7.

Turing-gépek

Ha az emberi agy olyan egyszerű lenne, hogy megérthessük, akkor túl egyszerűek lennénk ahhoz, hogy ezt megtehessük. AMERIKAI FALFIRKA

Ebben a fejezetben elsődleges célunk az *algoritmus* fogalmának egy lehetséges pontosabb körülhatárolása. A fő eszközünk ebben egy egyszerű számítási modell, az Alan M. Turing[1] angol kutató által kidolgozott Turing-gép lesz. A modell a számítógép, a program egyszerűsített és bizonyos tekintetben idealizált változatának tekinthető. Egyszerűsége ellenére igen erőteljes konstrukció: minden olyan probléma, ami megoldható valamilyen algoritmus segítségével, kezelhetőnek bizonyult Turing-géppel is. A fejezet elején a Turing-gépekkel kapcsolatos egyszerű fogalmak, eredmények szerepelnek, amiket aztán később gyakran fogunk használni.

Sokáig úgy gondolták, hogy minden algoritmikus problémára van megoldás. Ha elég szívósan próbálkozunk, akkor előbb-utóbb legyűrjük a problémát. A harmincas években, főként A. Church, K. Gödel és A. Turing munkássága nyomán kiderült, hogy ez távolról sem igaz: vannak olyan feladatok, amelyekre nem létezik

[1] Alan Mathison Turing (1912-1954) egyike volt századunk legjelentősebb gondolkodóinak. Sikeresen foglalkozott matematikával, a számítások elméletével és gyakorlatával, de filozófiával és agykutatással is. Munkásságának egyik legnevezetesebb része a Turing-gép, és ehhez kapcsolódóan a kiszámíthatóság elméletének alapjai. Ezekről a fejezetben részletesen is lesz szó. A II. világháború alatt ő vezette azt a csoportot, amelyik feltörte a német tengeralattjárók által használt *Enigma*-kódot. Ennek óriási szerepe volt abban, hogy a szövetséges konvojok a normandiai invázióra készülve zavartalanul közlekedhettek az atlanti vizeken. A háború után egyik vezéralakja volt az első brit számítógép, az *ACE* tervezőgárdájának. Őt tekinthetjük a mesterséges intelligencia mint kutatási irány megalapozójának is. Ilyen tárgyú gondolatai – köztük a nevezetes *Turing-teszt* – ma is nagy hatásúak. Megemlítjük még, hogy a sokak által számítástudományi-számítástechnikai Nobel-díjnak tekintett kitüntetés neve: *ACM Turing Award*.

megoldó módszer. Ennek a kérdéskörnek, a *kiszámíthatóság elméletének* az elemei jelentik a következő témát, amelyet tárgyalni fogunk. Megismerkedünk néhány nevezetes algoritmikusan eldönthetetlen feladattal (Megállási probléma, Hilbert 10. problémája stb.).

Korábban már foglalkoztunk információtömörítő módszerekkel. A *Kolmogorov-bonyolultság* segítségével itt azt tanulmányozzuk, hogy a véges jelsorozatok mennyire nyomhatók össze valamilyen algoritmussal. Ki fog derülni, hogy vannak nem vagy csak alig összenyomható szavak, sőt, bizonyos értelemben az ilyen szavak a tipikusak. A módszer erejét alkalmazásokkal illusztráljuk.

A fejezet végén egy másik gépmodellel, a *közvetlen elérésű géppel* ismerkedünk meg. A Turing-gépnek mint elméleti eszköznek az erénye az egyszerűség. Ugyanezen tulajdonsága miatt viszont túl nehézkes ahhoz, hogy a segítségével igazi algoritmusokat írjunk le. Programozási eszközként nagyobb büntetést jelentene a a legprimitívebb gépi kódnál is. A közvetlen elérésű gép számítóerő dolgában, tehát a megoldható feladatok összességét tekintve egyenértékű a Turing-gépekkel. Másfelől a közvetlen elérésű gép sokkal jobban hasonlít az igazi számítógépek architektúrájára. Ezáltal módot ad arra, hogy szabatos, ugyanakkor gyakorlatias módon definiáljuk a programok időköltségét.

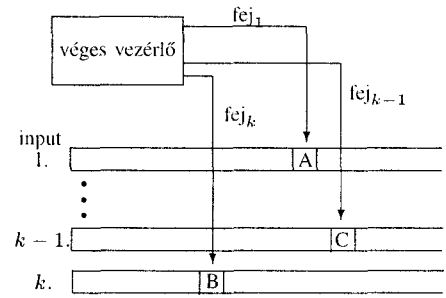
7.1. A Turing-gép fogalma

Hosszabb távú törekvésünk, hogy az *algoritmus* fogalmának egyik lehetséges pontos megközelítését adjuk. Ezt nagyjából úgy fogjuk tenni, hogy definiálunk egy számítási modellt, egy gépet, és azt tekintjük algoritmusnak, amit ezzel a géppel meg lehet valósítani. Ez a gép a *Turing-gép* lesz. A Turing-gép kevéssé hasonlít a ma használatos számítógépekre. Egy igen egyszerű központi részből és néhány *szalagegységből* áll. A központi rész, amit *véges vezérlőnek* is neveznek, a gép munkája során véges sok lehetséges *állapot* valamelyikében van. A szalagok cellákra vannak osztva. (Ezeket szokás még mezőknek, olykor négyzeteknek is nevezni.) Egy cellán a gépre jellemző rögzített véges jelkészlet, a *szalagjelek* halmazának egy eleme lehet. A szalagokat egyirányban végtelennek tekintjük. Ez az egyetlen sajátosság, amiben a Turing-gép erősebb a valóságos gépeknél. Mindegyik szalaghoz tartozik egy *író–olvasó fej*, amely írhatja/olvashatja a fej alatti mezőt. A fejek mindkét irányban léphetnek a szalagjuk mentén. A szalagok adják a Turing-gép adattárolási lehetőségeit; nincs külön belső memória és háttértár, mint az igazi számítógépeknél.

A gép bemenete egy szalagjelekből álló véges sorozat, ami a számítás megkezdésekor az egyik szalagra van írva. A gép számítása *lépések* egymásutánjából áll. A gép egy lépésében elolvassa az éppen a fejek alatt levő jeleket, majd ezektől

és a központi egység belső állapotától függően „cselekszik". Az egyik lehetőség, hogy megáll, azaz nem tesz semmit. A másik lehetőség, hogy egy-egy jelet ír a fejek alatti cellákba, majd egy mezővel jobbra vagy balra lépteti, esetleg helyben hagyja a fejeket, és végül a központi egység új állapotot vesz föl.

Ha a gép megállt, akkor megnézhetjük, hogy mi a számítás eredménye. A központi egység végső állapotát vagy pedig az egyik szalag tartalmát fogjuk eredményként értelmezni. Előfordulhat, hogy az adott bemenetre a gép sohasem áll meg. Ezt a jelenséget az igazi gépek világából vett, de itt a végtelen szalagok miatt kissé pongyola szóhasználattal *végtelen ciklusnak* is nevezik. A következő ábra a Turing-gép felépítését szemlélteti.



Ezek után ismertetjük a Turing-gép formális definícióját. Legyen k egy pozitív egész. Egy *k-szalagos Turing-gép* egy hetessel jellemezhető:

$$M = (Q, T, ü, I, q_0, F, \delta),$$

ahol

Q : egy véges halmaz, az M gép *belső állapotainak* halmaza. A gép a működése során mindig valamelyik $q \in Q$ állapotban van.

T : egy véges halmaz, a *szalagjelek* halmaza. Egy szalagmezőn mindig egy T-beli jel van.

$ü$: egy kitüntetett szalagjel, az *üresjel*. A bemenetként felírt jeleken és a gép számítása során kiírt jeleken kívül minden szalagcellában $ü$ van. Az üresjelet tekinthetjük úgy, mint a még nem használt mezők tartalmát.

I : $I \subseteq T \setminus \{ü\}$ az *input jelek* vagy *bemenő jelek* halmaza, más szóval az input *abc*. A gépnek adott bemenetet az I elemeiből kell összeállítani. Az üresjel nem lehet input jel.

q_0 : $q_0 \in Q$ a *kezdő állapot*. A gép a munka megkezdése előtt a q_0 állapotban van.

F : $F \subseteq Q$ az *elfogadó állapotok* halmaza. Bizonyos számításoknál csak egy kétértékű döntést várunk a géptől (pl. az igen-nem választ igénylő feladatok esetében). Ekkor csak azt nézzük, hogy a gép milyen állapotban állt meg. Az egyik választ az jelenti, hogy ez a végállapot F-beli, a másikat pedig az, hogy nem F-beli. Ezzel összhangban a $Q \setminus F$-be tartozó állapotokat *elutasító állapotoknak* is nevezik.

δ : A $\delta : Q \times T^k \longrightarrow Q \times (T \times \{jobb, bal, helyben\})^k$ egy parciálisan értelmezett függvény, a gép *átmenetfüggvénye*. Az átmenetfüggvény tekinthető a gép programjának. A δ függvény mondja meg, hogy ha a gép a q állapotban van, és az i-edik fej alatt levő szalagjel a_i $(1 \leq i \leq k)$, akkor mit kell lépni. Ennek megfelelően a $\delta(q; a_1, a_2, \ldots, a_k)$ nincs értelmezve, ha ez a lépés a megállás. A többi esetben a $\delta(q; a_1, a_2, \ldots, a_k)$ érték $2k + 1$ összetevőből áll. Ezek: a gépnek a lépés utáni $q' \in Q$ állapota; továbbá az i-edik szalagra kiírt b_i szalagjel, valamint az i-edik fej elmozdulása $(1 \leq i \leq k)$. Az elmozdulás legfeljebb mezőnyi, ezért leírható a *jobb*, *bal* és *helyben* lehetőségek egyikeként. A δ függvényről még feltesszük, hogy az F-beli állapotokon nincs értelmezve.

A δ parciális függvény lényegében egy (nem mindenütt kitöltött) *táblázatként* fogható fel. Ez a táblázat írja le a gép programját. A Turing-gép a korábbiaknál kicsit pontosabban olyan gépnek tekinthető, amely egyetlen program futtatására képes.

Az I input *abc* elemeiből álló véges sorozatokat *szavaknak* nevezzük. Az I jeleiből (betűiből) alkotott szavak halmazát I^*-gal jelöljük. I^* részhalmazait *nyelveknek* nevezzük. Egy $L \subseteq I^*$ nyelv elemeit az L nyelv szavainak nevezzük.

A Turing-gép első szalagját *input szalagnak* is nevezzük, mivel azon lehet a bemenetet megadni. Bizonyos esetekben hasznos, ha kijelölünk egy *output szalagot* is. Ez annyit tesz, hogy a gép megállása után az eredményt ennek a szalagnak az elejéről olvassuk le. Az output szalag megadása nem több, mint a végeredmény megjelenési helyére vonatkozó előírás.

A Turing-gép működése

A gép a működés előtt *kezdőhelyzetben* van. Ez azt jelenti, hogy a gép állapota a q_0 kezdőállapot, az író-olvasó fejek a szalagjaik első cellájára mutatnak, az $s \in I^*$ bemenet az első szalag elejére van írva, az összes többi mezőn az $\ddot{u}$ üresjel található.

A gép a kezdőhelyzetből indulva lépéseket tesz. A lépések tartalmát jelentő változásokat a δ függvény írja le: a lépés előtti állapotnak és a fejek alatti jeleknek megfelelő δ-érték adja meg az új állapotot, a kiírandó jeleket és a fejmozgásokat. Ha a gép egy olyan helyzethez ér, amelyre δ nincs értelmezve, akkor megáll. Ez történik egyebek között akkor, ha a gép elfogadó (azaz F-beli) állapotban van.

A Turing-gép számításának eredménye

A Turing-gépeket kétféle üzemmódban fogjuk tekinteni. Egyfelől *nyelvek felismerésére*, másfelől *függvények kiszámításra* használjuk őket. Az utóbbi típusú feladatok eléggé természetesek a számítástechnikában, a velük való foglalkozás így nem igényel különösebb indoklást. Ami a nyelvek felismerését illeti, ezek olyan feladatoknak tekinthetők, ahol a számítás eredménye egyetlen bit: egyetlen eldöntendő kérdésre kell választ adni. Az ilyen feladatok elméleti szempontból egyszerűbben megfoghatók, ugyanakkor elég gazdag a struktúrájuk ahhoz, hogy segítségükkel a számítások általános tulajdonságairól képet kaphassunk.

Definíció (Turing-gép által felismert nyelv):
Az M Turing-gép által felismert L_M nyelv *azokból az $s \in I^*$ szavakból áll, amelyekkel mint bemenetekkel elindítva az M megáll, mégpedig elfogadó (azaz F-beli) állapotban.*

Ez a fogalom tulajdonképpen az *igen–nem kérdések* eldöntésének felel meg: képzeljük el, hogy van egy általános kérdésünk, ami az I^* minden egyes szavára értelmes, és a válasz egy szó esetén igen vagy nem. Például a kérdés lehet az, hogy az $s \in I^*$ szó tartalmazza-e a 0 jelet. Egy ilyen kérdésnél azok a szavak, amelyekre a válasz igenlő – az ún. igenpéldányok –, egy $L \subseteq I^*$ nyelvet alkotnak. Ha $L = L_M$, vagyis az igenpéldányokból álló nyelv éppen az M által felismert nyelv, akkor úgy tekinthetjük, hogy az M gép gyenge értelemben eldönti a kérdést. Ha $s \in I^*$ igenpéldány, akkor ezt M véges sok lépésben megállapítja azzal, hogy megáll elfogadva s-et. Ha s nem egy igenpéldány, akkor cizelláltabb a kép. Lehet, hogy M ekkor is megáll, szükségképpen elutasító állapotban, amit úgy értelmezhetünk, hogy ebben az esetben is sikerült megválaszolnia a kérdést. A definíció megengedi viszont azt is – és ez a *gyenge* magyarázata –, hogy a gép $s \notin L_M$ esetén sose álljon meg. Az M tehát általában véve egy fél algoritmusnak tekinthető az L-nek megfelelő kérdés megválaszolására. Az L_M szavaira jól működik, az $I^* \setminus L_M$ szavainál viszont végtelen ciklusba keveredhet.

Definíció (Turing-gép által kiszámított függvény):
Legyen M egy kitüntetett output szalaggal rendelkező Turing-gép. Az M által kiszámított $f_M : I^* \to I^*$ parciális függvényt *így értelmezzük: $f_M(s) = w$, ha M*

az $s \in I^*$ inputtal indulva megáll, és megállás után az output szalagon a $w \in I^*$ szó szerepel az üresjelek óceánja előtt.

Az f_M függvény tehát csak azokra az $s \in I^*$ szavakra értelmezett, amelyekkel mint bemenetekkel az M gép véges sok lépés megtétele után megáll. Ennél a számítási módnál közömbös, hogy a megállás elfogadó vagy elutasító állapotban történt-e. Az $f_M(s)$ eredmény pedig a kitüntetett szalag legnagyobb kezdődarabja, ami csupa I-beli betűből áll.

Példák: 1. A következő igen egyszerű egyszalagos M gép bizonyos értelemben a hárommal való oszthatóságot dönti el. Legyen tehát $k = 1$, és

$$Q = \{q_0, q_1, q_2, q_v\},\ T = \{0, 1, ü\},\ I = \{0, 1\},\ F = \{q_v\}.$$

A gépnek négy állapota van, ebből egy elfogadó állapot. A bemenetet a 0 és 1 jelekből lehet összeállítani. Mivel M-nek egy szalagja van, a δ függvény *(állapot, szalagjel)* alakú párokhoz rendel *(állapot, szalagjel, fejmozgás)* összetételű hármasokat. Legyen ezek után δ a következő:

$$\begin{aligned} \delta(q_0, 1) &= (q_1, 1, jobb),\ \ \delta(q_0, ü) = (q_v, ü, helyben), \\ \delta(q_1, 1) &= (q_2, 1, jobb),\ \ \delta(q_2, 1) = (q_0, 1, jobb). \end{aligned}$$

Tegyük még fel, hogy más párokra δ nincs értelmezve. Az M átmeneteit nézve megállapíthatjuk, hogy ha 0 jelet olvas, akkor megáll elutasító, azaz nem elfogadó állapotban. Nincs ugyanis a δ értelmezési tartományában olyan pár, ami a 0 szalagjelet tartalmazza. Ugyanez a helyzet az $ü$ üresjellel is, kivéve, ha az állapot q_0, mely esetben M a fejet helyben hagyva átmegy a q_v elfogadó állapotba. Érdekesebb, ami egyes olvasásakor történik. Ha a gép állapota ekkor q_i, akkor a gép érdemi írás nélkül (ugyanazt írja vissza amit olvasott) jobbra lép, és átmegy a q_{i+1} állapotba; itt az összeadás modulo 3 értendő. Figyelembe véve még azt is, hogy M a q_0 állapotból indul, mindezekből arra jutunk, hogy M pontosan azokat a szavakat fogadja el, amelyek csupa egyesekből állnak, és a hosszuk osztható hárommal. Az M által felismert L_M nyelv tehát

$$L_M = \{1^n :\ n \text{ hárommal osztható természetes szám}\}.$$

Itt 1^n az n darab egyesből álló szó jelölése; speciálisan $1^0 = ü$.

2. Az itt szereplő egyszalagos M' gépet mindkét üzemmódjában megnézzük. Meghatározzuk az $L_{M'}$ nyelvet és az $f_{M'}$ függvényt is. Utóbbihoz output szalag is kell. Nincs sok választásunk: az egy szem szalagot használjuk erre a célra is. Legyen

$$Q = \{q_0, q_v\},\ T = \{0, 1, ü\},\ I = \{0, 1\},\ F = \{q_v\}$$

és

$$\delta(q_0, 0) = (q_0, 0, helyben),\ \delta(q_0, 1) = (q_0, 1, jobb),\ \delta(q_0, ü) = (q_v, 1, helyben).$$

Másutt δ nem definiált. Az M' a q_0 belső állapotban maradva lépdel jobbra a szalag mentén, amíg 0 vagy *ü* jelet nem talál. Az előbbi esetben végtelen ciklusba kerül, az utóbbiban pedig még egy 1-est ír az input után, majd megáll elfogadó állapotban. A gép tehát a csupa egyesekből álló szavakat fogadja el: $L_{M'} = \{1^n; n \geq 0 \text{ egész}\}$. Az M' csak az $L_{M'}$ szavaira áll meg, így ez a nyelv az $f_{M'}$ parciális függvény értelmezési tartománya. A gép az 1^n bemeneten az 1^{n+1} eredményt adja, vagyis $f_{M'}(1^n) = 1^{n+1}$. Az M' gépet tekinthetjük az $f(n) := n + 1$ függvényt kiszámító algoritmusnak.

Feladat: Az 1. példabeli M gép az unárisan ábrázolt n számról eldönti, hogy az hárommal osztható-e. Szerkesszünk olyan N Turing-gépet, amelyik a *binárisan* megadott n bemenettel teszi ugyanezt.

Talán az előző példák hihetővé teszik, hogy tényleg lehet számításokat végezni Turing-gépekkel. A tényleges gépek és a Turing-gépek számítóereje közötti kapcsolat további hangsúlyozásául megjegyezzük, hogy utóbbiak leírhatók programmal. Egy lehetséges megoldás, hogy helyet foglalunk a szalagok véges darabjainak. Ezeken nyilvántartjuk a fejek helyzetét; legyenek ezek az $f_1, f_2, \ldots, f_k$ változókban. A gép mindenkori belső állapotát pedig a q változóban tároljuk. Ezután egy végtelen ciklust szervezünk, aminek magjában a következők szerepelnek:

- Az olvasófejek alatti szalagjelek beolvasása az $x_1, \ldots, x_k$ változókba.

- a δ táblázat minden bejegyzésére egy programrészlet a következő minta szerint: a $\delta(q_i; a_1, a_2 \ldots, a_k) = (q_j; (b_1, bal), (b_2, helyben), \ldots, (b_k, jobb))$ érték megfelelője legyen

  if $q = q_i$ **and** $x_1 = a_1$ **and** $x_2 = a_2$ **and** $\ldots$ **and** $x_k = a_k$ **then begin**
  $q := q_j$;
  $x_1 := b_1$; $x_2 := b_2$; $\ldots$; $x_k := b_k$;
  Az x_i tartalmának kiírása az f_i változókkal azonosított helyekre
  $(i = 1, 2, \ldots, k)$;
  $f_1 := f_1 - 1$; $\ldots$; $f_k := f_k + 1$;
  end;

- Megállás abban az esetben, ha δ az adott helyen nincs értelmezve.

Az irodalomban a Turing-gépeknek többféle, a részletekben kisebb-nagyobb eltéréseket mutató meghatározásával találkozhatunk. Ilyen eltérés lehet, hogy a szalagokat mindkét irányban végtelennek vesszük. Az eltérések azonban elméleti szempontból nem jelentősek; a különböző géptípusok egyenértékűsége némi munkával igazolható. Könnyen átalakítható például egy Turing-gép ugyanazt a nyelvet felismerő, de csak egyetlen elfogadó állapottal rendelkező géppé. Ennek meggondolását az olvasóra bízzuk.

