

## 1. $\vec{st}$ -ELÉRHETŐSÉG $co\mathcal{NL}$ -ben, Immerman-Szelepcsényi tétel

### 1. Tétel (Immerman (1988)—Szelepcsényi (1987)).

$$\vec{st}\text{-ELÉRHETŐSÉG} \in co\mathcal{NL}.$$

**Megjegyzés.** A tétel kifejtve a következőket jelenti. Ha egy rögzített kódolás mellett az input  $\omega = \ulcorner(\vec{G}, a, z)\urcorner$ , akkor létezik olyan logaritmusos tárigenyű nem-determinisztikus  $T$  Turing-gép, amelynek

- ha nem létezik  $\vec{az}$  irányított út, akkor van elfogadó futása,
- ha létezik  $\vec{az}$  irányított út, akkor minden futása elvető.

**Észrevétel.** Neil Immerman és Róbert Szelepcsényi egymástól függetlenül igazolták a tételt 1987-ben, Szelepcsényi mindössze 20 éves volt ekkor. A közismert bonyolultsági osztályok gyorsan kialakultak az 50-es, 60-as évek környékén, így ez a tétel viszonylag későinek mondható. A tétel — a bizonyításával együtt — egyszerűsége és fontossága miatt ma már standard tananyagának számít.

**Bizonyítás.** A bizonyítás során mindig feltesszük, hogy  $\vec{G}$  gráf az  $\omega$  inputban kódolt gráf. Vezessük be a következő jelöléseket:

$$\begin{aligned} N_i &= \{v \in V(\vec{G}) : a \vec{\rightsquigarrow}_{\leq i} v\}, \\ n_i &= |N_i|. \end{aligned}$$

Tehát  $N_i$  jelenti azoknak a csúcsoknak a halmazát, melyek  $a$ -ból legfeljebb  $i$  lépésben irányítottan elérhetőek,  $n_i$  pedig az  $N_i$  számosságát jelöli. Érdeemes észrevenni, hogy  $N_0 = \{a\}$ ,  $N_1 = \{a\} \cup \{a \text{ ki-szomszédai}\}$ ,  $N_0 \subseteq N_1 \subseteq N_2 \subseteq \dots \subseteq N_i \subseteq \dots \subseteq V(\vec{G})$ .  $n_0 = 1$  illetve  $n_0 \leq n_1 \leq n_2 \leq \dots \leq n_i \leq \dots \leq |V(\vec{G})| := n$ .

A bizonyítás során az lesz a célunk, hogy az  $n_0, n_1, n_2, \dots$  értékeket kiszámoljuk. A következő észrevétel alapján ez elegendő célunkhoz.

**Észrevétel.** Akkor és csak akkor nincs  $\vec{G}$ -ben egy irányított  $az$  út, ha létezik  $i \leq |V(\vec{G})|$ , hogy  $n_i = n_{i+1}$  és  $z \notin N_i$ .

Ez azzal indokolható, hogy ha valamely  $i$ -re  $n_i = n_{i+1}$ , akkor  $N_i = N_{i+1}$  és az egyenlőség az  $(i + 1)$ -nél nagyobb indexekre is fennáll, így az  $N_i$  az összes  $a$ -ból

elérhető csúcsot felsorolja. Ilyen  $i \leq |V(G)|$  mindig létezik. Az irányított út létezése ekvivalens azzal, hogy  $z$  eleme-e  $N_i$ -nek.

A bizonyítás során egy nem-determinisztikus rekurzív algoritmust fogunk felírni. A rekurzió feltétele, hogy ismerjük  $n_i$ -t. Az algoritmus segítségével  $n_i$ -ből meghatározzuk  $n_{i+1}$ -et és képesek leszünk  $N_i$  elemeinek felsorolására. Ez elegendő lesz számunkra: az ELFOGAD állapot elérése ekvivalens lesz azzal, hogy valamely  $i \leq |V(G)|$  esetén  $n_i$  megegyezik  $n_{i+1}$ -gyel és  $z \notin N_i$ .

A bizonyítás során szükségünk lesz négy darab munkaszalagtöredékre. Gondolhatunk rájuk úgy is, mint ugyanannak a munkaszalagnak a különböző, de jól meghatározott része, vagy úgy is, mint külön munkaszalagok (ekkor többszalagos Turing-gép modellel dolgozunk).

$$(1) \quad \boxed{\triangleright \mid i \mid : \mid n_i \mid \triangleleft}$$

Az (1) munkaszalagtöredéken tároljuk a megfelelő indexekhez tartozó  $n_i$ -ket, és biztosnak kell lennünk abban, hogy  $n_i = |N_i|$ .  $i$  és  $n_i$  területe is  $\mathcal{O}(\log n)$  mezőnyi.

