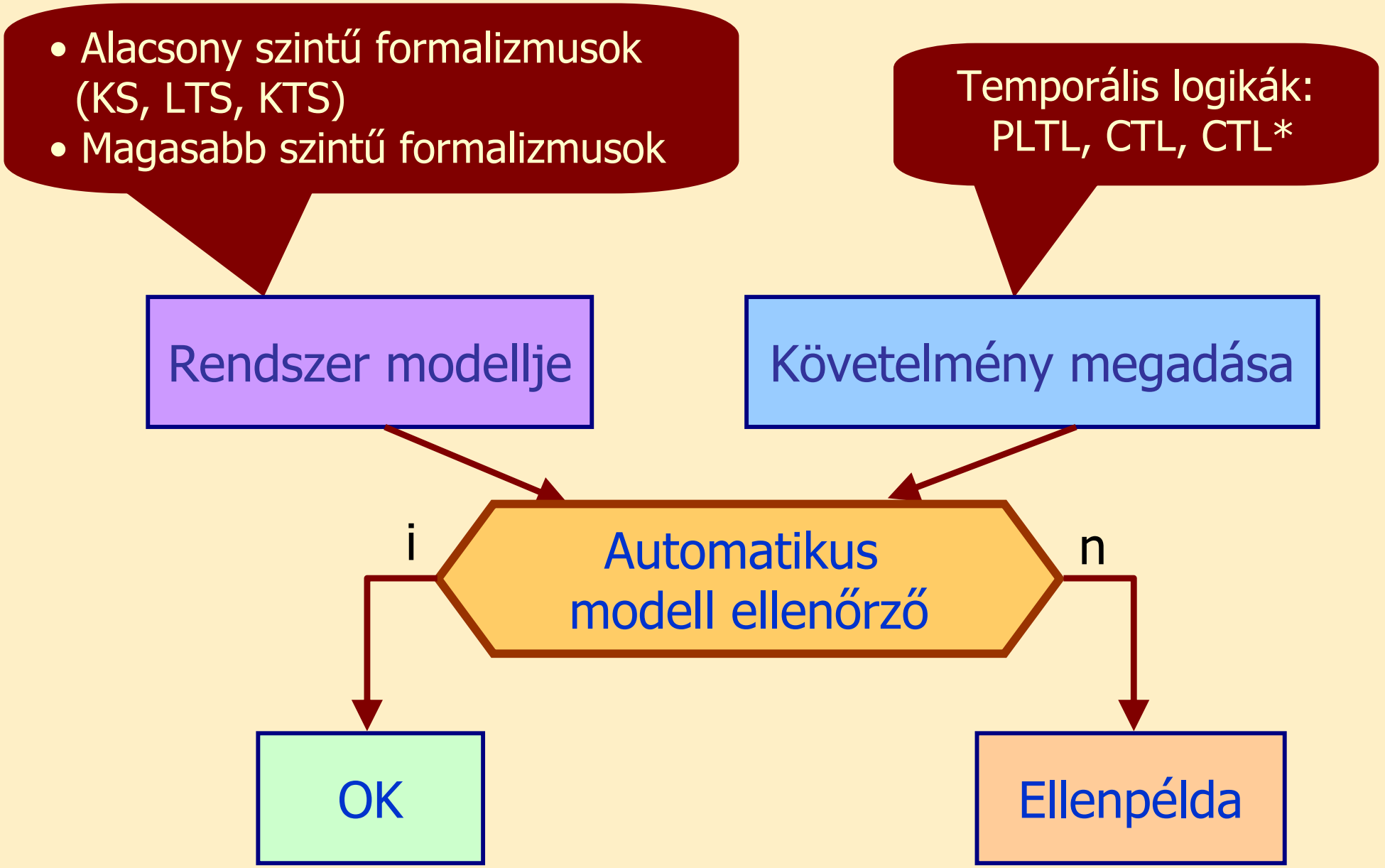
A modellellenőrzés hatékony technikái

dr. Majzik István
dr. Pataricza András
dr. Bartha Tamás

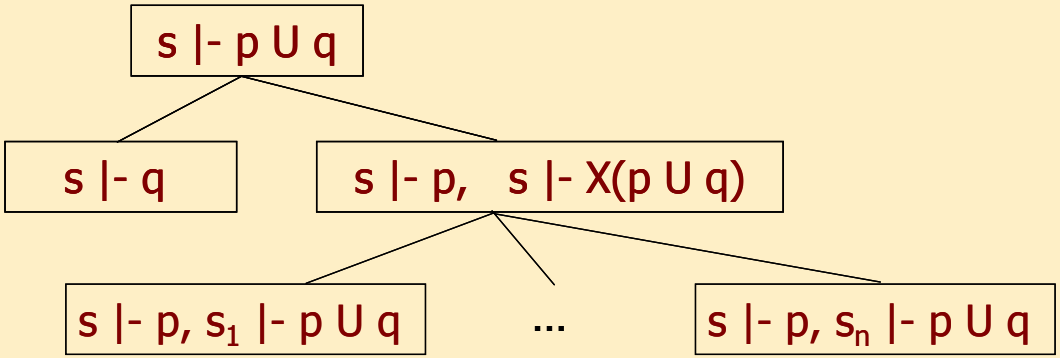
BME Méréstechnika és Információs Rendszerek Tanszék

Motiváció: Mit szeretnénk elérni?

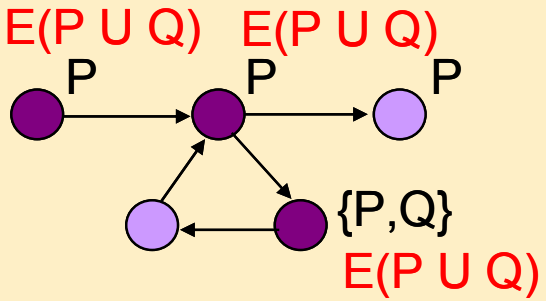


Ismétlés: A modellellenőrzés tanult technikái

- PLTL modellellenőrzés:
  - Tabló módszer: Kifejezések felbontása a modell mentén

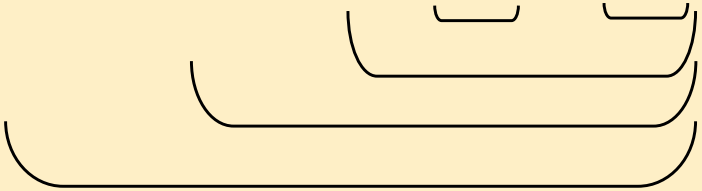


  - Automata alapú megközelítés (kiegészítő anyag)
- CTL modellellenőrzés:
  - Szemantika alapú módszer: Állapotok iteratív címkézése



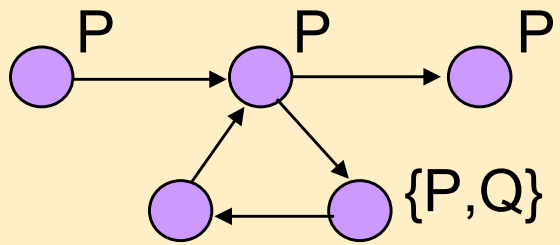
Ismétlés: CTL modellellenőrzés állapot címkézéssel

- Kifejezések felbontása azok struktúrája alapján, és „belülről kifelé” Sat(..) számítások:
  AF (P $\wedge$ E (Q U R))

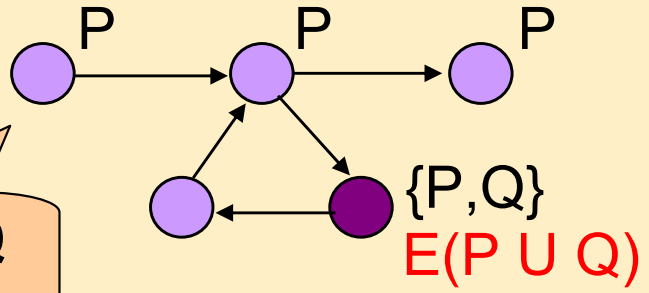


- Állapotok címkézése: Hol igaz egy adott kifejezés?
  - Kiindulás: KS címkézve van atomi kijelentésekkel
  - Továbblépés: Címkézés az összetettebb kifejezésekkel
    - Ha p illetve q címkék már vannak, akkor megadható, hol lehet $\neg$p, p$\wedge$q, EX p, AX p, E(p U q), A(p U q) címke
    - Inkrementális címkézési algoritmus az operátorok szemantikája alapján

Ismétlés: Az E(P U Q) címkézés iterációja

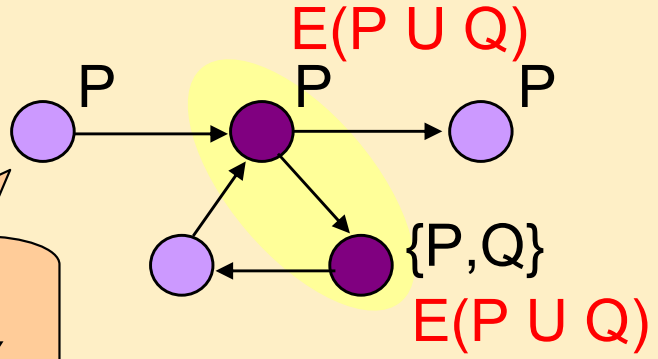


Kripke struktúra a kezdő címkézéssel



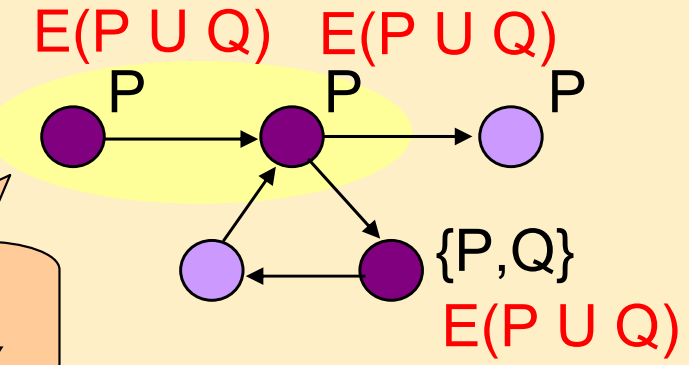
Első lépés: Q

- Kihasználható:
  E(P U Q) =
  Q $\vee$ (P $\wedge$ EX(P U Q))



Második lépés: P $\wedge$ EX

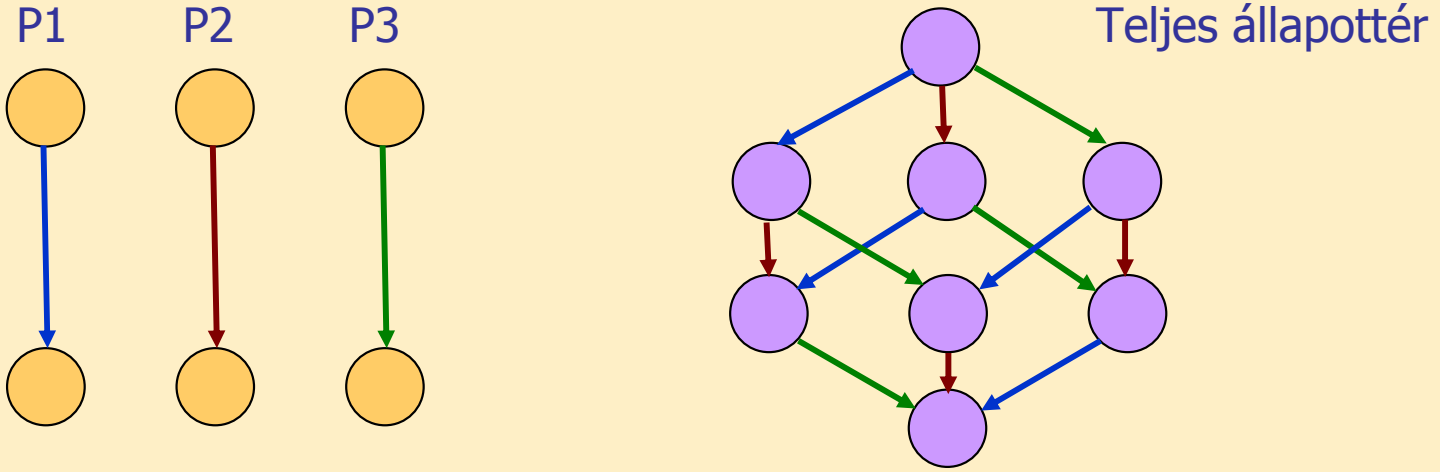
- Az iteráció addig tart, míg nő az állapothalmaz (fixpontot érünk el)



Harmadik lépés: P $\wedge$ EX

Problémák

- Nagy méretű állapottér bejárása szükséges
  - Az alapszintű formalizmusok esetén nincs hierarchia
  - Elosztott alkalmazások esetén nagy méretű állapottér adódik
    - Független állapotátmenetek végrehajtása sokféle (átlapolt) sorrendben



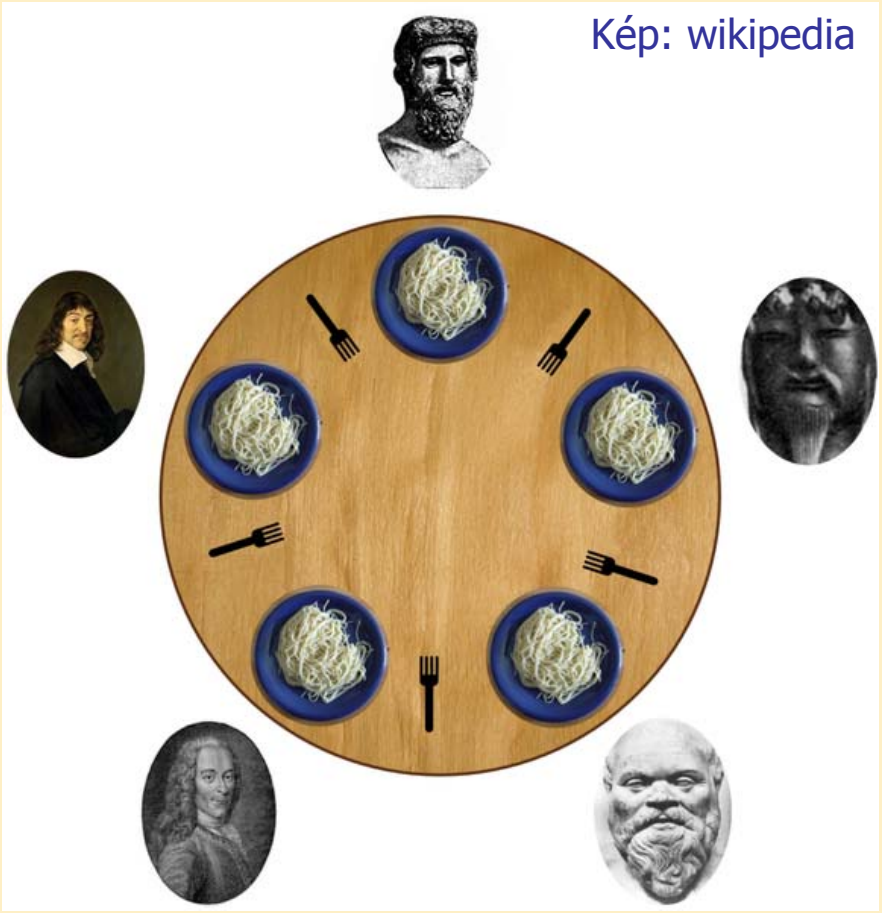
- Hogyan lehet nagy méretű modelleket ellenőrizni?
  - Ígéret: CTL modellellenőrzés: 10^{20}, egyes esetekben 10^{100} állapot
  - Milyen technológia tudja ezt biztosítani?

