, x_2, \ldots, x_{n-3}$ új változók$)$

Látható, hogy ψ konstrukciója elvégezhető egy polinom időigényű Turing-géppel, valamint $\phi \in$ SAT akkor és csak akkor, ha $\psi \in$ 3SAT. A fenti konstrukció tehát a SAT egy polinom idejű visszavezetése a 3SAT-ra, azaz SAT $\leq_p$ 3SAT. Következik, hogy 3SAT **NP**-teljes □

A 3SAT segítségével további problémákról mutatjuk meg, hogy **NP**-teljesek. Először a következő, majd mint látni fogjuk egymással szoros kapcsolatban álló, gráfelméleti problémákkal foglalkozunk: TELJES RÉSZGRÁF, FÜGGETLEN CSÚCSHALMAZ és CSÚCSLEFEDÉS. Most ezeket a problémákat definiáljuk.

Definíció

TELJES RÉSZGRÁF $= \{\langle G, k\rangle \mid G$ véges gráf, $k \geq 1, G$-nek létezik k csúcsú teljes részgráfja $\}$.

Tehát a TELJES RÉSZGRÁF azon G és k párokat tartalmazza, megfelelő ábécé feletti szavakban elkódolva, melyekre igaz, hogy G-ben van k csúcsú teljes részgráf, azaz olyan részgráf, melyben bármely két csúcs között van él.

FÜGGETLEN CSÚCSHALMAZ $= \{\langle G, k\rangle \mid G$ véges gráf, $k \geq 1, G$-nek van k elemű független csúcshalmaza $\}$.

Vagyis a FÜGGETLEN CSÚCSHALMAZ azon G és k párokat tartalmazza, melyekre igaz, hogy G-ben van k olyan csúcs, melyek közül egyik sincs összekötve a másikkal.

CSÚCSLEFEDÉS $= \{\langle G, k\rangle \mid G$ véges gráf, $k \geq 1, G$-nek van olyan k elemű csúcshalmaza, mely tartalmazza G minden élének legalább egy végpontját $\}$.

□

Ezek után megmutatjuk, hogy a fenti problémák **NP**-teljesek.

Tétel TELJES RÉSZGRÁF **NP**-teljes.

Bizonyítás Először is, TELJES RÉSZGRÁF $\in$ **NP**, hisz megadható egy nemdeterminisztikus Turing-gép, ami az egyik szalagjára írja a bemenetként kapott G gráf k darab csúcsát (azaz megsejt k darab csúcsot G-ből) és polinom időben leellenőrzi, hogy ezek a csúcsok teljes részgráfot alkotnak-e G-ben.

Másrészt visszavezetjük a 3SAT-ot TELJES RÉSZGRÁF-ra a következő módon. Legyen ϕ a 3SAT egy példánya. Akkor $\phi = c_1 \wedge \ldots \wedge c_k$, valamely $k \geq 0$-ra, ahol minden $0 \leq i \leq k$-ra $c_i = l_{i_1} \vee l_{i_2} \vee l_{i_3}$.

ϕ-hez megadunk egy G_ϕ gráfot az alábbi módon. ϕ minden c_i tagja meghatároz három csúcsot (azaz egy háromszöget) G-ben. Ezeket a csúcsokat l_{i_1}-el l_{i2}-vel és l_{i3}-mal jelöljük. Az így kapott csúcsok között az összes élet behúzzuk, kivéve az alábbiakat:

- az egy háromszögön belül lévő csúcsok közti éleket és
- az ellentétes literálokkal címkézett csúcsok közti éleket.

Nyilvánvaló, hogy G_ϕ megkonstruálható egy polinom időigényű Turing-géppel. Továbbá az is könnyen belátható, hogy $\phi \in$ 3SAT $\Leftrightarrow \langle G, k\rangle \in$ TELJES RÉSZGRÁF. Tehát a fenti konstrukció 3SAT polinom idejű visszavezetése TELJES RÉSZGRÁF-ra. Mivel 3SAT **NP**-teljes, kapjuk, hogy TELJES RÉSZGRÁF is **NP**-teljes. □

Most azt mutatjuk meg, hogy FÜGGETLEN CSÚCSHALMAZ is **NP**-teljes.

Tétel FÜGGETLEN CSÚCSHALMAZ **NP**-teljes.

Bizonyítás FÜGGETLEN CSÚCSHALMAZ $\in$ **NP**, hisz megadható egy nemdeterminisztikus Turing-gép, ami megsejt k darab csúcsot G-ben és polinom időben leellenőrzi, hogy ezek a csúcsok független csúcshalmazt alkotnak-e G-ben.

