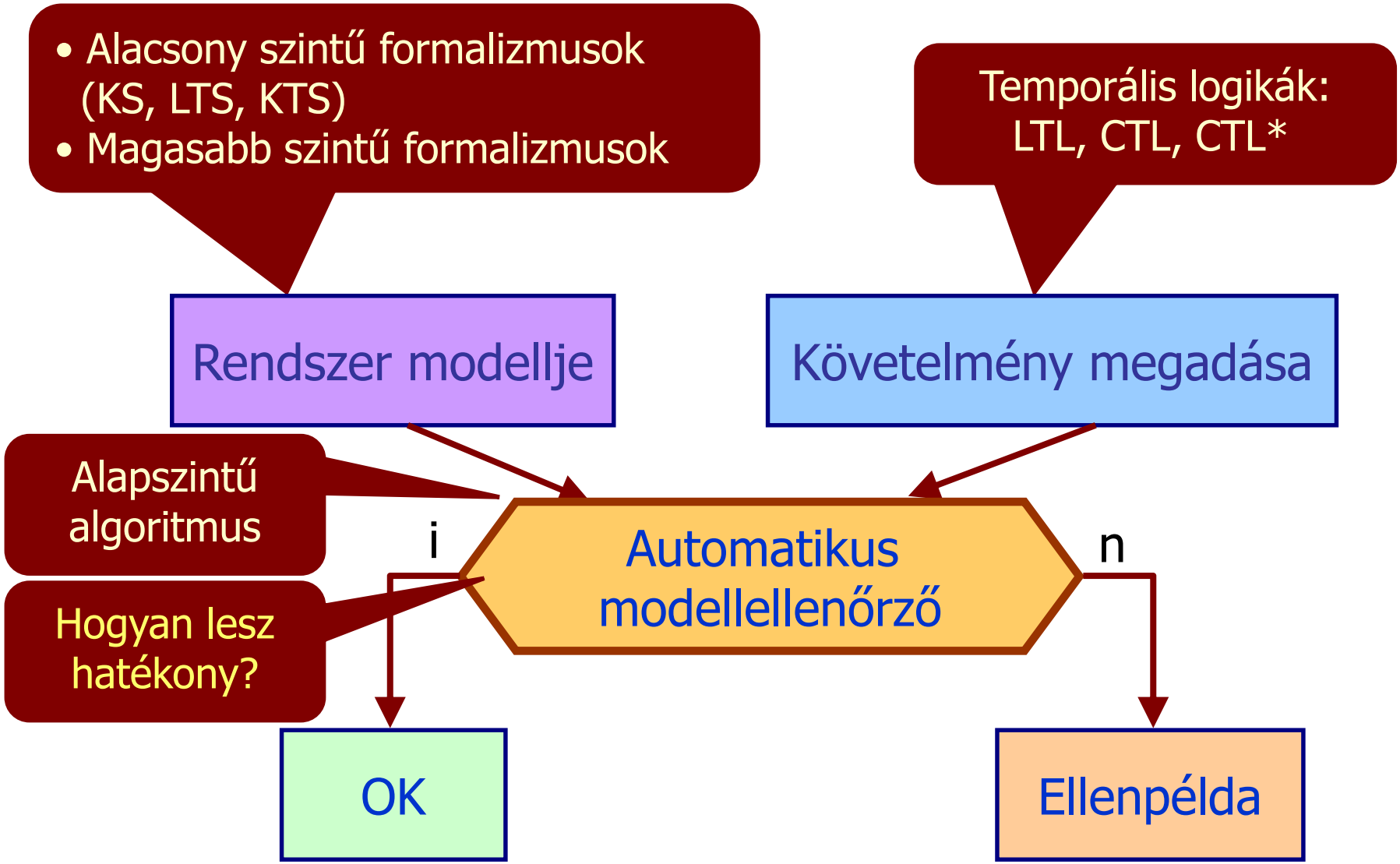
Hatékony technika modellellenőrzéshez: Szimbolikus modellellenőrzés és ROBDD

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

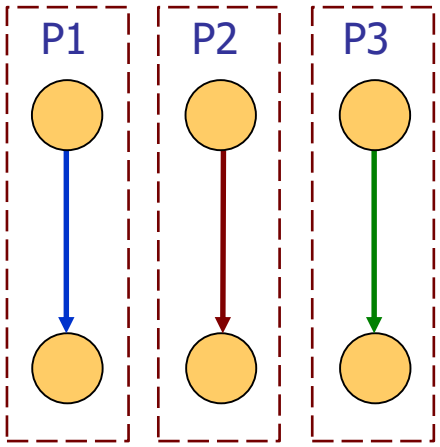
BME Méréstechnika és Információs Rendszerek Tanszék

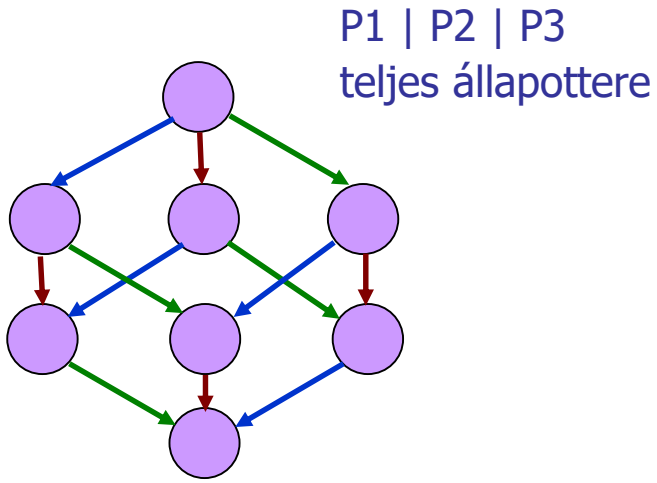
Hol tartunk?


• Alacsony szintű formalizmusok (KS, LTS, KTS)
• Magasabb szintű formalizmusok
Temporális logikák: LTL, CTL, CTL*
Rendszer modellje
Követelmény megadása
Alapszintű algoritmus
Hogyan lesz hatékony?
i
Automatikus modellellenőrző
n
OK
Ellenpélda

Problémák

- Nagyméretű állapottér ellenőrzése lehet szükséges
  - Konkurens komponensek esetén nagyméretű állapottér adódik: Független állapotátmenetek végrehajtása sokféle sorrendben történhet
  - „Állapottér robbanás”: exponenciális lehet a komponensek számával





- Hogyan lehet nagyméretű modelleket ellenőrizni?

Lehetőségek

Symbolic Model Checking: 10^{20} States and Beyond

J. R. Burch E. M. Clarke K. L. McMillan*
School of Computer Science
Carnegie Mellon University

D. L. Dill L. J. Hwang
Stanford University

1992

Handbook of Model Checking, 2018

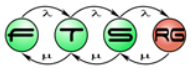
TDK 2011

With BDDs, blocks with 10^{50} states or more could be checked. This corresponds to 100–200 state bits, in contrast to around 20 state bits for explicit search.

Szaturáció alapú automatikus modellellenőrző fejlesztése aszinkron rendszerekhez

Darvas Dániel

Konzulensek:
dr. Bartha Tamás (BME MIT)
Vörös András (BME MIT)



Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Legnagyobb bejárt állapottér

10^{62900}

DPhil-100k modell

Ennek eléréséhez nagyfokú szimmetria és helyes dekompozíció szükséges a modellben.

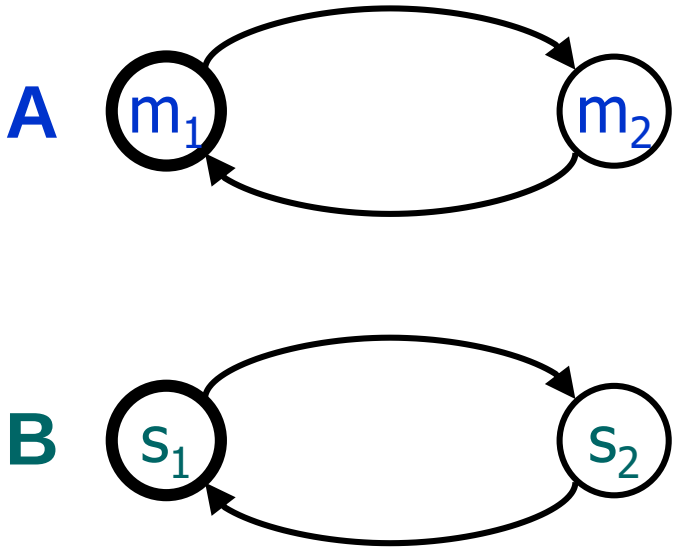
Kitérő: Mi határozza meg az állapottér nagyságát?

Konkurens automaták együttes működése

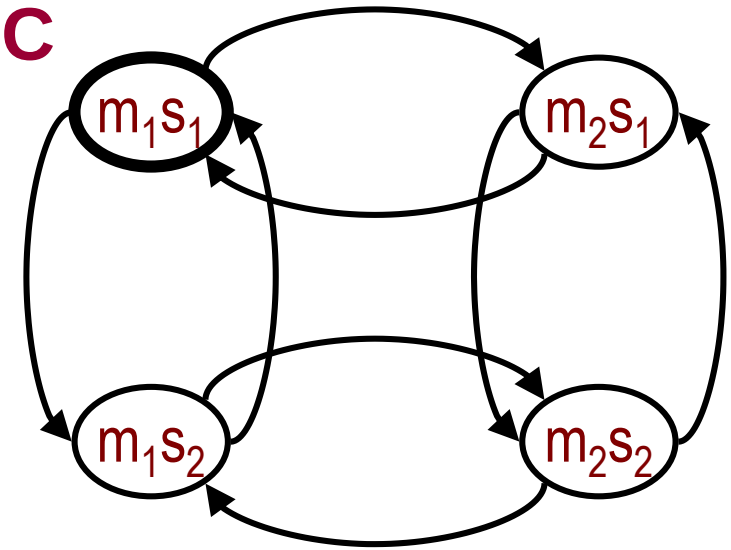
Automaták direkt szorzata,
átlapolás, szinkronizáció

Példa: Konkurens automaták működése

- Két független automatából álló rendszer:



- (Direkt) szorzatautomata: a rendszer állapottere



- Az automaták állapotai:

  $A = \{m_1, m_2\}$

  $B = \{s_1, s_2\}$

- Az állapotok halmaza:

  $C = A \times B$

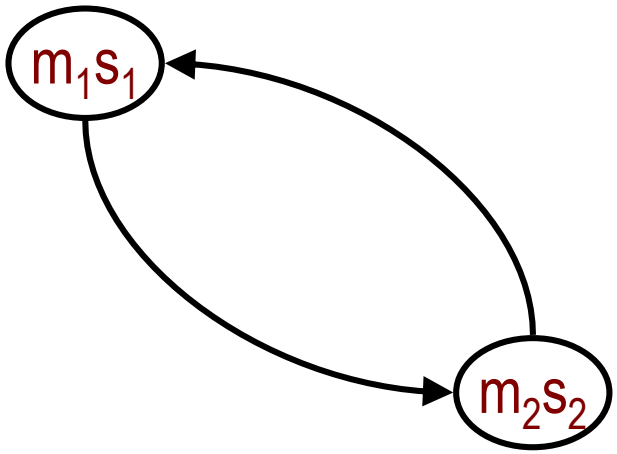
  $C = \{m_1s_1, m_1s_2, m_2s_1, m_2s_2\}$

Példa: Megkötések az állapotátmenetekre

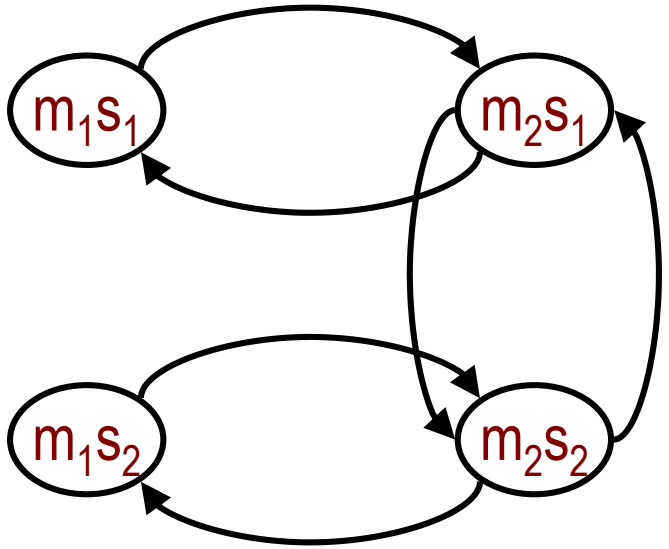
- Szinkronizáció: Élpárok együttes lépése
- Példa: A és B csak egyszerre válthat állapotot az azonos indexű állapotokból

- Korlátozó feltétel: állapotátmenetek tiltása
- Példa: B csak akkor vált állapotot, ha A az m_2 állapotban van

C'

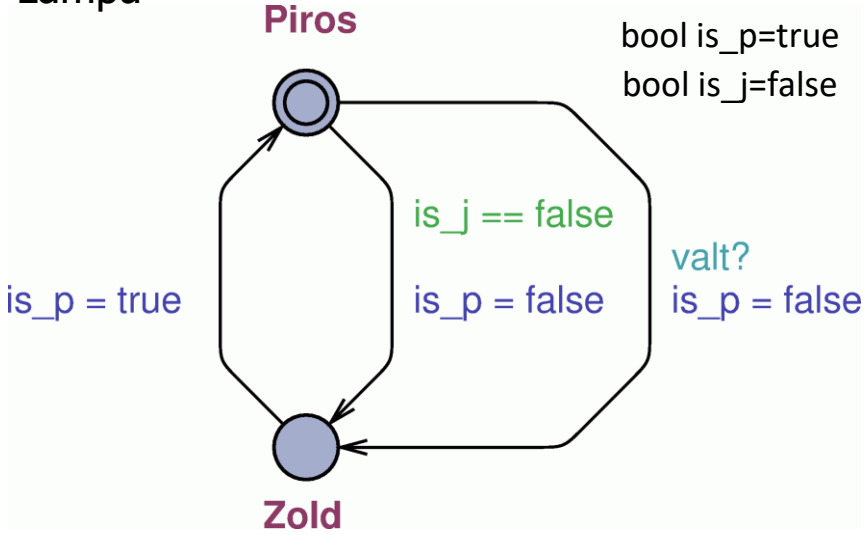


C"

