Algoritmusok és bonyolultságelmélet
Matematikus MSc hallgatók számára
7. előadás (2011. március 21.)
Előadó: Hajnal Péter
Jegyzetelő: Vizi Zsolt

Egy $\mathcal{P}$-teljes probléma

■ A hálózat fogalma

Definíció: $H = \left(\vec{G}, c, O\right)$ rendezett hármast *hálózat*nak nevezzük, ha

i) $\vec{G}$ irányított gráfra teljesülnek:

a) nem tartalmaz irányított kört ≡ lerajzolható úgy, hogy minden éle szigorúan lefelé mutat

≡ $\exists \pi$ sorrendje a csúcsoknak, hogy $\pi : v_1, v_2, \ldots v_n$ és $\forall \overrightarrow{v_i v_j} \in E(\vec{G})$ ha $i < j$;

b) minden pont befoka legfeljebb 2:

- ha egy pont befoka 0, akkor a csúcsot *inputcsúcs*nak nevezzük,
- ha a befoka 1 vagy 2, akkor a csúcsot *kapu*nak nevezzük;

ii) $c : V(\vec{G}) \to \{\bigwedge, \bigvee, \neg\} \dot{\cup} V \dot{\cup} \{0, 1\}$, ahol V a változók halmaza, {0, 1} a logikai konstansok halmaza és

- v inputcsúcs $\Leftrightarrow c(v) \in V \dot{\cup} \{0, 1\}$,
- $d_{be}(v) = 1 \Leftrightarrow c(v) = \neg$,
- $d_{be}(v) = 2 \Leftrightarrow c(v) \in \{\bigwedge, \bigvee\}$;

iii) $O \subseteq V(\vec{G})$ az *outputcsúcs*ok halmaza, $O = \{o_1, o_2, \ldots, o_k\}$.

Megjegyzés:

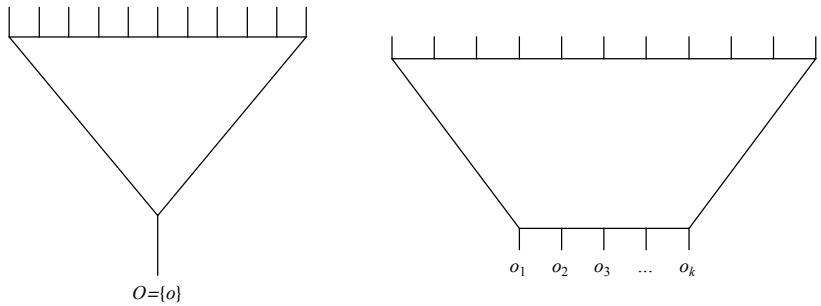
• A definíció i) pontjának a) részében az első alternatív megfogalmazás azért fontos, mivel ezt a geometriai szemléletet fogjuk használni a hálózat szemléltetésekor.

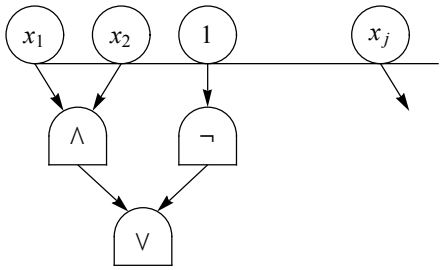
• A szakirodalomban előfordul, hogy a felfelé folyik a hálózat; a folyás iránya a számolás irányát határozza meg.

• Természetesen az irányított gráf irányítatlan értelemben tartalmazhat kört.

• A definíció ii) pontjában a visszairányok nem lennének szükségesek, de a fogalom leírásából közvetlenül kiolvashatók (például, ha egy csúcs befoka nem nulla, akkor 1 vagy 2, de a fogalmat úgy alkottuk meg, hogy a nemnulla befokú csúcs címkéje nem lehet változó vagy konstans.)

• A hálózat fogalmát a következő ábrákkal lehet szemléltetni:





Ábra: Egy és több outputcsúccsal rendelkező hálózat sematikus rajza (felül) és részlet egy hálózatból (alul)

• Az outputcsúcsok alatt elhelyezkedő csúcsok számunkra érdektelenek, mivel a vizsgálódásaink szempontjából azok a csúcsok jelentősek, amelyekből irányított út vezet egy outputcsúcsba.