A következő (2)-es munkaszalagok az  $N_i$ -beli elemeket sorolják föl.

$$(2a) \quad \boxed{\triangleright \mid N_i\text{-beli elemek helye} \mid : \mid \text{számláló} \mid \triangleleft \mid \dots \mid}$$

$\overleftarrow{\hspace{1.5cm}}$   
 $\lceil \log V \rceil$ , azaz egy  
csúcsnyi hely

$$(2b) \quad \boxed{\triangleright \mid \text{bizonyító séta csúcsa} \mid \mid \text{következő csúcs} \mid \mid \text{megtett lépések száma} \mid \dots \mid}$$

$\overleftarrow{\hspace{1.5cm}}$        $\overleftarrow{\hspace{1.5cm}}$   
 $\lceil \log V \rceil$ , azaz egy       $\lceil \log V \rceil$ , azaz egy  
csúcsnyi hely                      csúcsnyi hely

A (3) munkaszalagok  $V$  tesztelésére szolgálnak, egy  $V$ -beli csúcs  $N_{i+1}$ -hez tartozását vizsgálja.

$$(3a) \quad \boxed{\triangleright \mid \text{tesztelt csúcs} \mid \triangleleft}$$

$\overleftarrow{\hspace{1.5cm}}$   
csúcsnyi hely

$$(3b) \quad \boxed{\triangleright \mid \text{tesztelő séta csúcsa} \mid \mid \text{következő csúcs} \mid \mid \text{megtett lépések száma} \mid \dots \mid}$$

$\overleftarrow{\hspace{1.5cm}}$        $\overleftarrow{\hspace{1.5cm}}$   
 $\lceil \log V \rceil$ , azaz egy       $\lceil \log V \rceil$ , azaz egy  
csúcsnyi hely                      csúcsnyi hely

A (4) pedig a megtalált  $N_{i+1}$ -beli elemeket számolja.

$$(4) \quad \boxed{\triangleright \mid \hspace{1.5cm} \mid \triangleleft}$$

$\overleftarrow{\hspace{1.5cm}}$   
 $\log |V|$

A (1)-es szalagon kezdetben  $0 : 1$  van, a többi szalag pedig üres.  $|N_0| = |\{a\}| = 1$  alapján kiinduló hipotézisünk helyes. A szalagtartalom megbízható.

A rekurzív lépés ( $n_i$ -re alapulva, felsoroljuk  $N_i$  elemeit, kiszámoljuk  $n_{i+1}$ -et) a következő.

1. Felsoroljuk az  $N_i$  elemeit, azaz a (2a) szalagon a  $v_1, v_2, \dots, v_{n_i}$  csúcsok sorolódnak. Természetesen oly módon, hogy  $v_1$  leíródik (számláló értéke 1), azt  $v_2$  felülírja (számláló eggyel növeledik), és ez így megy tovább, míg végül  $v_{n_i}$  lesz ráírva (számláló eléri  $n_i$ -t).

Az algoritmusnak ez a része egy nem-determinisztikus fázis. Az összes csúcst „tippelt”, nem-determinisztikus lépés eredménye. Erre a felsorolásra úgy is lehet majd gondolni, mint egy szubrutin. Ismételten alkalmazzuk (természetesen ugyanazon tárterület felhasználásával).

- Azért, hogy elkerüljük, hogy egy csúcsot többször is felsoroljunk, felteesszük, hogy a csúcsok kódjai az indexekkel növekednek, azaz  $\lceil v_1 \rceil < \lceil v_2 \rceil < \dots$
- Mielőtt felsorolnánk a következő csúcsot, a (2b) részben minden  $v_j$ -re legyártunk egy bizonyítást arra, hogy  $v_j \in N_i$ . Ez is alapvetően nem-determinisztikus része az algoritmusnak. Ezért ha  $v_j \in N_i$ , akkor van olyan futás/tanúsorozat-tartalom, ami ezt bizonyítja, és ezt logtárral le tudjuk ellenőrizni a (2)-es szalag bizonyító részében, mivel a probléma  $\mathcal{NL}$ -be tartozását tudjuk. A bizonyító rész tartalma a következőképpen alakul: Kezdetben  $a$ -ból átlépünk egy szomszédos  $a^+$  csúcsba, majd növeljük a számlálót. Ezután az  $a$  felülíródik a szomszédjával ( $a^+$ -szal) és most annak egy szomszédja lesz a második csúcs ( $a^{++}$ ), ami pedig az  $a^+$  csúcsot írja felül. Ez a művelet addig ismétlődik, míg a második csúcs  $v_j$  nem lesz vagy a számláló túlszordul (nagyobb lesz, mint  $i$ ). Két állapot léphet fel: STIMMEL, illetve NEM-STIMMEL. A STIMMEL állapotban elérjük  $v_j$ -t a számláló túlszordulása nélkül. A komplementer NEM-STIMMEL állapot ekvivalens azzal, hogy elvetjük az egész futást. Ha NEM-STIMMEL állapot nélkül a felsorolás számlálóját  $n_i$  értéket vesz fel, akkor SIKERES-FELSOROLÁS állapotba jutunk.

