

# Az informatika logikai alapjai

## 11. előadás

Vaszi György

[vaszil.gyorgy@inf.unideb.hu](mailto:vaszil.gyorgy@inf.unideb.hu)

I. emelet 110-es szoba

# A múlt órán

- Prenex normálforma
- Egy bizonyítási módszer: a szemantikus táblák módszerének adaptációja elsőrendű formulákra

## Definíció

Legyen  $L^{(1)} = \langle LC, Var, Con, Term, Form \rangle$  egy tetszőleges **elsőrendű nyelv**.

Az  $A \in Form$  formulát **prenex alakúnak nevezzük**, ha az alábbi két feltétel valamelyike teljesül:

1. az  $A$  formula kvantormentes, azaz sem a  $\forall$  sem a  $\exists$  kvantor nem szerepel benne;
2. az  $A$  formula  $Q_1 x_1 Q_2 x_2 \dots Q_n x_n B$  ( $n = 1, 2, \dots$ ) alakú, ahol
  - a.  $B \in Form$  kvantormentes formula;
  - b.  $x_1, x_2, \dots, x_n \in Var$  különböző változók;
  - c.  $Q_1, Q_2, \dots, Q_n \in \{\forall, \exists\}$  kvantorok.

## Megjegyzés

- A definíció értelmében ha az  $A$  formula kvantormentes, azaz egyetlen kvantor sem szerepel benne, akkor az  $A$  formula prenex alakú.

Például: Prenexformulák:  $\neg P(x, x), \forall x \forall y (Q(x, y) \supset \neg P(x))$

Nem prenexformula:  $\forall x \forall y Q(x, y) \supset \neg P(x)$

## Tétel

Legyen  $L^{(1)} = \langle LC, Var, Con, Term, Form \rangle$  egy tetszőleges **elsőrendű nyelv** és  $A \in Form$ .

Ekkor létezik olyan  $B \in Form$ , hogy

1. a  $B$  formula prenex alakú,
2.  $A \Leftrightarrow B$ .

### A prenex alakra hozás lépései

1.  $A (A \equiv B) \Leftrightarrow ((A \supset B) \wedge (B \supset A))$  ekvivalencia segítségével a **(materiális) ekvivalencia** műveletét fel kell oldani.
2. Változótiszta alakra hozás, azaz meg kell határozni az eredeti formulával kongruens **változóiban tiszta formulát**.
3. A **kvantifikáció De Morgan törvényeivel** és az **állításlogikában megtanult ekvivalenciákkal** el kell érni, hogy egyetlen negáció hatókörében se szerepeljen kvantor.
4. **Kvantormozgatási ekvivalenciák** alkalmazásával a kvantorok a formula elejére vihetők.

$$1. \neg \exists x A \Leftrightarrow \forall x \neg A$$

$$2. \neg \forall x A \Leftrightarrow \exists x \neg A$$

Például:

Ha  $x \notin \text{FreeVar}(A)$ , akkor

$$1. A \supset \forall x B \Leftrightarrow \forall x (A \supset B)$$

$$2. \forall x B \supset A \Leftrightarrow \exists x (B \supset A)$$

Ha  $x \notin \text{FreeVar}(A)$ , akkor

$$1. A \supset \exists x B \Leftrightarrow \exists x (A \supset B)$$

$$2. \exists x B \supset A \Leftrightarrow \forall x (B \supset A)$$

Hozzuk a

$$\forall x (\forall y Q(x, y) \supset \neg \exists x P(x)) \supset \forall y Q(x, y)$$

formulát prenex alakúra.

① Változóiban tiszta alakra hozás:

$$\forall v (\forall w Q(v, w) \supset \neg \exists z P(z)) \supset \forall y Q(x, y)$$

② De Morgan törvényeinek alkalmazása:

$$\forall v (\forall w Q(v, w) \supset \forall z \neg P(z)) \supset \forall y Q(x, y)$$

③ Kvantorkiemelés:

$$\forall v \exists w \forall z (\forall y (Q(v, w) \supset \neg P(z)) \supset \forall y Q(x, y))$$

④ Kvantorkiemelés:

$$\exists v \forall w \exists z \forall y ((Q(v, w) \supset \neg P(z)) \supset Q(x, y))$$

Mi történik, ha más sorrendben végezzük a kvantorkiemelést? (semmi, tábla)

# A múlt órán

- Prenex normálforma
- Egy bizonyítási módszer: a szemantikus táblák módszerének adaptációja elsőrendű formulákra

# Emlékeztető: szemantikus táblák nulladrendű logikában

Kielégíthető-e :

$$p \wedge (\neg q \vee \neg p)$$

$$p \wedge (\neg q \vee \neg p)$$

↓

$$p, \neg q \vee \neg p$$

↙

↘

$$p, \neg q$$

$$p, \neg p$$

nyitott

zárt

Kielégíthető, ha  $\{p, \neg q\}$  vagy  $\{p, \neg p\}$  kielégíthető.