Az eddigiek a hálózat fogalmának *statikus* képét adták meg - ahhoz hasonló, mikor egy algoritmust valamilyen rögzített programozási nyelv megállapodásai alapján leírunk. A következőkben a hálózat *dinamikus* képét dolgozzuk ki - mint amikor egy algoritmus esetén előírjuk, hogy mit csináljon, hogyan kell a futásának zajlani.

Definíció: Legyen $\kappa : V \to \{0, 1\}$ a változók egy kiértékelését végző függvény. Ekkor a κ alapján definiáljuk a következő $\hat{\kappa} : V(\vec{G}) \to \{0, 1\}$ függvényt:

i) konstanssal címkézett v csúcsnál $\hat{\kappa}(v) := c(v)$;

ii) változóval címkézett v csúcsnál $\hat{\kappa}(v) := \kappa(c(v))$;

iii) egyébként legyen v_i az első csúcs, amelyre még nem értelmeztük $\hat{\kappa}$-ot

- $c(v_i) = \neg$ esetén $\hat{\kappa}(v_i) := \neg\, \hat{\kappa}(v_l)$, ahol $\overrightarrow{v_l v_i} \in E(\vec{G})$;
- $c(v_i) = \wedge$ esetén $\hat{\kappa}(v_i) := \hat{\kappa}(v_l) \bigwedge \hat{\kappa}(v_m)$, ahol $\overrightarrow{v_l v_i}$, $\overrightarrow{v_m v_i} \in E(\vec{G})$;
- $c(v_i) = \vee$ esetén $\hat{\kappa}(v_i) := \hat{\kappa}(v_l) \bigvee \hat{\kappa}(v_m)$, ahol $\overrightarrow{v_l v_i}$, $\overrightarrow{v_m v_i} \in E(\vec{G})$.

Legyen $f_\kappa(\mathrm{H}) : \{0, 1\}^n \to \{0, 1\}^k$, $(x_1, x_2, \ldots, x_n) \mapsto (y_1, y_2, \ldots, y_k)$, ahol $y_i = \hat{\kappa}(o_i)$, vagyis y_i az i-edik output értéke.

Megjegyzés:
• Az eddigiek alapján a hálózatra gondolhatunk úgy, mint egy algoritmusra, amely kiszámol valamit.
• Szemléletesen úgy képzelhetjük el a hálózat működését, mintha a felső sorban lévő változók üveggolyók lennének és miután értéket kaptak, elkezdenek lefelé esni; mikor két golyó befut egy kapuba, akkor a kapu az értékeknek és saját címkéjének (vagyis logikai feladatának) megfelelően továbbít egy golyót. Minket a legalján kipottyanó golyók érdekelnek. Tehát úgymond az információ "leesik", a kapuk pedig összekombinálják a biteket.
• Láthatjuk, hogy az f_κ nagyban függ a hálózattól.

1. Emlékeztető: Legyen T Turing-gép, $\omega \in \Sigma^n$, T időigénye $t(n)$, κ olyan konfiguráció, melynél a munkaszalagot $t(n)$ mező után levágjuk. Ekkor $\lceil\kappa\rceil \in \{0, 1\}^{\alpha_T \cdot t(n)}$ (ahol $\lceil\kappa\rceil$ κ kódját jelöli) és $\lceil\kappa^+\rceil$ (κ rákövetkezője) könnyen kiszámolható $\lceil\kappa\rceil$-ból.

Lemma: $\exists\, C_{T,n}$ hálózat $\alpha_T \cdot t(n)$ darab input- és outputcsúccsal, amely $\lceil\kappa\rceil$-ból $\lceil\kappa^+\rceil$-t számolja ki.