Ha feltevésünk helyes (ha (1) tartalma valóban  $n_i$ ), továbbá egy „jogkövető” futásról van szó akkor az  $N_i$  elemeit tényleg fel tudjuk sorolni.

**2. Lemma.** *Ha végigfutunk a SIKERES-FELSOROLÁS állapotba kerülünk, akkor biztosak lehetünk abban, hogy  $N_i$  elemeit soroltuk fel.*

**Bizonyítás.** Mivel elkerültük a NEM-STIMMEL állapotot, ezért bizonyítottuk, hogy  $n_i$  darab különböző elemet soroltunk fel  $N_i$ -ből. Mivel  $n_i$ -ben megbízunk, így  $N_i$  összes elemét felsoroltuk. ■

2. Az algoritmus következő fázisa a  $V$  tesztelése, azaz a (3)-as szalagon felsoroljuk  $V$  összes elemét, és ellenőrizzük, hogy benne vannak-e  $N_{i+1}$ -ben. Legyenek az  $u_1, \dots, u_n$  csúcsok a  $V$  elemei. Kezdetben a (4) szalag tartalma 0.

- Minden  $u_j$  csúcs esetében meghívjuk a fenti szubrutint, azaz (nem-determinisztikusan) felsoroljuk az  $N_i$  elemeit.
- Azt teszteljük, hogy az aktuális  $u_j$  az fel van-e sorolva  $N_i$ -ben, vagy ki-szomszédja egy olyan csúcsnak, ami fel van sorolva. Ha ez megtörténik, akkor a (4)-es szalag(töredéken) növeljük eggyel a számlálót és a következő csúcsra térünk át. Ha az  $N_i$  felsorolása SIKERES-FELSOROLÁS állapotba ér, az aktuális  $V$ -beli elem nem lett felismerve mint  $N_{i+1}$  egy eleme, akkor (a számláló növelése nélkül) a következő csúcsra térünk át. A következő lemma eddigi észrevételeinket foglalja össze.

- 3. Lemma.** (i)  $x \in N_{i+1}$  akkor és csak akkor, ha  $x \in N_i$  vagy alkalmas  $y \in N_i$  esetén:  $\vec{y}\vec{x} \in E$ .
- (ii) Ha  $V$  összes elemének tesztelése megtörténik a NEM-STIMMEL állapot elkerülésével, akkor a (4) szalag tartalma biztos  $n_{i+1}$

Az előző lemma tulajdonképpen egy trivialisítás, a bizonyításához elég egyszerűen meggondolni azt, hogy mit jelent. Ez a lemma biztosítja a rekurziós lépést, és így a rekurzív lépés leírásának vége van.

3. Már csak annak az ellenőrzése van hátra, hogy  $n_i$  megegyezik-e  $n_{i+1}$ -gyel.

- Ha nem, akkor az (1)-es szalagon  $i$ -t kicseréljük  $(i+1)$ -re és  $n_i$ -t kicseréljük a (4)-es szalagon lévő számra, ami az előző lemma szerint pontosan az  $n_{i+1}$ , és visszatérünk az algoritmus 1. részéhez.
- Ha igen, akkor újból felsoroljuk  $N_i$  elemeit, és csak azt vizsgáljuk, hogy  $z$  előjön-e közöttük. Ha nincs NEM-STIMMEL állapot a felsorolás során és a  $z$  sem jön elő, akkor tudjuk, hogy nem létezik  $\vec{a}\vec{z}$  út, így ELFOGAD állapottal leállunk.