# Elsőrendű formulahalmazok kielégíthetlensége vizsgálható hasonló módszerrel - Észrevételek:

1. Az egzisztenciális kvantifikált formula paramétereit levezetivel kezelendő.
2. Különböző egzisztenciális kvantifikált formulaiban különböző paramétereket kell használni.
3. Az univerzális formulaat ~~on~~ ~~is~~ levezetett paramétermel kell írni.
4. Az igaz nem feltétlenül igaz.
5. Figyelni kell a néhány algaluorai és ~~some~~ ~~is~~ ~~is~~.



Tela'4 : A saha' lya

$\alpha$	$\alpha_1$	$\alpha_2$
$\neg\neg A_1$	$A_1$	
$A_1 \wedge A_2$	$A_1$	$A_2$
$\neg(A_1 \vee A_2)$	$\neg A_1$	$\neg A_2$
$\neg(A_1 \supset A_2)$	$A_1$	$\neg A_2$

$\beta$	$\beta_1$	$\beta_2$
$\neg(B_1 \wedge B_2)$	$\neg B_1$	$\neg B_2$
$B_1 \vee B_2$	$B_1$	$B_2$
$B_1 \supset B_2$	$\neg B_1$	$B_2$

$\gamma$	$\gamma(a)$
$\forall x A(x)$	$A(a)$
$\neg \exists x A(x)$	$\neg A(a)$

$\delta$	$\delta(a)$
$\exists x A(x)$	$A(a)$
$\neg \forall x A(x)$	$\neg A(a)$

Hogyan alkalmazzuk a szabályokat, példa:

$$\neg (\forall x(p(x) \vee q(x)) \supset (\forall x p(x) \vee \forall x q(x)))$$

↓

$$\forall x(p(x) \vee q(x)), \neg (\forall x p(x) \vee \forall x q(x))$$

↓

$$\forall x(p(x) \vee q(x)), \neg \forall x p(x), \neg \forall x q(x)$$

↓

$$\forall x(p(x) \vee q(x)), \exists x \neg p(x), \exists x \neg q(x)$$

↓

$$\forall x(p(x) \vee q(x)), \exists x \neg p(x), \neg q(a_1)$$

↓

$$\forall x(p(x) \vee q(x)), \neg p(a_2), \neg q(a_1).$$

↓

$$\downarrow$$

$$\forall x(p(x) \vee q(x)), \neg p(a_2), \neg q(a_1)$$

$$\downarrow$$

$$\forall x(p(x) \vee q(x)), p(a_1) \vee q(a_1), \neg p(a_2), \neg q(a_1)$$

$$\downarrow$$

$$\forall x(p(x) \vee q(x)), p(a_2) \vee q(a_2), p(a_1) \vee q(a_1), \neg p(a_2), \neg q(a_1).$$

$$\swarrow \quad \searrow$$

$$\forall x(p(x) \vee q(x)), p(a_2), p(a_1) \vee q(a_1), \neg p(a_2), \neg q(a_1)$$

$$\swarrow \quad \searrow$$

$$\forall x(p(x) \vee q(x)), p(a_2), p(a_1),$$

$$\neg p(a_2), \neg q(a_1)$$

zählt

$$\forall x(p(x) \vee q(x)), p(a_2),$$

$$q(a_1), \neg p(a_2), \neg q(a_1)$$

zählt

$$\forall x(p(x) \vee q(x)), q(a_2), p(a_1) \vee q(a_1), \neg p(a_2),$$

$$\neg q(a_1)$$

$$\swarrow \quad \searrow$$

$$\forall x(p(x) \vee q(x)), q(a_2),$$

$$p(a_1), \neg p(a_2), \neg q(a_1)$$

neu zählt

$$\forall x(p(x) \vee q(x)), q(a_2),$$

$$q(a_1), \neg p(a_2), \neg q(a_1)$$

zählt

Keg es pilda - lehetőségese  
vezeteni a'gar

$$\forall x \exists y p(x, y), \exists y p(a_1, y)$$

↓

$$\forall x \exists y p(x, y), p(a_1, a_2)$$

↓

$$\forall x \exists y p(x, y), \exists y p(a_2, y), p(a_1, a_2)$$

↓

$$\forall x \exists y p(x, y), p(a_2, a_3), p(a_1, a_2).$$

és így tovább ...