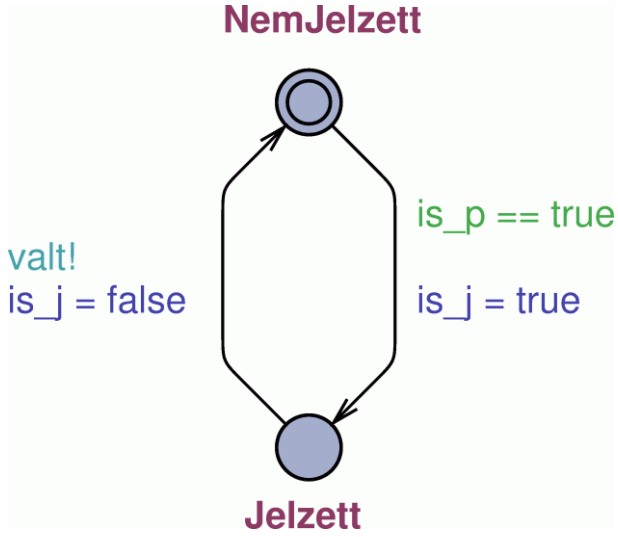


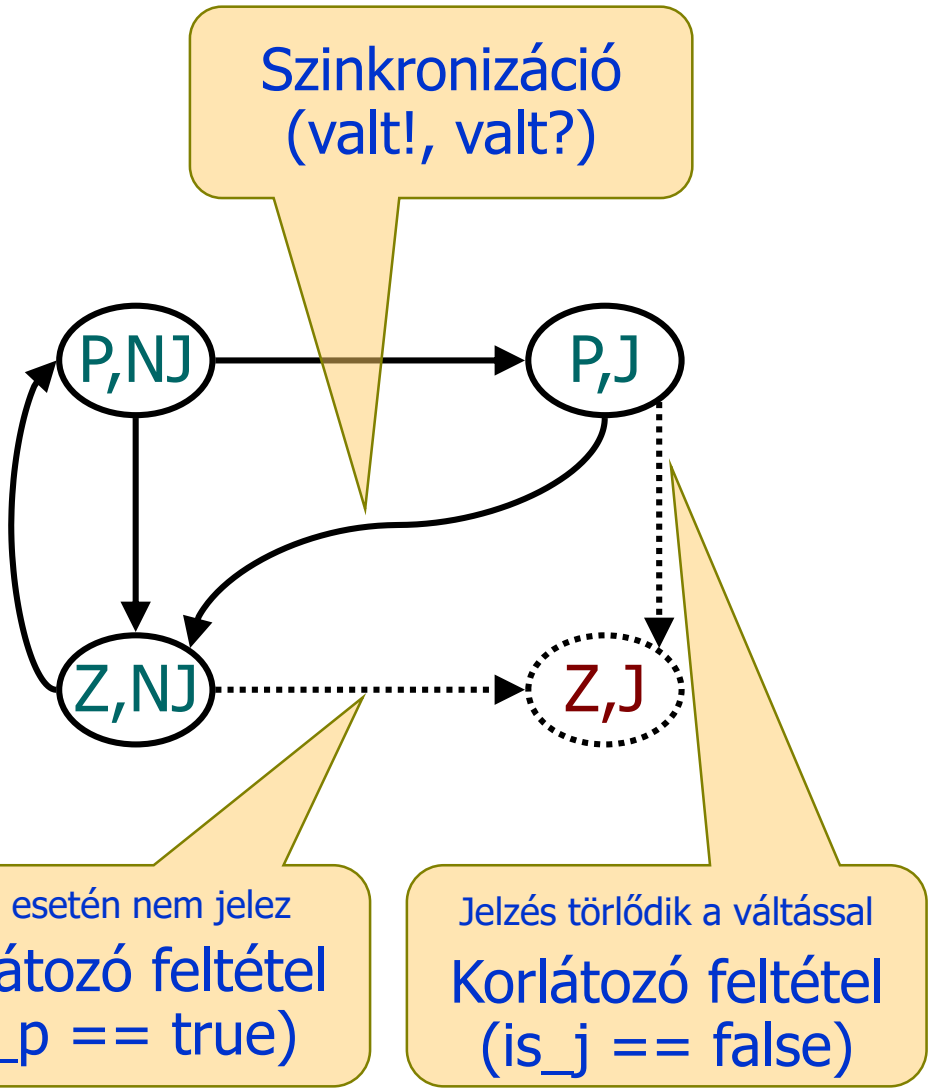
Példa: Gyalogos lámpa jelzőgombbal

Lámpa

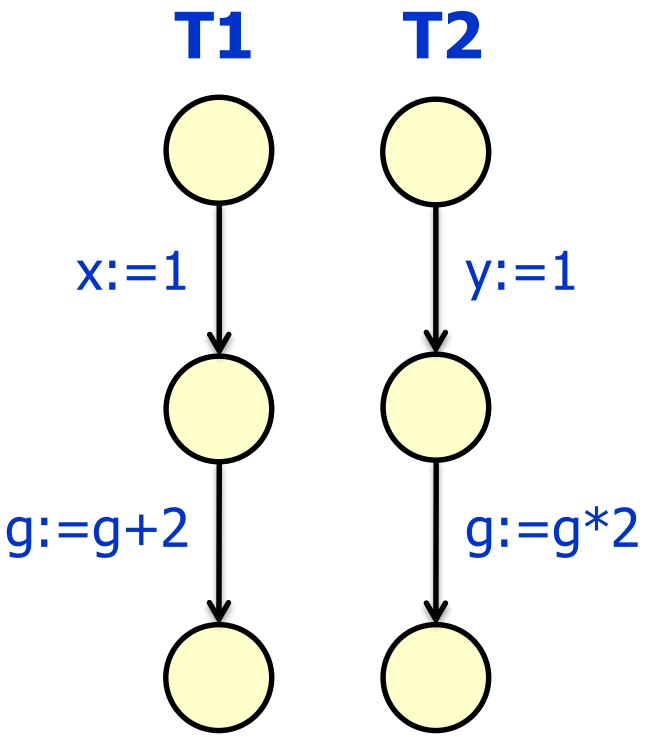


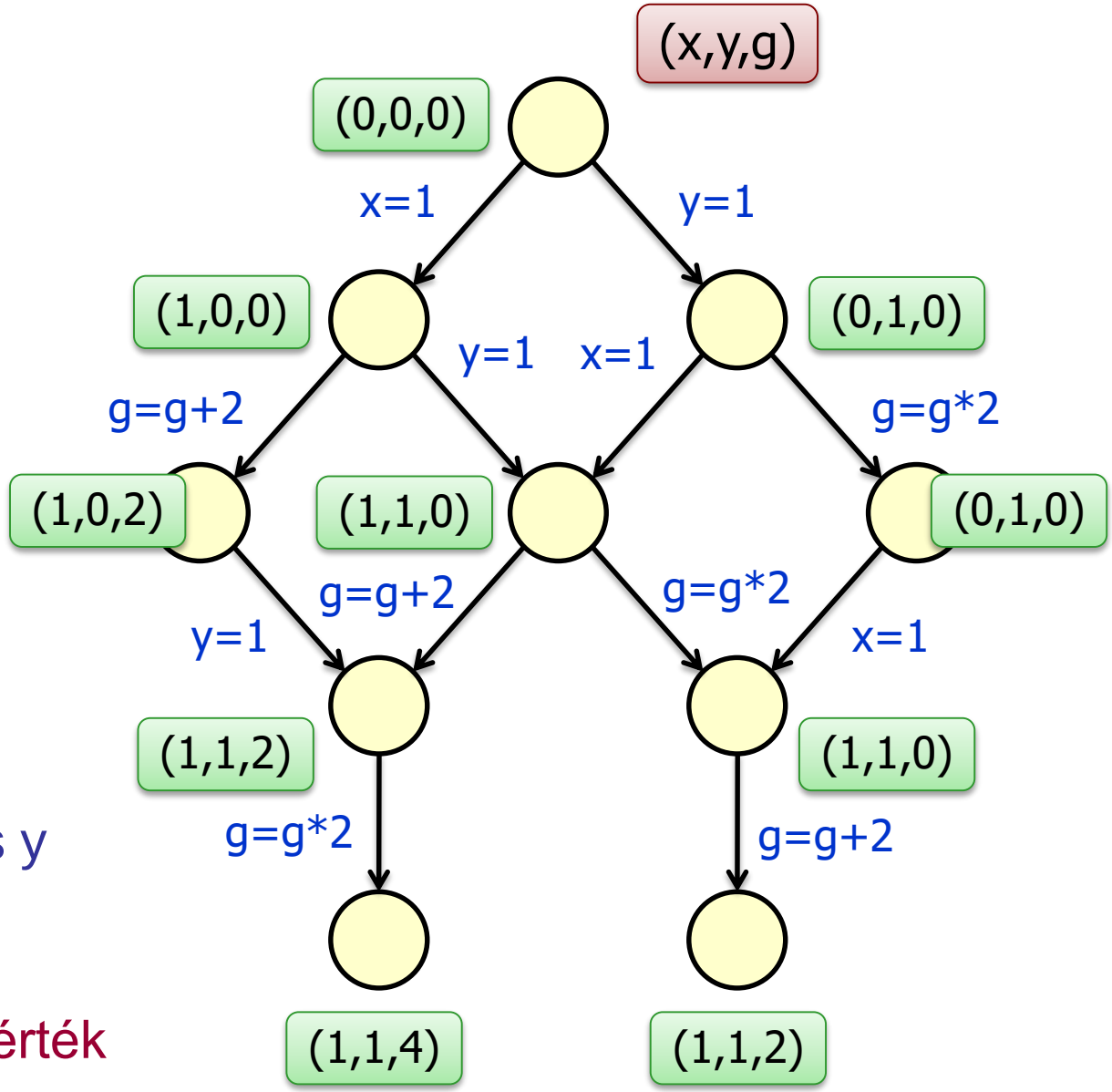
Gyalogos





Példa: Változók értékének figyelembe vétele





- Lokális változók: x és y
- Globális változó: g=0
- Utasítás végrehajtási sorrendtől függő végérték

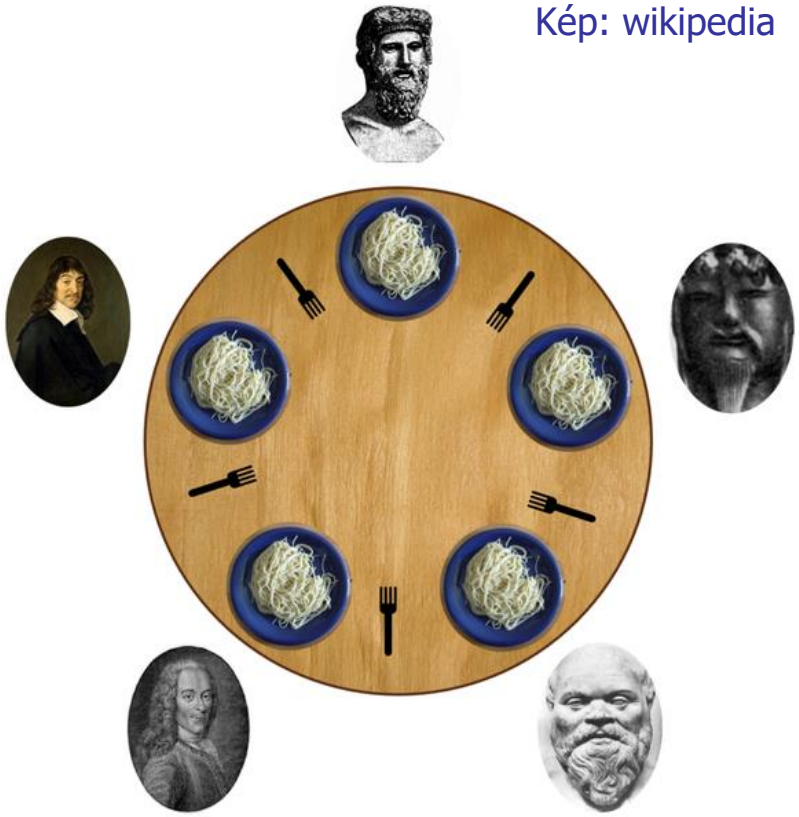
Példa nagy állapottérre: Étkező filozófusok

- 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	$> 10^{200}$
…	…

Ha 64 bittel címezek egy tárat, max. $2^{64} = 1{,}8 \cdot 10^{19}$ tárhely címezhető

Kép: wikipedia

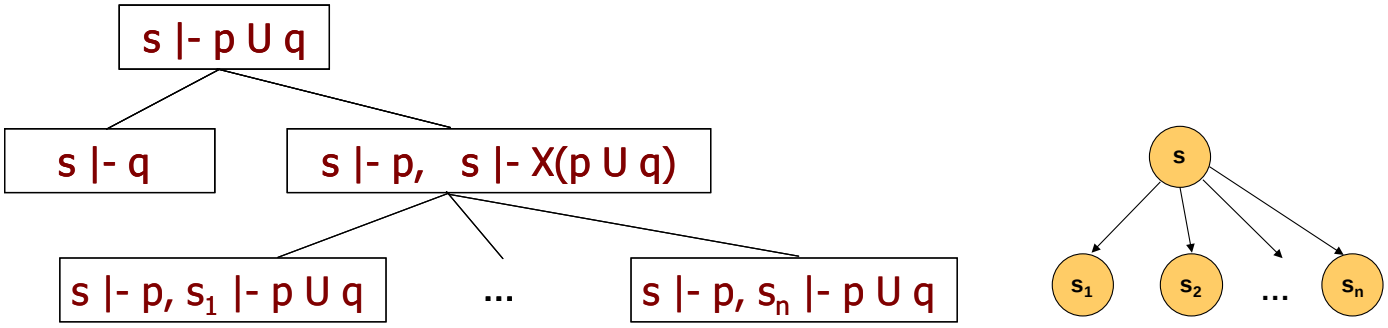


Okos (de nem feladat-specifikus) állapottér tárolással:
kb. 100 000 filozófus, azaz $10^{62\,900}$ állapottér is ellenőrizhető!