Megjegyzés:
• Ez a lemma a korábbi előadás eredményének megfogalmazása a hálózat fogalmának segítségével.
• A kódolás olyan volt, hogy maradtak 0-1 sorozatok, amelyek nem kódoltak konfigurációt, mivel az állapotok halmazát bitsorozatokkal kódoljuk, de az állapotok halmazának számossága nem szükségszerűen kettő hatvány. Például, ha van 7 állapotunk, akkor 3 hosszú bitsorozatokkal kódoljuk őket, de lesz egy darab olyan három hosszú bitsorozat, ami nem kódol állapotot. Ha a kódban legelőre pont ez a hármas kerül, akkor az a kód nem jó konfigurációnak. Az olyan 0-1 inputokkal, amelyek nem kódolnak konfigurációt, azokkal a hálózat azt csinál, amit akar, de ha jó (konfigurációt leíró) 0-1 sorozatot adunk a hálózatnak, akkor a kiszámolt sorozat is konfiguráció lesz - méghozzá a következő konfiguráció.

2. Emlékeztető: Ha $L \in_T \mathcal{P}$, akkor $\forall\, \omega \in \Sigma^n$ esetén $\exists\, \kappa_0(\omega) \to \kappa_1(\omega) \to \ldots \to \kappa_l(\omega)$ konfiguráció sorozat, ahol $\kappa_l(\omega)$ a leálló konfiguráció, $l \le t(n)$ és κ_l ELFOGADó $\Leftrightarrow \omega \in L$.

Következmény: $\exists\, \mathcal{D}_n(\mathrm{T})$ hálózat úgy, hogy $\omega \in \Sigma^n$ esetén

$$f_{\lceil\omega\rceil}(\mathcal{D}_n(\mathrm{T})) = \begin{cases} 1, & \omega \in L \\ 0, & \omega \notin L \end{cases}$$

Bizonyítás: A Lemma biztosít nekünk olyan hálózatot, amely kiszámítja a rákövetkező konfigurációt. Ilyen hálózattal a $\kappa_0(\omega) \to \kappa_1(\omega) \to \ldots \to \kappa_l(\omega)$ konfigurációsorozat rekurzívan (mindegyiket az előző konfigurációból megkapva) kiszámolható. Feltehető, hogy $l = t(n)$, mivel $t(n)$ szép időfüggvény (polinom). Ez a feltétel nem jelent megszorítást, mivel $l < t(n)$ esetén a Turing-gépbe építünk egy "órát", ami $t(n)$ lépés után "csörög": ha az óra csörgése előtt kiszámítottuk az utolsó konfigurációt, akkor az órát futtatjuk, megvárjuk a csörgést és csak akkor állunk le. Mivel a kódolást úgy oldottuk meg, hogy a konfigurációk kódjának eleje az állapotot tartalmazza, ezért az utolsó konfiguráció elején az ELFOGAD vagy ELVET állapotok valamelyikének kódja szerepel. A célunk egy olyan hálózat konstrukciója, ahol az outputcsúcsok száma 1 és majd ennek az értéke fogja eldönteni a problémát. Ennek érdekében a következő megállapodást tesszük: T kódolása legyen olyan, hogy az ELFOGAD kódja az 111...11 bitsorozat legyen. Az állapotok halmazának méretéből tudjuk, hogy milyen hosszú bitsorozatokra lesz szükség (a számosság kettes alapú logaritmusa felfelé kerekítve), de általában mást nem rögzítünk az állapotok kódolásánál, amíg nem látjuk magunk előtt a Turing-gépet és az állapotokat. Azt tudjuk, hogy a 111...11 sorozat ott van a bitsorozatok között és egy injektív leképezésre van szükségünk az állapotok halmazáról a bitsorozatok halmazára; érezzük, hogy van akkora szabadságunk, hogy megszorítás nélkül feltehessük: az ELFOGAD állapot kapja a 111...11 kódot. Ezt a megállapodást figyelembe véve már csak annyi dolgunk van, hogy az utolsó konfigurációban szereplő állapothoz tartozó bitsorozatot egy binárisan össze'és'elő hálózatba tesszük, amelynek egyetlen outputcsúcsa van. Ez a hálózat csak $\wedge$ kapukat tartalmaz, az inputcsúcsok balról jobbra haladva párba állnak és az értékük egy szinttel lejjebb lévő $\wedge$ kapuban találkozik; a pár nélkül maradt csúcs egy későbbi szinten talál magának párt. Ezzel a konstrukcióval látjuk, hogy pontosan akkor kapunk 1-est a hálózat végén, ha az utolsó állapot az ELFOGAD volt vagyis, ha $\omega \in L$ (ha a kimeneti csúcs értéke 0, akkora hálózat utolsó része inputként nem az 111...11 sorozatot kapta, vagyis az utolsó konfigurációban az állapot nem az ELFOGAD volt). (Azt is láthatjuk, hogy az ELVET állapot kódjára nem kell megállapodást tennünk.) ■

Megjegyzés: Az utolsó konfigurációból az állapoton kívüli részt ki lehet törölni, már nincs szükségünk rá.