## Teljesít : Az algoritmus

Adott :  $\emptyset$  famula  $\swarrow$  fa

Érdek : Egy gráf (nematiós, felele), ahol az ágak

végződhetnek zárt levéllel, nyílt levéllel, vagy lehetnek végtelenek.

záró ág

nyílt ág

- A fa egy csúcsa  $l$ ,  $U(l)$ ,  $C(l)$

Kérdésben  $\emptyset$  is a leme  
lévő csúcsok.

famula

## Ar algant tuis / lalg lates

Veegmii 2 eeg llooleet, ami uic yiffner uen  
2istner jeli lue. Fi'gelle a comadre  
legni 4 an adalhiat:

1. • Ha  $U(l)$ -her uen uenpallmenter literalfoer,  
jeli'fii l-et 2aistral
2. • Ha  $U(l)$ -her uener ojan famla'4, ami'4 uen  
literalfoer, uegmii'4 les  $\alpha, \beta, \gamma$  famla'4, A-t  
- Ha  $A\alpha$  famla, jaijind el no'raisan  
- Ha  $A\beta$  famla, jaijind  $k$  el no'raisan  
2 a uastander  
hulueris  
u uaiter tene

## Az algoritmus / feladatok 2

- Ha  $A$   $\sigma$  formula, akkor  $l'$  egy új csúcs, ahol

$$U(l') = U(l) - \{A\} \cup \{\sigma(a')\}$$

$$C(l') = C(l) \cup \{a'\}$$

$a'$  új konstans

Ha nincs komplement literálpár és elfogytak az  $\alpha$ ,  $\beta$ ,  $\delta$  formulák:

- Az  $U(l)$  belüli  $\gamma$  formulák leegyszeresítésére

$$\{\gamma_{l_1}, \dots, \gamma_{l_m}\} \text{ és } C(l) = \{c_{l_1}, \dots, c_{l_k}\}$$

$\sigma(a)$  a  
nagyság  
szint

$$U(l') = U(l) \cup \left\{ \bigcup_{i=1}^m \bigcup_{j=1}^k \gamma_{l_i}(c_{l_j}) \right\}$$

$$C(l') = C(l)$$

$\delta$	$\delta(a)$
$\exists x A(x)$	$A(a)$
$\neg \forall x A(x)$	$\neg A(a)$

- Ha csak  $\gamma$  formulák vannak, és  $U(l') = U(l)$ , akkor  $l$ -et jelöljük újítottunk.



vor uns sein ag:

- Zeit, da Zeit leinlich neigro'di'
- Zitt, da Zitt leinlich neigro'di',  
von wegtelen

von laichla:

- Zeit, da wir der a'ga Zeit
- Zitt, wir lichen.

# A múlt órán

- Prenex normálforma
- Egy bizonyítási módszer: a szemantikus táblák módszerének adaptációja elsőrendű formulákra
- Helyesség? Teljesség?

(1) • A formula kielégíthetetlen  
minden letelezésre  $(\Leftrightarrow A_{\text{„letelezés”}})$

akkor és  
csak akkor,  
ha a

Helyes és helyesre

A nemantikus formula letelezési egy módszer a  
formula kielégíthetőségének eldöntésére.

• A módszer Helyes:

Ha a módszerrel kapott eredmény szerint a formula kielégíthetetlen,  
akkor a formula valóban kielégíthetetlen.

• A módszer Helyes:

Bármilyen kielégíthetetlen formulára alkalmazom is a módszert,  
a módszerrel kapott eredmény az, hogy a formula kielégíthetetlen.

(1). A formula kielégíthetetlen  
minden lezárta zárt  $(\Leftrightarrow A_{\text{„tábla zárt”}})$

akkor és  
csak akkor,  
ha a

Helyes és teljes

A nemantikus tábla konstrukcióján egy módszer a  
formula kielégíthettségének eldöntésére.

- A módszer Helyes:

Ha a kapott tábla zárt,  
akkor a formula valóban kielégíthetetlen.

- A módszer teljes:

Bármilyen kielégíthetetlen formulára alkalmazom is a módszert,  
minden kapott tábla zárt.

# Tehát:

Beláttuk (helyesség/teljesség):

- Ha **egy tábla zárt**, akkor a hozzá tartozó formulának **nincs modellje**.
- Ha **van** a formulához tartozó **táblák között nyitott**, akkor a formulának **van modellje**.


Vegyük észre:

Ha van a formulához tartozó táblák közt zárt, akkor mindegyik zárt. (Miért?)


Ha van a formulához tartozó táblák közt nyitott, akkor mindegyik nyitott. (Miért?)

# Tehát:

Vegyük észre:

Ha van a formulához tartozó táblák közt zárt, akkor mindegyik zárt. (Miért?) 

Mert a módszer helyes, azaz ha a formulához tartozó egyik tábla zárt, akkor a formulának nincs modellje. Viszont, mivel a módszer teljes, ha a formulának nincs modellje, akkor az összes hozzá tartozó tábla zárt.

Ha van a formulához tartozó táblák közt nyitott, akkor mindegyik nyitott. (Miért?) 

Az 1. pont miatt

# A múlt órán

- Prenex normálforma
- Egy bizonyítási módszer: a szemantikus táblák módszerének adaptációja elsőrendű formulákra
- A szemantikus táblák módszerének elsőrendű adaptációja helyes és teljes



# A mai órán

- Az elsőrendű logikában *algoritmikusan nem eldönthető a formulák érvényességének kérdése*

# Kiállítások

A nulladrendű logikában  
így volt. Érvényesek ezek  
a pontok most is?