Egy jellegzetes példa: Étkező filozófusok

- Konkurens rendszer
  - Holtpont kialakulhat
  - Livelock kialakulhat
- Állapottér mérete gyorsan nő

Filozófusok száma	Állapottér mérete
16	$4{,}7 \cdot 10^{10}$
28	$4{,}8 \cdot 10^{18}$
…	…
200	$> 10^{40}$
1000	$> 10^{200}$
…	…

$2^{64} = 1{,}8 \cdot 10^{19}$



Okos (de nem feladat-specifikus) állapottér tárolással:
kb. 100 000 filozófus, azaz 10^{62900} állapottér is ellenőrizhető!

Előzetes áttekintés

- CTL modellellenőrzés: Szimbolikus technika

Szemantika alapú technika	Szimbolikus technika
Címkézett állapothalmazok	Karakterisztikus függvények (Boole logikai függvények): ROBDD reprezentáció
Műveletek állapothalmazokon	Hatékony műveletek ROBDD-ken

- Invariánsok modellellenőrzése: Korlátos modellellenőrzés
  - Logikai függvények igazságának keresése SAT technikával
  - Adott mélységig folytatható modellellenőrzés: Korlátos hosszúságú ellenpéldák keresése
    - Egy megtalált ellenpélda mindenképpen érvényes ellenpélda
    - Ha nincs ellenpélda, az nem végleges eredmény

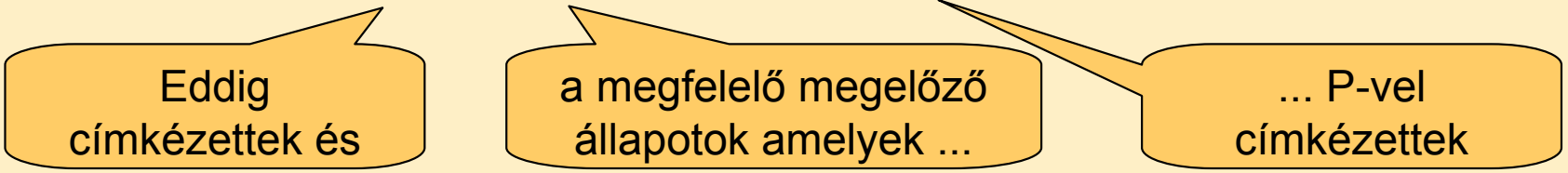
Szimbolikus modellellenőrzés

Iteráció halmazműveletekkel

- A címkézés bővítése halmazműveletekkel történik
  - Kezdőhalmaz: Rész-kifejezésekkel már címkézett állapotok
  - Címkézés bővítése:
    - E(p U q) esetén: „Legalább egy rákövetkező állapota már címkézett …”
    - A(p U q) esetén: „Minden rákövetkező állapota már címkézett …”
  - Ez alapján a megelőző állapotok valamelyikére tehető az új címke
- Megelőző állapotok jelölése már címkézett Z halmazhoz:

  $pre_E(Z)=\{s\in S \mid$ létezik olyan s', hogy $(s,s')\in R$ és $s'\in Z\}$

  $pre_A(Z)=\{s\in S \mid$ minden s'-re, ahol $(s,s')\in R$: $s'\in Z\}$
- Példa: E(P U Q) iterációja:
  - Kezdőhalmaz: $Z_0 = \{s \mid Q\in L(s)\}$
  - Iteráció: $Z_{i+1}= Z_i \cup (pre_E(Z_i) \cap \{s \mid P\in L(s)\})$

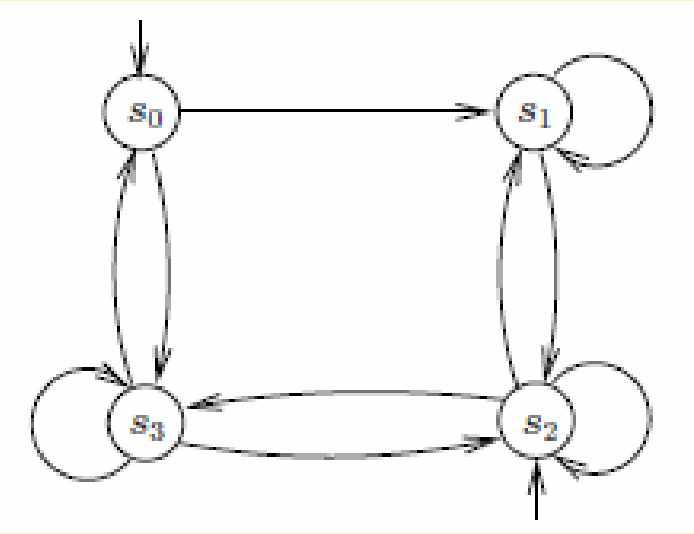


  - Iteráció vége: Ha $Z_{i+1}= Z_i$, azaz nem bővül a halmaz

Alapötlet

- Állapothalmazok tárolása: Az állapotok felsorolása helyett logikai függvények formájában
  - Állapotok „kódolása” bitvektorokkal
    - S állapottér kódolásához $n=\lceil \log_2|S| \rceil$ bit elég, azaz legyen $2^n \geq |S|$
  - Állapothalmazok „kódolása” n-változós logikai (Boole) függvényekkel
    - A logikai függvény akkor és csak akkor legyen igaz egy-egy bitvektor behelyettesítésére,
      ha a bitvektor által kódolt állapot az adott állapothalmazban van
- Karakterisztikus függvény: $C: \{0,1\}^n \rightarrow \{0,1\}$
  - A karakterisztikus függvényekkel fogunk dolgozni a halmazok helyett

Egy példa



Változók: x_1, x_2

Állapot	Bitvektor	Boole függvény
s_0	$\langle 0,0 \rangle$	$\neg x_1 \wedge \neg x_2$
s_1	$\langle 0,1 \rangle$	$\neg x_1 \wedge x_2$
s_2	$\langle 1,0 \rangle$	$x_1 \wedge \neg x_2$
s_3	$\langle 1,1 \rangle$	$x_1 \wedge x_2$

Az {s0, s2} állapothalmaz kar. függvénye:

$$f_I(x_1, x_2) = (\neg x_1 \wedge \neg x_2) \vee (x_1 \wedge \neg x_2)$$

Karakterisztikus függvények

- s állapotra: $C_s(x_1, x_2, \ldots, x_n)$

  Legyen az s „kódolása” $(u_1, u_2, \ldots, u_n)$ vektor, itt $u_i \in \{0,1\}$

  Ekkor $C_s(x_1, x_2, \ldots, x_n)$ csak az $(u_1, u_2, \ldots, u_n)$ esetén adjon 1 értéket

  $C_s(x_1, x_2, \ldots, x_n)$ konstruálása: $\wedge$ operátorral

  - x_i szerepel, ha $u_i=1$
  - $\neg x_i$ szerepel, ha $u_i=0$

  Példa: (0,1) kódolású s állapotra: $C_s(x_1, x_2) = \neg\, x_1 \wedge x_2$

- $Y \subseteq S$ állapothalmazra: $C_Y(x_1, x_2, \ldots, x_n)$

  $C_Y(x_1, x_2, \ldots, x_n)$ a.cs.a. legyen igaz $(u_1, u_2, \ldots, u_n)$ behelyettesítésre, ha $(u_1,u_2,\ldots, u_n) \in Y$

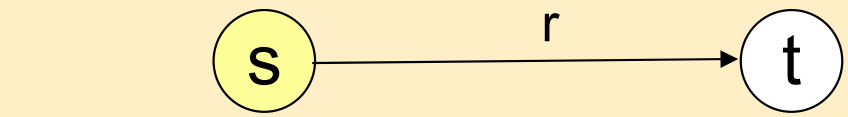
  $C_Y(x_1, x_2, \ldots, x_n)$ konstruálása: $C_Y(x_1, x_2, \ldots, x_n) = \bigvee_{s \in Y} C_s(x_1, x_2, \ldots, x_n)$

- Állapothalmazokra általában:

  $C_{Y \cup W} = C_Y \vee C_W$, $\quad C_{Y \cap W} = C_Y \wedge C_W$

Karakterisztikus függvények (folytatás)

- Állapotátmenetekre: C_r



$(u_1, u_2, \ldots, u_n)$ $(v_1, v_2, \ldots, v_n)$

r=(s,t) állapotátmenet, ahol $s=(u_1, u_2, \ldots, u_n)$ és $t=(v_1, v_2, \ldots, v_n)$

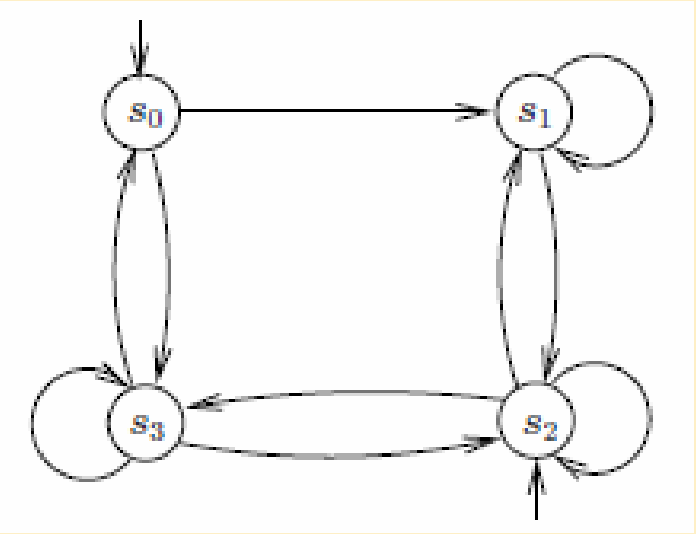
- Karakterisztikus függvény $C_r(x_1, x_2, \ldots, x_n, x'_1, x'_2, \ldots, x'_n)$ alakban
  - „Vesszős” változók jelzik a cél állapotot

C_r a.cs.a. legyen igaz, ha $x_i=u_i$ és $x_i'=v_i$ a behelyettesítés

C_r konstruálása:

$$C_r = C_s(x_1, x_2, \ldots, x_n) \wedge C_t(x'_1, x'_2, \ldots, x'_n)$$

Egy példa



Állapot	*Bitvektor*	*Boole függvény*
s_0	$\langle 0,0\rangle$	$\neg x_1 \wedge \neg x_2$
s_1	$\langle 0,1\rangle$	$\neg x_1 \wedge x_2$
s_2	$\langle 1,0\rangle$	$x_1 \wedge \neg x_2$
s_3	$\langle 1,1\rangle$	$x_1 \wedge x_2$

$$f_I(x_1, x_2) = (\neg x_1 \wedge \neg x_2) \vee (x_1 \wedge \neg x_2)$$

Változók: x1, x2

$f_\rightarrow$	$\langle 0,0\rangle$	$\langle 0,1\rangle$	$\langle 1,0\rangle$	$\langle 1,1\rangle$
$\langle 0,0\rangle$	0	1	0	1
$\langle 0,1\rangle$	0	1	1	0
$\langle 1,0\rangle$	0	1	1	1
$\langle 1,1\rangle$	1	0	1	1

$$\begin{aligned} f_\rightarrow(x_1, x_2, x_1', x_2') = \quad & (\neg x_1 \wedge \neg x_2 \wedge \neg x_1' \wedge x_2') \\ \vee \quad & (\neg x_1 \wedge \neg x_2 \wedge x_1' \wedge x_2') \\ \vee \quad & (\neg x_1 \wedge x_2 \wedge x_1' \wedge \neg x_2') \\ \vee \quad & \ldots \\ \vee \quad & (x_1 \wedge x_2 \wedge x_1' \wedge x_2') \end{aligned}$$

Karakterisztikus függvények (folytatás)

- $pre_E(z)$ képzése: $pre_E(z)=\{s \mid \exists t: (s,t)\in R \text{ és } t\in z\}$

  z reprezentációja: C_z

  R reprezentációja: $C_R=\vee_{r\in R}C_r$

  $pre_E(z)$: kikeresni a z-beli állapotokra az előzőeket

$$C_{\mathrm{pre_E}(z)} = \exists_{x'_1,x'_2,\ldots,x'_n} C_R \wedge C'_z$$

ahol $\exists_x C = C[1/x] \vee C[0/x]$ „egzisztenciális absztrakció”