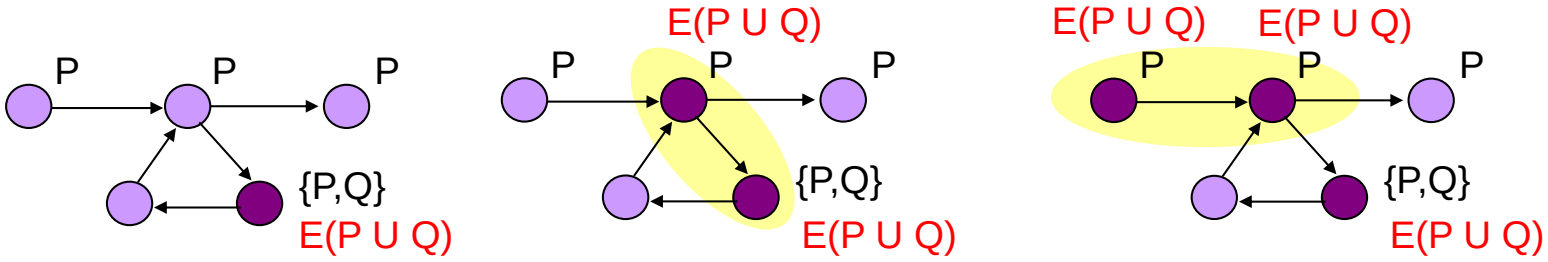
Szimbolikus modellellenőrzés

Ismétlés: A modellellenőrzés tanult technikái

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

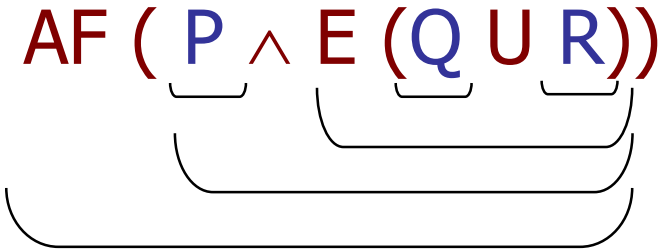


- CTL modellellenőrzés:
  - Szemantika alapú módszer: Állapotok iteratív címkézése



Ismétlés: CTL modellellenőrzés állapotcímkézéssel

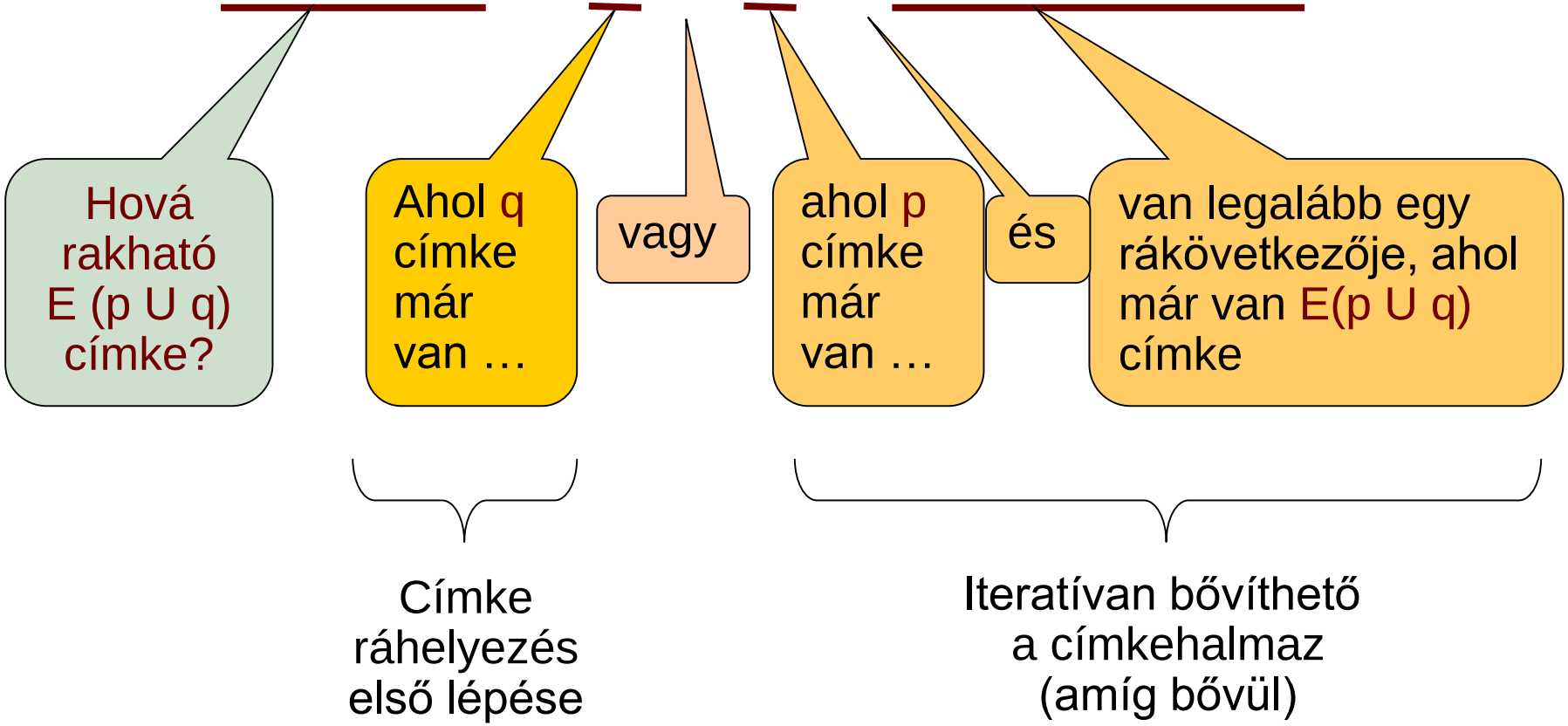
- Állapotcímkézés: Hol igaz egy adott (rész)kifejezés?



- Címkézési módszer
  - Kiindulás: KS címkézve van atomi kijelentésekkel
  - Tovább: Címkézés egyre összetettebb részkifejezésekkel
    - 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: E(p U q) modellellenőrzés

$$E(p\ U\ q) = q \vee (p \wedge EX\ E(p\ U\ q))$$



Ismétlés: Definíciók halmazműveletekhez

- Ismétlés: Címkézési szabályok
  - Mely s állapotok címkézhetők E(p U q)-val?
    - Ha s címkézett q-val, vagy
    - ha s címkézett p-vel,
      és legalább egy rákövetkezője már címkézett E(p U q)-val
  - Mely s állapotok címkézhetők A(p U q)-val?
    - Ha s címkézett q-val, vagy
    - ha s címkézett p-vel,
      és minden rákövetkezője már címkézett A(p U q)-val
- Állapothalmaz definíciók
  - Már címkézett Z állapothalmaz alapján:

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

$pre_A(Z) = \{s \in S \mid$ minden s'-re, ahol $(s,s') \in R$: $s' \in Z\}$

pre_E(címkézett)

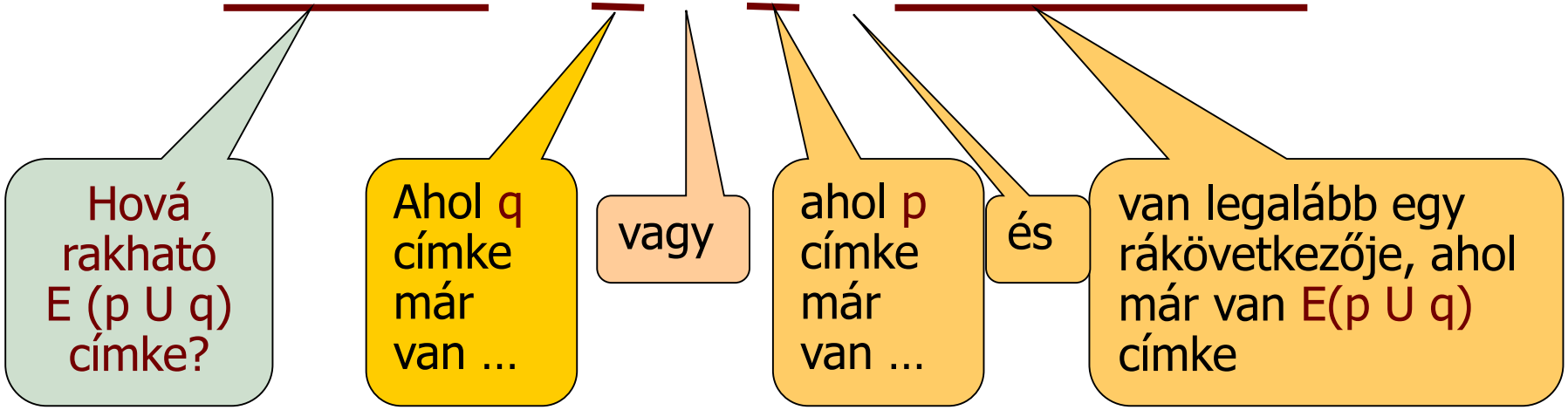
pre_A(címkézett)

Legalább egy rákövetkezője címkézett (Z-ben)

Minden rákövetkezője címkézett (Z-ben)

Ismétlés: E(p U q) modellellenőrzés algoritmusa

$$E(p\ U\ q) = q \vee (p \wedge EX\ E(p\ U\ q))$$



Címke ráhelyezés első lépése

$Z_0 = \{s \mid q \in L(s)\}$

Iteratívan bővíthető a címkehalmaz (amíg bővül)

$Z_{i+1} = Z_i \cup (\{s \mid p \in L(s)\} \cap pre_E(Z_i))$

A szimbolikus modellellenőrzés alapötlete

- Az állapotok és állapothalmazok felsorolás helyett logikai függvényekkel vannak megadva
  - Állapot „kódolása” n bites bitvektorral
    - S állapothalmaz kódolásához $n=\lceil \log_2|S| \rceil$ bit elég, azaz válasszunk olyan n értéket, hogy legyen $2^n \geq |S|$
  - Állapot reprezentálása n-változós logikai függvénnyel
    - Karakterisztikus függvény: $C: \{0,1\}^n \rightarrow \{0,1\}$
    - Egy állapot karakterisztikus függvénye csakis az állapotot kódoló bitvektor behelyettesítésére legyen igaz
  - Állapothalmaz reprezentálása n-változós logikai függvénnyel
    - Egy állapothalmaz karakterisztikus függvénye akkor és csak akkor legyen igaz egy-egy bitvektor behelyettesítésére, ha a bitvektor által kódolt állapot az adott állapothalmazban van
- Cél: Állapothalmazokon végzett műveletek halmazok helyett a karakterisztikus függvényeken

Precízen: Karakterisztikus függvények

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

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

  Cél: $C_s(x_1, x_2, \ldots, x_n)$ 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: $C_s(x_1, x_2) = \neg x_1 \wedge x_2$

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

  Cél: $C_Y(x_1, x_2, \ldots, x_n)$ akkor legyen igaz egy $(u_1, u_2, \ldots, u_n)$ behelyettesítésre, ha $(u_1,u_2,\ldots, u_n) \in Y$

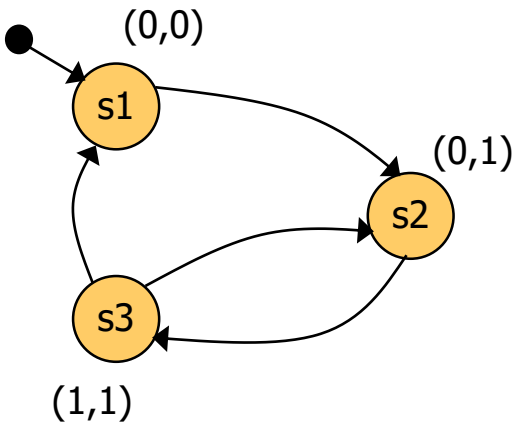
  $C_Y(x_1, x_2, \ldots, x_n)$ konstruálása:

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

- Állapothalmazok uniójára, metszetére:

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

Példa: Állapotok karakterisztikus függvénye



Változók: x, y

Állapotok karakterisztikus függvényei:

s1 állapot:

$$C_{s1}(x,y) = (\neg x \wedge \neg y)$$

s2 állapot:

$$C_{s2}(x,y) = (\neg x \wedge y)$$

s3 állapot:

$$C_{s3}(x,y) = (x \wedge y)$$

Állapothalmazok karakterisztikus függvénye:

{s1,s2} állapothalmaz:

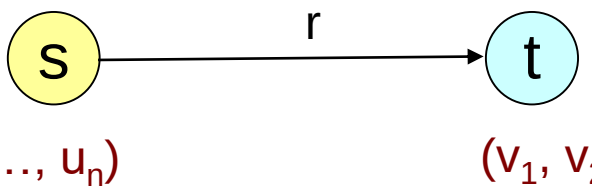
$$C_{\{s1,s2\}} = C_{s1} \vee C_{s2} = (\neg x \wedge \neg y) \vee (\neg x \wedge y)$$

{s1,s2,s3} állapothalmaz:

$$C_{\{s1,s2,s3\}} = C_{s1} \vee C_{s2} \vee C_{s3} = (\neg x \wedge \neg y) \vee (\neg x \wedge y) \vee (x \wedge y)$$