- $A \in \mathcal{F}_{oma}$  akkor és csak akkor van modellje,  
ha  $T$  igaz (ezen részt).
- $A \in \mathcal{F}_{on}$  érvényes (logikai törvény) akkor és  
csak akkor, ha  $\neg A$  nem tartozik seholba tart.
- A nematikus seholba tartozó jel nem rögzítve  
eldőntendő, hogy egy formula érvényes-e.

"eldőntési eljárás"  $\rightarrow$

# Kétféle művelet

A nulladrendű logikában  
így volt. Érvényesek ezek  
a pontok most is?

- $A \in \mathcal{F}_{oma}$  akkor és csak akkor van modellje,  
ha  $T$  igaz (nem rejt).
- $A \in \mathcal{F}_{on}$  érvényes (logikai törvény) akkor és  
csak akkor, ha  $\neg A$  nem tartozik seholba tart.
- A nemtriviális seholba tartozásokról nem mindig lehet  
elbírálni, hogy egy formula érvényes-e.

„eldöntési eljárás” = algoritmus a kérdés eldöntésére, azaz:

- mechanikusan, pontos szabályok szerint végezhető  
„számítási” lépések sorozata, amelyek
- véges sok lépés után véget ér és „igen – nem” típusú választ  
produkál

# Közelítő

Az elsőrendű  
logikában a  
3. pont nem teljesül

- $A \in \text{Forma}$  akkor és csak akkor van modellje,  
ha  $T$  igaz (nem rejt).
- $A \in \text{Form}$  érvényes (logikai tétel) akkor és  
csak akkor, ha a  $\neg A$  nem tartozik a halmazba.
- ~~A nem mechanikus eljárással nem lehet megmondani,  
elvégezhető-e, hogy egy formula érvényes-e.~~

„elöntési eljárás” = algoritmus a kérdés elöntésére, azaz:

- mechanikusan, pontos szabályok szerint végezhető  
„számítási” lépések sorozata, amelyik
- véges sok lépés után véget ér és „igen – nem” típusú választ  
produkál

## Kiértékelés elsőrendű logika esetén

- $A \in \mathcal{F}_{\text{oma}}$  akkor is szer akkor van modellje ha  $T$  igaz (nem rejt).
- $A \in \mathcal{F}_{\text{on}}$  mindig (logikai tényszerű) akkor is igaz akkor, ha a  $\neg A$  nem lehet igaz.
- A tábla nem feltétlen konstruálható meg véges sok lépésben (a konstrukciós eljárás nem feltétlen fejeződik be), azaz a módszer **általánosan nem alkalmazható** formulák **érvényességének** eldöntésére.

# Miért akadálya az „eldöntésnek”, hogy végtelen ágak is előfordulhatnak?

- Mert az eldöntési eljárásnak (algoritmusnak) véges sok lépésben be kell fejeződni.
  - Ha a **formula érvényes**, akkor negáltnak nincs modellje, azaz a **negálthoz tartozó tábla** minden ága **zárt** lesz  
→ ez véges sok lépésben kiderül
  - Ha a **formula ellentmondás**, akkor nincs modellje, azaz a **formulához tartozó tábla** minden ága **zárt** lesz  
→ ez véges sok lépésben kiderül
  - Ha a **formula nem érvényes** ugyan, de **van modellje**, akkor a **formulának és a negáltjának a táblájában** is van **nyitott ág**  
→ ha a nyitott ág végtelen, akkor ez **nem feltétlen** derül ki **véges** sok lépésben. Nem mindig világos hogy egy adott ágat „növelő” lépések egy idő után nem „fogynak-e el”, vagyis az, hogy az ág esetleg mégis lezárul-e.

Van esetleg más eljárás (algoritmus) elsőrendű formulák érvényességének eldöntésére?

Nincs.



# A mai órán

- Az elsőrendű logikában *algoritmikusan nem eldönthető a formulák érvényességének kérdése*

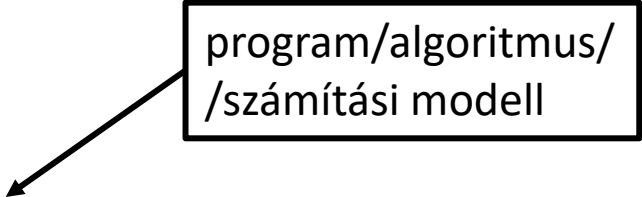
Van esetleg más eljárás (algoritmus) elsőrendű formulák érvényességének eldöntésére?

Nincs.

Bizonyítás ötlet: Ha lenne, akkor lenne eljárás (algoritmus) a „két regiszteres gépek” megállási problémájának eldöntésére is, amiről viszont tudjuk, hogy nincsen.