Következmény$^+$: A $\mathcal{D}_n(\mathrm{T})$ fenti leírása konstruktív. Adott $\omega \in \Sigma^n$ és ismerjük T-t, akkor $\exists\, \mathrm{T}^+$ Turing-gép, ami ω-ból kiolvassa n-t és kiszámolja $\lceil\mathcal{D}_n(\mathrm{T})\rceil$-t. Ez LOG-tárban megoldható.

Megjegyzés:
• Ez a $\lceil\mathcal{D}_n(\mathrm{T})\rceil$ kód egy hármast ad: van egy irányított gráf, amelynek kódolása lehet az, hogy felsoroljuk a csúcskódokat egy listában és mindegyik csúcskód után rakunk egy ':'-ot, majd utána írjuk a csúcs kiszomszédainak kódlistáját. Minden csúcsnak van egy címkéje: ezt a csúcs és a : közé betehetjük; ez a címke lehet változó (csak index lehet), $\wedge$, $\vee$ és $\neg$ amelyek kódja megoldható 2 bittel.
• A három lehetséges kaput a hálózat *bázis*ának nevezzük. Vannak olyan hálózatok is, amelyek más logikai jeleket is használnak; de vigyázni kell, ha valaki például implikáció-kaput szeretne használni, akkor a gráfhoz is hozzá kell nyúlnia, mivel az implikáció nem szimmetrikus: ezért a gráfnál meg kell oldani, hogy melyik befokél adja a jobb oldalt és melyik a bal oldalt. Az $\wedge$ és a $\vee$ kapu azért praktikus, mert szimmetrikus 2-aritású függvények; az általunk használt bázist *de Morgan-bázis*nak szokás nevezni. Tehát egy de Morgan-bázisú hálózatban a kapuk két bittel kódolhatók és egy csúcs lehetséges kódolása a következő lehet: csúcs sorszáma | címke típusa | címke | output?. A címke lehet konstans (0/1), kapu (2 bit), változó ($\log_2 n$ bit, ami kódolja, hogy melyik indexű). Az "output?" egy bitet jelent: bináris válasz arra a kérdésre, hogy a csúcs outputcsúcs-e. A kódolás fontos, hogy el tudjuk dönteni, egy csúcsból mely csúcsokba vezetnek élek.
• A konstans számú csúcs kódolásához $O(\log n)$ tár elég: ennek kivitelezése hosszadalmas, a matematikai probléma programozási része.

■ Egy $\mathcal{P}$-teljes probléma

Definíció: HÁLÓZAT-KIÉRTÉKELÉS = { $\lceil$H, é$\rceil$: é-n 1-et számol ki H }.

Tétel: HÁLÓZAT-KIÉRTÉKELÉS $\mathcal{P}$-teljes.

Azaz i) HÁLÓZAT – KIÉRTÉKELÉS $\in \mathcal{P}$

ii) $L \in_T \mathcal{P}$ $\exists$ R redukciós algoritmus, amelyre

a) R log-tárat használ,

b) $\omega \underset{R}{\rightarrow} \tilde{\omega}$: $\omega \in L \Leftrightarrow \tilde{\omega} \in$ HÁLÓZAT – KIÉRTÉKELÉS.