Állapotátmenetek karakterisztikus függvénye

- Egy r állapotátmenetre: C_r karakterisztikus függvény

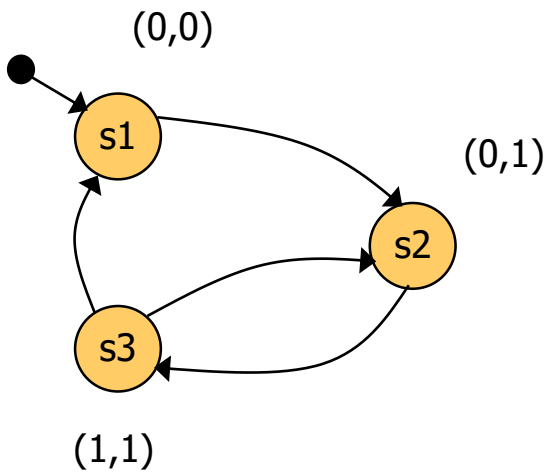


r=(s,t) állapotátmenet, ahol $s=(u_1, u_2, \ldots, u_n)$ és $t=(v_1, v_2, \ldots, v_n)$

  - Karakterisztikus függvény $C_r(x_1, x_2, \ldots, x_n, x'_1, x'_2, \ldots, x'_n)$ alakban
    - „Eredeti” n változó a forrás állapothoz
    - „Vesszős” n változó a cél állapothoz
  - Cél: C_r a.cs.a. legyen igaz, ha a forrás és a cél állapot bitvektorát helyettesítjük be, azaz ha $x_i=u_i$ és $x_i'=v_i$ a behelyettesítés
    - C_r konstruálása:

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

Példa: Állapotátmenetek karakterisztikus függvénye



s1 állapot:

$$C_{s1}(x,y) = (\neg x \wedge \neg y)$$

s2 állapot:

$$C_{s2}(x,y) = (\neg x \wedge y)$$

(s1,s2)$\in$R állapotátmenet:

$$C_{(s1,s2)} = (\neg x \wedge \neg y) \wedge (\neg x' \wedge y')$$

Állapotátmeneti reláció: Összes átmenet

$$R(x,y,x',y') = (\neg x \wedge \neg y \wedge \neg x' \wedge y') \vee (\neg x \wedge y \wedge x' \wedge y') \vee (x \wedge y \wedge \neg x' \wedge y') \vee (x \wedge y \wedge \neg x' \wedge \neg y')$$

$pre_E(Z)$ halmaz karakterisztikus függvénye

- $pre_E(Z)$ képzése: $pre_E(Z)=\{s \mid \exists s': (s,s') \in R \text{ és } s' \in Z\}$
  - adott a Z karakterisztikus függvénye: C_Z
  - adott az R karakterisztikus függvénye: $C_R=\vee_{r \in R} C_r$
  - képezhető $pre_E(Z)$: kikeresni a Z-beli állapotokra az előzőeket

$$C_{\mathrm{pre}_E(Z)} = \exists_{x'_1, x'_2, \ldots, x'_n} C_R \wedge C'_Z$$

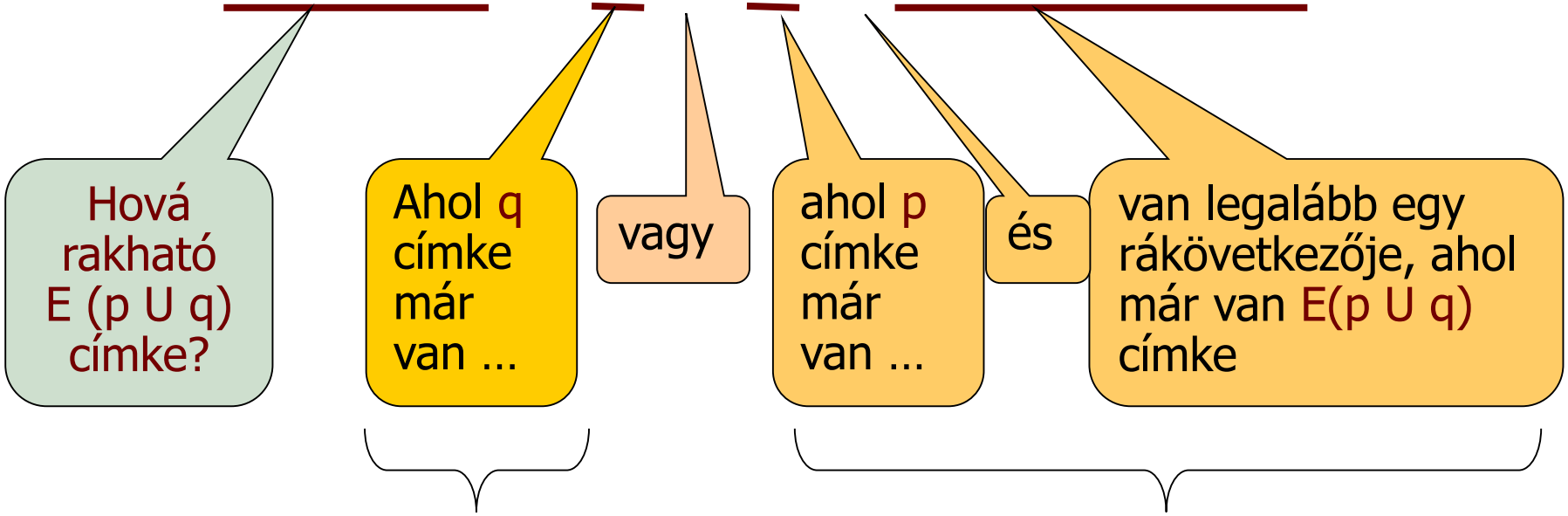
Z-ben a cél (rákövetkező) állapotok vannak

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)

E(p U q) modellellenőrzés algoritmusa

$$E(p\ U\ q) = q \vee (p \wedge EX\ E(p\ U\ q))$$



Címke ráhelyezés első lépése

Iteratívan bővíthető a címkehalmaz (amíg bővül)

Halmazműveletek: $Z_0 = \{s \mid q \in L(s)\}$ $\quad Z_{i+1} = Z_i \cup (\{s \mid p \in L(s)\} \cap pre_E(Z_i))$

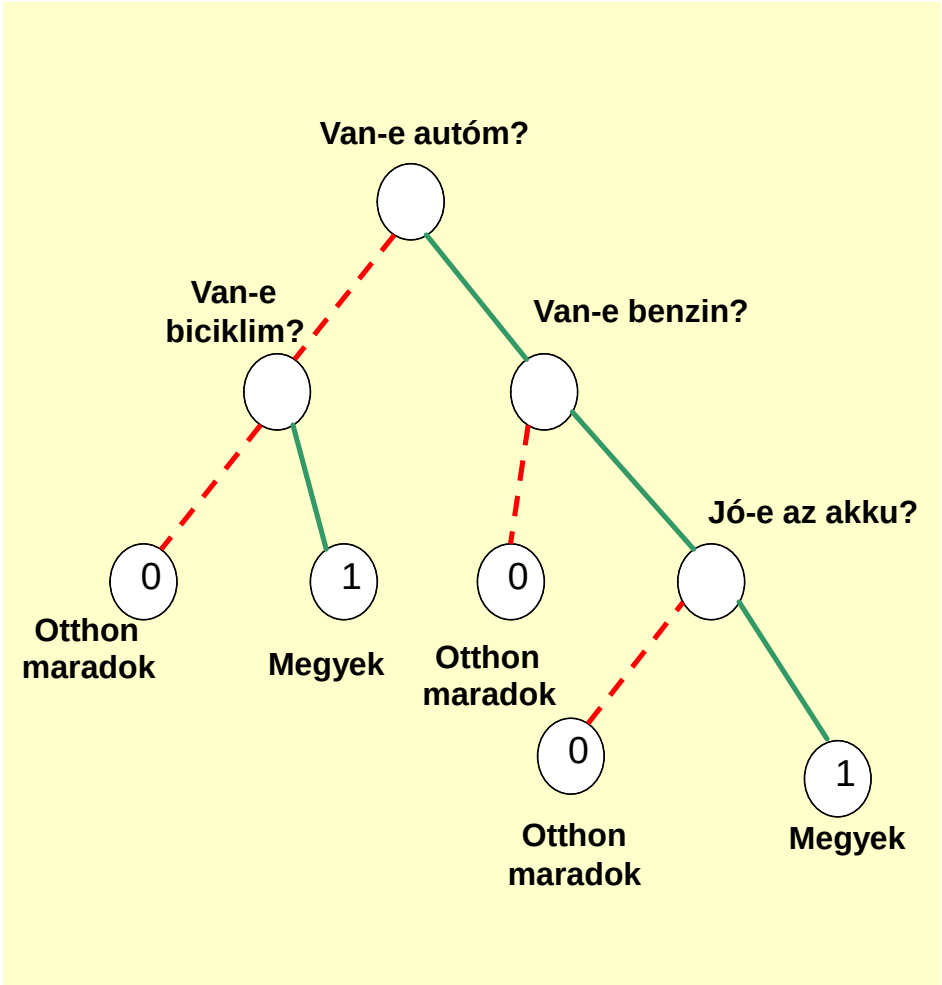
Függvényműveletek: $C_{Z_0} = C_q$ $\quad C_{Z_{i+1}} = C_{Z_i} \vee (C_p \wedge C_{pre_E(Z_i)})$

Logikai függvények reprezentációja: Az ROBDD bevezetése

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 (1) / nem (0)

Többértékű kiterjesztés is létezik



Boole függvények bináris döntési fa alakban

- Döntés: Egy x változó behelyettesítése a függvényben
  - „1” ág: A függvény x=1 behelyettesítéssel (jelölés: legyen f_x)
  - „0” ág: A függvény x=0 behelyettesítéssel (jelölés: legyen $f_{\underline{x}}$)
- A behelyettesítés formálisan (Shannon-felbontás):

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

  - Itt jelölés: Az if-then-else szerkezet

$$x \rightarrow f_x, f_{\underline{x}} \equiv (x \wedge f_x) \vee (\neg x \wedge f_{\underline{x}})$$

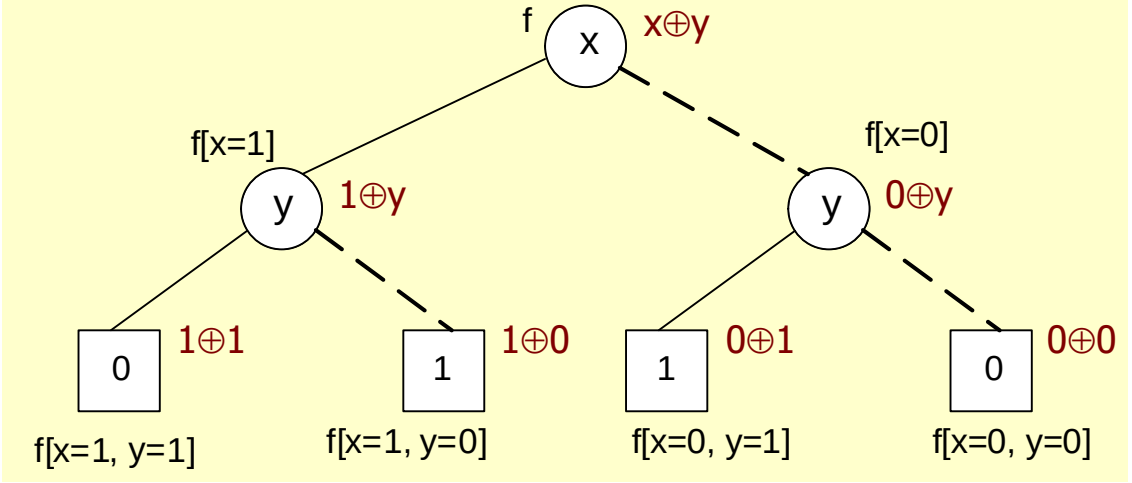
  - A kiemelt változó neve tesztváltozó, értékének vizsgálata a teszt
  - Tehát a függvényt az if-then-else szerkezet alapján felbonthatjuk
  - A then-else ágakban ezzel egy változóját eltüntettük, redukáltuk
  - Ciklikusan ismételjük, amíg van változó;
    a végén konstans 0 vagy 1 marad, ami a függvény értéke

Döntési fák típusai

Példa: Az xor függvény

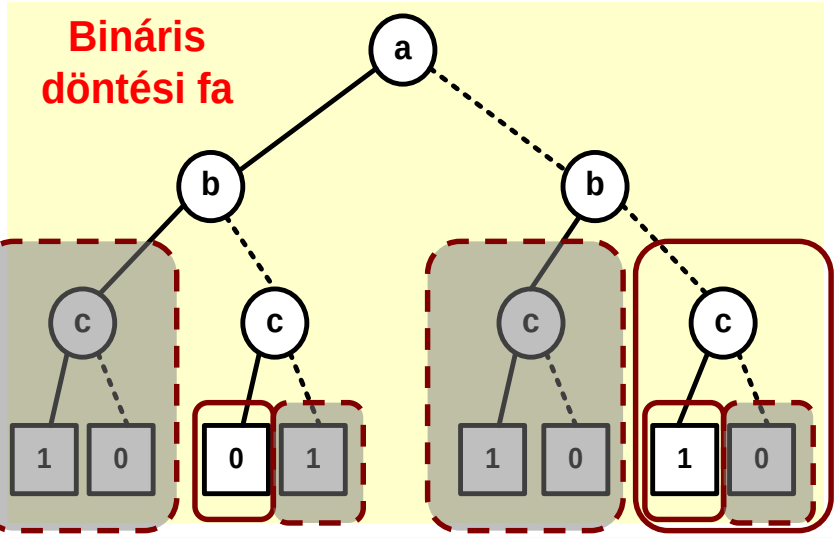
$$f(x,y)=x \oplus y$$

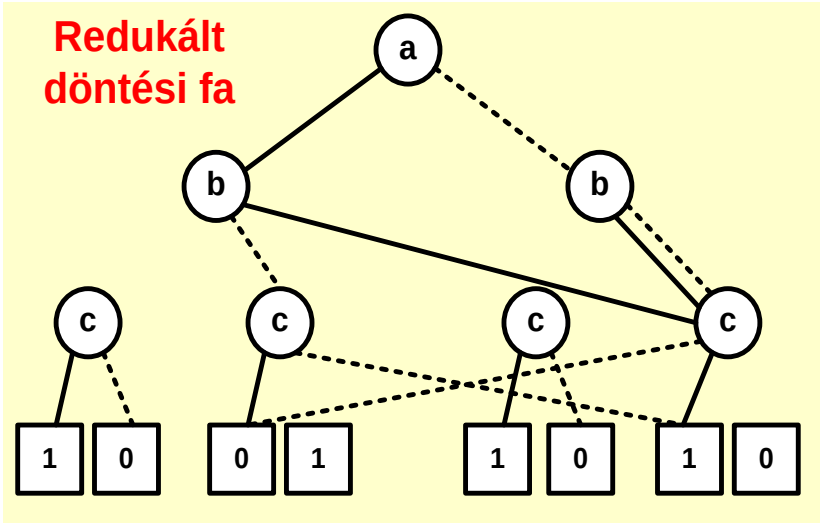
A döntési fában a leveleken jelennek meg f(x,y) egyes behelyettesítési értékei