# Mi az, a „megállási probléma”?

program/algorithmus/  
/számítási modell



- A probléma: Adott **számítási eszköz** adott **bemenettel** elindítva megáll vagy nem (végtelen ciklusba kerül)?
- Eldönthető-e a megállási probléma:  
**Van** olyan **algorithmus**, ami tetszőleges **(sz,b)** párról megmondja, hogy **sz** eszköz **b** bemenettel indítva megáll-e?

számítási eszköz



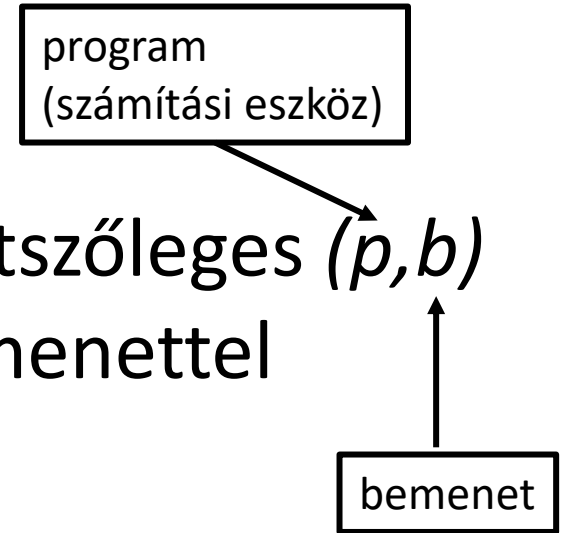
bemenet



# A megállási probléma gyakran eldönthetetlen – pl. programkódok esetén

Például:

Nem létezik olyan algoritmus, ami tetszőleges  $(p, b)$  párról eldönti, hogy  $p$  megáll-e  $b$  bemenettel indítva.



Azaz, nincs ilyen program:

$halts(p, b)$ : igaz – ha  $p$  megáll  $b$ -n

hamis – ha  $p$  végtelen ciklusba  
kerül  $b$ -n

# A megállási probléma gyakran eldönthetetlen - pl. programkódok esetén

Ha létezne

*halts(p,b)*: igaz – ha *p* megáll *b*-n

hamis – ha *p* végtelen ciklusba  
kerül *b*-n

akkor írhatnánk egy ilyen programot:

```
void contrarian(int input) {  
    if (halts(contrarian, input))  
        while (TRUE) { /* loop infinitely */ }  
}
```

Mi a gond ezzel a programmal?

Az ötlet tehát: A megállási probléma  
eldönthetetlen két regiszteres gépek esetén is  
(ezt elhisszük, ebből indulunk ki)

Két regiszteres gép:

- $x, y$  regiszterek
- $P = \{L_0, \dots, L_n\}$  program, ahol az utasítások lehetnek:
  - $x = x + 1;$
  - $y = y + 1;$
  - $\text{if } (x == 0) \text{ goto } L_j; \text{ else } x = x - 1;$
  - $\text{if } (y == 0) \text{ goto } L_j; \text{ else } y = y - 1;$
  - $\text{halt}$
- Kezdő konfiguráció:  $(L_0, m, 0)$   
megállási konfiguráció:  $(\bar{L}_n, x, y)$

Megállási probléma: döntjük el, hogy egy adott  
program nulla regiszterértékekkel indítva megáll-e

$$S_M = \left( p_0(a, a) \wedge \bigwedge_{i=0}^{n-1} S_i \right) \supset \exists z_1 \exists z_2 p_n(z_1, z_2),$$

ahol: $L_i$	$S_i$
$x = x + 1;$	$\forall x \forall y (p_i(x, y) \supset p_{i+1}(s(x), y))$
$y = y + 1;$	$\forall x \forall y (p_i(x, y) \supset p_{i+1}(x, s(y)))$
if (x == 0) goto Lj; else x = x - 1;	$\forall x (p_i(a, x) \supset p_j(a, x)) \wedge$ $\forall x \forall y (p_i(s(x), y) \supset p_{i+1}(x, y))$
if (y == 0) then goto Lj; else y = y - 1;	$\forall x (p_i(x, a) \supset p_j(x, a)) \wedge$ $\forall x \forall y (p_i(x, s(y)) \supset p_{i+1}(x, y))$

Egy interpretáció (U: a regiszterekbe írható számok):

- $p_i(x, y)$  predikátum teljesül, ha  $P = \{L_0, \dots, L_n\}$  program végrehajtása közben  $L_i$  a következő utasítás és  $x, y$  a regiszterek értéke
- $a$  jelentése 0
- $s(x)$  jelentése  $x+1$



$$S_M = \left( \boxed{p_0(a, a)} \wedge \bigwedge_{i=0}^{n-1} S_i \right) \supset \exists z_1 \exists z_2 p_n(z_1, z_2),$$

ahol:

$L_i$	$S_i$
$x = x + 1;$	$\forall x \forall y (p_i(x, y) \supset p_{i+1}(s(x), y))$
$y = y + 1;$	$\forall x \forall y (p_i(x, y) \supset p_{i+1}(x, s(y)))$
if $(x == 0)$ goto $L_j$ ; else $x = x - 1;$	$\forall x (p_i(a, x) \supset p_j(a, x)) \wedge$ $\forall x \forall y (p_i(s(x), y) \supset p_{i+1}(x, y))$
if $(y == 0)$ then goto $L_j$ ; else $y = y - 1;$	$\forall x (p_i(x, a) \supset p_j(x, a)) \wedge$ $\forall x \forall y (p_i(x, s(y)) \supset p_{i+1}(x, y))$

$S_M$  leírja a kétszámlálós gép működését, ha  $P = \{L_0, \dots, L_n\}$ .