Ahhoz, hogy FÜGGETLEN CSÚCSHALMAZ **NP**-teljességét belássuk elegendő megmutatni, hogy TELJES RÉSZGRÁF $\leq_p$ FÜGGETLEN CSÚCSHALMAZ. Ehhez viszont elég észrevenni, hogy egy G gráfban akkor és csak akkor van k elemű teljes részgráf, ha G komplementer gráfjában ($\overline{G}$-ben) van k elemű független csúcshalmaz. $\overline{G}$ alatt azt a gráfot értjük, melynek ugyanazok a csúcsai, mint G-nek és két csúcs akkor és csak akkor alkot élet G-ben, ha nem alkot élt $\overline{G}$-ben. Világos, hogy $\overline{G}$ polinom időben megkonstruálható G-ből, és $\langle G, k\rangle \in$ TELJES RÉSZGRÁF $\Leftrightarrow \langle \overline{G}, k\rangle \in$ FÜGGETLEN CSÚCSHALMAZ. Tehát TELJES RÉSZGRÁF polinom időben visszavezethető FÜGGETLEN CSÚCSHALMAZ-ra. Mivel TELJES RÉSZGRÁF **NP**-teljes következik, hogy FÜGGETLEN CSÚCSHALMAZ is **NP**-teljes. □

Tétel CSÚCSLEFEDÉS **NP**-teljes.

Bizonyítás Először megint azt kell belátni, hogy CSÚCSLEFEDÉS $\in$ **NP**. Ez viszont igaz, hisz CSÚCSLEFEDÉS eldönthető egy olyan nemdeterminisztikus Turing-gépel ami megsejt k darab csúcsot G-ben és polinom időben leellenőrzi, hogy G minden éle rajta van-e a k csúcs valamelyikén.

A CSÚCSLEFEDÉS probléma **NP**-teljességét úgy bizonyítjuk, hogy visszavezetjük rá az **NP**-teljes FÜGGETLEN CSÚCSHALMAZ problémát. Tekintsünk egy n csúcsú ($n \geq 1$) véges G gráfot és annak egy k elemű ($k \leq n$) G' részgráfját. Nem nehéz belátni, hogy a G'-beli csúcsok pontosan akkor alkotnak egy k elemű független csúcshalmazt G-ben, ha G-nek a G'-n kívüli csúcsai egy $n-k$ elemű

csúcslefedést alkotnak G-ben. Tehát $\langle G, k\rangle \in$ Független csúcshalmaz $\Leftrightarrow$ $\langle G, n-k\rangle \in$ Csúcslefedés. Világos, hogy az $n-k$ szám polinom időben kiszámítható, vagyis Független csúcshalmaz $\leq_p$ Csúcslefedés. Következésképpen Csúcslefedés is **NP**-teljes. □

4.1.3. Hamilton-úttal kapcsolatos NP-teljes problémák

A következőkben azt szeretnénk megmutatni, hogy az Utazó ügynök probléma **NP**-teljes. Ehhez először megmutatjuk, hogy a Hamilton-út probléma **NP**-teljes, majd ezt felhasználva megmutatjuk, hogy az Irányítatlan Hamilton-út probléma is **NP**-teljes. Akkor mondjuk, hogy egy adot G irányított gráfban van Hamilton-út, ha van olyan út G-ben, ami minden csúcsot pontosan egyszer érint. Ezek után a Hamilton-út problémát az alábbi módon definiáljuk:

$$\text{Hamilton-út} = \{\langle G, s, t\rangle \mid G \text{ véges irányított gráf, } s, t \text{ csúcsok, } \exists s\text{-ből } t\text{-be Hamilton-út}\}.$$

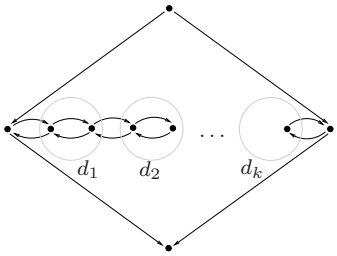
Tétel A Hamilton-út probléma **NP**-teljes.

Bizonyítás Az, hogy a probléma **NP**-beli könnyen belátható: megadható egy nemdeterminisztikus turing-gép, ami megsejti a bemenő gráf csúcsainak egy $s, \ldots, t$ felsorolását és polinom időben leellenőrzi, hogy a felsorolásban az egymást követő csúcsok között van-e él.