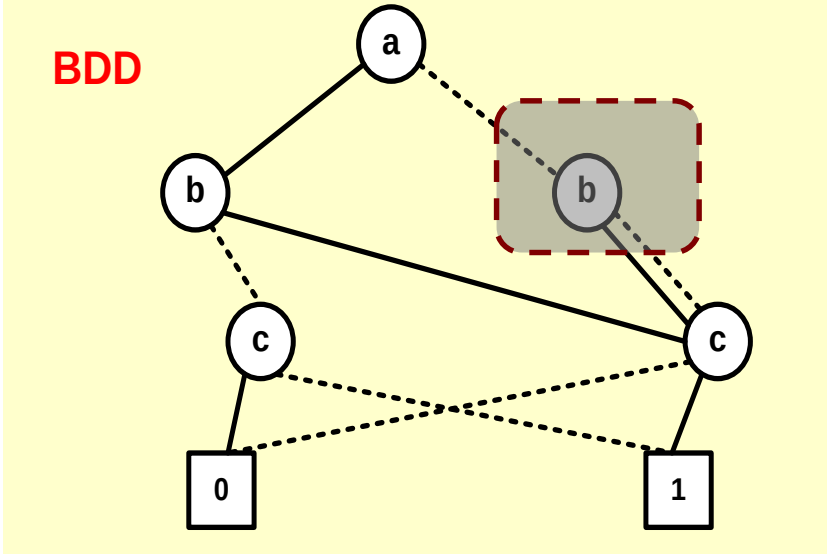


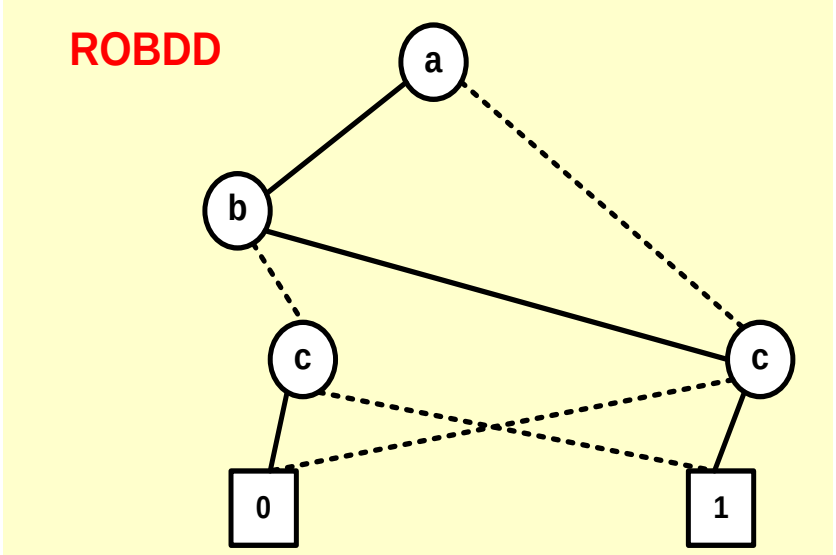
- Bináris döntési diagramot (BDD) kapunk, ha az azonos részfákat összevonjuk
- Rendezett bináris döntési diagramot (OBDD) kapunk, ha a felbontás során minden ágon azonos sorrendben vesszük fel a teszt változókat
- Redukált rendezett bináris döntési diagramot (ROBDD) kapunk, ha a szükségtelen csomópontokat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

Példa: Egy bináris döntési fa átalakítása









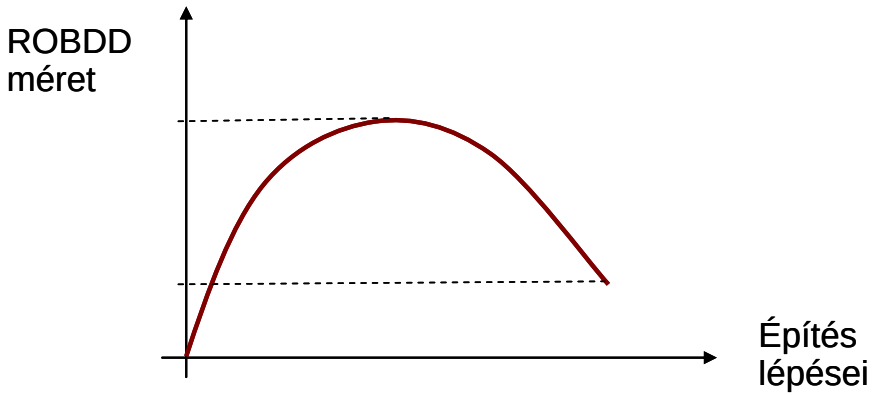
ROBDD tulajdonságok

- 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 a kimenő élek ugyanahhoz a csomóponthoz vezetnek, redukálva vannak

Egy adott függvény esetén két,
azonos változósorrendezésű ROBDD izomorf

ROBDD változók sorrendezése

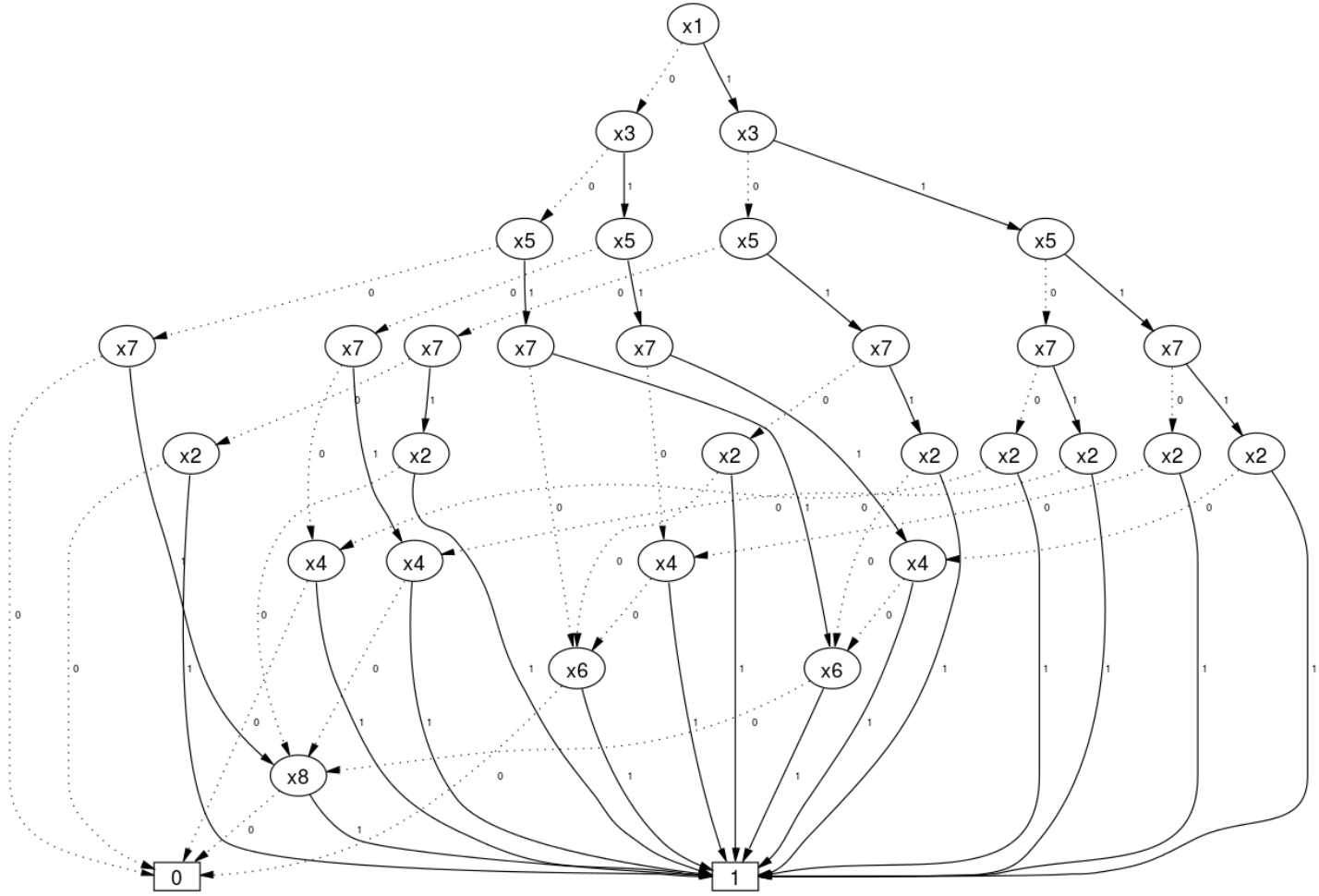
- ROBDD mérete
  - Egyes függvények (pl. páros paritás) esetén nagyon kompakt
  - Más függvények esetén exponenciális méret is lehet
- A méret szempontjából nagyon fontos a változók sorrendjének megválasztása
  - Más sorrend nagyságrendbeli különbséget is jelenthet
  - Optimális sorrend megtalálása NP-teljes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

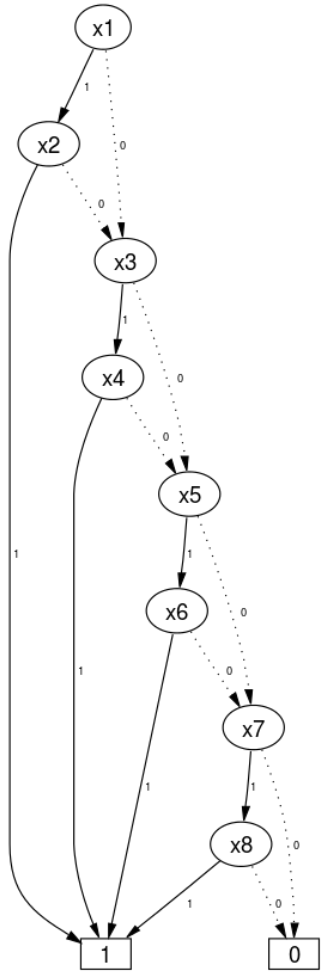


Példa: ROBDD mérete vs. változók rendezése

$f(x1, \ldots, x8) = (x1 \wedge x2) \vee (x3 \wedge x4) \vee (x5 \wedge x6) \vee (x7 \wedge x8)$

https://en.wikipedia.org/wiki/Binary_decision_diagram



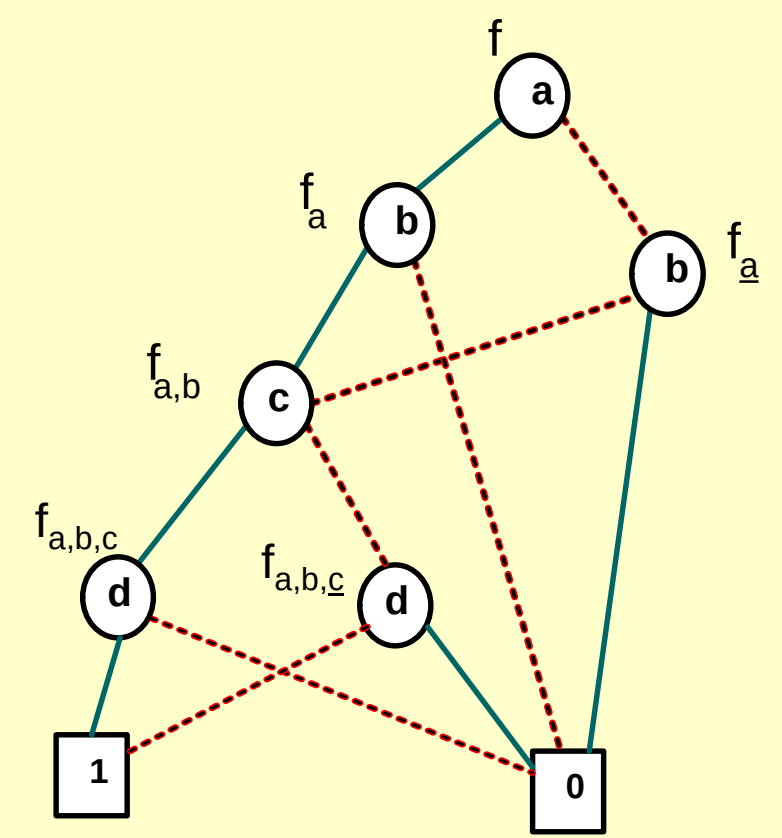


Példa: ROBDD kézi előállítása

Legyen az

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

Sorrend legyen: a, b, c, d



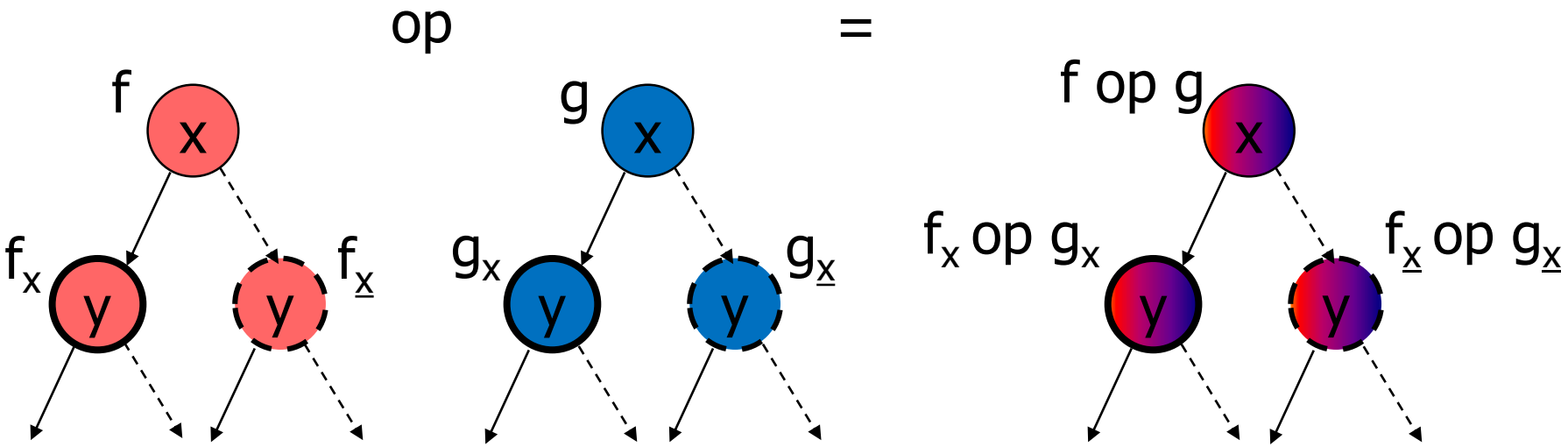
- $f = a \rightarrow f_a, f_{\underline{a}}$
  $f_a=(1\Leftrightarrow b)\wedge(c\Leftrightarrow d)$, $f_{\underline{a}}=(0\Leftrightarrow b)\wedge(c\Leftrightarrow d)$
- $f_a = b \rightarrow f_{a,b}, f_{a,\underline{b}}$
  $f_{a,b}= (1\Leftrightarrow 1)\wedge(c\Leftrightarrow d) = (c\Leftrightarrow d)$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