7.2. Idő- és tárigény

A Turing-gép mint modell egyik erénye, hogy segítségével elég egyszerűen vizsgálhatjuk a számítások idő- és tárigényét. Az M Turing-gép *számolási ideje* az s inputon a megállásáig végrehajtott lépések száma, *tárigénye* pedig a felhasznált (olvasott) szalagcellák száma. Ha a gép input szalagja csak olvasható, akkor előírhatjuk, hogy ez a szalag ne számítson bele a tárigénybe. Hasonló a helyzet a csak írásra szolgáló output szalaggal (amin sosem léphetünk balfelé). Az a megfontolás áll emögött, hogy kizárólag az érdemi munka helyigényét mérjük, és ne foglalkozzunk a bemenet olvasásának és az eredmény írásának (elkerülhetetlenül fellépő) költségével. Úgy is fogalmazhatunk, hogy csupán a *munkaszalagokon*, az írásra és olvasásra is használatos szalagokon felhasznált hely számít.

A hárommal való oszthatóságot eldöntő M gép számolási ideje az 1^n bemeneten $n+1$. A tárigény függ attól, hogy miként szemléljük a szalagot. Tekinthetjük csak olvasható input szalagnak, hiszen a gép sohasem írja (ami azt jelenti, hogy mindig a régi értéket írja a fej alatti mezőbe). Ekkor a tárigény minden bemeneten 0. Ha viszont a szalagot munkaszalagnak vesszük, akkor az 1^n bemeneten a tárigény $n+1$ lesz. Az $f(n) := n+1$ függvényt kiszámító M' gép számolási ideje az 1^n szóra $n+1$, az 110 szóra pedig végtelen, hiszen a 0 olvasása után M' végtelen ciklusba esik. A szalagot tekinthetjük csak írható output szalagnak, mert a gép sohasem lép balra. Ekkor a tárigény minden bemeneten 0. Ha viszont munkaszalagként szemléljük, akkor 1^n-nél a tárigény $n+1$, az 110 bemeneten pedig 3.

Szeretnénk beszélni az algoritmusok (Turing-gépek) sebességéről, illetve tárfelhasználás szerinti hatékonyságáról. Ebben túlságosan elaprózott megközelítés lenne, ha minden $s \in I^*$ szóra külön vizsgálnánk a lépésszámot vagy a tárigényt. Ennél egyszerűbb, de még mindig elég informatív megoldást kapunk, ha a bemenet hosszának függvényében vizsgáljuk ezeket az igényeket.

Jelölje $T_M(n)$ az M gép maximális számolási idejét az n jelből álló bemeneteken. Az n hosszú szavakon a maximális tárigényt $S_M(n)$-nel jelöljük.

Ha van olyan n jelből álló $s \in I^*$ szó, amellyel elindítva M nem áll meg véges sok lépés után, akkor $T_M(n)$ értékét végtelennek tekintjük. Hasonló mondható az S_M függvényről is.

A hárommal való oszthatóságot tesztelő M gépre $T_M(n) = n + 1$, ha pedig a szalagját munkaszalagnak vesszük, akkor $S_M(n) = n + 1$. Ami az M' gépet illeti, $T_{M'}(n) = \infty$, ha $n > 0$, hiszen a nullát tartalmazó bemeneteken a gép nem áll meg. Ha M' szalagja munkaszalag, akkor $S_{M'}(n) = n + 1$.

A T_M és S_M függvények segítségével lehetőség nyílik arra, hogy algoritmusokat hatékonyság szempontjából összehasonlítsunk. Ha például M és N két Turing-gép, melyekre $T_M(n) < T_N(n)$ teljesül minden elég nagy n-re, akkor az M algoritmust gyorsabbnak mondhatjuk az N algoritmusnál. Ily módon beszélhetünk arról, hogy egy adott feladatot megoldó két módszer közül az egyik hatékonyabb, mint a másik.

Algoritmikus problémák megoldása során igyekszünk minél jobb módszereket találni. Ha például az L nyelv időben hatékony felismerése a feladat, akkor olyan M algoritmust keresünk, amelyre a $T_M(n)$ függvény elég lassan nő. Természetes törekvésnek tűnhet, hogy ilyen értelemben a legjobb módszert keressük. Legjobb módszer azonban nem feltétlenül létezik, amint azt a következő tétel mutatja.

Tétel (gyorsítási tétel): *Van olyan L nyelv, amelyre igazak az alábbiak:*
1. Az L felismerhető egy olyan M Turing-géppel, melyre $T_M(n)$ véges minden n-re.
2. Tetszőleges, az L-et felismerő N Turing-géphez van olyan N' Turing-gép, amelyre $L = L_{N'}$ szintén teljesül, továbbá $T_{N'}(n) = O(\log T_N(n))$.

□

A bizonyítást mellőzzük. A tételbeli L nyelvet felismerő bármely algoritmus exponenciálisan felgyorsítható. Ezért nem létezhet a feladatra leggyorsabb algoritmus. A feladatok idő- és tárigényének értelmezésében tehát óvatosabban kell eljárni. A következő fejezet elején visszatérünk ehhez a gondolatkörhöz.

7.3. Néhány szimuláció

Ebben a szakaszban néhány olyan tényt ismertetünk, amelyek azt mondják, hogy bizonyos típusú Turing-gépek helyettesíthetők, *szimulálhatók* másfélékkel. A szimuláló gép egyenértékű az eredetivel abban az értelemben, hogy ugyanazt a nyelvet ismeri fel, illetve ugyanazt a függvényt számítja ki. Bemelegítőnek egy feladatot ajánlunk:

Feladat: Legyen M egyszalagos Turing-gép. Módosítsuk M-et úgy, hogy a kapott M' gép egyenértékű legyen M-mel, azaz $L_M = L_{M'}$, $f_M = f_{M'}$ teljesüljön;

továbbá az M' gépnek pusztán az aktuális állapotából kiderüljön, hogy mi volt a szalagjel, amit utoljára olvasott. (Az M' gép állapotai legyenek q_0 és a (q, a) alakú párok, ahol q_0, q az M állapotai, a pedig az M szalagjele. Az M gép δ átmenetfüggvényét terjesszük ki a párokra úgy, hogy az első komponensnek megfelelően utánozzuk M lépését, a második komponens pedig az éppen olvasott szalagjelnek feleljen meg.)

A feladattal azt szerettük volna érzékeltetni, hogy a belső állapotok segítségével fenntarthatunk a gépben egy rögzített méretű memóriát. Hasonló megoldással tárolhattuk volna mondjuk a fej helyzetének modulo 5 osztási maradékát vagy bármi más korlátos mennyiségű információt.

A szimulációs eredmények egy része olyasmit mond, hogy az egyszerűbb gépek viszonylag kézben tartható hatékonyságvesztéssel meg tudják tenni ugyanazt, amit a bonyolultabbak. A következő eredmény szerint az, amit el lehet végezni sok szalaggal, véghezvihető egyetlen szalaggal is. A tétel gyakran ad módot arra, hogy csak egyszalagos gépekre korlátozzuk a figyelmünket.

Tétel: *Legyen M egy k-szalagos Turing-gép. Van olyan egyszalagos M' Turing-gép, melyre*

$$L_M = L_{M'} \ (\textit{vagy } f_M = f_{M'}), \textit{ továbbá}$$
$$T_{M'}(n) \leq 2T_M^2(n),$$
$$S_{M'}(n) \leq S_M(n) + n.$$

Bizonyítás: M' építésekor az M-hez képest alaposan felfújjuk a szalag abc-t, és megnöveljük a belső állapotok számát is. Az M' egyetlen szalagján $2k$ csík lesz. A $2i-1$-edik csík felel meg az M i-edik szalagjának, míg a $2i$-edik csíkban az i-edik fej helyén egy megkülönböztető jel, x szerepel. Az M' szalagjának egy mezején k eredeti szalagjelet és k darab fej-helyzetről tájékoztató jelet kódolunk.

1. szalag	D	$\cdots$	A	$\cdots$	B	$\cdots$
1. fej	ü	$\cdots$	x	$\cdots$	ü	$\cdots$
.						
.						
.						
k. szalag	D	$\cdots$	D	$\cdots$	B	$\cdots$
k. fej	ü	$\cdots$	ü	$\cdots$	x	$\cdots$

A rajzon egy oszlop felel meg az M' egyetlen szalagmezejének. Az ábrázolt helyzet az M szalagjainak azt az állapotát tükrözi, amikor az első és a k-adik szalag első mezején is D van, az első fej alatt A, a k-adik fej alatt pedig B olvasható, stb. Az M'-nek összesen $(2t)^k$ szalagjele van, ahol t az M szalagjeleinek a száma.

M' az M gép egy lépését egy legfeljebb $2T_M(n)$ lépésből álló menetben utánozza. Először a szalag elejéről indulva jobbra elmegy a legmesszebb levő x jelig, és közben leolvassa az x-ek felett található eredeti szalagjeleket. Ezeket a belső állapotaiban tárolja. (Ezt a feladatban javasolt módszerrel teheti.) A menetelés során egy hellyel jobbra mozdítja az útjába eső x-eket, kivéve az utolsó oszlopban levő(ke)t. Ebben a helyzetben a gép ismeri M állapotát és a fejei alatti jeleket, így meghatározhatja a M következő állapotát és a kiírandó jeleket.

Ezután M' legfeljebb egyet lép jobb felé, majd egyenesen visszabaktat a szalag elejére. Eközben kiírja az M fejeinek régi helyére a k darab jelet, amiket M írt volna, és az M fejmozgásainak megfelelően áthelyezi az x-eket. Utóbbiaknál hasznos, hogy korábban jobbra léptettük ezeket a jeleket. Így mindezek megoldhatók úgy is, hogy csak balfelé megyünk.

Ha n jelből álló bemenettel kezdjük a munkát, akkor egy meneteben az M' feje legfeljebb $T_M(n)$ lépést tesz jobbra, következésképpen legfeljebb ugyanennyit mehet balra. Az M egy lépését így nem több, mint $2T_M(n)$ lépéssel szimuláltuk. Az M' pontosan akkor álljon meg (fogadja el a bemenetet), ha M ezt teszi. Így a számolási idő $T_{M'}(n) \leq 2T_M(n)T_M(n) = 2T_M^2(n)$.

Ha függvényt számítunk, akkor az utolsó lépés visszatérő fázisában az M output szalagjának megfelelő csíkot kivéve mindenhova üresjelet írunk. Tesszük mindezt azért, hogy az eredmény az M bemenő betűivel legyen írva.

A tárra vonatkozó állításból ezután csak a $+n$ tag szorul némi magyarázatra. Ez azért szerepel, mert ha M-nek kitüntetett input szalagja volt, akkor a bemenet hosszát, ami n, nem számítottuk bele $S_M(n)$-be.

□

A következő tétel szerint a szimuláció időveszteségе kisebb lesz, ha egy helyett két szalagot engedünk meg. A bizonyítás az előbbinél bonyolultabb; nem részletezzük.

Tétel: *Az M k-szalagos Turing-géphez megadható olyan 2-szalagos M' Turing-gép, amely az előbbi értelemben szimulálja M-et,*

$$T_{M'}(n) \leq O(T_M(n) \log T_M(n)), \text{ és}$$

$$S_{M'}(n) \leq S_M(n) + n.$$

□

A most terítékre kerülő tény azt sugallja, hogy az algoritmusok időigényét sok esetben legfeljebb csak állandó szorzó erejéig érdemes nézni. A Turing-gép alkalmas megválasztásával ugyanis a multiplikatív konstans majdnem tetszőlegesen lefaragható.

Tétel: *Tegyük fel, hogy az M Turing-gép az L nyelvet ismeri fel, és $T_M(n) \leq cn$ teljesül egy $c > 0$ állandóval. Ekkor tetszőleges $\epsilon > 0$-ra van olyan L-et felismerő M' Turing-gép is, hogy alkalmas $n_0 \in \mathbb{N}$ számmal*

$$T_{M'}(n) \leq n(1+\epsilon), \text{ ha } n \geq n_0.$$

Bizonyítás: Az M' gépet úgy tervezzük, hogy az M sok egymás utáni lépését kevés lépésben utánozni tudja. Pontosabban egy kellően nagynak választott m számmal az fog teljesülni, hogy a régi gép m lépését az új legfeljebb 7 lépésben elvégzi. Osszuk fel evégből az M szalagjait m egymás utáni mezőből álló blokkokra. Egy ilyen blokk tartalmát az M' egyetlen szalagjelben tárolja. Eszerint, ha M-nek t szalagjele van, akkor az új gép t^m betűt használ. Az M'-nek eggyel több szalagja lesz, mint M-nek. Az elsőn, amit csak olvas, az M bemenete szerepel. Ennek tartalmát az érdemi munka előtt átkódolja egy másik szalagra, utána a régi bemenő szalaggal nem foglalkozik. A többi szalag – tömörebb kódolással – ugyanazt fogja tartalmazni, mint az M megfelelő szalagjai.

Nézzük most, hogy mi történik a szalagokkal az M gép m egymást követő lépése során! A lényeges észrevétel, hogy ami ezalatt végbemegy, az csak a fejeket tartalmazó blokkoktól és azok közvetlen szomszédaitól függ, hiszen a fejek legfeljebb m cellányira mozdulhatnak el. Változások is csak eme blokkokban lehetségesek.

M' tehát nagy vonalakban a következőket teszi: szalagonként három szomszédos jel (ami három blokkot jelent M-nél) megvizsgálása után M átmeneteinek ismeretében a „memóriájában" meglépi M következő m lépését. Ezután az eredménynek megfelelően felülírja a szomszédos mezőhármasokat, majd helyükre teszi a fejeket. A szükséges memória a feladatban vázolt eljárással képezhető.

Ezek szerint az m lépés szimulációjához a szalagok olvasása+írása elvégezhető egy bal–jobb–jobb–bal–bal lépéssorozatban (a szalagok elején ennél rövidebben is). Legfeljebb további két jobbralépés szükséges lehet, amikor az M helyzetének megfelelő blokkokra állítjuk az M' fejeit. Így M' legfeljebb 7 lépésben elvégzi az M gép m lépését. A bemenet átkódolása, majd a kódolt szalagon a fejnek a szalag elejére mozgatása összesen $n + \lceil \frac{n}{m} \rceil$ lépésben megtehető. Az M' lépéseinek számát így becsülhetjük:

$$T_{M'}(n) \leq n + \left\lceil \frac{n}{m} \right\rceil + 7 \left\lceil \frac{T_M(n)}{m} \right\rceil \leq n + \frac{n}{m} + \frac{7T(n)}{m} + 8 \leq$$

$$\leq n + \frac{n}{m} + \frac{7cn}{m} + \frac{8n}{n} \leq n(1 + \frac{1}{m} + \frac{7c}{m} + \frac{8}{n}) \leq n(1+\epsilon),$$

ha m és n olyan nagyok, hogy $\frac{1}{m} + \frac{7c}{m} + \frac{8}{n} < \epsilon$. □

Hasonló érveléssel megmutatható, hogy ha $\liminf_{n\to\infty} \frac{T_M(n)}{n} = \infty$, akkor M' választható úgy, hogy tetszőleges $c > 0$-ra és elég nagy n-re $T_{M'}(n) \leq cT_M(n)$ teljesüljön.

A szimulációs eredményeket azon az áron értük el, hogy mind a jelek számát, mind a belső állapotok számát megnöveltük. Számítógépes hasonlattal élve: komolyabb, bonyolultabb „hardvert" használtunk. Nagyobb (több állapotú) a központi egység, és nagyobb a jelhalmaz is. A fordított irányt illetően megjegyezzük, hogy – konstansszoros hatékonyságvesztés árán – a szalagjelek száma a szokásos bináris kódolással 2-re csökkenthető.

Megemlítjük még, hogy a legutóbbi tétel huncutság abban az értelemben, hogy a szalagjelek számának korlátlan növelhetőségére építettük a bizonyítást. Az érdekes tanulság ebből az, hogy a Turing-gépmodellben a feladatok időigénye függ a jelkészlet méretétől.

7.4. A kiszámíthatóság alapfogalmai

Ebben és a következő néhány részben az algoritmussal való kiszámíthatóság határait feszegetjük. Hogy erről értelmesen beszélni tudjunk, pontosan meg kell határoznunk az algoritmus fogalmát. Azt fogjuk algoritmikusan kiszámíthatónak tekinteni, ami Turing-géppel kiszámítható. A továbbiakban legyen I egy nem üres véges halmaz. Olyan gépekkel fogunk foglalkozni, melyeknek az input *abc*-je I.

Definíció (rekurzíve felsorolható nyelv): *Az $L \subseteq I^*$ nyelvet* rekurzíve felsorolható*nak nevezzük, ha van olyan M Turing-gép, melyre $L = L_M$, azaz a gép által felismert nyelv éppen L.*

Ahogy már korábban említettük, ez annyit tesz, hogy van egy fél algoritmusunk az L (illetve a neki megfelelő igen-nem kérdés) eldöntésére. Megeshet ugyanis, hogy egy $s \in I^* \setminus L$ szóval elindítva az M nem áll meg. A végtelen ciklusok kizárásával kapjuk a következő fogalmat:

Definíció (rekurzív nyelv): *Az $L \subseteq I^*$ nyelv* rekurzív, *ha van olyan M Turing-gép, melyre $L = L_M$, és M minden $s \in I^*$ szóra megáll.*

Nyilvánvaló, hogy egy rekurzív nyelv rekurzíve felsorolható is. A rekurzív, illetve rekurzíve felsorolható nyelvek halmazára az $\mathcal{R}$, illetve $\mathcal{RE}$ jelölést használjuk:

$$\mathcal{RE} = \{L \subseteq I^* : \ L \text{ rekurzíve felsorolható}\}.$$

$$\mathcal{R} = \{L \subseteq I^* : \ L \text{ rekurzív}\}.$$

Analóg fogalmak értelmezhetők (parciális) függvényekre:

Definíció (parciálisan rekurzív függvény): *Az $f : I^* \to I^*$ parciális függvény* parciálisan rekurzív *függvény, ha létezik olyan M Turing-gép, hogy $f = f_M$. Ha ezen túl még f minden $s \in I^*$ inputra értelmezve van, akkor f egy* rekurzív *függvény.*

A meghatározásból azonnal kiviláglik, hogy egy rekurzív függvény egyben parciálisan rekurzív is. A *rekurzív* elnevezés onnan ered, hogy a rekurzív függvények éppen azok a függvények, melyek bizonyos egyszerű függvényekből rekurzió segítségével felépíthetők. Ezzel a szintetikusnak mondható megközelítéssel nem foglalkozunk. A *rekurzíve felsorolható* kifejezés arra szándékozik utalni, hogy a nyelv szavai egy alkalmas eljárással felsorolhatók. Erre később (7.7.2. szakasz) még visszatérünk.

A rekurzív nyelveket és a rekurzív függvényeket fogjuk algoritmussal kezelhető nyelveknek és függvényeknek tekinteni. Felmerülhet a kérdés, hogy nem túl szűk-e ez a meghatározás. Nincsenek-e olyan algoritmusok, amelyek nem valósíthatók meg Turing-géppel? Például Pascal-programmal nem lehet-e több függvényt kiszámítani? Első hallásra talán meglepő lesz a válasz: számos kísérlet ellenére sem sikerült eddig olyan algoritmusfogalmat megadni, amely egyrészt megvalósítható a gyakorlatban, másrészt a Turing-kiszámíthatóságnál erősebb. Így például, ami elvégezhető Pascal-programmal, arra szerkeszthető Turing-gép is. Erre – a közvetlen elérésű gép kapcsán – látunk majd bizonyítékot. A Turing-gépeknek ezt a figyelemreméltó tulajdonságát fogalmazza meg a *Church–Turing-tézis.*

Church–Turing-tézis: *Ami algoritmussal – azaz véges eljárással – kiszámítható (eldönthető), az Turing értelmében kiszámítható (eldönthető). Nevezetesen:*

- *Egy $f : I^* \to I^*$ parciális függvény kiszámítható $\Leftrightarrow$ f parciálisan rekurzív.*
- *Egy $f : I^* \to I^*$ (teljes) függvény kiszámítható $\Leftrightarrow$ f rekurzív.*
- *Egy $L \subseteq I^*$ nyelvre a nyelvbe tartozás problémája algoritmussal eldönthető $\Leftrightarrow$ L rekurzív.*

Hangsúlyoznunk kell, hogy ez a tézis nem tekinthető formálisan bizonyított állításnak. Azt a *tapasztalati tényt* fejezi ki csupán, hogy eddig nem sikerült a Turing-gépek erejét meghaladó realisztikus számítási modellt találni. Megtörténhet – noha nehéz elképzelni –, hogy valaki kitalál egy megvalósítható számítóeszközt, amivel nem rekurzív függvények is kiszámíthatók. Egy ilyen fejlemény megdöntené a Church–Turing-tézist.