Az algoritmus logaritmikustárat használ fel, a helyes működése a fentiekben bizonyítva lett, ezért az Immerman-Szelepcsényi tétel bizonyítása kész. ■

#### 4. Következmény. $\mathcal{NL} = co \mathcal{NL}$

**Bizonyítás.** Ha  $L$  egy  $\mathcal{NL}$ -beli nyelv, akkor létezik egy  $L \prec_{\mathcal{L}} \vec{st}$ -ELÉRHETŐSÉG visszavezetés (mivel az  $\vec{st}$ -ELÉRHETŐSÉG  $\mathcal{NL}$ -teljes), így tudjuk, hogy az  $L$  nyelv  $co \mathcal{NL}$ -ben is benne van. Az  $L \in co \mathcal{NL}$  pontosan azt jelenti, hogy  $L$  komplementere benne van  $\mathcal{NL}$ -ben, azaz  $\bar{L} \in \mathcal{NL}$ . Ezt a gondolatmenetet megismételve  $\bar{L}$ -re, azt kapjuk, hogy az  $\bar{L} \in \mathcal{NL}$ -ből következik, hogy  $L \in \mathcal{NL}$ . Ebből a két megállapításból pedig azt kapjuk, hogy  $L$  akkor és csak akkor van  $\mathcal{NL}$ -ben, ha a komplementere is ott van, és ez azt jelenti, hogy  $\mathcal{NL} = co \mathcal{NL}$ . ■

Ahogy a bizonyítás is mutatta a tétel lényege, hogy  $\mathcal{NL}$  zárt a komplementálásra nézve.

#### 5. Következmény. Ha $s(n)$ szép tárfüggvény, akkor

$$\mathcal{NSPACE}(\mathcal{O}(s(n))) = co \mathcal{NSPACE}(\mathcal{O}(s(n))).$$

**Bizonyítás.** A bizonyítás hasonlóan történhet, mint az Immerman-Szelepcsényi tétel bizonyítása, csak itt a blokkok nagyobbak. ■

**Megjegyzés.** Az eredeti tétel, illetve a két következmény közül mindegyiket szokták Immerman-Szelepcsényi-tételnek nevezni.

ELÉRHETŐSÉG az eddig vizsgált probléma változata irányítatlan gráfokra. A következő tétel azt mutatja, hogy az irányítatlan eset valószínűleg könnyebb az irányítottnál.

#### 6. Tétel (Omer Reingold, 2004). $ELÉRHETŐSÉG \in \mathcal{L}$ .

Attól függetlenül, hogy klasszikus szélességi és mélységi keresések a problémát  $\mathcal{P}$ -be rakják, ez egy nagyon nehéz tétel. Omer Reingold bizonyította 2004-ben. Maga az algoritmus nem túl használható, de bonyolultságelméleti szempontból nagy jelentőséggel bír. Viszont az  $\vec{st}$ -ELÉRHETŐSÉG és az  $\mathcal{L}$  osztály viszonyáról még semmit sem tudunk. Az  $\vec{st}$ -ELÉRHETŐSÉG  $\in \mathcal{L}$  bizonyítása azt jelentené, hogy  $\mathcal{L} = \mathcal{NL}$ .

## 2. Hálózatok és egy $\mathcal{P}$ -teljes probléma

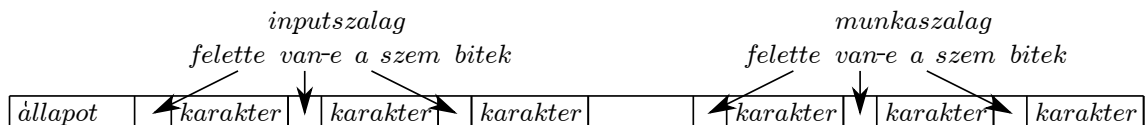
**Emlékeztető.**  $L \in_T \text{TIME}(t(n))/\text{NTIME}(t(n))$  esetén mindig feltesszük, hogy  $t(n)$  szép idő függvény, azaz Turing-géppel megvalósítható egy óra, ami  $n$  hosszú inputra „ $t(n)$  idő után üt”.

$\omega \in \Sigma^n$  inputon  $T$  futása a

$$\kappa_0(\omega) \rightarrow \kappa_1 \rightarrow \kappa_2 \rightarrow \dots \rightarrow \kappa_\ell,$$

konfigurációsorozat, ahol  $\kappa_0(\omega)$  az  $\omega$ -hoz tartozó kiinduló konfiguráció,  $\kappa_{i+1}$  a  $\kappa_i$  konfiguráció rákövetkezője, és  $\kappa_\ell$  az első olyan konfiguráció, ahol az állapot ELFOGAD vagy ELVET. Feltehető, hogy  $\ell = t(n)$ .