- Modellellenőrzés állapothalmaz műveletekkel: visszavezetve logikai függvényeken végzett műveletekre!
  - Halmazok uniója: Függvények $\vee$ kapcsolata
  - Halmazok metszete: Függvények $\wedge$ kapcsolata
  - $pre_E(z)$ képzése: Összetett művelet (egzisztenciális absztrakció)

Logikai függvények reprezentációja

Kanonikus forma: ROBDD

Reduced, Ordered Binary Decision Diagram

Redukált, rendezett, bináris döntési diagram

Lépések:

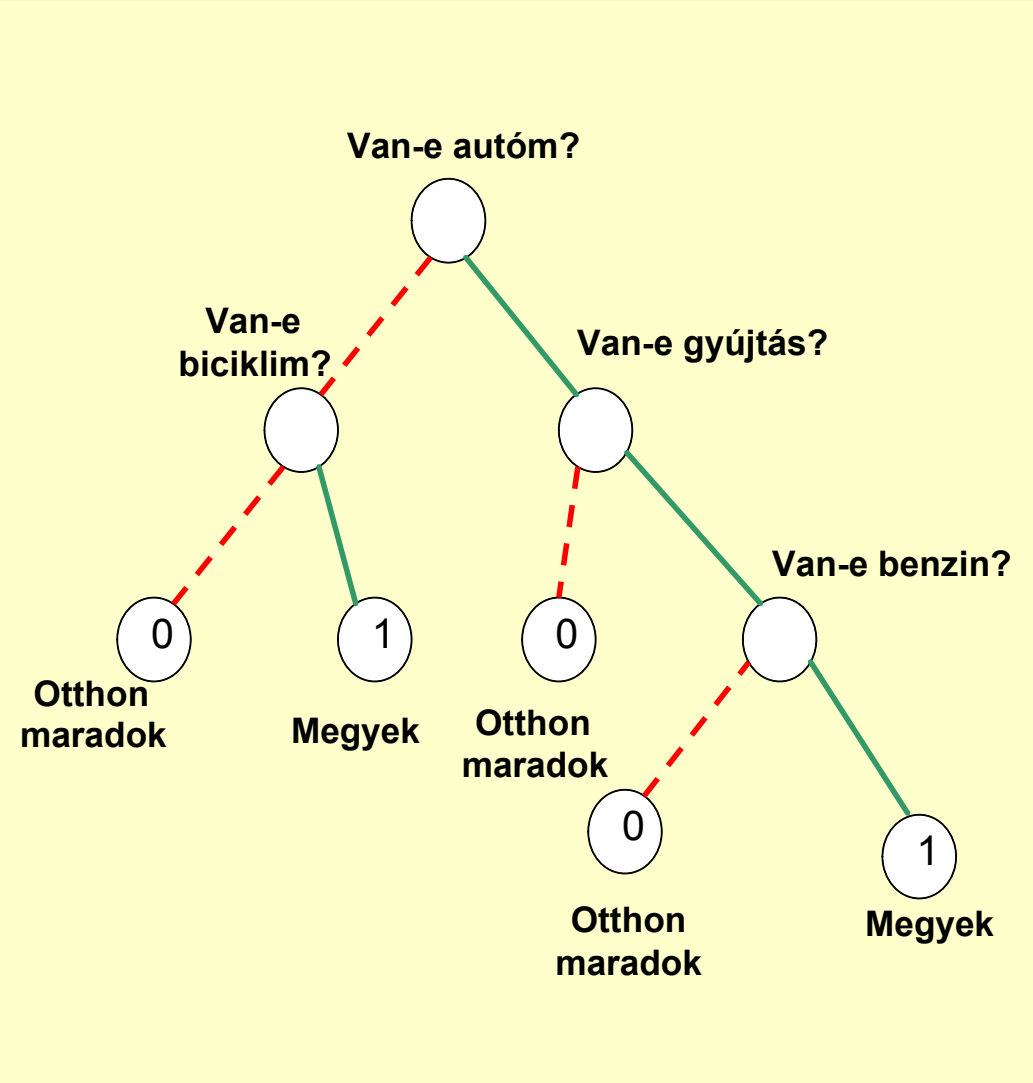
- Bináris döntési fa: bináris döntések ábrázolása
- BDD: azonos részfák összevonva
- OBDD: minden ágon azonos változósorrendezés
- ROBDD: szükségtelen csomópontok redukálása
  - Ha mindkét ág ugyanahhoz a csomóponthoz vezet

Az ROBDD-ről részletesen

A bináris döntési fa

- Egy cél elérését több döntés befolyásolja
- Csomópontokban bináris döntések
  - Igen/Nem ágak
- Eredmény: Válasz a cél elérésére egy döntéssorozat után:
  - Igen / nem

Többértékű kiterjesztés is létezik



Boole függvények döntési fa alakban

- Mindegyik változó behelyettesítése egy-egy döntés
- Az if-then-else szerkezet:

$$x \rightarrow f_1, f_0 = (x \wedge f_1) \vee (\neg x \wedge f_0)$$

  - f_1 értéket veszi fel ha x igaz (true, 1)
  - f_0 értéket veszi fel ha x nem igaz (false, 0)
  - x szokásos neve tesztváltozó, értékének vizsgálata a teszt
- Boole függvények Shannon felbontása:

$$f = x \rightarrow f[1/x], f[0/x]$$

legyen $f_x = f[1/x]$; $f_{\underline{x}} = f[0/x]$

$$\Big\} \quad f = x \rightarrow f_x, f_{\underline{x}}$$

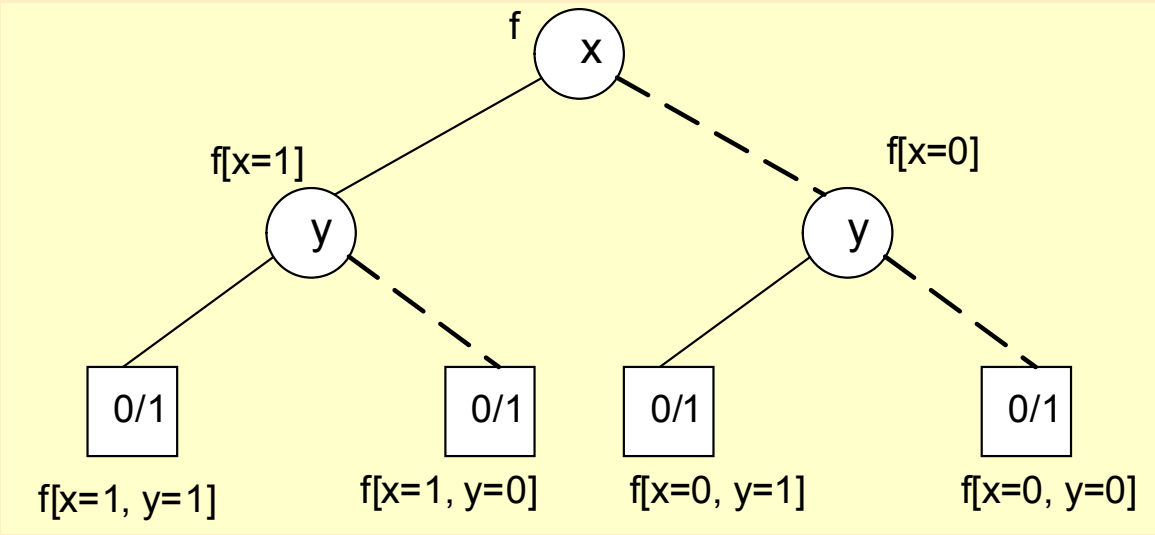
  - Tehát a függvényt az if-then-else alapján szétbonthatjuk
  - A then-else ágakban ezzel egy változóját eltüntettük, redukáltuk
  - Ciklikusan ismételjük, amíg van változó

Döntési fák típusai

Példa:

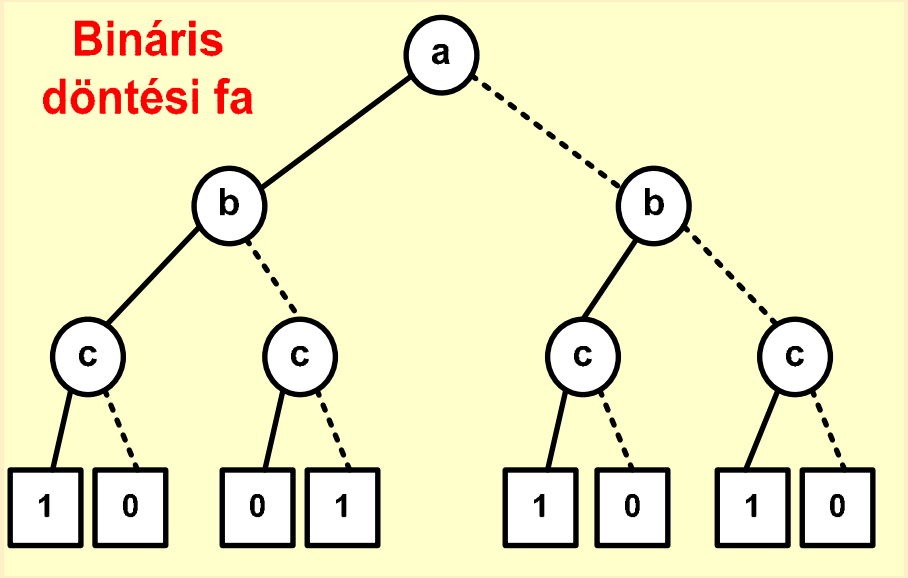
f(x,y)

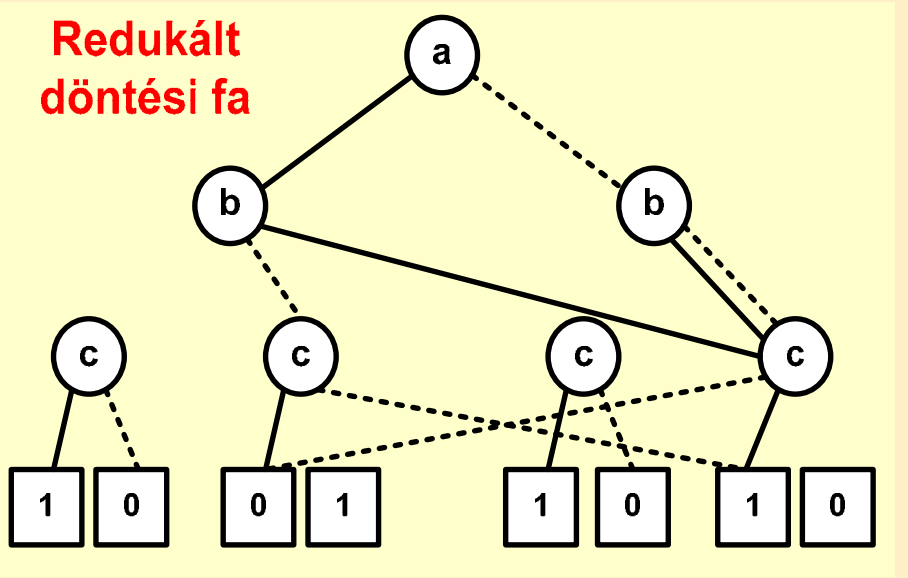
Ekkor a fán a négyzetekben határozhatók meg f(x,y) lehetséges értékei

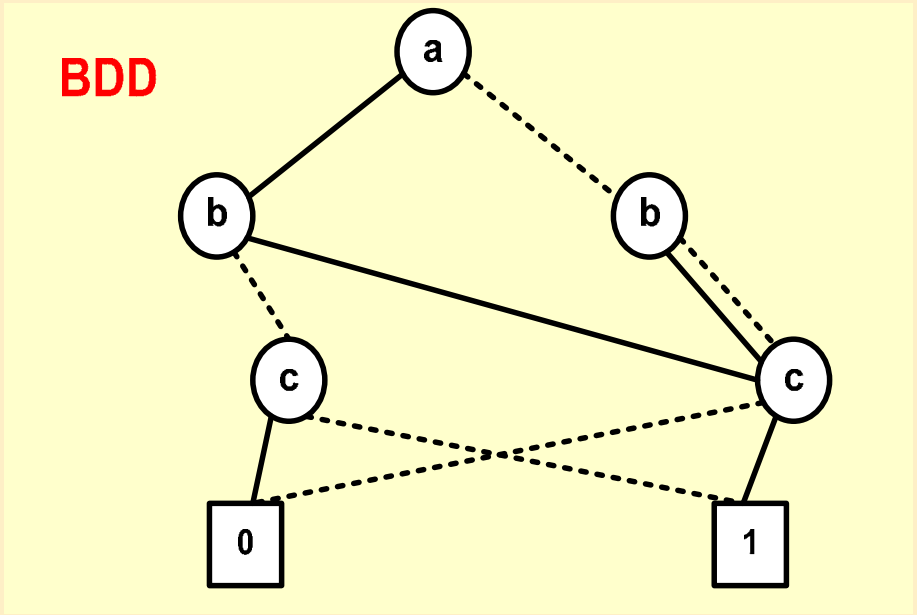


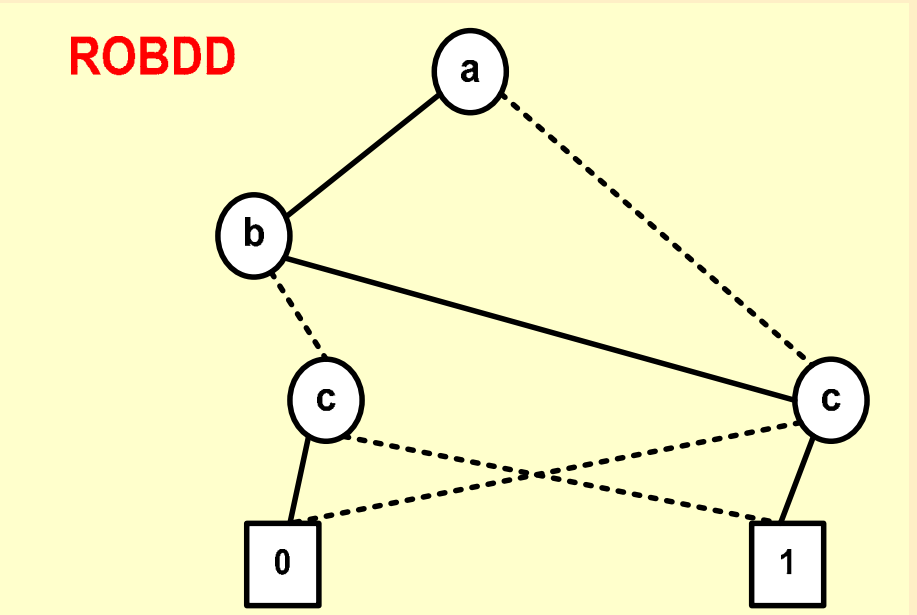
- Bináris döntési diagramot (BDD) kapunk, ha az azonos részfákat összevonjuk
- Rendezett bináris döntési diagramot (OBDD) kapunk, ha a felbontás során minden ágon azonos sorrendben vesszük fel a teszt változókat
- Redukált rendezett bináris döntési diagramot (ROBDD) kapunk, ha a szükségtelen csomópontokat töröljük