A Church–Turing-tézist elfogadva a következő két állítás úgy értelmezhető, hogy nem lehet mindent algoritmussal eldönteni, illetve kiszámítani. Vannak algoritmussal nem felismerhető nyelvek és nem kiszámítható függvények.

Állítás: *Van olyan $L' \subseteq I^*$ nyelv, amely nem rekurzíve felsorolható.*

Bizonyítás: Egy Turing-gép leírható véges jelsorozattal, pl. magyarul. Ezért az összes gép számossága megszámlálható. Ez annyit tesz, hogy mind felsorolható természetes számokkal sorszámozva. Legyen $M_0, M_1, M_2, \ldots$ egy ilyen felsorolás. A gép egyértelműen meghatározza az általa felismert nyelvet, vagyis a rekurzíve felsorolható nyelvek is sorszámozhatók a természetes számokkal úgy, hogy egy se maradjon ki: $L_{M_0}, L_{M_1}, L_{M_2}, \ldots$. Elég megmutatni ezután, hogy az $L \subseteq I^*$ nyelveket nem lehet ilyen módon megszámozni. Ebből egyből következik, hogy van olyan L' nyelv, ami nem nyelve egyetlen gépnek sem. (A halmazelmélet elemeiben jártas olvasónak: a rekurzíve felsorolható nyelvek halmaza megszámlálható, az összes nyelvé pedig kontinuum számosságú.)

Az I^* elemei, a véges hosszúságú, I-beli jelekből képzett szavak is megszámozhatók természetes számokkal. Hasznos számozást ad a *kanonikus felsorolás*: vegyük először az üres szót, majd soroljuk fel az 1 hosszúakat, utánuk a 2 hosszúakat, és így tovább. Az azonos hosszúságú szavak az I^* lexikografikus rendezése szerint kövessék egymást. Ehhez feltesszük, hogy adott az I halmazon egy rendezés. A kanonikus felsorolás az I^* szavainak egy $w_0, w_1, w_2, \ldots$ sorrendjét adja.

Megmutatjuk ezután, hogy van olyan $L' \subseteq I^*$ nyelv, amely nem lehet benne a rekurzíve felsorolható nyelvek $L_{M_0}, L_{M_1}, L_{M_2}, \ldots$ sorozatában. Az L' nyelvnek a w_i szó pontosan akkor legyen eleme, ha $w_i \notin L_{M_i}$, $i = 1, 2, \ldots$. Az L' definíciója korrekt. Ugyanis tetszőleges $w \in I^*$ szóról egyértelműen rendelkeztünk, mert minden szó pontosan egyszer szerepel a kanonikus felsorolásban.

Az L' nyelv nem egyezhet meg egyetlen L_{M_i} alakú nyelvvel sem, hiszen a w_i szó L' és L_{M_i} közül az egyiknek eleme, a másiknak nem. Az L' tehát nem rekurzíve felsorolható. □

Az eljárást, amellyel L'-t meghatároztuk, *Cantor-féle átlós módszer*nek nevezzük. Az elnevezés egyik lehetséges magyarázata a következő. Az I feletti nyelvek és a végtelen „igen–nem" sorozatok között I^* kanonikus felsorolása alapján kölcsönösen egyértelmű megfeleltetés hozható létre: egy $L \subseteq I^*$ nyelvnek az a sorozat felel meg, melynek az i-edik tagja pontosan akkor „igen", ha L-ben benne van a w_i szó. Készítsünk egy táblázatot, aminek az i-edik sora az L_{M_i}-nek megfelelő igen-nem sorozat. A táblázat oszlopait a w_i szavakkal indexeljük.

	w_0	w_1	...	w_i	w_{i+1}	...
L_{M_0}	**nem**	nem	...	nem	nem	...
L_{M_1}	igen	**nem**	...	nem	nem	...
⋮						
L_{M_i}	nem	igen	...	**igen**	nem	...
$L_{M_{i+1}}$	igen	nem	...	nem	**nem**	...
⋮						
L'	**igen**	**igen**	...	**nem**	**igen**	...

Az L' nyelv sorozatát – ami a rajzon az alsó sorban van – úgy kapjuk , hogy a táblázat főátlójában levő elemeket az ellenkezőjükre változtatjuk. A Cantor-módszer alapvető eszköz a kiszámíthatatlansági eredmények igazolásában.

Függvényekre a megfelelő állítás az előzőhöz hasonlóan bizonyítható. Itt azt lehet megmutatni, hogy az összes $f : I^* \to I^*$ függvényt nem lehet felsorolni.

Állítás: *Létezik olyan* $f : I^* \to I^*$ *parciális függvény, amely nem parciálisan rekurzív.* □

7.5. Az univerzális Turing-gép

Ennek az előkészítő jellegű résznek a tanulsága úgy foglalható össze, hogy *fordítóprogramok pedig léteznek.* Ezen senki sem lepődik meg különösebben, hiszen se szeri, se száma a különféle fordítóprogramoknak. Olyan programok ezek, amelyek képesek más programokat értelmezni, futtatni. Szinte el sem lehet képzelni nélkülük a számítógépes világot. Létezésük tehát hasznos tény. Ugyanakkor, mint később látni fogjuk, természetesen vezetnek algoritmussal kezelhetetlen feladatokhoz.

Az univerzális Turing-gépek a fordítóprogramok népes családján belül az interpreterek közé tartoznak. Bemenetük két részből áll. Az egyik komponens az interpretálandó program, ami egy M Turing-gép *leírása.* Az M-ről kényelmi okokból feltesszük, hogy egy szalagja van, a bemenő *abc*-je $I = \{0, 1\}$, és egyetlen elfogadó állapota van ($|F| = 1$). A szimulációk kapcsán láttuk, hogy az elfogadó/kiszámító képesség szempontjából ezek nem lényeges megszorítások: bármely N gép átalakítható ilyenné úgy, hogy a kapott új gép is az L_N nyelvet ismerje fel, illetve az f_N függvényt számolja ki.

Az univerzális gép bemenetének másik része az interpretálandó M gép tetszőleges $s \in I^*$ bemenete. Az univerzális gép az M leírását, más szóval *kódját* értelmezve lépésről lépésre utánozza M működését az s bemeneten.

Az első teendő tehát, hogy az interpretálandó M gépből „adatot" csináljunk, amit majd az univerzális gép olvasni tud. Ez nem különösebben nehéz. Arra kell ügyelnünk csupán, hogy minden M gép kódja egy véges bitsorozat legyen, és a kódolás/dekódolás algoritmussal (Turing-géppel vagy Pascal programmal) elvégezhető legyen.

A Turing-gépek egy lehetséges kódolása

Tegyük fel, hogy $k = 1$, $M = (Q, T, I, ü, \delta, q_0, F)$, $I = \{0,1\}$ és $|F| = 1$. Azt is feltehetjük, hogy a gép megadásában szereplő szimbólumok mind számok. Mondjuk legyen

- $I = \{0,1\}$, $T = \{0,1,\ldots,t\}$, $ü = t$,
- $Q = \{0,1,\ldots,q\}$, $q_0 = 0$, $F = \{q\}$,
- balra $= 0$, jobbra $= 1$, helyben $= 2$

Ekkor az M Turing-gép leírása, kódja legyen:

$$q\#t\#q_1\#x_1\#q_1'\#x_1'\#m_1'\#\ldots\#q_r\#x_r\#q_r'\#x_r'\#m_r'\#\#,$$

ahol a megfelelő számokat binárisan írjuk le, továbbá δ értékeit (mindazokon a helyeken, ahol δ értelmezett) a következőképpen soroljuk fel:

$$\text{a } \delta(q_i, x_i) = (q_i', x_i', m_i') \text{ tény kódja } q_i\#x_i\#q_i'\#x_i'\#m_i'.$$

Itt q_i, q_i' állapotok, x_i, x_i' szalagjelek kódjai, m_i' pedig fejmozgást jelöl. Feltehető az is (pl. az $\# = 11$, $0 = 00$, $1 = 01$ átírással), hogy M leírása egy I^*-beli ($I = \{0,1\}$) szó. A kódolás lényeges vonásait így összegezhetjük:

Minden szóba jövő M gépet egy $w \in I^*$ szóval írunk le. Tetszőleges $w \in I^*$ szóhoz legfeljebb egy gép van – ennek jele legyen M_w –, amelynek a kódja w. Az M ismeretében a w kód algoritmussal kiszámítható. A w leírás ismeretében pedig az M_w gép jellemzői (állapotai, átmenetfüggvénye, stb.) algoritmussal megkaphatók.

A következő tételben pontosan megfogalmazzuk, hogy mit értünk univerzális Turing-gép alatt, és vázlatosan meg is adunk egyet. Figyelemre méltó a konstrukció analógiája a számítógépes programozási gyakorlatból ismert interpreterekkel (mint pl. a BASIC nyelv interpreterei).

Tétel: *Van olyan 3-szalagos U Turing-gép, amelyre teljesül a következő: ha $w, s \in I^*$, és M_w létezik, akkor az U gép a $w\#s$ bemenetet pontosan akkor fogadja el (utasítja el, kerül vele végtelen ciklusba), ha M_w az s bemenetet elfogadja (elutasítja, végtelen ciklusba kerül vele).*

Bizonyítás: Csak vázoljuk az U gép szerkezetét. A nem különösebben érdekes részleteket mellőzzük. Az U első szalagja a $w\#s$ inputot tartalmazza, és csupán a w értelmezgetésére (lényegében az M_w átmenetfüggvényét leíró táblázatnak a böngészésére) szolgál. Az U gép második szalagja M_w egyetlen szalagjának felel meg. Tartalma és a fej helyzete az U működése során mindig azonos az M_w éppen szimulált lépésekor tapasztalható szalagtartalommal, illetve fejpozícióval. U harmadik szalagja az M_w-nek a szimulált helyzetben érvényes belső állapotát (pontosabban annak leírását) tartalmazza.

U a szimuláció tényleges megkezdése előtt egy kis előkészítő munkát végez. Először ellenőrzi, hogy M_w létezik-e. Ha nem, akkor itt megáll elutasító állapotban. Ellenkező esetben átmásolja az s inputot a második szalagjára, és a harmadik szalagra a kezdőállapot kódját jegyzi fel. Az előkészítő munka után U szalagjai így festenek:

$w\#s$
s
q_0

Az U az M_w gép egy lépését több lépésben szimulálja a következők szerint: a w leírás alapján meghatározza, hogy M_w mit lépne, ha U második szalagján a fej alatt található jelet a harmadik szalagon ábrázolt belső állapotban látja. Ha mondjuk a szalagjel a, az állapot pedig q, akkor w-ből a $\delta(q,a)$ leírását kell értelmeznie. Ennek ismeretében meglépi M_w lépését; ennek megfelelően módosítja a második és harmadik szalagot. Könnyen látható, hogy egy ilyen szimulációs lépés tényleg megvalósítható Turing-géppel. U szalagjai közvetlenül az M_w gép i-edik lépésének szimulációja után így néznek ki:

$w\#s$
M_w szalagja az i-edik lépés után
M_w belső állapota az i-edik lépés után

U akkor áll meg, ha M_w megáll. Ez esetben pontosan akkor fogadja el a $w\#s$ bemenetét, ha a megállás után az M_w elfogadó állapotának kódja van az utolsó szalagon. Mindezekből könnyen látható, hogy az itt körvonalazott U megfelel a tétel követelményeinek. □

7.6. Alapvető kiszámíthatatlansági tételek

Ha nem vagyok itt, a Hörpentőben keressen. Ha nem vagyok a Hörpentőben, akkor itt vagyok. Kivétel Violin, a fülrepesztő zenész. Ő, ha itt vagyok, keressen a Hörpentőben, ha meg a Hörpentőben vagyok, itt keressen. LÁZÁR ERVIN
(A Sróf mester ajtaján levő felirat
Berzsián és Dideki történetéből.)

Most már megvannak az eszközeink, amelyekkel a kiszámíthatóság korlátaira vonatkozó első klasszikus eredményeket igazolni tudjuk. Megmutatjuk, hogy nem minden nyelv rekurzíve felsorolható; továbbá, hogy nem minden rekurzíve felsorolható nyelv rekurzív. Érvényesek tehát a következő szigorú tartalmazási relációk (2^{I^*} jelöli az összes I feletti nyelvből álló halmazt):

$$\mathcal{R} \subset \mathcal{RE} \subset 2^{I^*}.$$

A nyelvek definíciójában továbbra is az előző részben szereplő Turing-gépekre fogunk szorítkozni. Ezek egyszalagosak, az input *abc*-jük $\{0, 1\}$, és egyetlen elfogadó állapotuk van. Építeni fogunk az univerzális gép tárgyalásakor ismertetett kódolásra is.

7.6.1. A diagonális nyelv – egy nem rekurzíve felsorolható nyelv

Mint már láttuk, létezik nem rekurzíve felsorolható nyelv. Egy ilyen L' nyelv létezését a Cantor-féle diagonális eljárással mutattuk ki. Itt egy másik ilyen konstrukciót ismertetünk, amit Cantor ötletének közvetlenebb alkalmazásával nyerünk.

Tekintsük azon Turing-gépeket, amelyek *nem fogadják el a saját kódjukat*. Tegyük fel, hogy az ezen gépek kódjaiból álló L_d nyelv rekurzíve felsorolható, azaz létezik olyan M Turing-gép, amely éppen ezt a nyelvet ismeri fel. A felismert nyelv fogalmából tudjuk, hogy M akkor és csak akkor ismeri fel egy N gép kódját, ha N kódja benne van a nyelvben. Másrészt a nyelvet éppen úgy adtuk meg, hogy az N kódja pontosan akkor van benne, ha N nem ismeri fel a saját kódját. A két állítást az $M = N$ speciális esetre összerakva azt kapnánk, hogy M akkor és csak akkor ismeri fel a saját kódját, ha nem ismeri fel a saját kódját, ami képtelenség. Tehát az L_d nyelv nem lehet rekurzíve felsorolható. Nézzük ezt egy kicsit pontosabban! Az L_d *diagonális nyelv* a következő:

$$L_d = \{w \in I^*;\ \text{az } M_w \text{ gép létezik, és } w \notin L_{M_w}\}.$$

Tétel: *L_d nem rekurzíve felsorolható.*

Bizonyítás: Indirekt érvelve tegyük fel, hogy a diagonális nyelv rekurzíve felsorolható. Ez annyit tesz, hogy van olyan M Turing-gép, amelyre $L_d = L_M$. Nézzük most ennek az M gépnek a $w \in I^*$ kódját, másként mondva azt a szót, amellyel $M = M_w$ fennáll. Az L_d nyelv és a w szó viszonyát illetően két lehetőség van: vagy $w \in L_d$, vagy pedig $w \notin L_d$. Megmutatjuk, hogy mindkét feltevés ellentmondáshoz vezet, igazolva ezzel a tételt.

Ha $w \in L_d$, akkor L_d definíciója szerint $w \notin L_{M_w}$. De az utóbbi nyelv éppen L_d, tehát $w \notin L_d$, ami ellentmond kiinduló feltevésünknek.

Ha pedig abból indulunk ki, hogy $w \notin L_d$, akkor L_d definíciója szerint $w \in L_{M_w}$, hiszen M_w létezik. Újfent használva az $L_d = L_{M_w}$ egyenlőséget azt kapjuk, hogy $w \in L_d$, ami ismét képtelenség. A bizonyítás ezzel teljes. □

7.6.2. Az univerzális nyelv – egy rekurzíve felsorolható, de nem rekurzív nyelv

A következő algoritmikus kérdés is gépekkel foglalkozik: adott egy M gép és egy $s \in I^*$ szó; döntsük el, hogy M elfogadja-e s-et. Első megoldási kísérletünk az lehetne, hogy futtatjuk M-et, és megnézzük, mi történik. Ez megy is akkor, ha a kérdésre a válasz igenlő, azaz $s \in L_M$. Véges sok lépés után megtudjuk a választ. Ha viszont s végtelen ciklust eredményez, akkor bajban vagyunk. A gép csak megy és megy, és mi nem tudjuk, hogy tényleg végtelen ciklusban van-e, vagy csak egyszerűen egy sokáig tartó véges számítással van dolgunk.

Alaposabban megnézve a kérdést arra jutunk, hogy az algoritmusnak, amit keresünk, valamiféle szuperalgoritmusnak kellene lennie. Olyannak, ami minden gépnél okosabb, hiszen mindent tud, amit a többiek. Mindjárt kiderül, hogy ilyen módszer nincs. A problémának – ami egy eldöntendő kérdés – megfelelő nyelv L_u, az *univerzális nyelv*:

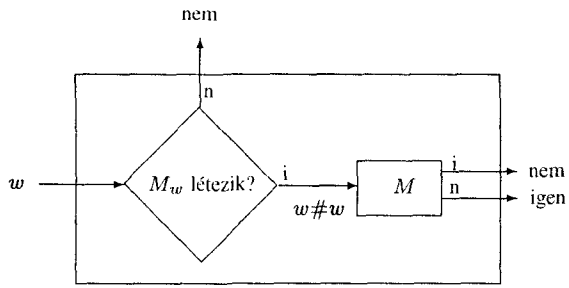
$$L_u = \{w\#s \in I^*;\ \text{az } M_w \text{ gép létezik, és } s \in L_{M_w}\}.$$

Az univerzális nyelv tehát azon *(kód,input szó)* párokból áll, melyekre a kóddal leírt Turing-gép felismeri az illető input szót. Az elnevezés azon az egyszerű észrevételen alapszik, hogy ez a nyelv éppen *az univerzális Turing-gépek nyelve*: pontosan azokból a szavakból áll, amelyeket egy univerzális gép elfogad. Tömören ezt úgy írhatjuk, hogy $L_u = L_U$, ahol U tetszőleges univerzális gép. Innen azonnal látjuk, hogy L_u rekurzíve felsorolható.

Meg fogjuk mutatni másfelől, hogy L_u nem rekurzív. Ha volna egy mindig megálló M algoritmus, ami L_u-t felismerné, akkor szerkeszteni tudnánk egy másik – az M eljárást „hívó" – módszert, ami felismerné a diagonális nyelvet. Ez pedig lehetetlen, hiszen L_d-ről láttuk, hogy nem rekurzíve felsorolható. Érvényes tehát a következő:

Tétel (A. Turing, 1936): *L_u egy rekurzíve felsorolható, de nem rekurzív nyelv.*

Bizonyítás: L_u-t éppen az univerzális Turing-gépek ismerik fel, tehát L_u rekurzíve felsorolható.
Tegyük fel ezután indirekte, hogy L_u rekurzív; legyen M egy mindig megálló Turing-gép, ami felismeri L_u-t. Legyen M' az a Turing-gép, amelynek működését az alábbi folyamatábra mutatja:



Az M' gép a $w \in I^*$ bemenettel elindítva a következő fő lépéseket végzi:
(1) Ellenőrzi, hogy M_w létezik-e. Ha nem, akkor elutasító állapotban megáll.
(2) Ellenkező esetben, ha M_w létezik, akkor elindítja M-et a $w\#w$ inputtal. Ha M elfogadó állapotban áll meg, akkor M' elutasító állapotban végez, ha M elutasító állapotban állt meg, akkor elfogadja a w bemenetet.

Az M' gép mindig megáll, mert az (1) lépés tesztje algoritmussal megvalósítható, és M mindig megáll. Világos továbbá, hogy M' pontosan akkor fogadja el w-t, ha M_w létezik és $w \notin L_{M_w}$. Más szóval M' éppen az L_d nyelvet fogadja el. Ez pedig képtelenség, hiszen L_d nem rekurzíve felsorolható, és ezért nem lehet egyetlen gép nyelve sem. M létezését feltételezve ellentmondást kaptunk, igazolva ezzel, hogy L_u nem rekurzív. □

7.7. Összefüggések a kiszámíthatósági fogalmak között

Az eddig vett kiszámíthatósági fogalmak közötti viszonyokkal foglalkozunk a következőkben. Először a rekurzíve felsorolható és a rekurzív nyelvek közötti kapcsolatot vesszük szemügyre, majd a függvények számításának és a nyelvek felismerésének feladatait vetjük össze.

7.7.1. Rekurzivitás és rekurzíve felsorolhatóság

A nyelvfelismerési feladatokról megállapítottuk, hogy igen-nem problémáknak felelnek meg. Valamely P tulajdonsággal rendelkező szavak halmazát kell felismerni. A nyelvbe azok a szavak tartoznak, melyekre P igaz. Érdekes szerepe van itt a P tulajdonság $\neg P$ tagadásának. A $\neg P$ pontosan akkor teljesül egy $s \in I^*$ szóra, ha P nem teljesül s-re. Ezért, ha a P igenpéldányaiból álló nyelv L, akkor a $\neg P$ igenpéldányaiból álló nyelv az L *komplementere*: $I^* \setminus L$. Ha van egy mindig megálló M algoritmusunk a P eldöntésére, azaz L felismerésére, akkor a $\neg P$ tulajdonságot is el tudjuk dönteni. Csupán az M elfogadó és elutasító állapotait kell felcserélni. Az így kapott M' gép az $I^* \setminus L$ nyelvet ismeri fel.