**Észrevétel 1.**  $\kappa$  konfigurációk kódolhatók bitsorozatokkal.



1. ábra.

A fenti ábra (megjegyzéseivel) magáért beszél. A fej pozícióját is kell kódolnunk. Ezt „szétszórtuk”, minden mező raktunk egy bitet. Így nem minden adott hosszúságú sorozat kódol konfigurációt. Az állapotot és karaktereket kódoló blokkok hossza függ  $|S|$ ,  $|\Sigma|$  és  $|\Gamma|$  értékétől. Minden esetre konstans sok bit szükséges (ami a Turing-géptől függ).  $S$  elemeinek kódolására  $\lceil \log_2 |S| \rceil$  bit legendő. Ha  $|S|$  nem kettőhatvány, akkor lesznek olyan 0-1 bitsorozatok, amik a megfelelő pozíciókban állnak, de nem kódolnak állapotot. Ennek ellenére könnyű tervezni egy olyan tesztelő hálózatot, ami egy adott (megfelelő hosszú) kódról eldönti, hogy konfigurációt kódol-e.

**Megjegyzés (FONTOS).** A megállapodást úgy választhatjuk adott  $n$  hosszú  $\omega$  input esetén  $\lceil \omega \rceil$  hossza  $\alpha_T \cdot n$  legyen. Ha  $T$  időigénye legfeljebb  $t(n)$ , akkor konfigurációk kódjának hossza  $\beta_T \cdot t(n)$  legyen. (A konfigurációban a munkaszalag elvágható az első  $t(n)$  mező után. Az elhagyott/levágott rész csak érintetlen mezőket tartalmaz.)

A továbbiakban  $n$  és a kódolási megállapodás mindig rögzített (ennek megfelelően a megfelelő kódok hossza mindig ismert).

**Észrevétel 2.**  $\lceil \kappa_i \rceil \rightarrow \lceil \kappa_i^+ \rceil = \lceil \kappa_{i+1} \rceil$  egyszerű hozzárendelés/függvény.

Az észrevétel állítása matematikailag nem pontos, az „egyszerű” jelző értelmezése nem jól definiált.

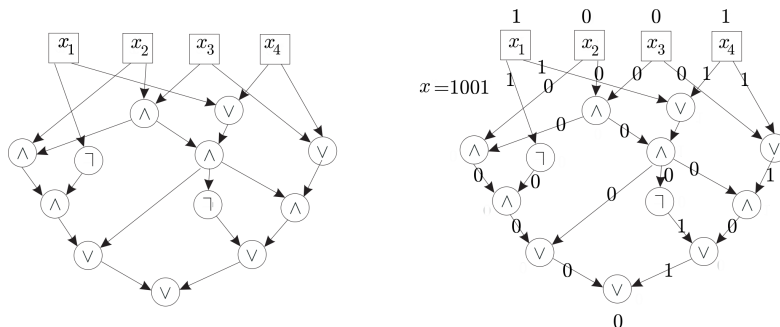
Azt mutatjuk, hogy egyszerűen meghatározható/kiszámolható egy kicsi (polinomiális méretű) hálózat, amely egy konfiguráció kódjából kiszámolja a rákövetkező konfiguráció kódját. Először azonban lássuk a szükséges fogalmakat.

**Definíció.** Egy hálózat egy  $\vec{G}$  irányított gráf, amely nem tartalmaz irányított kört (azaz lerajzolható úgy, hogy minden él „lefelé haladjon”). Minden csúcs befoka 0, 1 vagy 2. A 0 befokú csúcsok neve inputcsúcs. Legye  $I$  az input csúcsok halmaza. A nem inputcsúcsokra mint kapuk hívkozunk. A kapuk halmaza  $K = V(\vec{G}) - I$ . Speciális csúcsot, esetleg csúcsokat nevezünk ki, amelyekre mint output csúcsok hívkozunk.

Legyen  $\ell_I : I \rightarrow \{x_1, x_2, \dots, x_n\} \cup \{0, 1\}$  egy címkézése az output csúcsoknak. Legyen  $\ell_K : K \rightarrow \{\neg, \vee, \wedge\}$  egy címkézése a kapuknak, amelyre teljesül, hogy egy kapunknak akkor és csak akkor  $\neg$  a címkéje, ha befoka 1. Legyen  $\ell = (\ell_I, \ell_K)$  az összes csúcsot címkéző függvény.