- $p_0(a, a)$ : A kezdőkonfiguráció, azaz a regiszterek tartalma 0, a következő utasítás  $L_0$
- $S_i$  leírja az  $L_i$  utasítást, hiszen  $p_i(x, y)$  teljesül, ha  $L_i$  a következő utasítás és  $x, y$  a regiszterek értéke

$$S_M = \left( p_0(a, a) \wedge \bigwedge_{i=0}^{n-1} S_i \right) \supset \exists z_1 \exists z_2 p_n(z_1, z_2),$$

ahol:

$L_i$

`x = x + 1;`

`y = y + 1;`

`if (x == 0) goto Lj;`  
`else x = x - 1;`

`if (y == 0) then goto Lj;`  
`else y = y - 1;`

$S_i$

$\forall x \forall y (p_i(x, y) \supset p_{i+1}(s(x), y))$

$\forall x \forall y (p_i(x, y) \supset p_{i+1}(x, s(y)))$

$\forall x (p_i(a, x) \supset p_j(a, x)) \wedge$

$\forall x \forall y (p_i(s(x), y) \supset p_{i+1}(x, y))$

$\forall x (p_i(x, a) \supset p_j(x, a)) \wedge$

$\forall x \forall y (p_i(x, s(y)) \supset p_{i+1}(x, y))$

$S_M$  leírja a kétszámlálós gép működését, ha  $P = \{L_0, \dots, L_n\}$ .

- $p_0(a, a)$ : A kezdőkonfiguráció, azaz a regiszterek tartalma 0, a következő utasítás  $L_0$
- $S_i$  leírja az  $L_i$  utasítást, hiszen  $p_i(x, y)$  teljesül, ha  $L_i$  a következő utasítás és  $x, y$  a regiszterek értéke

$$S_M = \left( p_0(a, a) \wedge \bigwedge_{i=0}^{n-1} S_i \right) \supset \exists z_1 \exists z_2 p_n(z_1, z_2),$$

ahol:

$L_i$

$x = x + 1;$

$y = y + 1;$

if (x == 0) goto Lj;  
    else x = x - 1;

if (y == 0) then goto Lj;  
    else y = y - 1;

$S_i$

$\forall x \forall y (p_i(x, y) \supset p_{i+1}(s(x), y))$

$\forall x \forall y (p_i(x, y) \supset p_{i+1}(x, s(y)))$

$\forall x (p_i(a, x) \supset p_j(a, x)) \wedge$

$\forall x \forall y (p_i(s(x), y) \supset p_{i+1}(x, y))$

$\forall x (p_i(x, a) \supset p_j(x, a)) \wedge$

$\forall x \forall y (p_i(x, s(y)) \supset p_{i+1}(x, y))$

$S_M$  leírja a kétszámlálós gép működését, ha  $P = \{L_0, \dots, L_n\}$ .

- $p_0(a, a)$ : A kezdőkonfiguráció, azaz a regiszterek tartalma 0, a következő utasítás  $L_0$
- $S_i$  leírja az  $L_i$  utasítást, hiszen  $p_i(x, y)$  teljesül, ha  $L_i$  a következő utasítás és  $x, y$  a regiszterek értéke

$$S_M = \left( p_0(a, a) \wedge \bigwedge_{i=0}^{n-1} S_i \right) \supset \exists z_1 \exists z_2 p_n(z_1, z_2),$$

ahol:

$L_i$

$x = x + 1;$

$y = y + 1;$

if (x == 0) goto Lj;  
else x = x - 1;

if (y == 0) then goto Lj;  
else y = y - 1;

$S_i$

$\forall x \forall y (p_i(x, y) \supset p_{i+1}(s(x), y))$

$\forall x \forall y (p_i(x, y) \supset p_{i+1}(x, s(y)))$

$\forall x (p_i(a, x) \supset p_j(a, x)) \wedge$   
 $\forall x \forall y (p_i(s(x), y) \supset p_{i+1}(x, y))$

$\forall x (p_i(x, a) \supset p_j(x, a)) \wedge$   
 $\forall x \forall y (p_i(x, s(y)) \supset p_{i+1}(x, y))$

$S_M$  leírja a kétszámlálós gép működését, ha  $P = \{L_0, \dots, L_n\}$ .