Ha csak egy fél algoritmusunk van, azaz $L = L_M$, de M nem áll meg minden bemeneten, akkor az állapotok felcserélésével nem érünk sokat. Az olyan $s \in I^*$ bemenetekre, amelyekre M nem áll meg, sosem tudjuk meg a választ véges sok lépés után. Ha viszont a P és a $\neg P$ eldöntésére is van egy-egy fél algoritmusunk, akkor ezek összerakhatók egy egésszé. A két gépet párhuzamosan működtetve véges időben megkapjuk a választ minden bemenetre. Ezeket az észrevételeket pontosan is megfogalmazzuk. Bevezetünk előbb egy fogalmat, amit a bonyolultságelméleti fejezetben is használni fogunk.

A nyelvekből álló halmazokat (2^{I^*} részhalmazait) *nyelvosztályoknak* nevezzük. Nyelvosztályokra eddigi példáink $\mathcal{R}$, $\mathcal{RE}$, 2^{I^*}. Legyen $X \subseteq 2^{I^*}$ egy nyelvosztály. Ekkor a *komplementer nyelvosztály*, co$X$ az X-beli nyelvek komplementereiből áll:

$$\text{co}X = \{L \subseteq I^* : \ I^* \setminus L \in X\}.$$

Az így értelmezett komplementerképzés fontos sajátja, hogy megőrzi a nyelvosztályok közötti tartalmazási viszonyokat:

$$X \subseteq Y \subseteq 2^{I^*} \implies \text{co}X \subseteq \text{co}Y.$$

Ugyanis ha $L \in \text{co}X$, akkor definíció szerint $I^* \setminus L \in X$. Az $X \subseteq Y$ tartalmazás miatt $I^* \setminus L \in Y$, amiből pedig $L \in \text{co}Y$. Szintén egyszerűen igazolható a tetszőleges X nyelvosztályra érvényes co(co$X$) $= X$ összefüggés.

A bevezetett jelöléssel a következőképpen fogalmazhatjuk meg észrevételeinket:

Tétel:
(1) $\mathcal{R} = co\mathcal{R}$
(2) $\mathcal{R} = \mathcal{RE} \cap co\mathcal{RE}$

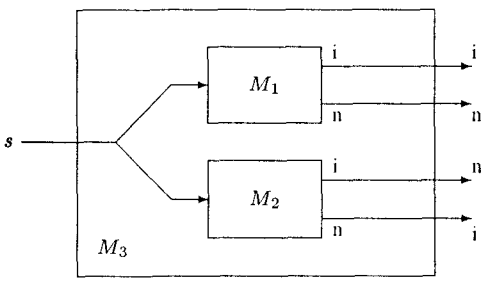
Bizonyítás: (1) Tegyük fel, hogy $L \in \mathcal{R}$. Legyen ennek megfelelően M egy Turing-gép, mely minden inputon megáll és az L nyelvet fogadja el. Az M elfo-

gadó és elutasítva megálló állapotainak felcserélésével olyan gépet nyerünk, amelyik éppen az L komplementerét, az $I^* \setminus L$ nyelvet ismeri fel. Nyilván az új gép is mindig megáll, tehát $I^* \setminus L$ is rekurzív. Beláttuk ezzel, hogy $\mathrm{co}\mathcal{R} \subseteq \mathcal{R}$. A fordított irányú tartalmazás innen már adódik a co-operátorra megismert szabályokból:

$$\mathcal{R} = \mathrm{co}(\mathrm{co}\mathcal{R}) \subseteq \mathrm{co}\mathcal{R}.$$

(2) A nyilvánvaló $\mathcal{R} \subseteq \mathcal{RE}$ egyenlőtlenségből komplementerképzéssel nyerjük, hogy $\mathcal{R} = \mathrm{co}\mathcal{R} \subseteq \mathrm{co}\mathcal{RE}$, így összességében $\mathcal{R} \subseteq \mathcal{RE} \cap \mathrm{co}\mathcal{RE}$. A fordított irányú tartalmazás igazolásához tegyük fel, hogy $L \in \mathcal{RE} \cap \mathrm{co}\mathcal{RE}$. Legyen ennek megfelelően M_1, illetve M_2 két Turing-gép, melyek az L, illetve az $I^* \setminus L$ nyelvet ismerik fel. Ezekből egy mindig megálló M_3 algoritmus szerkeszthető, melye $L = L_{M_3}$.

Legyen $s \in I^*$ a bemenő szó. Az M_3 gépet szemléletesen úgy képzelhetjük el, hogy az M_1 és M_2 gépek egymás mellett, egymástól függetlenül párhuzamosan dolgoznak ugyanazzal az s inputtal:



Az M_3 pontosan akkor álljon meg, ha M_1 és M_2 valamelyike megáll. M_3 akkor fogad el, ha a megállás M_1 elfogadó, vagy pedig M_2 elutasító állapotában történt. Világos, hogy az M_3 az L nyelvet ismeri fel. Az is igaz, hogy mindig megáll, hiszen ha $s \in L$, akkor M_1, ha pedig $s \notin L$, akkor M_2 fog biztosan megállni véges sok lépés után.

M_3 megvalósítható párhuzamosság nélkül is. A megfelelő Turing-gép szalagjai egy részén végzi az M_1 munkáját, egy ettől diszjunkt másik részén pedig az M_2 teendőit. M_3 felváltva szimulálja a két gép lépéseit: a $2i-1$-edik lépésben az M_1 i-edik, a $2i$-edik lépésben pedig az M_2 i-edik lépését teszi meg. Ezáltal, ha valamelyik gép megáll a k-adik lépésében, akkor M_3 is megáll, legkésőbb a $2k$-adik lépésben.

Beláttuk tehát, hogy L rekurzív nyelv, vagyis $\mathcal{R} \supseteq \mathcal{RE} \cap \mathrm{co}\mathcal{RE}$. A másik tartalmazással együtt ez bizonyítja a (2) állítást.□

Feladat: Az előző érvelésben az M_1 és M_2 párhuzamos futását úgy alakítottuk sorossá, hogy felváltva léptettük őket. Milyen más módokon fésülhetők össze a két gép lépései úgy, hogy a bizonyítás továbbra is működjön?

A tételt olyan állításnak tekinthetjük, amely a nyelvosztályainknak a *komplementer* műveletre való zártságával foglalkozik. Kiderül, hogy a rekurzív nyelvek összessége zárt a komplementer képzésére – azaz $L \in \mathcal{R}$ esetén $I^* \setminus L \in \mathcal{R}$ is igaz –, míg a rekurzíve felsorolható nyelvekre ez nem teljesül. A következő feladat az unió- és a metszetképzés műveleteire való zártságról szól.

Feladat: Tegyük fel, hogy az L_1 és L_2 nyelvek rekurzíve felsorolhatók (rekurzívek). Igazoljuk, hogy $L_1 \cap L_2$ és $L_1 \cup L_2$ is rekurzíve felsorolható (rekurzív). (Az előző tételben látottakhoz hasonlóan működtessük párhuzamosan az L_1 és L_2 nyelveket felismerő M_1 és M_2 gépeket.)

7.7.2. Függvények és halmazok (nyelvek)

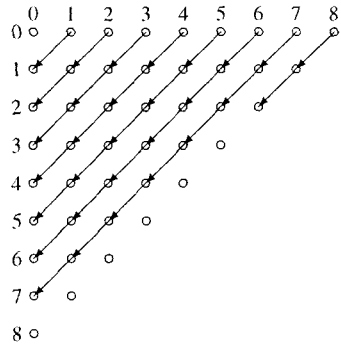
A Turing-gépeknek kétféle működési módjával ismerkedtünk meg. Az egyik a nyelvek felismerése, a másik pedig a függvények számítása. Az előbbinél a válasz egyetlen bittel ábrázolható (elfogadás vagy annak a hiánya), a másodiknál hosszabb is lehet az eredmény. Ebben az értelemben a függvények számítása általánosabb feladat, mint a nyelvek felismerése. Az általános és a speciális közötti összefüggések szinte mindig érdekesek. Itt a kiszámíthatóság szempontjából vetjük össze a két fogalmat. Később – a kisebb bonyolultságú feladatok tanulmányozásakor – újra előkerül majd a kétféle működés közötti viszony.

A kiszámíthatóság nézőpontjából a rekurzíve felsorolható nyelvek a parciálisan rekurzív függvényekkel, míg a rekurzív nyelvek a rekurzív függvényekkel hozhatók természetes kapcsolatba. Ezeket fogalmazza meg a következő két tétel.

Tétel: *Az $L \subseteq I^*$ nyelv akkor és csak akkor rekurzíve felsorolható, ha van olyan $f : I^* \to I^*$ parciálisan rekurzív függvény, melynek értékkészlete éppen az L nyelv (szokásos jelöléssel $Im(f) = L$).*

Bizonyítás: $\Rightarrow$: Tegyük fel, hogy L rekurzíve felsorolható. Ennek megfelelően van olyan M Turing-gép, melyre $L = L_M$. Legyen ezután M' olyan Turing-gép, mely kezdetben az $s \in I^*$ bemenő szót felmásolja az output szalagjára, azután pedig lépésről lépésre szimulálja az M gépet. Az egyetlen kivétel, hogy ha M elutasító állapotban áll meg, akkor M' végtelen ciklusba esik. Az M' gép tehát kizárólag az $s \in L$ szavakra áll meg; megálláskor s lesz az output szalagon. Az M' által kiszámított $f_{M'}$ függvény értékkészlete ezért éppen L.

$\Leftarrow$: Tegyük fel, hogy f egy parciálisan rekurzív függvény, mondjuk $f = f_M$, ahol M egy Turing-gép. Tekintsük az I^* szavainak $w_0, \ldots, w_n \ldots$ kanonikus felsorolását (a rendezés hosszúság szerint növekvően, azonos hosszúak között lexikografikusan). Használni fogjuk még a természetes számokból álló párok egy felsorolását is. A következő ábrán látható táblázaton megyünk végig a délnyugati irányú átlók mentén. A sorozat eleje így alakul: $(0,0), (0,1), (1,0), (0,2), (1,1), (2,0)$, stb.



A felsorolások két tulajdonsága lesz fontos a számunkra. Egyrészt mindkettő megvalósítható algoritmussal. A sorozatok adott elemének az ismeretében a következő elem számítással meghatározható. A másik lényeges vonásuk, hogy mindkét felsorolás kimerítő; mindegyik szóra, illetve párra sor kerül valamikor.

Az M' Turing-gép a teendőit az (i,j) párok sorozatához kapcsolódóan, a sorozat mentén haladva végzi. Tegyük fel, hogy $s \in I^*$ az M' bemenete. Az (i,j) párhoz kapcsolódóan M' a következőket teszi: szimulálja az M első $\leq i$ lépését a w_j szón. Ha ezalatt M megáll és o outputot produkál, akkor ellenőrzi, hogy $s = o$ teljesül-e. Ha igen, akkor M' megáll és elfogadja s-et. Minden más esetben (M nem áll meg i lépésen belül, vagy megálláskor az eredmény nem s) a felsorolás szerint következő párral folytatja a munkát. Ha ez (i', j') akkor M első i' lépését teszi meg a $w_{j'}$ szóval, stb.

Mi lesz az M' gép $L_{M'}$ nyelve? Világos, hogy $L_{M'} \subseteq \mathrm{Im}(f)$, hiszen M' csak olyan s szót fogad el, ami megkapható mint M egy számításának az eredménye. Fordítva, tegyük fel, hogy $s \in \mathrm{Im}(f)$. Van tehát egy olyan j, hogy $s = f_M(w_j)$. Tegyük fel, hogy M a w_j bemeneten i lépésben kapja meg az s eredményt. Ekkor M' – ha előbb nem – az (i,j) pár feldolgozásakor elfogadja s-et. Ezzel igazoltuk az $L_{M'} \supseteq \mathrm{Im}(f)$ tartalmazást is. Igaz tehát, hogy $L_{M'} = \mathrm{Im}(f)$. □

A bizonyítás második felének ötletét felhasználva tetszőleges rekurzíve felsorolható L nyelvhez szerkeszthető olyan algoritmus, ami éppen az L nyelv szavait *sorolja fel* valamilyen sorrendben. Innen ered a *rekurzíve felsorolható* kifejezés.

A rekurzivitásra vonatkozó tétel kimondása előtt emlékeztetünk egy hasznos fogalomra. Az egyszerűség kedvéért feltesszük, hogy $0, 1 \in I$. Egy $L \subseteq I^*$ nyelv *karakterisztikus függvénye*, χ_L a következő:

$$\chi_L(s) = \begin{cases} 1 \in I^*, & \text{ha } s \in L \\ 0 \in I^*, & \text{ha } s \notin L \end{cases}$$

Tétel: *Az $L \subseteq I^*$ nyelv pontosan akkor rekurzív, ha χ_L egy rekurzív függvény.*

Bizonyítás: $\Rightarrow$: Tegyük fel, hogy L rekurzív: M olyan Turing-gép, mely L-et ismeri fel, és minden inputra megáll. Legyen ezután M' olyan Turing-gép, mely szimulálja M-et, majd 1-et vagy 0-t ír az output szalagjára, annak megfelelően, hogy M elfogadó vagy elutasító állapotban állt meg. Világos, hogy M' éppen az L karakterisztikus függvényét számolja ki.

$\Leftarrow$: Tegyük fel, hogy χ_L rekurzív függvény, és ennek megfelelően az M Turing-gép a χ_L függvényt számítja ki. Legyen M' egy olyan Turing-gép, mely lépésről lépésre szimulálja M-et; végül elfogadó, illetve elutasító állapotban áll meg aszerint, hogy M output szalagján 1, vagy 0 a végeredmény. Ez az M' gép pontosan az L nyelvet ismeri fel. □

7.8. További eldönthetetlen problémák

Visszatérünk az algoritmussal nem megoldható feladatokhoz. Az ember először hajlamos ezeket afféle patologikus eseteknek tekinteni, amelyek nem mondanak sokat a természetesen felmerülő problémák iránt érdeklődőknek. Szeretnénk néhány példával érzékeltetni, hogy ez távolról sem igaz: ilyen reménytelenül nehéz problémákkal a legkülönbözőbb területeken is találkozhatunk. Egy meghatározással kezdjük, amellyel nem új fogalmat, hanem inkább egy új elnevezést rögzítünk.

Definíció (eldönthetetlen nyelv): *Az $L \subseteq I^*$ nyelvet* eldönthetetlen nyelv*nek nevezzük, ha L nem rekurzív.*

A szóhasználat azt takarja, hogy az L nyelvbe tartozás/nem-tartozás problémáját nem lehet algoritmussal eldönteni: nincs olyan algoritmus, mely véges lépésben meg tudná mondani, hogy egy adott szó benne van-e a nyelvben. Turing tétele szerint az a probléma, hogy egy Turing-gép egy adott inputot elfogad-e, algoritmikusan eldönthetetlen. Itt további példákat adunk eldönthetetlen problémákra. Az igazán érdekesek azok lesznek, amelyek a Turing-gépek formalizmusától függetlenül, természetesen megfogalmazhatók.

7.8.1. A Megállási probléma (Halting problem)

Egy program tesztelésekor fontos kérdés, hogy nincs-e benne végtelen ciklus. Jó lenne hatékony módszert találni ennek az eldöntésére. Legyünk szerényebbek: nézzük csak azt ez egyszerűbb kérdést, hogy egy adott program egy rögzített bemenettel megáll-e véges sok lépés után. Még ez az ártatlannak tűnő kérdés is eldönthetetlen[2].

A *Megállási probléma* kérdése az, hogy egy (a kódjával adott) Turing-gép adott bemenettel egyáltalán megáll-e. A problémához tartozó L_h nyelv a következő:

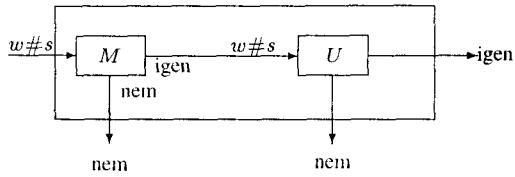
$$L_h = \left\{ w\#s \in I^* \;\middle|\; \begin{array}{l} \text{az } M_w \text{ gép létezik, és az } s \text{ bemenettel} \\ \text{elindítva véges sok lépésben megáll} \end{array} \right\}.$$

Az L_h nyelv nyilván tartalmazza az L_u univerzális nyelvet. A következő tétel gondolatmenete tulajdonképpen a két nyelv közötti algoritmikus kapcsolaton alapul.

Tétel: $L_h \in \mathcal{RE} \setminus \mathcal{R}$.

Bizonyítás: Lényegében az univerzális Turing-gép mutatja, hogy $L_h \in \mathcal{RE}$. Csak annyi módosítás szükséges, hogy megálláskor a gép *mindig* menjen át elfogadó állapotba. Az így megbütykölt gép pontosan akkor fogadja el a $w\#s$ bemenetet, ha ezzel U megáll, azaz ha $w\#s \in L_h$.

Az állítás fennmaradó részének igazolásához tegyük fel indirekte, hogy L_h rekurzív. Eszerint van olyan M Turing-gép, ami L_h-t ismeri fel, és mindig megáll. Legyen ekkor M' az a Turing-gép, mely a $w\#s$ inputon először az M gépet futtatja le. Ha M elutasítja a bemenetét, akkor M' álljon meg elutasító állapotban. Ha M elfogadó állapotban áll meg, akkor M' szimulálja az U univerzális Turing-gépet a $w\#s$ bemeneten, beleértve az elfogadást/elutasítást is.



[2]Itt komoly szerephez jut az a tény, hogy Turing-gépek szalagja végtelen. Ha – az igazi számítógépekhez hasonlóan – csak véges méretű tárat engedünk meg, akkor a feladat megoldhatóvá válik. A következő fejezetben a tár–idő-tételben adunk egy módszert a tárkorlátos végtelen ciklusok felismerésére. Ez azonban a tesztelés szemszögéből csak elvi jelentőségű; nem ismert gyakorlati szempontból elfogadható megoldás a feladatra.

Az M-re tett feltevés miatt M' mindig megáll. Nyilvánvaló ezen felül, hogy M' éppen az L_u univerzális nyelvet ismeri fel. Mindezek ellentmondanak annak, hogy L_u eldönthetetlen (Turing tétele). □

Valamivel több is mondható: a Megállási probléma akkor is algoritmikusan eldönthetetlen marad, ha az M_w gépet csupán az üres inputtal futtatjuk. Legyen

$$L_\epsilon = \{w \in I^* : \ M_w \text{ létezik és az } \epsilon \text{ (üres) inputon megáll}\}.$$

Tétel (A. Church, 1936):

$$L_\epsilon \in \mathcal{RE} \setminus \mathcal{R}.$$

Bizonyítás: Az L_ϵ nyelv rekurzíve felsorolhatósága így látható: a $w \in I^*$ inputból elkészítjük a $w\#\epsilon = w\#$ szót, majd ezzel a bemenettel lefuttatjuk az L_h megállási nyelvet felismerő gépet. Az így adódó algoritmus pontosan az L_ϵ nyelvet ismeri fel.

Az $L_\epsilon \notin \mathcal{R}$ állítás igazolására megmutatjuk, hogy az általános Megállási probléma visszavezethető erre a speciális esetre: egy M Turing-gépnek az s bemeneten való működése szimulálható egy olyan (az M-től és az s szótól egyaránt függő) Turing-géppel, amely üres input esetén felmásolja az s szót az input szalagjára, majd M működését lépésről lépésre utánozza. Ha tehát az L_ϵ nyelv eldönthető volna, akkor az általános Megállási feladat is eldönthető lenne. □

7.8.2. Hilbert 10. problémája

... ennek ősi és egyszerű íze van, mint talán az Ezeregyéjszakának...
JORGE LUIS BORGES: Ember a küszöbön

1900-ban Párizsban a világ matematikusainak kongresszusán David Hilbert egy azóta legendás hírűvé vált előadást tartott. Ebben a jövő fontos kutatási irányait próbálta meg kijelölni. Huszonhárom olyan kérdést fogalmazott meg, melyekről úgy vélte, hogy a megválaszolásuk lényeges előrelépést jelentene. Kifejezte azt a meggyőződését is, hogy a kérdések nehezek, és a XX. század nem lesz elég a megoldásukhoz. A két jóslat közül az utóbbi vált be kevésbé. A kérdések jó részét mára megoldották, és mindegyikkel kapcsolatban komoly eredmények születtek. A problémák jelentőségét illetően nem tévedett: óriási hatást gyakoroltak a tudomány fejlődésére. Nem kivétel ez alól a tizedik kérdés sem.