Példa: Egy bináris döntési fa átalakítása







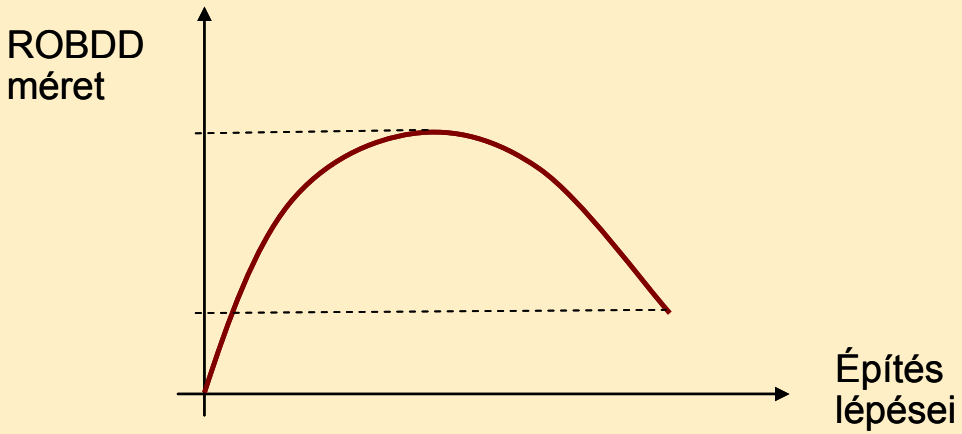


ROBDD tulajdonságok

- Két azonos változósorrendezésű ROBDD izomorf
- Irányított, aciklikus gráf, egy gyökérrel és két levéllel
  - A két levél értéke 1 vagy 0 (true vagy false)
  - Minden csomópontban egy-egy teszt változó van
- Minden csomópontból két él indul ki
  - Egyik a 0 behelyettesítésre (jelölés: szaggatott él)
  - Másik az 1 behelyettesítésre (jelölés: folytonos él)
- Minden útvonalon a teszt változók azonos sorrendben
- Az izomorf részfák egyetlen részfává összevonva
- Azon csomópontok, ahonnan kimenő él ugyanahhoz a csomóponthoz vezet, redukálva vannak

ROBDD változó sorrend

- ROBDD mérete
  - Egyes függvények (pl. páros paritás) esetén nagyon kompakt
  - Más függvények esetén (pl. szorzás) csak exponenciális méret
- A méret szempontjából nagyon fontos a változók sorrendjének megválasztása!
  - Más sorrend nagyságrendbeli különbséget is jelenthet
  - Optimális sorrend megtalálása NP-teljes probléma ($\rightarrow$ heurisztika)
- Tárigény: Ha folyamatosan építjük a ROBDD-t, akkor az építés közben ideiglenes csomópontokat kell tárolnunk, amiket le lehet redukálni

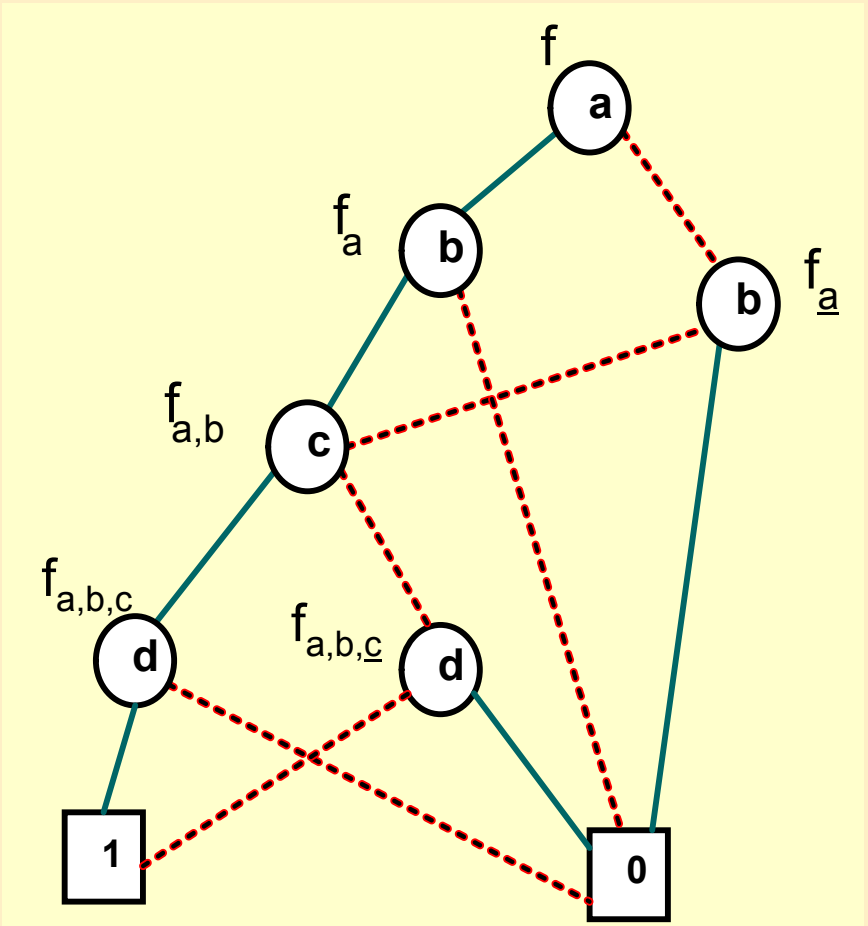


Példa: ROBDD kézi előállítása

Legyen az

$f = (a \Leftrightarrow b) \wedge (c \Leftrightarrow d)$

Sorrend legyen: a, b, c, d

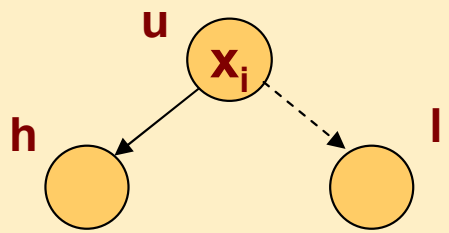


- $f = a \rightarrow f_a, f_{\underline{a}}$
  $f_a = (1 \Leftrightarrow b) \wedge (c \Leftrightarrow d)$, $f_{\underline{a}} = (0 \Leftrightarrow b) \wedge (c \Leftrightarrow d)$
- $f_a = b \rightarrow f_{a,b}, f_{a,\underline{b}}$
  $f_{a,b} = (1 \Leftrightarrow 1) \wedge (c \Leftrightarrow d) = (c \Leftrightarrow d)$
  $f_{a,\underline{b}} = (1 \Leftrightarrow 0) \wedge (c \Leftrightarrow d) = 0$
- $f_{\underline{a}} = b \rightarrow f_{\underline{a},b}, f_{\underline{a},\underline{b}}$
  $f_{\underline{a},b} = (0 \Leftrightarrow 1) \wedge (c \Leftrightarrow d) = 0$
  $f_{\underline{a},\underline{b}} = (0 \Leftrightarrow 0) \wedge (c \Leftrightarrow d) = (c \Leftrightarrow d)$
- $f_{a,b} = c \rightarrow f_{a,b,c}, f_{a,b,\underline{c}}$
  $f_{a,b,c} = (1 \Leftrightarrow d)$, $f_{a,b,\underline{c}} = (0 \Leftrightarrow d)$
- $f_{a,b,c} = d \rightarrow f_{a,b,c,d}, f_{a,b,c,\underline{d}}$
  $f_{a,b,c,d} = (1 \Leftrightarrow 1) = 1$,
  $f_{a,b,c,\underline{d}} = (1 \Leftrightarrow 0) = 0$
- $f_{a,b,\underline{c}} = d \rightarrow f_{a,b,\underline{c},d}, f_{a,b,\underline{c},\underline{d}}$
  $f_{a,b,\underline{c},d} = (0 \Leftrightarrow 1) = 0$, , $f_{a,b,\underline{c},\underline{d}} = (0 \Leftrightarrow 0) = 1$

$f_{a,b}$ és $f_{a,\underline{b}}$ izomorf!

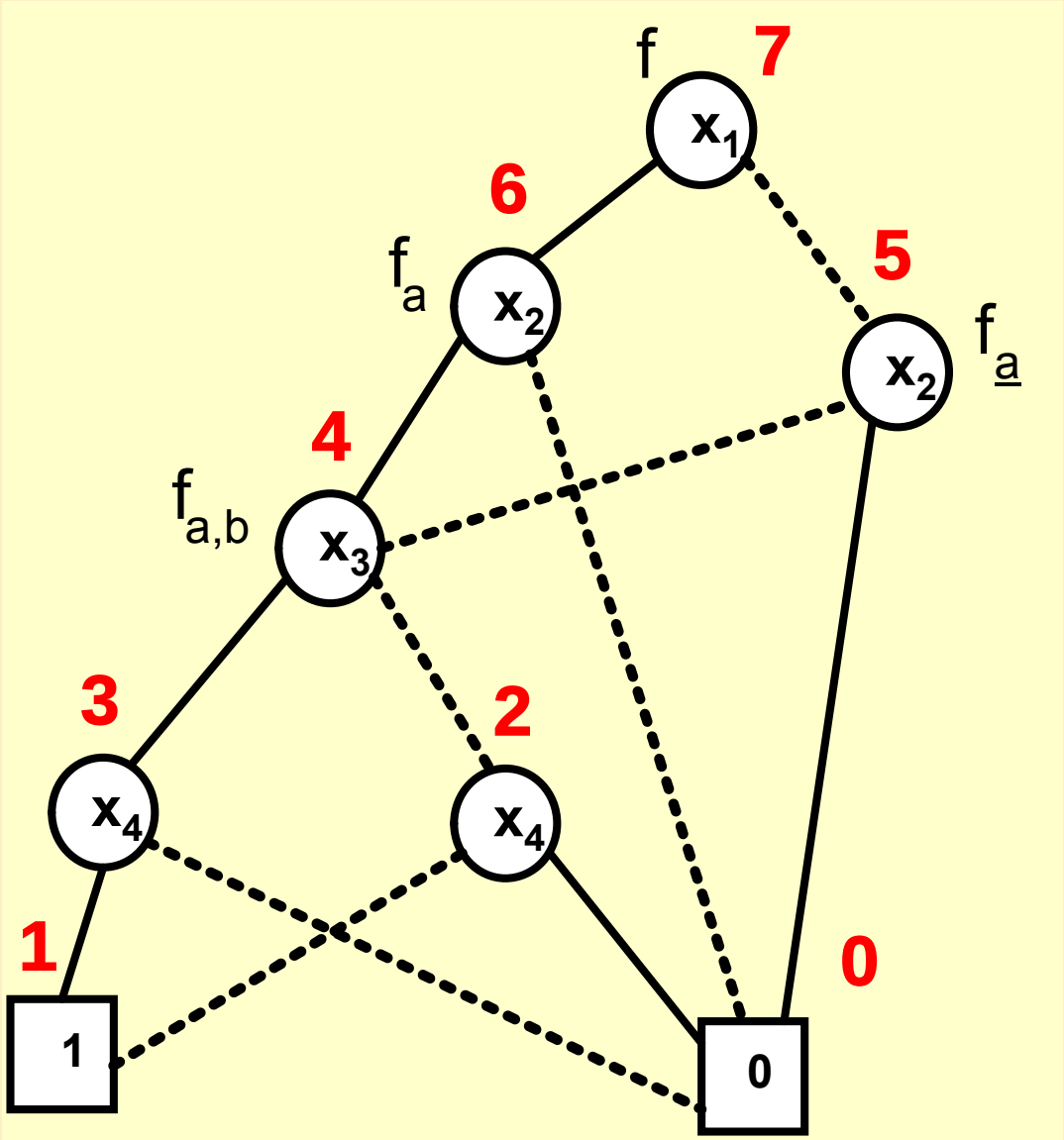
ROBDD gépi tárolása

- A ROBDD csomópontjait indexekkel azonosítjuk
- A ROBDD-t egy
  T: u $\rightarrow$ (i,l,h)
  táblázatban tároljuk:
  - u: csomópont indexe
  - i: a változó indexe (x_i, i=1…n)
  - l: a 0 behelyettesítési ágon elérhető csomópont indexe
  - h: az 1 behelyettesítési ágon elérhető csomópont indexe



u	i	l	h
0 1			
2	4	1	0
3	4	0	1
4	3	2	3
5	2	4	0
6	2	0	4
7	1	5	6

ROBDD gépi tárolása



u	i	l	h
0			
1			
2	4	1	0
3	4	0	1
4	3	2	3
5	2	4	0
6	2	0	4
7	1	5	6

ROBDD gépi előállítása 1.