Hamilton-út **NP**-teljességének belátásához elegendő tehát megmutatni, hogy egy **NP**-teljes probléma visszavezethető rá. Ez a probléma a 3sat lesz. Legyen adott egy ϕ knf, melyben minden tag három literált tartalmaz. Polinom időben megkonstruálunk egy G_ϕ gráfot úgy, hogy G_ϕ két kitüntetett csúcsa között pontosan akkor lesz Hamilton-út, ha $\phi \in$ 3sat.

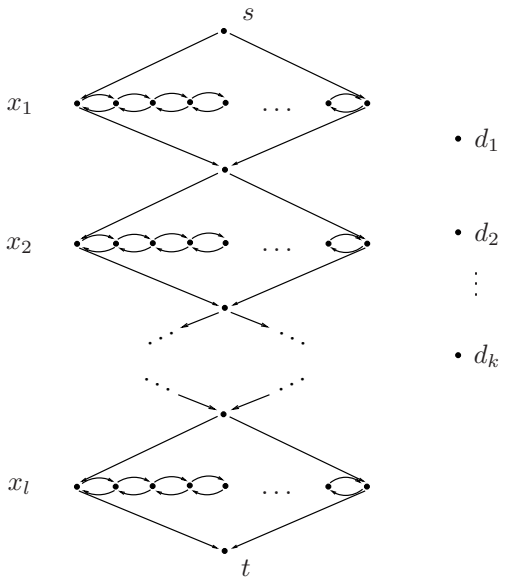
Legyen $\phi = d_1 \wedge \ldots \wedge d_k$, ahol $k \geq 0$ és minden i-re $(0 \leq i \leq k)$ $d_i = a_i \vee b_i \vee c_i$. Legyenek továbbá $x_1, x_2, \ldots, x_l$ $(l \geq 0)$ a ϕ-ben felhasznált változók.

Minden x_i változóhoz $(0 \leq i \leq l)$ lesz egy alábbi alakú részgráf G_ϕ-ben:

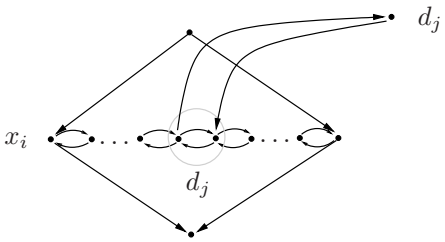


Ebben a részgráfban, a bal oldali csúcs melletti csúcstól kezdve, két-két egymás melleti csúcs rendre a ϕ egy-egy tagjának van megfeleltetve (az ábrán ezek a

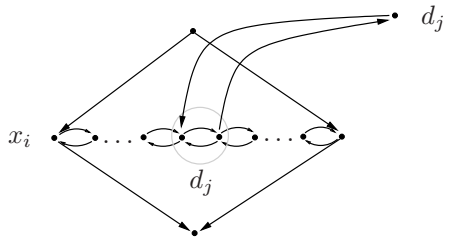
csúcspárok be vannak karikázva). Továbbá, minden d_j $(0 \leq j \leq k)$ taghoz lesz egy-egy külön csúcs G_ϕ-ben. Ezekből a részgráfokból felépítjük az alábbi gráfot:



Ebben a gráfban a változók által reprezentált eszközöket az alábbi stratégia szerint kötjük össze a tagokat reprezentáló csúcsokkal. Ha az x_i változó szerepel a d_j tagban, akkor az x_i-nek megfelelő részgráf d_j-nek megfelelő csúcsaiból vesszük balról jobbra haladva az első csúcsot és behúzunk egy élet ebből a csúcsból a d_j csúcsba. Ezután d_j-ből behúzunk egy élet a második csúcsba:



Ha a $\neg x_i$ literál szerepel a d_j tagban, akkor hasonlóan járunk el, de most az x_i-nek megfelelő részgráf d_j-nek megfelelő csúcsaiból vesszük balról jobbra haladva a második csúcsot és behúzunk egy élet ebből a csúcsból a d_j csúcsba. Ezután d_j-ből behúzunk egy élet az első csúcsba:



Legyen az így kapott gráf G_ϕ. Belátható, hogy $\langle G_\phi, s, t\rangle$ polinomidejű Turing-géppel elkészíthető $\langle\phi\rangle$-ből. Meg kell még mutatni, hogy $\langle\varphi\rangle \in$ 3SAT akkor és csak akkor teljesül, ha $\langle G_\varphi, s, t\rangle \in$ HAMILTON-ÚT.