A tizedik problémában Hilbert célul tűzte ki, hogy adjunk általános módszert *diofantikus egyenletek* megoldhatóságának eldöntésére. A probléma pontos

megfogalmazásához szükségünk lesz egy fogalomra. *Egész együtthatós polinomon* egy olyan kifejezést értünk, amely változókból és egész számokból építhető fel az összeadás, kivonás és szorzás műveleteivel. Eszerint az $x^3y^2 - 5zu$ kifejezés tekinthető az x, y, z, u változók egy egész együtthatós polinomjának. Jelölje $\mathbb{Z}[x_1, \ldots, x_m]$ azon egész együtthatós polinomok összességét, melyekben a változók $x_1, x_2, \ldots, x_m$ közül kerülnek ki; például $x^3y^2 - 5zu \in \mathbb{Z}[x, y, z, u]$.

Tetszőleges $f(x_1, \ldots, x_m) \in \mathbb{Z}[x_1, \ldots, x_m]$ polinom felírható

$$f(x_1, \ldots, x_m) = \sum_{i_1=0}^{n_1} \cdots \sum_{i_m=0}^{n_m} a_{i_1 \ldots i_m} {x_1}^{i_1} \cdots {x_m}^{i_m}$$

alakban, ahol $n_1, \ldots, n_m \in \mathbb{Z}^+$ és az $a_{i_1 \ldots i_m}$ együtthatók pedig egész számok. Erről könnyen meggyőződhetünk, ha az f-et megadó kifejezésben felbontjuk a zárójeleket. Az f polinom *foka* az előző felírásban előforduló legnagyobb kitevőösszeg:

$$\deg f = \max\{i_1 + \ldots + i_m \mid a_{i_1 \ldots i_m} \neq 0\}.$$

Az

$$(*) \qquad f(x_1, \ldots, x_m) = 0$$

alakú egyenleteket *diofantikus egyenleteknek* nevezzük, ahol $f(x_1, \ldots, x_m)$ egész együtthatós polinom. A $(*)$ diofantikus egyenlet *megoldásán* egy olyan $(u_1, \ldots, u_m) \in \mathbb{Z}^m$ egészekből álló m-est értünk, melyre $f(u_1, \ldots, u_m) = 0$. A diofantikus egyenletet *megoldhatónak* nevezzük, ha van ilyen értelemben vett megoldása. A diofantikus egyenletek megoldhatóságának vizsgálata, megoldásaik keresése szinte a szám fogalmának létezése óta művelt, rendkívül gazdag terület. Ilyen egyenlet (pozitív) megoldásaiként adhatók meg például a *pitagoraszi számhármasok*: azok a természetes számokból álló (x, y, z) hármasok, melyek egy derékszögű háromszög oldalai lehetnek:

$$x^2 + y^2 - z^2 = 0.$$

Hilbert tizedik kérdése így fogalmazható: *adjunk egy olyan véges sok lépésből álló eljárást, mely tetszőleges egész együtthatós f polinomra eldönti, hogy az f-nek megfelelő $(*)$ egyenlet megoldható-e.*

Úgy tűnik, Hilbert a problémát abban a meggyőződésében tette közzé, hogy minden értelmesen megfogalmazott számítási problémára van algoritmus. Ezt a harmincas években Kurt Gödel, Alonzo Church és Alan Turing munkássága alaposan megcáfolta. Mint láttuk, vannak olyan feladatok, amelyekre nem létezik megoldó algoritmus. Ezek után már természetesen merült fel a lehetőség, hogy

a tizedik probléma feladata is egy ilyen kemény dió. Ebbe az irányba mutatott az a régen megfigyelt tény is, hogy a diofantikus egyenleteknek központi szerepük van a matematikai hátterű algoritmikus kérdésekben: sok fontos igen–nem problémát át tudtak fogalmazni diofantikus egyenlet megoldhatóságává.

A tizedik probléma sokáig ellenállt a kutatók ostromának, míg végül Martin Davis, Julia Robinson és mások munkáira építve 1970-ben Jurij Matijaszevics megmutatta, hogy *nem létezik olyan algoritmus, ami a (∗) alakú egyenletek megoldhatóságát eldöntené.* A tizedik problémában megfogalmazott algoritmikus kérdés eldönthetetlen.

A nevezetes eredmény ismertetése kényelmesebb lesz, ha nyelvek helyett természetes számok halmazaival foglalkozunk. A kanonikus felsorolás kölcsönösen egyértelmű megfeleltetést létesít $\{0,1\}^*$ szavai és a természetes számok között: az $s \in \{0,1\}^*$ szónak azon i index felel meg, melyre $s = w_i$. Ezzel egyszersmind kölcsönösen egyértelmű megfeleltetést kapunk a természetes számok részhalmazai és az $L \subseteq \{0,1\}^*$ nyelvek között. E megfeleltetés révén beszélhetünk arról, hogy egy $H \subseteq \mathbb{Z}^+$ számhalmaz rekurzív, illetve rekurzíve felsorolható. Ez egyszerűen azt jelenti, hogy a H-beli sorszámokkal azonosított szavak halmaza rekurzív, illetve rekurzíve felsorolható.

Legyen $m \geq 0$ és $f(x_1, x_2, \ldots, x_m, y) \in \mathbb{Z}[x_1, x_2, \ldots, x_m, y]$ egy egész együtthatós $m+1$-változós polinom és legyen

$$H_f = \left\{ v \in \mathbb{Z}^+ \;\middle|\; \begin{array}{l} \text{vannak olyan } u_1, u_2, \ldots, u_m \text{ természetes számok,} \\ \text{hogy } f(u_1, u_2, \ldots, u_m, v) = 0 \end{array} \right\}.$$

Definíció: *A $H \subseteq \mathbb{Z}^+$ halmaz* diofantikus halmaz, *ha van olyan f polinom, hogy $H = H_f$.*

A diofantikus halmaz fogalma mögött egy számítási modell húzódik meg, ami első látásra bizarrnak és szárnyaszegettnek tűnik. Ebben az f polinom jelenti a programot. A $v \in \mathbb{Z}^+$ számot a program akkor fogadja el, ha vannak olyan u_i természetes számok, hogy $f(u_1, u_2, \ldots, u_m, v) = 0$. Az rögvest látszik, hogy ez a furcsán értelmezett számítás megvalósítható Turing-géppel: az adott v bemenethez sorra vesszük az $(u_1, u_2, \ldots, u_m) \in (\mathbb{Z}^+)^m$ vektorokat; az $(u_1, u_2, \ldots, u_m)$ vektor kapcsán kiszámítjuk az $f(u_1, u_2, \ldots, u_m, v)$ helyettesítési értéket. Ha ez nulla, akkor elfogadjuk v-t, ha nem, akkor a következő vektorral próbálkozunk. A vektorok felsorolásával mint számítási feladattal már találkoztunk. Az $m = 2$ esetre adtunk egy módszert. Ez általánosítható tetszőleges m-re: sorra vesszük azokat a vektorokat, amelyek komponenseinek összege i, ahol $i = 0, 1, 2, \ldots$. Az adott i összegű vektorokból véges sok van, ezeket pl. lexikografikus sorrendben vehetjük.

Az előző fejtegetés szerint a H_f alakú halmazok, vagyis a diofantikus halmazok rekurzíve felsorolhatók. Olyan eljárást adtunk, ami a H_f elemeit elfogadja és

a H_f-en kívüli bemenetekre sosem áll meg. A Matijaszevics által betetőzött erőfeszítések fő eredménye a fordított irányú állítás:

Tétel (Matijaszevics tétele): *Ha egy $H \subseteq \mathbb{Z}^+$ halmaz rekurzíve felsorolható, akkor diofantikus is.*

□

A tétel bizonyításától itt el kell tekintenünk. Ha tehát $H \subseteq \mathbb{Z}^+$ rekurzíve felsorolható, akkor van olyan f polinom, melyre $H = H_f$. Az f polinom választható úgy, hogy $m = 14$ és $\deg f \leq 4$ is teljesüljön. A tétel szerint a polinomokra épülő számítási modell felismerőképessége megegyezik a Turing-gépekével.

Matijaszevics tételéből elég egyszerűen adódik a tizedik probléma eldönthetetlensége. A bizonyításhoz hasznos lesz a számelmélet következő klasszikus gyöngyszeme:

Tétel (J. L. Lagrange, 1751): *Minden nemnegatív egész szám előáll négy egész szám négyzetének összegeként.* □

Ezek után megmutatjuk, hogy ha volna algoritmus a diofantikus egyenletek megoldhatóságának véges sok lépésben való eldöntésére, akkor az L_h megállási nyelv is eldönthető lenne.

Tétel: *A diofantikus egyenletek megoldhatósága algoritmikusan eldönthetetlen.*

Bizonyítás: A megoldható egyenletekből (illetve azok leírásaiból) álló nyelv rekurzíve felsorolható. Ez ismét csak a számokból álló m-esek (ahol m az egyenlet változóinak a száma) felsorolásán alapuló próbálgatással adódik. Ha az egyenlet megoldható, akkor a szisztematikus behelyettesítgetés előbb-utóbb megoldást ad.

Azt kell ezután megmutatnunk, hogy nincs olyan algoritmus, amely mindig megáll, és éppen a megoldható egyenleteket fogadja el. Tegyük fel indirekte, hogy létezik egy ilyen M módszer.

Legyen $H \subset \mathbb{Z}^+$ az L_h nyelvnek megfelelő számhalmaz. H rekurzíve felsorolható, ezért Matijaszevics tétele szerint diofantikus is. Van olyan $f(x_1, \ldots, x_{14}, y)$ polinom, hogy $v \in \mathbb{Z}^+$ éppen akkor tartozik H-ba, ha az $f(x_1, \ldots, x_{14}, v) = 0$ egyenlet megoldható a *nemnegatív egészek* körében. (Ennek az egyenletnek a változói $x_1, x_2, \ldots, x_{14}$.) Lagrange tételét figyelembe véve ez egyenértékű azzal, hogy a következő 56 változós egyenlet megoldható-e az *egészek* között:

$$f(p_1^2 + q_1^2 + r_1^2 + s_1^2, p_2^2 + q_2^2 + r_2^2 + s_2^2, \ldots, p_{14}^2 + q_{14}^2 + r_{14}^2 + s_{14}^2, v) = 0.$$

Az M eljárás véges sok lépésben eldönti ennek az egyenletnek a megoldhatóságát, így a ($v \in H$?) kérdést is; ez pedig ellentmondás, hiszen L_h nem rekurzív. □

Érintünk még egy érdekes problémakört, ami számhalmazoknak polinomok értékeiként való megadásával kapcsolatos. Az utóbbi századokban sokan próbáltak explicit formulákat adni bizonyos nevezetes sorozatokra. Így például meglehetősen népszerűek voltak a prímszámokat előállító képletek. Egy jellegzetes ilyen eredmény Leonhard Euler 1772-ből származó észrevétele, miszerint $n^2 - n + 41$ prímszám, ha $0 \leq n \leq 40$.[3]

Matijaszevics tételének van egy ilyesforma előállíthatósággal kapcsolatos szép következménye. Nevezzük a $H \subseteq \mathbb{Z}^+$ halmazt *explicitnek*, ha H előáll mint egy alkalmas egész együtthatós m-változós $g \in \mathbb{Z}[x_1, \ldots, x_m]$ polinomnak a nemnegatív egész helyeken felvett nemnegatív értékeinek H^g halmaza:

$$H = H^g := \left\{ u \in \mathbb{Z}^+ \;\middle|\; \begin{array}{l} \text{vannak olyan } u_1, \ldots, u_m \text{ természetes számok,} \\ \text{hogy } u = g(u_1, \ldots, u_m) \end{array} \right\}.$$

A definícióból az $(u_1, u_2, \ldots, u_m)$ vektorok felsorolásán alapuló érveléssel azonnal következik, hogy az explicit halmazok rekurzíve felsorolhatók. Matijaszevics tételéből egyszerűen adódik, hogy a fordított állítás is igaz.

Következmény: *A rekurzíve felsorolható halmazok explicitek.*

Bizonyítás: Legyen $H \subseteq \mathbb{Z}^+$ egy rekurzíve felsorolható halmaz. Matijaszevics tétele szerint alkalmas $f(x_1, x_2, \ldots, x_{14}, y)$ polinommal $H = H_f$ teljesül. Legyen ezután a $g \in \mathbb{Z}[x_1, \ldots, x_{14}, y]$ a következő polinom:

$$g(x_1, \ldots, x_{14}, y) = (y+1)(1 - f^2(x_1, \ldots, x_{14}, y)) - 1.$$

Ha $f(u_1, \ldots, u_{14}, v) \neq 0$, és $v \geq 0$, akkor $g(u_1, \ldots, u_{14}, v) < 0$, így nemnegatív értékeket $v \geq 0$ esetén g csak az f zérushelyeinél vehet fel. Ha pedig $f(u_1, \ldots, u_{14}, v) = 0$, akkor $g(u_1, \ldots, u_{14}, v) = (v+1)(1-0) - 1 = v$. A g nemnegatív értékei a természetes számokon tehát éppen a H_f elemei. □

A prímszámok halmaza rekurzív, hiszen a prímség eldönthető algoritmussal. Következésképpen a prímszámok halmaza explicit; ilyen értelemben tehát előállítható „képlet" segítségével. Ténylegesen meg is adtak ilyen polinomokat. A kisebbek ezek közül féltenyérnyi helyen leírhatók. Viselkedésük azonban túl bonyolult

[3] Euler polinomja csúcstartónak számít. Az első 41 természetes számnál mindenütt prím értéket vesz fel, mégpedig csupa különbözőt. Az 1 főegyüttható̋ju másodfokú polinomok között ez a leghosszabb ilyen sorozat. Ha más főegyüttható is megengedett, akkor valamivel hosszabb sorozatok is elérhetők. Például a $36x^2 - 810x + 2753$ (ún. Ruby-polinom) első 45 értéke prímszám.

ahhoz, hogy a gyakorlatban hasznavehetőek legyenek például prímek szisztematikus előállítására (csúnya, ámde népszerű szakszóval: generálására).

Az explicit halmazok értelmezésében két helyen is szerepel a nemnegativitás. Ezek közül az egyik nem lényeges.

Feladat: Mutassuk meg, hogy az explicit halmazok definíciójában az $u_i \in \mathbb{Z}^+$ feltétel helyettesíthető a gyengébb $u_i \in \mathbb{Z}$ feltétellel. (Lagrange tétele használható.)

A definíció másik érdekes eleme, hogy nem a g teljes értékkészlete a H halmaz, csak a nemnegatív u értékek számítanak. Ez a feltétel már nem engedhető el:

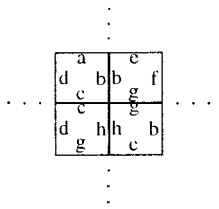
Feladat: Tegyük fel, hogy a g polinom értékei között van két különböző prímszám, p és q. Ekkor van g-nek olyan értéke is, ami pq-val osztható, következésképpen g értékkészlete nem lehet azonos a prímek halmazával. (Tegyük fel, hogy $g(u_1, \ldots, u_m) = p$ és $g(v_1, \ldots, v_m) = q$, ahol $u_i, v_i \in \mathbb{Z}$. Legyen w_i olyan egész, melyre $w_i \equiv u_i \pmod{p}$ és $w_i \equiv v_i \pmod{q}$. Ekkor $g(w_1, \ldots, w_m)$ osztható pq-val.)

7.8.3. A Dominóprobléma

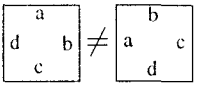
A kirakós játékok nehezek. Aki próbált már mondjuk egy 1500 darabkára gondosan felszabdalt képet összerakni, alighanem egyetért ezzel. Most egy olyan egyszerűen megfogalmazható, kirakó-típusú feladattal ismerkedünk meg, amelynél a megoldhatóság reménytelenül nehéz, pontosabban algoritmikusan eldönthetetlen problémát jelent.

Dominón olyan egységnyi élű négyzet alakú lapot értünk, melynek a négy élére egy-egy jel van írva. Ezekről a jelekről a kényelem kedvéért feltesszük, hogy $\{0,1\}^*$-beli szavak.

A dominó típusát az e jelekből álló rendezett négyes jelenti. Az előző rajz dominójának a típusa (a, b, c, d). A dominókat a jól ismert szabály szerint lehet egymás mellé rakni: az illeszkedő élek mentén azonos jeleknek kell szerepelni.



A dominókat nem szabad forgatni. Az (a, b, c, d) típusú dominót csak úgy tehetjük le, hogy a felső él mentén a legyen.



A Dominóprobléma ezek után a következő: *Adott dominó-típusok egy véges $\mathcal{F}$ halmaza; eldöntendő, hogy a sík lefedhető-e hézagtalanul szabályosan illeszkedő $\mathcal{F}$-beli típusú dominókkal.* Természetesen a fedéshez az egyes típusokból végtelen sok példány használható.

A k típusból álló $\mathcal{F}$ készlet leírható a következő tagozódású F szóval:

$$F := x_1 \# \dots \# x_k \#,$$

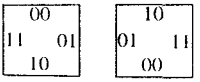
ahol az $x_i = a_i * b_i * c_i * d_i$ szó $(a_i, b_i, c_i, d_i \in \{0,1\}^*)$ az

dominó típusának a leírása.

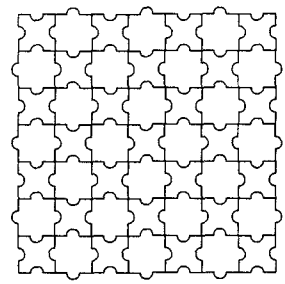
A Dominófeladatnak megfelelő D nyelv az alábbi:

$$D = \{F \in \{0,1,*,\#\}^*;\ \text{a sík lefedhető } \mathcal{F}\text{-típusú dominókkal }\}.$$

Példa: Az $F = 00 * 01 * 10 * 11 \# 10 * 11 * 00 * 01 \#$ szó a következő két típusból álló készlet kódja.



Ezzel a készlettel a sík kirakható, másként mondva $F \in D$. A kétféle dominóval a sakktábla fekete és fehér mezőinek váltakozását követő alakzatban parkettázhatjuk ki a síkot. A megoldáson túl az illeszkedési szabályból adódó kötöttségeket is szemlélteti az alábbi – XV. századi portugál padlómozaikról ellesett – minta:



A Dominóprobléma nem oldható meg algoritmussal. Ezt mondja ki a következő tétel. A terjedelmes, de nem különösebben nehéz bizonyítást mellőzzük.

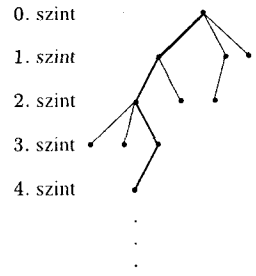
Tétel: *A D nyelv nem rekurzív.* □

Az előző tétel szerint nem lehetséges, hogy D és a komplementere $\{0, 1, *, \#\}^* \setminus D$ is rekurzíve felsorolható legyen. Az utóbbi nyelvről viszont megmutatjuk, hogy rekurzíve felsorolható.

Tétel: *$D \in co\mathcal{RE}$, azaz $\{0, 1, *, \#\}^* \setminus D \in \mathcal{RE}$.*

A tétel bizonyításában hasznunkra lesz egy nevezetes, ám egyszerű tény. Az állítás azzal a kijelentéssel fogalmazható meg szemléletesen, hogy *ha az emberiség sohasem hal ki, akkor létezik olyan dinasztia, amely sohasem szakad meg.* Pontosabban ez így mondható:

Lemma (König-lemma): *Legyen G egy végtelen, gyökeres irányított gráf, amelynek csúcshalmaza a természetes számokkal sorszámozott részekre (szintekre) osztható úgy, hogy:*
(0) A nulladik szinten egyedül a gyökér van. Egy csúcsból csak az eggyel nagyobb sorszámú szintre mehet él.
(1) Minden egyes szinten csak véges sok csúcs van.
(2) Minden – a gyökértől különböző – csúcsnak van őse abban az értelemben, hogy fut bele él az eggyel alacsonyabb sorszámú szintről.
Ekkor a gráfban van végtelen irányított út.



Bizonyítás: Nevezzük a gráf y csúcsát az x csúcs *leszármazottjának*, ha y az x-ből elérhető irányított úton. Jelölje x_0 a gráf gyökerét. A (2) és (0) feltételek miatt a G minden csúcsába vezet út x_0-ból. Mivel a gráf végtelen, ebből arra jutunk, hogy az x_0 gyökérnek végtelen sok leszármazottja van. Ezután indukcióval megadunk egy $y_0, y_1, \ldots, y_i, \ldots$ végtelen G-beli utat. Legyen $y_0 = x_0$. Tegyük fel ezután, hogy az $y_0, y_1, \ldots, y_i$ csúcsokat már kiválasztottuk úgy, hogy y_j a j-edik szintről való, és végtelen sok leszármazottja van $(0 \leq j \leq i)$, továbbá (y_j, y_{j+1}) éle a gráfnak, ha $(0 \leq j < i)$. Ezek a feltételek adják az indukciós feltevést. Ez nyilván teljesül $i = 0$-ra.