- Értelmezett műveletek:
  - init(T)
    - T kezdeti állapotát állítja be
    - csak a 0 és 1 csomópontok vannak a táblázatban
  - add(T,i,l,h):u
    - egy új csomópontot készít T-ben az adott paraméterekkel
    - ennek u indexét adja vissza
  - var(T,u):i
    - visszaadja T-ből az u csomópont változójának i indexét
  - low(T,u):l és high(T,u):h
    - T-ben az u két behelyettesítési ágán levő csomópont l illetve h indexét adja vissza

ROBDD gépi előállítása 2.

- ROBDD csomópontjainak visszakereséséhez egy H: (i,l,h) $\rightarrow$ u táblázatot is nyilvántartunk
- Műveletei:
  - init(H)
    - egy üres H táblázatot állít elő
  - member(H,i,l,h):t
    - ellenőrzi, hogy az (i,l,h) hármas szerepel-e H-ban; t Boole
  - lookup(H,i,l,h):u
    - kikeresi az (i,l,h) hármast a H táblázatban
    - visszaadja a hozzá tartozó u csomópont indexét
  - insert(H,i,l,h,u)
    - beilleszt egy új sort a táblázatba

ROBDD gépi előállítása 3.

Csomópont építése: Mk(i,l,h)

- Itt i index, l és h ágak
- Ha l=h, azaz azonos csomópontba vezetne a két él
  - akkor nem kell csomópontot létrehozni
  - bármelyik ágat vissza lehet adni
- Ha H-ban már van egy (i,l,h) hármas
  - akkor sem kell újat létrehozni

  $\Rightarrow$ létezik izomorf részfa, ennek indexét kell visszaadni
- Ha nincsen H-ban ilyen (i,l,h)
  - akkor létre kell hozni, és visszaadni indexét

```
Mk(i,l,h){
   if l=h then
        return l;
   else if member(H,i,l,h) then
        return lookup(H,i,l,h);
   else {
        u=add(T,i,l,h);
        insert(H,i,l,h,u);
        return u;
   }
}
```

ROBDD gépi előállítása 4.

ROBDD építése: Build(f) és a Build'(t,i) segédfüggvény

```
Build(f) {
   init(T); init(H);
   return Build'(f,1);
}

Build'(t,i){
   if i>n then
         if t==false then return 0 else return 1
   else {v0 = Build'(t[0/x_i],i+1);
         v1 = Build'(t[1/x_i],i+1);
         return Mk(i,v0,v1)}
}
```

Rekurzívan végigmegy majd a változókon

Terminális csomóponthoz értünk

Rekurzív építés; Mk() ellenőrzi az izomorf részfákat

Műveletek ROBDD-ken

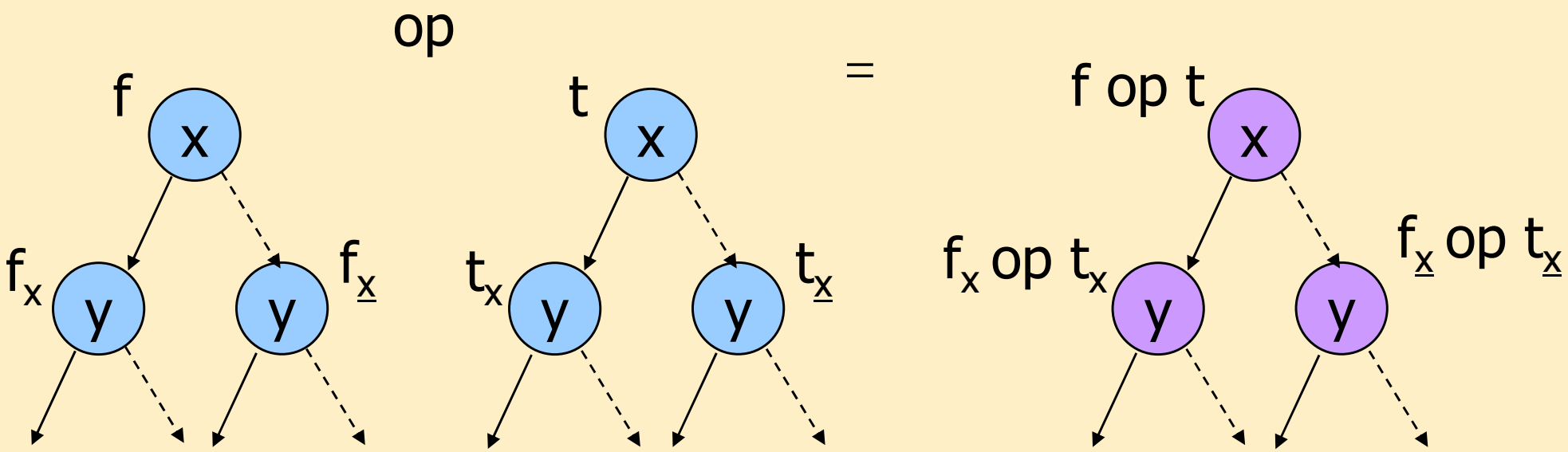
- Boole operátorokat közvetlenül ROBDD-ken hajtjuk végre
  - A két függvény változói azonosak legyenek, azonos sorrendben
- Alap azonosság f, t függvényekre (itt op Boole operátor):
  1. $f \text{ op } t = (x \rightarrow f_x, f_{\underline{x}}) \text{ op } (x \rightarrow t_x, t_{\underline{x}}) = x \rightarrow (f_x \text{ op } t_x), (f_{\underline{x}} \text{ op } t_{\underline{x}})$

- További szabályok (redukálás miatt hiányzó változó):
  2. $f \text{ op } t = (x \rightarrow f_x, f_{\underline{x}}) \text{ op } t = x \rightarrow (f_x \text{ op } t), (f_{\underline{x}} \text{ op } t)$
  3. $f \text{ op } t = f \text{ op } (x \rightarrow t_x, t_{\underline{x}}) = x \rightarrow (f \text{ op } t_x), (f \text{ op } t_{\underline{x}})$
- Ezen szabályok alapján definiálható App(op,i,j) rekurzívan
  - ahol i, j: a művelet operandusainak ROBDD-jében a csomópontok
- Hátrány: lassú
  - worst-case 2^n exponenciális

Műveletek illusztrálása

$f\ op\ t = (x \rightarrow f_x, \underline{f_x})\ op\ (x \rightarrow t_x, \underline{t_x}) = x \rightarrow (f_x\ op\ t_x), (\underline{f_x}\ op\ \underline{t_x})$

rekurzívan felépíthető



Gyorsított műveletvégzés

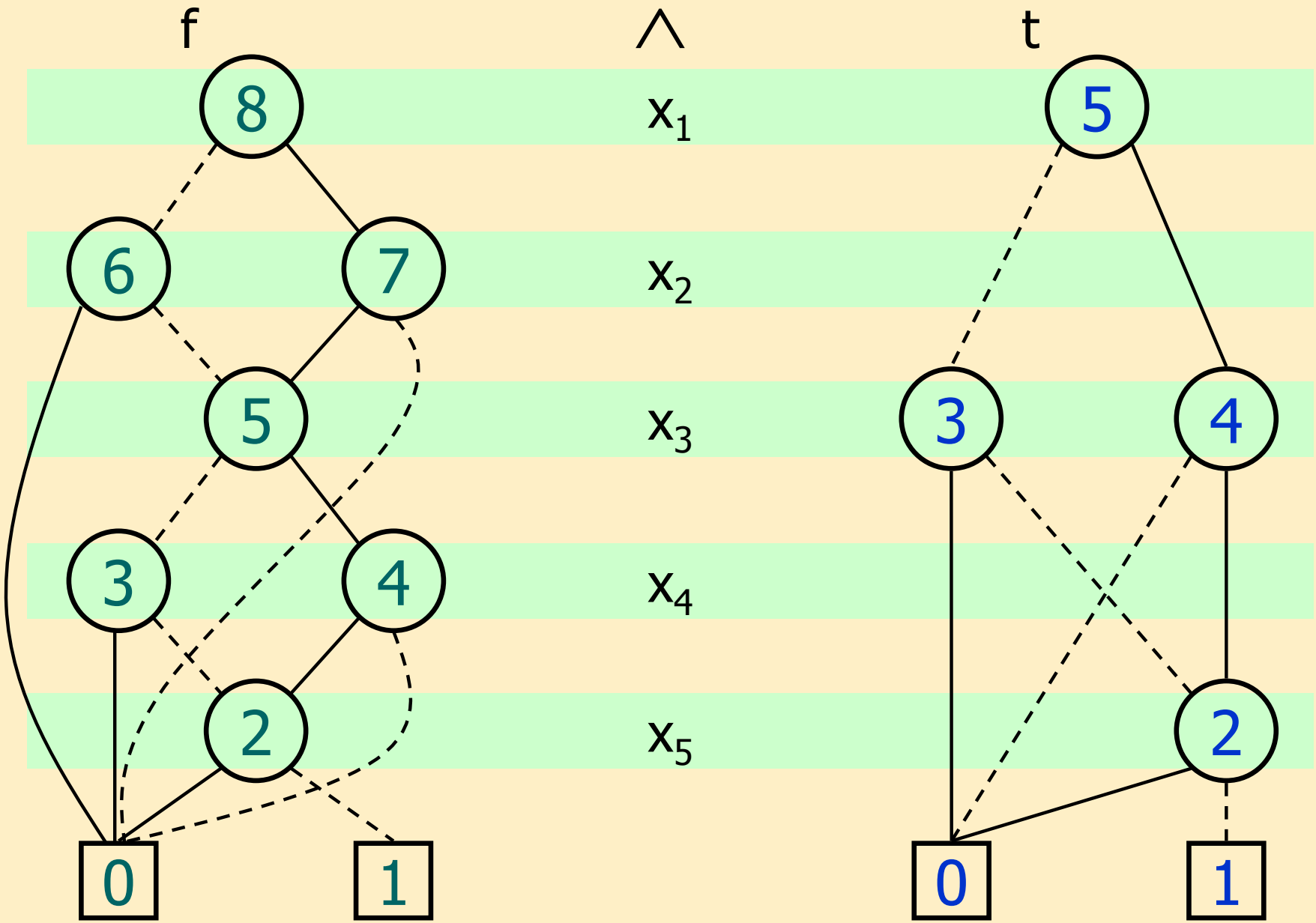
- Legyen G(op,i,j) egy gyorsítótáblázat, amely App(op,i,j) eredményét tartalmazza (csomópont)
- Az algoritmus négy esete:
  - Mindkét csomópont terminális: ekkor egy új terminális hozható létre az operátorral végzett eredmény alapján
  - Ha a csomópontokhoz tartozó változó indexe azonos, akkor a 0 és az 1 behelyettesítésű ágak párosíthatóak az App(op,i,j) alkalmazásából, az 1. azonosság szerint
  - Ha az egyik csomóponthoz tartozó változó indexe nagyobb, akkor ezt párosítjuk a kisebb változó-indexű csomópont 0 és 1 ágaival a 2. vagy 3. azonosság szerint

A műveletvégzés pszeudo-kódja

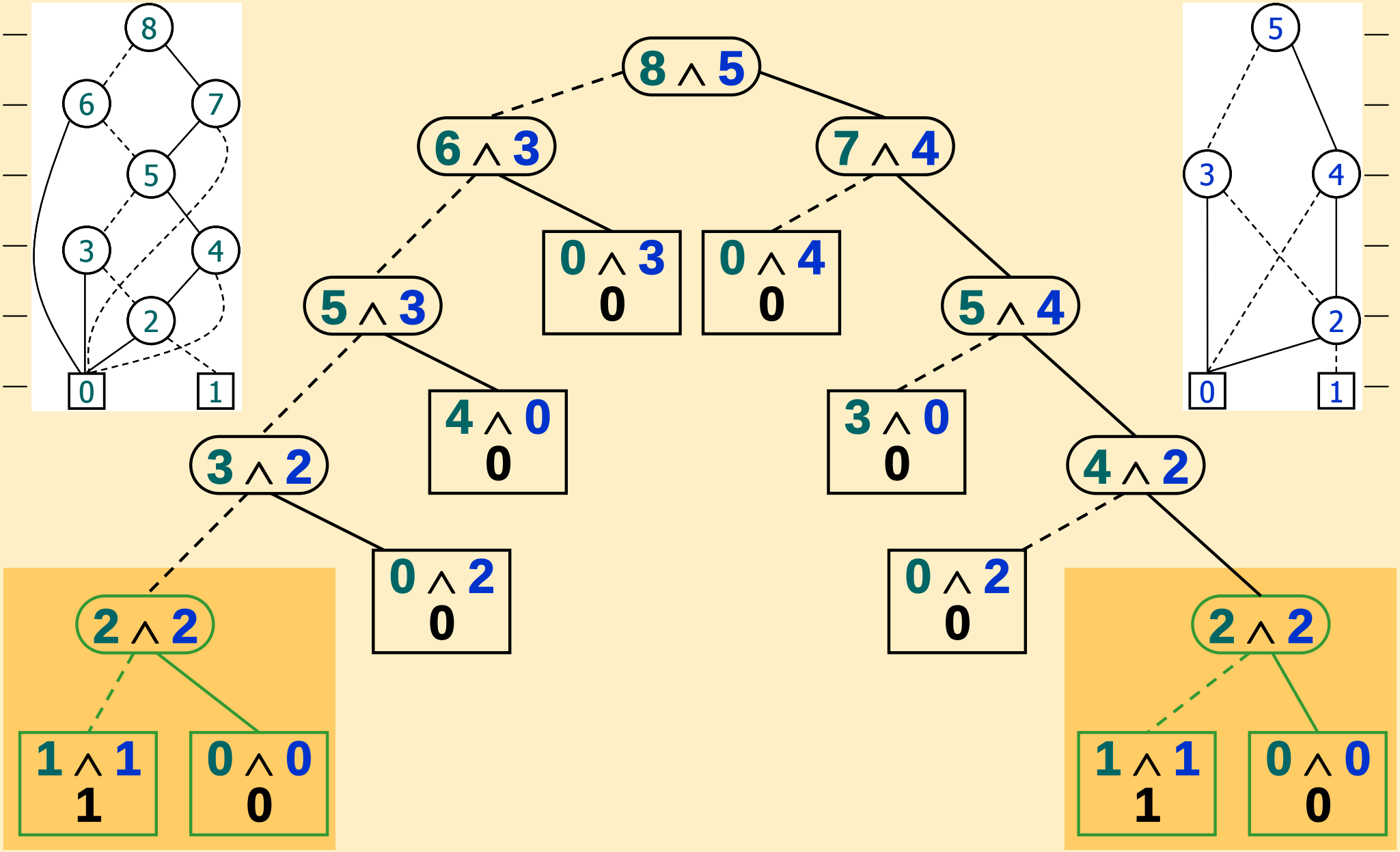
```
Apply(op,f,t){
  init(G);
  return App(op,f,t);
}

App(op,u1,u2) {
   if (G(op,u1,u2) <> empty) then return G(op,u1,u2);
   else if (u1 in {0,1} and u2 in {0,1}) then u = op(u1,u2);
   else if (var(u1) = var(u2)) then
        u=Mk(var(u1), App(op,low(u1),low(u2)),
                      App(op,high(u1),high(u2)));
   else if (var(u1) < var(u2)) then
        u=Mk(var(u1), App(op,low(u1),u2),App(op,high(u1),u2));
   else (* if (var(u1) > var(u2)) then *)
        u=Mk(var(u2), App(op,u1,low(u2)),App(op,u1,high(u2)));
  G(op,u1,u2)=u;
  return u;
}
```