Tegyük fel, hogy $\langle\phi\rangle \in$ 3SAT. Vegyük azt az $\mathcal{I}$ változóhozzárendelést, ami kielégíti ϕ-t. $\mathcal{I}$ meghatároz egy utat s-ből t-be a következő módon. Ha $\mathcal{I}$ egy x_i-hez ($0 \leq i \leq l$) igazat rendel, akkor az út az x_i-nek megfelelő „rombusz" felső csúcsából először balra megy, majd a rombusz bal oldalán lévő csúcsból jobbra lépked egészen addig, amíg eléri a jobb oldalon lévő csúcsot, végül pedig a rombusz alján lévő csúcsot érinti. Ha $\mathcal{I}$ az x_i-hez hamisat rendel, akkor az út az ellenkező irányban járja be az x_i-nek megfelelő részgráfot: a felső csúcsból először jobbra megy, majd a rombusz jobb oldalán lévő csúcsból balra lépked addig, amíg eléri a jobb oldalon lévő csúcsot, ahonnan pedig továbblép a rombusz alján lévő csúcsra.

Ezt a fent vázolt utat most kiegészítjük úgy, hogy a $d_1, \ldots, d_k$ tagoknak megfelelő csúcsokat is érintse. Tekintsük egy d_j tagot ($0 \leq j \leq k$), és vegyünk d_j-ből egy igaz literált (ilyen biztos, hogy létzik, hiszen $\mathcal{I}$ kielégíti a ϕ-t). Ez a literál x_i vagy $\neg x_i$ alakú valamely $0 \leq i \leq l$-re. Ha a literál x_i alakú, akkor az út balról jobbra halad az x_i-nek megfelelő rombuszban (hisz x értéke igaz). Ha a literál $\neg x_i$ alakú, akkor az út jobbról balra halad (mert az x_i értéke hamis). A d_j csúcs viszont úgy van hozzákötve az x_j-nek megfelelő rombuszban a d_j-nek megfelelő csúcspárhoz, hogy az utunk mindkét esetben képes megtenni egy kitérőt a d_j csúcshoz. Nem nehéz belátni, hogy az így vázolt út egy Hamilton-út a G_ϕ-ben s-ből t-be.

Tegyük fel most, hogy van egy Hamilton-út s-ből t-be a G_ϕ-ben. Nem nehéz belátni, hogy ez az út olyan, mint amilyet az előbb vázoltunk, azaz az út nem teheti meg azt, hogy egy x_i-nek megfelelő rombuszból kitérőt tesz egy d_j csúcshoz, de a d_j-ből nem ugyanabba a rombuszba megy vissza. Ellenkező esetben ugynis lennének olyan csúcsai G_ϕ-nek, amit az út nem lenne képes meglátogatni, ez viszont ellentmondana annak, hogy az út egy Hamilton-út. Így tehát a Hamilton-út meghatároz egy $\mathcal{I}$ változóhozzárendelést, ami pedig kielégíti ϕ-t. □

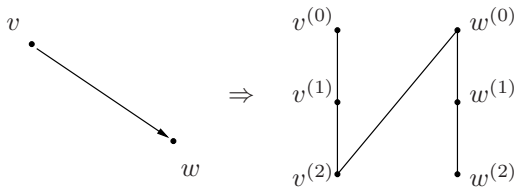
Most azt mutatjuk meg, hogy a HAMILTON-ÚT probléma akkor is **NP**-teljes marad, ha megengedjük azt, hogy a bemenetben szereplő gráf irányítatlan legyen.

Nevezzük az így kapott problémát IRÁNYÍTATLAN HAMILTON-ÚT-nak:

$$\text{IRÁNYÍTATLAN HAMILTON-ÚT} = \{\langle G, s, t\rangle \mid G \text{ véges irányítatlan gráf, s,t csúcsok, és } \exists s\text{-ből } t\text{-be Hamilton-út}\}.$$

Tétel Az IRÁNYÍTATLAN HAMILTON-ÚT probléma **NP**-teljes.