- $p_0(a, a)$ : A kezdőkonfiguráció, azaz a regiszterek tartalma 0, a következő utasítás  $L_0$
- $S_i$  leírja az  $L_i$  utasítást, hiszen  $p_i(x, y)$  teljesül, ha  $L_i$  a következő utasítás és  $x, y$  a regiszterek értéke

$$S_M = \left( p_0(a, a) \wedge \bigwedge_{i=0}^{n-1} S_i \right) \supset \exists z_1 \exists z_2 p_n(z_1, z_2),$$

ahol:

$L_i$

$x = x + 1;$

$y = y + 1;$

if (x == 0) goto Lj;  
    else x = x - 1;

if (y == 0) then goto Lj;  
    else y = y - 1;

$S_i$

$\forall x \forall y (p_i(x, y) \supset p_{i+1}(s(x), y))$

$\forall x \forall y (p_i(x, y) \supset p_{i+1}(x, s(y)))$

$\forall x (p_i(a, x) \supset p_j(a, x)) \wedge$   
 $\forall x \forall y (p_i(s(x), y) \supset p_{i+1}(x, y))$

$\forall x (p_i(x, a) \supset p_j(x, a)) \wedge$   
 $\forall x \forall y (p_i(x, s(y)) \supset p_{i+1}(x, y))$

$S_M$  leírja a kétszámlálós gép működését, ha  $P = \{L_0, \dots, L_n\}$ .

- $p_0(a, a)$ : A kezdőkonfiguráció, azaz a regiszterek tartalma 0, a következő utasítás  $L_0$
- $S_i$  leírja az  $L_i$  utasítást, hiszen  $p_i(x, y)$  teljesül, ha  $L_i$  a következő utasítás és  $x, y$  a regiszterek értéke

$$S_M = \left( p_0(a, a) \wedge \bigwedge_{i=0}^{n-1} S_i \right) \supset \exists z_1 \exists z_2 p_n(z_1, z_2),$$

ahol:

$L_i$	$S_i$
$x = x + 1;$	$\forall x \forall y (p_i(x, y) \supset p_{i+1}(s(x), y))$
$y = y + 1;$	$\forall x \forall y (p_i(x, y) \supset p_{i+1}(x, s(y)))$
if (x == 0) goto Lj; else x = x - 1;	$\forall x (p_i(a, x) \supset p_j(a, x)) \wedge$ $\forall x \forall y (p_i(s(x), y) \supset p_{i+1}(x, y))$
if (y == 0) then goto Lj; else y = y - 1;	$\forall x (p_i(x, a) \supset p_j(x, a)) \wedge$ $\forall x \forall y (p_i(x, s(y)) \supset p_{i+1}(x, y))$

$S_M$  leírja a kétszámlálós gép működését, ha  $P = \{L_0, \dots, L_n\}$ .

- $p_0(a, a)$ : A kezdőkonfiguráció, azaz a regiszterek tartalma 0, a következő utasítás  $L_0$
- $S_i$  leírja az  $L_i$  utasítást, hiszen  $p_i(x, y)$  teljesül, ha  $L_i$  a következő utasítás és  $x, y$  a regiszterek értéke

$$S_M = \left( p_0(a, a) \wedge \bigwedge_{i=0}^{n-1} S_i \right) \supset \exists z_1 \exists z_2 p_n(z_1, z_2),$$

ahol:

$L_i$

$x = x + 1;$

Ha a kezdő konfiguráció  $(L_0, 0, 0)$  és a program  $P = \{L_0, \dots, L_{n-1}, L_n = \text{HALT}\}, \dots$

$\text{if } (x == 0) \text{ then goto } L_j;$

$\text{else } x = x - 1;$

$\text{if } (y == 0) \text{ then goto } L_j;$

$\text{else } y = y - 1;$

$S_i$

$\forall x \forall y (p_i(x, y) \supset p_{i+1}(x, y))$

... akkor valamilyen  $z_1, z_2$ -vel eljutunk a  $(\text{HALT}, z_1, z_2)$  megállási konfigurációba.

$\forall x \forall y (p_i(s(x), y) \supset p_{i+1}(x, y))$

$\forall x (p_i(x, a) \supset p_j(x, a)) \wedge$

$\forall x \forall y (p_i(x, s(y)) \supset p_{i+1}(x, y))$

$S_M$  leírja a kétszámlálós gép működését, ha  $P = \{L_0, \dots, L_n\}$ .

- $p_0(a, a)$ : A kezdőkonfiguráció, azaz a regiszterek tartalma 0, a következő utasítás  $L_0$
- $S_i$  leírja az  $L_i$  utasítást, hiszen  $p_i(x, y)$  teljesül, ha  $L_i$  a következő utasítás és  $x, y$  a regiszterek értéke

$$S_M = \left( p_0(a, a) \wedge \bigwedge_{i=0}^{n-1} S_i \right) \supset \exists z_1 \exists z_2 p_n(z_1, z_2),$$

$S_M$  akkor és csak akkor érvényes, ha az  $M$  kétszámlálós gép megáll az  $(L_0, 0, 0)$  konfigurációból indítva.



$$S_M = \left( p_0(a, a) \wedge \bigwedge_{i=0}^{n-1} S_i \right) \supset \exists z_1 \exists z_2 p_n(z_1, z_2),$$

1. Ha  $M$  megáll, akkor  $S_M$  érvényes
2. Ha  $S_M$  érvényes, akkor  $M$  megáll

$$S_M = \left( p_0(a, a) \wedge \bigwedge_{i=0}^{n-1} S_i \right) \supset \exists z_1 \exists z_2 p_n(z_1, z_2),$$