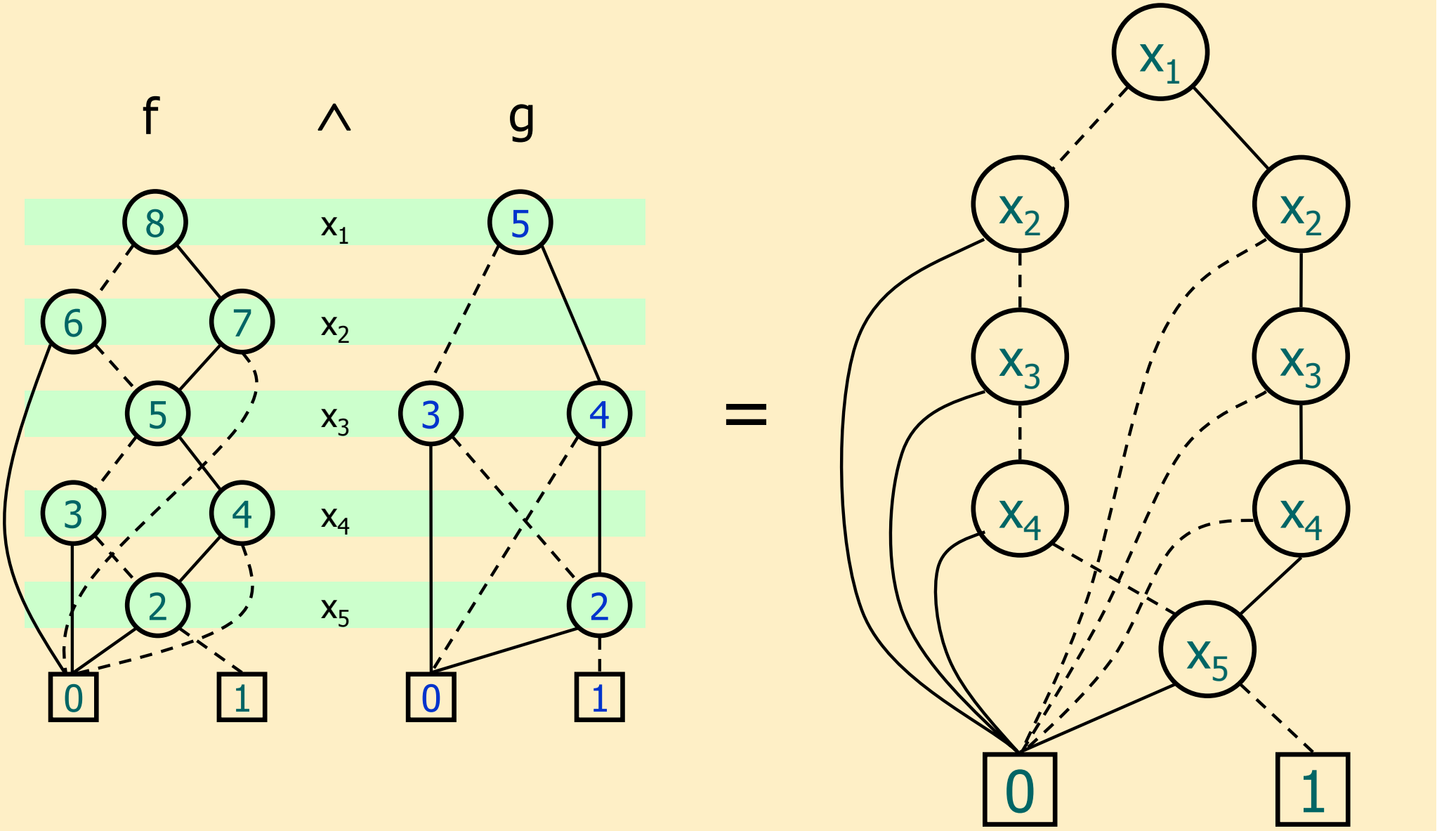
Példa: Művelet elvégzése (f $\wedge$ t)



Példa: Művelet elvégzése (f ∧ t)



Példa: Művelet eredménye (f ∧ g)



Behelyettesítés ROBDD alakjának előállítása

Változók konstans behelyettesítése (ld. $pre_E(z)$ is):

Itt az u ROBDD-ben az x_j változó értéke b legyen

```
Restrict(u,j,b) {
   return Res(u,j,b);
}

Res(u,j,b) {
  if var(u) > j then return u;
  else if var(u) < j then
    return Mk(var(u),
              Res(low(u),j,b),
              Res(high(u),j,b));
  else
    if b=0 then
       return Res(low(u),j,b)
    else
       return Res(high(u),j,b);
}
```

Ha lejjebb vagyok a behelyettesítendő változónál, marad az eredeti részfa

Ha feljebb vagyok a behelyettesítendő változónál, rekurzív építés kell

Ha ott vagyok a behelyettesítendő változónál, behelyettesítés kell

Összefoglalás: Modellellenőrzés ROBDD-vel

- A modellellenőrzés megvalósítása:
  - Modellellenőrzés algoritmusa: Műveletek állapothalmazokon (címkézés)
  - Szimbolikus technika: Állapothalmazok helyett logikai függvények
  - Hatékony megvalósítás: Logikai függvények ROBDD alakban kezelve
- Előnyök
  - A ROBDD kanonikus alak (függvények ekvivalenciája jól vizsgálható)
  - Algoritmusok hatékonyan gyorsíthatók
  - Tárigény csökken (változósorrend megválasztásától függ!)

Étkező filozófusok problémája:

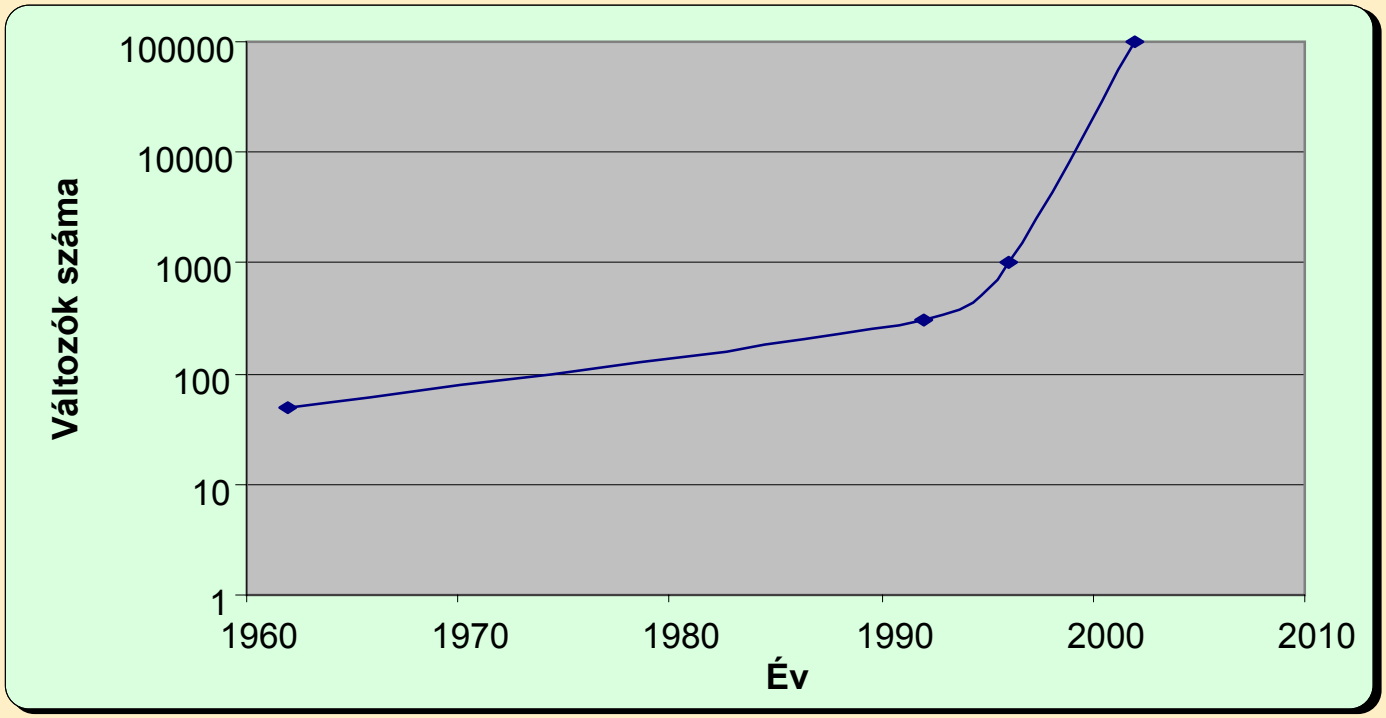
Filozófusok száma	Állapottér mérete	ROBDD csomópontok
16	$4{,}7 \cdot 10^{10}$	747
28	$4{,}8 \cdot 10^{18}$	1347

10^{18} méretű állapottér tárolása helyett kb. 21kB elég!

Korlátos modellellenőrzés (Bounded Model Checking)

Felhasználható eredmény: SAT megoldók

- SAT megoldó:
  - Adott logikai függvényhez olyan helyettesítési értékeket keres, amelyekkel a függvény értéke igaz
- NP-teljes probléma, de hatékony algoritmusok
  - zChaff, MiniSAT

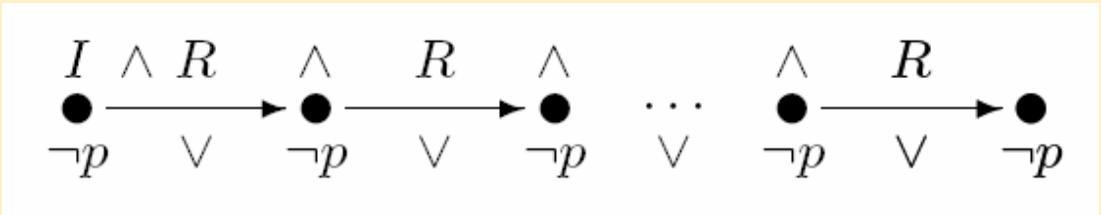


Célkitűzés

- A probléma alkalmas leképzése logikai függvényre
  - Modell és temporális logikai követelmény együttesen
  - Tipikusan invariáns követelmény
    - Minden állapotra előírt tulajdonság
- SAT megoldó használata modellellenőrzésre
  - Ha a követelmény teljesül, a SAT megoldó nem talál helyettesítési értéket a függvényhez
  - Ha a követelmény nem teljesül, a SAT megoldó által adott helyettesítési érték lesz egy ellenpélda
    - Az ellenpélda használható a hibakereséshez
    - Invariáns tulajdonságok esetén jól használható módszer

Informális bevezetés

- Modell leképzése logikai függvénybe:
  - R(s,s') állapotátmeneti reláció használata:
    - A modell „kihajtogatása” az állapotátmeneti reláció mentén
    - Új változók felvétele az egymást követő állapotokra
  - Kezdőállapotokra vonatkozó predikátum I(s)
- Kritérium (invariáns) leképzése logikai függvénybe:
  - Állapotok címkézése:
    p(s) állapot-predikátum (az állapotváltozókkal kifejezve)
- A SAT probléma: Konjunkció
  - Kezdőállapotra I(s) predikátum
  - Széthajtogatott átmenetek az R(s,s') reláció alkalmazásával
  - p(s) követelmény mint állapot-predikátum



Korlátos modellellenőrzés

- Az állapottér mérete kezelhetetlenül nagy lehet
  - Állapottér robbanás:
    Konkurens, lazán csatolt rendszerek
- Az útvonalak hosszát korlátozva végezzük az ellenőrzést
  - Állapottér bejárás csak adott útvonalhossz korlátig (eddig hajtogatjuk ki a modellt)
  - Egyes esetekben van „átmérője” az állapottérnek: a leghosszabb útvonal, ami bejárható
- A korlát megválasztása:
  - Intuíció a probléma méretéről
  - Worst case végrehajtási idő alapján