Bizonyítás Az, hogy a probléma **NP**-beli hasonlóan adódik, mint a HAMILTON-ÚT probléma esetében. Azt, hogy a probléma **NP**-teljes úgy bizonyítjuk, hogy visszavezetjük rá a HAMILTON-ÚT problémát. Adott $G = (V, E)$ irányított gráfhoz legyen $G_u = (V_u, E_u)$ a következő irányítatlan gráf. Minden $v \in V$ csúcsra legyen $v^{(0)}, v^{(1)}$ és $v^{(2)}$ egy-egy csúcs, $(v^{(0)}, v^{(1)})$ és $(v^{(1)}, v^{(2)})$ pedig egy-egy él G_u-ban. Továbbá, ha $(v, w) \in E$, akkor $(v^{(2)}, w^{(0)}) \in E_u$. Az alábbi ábra azt mutatja, hogy a (v, w) G-beli élnek milyen élek felelnek meg G_u-ban.



Világos, hogy G_u polinom időben megkonstruálható G-ből. Az is belátható, hogy $\langle G, s, t\rangle \in$ HAMILTON-ÚT akkor és csak akkor teljesül, ha $\langle G_u, s^{(0)}, t^{(2)}\rangle \in$ IRÁNYÍTATLAN HAMILTON-ÚT. Kapjuk tehát, hogy HAMILTON-ÚT $\leq_p$ IRÁNYÍTATLAN HAMILTON-ÚT, amit bizonyítani akartunk. □

Végezetül azt mutatjuk meg, hogy az úgynevezett utazóügynök probléma is **NP**-teljes. A problémát formálisan az alábbi módon definiálhatjuk:

$$\text{UTAZÓÜGYNÖK} = \{\langle G, k\rangle \mid G \text{ véges irányítatlan gráf az éleken egy-egy pozitív egész súllyal és } G\text{-ben van legfeljebb } k \text{ összsúlyú Hamilton kör}\}.$$

Tétel Az UTAZÓÜGYNÖK probléma **NP**-teljes.

Bizonyítás Azt a korábbiak alapján ismét nem nehéz belátni, hogy a probléma **NP**-beli. Az **NP**-teljességet úgy bizonyítjuk, hogy visszavezetjük rá az IRÁNYÍTATLAN HAMILTON-ÚT problémát. Ehhez legyen G egy irányítatlan gráf és s, t ennek két kitüntetett csúcsa. Konstruáljuk meg a G' gráfot az alábbi módon. G' az a gráf amit úgy kapunk G-ből, hogy G minden élének a súlya 1 lesz és felveszünk még egy élt 1 súllyal s-ből t-be. Továbbá, legyen k a G csúcsainak száma. Az így kapott G' gráfban pontosan akkor van k hosszú Hamilton-kör, amikor G-ben van Hamilton-út s-ből t-be. Mivel a G' konstrukciója polinom idejű, a tétel állítása bizonyított. □

Az **NP**-teljes problémák nagyon fontosak a bonyolultságelméletben. Általában ha egy új problémával találkozunk, akkor vagy azt próbáljuk megmutatni róla,

hogy **P**-beli vagy azt, hogy **NP**-teljes. Ha **P**-beli, akkor rendszerint hatékonyan megoldható a gyakorlatban is, ha **NP**-teljes, akkor viszont (valószínűleg) nincs a megoldására hatékony algoritmus és így fel is hagyhatunk annak keresésével. Azt már tudjuk, hogy $\mathbf{P} \subseteq \mathbf{NP}$. Elvileg lehetséges, hogy $\mathbf{P} = \mathbf{NP}$, de mint említettük az a sejtés, hogy ez az egyenlőség nem áll fenn. Azt is tudjuk, hogy az **NP**-teljes problémák a legnehezebbek az **NP**-beli nyelvek között. Vajon van-e olyan nyelv ami nem eleme **P**-nek, de nem is **NP**-teljes? Nyilvánvaló, hogy ha $\mathbf{P} = \mathbf{NP}$, akkor nincs ilyen nyelv, ellenkező esetben viszont:

Tétel Ha $\mathbf{P} \neq \mathbf{NP}$, akkor van olyan $L \in \mathbf{NP}$ nyelv, hogy $L \notin \mathbf{P}$, de L nem is **NP**-teljes.

Tekintsük például az alábbi problémát:

$$\text{Gráf izomorfizmus} = \{\langle G_1, G_2 \rangle \mid G_1 \text{ és } G_2 \text{ izomorf gráfok}\}.$$

Erről a problémáról ismert, hogy **NP**-ben van, de nem ismert, hogy **P**-ben van-e vagy **NP**-teljes-e. Ez a fentiek alapján szintén a $\mathbf{P} \subset \mathbf{NP}$ sejtést támasztja alá.