Azt kell csak megmutatnunk, hogy a feltétel az y_{i+1} alkalmas választásával örökíthető i-ről $i+1$-re. Ez pedig egyszerű: legyen y_{i+1} az y_i egy olyan gyermeke (az y_i-vel összekötött csúcs az $i+1$. szintről), melynek végtelen sok leszármazottja van. Az indukciós feltevés szerint y_i-nek végtelen sok leszármazottja van. Az (1) kikötés miatt viszont csak véges sok gyermeke lehet; van tehát olyan gyermeke is, amelynek végtelen sok leszármazottja van.

Ilyen választással az $y_0, y_1, \ldots, y_{i+1}$ sorozatra is igaz lesz az indukciós feltevés. A sorozat definíciójával tehát sehol sem akadunk el; $y_0, y_1, \ldots, y_i, \ldots$ tényleg egy végtelen út. □

A tétel bizonyítása: Álljon $L \subseteq \{0, 1, *, \#\}^*$ nyelv azokból a szavakból, amelyek kódjai valamilyen dominókészletnek. Világos, hogy L rekurzív: a helyes felépítésű leírások mindig megálló algoritmussal felismerhetők. Elég ezek után azt igazolni, hogy az $L \setminus D$ nyelv rekurzíve felsorolható.

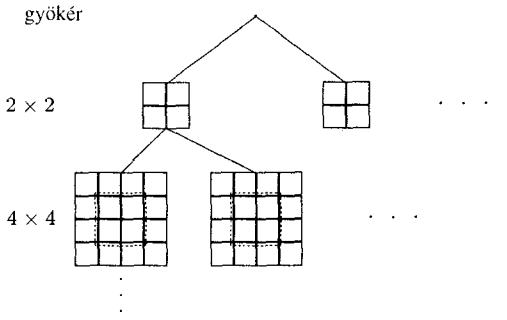
Álljon D^* azon $\mathcal{F}$ dominókészletek leírásaiból, melyekhez van olyan n, hogy már a $2n \times 2n$-es négyzet sem rakható ki $\mathcal{F}$-beli dominókkal. Egyszerű, és mostanra már bizonyára ismerős érvelés mutatja, hogy $D^* \in \mathcal{RE}$. Próbáljuk ugyanis sorban $n = 1, 2, \ldots$-re $\mathcal{F}$-beli dominókkal kirakni a $2n \times 2n$-es négyzetet. Ezt adott n-re – ha máshogy nem – az összes kitöltési lehetőség végignézésével tehetjük meg. Ha van olyan n, melyre a $2n \times 2n$-es négyzet nem parkettázható ki, akkor ezen az úton ez véges sok, bár esetleg csillagászati számú lépés után kiderül. Ekkor megállunk elfogadva az $\mathcal{F}$ készletet leíró F szót.

Ezután a tétel igazolásához elegendő belátni, hogy

$$D^* = L \setminus D.$$

Világos, hogy $D^* \subseteq L \setminus D$, hiszen D^* dominókészletek leírásaiból áll, továbbá D^*-beli készletekkel a sík nem rakható ki, hiszen már egy alkalmas négyzet sem rakható ki.

A fordított irányú $D^* \supseteq L \setminus D$ tartalmazás belátásához elég megmutatni, hogy ha egy $\mathcal{F}$ készlet nincs D^*-ban, akkor nincs $L \setminus D$-ben sem. Egyszerűbben fogalmazva: ha minden n-re a $2n \times 2n$-es négyzet lefedhető az $\mathcal{F}$ készlet dominóival, akkor az egész sík is. Ennek igazolására egy gyökeres, végtelen, szintezett gráfot definiálunk. A gráf csúcsai a négyzetek szabályos kitöltéseinek felelnek meg a következők szerint. A nulladik szinten van a gráf gyökere; ez felel meg az üres négyzet (üres) fedésének. Általában az n-edik szint csúcsai a $2n \times 2n$-es négyzet szabályos kitöltései $\mathcal{F}$-beli dominókkal. Ezzel megadtuk a gráf csúcsait. A $2n \times 2n$-es négyzet egyik kitöltésétől akkor vezet él a $2(n+1) \times 2(n+1)$-es négyzet valamely kitöltéséhez, ha utóbbinak a középső $2n \times 2n$-es négyzetét fedő része megegyezik az előbbivel.



Egyszerű ellenőrizni, hogy a gráf teljesíti a König-lemma feltételeit. A gráf azért lesz végtelen, mert tetszőlegesen nagy négyzetek kirakhatók $\mathcal{F}$-beli dominókkal. A lemma szerint van ebben a gráfban végtelen irányított út. Ez az út pedig az egész sík egy kirakását adja. □

7.8.4. Post megfeleltetési problémája

A következő – *Emil Posttól* származó – eldönthetetlen probléma is egyfajta kirakós feladat, de nem geometriai, hanem nyelvészeti jellegű. Azért szólunk róla, mert felettébb hajlékony eszközt nyújt a matematikai és számítógépes nyelvészet területén egy sereg probléma eldönthetetlenségének a bizonyítására.

Legyen Σ egy véges *abc. Post megfeleltetési problémájának* egy példánya (bemenete) egy (s,t) $(s,t \in \Sigma^*)$ alakú rendezett párokból álló véges $\mathcal{P}$ halmaz. A megfeleltetési feladat $\mathcal{P}$ bemenetét *megoldhatónak* nevezzük, ha vannak olyan (nem feltétlenül különböző) $\mathcal{P}$-beli $(s_1,t_1),(s_2,t_2),\ldots,(s_n,t_n)$ párok úgy, hogy

$$s_1 s_2 \cdots s_n = t_1 t_2 \cdots t_n.$$

A feltétel tehát az, hogy a kiválasztott párok első komponenseinek összeolvasásával adódó szó egyezzen meg a második komponensek egymás után fűzésével kapott szóval. Ilyenkor az $s_1 s_2 \cdots s_n$, vagy ami ugyanaz, a $t_1 t_2 \cdots t_n$ szót a $\mathcal{P}$ *megoldásának* nevezzük. Például a $\mathcal{P} = \{(iz,riz),(kar,ka),(ma,ma)\}$ rendszer megoldható. Egy lehetséges megoldás a *karizma* szó.

Feladat: Mutassuk meg, hogy a $\mathcal{P} = \{(ab,a),(ab,ba),(b,ba)\}$ rendszernek nincs megoldása.

A következő feladatból kitűnik, hogy a megoldáshoz tényleg szükség lehet arra, hogy bizonyos párokat többször is felhasználjunk. Legyen $\mathcal{P}' = \{(aa,aab),(bb,ba),(abb,b)\}$. A $\mathcal{P}'$ bemenet tehát három párból áll. Ugyanakkor igaz a következő.

Feladat: Mutassuk meg, hogy a $\mathcal{P}'$ rendszer megoldható, és a legrövidebb megoldás négy párból illeszthető össze (azaz 4 a legkisebb n, amivel megoldást kaphatunk).

A probléma angol nevéből eredően (Post's correspondence problem) PCP jelöli a megfeleltetési feladat megoldható példányainak leírásaiból álló nyelvet. Megmutatható, hogy nincs algoritmus, ami a megfeleltetési feladat megoldható példányait felismerné. *A PCP nyelv tehát nem rekurzív.* Érvényes másfelől a következő:

Feladat: Mutassuk meg, hogy a PCP nyelv rekurzíve felsorolható. (Tegyük fel, hogy a megfeleltetési feladat $\mathcal{P}$ bemenete k párból áll. Vegyük sorra $(n = 1,2,3,\ldots)$ az n hosszú sorozatokat, melyek tagjai az $1,2,\ldots,k$ sorszámok, és ellenőrizzük, hogy a sorozatnak megfelelő párok adnak-e megoldást.)

7.8.5. Egy nyitott kérdés: a kongruens számok felismerése

Az eldönthetetlenség témakörétől búcsúzva meg szeretnénk jegyezni, hogy nem egy lezárt kutatási területről van szó. Ennek érzékeltetésére egy olyan problémát ismertetünk, amelyre eddig nem találtak algoritmust, és az eldönthetetlenségét sem sikerült bizonyítani. A probléma ártatlanul egyszerűnek tűnik, és bizonyos szempontból igen régi. A fogalom, amihez kapcsolódik, már a X. század arab írástudóinak a munkáiban is megtalálható.

Definíció: *Az m pozitív egész* kongruens szám, *ha van olyan derékszögű háromszög, melynek a területe m, és az oldalai racionális hosszúságúak.*

Megmutatható, hogy 1, 2, 3 és 4 nem kongruens számok. Már e tények igazolása sem túl egyszerű. Kis könnyebbség, hogy a négyesre vonatkozó állítás azonnal következik az egyesről szólóból (és fordítva). Például, ha volna a négyeshez megfelelő derékszögű háromszög, akkor azt a felére kicsinyítve egységnyi területű háromszöget kapnánk.

Feladat: Használva, hogy az $x^4 + y^4 = z^2$ diofantikus egyenletnek nincs csupa pozitív megoldása[4], mutassuk meg, hogy a 2 nem kongruens szám. (Ha 2 kongruens szám volna, akkor léteznének olyan u, v, w pozitív egészek, melyekre $u^2 + v^2 = w^2$, továbbá u és v relatív prímek, és uv négyzetszám. Az utóbbi két feltétel miatt u és v is négyzetszámok.)

Mint azt Leonardo Pisano – ismertebb nevén: Fibonacci – 1220 körül megállapította, az 5 kongruens szám: $3/2$, $20/3$ és $41/6$ a legegyszerűbb olyan racionális derékszögű háromszög oldalai, amelynek a területe 5. A sokszor látott $3, 4, 5$ hármas mutatja, hogy 6 is kongruens szám.

Felmerül a kérdés, van-e általános recept a kongruens számok felismerésére. Ilyet eddig nem sikerült találni. Nem ismert, hogy a kongruens számokból álló nyelv rekurzív-e[5].

Feladat: Mutassuk meg, hogy a kongruens számokból álló nyelv rekurzíve felsorolható. (Legyen n a bemenet. Generáljuk valamilyen sorrendben a pozitív racionális számokból álló (p, r, s) hármasokat. Egy hármasról ellenőrizzük, hogy $p^2 + q^2 = r^2$ és $pq/2 = n$ teljesülnek-e. Ha igen, akkor elfogadjuk n-et. Ellenkező esetben folytatjuk a következő hármassal.)

---

[4] Az $x^4 + y^4 = z^2$ egyenlet ilyen értelmű megoldhatatlanságát Pierre Fermat igazolta.

[5] Az aritmetikai geometria egy nevezetes sejtéséből, a Birch–Swinnerton-Dyer-sejtésből következik, hogy a kongruens számok problémája eldönthető. Ez a hipotézis bizonyos harmadfokú síkgörbék racionális koordinátájú pontjairól szól. Jerrold Tunnell bizonyította, hogy ha a sejtés igaz, akkor a kongruens számoknak van hatékony jellemzése. Ekkor a páratlan négyzetmentes n egész pontosan akkor kongruens szám, ha a $2x^2 + y^2 + 8z^2 = n$ diofantikus egyenletnek kétszer annyi megoldása van, mint a $2x^2 + y^2 + 32z^2 = n$ egyenletnek. Ebből rögvest látszik, hogy legfeljebb $(n+1)^3$ számú (x, y, z) egész vektor vizsgálatával eldönthető, hogy n kongruens-e. Tunnell hasonló jellemzést adott a páros n-ekre is.

7.9. Kolmogorov-bonyolultság

Rosencrantz: *Fej.* (Megismétlik.)
Fej. (Zavart nevetéssel Guildensternre tekint.)
Kezd egy kicsit unalmas lenni, nem?
TOM STOPPARD: Rosencrantz és Guildenstern halott

Itt az információtartalom egy algoritmusokon alapuló megközelítésével foglalkozunk, amit Gregory J. Chaitin, Andrej N. Kolmogorov és Ray J. Solomonoff dolgoztak ki a hatvanas években. Kiindulásul próbáljuk megfogalmazni, hogy egy I^*-beli szó mennyire tömören írható le; másképpen fogalmazva: milyen rövid, milyen kevés jelből álló kóddal nyomható össze optimálisan. Az ilyen fogalomalkotó kísérletek egy lehetséges buktatójára világít rá az *írógép-paradoxon*:

Tekintsük azokat a természetes számokat, amelyeket magyar nyelven legfeljebb 100 billentyűleütéssel definiálni lehet. A billentyűk száma véges, így ezen számok halmaza is véges. Van tehát egy legkisebb természetes szám, amit nem lehet definiálni a fenti módon. De ekkor „az a legkisebb szám, amely nem definiálható magyar nyelven legfeljebb száz billentyűleütéssel" egy száznál rövidebb jelsorozat, tehát olyan számot határoz meg, amely mégis benne van a definiálható számok halmazában, ami nyilvánvaló képtelenség.

A paradoxon arra figyelmeztet bennünket, hogy a „röviden való leírhatóság" csak úgy önmagában még nem lesz használható fogalom. Egy lehetséges kritika, hogy a *magyar nyelven leírhatóság* nem köti meg eléggé a leírás értelmezésének módját. Kössük ki tehát, hogy a leírás értelmezését, más szóval a dekódolást algoritmussal, pontosabban – a Church–Turing-tézis alapján – Turing-géppel végezzük.

Milyen Turing-gépet használjunk erre a célra? Természetes elvárás, hogy a matematikai képlettel tömören leírható számok hatékonyan kódolhatók, jelentősen összenyomhatók legyenek. Például az $n = 2^k - 10$ alakú számok bináris kódja $k = \log_2 n$ hosszúságú, ha $k > 4$. Viszont a $2^k - 10$ kifejezés hossza csak a $2^x - 10$ képlet hosszából valamint k bináris hosszából tevődik össze, ami $\log_2 \log_2 n +$ konstans. Szeretnénk, ha a fogalom tükrözné az ilyen alakú n számok rövid leírhatóságát. Sőt, olyan jellegű számokat is hatékonyan szeretnénk kódolni, mint a k-adik Fibonacci-szám, a k-adik prímszám, a π első k számjegye, stb. A leírást kibontó gépnek ezek szerint „értenie" kell az egyszerű aritmetikai képleteknél általánosabban az algoritmusokat is. Ennek a feltételnek eleget tesznek az univerzális Turing-gépek.

Rögzítsünk tehát egy U univerzális Turing-gépet, és értelmezzük az $x \in I^*$ szó bonyolultságát mint a legrövidebb $y\#z$ input szó hosszát, melyre U az x szót

számítja ki. Az így adódó fogalom meglepően értelmesnek bizonyul. Az U gép választásától nagy mértékben független, és aszimptotikus értelemben jó (konstans eltérésen belüli) közelítését adja az „optimumnak". Például az $n = 2^k - 10$ alakú számok bonyolultsága, mint várjuk, a $2^x - 10$ képlet hosszától eltekintve általában $\log_2 \log_2 n$ lesz. Rövidebb akkor lehet, ha k speciális alakú (k-nak $\log_2 k$-nál rövidebb leírása van). Azt is igazolni tudjuk, hogy a legtöbb esetben k leírása nem lehet $\log_2 k$-nál sokkal rövidebb.

Legyen továbbra is $I = \{0, 1\}$. Olyan Turing-gépekre szorítkozunk, amelyeknek a bemenő *abc*-je I, és a számításaik eredménye is I-beli betűkből áll. Egy ilyen M gép a korábbiakkal összhangban az $f_M : I^* \to I^*$ parciális függvényt számolja ki. Jelöljük $C_M(x)$-szel annak a legrövidebb bemenő szónak a hosszát, mellyel elindítva M az x szót adja eredményül:

$$C_M(x) = \begin{cases} \min\{|y| : y \in I^*,\ f_M(y) = x\} & \text{ha ilyen } y \text{ létezik,} \\ \infty & \text{különben.} \end{cases}$$

A $C_M(x)$ szám méri, hogy x mennyire nyomható össze akkor, ha a kibontást, vagyis az összenyomott szó visszafejtését az M algoritmus végzi: $C_M(x)$ a legtömörebb kódszó hossza, mely az M algoritmus számára x-et kódolja. A $C_M(x)$ érték erősen függ M-től. Az x ismeretében könnyen szerkeszthetünk olyan M_1 gépet, mely az üres bemeneten x-et számolja ki. Olyan M_2 gép is van, amely sosem adja eredményül x-et. Ekkor $C_{M_1}(x) = 0$ és $C_{M_2}(x) = \infty$.

Most megmutatjuk, hogy az univerzális Turing-gépek körében a C érték nem függ ennyire vadul a gép választásától. Ki fog derülni, hogy a kibontó képességet illetően egy univerzális gép semelyik másik gépnél sem lehet számottevően rosszabb. Emlékeztetjük az olvasót, hogy egy U univerzális gép bemenete $w\#s$ alakú, ahol $w, s \in I^*$. Az érdekes esetekben w egy M Turing-gép leírása, amit M_w-vel is jelölünk. Az U gép a $w\#s$ bemeneten utánozza az M_w működését az s inputtal. A továbbiakban feltesszük, hogy az univerzális gépeink is egyszalagosak, és a szalag-*abc*-jük $\{0, 1, ü\}$. Ennek megfelelően az # elválasztó jelet egy véges bitsorozat kódolja.

Tétel (invariancia-tétel): *Legyen U egy univerzális Turing-gép. Ekkor tetszőleges M Turing-gépre létezik egy (csak M-től függő) $c_M \in \mathbb{Z}^+$ állandó, mellyel minden $x \in I^*$ szóra teljesül a következő egyenlőtlenség:*

$$C_U(x) \leq C_M(x) + c_M.$$

Bizonyítás: Tegyük fel, hogy az M gép leírása a $w \in I^*$ szó, és legyen y egy legrövidebb szó, amiből M az x-et bontja ki: $y \in I^*$, $f_M(y) = x$, és $|y| = C_M(x)$. Az univerzális gép definíciója szerint ekkor U a $w\#y$ bemeneten x-et

adja eredményül; más szóval $f_U(w\#y) = x$. Ebből a C_U függvény értelmezése alapján következik, hogy

$$C_U(x) \leq |w\#y| = |w\#| + |y| = |w\#| + C_M(x).$$

A $c_M = |w\#|$ választás tehát megfelel a követelményeknek. □

Az invariancia-tétel szerint a $C_U(x)$ érték csak legfeljebb egy x-től független állandóval haladhatja meg $C_M(x)$-et. Ezt úgy értelmezhetjük, hogy az U-hoz tartozó összenyomhatóság mértéke nem marad el számottevően semelyik másik M gép szerinti összenyomhatóságtól sem.

Következmény: *Legyenek U_1 és U_2 univerzális Turing-gépek, melyek input abcje $I = \{0, 1\}$. Ekkor van olyan $c = c_{U_1,U_2}$ állandó, hogy minden $x \in I^*$ szóra*

$$|C_{U_1}(x) - C_{U_2}(x)| \leq c.$$

Bizonyítás: Alkalmazzuk az invariancia-tételt először az $U = U_1$, $M = U_2$, majd pedig az $U = U_2$, $M = U_1$ szereposztással. Adódik, hogy tetszőleges $x \in I^*$ szóra teljesülnek a $C_{U_1}(x) \leq C_{U_2}(x) + c_{U_2}$ és $C_{U_2}(x) \leq C_{U_1}(x) + c_{U_1}$ egyenlőtlenségek, ahol c_{U_i} az U_i-től függő pozitív állandó. Arra jutunk ezekből, hogy a $c = \max\{c_{U_1}, c_{U_2}\}$ értékkel $|C_{U_1}(x) - C_{U_2}(x)| \leq c$ teljesül. □

Az előző szerint $C_U(x)$ nem függ nagyon erősen az U univerzális gép választásától. Ha U' egy másik univerzális gép, akkor a $C_U(x)$ és $C_{U'}(x)$ eltérése egy x-től független korláton belül marad. Ez alapot ad arra, hogy egyetlen – mostantól fogva rögzített – U univerzális gép segítségével tanulmányozzuk az összenyomhatóságot.

Definíció (Kolmogorov-bonyolultság):
Legyen $x \in I^*$. A $C(x) := C_U(x)$ mennyiség az x szó Kolmogorov-bonyolultsága.

Egy függvény definícióját látva az embernek olykor kedve támad, hogy kiszámolja néhány helyettesítési értékét. Mi lesz például $C(0010)$? Kíváncsiságunk mindjárt akadályba ütközik: az U-t nem adtuk meg elég pontosan ahhoz, hogy a meghatározást követve számolhassunk vele. Rövidesen kiderül, hogy $C(x)$ kiszámítása nehéz feladat; a C függvény nem rekurzív. Mihez lehet hát kezdeni egy ilyen függvénnyel, amelynek az értékei és számolhatósága körül ennyi gond van? A válasz az, hogy $C(x_n)$ alakú sorozatok növekedési rendjét érdemes vizsgálni, ahol $x_1, x_2, \ldots$ növekvő hosszúságú I^*-beli szavak sorozata. Például az $n = 2^k - 10$ alakú számokra $C(n) \leq \log_2 \log_2 n + c'$ teljesül alkalmas c' állandóval. Az egyenlőtlenség bal oldalán $n \in I^*$ az n szám szokásos bináris kódját jelöli. Az egyenlőtlenség az invariancia-tétel egyszerű következménye. Van ugyanis

olyan algoritmus, amely a binárisan írt, és ezért legfeljebb $\log_2 \log_2 n + 1$ hosszúságú k-ból n-et számítja ki. A megfelelő M géppel tehát $C_M(n) \leq \log_2 \log_2 n + 1$, és innen az invariancia-tétel adja a $C(n)$-re vonatkozó állítást.