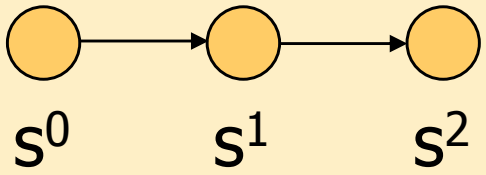
Jelölések

- M=(S,R,L) Kripke-struktúra
- Logikai függvények:
  - $C_s(s)$ állapotok „kódolása” n hosszú bitvektorokkal
  - $C_R(s,s')$ állapotátmenetek 2n változós karakterisztikus függvénye
  - L címkézés: Állapot-predikátumok, pl. P(s), Q(s)
  - I(s) kezdőállapotok predikátuma (több is lehet)
  - Útvonal: k hosszú útvonal (k+1)n változóval

$$path(s^0, s^1, ..., s^k) = \bigwedge_{0 \le i < k} C_R(s^i, s^{i+1})$$

Vesszős változók helyett felső index

Útvonalhoz tartozó logikai függvény



- Állapotok karakterisztikus függvényei:
  $C_s(x,y)$, itt x és y a bináris állapotváltozók („bitek”)
- Egy (s^0,s^1) átmenet karakterisztikus függvénye:
  $C_r(s^0,s^1)$ itt az alakja $C_r(x^0,y^0,x^1,y^1)$
  csak az s^0, s^1 -hez tartozó behelyettesítési értékekre igaz
- Egy (s^0,s^1,s^2) útvonal karakterisztikus függvénye:
  $C_p(s^0, s^1, s^2)$ itt az alakja $C_p(x^0,y^0,x^1,y^1,x^2,y^2)$
- Összes 2 hosszú útvonal karakterisztikus függvénye:
  $path(x^0,y^0,x^1,y^1,x^2,y^2)=C_R(s^0,s^1) \wedge C_R(s^1,s^2)$
- Az s^0 kezdőállapotból induló 2 hosszú útvonalak:

$$I(s^0) \wedge C_R(s^0,s^1) \wedge C_R(s^1,s^2)$$

A probléma formalizálása

- Bizonyítandó P(s) invariáns: Minden útvonal, ami a kezdőállapotból indul, olyan állapotokba jut, ahol P(s) igaz

$$\forall i: \forall s^0, s^1, ..., s^i : (I(s^0) \wedge path(s^0, s^1, ..., s^i) \Rightarrow P(s^i))$$

- Ha P(s) nem igaz valahol, akkor lesz olyan i, amire a következő függvény igaz értéket vesz fel:

$$I(s^0) \wedge path(s^0, s^1, ..., s^i) \wedge \neg P(s^i)$$

- A fv. igaz értékéhez tartozó behelyettesítést adja a SAT megoldó!
  - Azaz az (s^0,s^1,...,s^i) útvonalat meghatározó (i+1)n változó értéket

- Első ötlet: i=0,1,2,...-ra rendre megvizsgálni, hogy i hosszú útvonalon igaz lehet-e a következő függvény (ha igaz, akkor van ellenpélda):

$$\forall s^0, s^1, ..., s^i : (I(s^0) \wedge path(s^0, s^1, ..., s^i) \wedge \neg P(s^i))$$

Az algoritmus elemei

- Iteráció: i=0,1,2,... az útvonalak hosszára
- Ciklusmentes utakat kell vizsgálnunk: $lfpath$

$$lfpath(s^0, s^1, ..., s^k) = path(s^0, s^1, ..., s^k) \wedge \bigwedge_{0 \le i < j \le k} s^i \neq s^j$$

- Megállási feltétel az iteráció során:
  - Nincs i hosszú ciklusmentes út kezdőállapotból, azaz nem lehet igaz

$$I(s^0) \wedge lfpath(s^0, s^1, ..., s^i)$$

  - „Rossz” állapothoz (ahol P(s) nem igaz) nem is vezethet i hosszú ciklusmentes út (kezdőállapottól függetlenül), azaz nem lehet igaz

$$lfpath(s^0, s^1, ..., s^i) \wedge \neg P(s^i)$$

- Ha megáll az iteráció, akkor P(s) mindenütt igaz

Az algoritmus

$i = 0$

while True do

if $\text{not SAT}(I(s_0) \wedge lfpath(s^0, s^1, ..., s^i))$

or $\text{not SAT}((lfpath(s^0, s^1, ..., s^i) \wedge \neg P(s^i))$

then return True

if $\text{SAT}(I(s_0) \wedge path(s^0, s^1, ..., s^i) \wedge \neg P(s^i))$

then return $(s^0, s^1, ..., s^i)$

$i = i + 1$

end

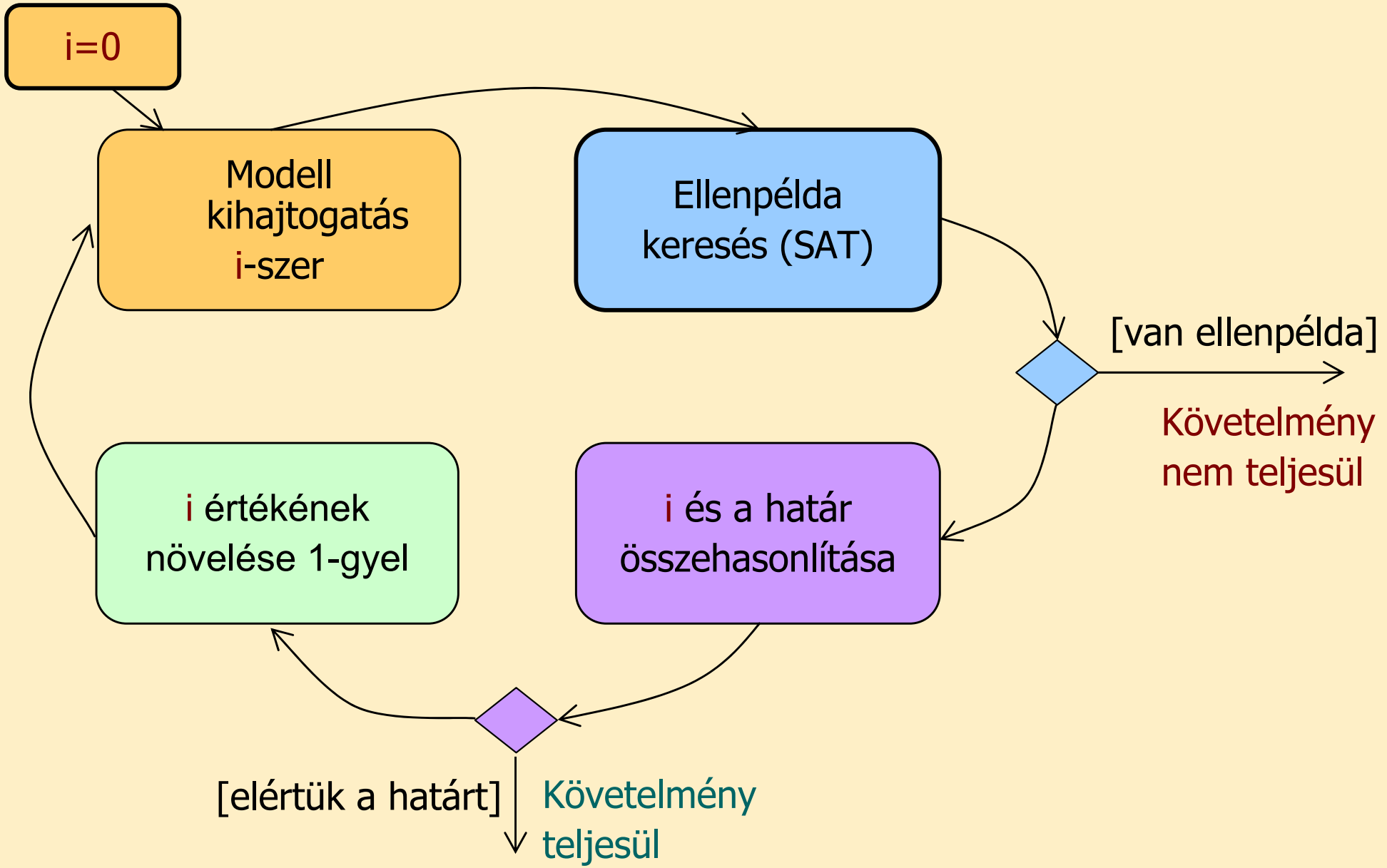
Nincs már i hosszú ciklusmentes út kezdőállapotból

Nincs i hosszú ciklusmentes út „rossz” állapotra

Van i hosszú út kezdőállapotból „rossz” állapotba

- Ha az eredmény True: Az invariáns igaz.
- Ha az eredmény egy $(s^0,s^1,...,s^i)$ útvonalat meghatározó (i+1)n bitérték: ez lesz az ellenpélda olyan állapot eléréséhez, ahol P(s) nem igaz

Korlátos modellellenőrzés iterációval



Az algoritmus finomítása

- Az iterálást nem 0-ról kezdjük
  - Adott k hosszú útvonallal kezdjük,
    és erre először az ellenpéldát próbáljuk meg generálni:
    - Ha van ilyen ellenpélda, akkor azt gyorsan megtaláljuk (iteráció nélkül)!
  - Ezután vizsgáljuk, hogy k+1-re terminál-e az iteráció,
    majd növeljük az útvonal hosszát (k-indukció)
- Nem garantált, hogy az eredményül kapott ellenpélda (útvonal) minimális hosszúságú
  - Nem 0-ról kezdtük az iterációt; ha k nagy, akkor „túllövünk a célon”
  - Itt k értékére heurisztika kell, ha rövid útvonalra törekszünk
- További megkötések a SAT megoldó bemenetére:
  - Az előre haladó útvonalakon ne legyenek kezdőállapotok
    (ez nem ciklust jelent, hiszen akár több kezdőállapot is lehet!)
  - A visszafelé haladó útvonalakon ne legyenek olyan köztes állapotok,
    ahol P(s) nem igaz (ezt a korábbi iteráció ellenpéldaként adná)

A finomított algoritmus

$i = k$

Kezdő érték

while True do

if $\mathrm{SAT}(I(s_0) \wedge path(s^0, s^1, ..., s^i) \wedge \neg \bigwedge_{j=0}^{i} (P(s^j))$

Van i hosszú útvonal kezdőállapotból „rossz” állapoton át

then return $(s^0, s^1, ..., s^i)$

if $\mathrm{not\ SAT}(I(s_0) \wedge \bigwedge_{j=1}^{i+1} (\neg I(s^j)) \wedge lfpath(s^0, s^1, ..., s^{i+1}))$

Nincs i+1 hosszú ciklusmentes út, (amiben nincs másik kezdőállapot)

or $\mathrm{not\ SAT}((lfpath(s^0, s^1, ..., s^{i+1}) \wedge \bigwedge_{j=0}^{i} P(s^j) \wedge \neg P(s^{i+1}))$

Nincs i+1 hosszú ciklusmentes út „rossz” állapotra (közben „jó” állapotokat érintve)

then return True

$i = i + 1$

end

Összefoglalás: A BMC használata

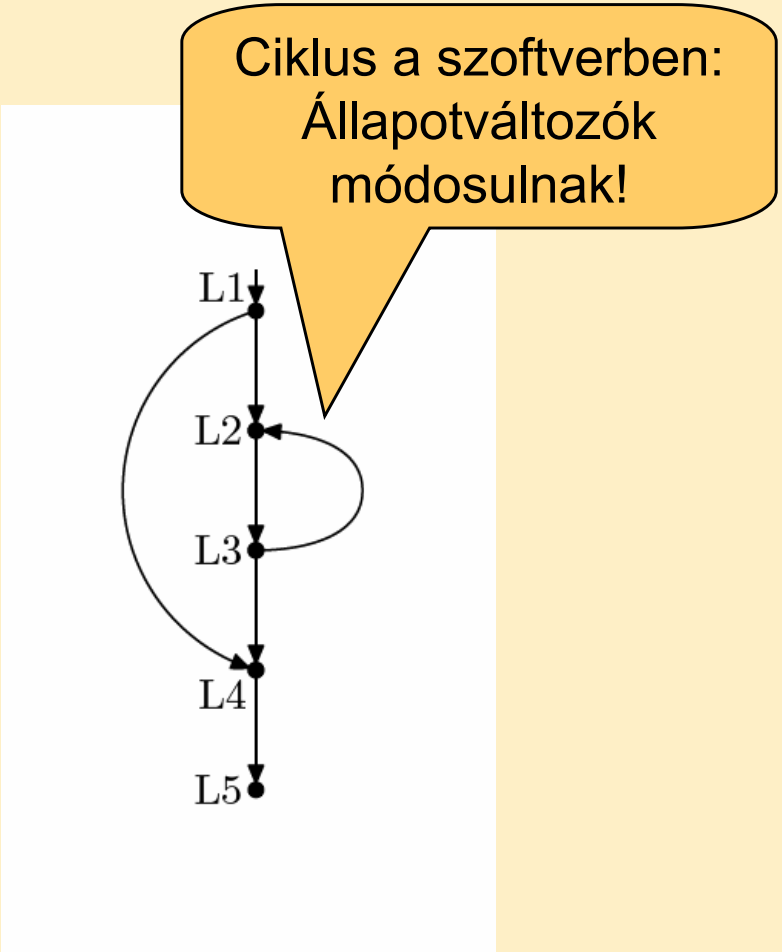
- Invariánsok vizsgálatára hatékony
  - Nem az általános modellellenőrzési feladat megoldása
- Helyes és teljes módszer ciklusmentes utakat használva
  - Ha van ellenpélda, azt megtalálja (egyébként az invariáns igaz)
  - Ha ellenpéldát talál, akkor az valódi ellenpélda
- Állapottér robbanásának „elkerülése”:
  - Adott számú iteráció vizsgálatával részleges eredmény kapható
- Legrövidebb ellenpélda keresése
  - Teszt generáláshoz használható
- Automatikus módszer
  - A korlát kijelölése lehet heurisztikus (az állapottér „átmérője”)
- Eszközök:
  - SAL: sal-smc, sal-bmc, sal-atg