$(\vec{G}, \ell)$  címkézett irányított gráfot hálózatnak nevezünk.

A fenti egy statikus számítási modell. Lássuk a fenti modell dinamikus változatát. Egy hálózat kiszámol egy Boole-függvényt a következő módon. Az  $\{x_1, x_2, \dots, x_n\}$  változók egy kiértékelését kiterjesztjük a kapuk egy kiértékelésévé: Feltesszük, hogy hálózatunkat úgy rajzoljuk le, hogy minden él lefelé haladjon. A kapuk kiértékelése fentről lefelé halad.



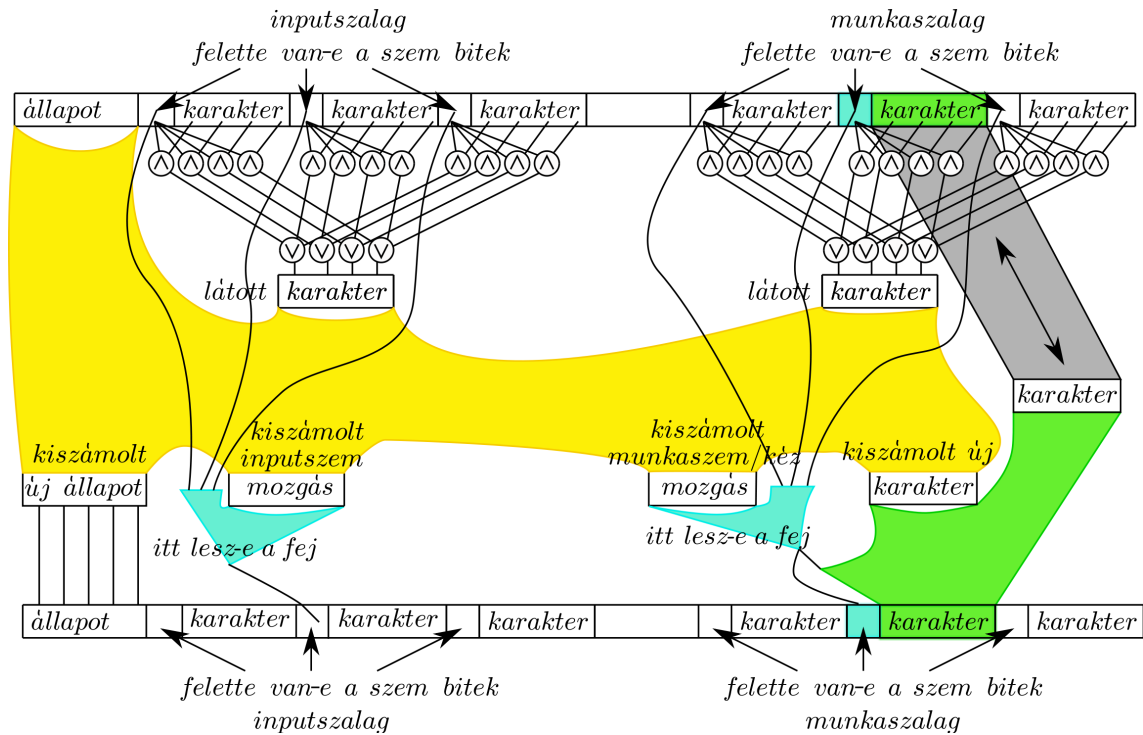
2. ábra.

Ha egy kapuhoz jutunk, akkor azok a kapuk, ahonnan él vezet hozzá már kiértékelték, azaz egy kiszámolt bitet rendeltünk hozzá. Az aktuális kapu által kiszámolt bit az hozzá vezető éleken áramló bit és a kapu címkéje alapján természetesen értelmezhető. A hálózat által kiszámolt bitsorozat az output csúcs(ok) által kiszámolt bit(ek).

Azaz a  $\mathcal{C}$  hálózat egy  $f_{\mathcal{C}}$  Boole-függvényt számol ki/valósít meg.

A fentiek „matematizálása”: Egy konfigurációt kódoló bitsorozatból egy egyszerűen leírható, kicsi hálózat kiszámolja a rákövetkező konfiguráció kódját. A konstrukciónk egyszerű, de sok esetlegességet, megállapodást követel. Egy formális leírás helyett egy példán szemléltetjük milyen ötletek vezethetnek el egy megoldáshoz.