Az invariancia-tétel további egyszerű következménye, hogy egy I^*-beli szó Kolmogorov-bonyolultsága nem haladja meg lényegesen a szó hosszát:

Következmény: *Legyen $x \in I^*$. Ekkor $C(x) \leq |x| + k$, ahol k egy x-től független állandó.*

Bizonyítás: Jelölje M azt a Turing-gépet, amely az inputot érintetlenül hagyva egy lépésben megáll. Nyilván $f_M(x) = x$, sőt $C_M(x) = |x|$ is teljesül bármely $x \in I^*$ szóra. Az M gépre alkalmazhatjuk az invariancia-tételt; éppen a kívánt egyenlőtlenséget kapjuk. □

A fordított irányt illetően megmutatjuk, hogy a szavak túlnyomó többségének a Kolmogorov-bonyolultsága közel van a szó hosszához. Vannak olyan szavak is, amelyek nem írhatók le rövidebben, mint a saját hosszuk. Ezeket külön névvel illetjük:

Definíció: *Az $x \in I^*$ szó* összenyomhatatlan, *ha $C(x) \geq |x|$.*

Tétel: *Legyen $k \in \mathbb{Z}^+$. Legfeljebb $2^{k+1} - 1$ $x \in I^*$ szó van, melyre $C(x) \leq k$. Következésképpen minden $n \geq 1$ egészre létezik n hosszúságú összenyomhatatlan szó. Ha $n > 8$, akkor az n hosszú I^*-beli szavak több, mint 99 százalékának a Kolmogorov-bonyolultsága nagyobb, mint $n - 8$.*

Bizonyítás: Legyen

$$H_k = \{x \in I^* : C(x) \leq k\}.$$

Ha $x, x' \in H_k$, akkor a Kolmogorov-bonyolultság definíciója szerint vannak olyan $y, y' \in I^*$ szavak, melyekre $f_U(y) = x$, $f_U(y') = x'$, továbbá $|y| \leq k$ és $|y'| \leq k$. Nyilvánvaló, hogy ha $x \neq x'$, akkor $y \neq y'$, hiszen U egy bemenetéhez nem tartozhat két különböző eredmény. A H_k halmaznak eszerint legfeljebb annyi eleme lehet, mint ahány k-nál nem hosszabb szó van I^*-ban. Az utóbbi szavak száma pedig $1 + 2 + \cdots + 2^{k-1} + 2^k = 2^{k+1} - 1$. Ezzel az első állítást igazoltuk.

A második a $k = n - 1$ választással azonnal adódik az elsőből: az n hosszú szavak száma 2^n, a H_{n-1} halmazban pedig legfeljebb $2^n - 1$ szó van. Létezik tehát olyan n hosszú szó, ami nincs H_{n-1}-ben; ez egy összenyomhatatlan szó.

Hátra van még az utolsó állítás. A H_{n-8} halmaznak legfeljebb $2^{n-7} - 1 < 2^{n-7}$ eleme van. A kedvezőtlen esetek aránya az n hosszú szavak között így legfeljebb $2^{n-7}/2^n = 1/128$, ami kisebb, mint $1/100$. □

Feladat: Legyen $\omega : \mathbb{Z}^+ \to \mathbb{Z}^+$ egy végtelenhez tartó függvény, azaz $\lim_{n\to\infty} \omega(n) = \infty$ teljesüljön. Igazoljuk, hogy ekkor

$$\lim_{n\to\infty} \frac{|\{x \in I^* : |x| = n,\ C(x) > n - \omega(n)\}|}{|\{x \in I^* : |x| = n\}|} = 1.$$

Vagyis „majdnem minden" x szóra igaz a $C(x) > |x| - \omega(|x|)$ egyenlőtlenség. (Mutassuk meg, hogy a $|H_{n-\omega(n)}|/2^n$ hányados nullához tart.)

Egy tipikus szó Kolmogorov-bonyolultsága tehát közel van a hosszához. Ennek furcsa ellenpontjaként említhetjük, hogy mégsem tudunk minden n-re egy olyan n hosszú x_n szót *algoritmussal előállítani*, amelyre mondjuk $C(x_n) \geq n/2$. Ha volna ugyanis egy M Turing-gép, ami a binárisan írt n inputra egy ilyen x_n szót adna, akkor $C_M(x_n) \leq \log_2 n + 1$, és az invariancia-tétel alapján $C(x_n) \leq \log_2 n + c$ teljesülne. Az utóbbi mennyiség viszont elég nagy n-re kisebb lesz, mint $n/2$. Ez pedig ellentmond a $C(x_n) \geq n/2$ feltételnek. Ezek után már az sem meglepő, hogy a Kolmogorov-bonyolultságot nem lehet kiszámítani:

Tétel: *A $C : I^* \to I^*$ függvény nem rekurzív.*

Bizonyítás: A szokásos bináris ábrázolás segítségével a $\mathbb{Z}^+$ halmazt I^* részének tekinthetjük. Legyen ezután $F : \mathbb{Z}^+ \to I^*$ a következő függvény:

$$F(m) := \min\{x \in I^*; C(x) \geq m\},$$

ahol a minimumot az I^*-beli szavak kanonikus rendezése szerint értjük. Az F minden természetes számra értelmezett, mert létezik tetszőlegesen nagy Kolmogorov-bonyolultságú szó. Tegyük fel indirekte, hogy C rekurzív. Ezt a feltevést használva megmutatjuk, hogy F parciálisan rekurzív, azaz van olyan M Turing-gép, melyre $F = f_M$. Az M algoritmus munkája az m bemenetnél felettébb egyszerű: vesszük az $x \in I^*$ szavakat a kanonikus sorrendben, és kiszámítjuk a $C(x)$ értéket. Ezen a ponton aknázzuk ki a C rekurzivitását. Kiírjuk az első olyan x szót, amelyre $C(x) \geq m$, és megállunk. Mármost az F definíciója szerint

$$m \leq C(F(m)).$$

Másrészt az invariancia-tétel alapján alkalmas c-vel igaz, hogy

$$C(F(m)) \leq C_M(F(m)) + c.$$

Ugyanakkor a C_M értelmezése szerint, ha $m > 0$, akkor

$$C_M(F(m)) \leq \log_2 m + 1,$$

hiszen M a binárisan ábrázolt m inputból előállítja $F(m)$-et. A kapott egyenlőtlenségeket összerakva $m > 0$ esetén arra jutunk, hogy

$$m \leq \log_2 m + 1 + c,$$

ami elég nagy m-re lehetetlen. Az így nyert ellentmondás bizonyítja, hogy C nem rekurzív. □

A Kolmogorov-bonyolultság néhány alkalmazása

A Kolmogorov-bonyolultság önmagában is érdekes fogalom, de ezenfelül kitűnik még a sokoldalú, színes alkalmazási lehetőségeivel. Eme alkalmazások közül itt hármat említünk.

1. A Megállási probléma eldönthetetlensége

A Kolmogorov-bonyolultság tulajdonságait használva egyszerűen igazolható, hogy az L_h megállási nyelv eldönthetetlen. Tegyük fel ugyanis indirekte, hogy létezik egy az L_h nyelvet felismerő, mindig megálló M algoritmus. Megmutatjuk, hogy az M segítségével a C függvény kiszámítható, ami ellentmond az előző tételnek.

Legyen $x \in I^*$. A $C(x)$ érték az (egyik) legrövidebb olyan $y \in I^*$ szó hossza, melyre $f_U(y) = x$. Itt nyilván elegendő az olyan y szavakra szorítkozni, melyekkel mint bemenetekkel az U valamikor megáll. A $C(x)$ meghatározásához tehát a kanonikus sorrendben vesszük az $y \in I^*$ szavakat. Egy y szóra először az M algoritmussal ellenőrizzük, hogy U megáll-e az y inputtal. Ha a válasz nemleges, akkor eldobjuk y-t, és vesszük a következő szót. Ha a válasz igenlő, akkor az y-nal elindítjuk az U-t. Az U munkájának végeztével ellenőrizzük, hogy az $f_U(y)$ eredmény egyenlő-e x-szel. Ha nem, akkor a kanonikus sorrend szerint következő y-nal próbálkozunk. Az első olyan y szónál állunk meg, amelyre $f_U(y) = x$, és kiírjuk az y hosszát. Ilyen y létezik, és a vázolt recepttel véges sok lépésben meg is találjuk. □

Feladat: Mutassuk meg, hogy a

$$\{x\#n : \ x \in I^*, \ n \in \mathbb{Z}^+, \ C(x) \leq n\}$$

nyelv rekurzíve felsorolható, de nem rekurzív. (Az első állításhoz a $(\mathbb{Z}^+)^2$-beli párok algoritmikus felsorolása használható. A másodikhoz mutassuk meg, hogy ha a nyelv rekurzív lenne, akkor a C függvényt ki tudnánk számítani.)

2. Palindrómák felismerése egyszalagos Turing-géppel

Korábban láttuk, hogy egy k szalagos M Turing-gép szimulálható olyan egyszalagos N géppel, melynek az időfüggvénye legfeljebb az M időfüggvényének négyzetével arányosan nő: $T_N(n) = O(T_M^2(n))$. Most megmutatjuk, hogy ez a kvadratikus becslés éles: egy olyan nyelvet ismertetünk, mely kétszalagos géppel lineáris (azaz $O(n)$) időben felismerhető, egyszalagos géppel viszont nem ismerhető fel $o(n^2)$ lépésen belül.

Egy szót *palindrómának* vagy *tükörszónak* nevezünk, ha visszafelé olvasva is magát a szót kapjuk. Például a *kajak* egy tükörszó. Az egyik legnevezetesebb palindróma a Napóleon sírkövén levő felirat: ABLE WAS I ERE I SAW ELBA.

Legyen L_p az I^*-beli palindrómákból álló nyelv. Könnyen látható, hogy kétszalagos Turing-géppel egy n hosszú $w \in I^*$ szónak az L_p-be tartozása $O(n)$ lépésben eldönthető: először felmásoljuk a w input szót a második szalagra, majd az első fejjel visszamegyünk a szalag elejére. Végül a két fejjel ellentétes irányban haladva ellenőrizzük, hogy a tükrösen elhelyezkedő jelek egyeznek-e. Egyszalagos géppel már nem megy ilyen gyorsan:

Tétel: *Nincs olyan egyszalagos M Turing-gép, amely az L_p nyelvet ismeri fel, és amelynek időfüggvényére $T_M(n) = o(n^2)$ teljesül.*

Először megpróbálunk szemléletes képet adni a bizonyításban szereplő gondolatokról. Legyen M egyszalagos Turing-gép, amelynek a nyelve L_p és nézzük M munkáját egy n hosszú $s \in I^*$ input szón. Osszuk három egyenlő részre a szalagnak a bemenetet tartalmazó részét. Az ($s \in L_p$?) megválaszolásához a gépnek valahogy össze kell vetnie az s első harmadát az utolsóval. Az M konstans mennyiségű – mondjuk b bit – információt tud a belső állapotaiban tárolni, és ezért a fej mozdításával egy hellyel arrébb vinni. Az M munkáját úgy képzeljük el, hogy az s első harmadából való információdarabokat vet össze az s végén található párjukkal. Az összehasonlítás elvégzéséhez viszont a gépnek az egyik darabkát el kell vinnie a másikhoz, ami legalább $n/3$ lépést jelent. A gép így összesen legalább $n/3$ bit információt szállít el legalább $n/3$ cellányira. Egy lépésben b bitet képes egy hellyel arrébb vinni. Tehát az $n/3 \cdot n/3 = n^2/9 \cdot bit \cdot cella$ fuvar teljesítéséhez legalább $n^2/(9b)$ lépés kell.

Hogy mindezeket pontossá tegyük, meg kell fogalmaznunk, *mi* is az információ, amit M szállít. Erre szolgál az átkelőnapló. Legyen c az M gép egy szalagcellája. Tekintsük M futását az $s \in I^*$ bemeneten, és jegyezzük fel sorra az M állapotát azokban a pillanatokban, amikor a fej átlépi a c cella jobb oldali határát. Az így adódó $q_1, q_2, \ldots, q_m$ sorozat a c cella *átkelőnaplója*. Az első átlépéskor (ez szükségképpen jobbra lépés) M a q_1 állapotban van, a másodiknál (ami balra lépés) q_2, stb. Mint a bizonyításban látni fogjuk, az átkelőnapló tényleg úgy vehető,

mint a c határán átmenő információ hű jegyzőkönyve. Világos, hogy az átkelőnapló $m|Q|$ bittel leírható (Q az M belső állapotainak halmaza). Az átjutó információ *mennyiségének* mérésére pedig a Kolmogorov-bonyolultság lesz alkalmas. A következő takaros érvelés Wolfgang J. Paultól származik.

A tétel bizonyítása: Tegyük fel, hogy M egyszalagos Turing-gép, melyre $L_M = L_p$. Elég igazolni, hogy az $n = 3k$ hosszúságú bemeneteken M legalább dn^2 lépést tesz, ahol $d > 0$ állandó, és n elég nagy. Feltehetjük, hogy megálláskor M feje az első cellára mutat; ez a lépésszám duplázása árán elérhető.

Nézzük M működését egy $s = w1^k w^*$ alakú bemeneten, ahol w egy k hosszúságú összenyomhatatlan szó, és w^* jelöli a w szó fordítottját. Az s egy n hosszúságú palindróma.

Tekintsük az s középső harmadát tartalmazó cellák átkelőnaplóit. Ha ezek mindegyike több, mint $n/(10|Q|)$ bejegyzést tartalmaz, akkor M ezen a bemeneten több, mint $n/(10|Q|) \cdot n/3 = n^2/(30|Q|)$ lépést tett. Ekkor készen vagyunk; az ilyen bemeneten M tényleg nem végezhet $o(n^2)$ lépésben. Feltehető tehát, hogy van olyan c cella a középső harmadban, amelynek az átkelőnaplójában legfeljebb $n/(10|Q|)$ állapot szerepel. Megmutatjuk, hogy ez ellentmondáshoz vezet, ha n elég nagy.

Pontosabban azt igazoljuk, hogy *van olyan M' algoritmus, amely az n számból, a c cella i sorszámából és a c-hez tartozó átkelőnapló leírásából rekonstruálja a w szót.* M'-nek ez a bemenete leírható $2\log_2 n + 2 + n/10$ bittel. Ebből tehát $C_{M'}(w) \leq 2\log_2 n + 2 + n/10$ és az invariancia-tétel alapján $C(w) \leq 2\log_2 n + 2 + n/10 + c_{M'}$ következne. Kellően nagy n-re ez tényleg ütközik a $C(w) \geq k = n/3$ feltevéssel.

Elegendő ezután a dőlt betűs állítást bizonyítani. Az M' algoritmus sorra veszi a k hosszúságú $x \in I^*$ szavakat. Egy ilyen szóval a következőket teszi:
(1) Felírja M szalagjának elejére az $x1^{i-k}$ szót.
(2) Szimulálja M lépéseit, amíg az nem lép túl a c cellán.
(3) Ha M megáll elutasító állapotban, akkor M' az (1)-től újrakezdi a következő x szóval.
(4) Ha M elfogadó állapotban áll meg, akkor M' kiírja az x szót, és megáll.
(5) Ha M feje jobbra el akarja hagyni c-t, akkor ellenőrzi, hogy M belső állapota megegyezik-e az átkelőnaplóban következővel (ez kezdetben q_1). Ha nem, akkor újra kezdi a munkát a következő x szóval.
(6) Ha az állapotok egyeznek, akkor M' átugorja M-nek a c-n túli működését. Ez annyit tesz, hogy M-et az átkelőnapló szerint következő (ez először q_2 lehet) állapotba teszi, a fejét a c cellára állítja. Ezután (2) szerint folytatja a szimulációt.

A lényeges észrevétel az, hogy ha az (5)-beli teszteknél mindig *igen* a válasz, akkor a szimulációnál az első i cellán pontosan ugyanaz történik, mint ami akkor

történne, ha M-et az $x1^k w^*$ szóval indítanánk el. Ez közvetlenül adódik a Turing-gép definíciójából: a c-től jobbra eső részen a működés csak az átlépő állapoton keresztül függ a szalag elején történtektől. Ugyanígy a szalag elején a c-től jobbra levő munka csak a visszatérő állapoton keresztül tud hatni.

Az $x = w$ szó nyilván minden tesztet túlél. M éppen a palindrómákat fogadja el, ezért a szimuláció pontosan akkor áll meg M elfogadó állapotánál, ha $x = w$. Az M' algoritmus tehát tényleg előállítja a w szót. Ezzel a bizonyítás teljes. □

3. Véletlen sorozatok

A Kamra színházban vagyunk. Kezdődik a *Rosencrantz és Guildenstern halott*. Kihunynak a fények, minden sötét. Valami nesz hallható, aztán valaki – később megtudjuk, Rosencrantz az – azt mondja, hogy *fej*. Majd megint nesz, és megint *fej*. Ez többször ismétlődik. Egy idő után a közönség felismeri az „algoritmust", enyhe derültség, és az egyik pénzkoppanás után már a nézők is súgják, hogy *fej*. A publikum érzékeli, hogy nagyon nem szokványos, ami történik.

Ha pénzdarabot dobálunk fel, és mondjuk ötvenszer egymás után *fej* adódik, mint ahogy ez Stoppard darabjában történik, akkor úgy érezzük, hogy ez igen különös, és egyáltalán nem véletlenszerű. Egy véletlen sorozatot kuszának, szabályosságok nélkülinek képzelünk. A valószínűségszámítás jól ismert alapfogalmai viszont nem adnak módot ilyesfajta különbségtételre. A csupa fej sorozat ugyanolyannak látszik, mint bármelyik másik sorozat, hiszen minden egyes sorozat ugyanazzal a 2^{-50} eséllyel fordul elő.

A Kolmogorov-bonyolultság talán legérdekesebb vonása, hogy segítségével értelmesen definiálható a véletlen sorozat. A kapott fogalom igen határozottan megkülönbözteti a szabályos sorozatokat a kuszáktól. A többféle lehetséges meghatározás közül itt csak egyet ismertetünk. Jelölje I^∞ a végtelen 0-1 sorozatok halmazát. Egy $x \in I^\infty$ sorozatra legyen x_n az x első n bitjéből álló szó.

Definíció (Kolmogorov-véletlen sorozat): *Az $x \in I^\infty$ sorozat egy* véletlen sorozat, *ha* $\lim_{n\to\infty} \frac{C(x_n)}{n} = 1$.

Egy sorozatot tehát akkor tekintünk véletlennek, ha a kezdőszeletei véges sok kivétellel nagy Kolmogorov-bonyolultságúak (közel összenyomhatatlanok). Nevezzünk egy $x \in I^\infty$ sorozatot algoritmussal előállíthatónak, ha van olyan Turing-gép, amely a binárisan megadott n bemeneten éppen az x_n szót adja eredményül.

Állítás: *Ha az $x \in I^\infty$ sorozat algoritmussal előállítható, akkor x nem véletlen sorozat.*

Bizonyítás: Legyen M egy az x-et előállító Turing-gép. Ekkor $C_M(x_n) \leq \log_2 n + 1$. Az invariancia-tétel szerint $C(x_n) \leq \log_2 n + 1 + c_M$, amiből $\lim_{n\to\infty} \frac{C(x_n)}{n} = 0$. □

A „csupa fej" sorozat tehát nem véletlen sorozat. A fogalom sok más értelemben is megfelel a véletlenről meglevő intuitív képünknek. Igazolható például, hogy ha x egy véletlen sorozat, akkor x_n-ben körülbelül ugyanannyi nulla van, mint egyes.

A véletlen sorozat definíciójával kapcsolatban nem árt némi óvatosság. Voltak olyan nevezetes kísérletek, amelyekről később kiderült, hogy túl szigorúak abban az értelemben, hogy egyetlen sorozat sem teljesíti az előírt kikötéseket. Itt most nem ez a helyzet. Sőt, az is igaz, hogy I^∞ majdnem minden eleme véletlen sorozat.