Az Intel eredményei (hardver verifikáció)

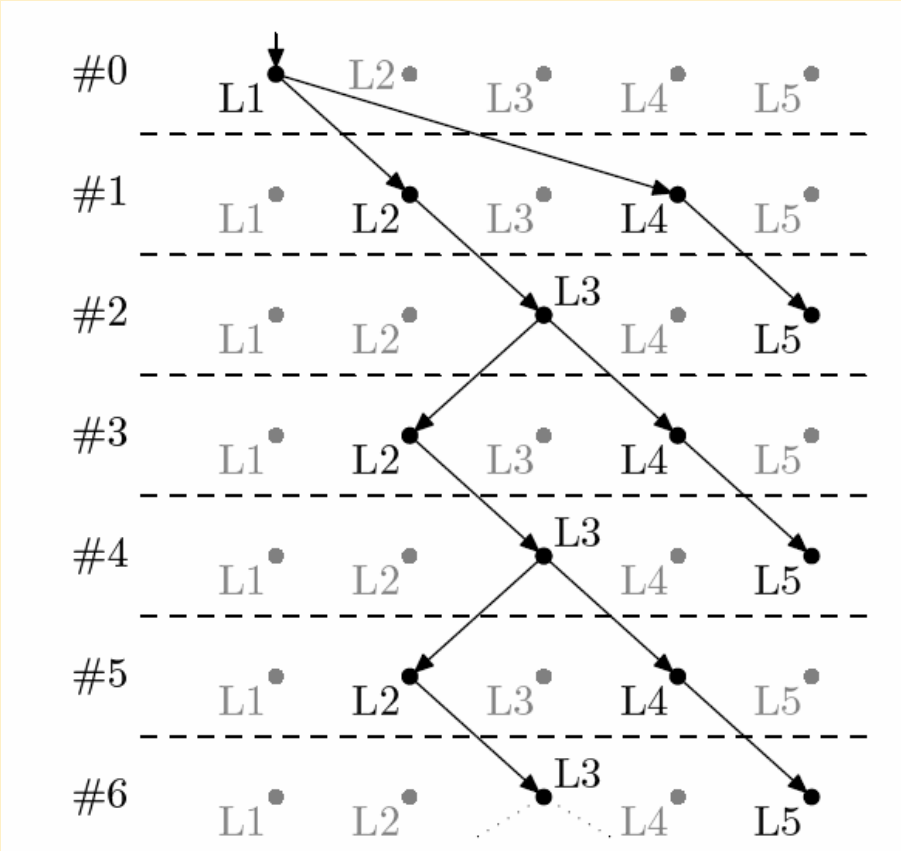
Model	k	Forecast (BDD)	Thunder (SAT)
Circuit 1	5	114	2.4
Circuit 2	7	2	0.8
Circuit 3	7	106	2
Circuit 4	11	6189	1.9
Circuit 5	11	4196	10
Circuit 6	10	2354	5.5
Circuit 7	20	2795	236
Circuit 8	28	—	45.6
Circuit 9	28	—	39.9
Circuit 10	8	2487	5
Circuit 11	8	2940	5
Circuit 12	10	5524	378
Circuit 13	37	—	195.1
Circuit 14	41	—	—
Circuit 15	12	—	1070

Alkalmazás szoftverek esetén: A ciklusok problémája

A ciklusok bejárása új állapotokat eredményez!



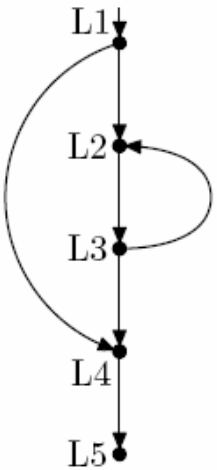
Példa modell

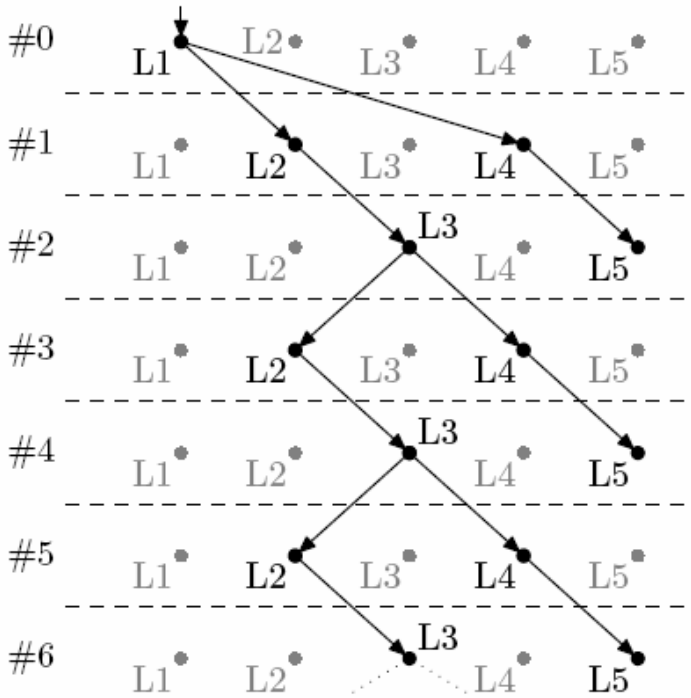


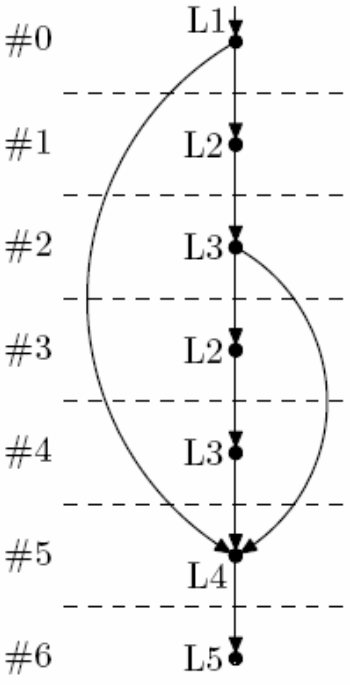
Teljes kihajtogatás

Korlátozott ciklusbejárás

- Modell kihajtogatás optimalizálása:
  - Alapeset: Teljes kihajtogatás (path enumeration)
    - Mindig szisztematikusan előre
  - Korlátozott ciklusbejárás (loop unrolling)
    - Ciklusokra egyenként lefutási korlátot adni és úgy kibontani







Szoftver modellellenőrzés

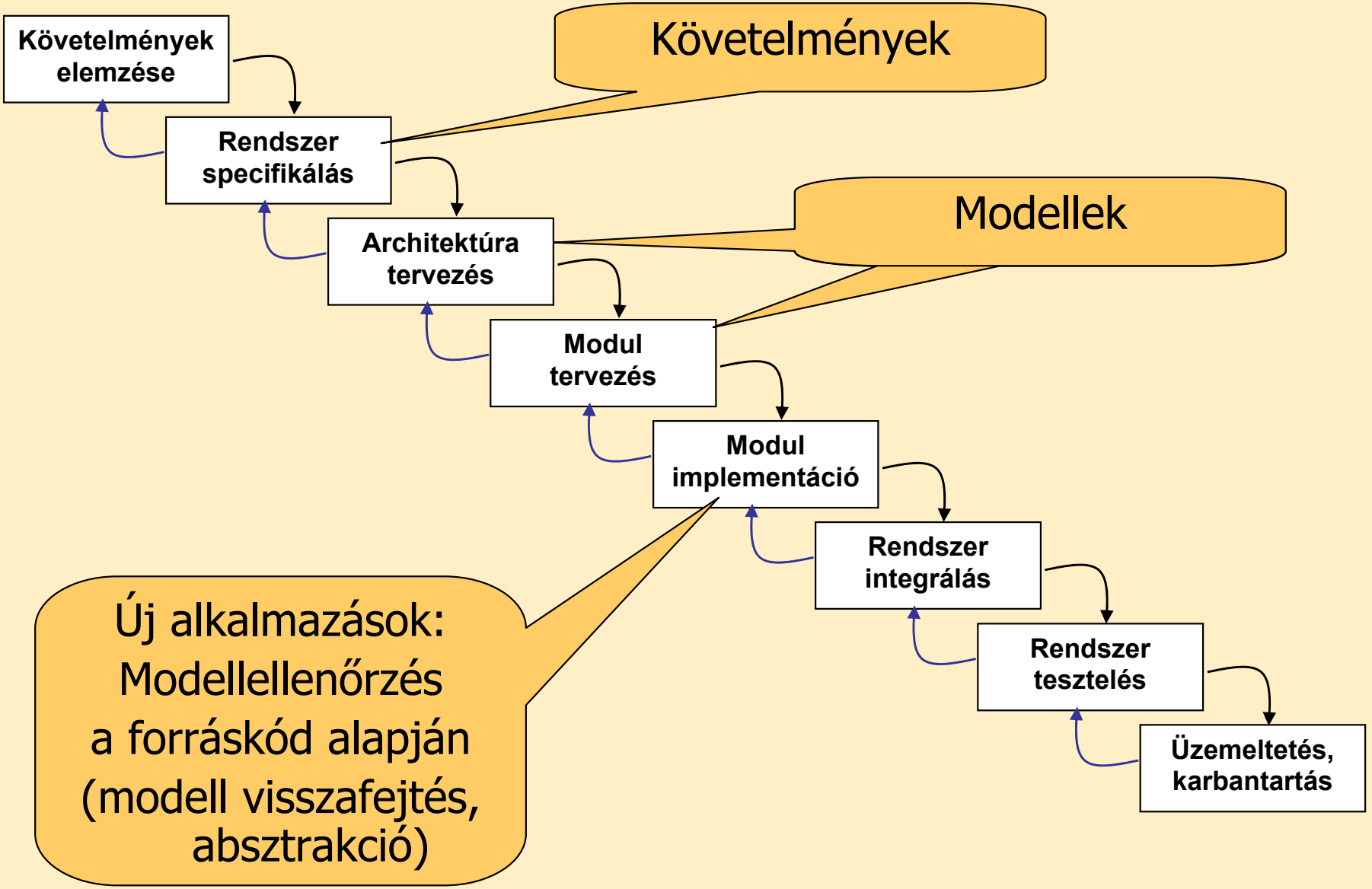
- F-SOFT (NEC):
  - Hagyományos teljes széthajtogatás
  - Unix rendszerprogramokra alkalmazták (pl. pppd)
- CBMC (CMU, Oxford University):
  - C, SystemC támogatása
  - Korlátozott ciklusbejárás (loop unrolling)
  - Egyes Linux, Windows, MacOS rendszerkönyvtárak támogatása
  - Integer aritmetikai műveletek leképzése:
    - Bitvektor szintű megvalósítás (bit-flattening, bit-blasting)
  - CBMC SMT megoldóval:
    - Satisfiability Modulo Theories: A SAT megoldó kiterjesztése különböző domének kezelésére (pl. integer aritmetika)
- SATURN:
  - Korlátozott ciklusbejárás: Max. 2 lefutás
  - Teljes Linux kernel ellenőrizhető: Null pointer hivatkozásokra

Összefoglalás

- Szimbolikus modellellenőrzés
  - ROBDD alakban kezelt karakterisztikus függvények
  - „Szimmetrikus” problémák esetén hatékony
    - Pl. azonos viselkedésű résztvevők protokollokban
  - Változók sorrendezésétől is függ a méret
- Korlátos modellellenőrzés invariánsokra
  - Logikai függvények igazságának keresése (SAT eszköz)
  - Korlátos hosszúságú ellenpéldák keresése
    - Ha ad ellenpéldát, az mindenképpen érvényes
    - Ha nincs ellenpélda, az még nem végleges eredmény (lehet, hogy hosszabb az ellenpélda)
  - Tesztgenerálásra jól alkalmazható

A modellellenőrzés jellemzői

Modellellenőrzés a tervezési folyamatban



A modellellenőrzés kedvező tulajdonságai

- Lehetséges nagy modellméretet is kezelni
  - Akár 10^{20}, de példa van 10^{100}-nál nagyobb méretre is
  - Ez a rendszermodell mérete (pl. automaták hálózata)
  - Hatékony technikák: Szimbolikus, SAT alapú (korlátos)
- Általános módszer
  - Szoftver, hardver, protokollok, …
- Teljesen automatikus eszköz, nem szükséges tervezői intuíció, erős matematikai háttérismeret
  - Tételbizonyítás ennél bonyolultabb!
- Ellenpéldát generál, ami segít a hibák javításában

2007. évi Turing Award a modellellenőrzés kifejlesztőinek:
E. M. Clarke, E. A. Emerson, J. Sifakis (1981)

A modellellenőrzés hiányosságai

- Skálázhatóság
  - Explicit állapottér bejárást alkalmaz
  - Hatékony technikák vannak ugyan erre, de nem garantált a jó skálázhatóság
- Elsősorban vezérlés-orientált alkalmazásokra
  - Komplex adatstruktúrák hatalmas állapotteret jelentenek
- Nehéz az eredményeket általánosítani
  - Ha egy protokoll helyes 2 résztvevő esetén, akkor helyes-e N résztvevő esetén?
- A követelmények formalizálása nehéz egy átlagos szoftver mérnök számára
  - Temporális logika „nyelvjárások” alakultak ki különböző alkalmazási területeken
  - Példa: PSL (Property Specification Language, IEEEE szabvány)