Bizonyítás: Az eddigiekből kijön. ■

Megjegyzés:

• A hálózat definíciója pontosan megmondja, hogyan kell kiértkelni. Ha óvatosak vagyunk, akkor észrevesszük, hogy a hálózat definíciójában csak annyi megkötés volt, hogy az irányított gráf ne tartalmazzon irányított kört, tehát még meg kell keresnünk azt a sorrendet, amelyet felgöngyölítve megkapjuk az egyes csúcsokhoz kiszámított biteket - ezt *topologikus rendezés*nek nevezzük. Ha egy irányított gráf nem tartalmaz irányított kört, akkor van olyan csúcsa, amelynek befoka 0; ezt a csúcsot vesszük előre, eltöröljük a csúcsot az éleivel együtt, majd a maradék gráfban keresünk és törlünk; így rekurzívan kialakul egy sorrend. Ez egy polinomiális algoritmus.

• Az R redukciós algoritmus $\lceil \mathcal{D}_n(T) \rceil$-t és $\lceil \omega \rceil$-t fogja kiszámolni.

• A redukciós algoritmusnak nincs értelme polinomiális időt adni, mivel ennyi idővel már a redukció helyett el is dönthetné L-t. A redukciós algoritmus feladata, hogy a problémát egy másik, ugyanolyan nehéz problémába vigyen át rövidebb idő alatt.

• Még mindig fontos szem előtt tartanunk, hogy a teljesség azt jelenti, hogy ha valaki "okosat" tud mondani a HÁLÓZAT-ÉRTÉKELÉS problémáról, akkor az egész $\mathcal{P}$-ről, az összes $\mathcal{P}$-nehéz problémáról tud valami "okosat" mondani.

$\mathcal{NP}$-teljes problémák

▪ HÁLÓZAT-SAT

Megjegyzés: Fontos, hogy többes számban beszélünk. $\mathcal{P}$-teljes problémát csak egyet adtunk; vannak más $\mathcal{P}$-teljes problémák is, de nincs belőlük túl sok. Az $\mathcal{NP}$-teljes problémák különlegessége, hogy a matematika nagyon különböző területei produkálják őket.

Definíció: HÁLÓZAT – SAT = {$\lceil H \rceil$: létezik kiértékelése a változóknak, hogy H 1-et számoljon ki }.

Tétel: HÁLÓZAT – SAT $\mathcal{NP}$-teljes.

Azaz i) HÁLÓZAT – SAT $\in \mathcal{NP}$

ii) $L \in_T \mathcal{NP}$ $\exists$ R polinomiális idejű redukció, hogy $\omega \underset{R}{\rightarrow} \tilde{\omega}$, melyre $\omega \in L \Leftrightarrow \tilde{\omega} \in$ HÁLÓZAT – SAT.

Bizonyítás:

i) Tekintsünk egy olyan nemdeterminisztikus$_2$ Turing-gépet, amely

• a tanúszalag tartalmat, mint kiértékelést értelmezi,

• az input szalagon lévő H-t kiértékeli.

Láttuk, hogy ez polinomiális idő alatt elvégezhető. Ha az eredmény 1, akkor ELFOGAD állapotba, ha az eredmény 0, akkor NEM-STIMMEL állapotba lépünk.

ii) Legyen T az a tanúszalagos Turing-gép, amely megoldja L-t: ez tulajdonképpen egy polinomiális idejű, determinisztikus Turing-gép egy plusz szalaggal, ezért a konfigurációk ugyanúgy kódolhatók $\{0, 1\}^{\alpha_T \cdot t(n)}$ bitsorozatokkal, mint egy $\mathcal{P}$-beli gépnél (itt n az ω input hossza). Ez a T is $t(n)$ hosszú szalagokkal rendelkezik, mert a munkaszalagjának és a tanúszalagjának is levágjuk a $t(n)$ utáni részét. Ennél a T-nél is megadhatunk egy C_n^+ hálózatot, ami a rákövetkező konfigurációt számolja, csak a plusz szalag miatt kicsit eltér a korábbiakban látott hálózattól; ennek a segítségével felépíthető egy $\mathcal{D}_n^+$ hálózat is, ami szimulálja a Turing-gép számolását ω input és τ tanúszalag-tartalom alapján.