Tétel: *Tegyük fel, hogy az $x \in I^\infty$ sorozat bitjeit egymástól függetlenül $1/2$ valószínűséggel választjuk. Ekkor x 1 valószínűséggel véletlen sorozat lesz.*

Bizonyítás: A $C(x_n) \leq n + c$ egyenlőtlenségek miatt ha x nem véletlen, akkor van olyan $\epsilon > 0$, hogy $C(x_n)/n < 1-\epsilon$ teljesül végtelen sok n-re. Legyen $k \in \mathbb{Z}^+$ és

$$S_k = \{x \in I^\infty;\ \text{van végtelen sok } n, \text{melyre } C(x_n)/n < 1 - 1/k\}.$$

Ha x nem véletlen, akkor az előzőek szerint $x \in S_k$ alkalmas k-ra. Innen

$$Prob(x \text{ nem véletlen}) \leq \sum_{k=1}^{\infty} Prob(x \in S_k).$$

Elég tehát megmutatni, hogy a $Prob(x \in S_k)$ valószínűségek mind nullák. Legyen ezután k egy rögzített pozitív egész, és jelölje A_n azt az eseményt, hogy $C(x_n)/n < 1 - 1/k$. Világos, hogy $x \in S_k$ egyenértékű azzal, hogy az A_n események közül végtelen sok következik be. A Borel–Cantelli-lemma [6] alapján elég belátni, hogy a $\sum_{n=1}^{\infty} Prob(A_n)$ sor konvergens. A $Prob(A_n)$ mennyiség becsléséhez először megjegyezzük, hogy a korábbiak szerint kevesebb, mint $2^{(1-1/k)n+1}$ szó Kolmogorov-bonyolultsága lehet kisebb, mint $(1 - 1/k)n$. Ezt használva

$$Prob(A_n) < \frac{2^{(1-1/k)n+1}}{2^n} = 2 \cdot 2^{(1-1/k)n-n} = 2(2^{-1/k})^n = 2(1/\sqrt[k]{2})^n.$$

A $\sum_{n=1}^{\infty} Prob(A_n)$ sor majorálható egy konvergens geometriai sor kétszeresével. A bizonyítás ezzel teljes. □

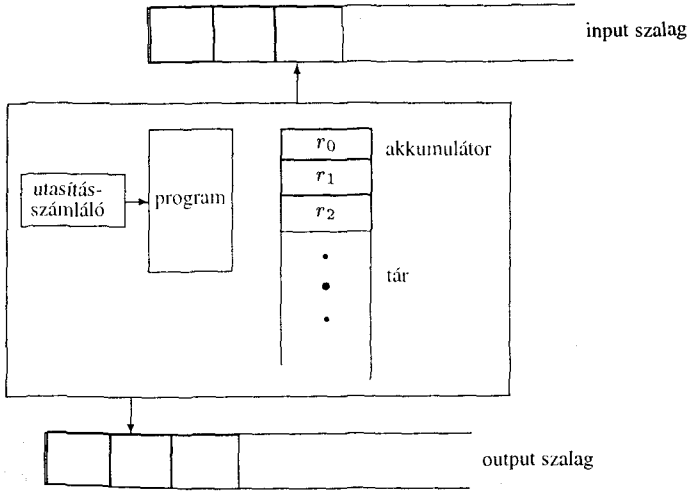
[6] A Borel–Cantelli-lemma szerint ha $A_1, A_2, \ldots$ olyan események egy valószínűségi mezőből, amelyekre $\sum_{n=1}^{\infty} Prob(A_n)$ konvergens, akkor 0 a valószínűsége annak a B eseménynek, hogy az A_i események közül végtelen sok bekövetkezik. A lemma azonnal adódik abból a tényből, hogy tetszőleges n-re a B benne van az $A_n, A_{n+1}, \ldots$ események egyesítésében, tehát $Prob(B) \leq \sum_{l=n}^{\infty} Prob(A_l)$, és a jobb oldali összeg n választásával tetszőlegesen kicsivé tehető.

7.10. A közvetlen elérésű gép (RAM)

Itt egy olyan gépmodellt tárgyalunk, mely a Turing-gépeknél sokkal jobban hasonlít a szokásos számítógépekre. A RAM rövidítés az angol *random access machine* elnevezés kezdőbetűiből származik. A gép nevében a *közvetlen elérés*, illetve a *random access* arra utal, hogy a memóriája a tömbökhöz hasonlóan egyetlen elemi lépéssel elérhető cellákból áll. A modell bevezetésével több célunk is van. Először is: a közvetlen elérésű gép a Turing-gépeknél sokkal kényelmesebb eszköz, ha algoritmusokat akarunk tervezni. Másfelől segítségével pontosan, ugyanakkor eléggé valósághűen definiálható a számítások időköltsége. Végezetül pedig a modell fontos adalékot szolgáltat a Church–Turing-tézishez. Megmutatjuk, hogy a közvetlen elérésű gépek szimulálhatók Turing-gépekkel.

A közvetlen elérésű gép alkotórészei a következők: egy kizárólag olvasásra használható *input szalag*, egy csak írható *output szalag*, a *belső tár* és a *programtár*. A szalagok és a belső tár cellákból állnak és (egyirányban) végtelen hosszúak. Egy cella egy tetszőleges egész számot tartalmazhat. A belső tár celláit a természetes számokkal sorszámozottaknak tekintjük. Kitüntetett a nulladik cella, az *akkumulátor*, ami az elemi műveletek egyik operandusát, és általában az eredményét is tartalmazza.

A programtárban található a gép *programja*. A program sorszámozott – szokásos kifejezéssel: *címkézett – utasítások* véges sorozata. Az *utasításszámláló* egy mutató, ami az éppen végrehajtandó utasításra mutat. A gép – értelemszerűen – az input szalagról veszi a számítás bemenő adatait; az eredmény pedig az output szalagon jelenik meg.



A következő táblázat a közvetlen elérésű gép utasításait tartalmazza. Ez egyike csupán a lehetséges és értelmes utasításkészleteknek. Az irodalomban többféle változattal találkozhat az olvasó. A táblázatban az utasítás neve után az utasítás paramétere szerepel.

Aritmetika		*Adatmozgatás*		*Vezérlés*	
ADD	*op*	LOAD	*op*	JUMP	*címke*
SUB	*op*	STORE	*op*	JGTZ	*címke*
MULT	*op*	READ	*op*	JZERO	*címke*
DIV	*op*	WRITE	*op*	HALT	

Itt *op* az adott utasítás (egyik) operandusa. Az operandusok háromfélék lehetnek. Jelöljön i egy egész számot. Az $= i$ alakú operandus – az ún. közvetlen operandus – az i egészet jelenti. Az i operandus a belső tár i-edik cellájának a tartalmát jelenti. A $*i$ formájú operandus az indirekt címzés eszköze. Jelentése az i-edik tárcellában található sorszámmal azonosított cella tartalma. Tehát ha például $r[0] = -4$ és $r[1] = 0$, akkor a $*1$ operandus jelentése -4. Az i és $*i$ alakú hivatkozásokban i szükségképpen nemnegatív egész. (A hibák kezelésével kapcsolatos kérdésektől eltekintünk.)

Az aritmetikai műveletek az összeadás (ADD), kivonás (SUB), szorzás (MULT) és osztás (DIV) első argumentuma az akkumulátor, a másodikat az *op* paraméter adja. A művelet eredménye az akkumulátorba kerül. Például ha $r[0] = 17$ és $r[1] = 2$, akkor DIV 1 hatására a $8 = \lfloor 17/2 \rfloor$ érték kerül az akkumulátorba.

A READ *op* utasítás hatására az aktuális input cella tartalma az *op* paraméter által leírt tárcellába kerül. Az input szalagon az utasítás végrehajtása után a fej egy cellányit jobbra lép. A WRITE *op* az aktuális output cellába írja az *op*-pal azonosított egészet. Itt *op* közvetlen operandus is lehet. A fej az írás után egy hellyel jobbra lép az output szalagon. A STORE *op* utasítás hatására az akkumulátor tartalma az operandus által megadott (esetleg indirekt) című tárcellába kerül; közvetlen operandus itt nem használható, mert nem jelöl címet. A LOAD *op* utasítás a fordított irányú adatmozgatásra szolgál; itt megengedett a közvetlen (konstans) operandus is.

A HALT utasítás hatására a program megáll. A további három vezérlést szabályozó utasítás ugró utasítás. Ha az akkumulátor tartalmára teljesül az ugrási feltétel, akkor a program a *címke* által megadott sorszámú utasítással folytatódik. Minden más esetben a programtár következő utasításával folytatódik a munka. Az ugrási feltételek a következők: JZERO: $r[0] = 0$, JGTZ: $r[0] \geq 0$, JUMP: üres – azaz mindig teljesül.

A RAM-modell két értelemben idealizálja a szokásos egyprocesszoros architektúrákat: nincs megkötés a tár, illetve a szalagok méretére. Ezenfelül nincs „szó-

hossz", azaz egy cella tetszőlegesen széles egész számot tartalmazhat. Egyebekben a modell megfelel az assemblerben programozható és némi perifériával kiegészített processzornak.

Költségszámítás

A RAM-programok időigényének mérésére kétféle mérőszámot szokás alkalmazni. Az egyik az *uniform költség*. Egy program uniform költsége a futása során végrehajtott utasítások száma. Úgy is mondhatjuk, hogy minden elemi utasítás költsége 1. A rendezés, a keresés és a gráfalgoritmusok kapcsán lényegében ezt a mérési módot használtuk. Becsléseink a módszerek uniform költségére adtak nagyságrendi korlátokat.

Az uniform költség használatakor nem árt egy kis körültekintés. Mindenféle huncutságot el lehet követni úgy, hogy óriási méretű számokkal dolgozunk. Így alacsony költséget kaphatunk olyan számításokra is, amelyek gyakorlati megvalósítása igen drága. Egyszerű példaként tekintsük az $f(n) = 3^{2^n}$ függvényt. Az $f(n)$ érték kiszámítható lineáris, azaz $O(n)$ uniform költséggel: legyen $x := 3$, majd n-szer iterálva $x := x^2$. Ugyanakkor vegyük észre, hogy a k-adik iterációs lépésben két 2^k-jegyű számot szorzunk össze. Az eredménynek tehát 2^{n+1} jegye lesz. Ha mondjuk $n = 1000$, akkor 1000 elemi műveletet végzünk, a 2^{1001} bitből álló eredmény leírásához viszont a világ minden papírja sem elegendő.

A *logaritmikus költség* kiküszöböli ezt a fogyatékosságot, amennyiben érzékeny a számításban szereplő adatok méretére is. Ennél a számolási módnál egy utasítás költsége a benne szereplő adatok összhossza. Egy egész szám hossza a bináris jegyeinek száma +1, hogy az előjelre is tekintettel legyünk. Az adatok nélküli utasítások (JUMP, HALT) költsége 1. Egy program logaritmikus költsége a végrehajtott utasítások költségeinek az összege.

Példa: ADD $*1$ uniform költsége 1. A logaritmikus költsége viszont

$$\text{hossz}(r[0]) + \text{hossz}(r[1]) + \text{hossz}(r[r[1]]) + \text{hossz}(1),$$

ahol $r[i]$ a belső tár i-edik cellájának a tartalmát jelöli.

A logaritmikus költség a valósághoz hű mérőszámot ad olyan programok esetén, amelyek nem tartalmaznak MULT, illetve DIV utasításokat. Ennek az alapja az, hogy a többi elemi művelet ténylegesen megvalósítható a benne szereplő adatok hosszával arányos időköltséggel. Például az összeadás és a kivonás esetén az iskolában tanult táblázatkitöltő módszer ilyen tulajdonságú. A szorzásra és az osztásra viszont mindeddig nem találtak olyan algoritmust, ami n bites bemenetek

esetén n-nel arányos számú bit-művelettel megadná az eredményt[7].

A két költségfogalmat összevetve megállapíthatjuk, hogy az uniform költséget könnyebb számolni, becsülni, a logaritmikus költség pedig pontosabb, valóságosabb mérőszámot ad. Az uniform költséget akkor értelmes használni, ha tudunk valami korlátot a számítás során keletkező adatok méretére. Ha egy RAM-program végig legfeljebb l hosszúságú adatokkal dolgozik, és az uniform költsége m, akkor a logaritmikus költsége $O(lm)$.

Az igazi számítógépek processzorai szó-szervezésűek. Ez egyebek közt azt jelenti, hogy a szóhossz korlátozza az elemi műveletek operandusainak hosszát. Másfelől az alkalmazások tekintélyes részénél a természetesen adódó elemi adatok és az ezekből nyert részeredmények belefférnek egy gépi szóba. Az ilyen esetekben az uniform költség, vagyis az elemi műveletek száma használható mérőszámot ad.

Szimulációk

A következő tétel azt állítja, hogy a közvetlen elérésű gépek és a Turing-gépek számító ereje megegyezik. Ezzel adalékot szolgáltat a Church–Turing-tézishez. Másfelől a RAM-ok csiszoltabb architektúrájuknak köszönhetően gyorsabbak. A hatékonyságbeli nyereség azonban egy négyzetes korláton belül marad. A tétel lehetővé teszi, hogy ha hatékony algoritmust akarunk készíteni, akkor a programozás természetéhez közelebb álló RAM-modellt használjuk.

Tétel (Turing-gép↔RAM szimulációk):
(1) Tetszőleges M Turing-gép szimulálható $O(T_M(n)\log T_M(n))$ logaritmikus költségű RAM programmal. A szimuláció uniform költsége $O(T_M(n))$.
(2) Egy $t(n)$ logaritmikus költségű, MULT és DIV utasításokat nem tartalmazó RAM-program szimulálható olyan N Turing-géppel, melynek az időigényére $T_N(n) = O(t^2(n))$ teljesül.

Bizonyítás: (vázlat)
(1) Legyen M egy k-szalagos Turing-gép. Az általánosság rovása nélkül feltehetjük, hogy M szalagjelei és belső állapotai is természetes számok. Feltehetjük azt is, hogy az elfogadás/elutasítás tényét M az output szalagjának első mezején jelzi. A szimuláció megkezdése előtt M bemenete a RAM input szalagjára van írva. Egy cellában egyetlen szalagjel szerepel. A RAM belső tárának első c celláját munkaterületként használjuk. A c egy az M-től függő állandó. Ebben az első részben tároljuk az M aktuális belső állapotát, és itt kap helyet az a k cella is, amelyekben az M fejeinek helyzetét ábrázoljuk.

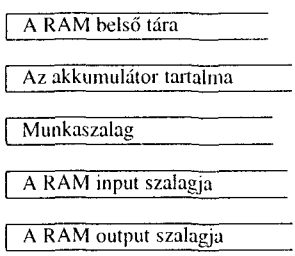
[7] A számítógépes aritmetika talán legnevezetesebb nyitott kérdése, hogy van-e ilyen algoritmus. A ma ismert leggyorsabb módszert Arnold Schönhage és Volker Strassen találta. Az algoritmus két n jegyű szám szorzásához $O(n \log n \log\log n)$ bit-műveletet használ, és nem praktikus.

A belső tár további részében *összefésülve*[8] tároljuk az M szalagjainak tartalmát: az M j-edik szalagjának i-edik celláját a belső tár $c - 1 + j + ki$ sorszámú cellája jelenti.

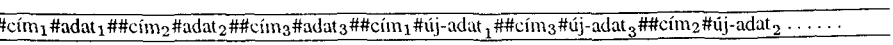
A RAM programja tartalmazza az M átmeneteinek leírását. Az M egy lépését a közvetlen elérésű gép a bemenet hosszától független konstans számú lépésben utánozza. A fejezet elején vázolt **if...then...** utasítások megvalósíthatók konstans sok RAM-alapművelettel. A RAM az M celláinak megfelelő helyeket indirekt címzéssel éri el a tár első részében levő mutatók alapján.

A programnak n jelből álló bemenet esetén szimulálnia kell az M legfeljebb $T_M(n)$ lépését. Ennek az uniform költsége $O(T_M(n))$. Ugyanezen a korláton belül maradva átmásolhatjuk az M output szalagjának (érdemi) tartalmát a RAM output szalagjára. A szimuláció uniform költsége tehát összesen is $O(T_M(n))$. A logaritmikus költség becsléséhez nézzük a szimuláció során fellépő számok méretét: az M szalagjelei és belső állapotai (M-től függő) konstans hosszúságúak, a fejek helyét leíró mutatók pedig $O(\log T_M(n))$ bittel ábrázolhatók. A szimuláció logaritmikus költsége tehát $O(T_M(n) \log T_M(n))$.

(2) A RAM-programot szimuláló N gép szalagjelei legyenek $\{0, 1, +, -, \#, \ddot{u}\}$. Az első négy jellel binárisan írt előjeles egészeket kódolunk, a # elválasztójelként fog szolgálni. N-nek öt szalagja lesz. Az elsőn a RAM belső tárát ábrázoljuk, a másodikon az akkumulátor tartalmát. A harmadik szalag munkaszalag lesz, a negyedik és az ötödik pedig a RAM input, illetve output szalagjának felel meg. Az utóbbi két szalagon a binárisan ábrázolt egészeket # jelek választják el egymástól. N szalagjai tehát így néznek ki.



Külön figyelmet érdemel az első szalag, amin a RAM belső memóriáját szimuláljuk. Ez a szalag *cím# adat* alakú feljegyzéseket tartalmaz, ## jelekkel elválasztva.



[8]Ezt az ötletet érdemes megjegyezni. Változatait gyakran használják az ún. *dinamikus allokációt* igénylő helyzetekben: amikor is egyetlen tartományban több, dinamikusan változó méretű állományt kell kezelni.

Például az $110\# - 10001$ pár azt jelenti, hogy $r[6] = -17$. Egy adott című rekesz „beolvasását" N úgy végzi, hogy végigpásztázza az első szalagot a legutolsó olyan feljegyzésig, amelynek a *cím* része egyezik az adott címmel. A megfelelő adatrészt ezután az akkumulátort ábrázoló szalag elejére írja, és a végére üresjelet tesz. A „kiírás" úgy történik, hogy a megfelelő *cím#adat* párt az első szalag végére írjuk. Azért használunk mindig új helyet, mert előfordulhat, hogy a szóban forgó cellában levő szám mérete megnőtt, és már nem férne el a korábbi helyére.

A RAM alaputasításainak az N állapotcsoportjai felelnek meg. Például van egy az összeadáshoz tartozó állapotcsoport. Amíg N az összeadást végzi, addig az aktuális belső állapota ebből a csoportból kerül ki. A következő RAM-utasításra lépést, illetve az ugrásokat N az állapotcsoportok közötti átmenettel valósítja meg.

Az alaputasítások közül a LOAD és a STORE szimulációját már vázoltuk. További példaként tekintsük az ADD $*9$ megvalósítását:
1. Először N megkeresi az első szalagon a legutolsó olyan feljegyzést, aminek a címrésze $\#\#1001\#$. Ha nincs ilyen, akkor N megáll, a szimuláció nem folytatható, hiszen az indirekt címzés értelmetlen. Ha a keresett cím szerepel a szalagon, akkor a mögötte levő a adatot N a harmadik szalag elejére másolja.
2. Megkeresi az első szalagon az utolsó feljegyzést, aminek a címrésze a harmadik szalagon levő a érték. Ha ez nem lehetséges, akkor N megáll. Ha a keresés sikeres volt, akkor az a címhez tartozó b adatrészt a harmadik szalag elejére másolja.
3. A harmadik szalagon található b számot hozzáadja az akkumulátor tartalmához, vagyis a második szalagon levő számhoz; az eredmény a második szalagon jelenik meg.

Ezeket a mintákat követve nem nehéz meggondolni, hogy az alaputasítások – a MULT és DIV kivételével – megvalósíthatók úgy, hogy N lépésszáma az utasításban szereplő adatok hosszával plusz az első szalag érdemi részének hosszával arányos legyen. Ha a RAM bemenetének mérete n volt, akkor a logaritmikus költség definíciója szerint ez az összhossz $O(t(n))$, így egy lépés szimulációjának a költsége is $O(t(n))$.

A RAM $t(n)$ lépésének szimulációjához tehát N legfeljebb $t(n)O(t(n)) = O(t^2(n))$ lépést tesz. □

Feladat: Mutassuk meg, hogy két n jegyű egész szorzása, illetve osztása Turing-géppel $O(n^2)$ lépésben elvégezhető.

Feladat: Mutassuk meg, hogy egy tetszőleges (MULT és DIV utasításokat is használó) RAM-program szimulálható egy olyan N Turing-géppel, amelyre $T_N(n) = O(t^3(n))$.

A tétel, illetve az utóbbi feladat szerint ha egy algoritmikus probléma megoldható $T(n)$ költséggel az egyik modellben, akkor megoldható legfeljebb $O(T^3(n))$

költséggel a másik modellben is. A két modell tehát *polinomiálisan összehasonlítható* abban az értelemben, hogy az egyiknél számított költség becsülhető a másikon vett költség egy polinomjával. Ha tehát pusztán az érdekel bennünket, hogy egy feladatra van-e $O(n^c)$ költségű algoritmus valamilyen $c > 0$ kitevővel[9], akkor közömbös, hogy melyik gépmodellben gondolkodunk.

[9]Hatékony algoritmusok keresésekor gyakran ez az első kérdés, amivel foglalkozunk.