Egy mezőnél a szem/kéz-ott-van-e bitet és a mező tartalmát kódoló bitek mindegyikét össze-és-eljük. A kapott bitsorozat vagy a csupa 0 (ha nincs ott a szem/kéz, mindent 0-val és-eltünk) vagy a látott karakter kódja (ha ott a szem/kéz). Az input-szalag mezőinél így kapott bitsorozatok mindegyikének első, majd második és így



3. ábra.

tovább karakterét össze-vagy-olva kapjuk az inputszalagon látott karakter kódját. Hasonlóan számolhatunk a munkaszalagnál.

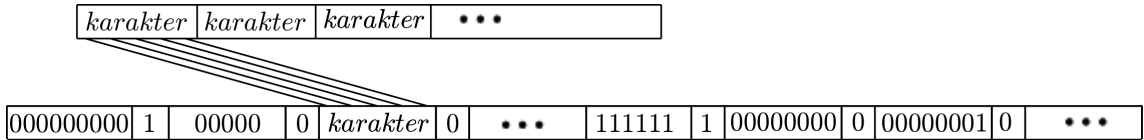
A sárga területben egy összetettebb számolás történik: konstans sok bitből számolunk ki konstans sok bitet (a konstanok függenek a Turing-géptől). Konkrét kidolgozása az átmenetifüggvénytől függ. Nem okozhat problémát akkor sem, ha az egyes bitek függését a nyilvánvaló DNF formula alapján írjuk fel. Ekkor is konstans sok kapuval dolgozva megoldjuk a feladatunkat.

A világos kék területen számolódik kis a fej pozícióját leíró egyik bit. Ez attól függ, hogy a fej ott volt-e vagy valamelyik szomszéd felett állt, illetve merre írja elő mozgását az átmeneti függvény. A hálózatunk ezen részét konkrétan felírhatnánk, de ez a munka felesleges az előző megjegyzésünk alapján. Ez a kék rész minden fej-pozíció-bithez ott van. Az átláthatóság kedvéért rajzoltuk fel csak egyszer az input- és egyszer a munkaszalaghoz. A zöld területen számoljuk ki a munkaszalag új tartalmát. Minden munkaszalag mezőhöz tartozik egy ilyen zöld blokk (egyszerűség kedvéért csak egyet tüntettünk fel). Az új karaktertől, a régitől és attól az információtól függ, hogy a fej ott áll-e. Ez a rész is könnyen megvalósítható lenne, ha tudnánk  $\Gamma$  elemeinek kódolásához használt bitek  $\ell$  számát ismernénk. A zöld területen az  $f(\epsilon, k_0, k_1) = k_\epsilon$  függvényt számoljuk ki, ami  $1 + 2\ell$  bitből számol ki  $\ell$ -et.

**Észrevétel 3.**  $[\omega] \rightarrow [\kappa_0(\omega)]$  egy egyszerű hozzárendelés/függvény.

Ez a korábbiaknál egyszerűbb észrevétel. Ismét egy ábrára utalunk.

Feltettük, hogy a START állapot kódja  $00 \dots 0$  (hossza  $\lceil \log_2 |S| \rceil$ , példánkban 9). Az inputszalagon  $\triangleright$  kódja  $00 \dots 0$ , a  $\triangleleft$  kódja  $11 \dots 1$  (hosszaik  $\lceil \log_2 |\Sigma| \rceil$ , példánkban 5). Az munkaszalagon  $\triangleright$  kódja  $0 \dots 00$ , a szűzkarakter kódja  $0 \dots 01$ , (hossza  $\lceil \log_2 |\Gamma| \rceil$ , példánkban 8).



4. ábra.

A fentiek alapján egyszerűen adódik a következő:

**Definíció.** Legyen

$$\text{HÁLÓZAT-KIÉRTÉLÉS} = \{[\mathcal{C}, \omega] : \mathcal{C}(\omega) = 1\},$$

azaz az a döntési probléma, amely egy  $\mathcal{C}$  hálózat és  $\omega$  bitsorozat esetén eldönti, hogy az  $\omega$  bitsorozatot adva az  $x_1, x_2, \dots$  inputkapuk értékeinek a hálózat az 1 bitet számolja-e ki (azaz kiértékeli  $\mathcal{C}_n(\omega)$ -t).