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

Műveletek ROBDD-ken

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

     Itt „1” ágat az „1” ággal, „0” ágat a „0” ággal kell bevonni



Műveletek ROBDD-ken (folytatás)

- Boole operátorokat közvetlenül ROBDD-ken hajtjuk végre
  - A két függvény változói azonosak legyenek, azonos sorrendben
- Alap azonosság f, g függvényekre (itt op Boole operátor):

1. $f \text{ op } g = (x \rightarrow f_x, f_{\underline{x}}) \text{ op } (x \rightarrow g_x, g_{\underline{x}}) = x \rightarrow (f_x \text{ op } g_x), (f_{\underline{x}} \text{ op } g_{\underline{x}})$

   Itt „1” ágat az „1” ággal, „0” ágat a „0” ággal kell bevonni

- További szabályok (redukálás miatt hiányzó változó esetei):

2. $f \text{ op } g = (x \rightarrow f_x, f_{\underline{x}}) \text{ op } g = x \rightarrow (f_x \text{ op } g), (f_{\underline{x}} \text{ op } g)$

   Itt g-ből hiányzik az x változó, nincsenek új ágak g-ben

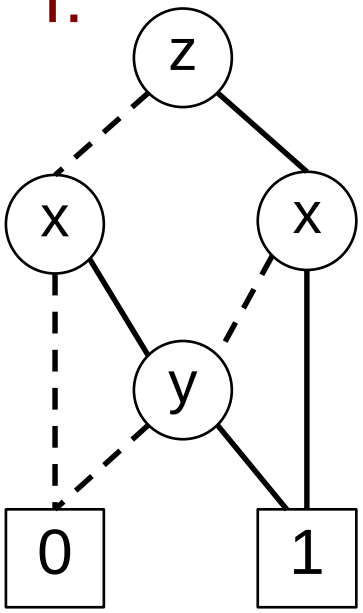
3. $f \text{ op } g = f \text{ op } (x \rightarrow g_x, g_{\underline{x}}) = x \rightarrow (f \text{ op } g_x), (f \text{ op } g_{\underline{x}})$

   Itt f-ből hiányzik az x változó, nincsenek új ágak f-ben

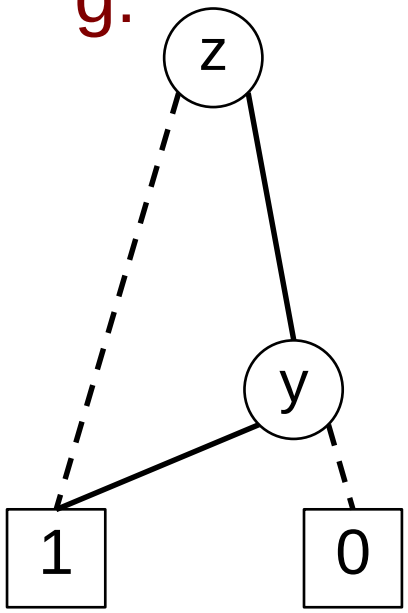
Példa: Művelet elvégzése (f∧g)

Az ROBBD alakok alapján készítsük el f∧g ROBBD-jét!

f:



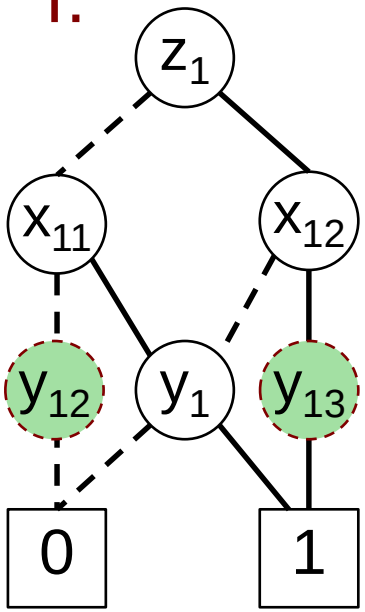
g:



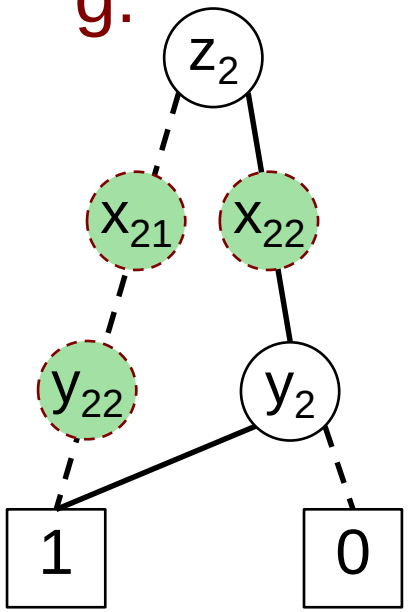
Példa: Művelet elvégzése (f∧g)

Az ROBBD alakok alapján készítsük el f∧g ROBBD-jét!

f:

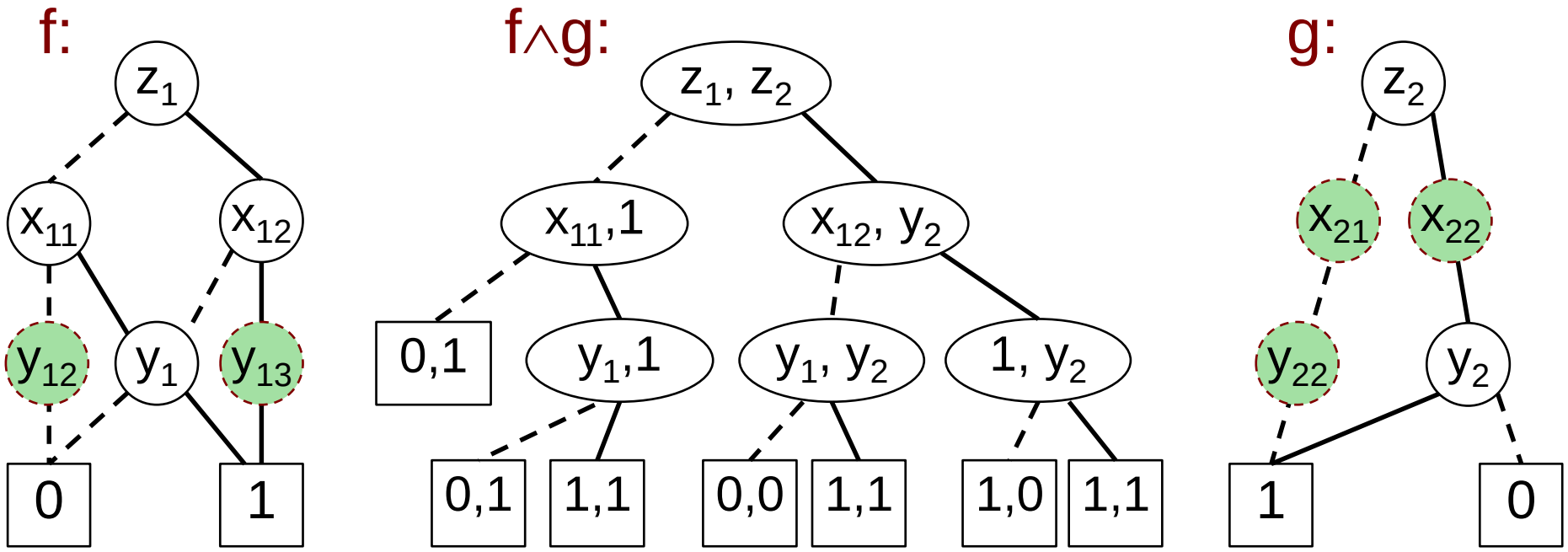


g:



Példa: Művelet elvégzése (f∧g)

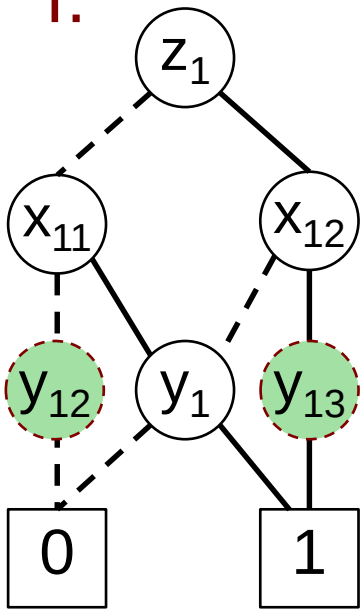
Az ROBBD alakok alapján készítsük el f∧g ROBBD-jét!



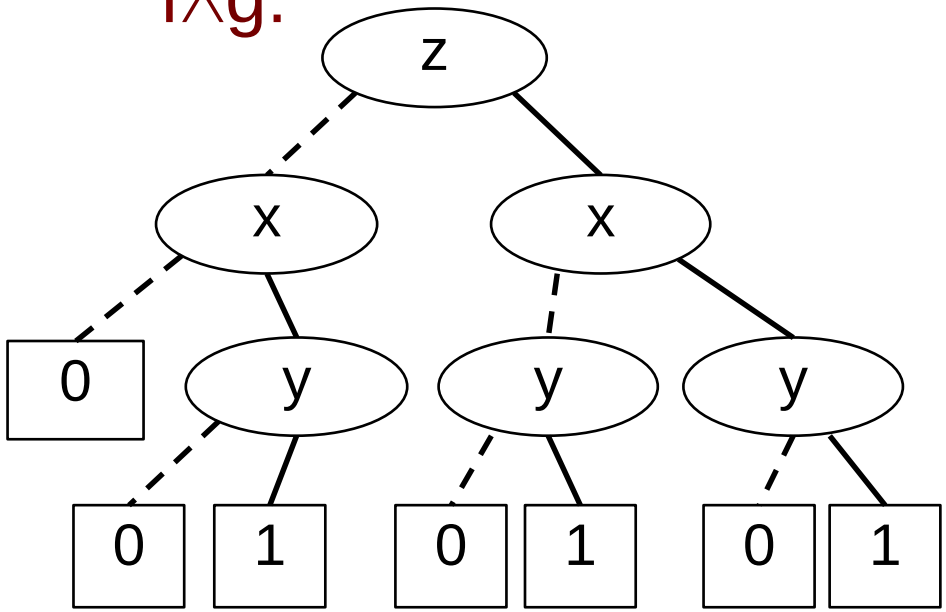
Példa: Művelet elvégzése (f∧g)

Az ROBBD alakok alapján készítsük el f∧g ROBBD-jét!