1. Ha  $M$  megáll, akkor  $S_M$  érvényes
  - Be kell látni, hogy ha  $M$  megáll, akkor  $S_M$  minden interpretációban igaz.
  - Ha a bal oldal hamis, akkor  $S_M$  automatikusan igaz, azok az interpretációk érdekesek, melyekben a bal oldal igaz.
  - Ha a bal oldal igaz és  $M$  megáll, akkor a jobb oldal is igaz (indukcióval precízen be kell látni).
  - Azaz:  $S_M$  minden interpretációban igaz.
2. Ha  $S_M$  érvényes, akkor  $M$  megáll

$$S_M = \left( p_0(a, a) \wedge \bigwedge_{i=0}^{n-1} S_i \right) \supset \exists z_1 \exists z_2 p_n(z_1, z_2),$$

1. Ha  $M$  megáll, akkor  $S_M$  érvényes
2. Ha  $S_M$  érvényes, akkor  $M$  megáll
  - Be kell látni, hogy ha  $S_M$  minden interpretációban igaz, akkor  $M$  megáll.
  - Ha  $S_M$  minden interpretációban igaz, akkor igaz az  $M$  gépet leíró interpretációban is.
  - $p_0(0,0)$  teljesül, mert ez a kezdőkonfiguráció, tehát  $p_1(z_1, z_2)$ : teljesül valamilyen  $z_1, z_2$ -re
  - és így tovább egészen  $p_n(z_1, z_2)$ -ig (indukcióval precízen be kell látni) ami a  $(HALT, z_1, z_2)$  konfiguráció megfelelője
  - Azaz:  $M$  megáll.

$$S_M = \left( p_0(a, a) \wedge \bigwedge_{i=0}^{n-1} S_i \right) \supset \exists z_1 \exists z_2 p_n(z_1, z_2),$$

ahol:

$L_i$	$S_i$
$x = x + 1;$	$\forall x \forall y (p_i(x, y) \supset p_{i+1}(s(x), y))$
$y = y + 1;$	$\forall x \forall y (p_i(x, y) \supset p_{i+1}(x, s(y)))$
if $(x == 0)$ goto $L_j$ ; else $x = x - 1;$	$\forall x (p_i(a, x) \supset p_j(a, x)) \wedge$ $\forall x \forall y (p_i(s(x), y) \supset p_{i+1}(x, y))$
if $(y == 0)$ then goto $L_j$ ; else $y = y - 1;$	$\forall x (p_i(x, a) \supset p_j(x, a)) \wedge$ $\forall x \forall y (p_i(x, s(y)) \supset p_{i+1}(x, y))$

Tehát: Ez a formula akkor és csak akkor érvényes, ha a program ami alapján készült  $0,0$ -t tartalmazó regiszterekkel indítva megáll.

**Az érvényesség eldöntése eldöntően a megállás kérdését is, azaz az érvényesség sem lehet eldönthető.**

*„Az érvényesség eldöntése eldöntené a megállás kérdését is, azaz az érvényesség sem lehet eldönthető.”*

Miről is van szó?

- A kétregiszteres gépek megállási problémája algoritmikusan eldönthetetlen
  - Nincs ilyen algoritmus:  
Bemenet: a gép leírása,  
Kimenet: igen/nem (megáll-e az üres bemeneten)
- Viszont: Képesek vagyunk tetszőleges  $M$  géphez (algoritmikusan) olyan  $A$  formulát konstruálni, hogy  $A$  akkor és csak akkor érvényes, ha  $M$  megáll az üres bemeneten
- Ha létezne a formulák érvényességét eldöntő algoritmus, akkor megvizsgálhatnám vele az  $A$  formulát

*„Az érvényesség eldöntése eldöntené a megállás kérdését is, azaz az érvényesség sem lehet eldönthető.”*

Miről is van szó?

- Ha létezne a formulák érvényességét eldöntő algoritmus, akkor megvizsgálhatnám vele az A formulát:

- Ha az A érvényes, akkor M megáll
- Ha az A nem érvényes, akkor M nem áll meg

→ Létrehoztam egy algoritmust a kétszámlálós gépek megállási problémájára

*„Az érvényesség eldöntése eldöntené a megállás kérdését is, azaz az érvényesség sem lehet eldönthető.”*

Miről is van szó?

→ Létrehoztam egy algoritmust a kétszámlálós gépek megállási problémájára:

- Bemenet: M, 1. lépés: M alapján megkonstruálom A-t  
2. lépés: futtatom az érvényességvizsgálatot  
A-n

Kimenet: igen, ha A érvényes, nem ha nem az

# A múlt és mai órán

- Egy bizonyítási módszer: a szemantikus táblák módszerének adaptációja elsőrendű formulákra
- Helyesség, teljesség
- Az elsőrendű logikában *algoritmikusan nem eldönthető a formulák érvényességének kérdése*