**7. Tétel.** *HÁLÓZAT-KIÉRTÉLÉS  $\mathcal{P}$ -teljes (az  $\mathcal{L}$  redukcióra nézve).*

**Bizonyítás.** Legyen  $L \in_T \mathcal{P}$ . Legyen  $t(n)$  egy polinom,  $T$  időkorlátja. Feltehető, hogy  $T$  olyan, hogy ELFOGAD/ELVET állapot elérése után „tartja” állapotát. Így  $\omega \in L?$  kérdés megválaszolása ( $\omega \in \Sigma^n$ ) ekvivalens annak megállapításával, hogy az  $\omega$ - történet futás során a  $\kappa_{t(n)}$  konfigurációban az állapot ELFOGAD-e. Megállapodhatunk abban, hogy a konfigurációkat 0-1 sorozatokkal kódoljuk, hogy az állapotot kódoló blokkban az ELFOGAD állapot kódja a csupa 1 sorozat legyen.

A fentiek alapján bármi volt is  $L \in_T \mathcal{P}$   $\omega \in \Sigma^n$  tetszőleges eleméhez felépíthető egy  $\mathcal{T}_{T,\omega}$  hálózat, amely inputkapui  $\omega$ -t kódolják (3. észrevétel) és bizonyos szintje a Turing-számítás konfiguráció-sorozatának elemeit kódolja (2. észrevétel).  $t(n)$  ilyen szint felépítése után az érdekel minket, hogy egy bizonyos blokkban csupa 1-es bit szerepel-e. Ezt ÉS kapuk segítségével könnyen kifejezhetjük.

Ezzel leírtuk a redukáló algoritmust. A redukció elméleti része a korábbiakból adódik. A konstrukció/redukció logtár-beli. ■

### 3. Egy $\mathcal{NP}$ -teljes nyelv

**Definíció.** Legyen

$$\text{HÁLÓZAT-SAT} = \{[\mathcal{C}] : \text{van olyan } \omega \text{ bitsorozat, amelyre } \mathcal{C}(\omega) = 1\},$$

azaz az a döntési probléma, amely egy  $\mathcal{C}$  hálózat esetén el kell döntenünk, hogy kielégíthető-e.

**8. Következmény.** • (i) *Tetszőleges  $L \in \mathcal{NP}$  nyelvre  $L \prec_{\mathcal{L}} \text{HÁLÓZAT-SAT}$*

• (ii) *HÁLÓZAT-SAT  $\mathcal{NP}$ -teljes*

• (iii)  *$\mathcal{P} = \mathcal{NP} \Leftrightarrow \text{HÁLÓZAT-SAT} \in \mathcal{P}$*

**Bizonyítás.** (i)  $L \in_T \mathcal{NP}$ , így egy tetszőleges  $\omega \in L$  esetén létezik hozzá egy tanú:  $\tau = (t_1, t_2, \dots, t_{p(n)})$ , amelyre  $T(\omega, \tau)$  ELFOGAD állapotba jut. Azaz az előbb megkonstruált  $\mathcal{C}$  hálózatra,  $\mathcal{C}([\omega], y_1, y_2, \dots, y_{q(n)})$  az 1 értéket számolja ki,



ha az  $y$  változók helyére a  $\tau$  kódjának bitjeit írjuk. Megfordítva is igaz. Ha  $C([\omega], y_1, y_2, \dots, y_{q(n)})$ -nek találunk egy kielégítését, akkor egy tanú kódját találjuk. Azaz  $C([\omega], y_1, y_2, \dots, y_{q(n)})$  kódjának legyártása (ami az emlékeztető alapján  $\mathcal{L}$ -ben megoldható) egy jó redukció.

(ii) Az (i) részből és abból, hogy HÁLÓZAT-SAT  $\in \mathcal{NP}$  (tanú egy kielégítő bemenet), következik, hogy HÁLÓZAT-SAT  $\mathcal{NP}$ -teljes.

(iii) Mivel HÁLÓZAT-SAT  $\in \mathcal{NP}$ , így ha  $\mathcal{P} = \mathcal{NP}$ , akkor  $\mathcal{P}$ -beli is. Viszafelé, ha HÁLÓZAT-SAT  $\in \mathcal{P}$ , akkor tetszőleges  $L \in \mathcal{NP}$  nyelvet redukáljunk HÁLÓZAT-SAT-ra, a redukált problémát  $\mathcal{P}$ -ben el tudjuk dönteni. A két lépés együtt is polinomiális és az  $L$  nyelv eldöntési problémáját oldja meg. Ebből  $L \in \mathcal{P}$  következik, így  $\mathcal{NP} \subseteq \mathcal{P}$ , tehát  $\mathcal{P} = \mathcal{NP}$  adódik. ■