A *célunk*, hogy adjunk egy olyan redukciós algoritmust, amely egy ω L-be tartozási problémáját átviszi egy H hálózat kielégíthetőségének problémájába. Legyen az R feladata, hogy egy $\omega \in \Sigma^n$-re kiszámolja n-et, $\mathcal{D}_n^+$-t és beírja $\lceil\omega\rceil$-t az inputszalag tartalmát vivő változókhoz; tulajdonképpen R tud egy kérdést, amire a felhasználó választ vár és ezt beírja a T-t szimuláló hálózatba. Ekkor az outputcsúcs tartalmazni fogja a T által az ω-ból és τ-ból kiszámolt eredményt: ekkor 1 ≡ ELFOGAD, 0 ≡ ELVET. Ez jó lesz R-nek, mivel polinomiális (korábban láttuk) és

• $\omega \in L$ esetén T-nek létezik ELFOGADó futása, ami ekvivalens azzal, hogy létezik jó tanúszalag-tartalom, amiből következik, hogy ha a τ-t kódoló bitekhez a jó tanúszalag-tartalom kódját írjuk, akkor kielégítjük a hálózatot,

• $\omega \notin L$ esetén minden futás NEM-STIMMEL állapottal ér véget, amiből következik az állítás. ■

Célunk az elkövetkezendőkben: további $\mathcal{NP}$-teljes feladatok megadása.

Észrevétel (NAGYON FONTOS): C $\mathcal{NP}$-teljes nyelv és $C \prec_{\mathcal{P}} L \in \mathcal{NP}$, akkor L is $\mathcal{NP}$-teljes.

Indoklás: A feltételekből szeretnénk levezetni, hogy egy tetszőleges $M \in \mathcal{NP}$ esetén $M \prec_{\mathcal{P}} L$. Ekkor C teljességéből következik, hogy $M \prec_{\mathcal{P}} C$, a feltevésünk, hogy $C \prec_{\mathcal{P}} L$: ezeket felhasználva a $\prec_{\mathcal{P}}$ tranzitivitásából következik, hogy $M \prec_{\mathcal{P}} L$, ami adja az észrevételt. ■

Megjegyzés:

• Az észrevételünk egyszerű, mégis nagy jelentőségű. Eddig háromszor mentünk végig ugyanazon az úton: $\mathcal{NL}$ teljes nehézségét visszavezettük egy irányított gráfbeli irányított értelemben vett ELÉRHETŐSÉG-re; $\mathcal{P}$-beli, konfiguráció-sorozattal leírható problémát hálózattal írtuk le, az $\mathcal{NP}$ esetén pedig egy hálózat kielégíthetőségre vezettünk vissza. Mostmár egy $\mathcal{NP}$-beli nyelv $\mathcal{NP}$-teljességéhez csak azt kell belátnunk, hogy a HÁLÓZAT-SAT (esetleg más $\mathcal{NP}$-teljes nyelv) rávezethető; tehát csak egy redukciót kell adnunk, a teljességet már nem kell bizonyítanunk.

■ További $\mathcal{NP}$-teljes problémák

Definíció: BOOLE-EGYENLETRENDSZER = {$\lceil\varphi_1, \varphi_2, \ldots \varphi_l\rceil$: φ_i egy V változóhalmazon definiált ítéletkalkulusbeli formula és létezik V-n olyan kiértékelés, hogy mindegyik igaz értékű}

Megjegyzés: Tehát a BOOLE-EGYENLETRENDSZER-ben olyan formula l-esek vannak, melyekre a következő Boole-egyenletrendszernek létezik megoldása:

$$\begin{cases} \varphi_1 \leftrightarrow 1 \\ \varphi_2 \leftrightarrow 1 \\ \ldots \\ \varphi_l \leftrightarrow 1 \end{cases}$$

Tétel: BOOLE-EGYENLETRENDSZER $\mathcal{NP}$-teljes. *Azaz*

i) BOOLE – EGYENLETRENDSZER $\in \mathcal{NP}$,

ii) HÁLÓZAT – SAT $\prec_{\mathcal{P}}$ BOOLE – EGYENLETRENDSZER.

Bizonyítás:

i) Egy tanúszalag-tartalom segítségével polinomiális idő alatt kiértékelhető egy Boole-egyenletrendszer, így ennyi idő alatt ellenőrizhető, hogy a tanúszalag-tartalom jó volt-e.

ii) Legyen H egy hálózat (egy output csúccsal rendelkezik, mint a HÁLÓZAT-SAT nyelvbeliek). A hálózat alapértelmezett változói mellett az összes irányított élt tekintsük változónak (ez $\left|E\left(\vec{G}\right)\right|$ darab új változó, nevezzük őket *huzal*oknak), a kapukat pedig tekintsük egyenlőségnek (↔) (a címkéjük nem változik). Ekkor minden kapuhoz tartozik egy egyenlet:

• ha $\neg$ kapuhoz érünk h_1 huzallal és kifutunk h_2 huzallal, akkor az egyenlet: $\neg h_1 \leftrightarrow h_2$;
• ha $\wedge$ kapuhoz érünk h_1 és h_2 huzallal, majd kimegyünk h_3 huzallal, akkor: $h_1 \bigwedge h_2 \leftrightarrow h_3$;
• ha $\vee$ kapuhoz érünk h_1 és h_2 huzallal, és kimegyünk h_3 huzallal, akkor: $h_1 \bigvee h_2 \leftrightarrow h_3$.
Ezen kívül még minden inputcsúcs ad egy egyenletet: ha x az inputcsúcs és belőle egy h huzal indul ki, akkor az egyenlet: $x \leftrightarrow h$. Ehhez még hozzátesszük az $o \leftrightarrow 1$ (outputcsúcsra vonatkozó) egyenletet. Ennek a rendszernek a megoldhatósága ekvivalens azzal, hogy a hálózat 1-et számol ki, ha a hálózat (eredeti) változóit a megoldás szerint rögzítjük. Ezzel az állítást bebizonyítottuk. ■

Megjegyzés: A redukció során mindegy egyenlet legfeljebb 3 változót tartalmaz.

Emlékeztető: Legyen $V = \{x_1, x_2, ..., x_n\}$ változók egy halmaza. Ekkor $L = V \bigcup \bar{V}$a *literál*ok halmaza, ahol $\bar{V}$ a *negált változó*k halmaza. A $C \subseteq L$-t *klóz*nak nevezzük és a $\bigvee_{l \in C} l$ formulaként gondolunk rá. A φ formula *konjunktív normálforma* (röviden CNF), ha $\varphi = \{C_i\}$ klózok egy rendszere és a $\varphi = \bigwedge_i C_i$ formulaként gondolunk rá.

Definíció: SAT = $\{\ulcorner\varphi\urcorner$: φ CNF formula és kielégíthető$\}$

Tétel (*Cook-Levin*): SAT $\mathcal{NP}$-teljes.

Bizonyítás: Világos, hogy SAT $\in \mathcal{NP}$, hiszen egy SAT-ot megoldó nemdeterminisztikus$_2$ Turing-gép jó tanúszalag-tartalom esetén polinomiális idő alatt ellenőrzi, hogy a formula kielégíthető-e (az ellenőrzés könnyű, mivel egy klóz igaz, ha valamelyik eleme igaz értéket vesz fel, illetve a formula igaz, ha mindegyik klóz igaz). A korábbi észrevételünket felhasználva az a célunk, hogy redukciót adjunk a BOOLE-EGYENLETRENDSZER nyelvről. A redukciós algoritmusunknak az legyen a feladata, hogy a φ_i formulákat átírja CNF alakba, majd összekapcsolja a kapott CNF-eket konjunkcióval. Így egy CNF-et kapunk, amelynek a kielégíthetősége ekvivalens az egyenletrendszer megoldhatóságával. ■

Megjegyzés: Nagyvonalúan kijelenthetjük, hogy ezek a problémák matematikailag nem túl bonyolultak. Nem is az a nagy eredmény, hogy egy $\mathcal{NP}$-teljes probléma helyett mutattunk hármat, hanem, hogy a következőkben bemutatásra kerülő problémák, amelyek a gráfelméletből, számelméletből, a matematika különböző területeiről kerülnek elő magukban hordozzák $\mathcal{NP}$ teljes nehézségét és bármelyikről tudunk valamit mondani, akkor egész $\mathcal{NP}$-ről tudunk valamit mondani. Emellett nem szabad elfelejteni, hogy a faktorizáció is $\mathcal{NP}$-beli probléma.