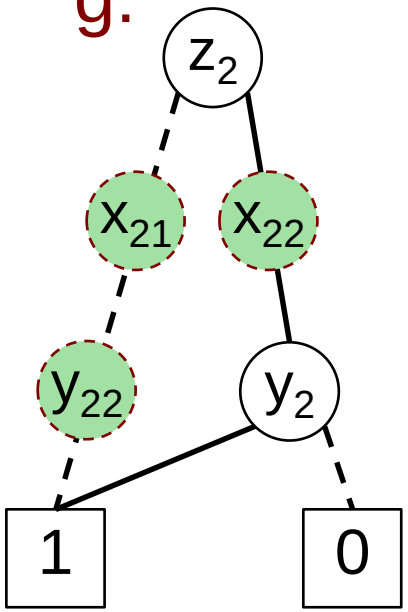
f:



f∧g:

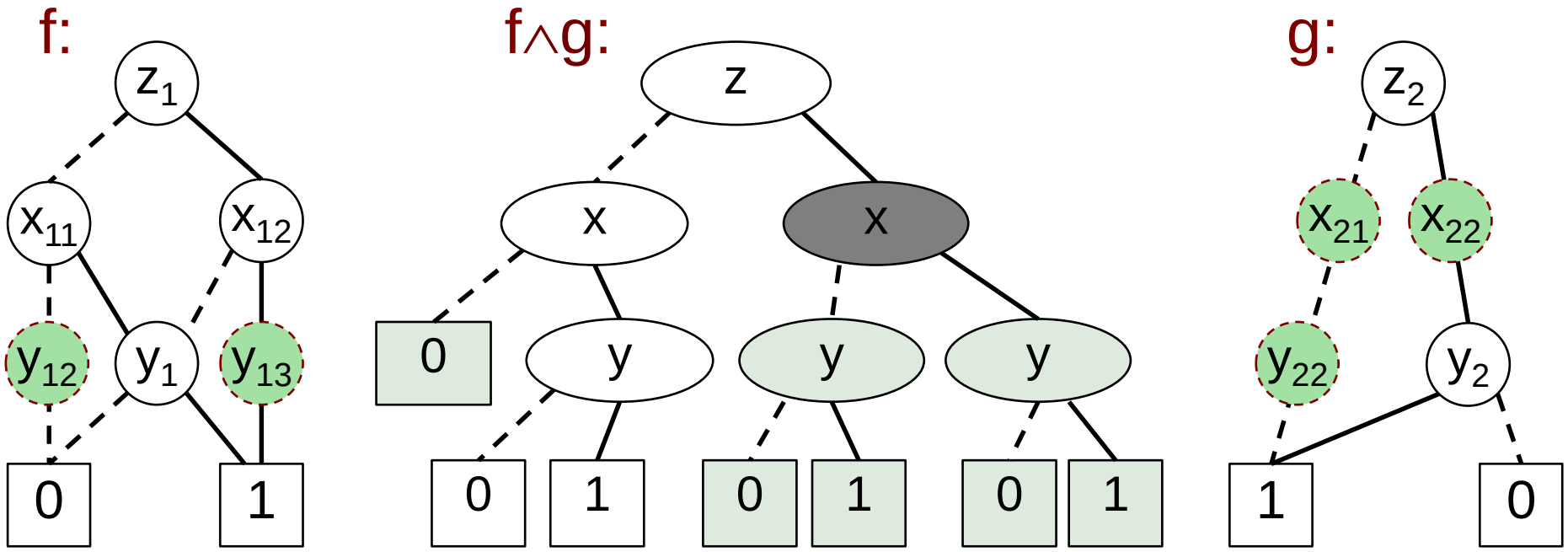


g:



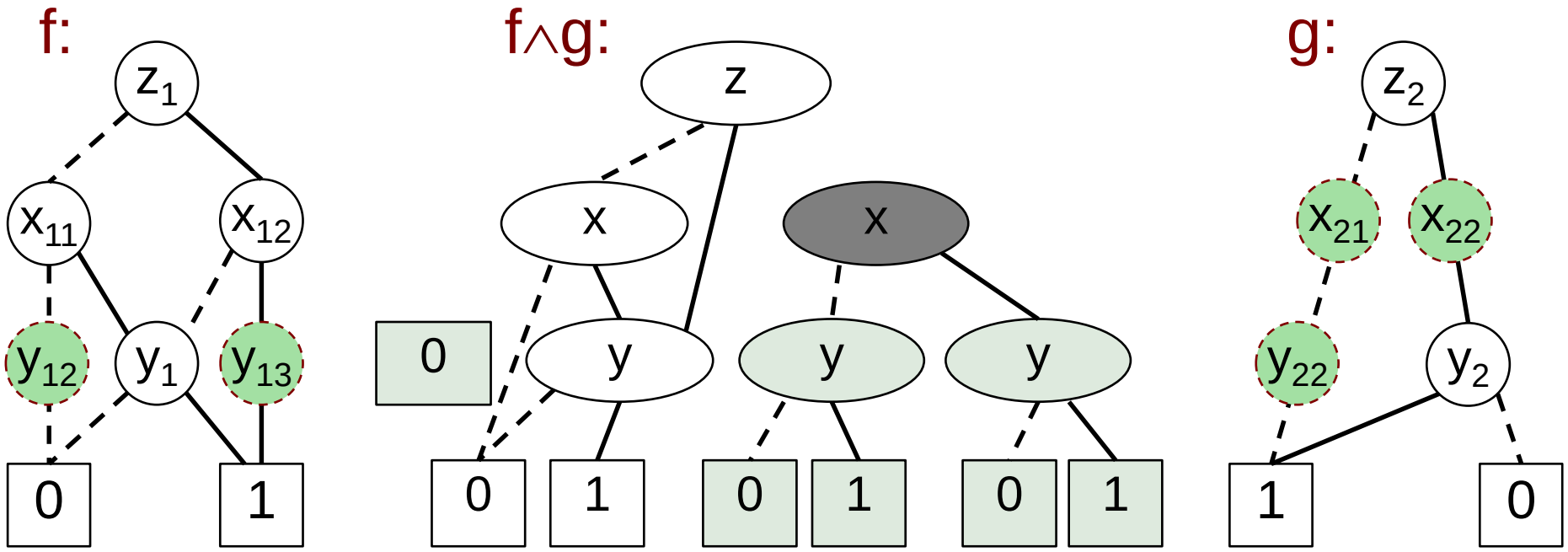
Példa: Művelet elvégzése (f∧g)

Az ROBBD alakok alapján készítsük el f∧g ROBBD-jét!



Példa: Művelet elvégzése (f∧g)

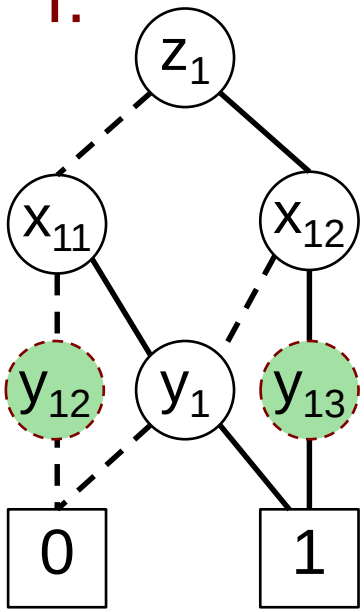
Az ROBBD alakok alapján készítsük el f∧g ROBBD-jét!



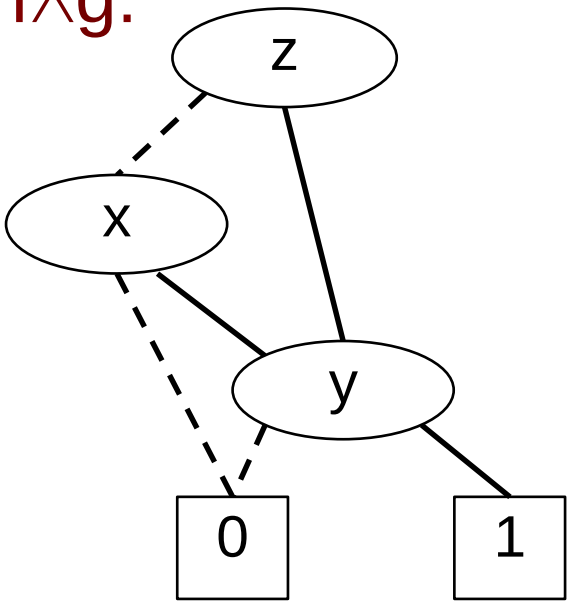
Példa: Művelet elvégzése (f∧g)

Az ROBBD alakok alapján készítsük el f∧g ROBBD-jét!

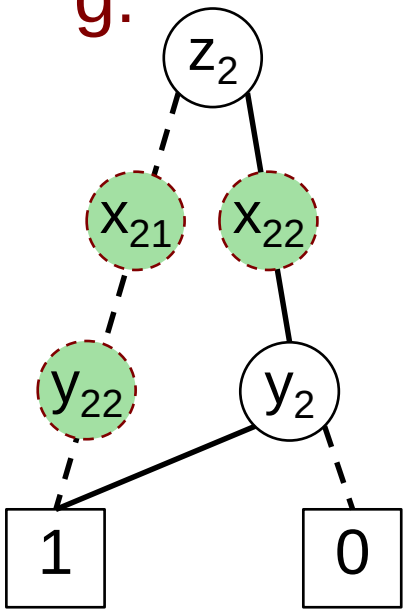
f:



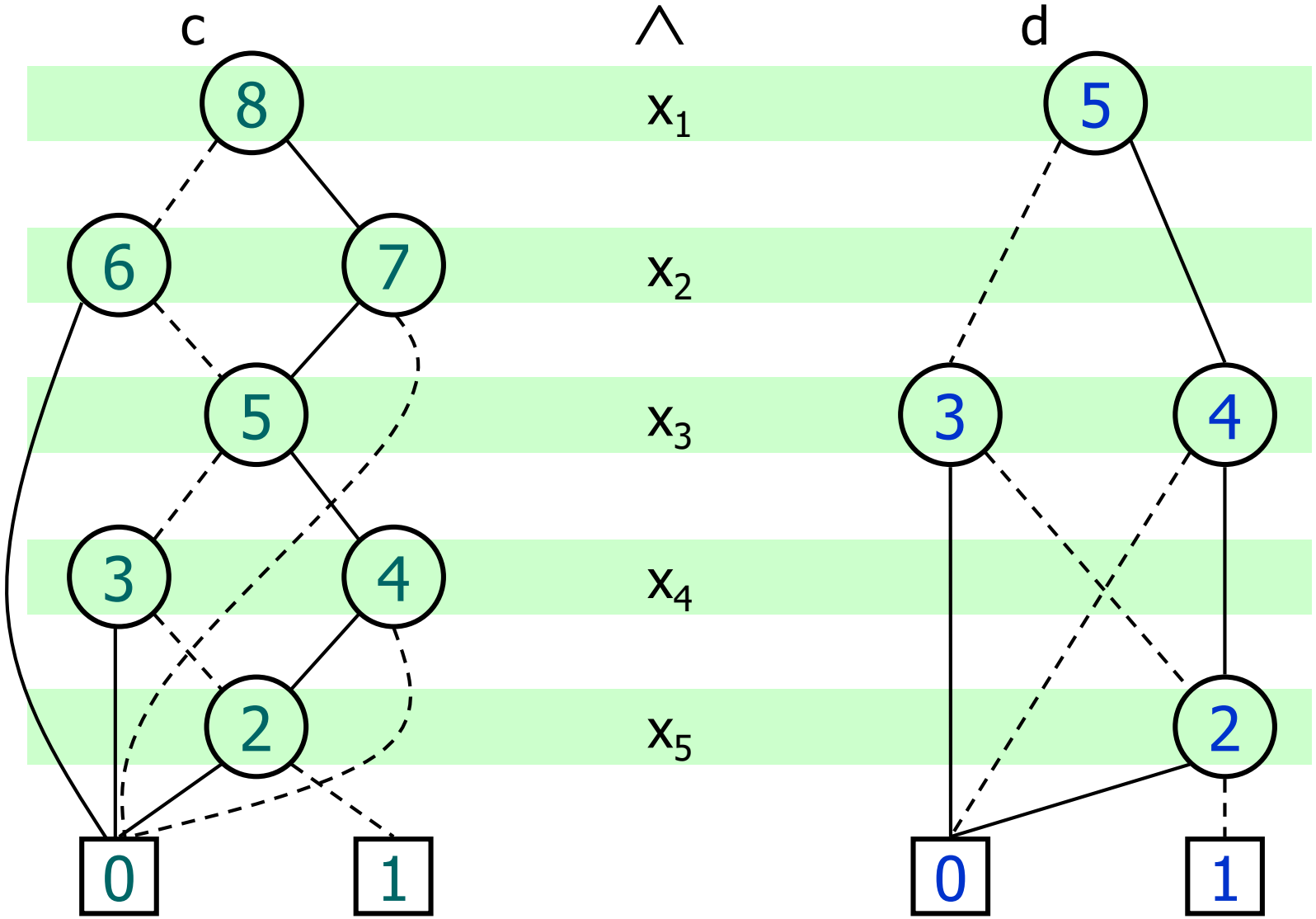
f∧g:



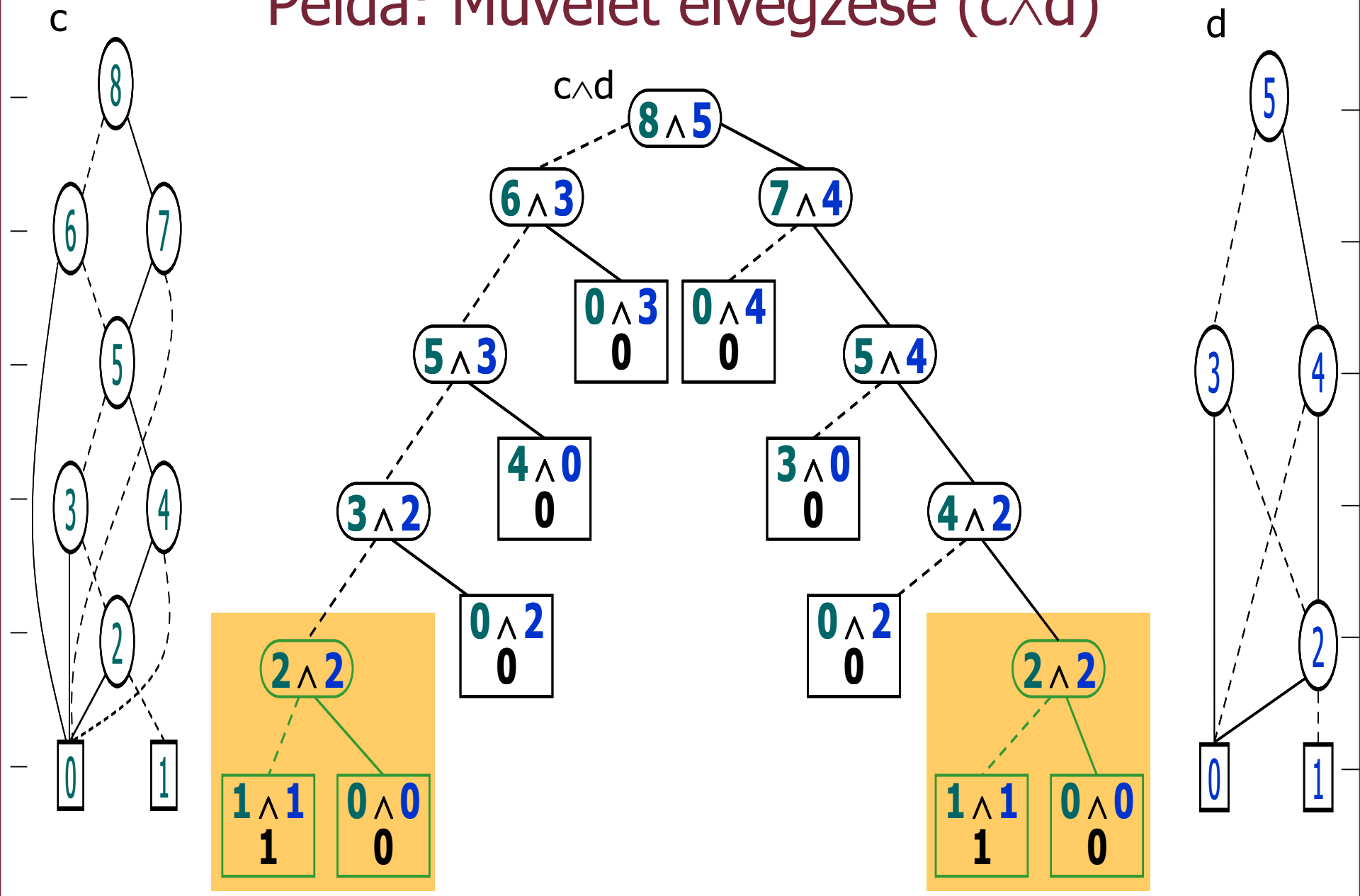
g:



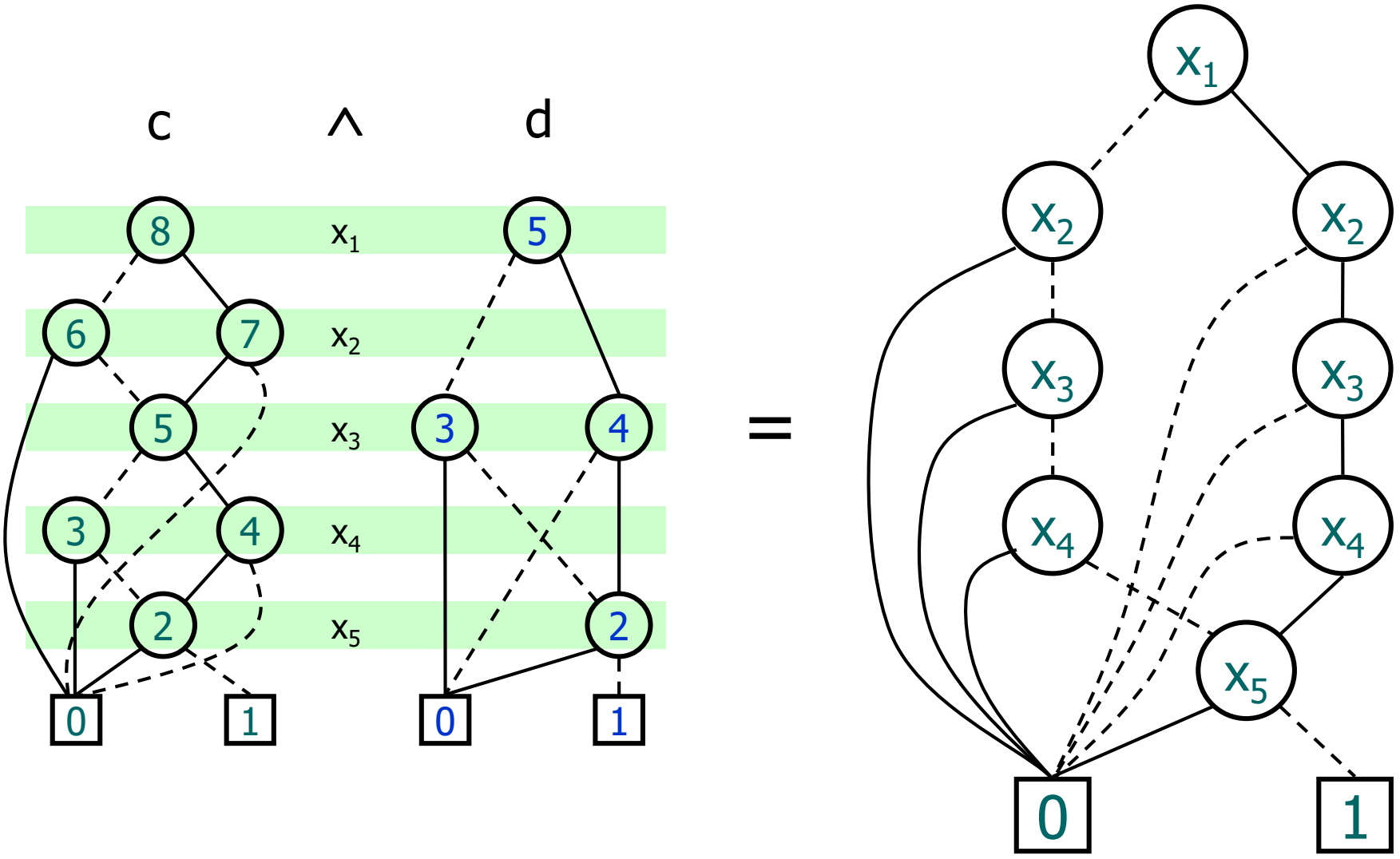
Példa: Művelet elvégzése (c∧d)



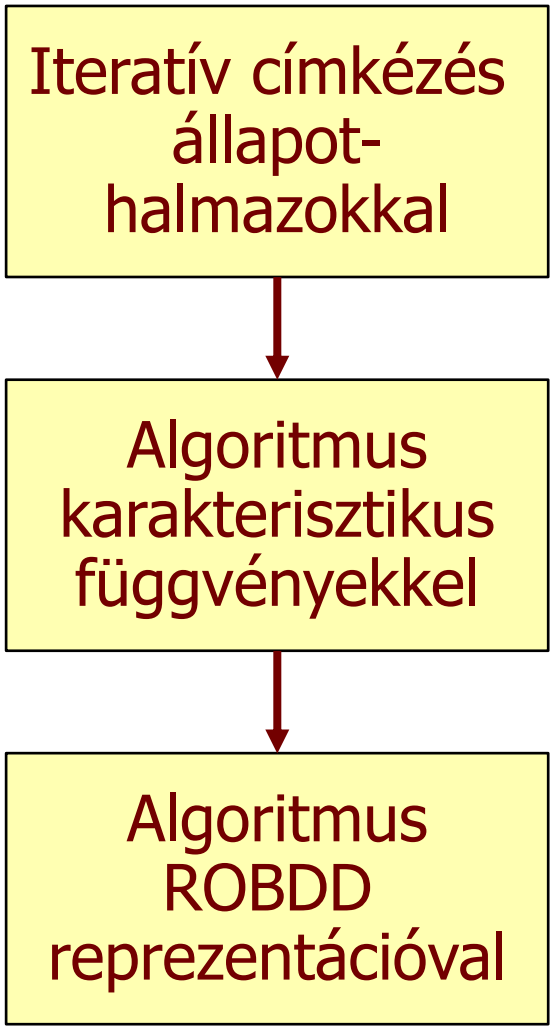
Példa: Művelet elvégzése (c∧d)



Példa: Művelet eredménye (c∧d)



Összefoglalás: Modellellenőrzés ROBDD-vel

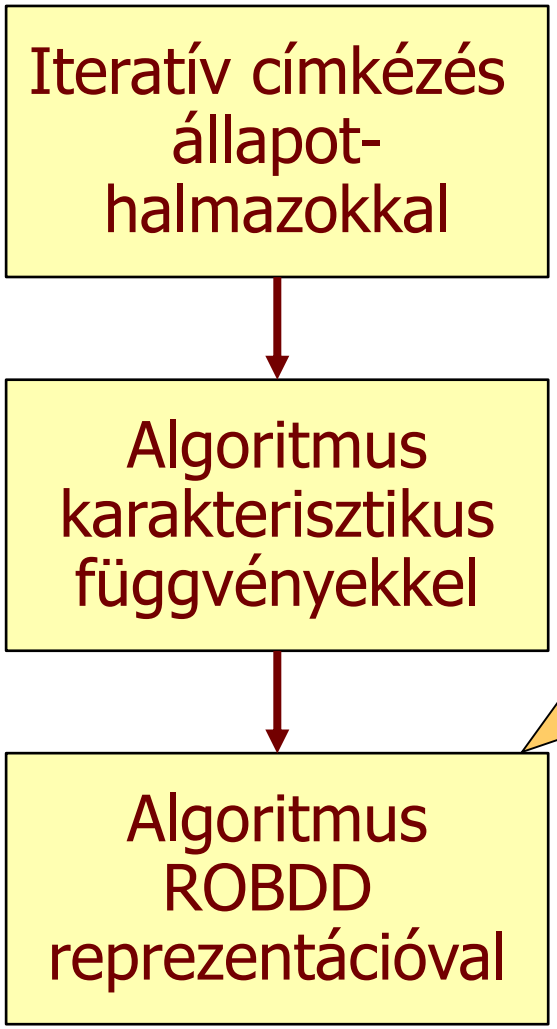


Halmaz
unió, metszet,
pre_E, pre_A

Boole függvény
$\vee$, $\wedge$,
C_{preE}, C_{preA}

ROBDD
konstrukció,
logikai
műveletek

Összefoglalás: Modellellenőrzés ROBDD-vel



ROBDD előnyök kihasználhatók:

- Kanonikus alak (függvények ekvivalenciája jól vizsgálható – lásd az iteráció vége)
- 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	$4{,}8 \cdot 10^{18}$	1.347

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

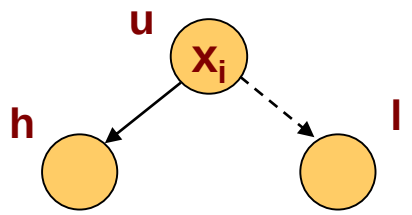
Az ROBDD gépi előállítása és műveletei (kiegészítő anyag)

Hasznos ötletek, példák:

- Adatstruktúra kialakítása
- Redundáns tárolás elkerülése (segédtáblázat)
- Rekurzív algoritmus kialakítása
- „Gyorsítótár” ismételt műveletek elkerülésére

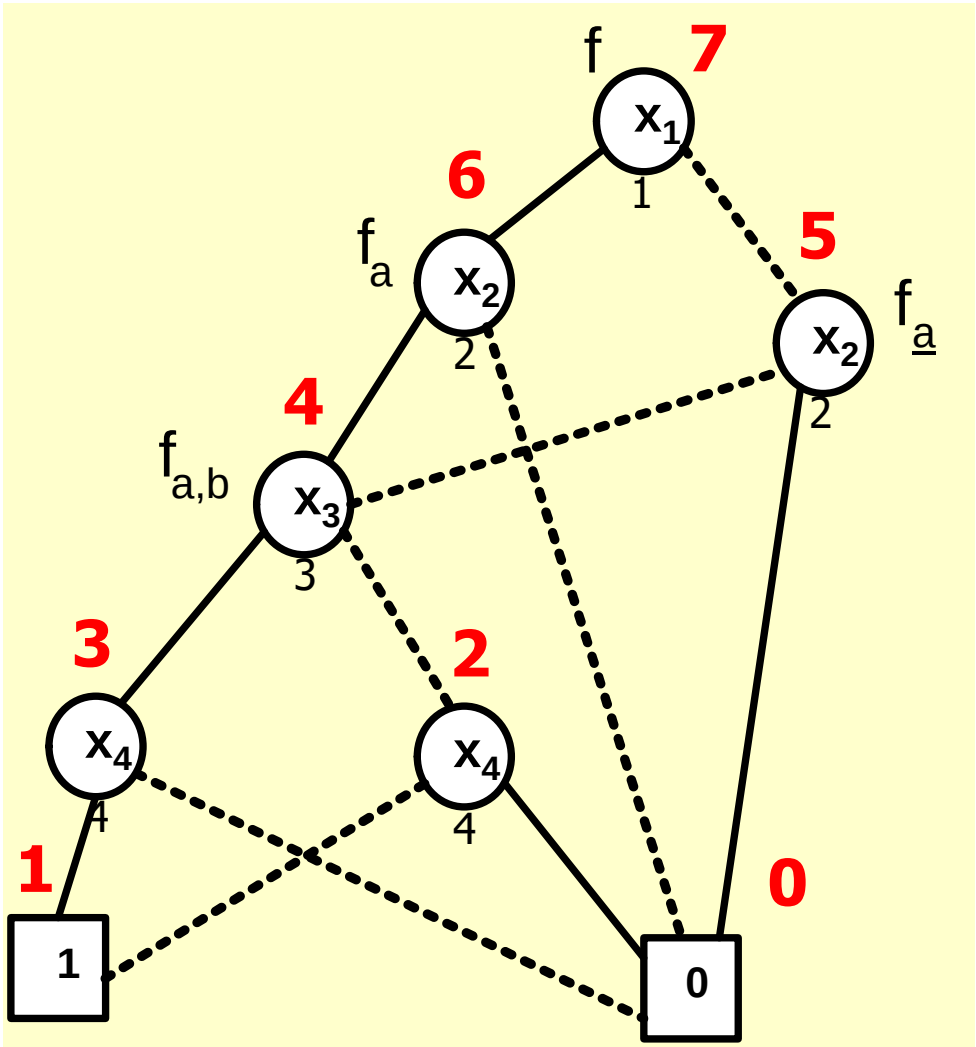
ROBDD gépi tárolása

- A ROBDD csomópontjait indexekkel azonosítjuk
- A ROBDD-t egy
  T: u → (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 segédtáblázatot 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 érték
  - 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 a változó index, l és h az 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
  - ⇒ 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) rekurzív 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 (minden változó behelyettesítve)

Rekurzív építés; Mk() ellenőrzi az izomorf részfákat

Gépi műveletek ROBDD-ken

- Boole operátorokat közvetlenül ROBDD-ken hajtjuk végre
  - A két függvény változói azonosak legyenek, azonos sorrendben
- Alap azonosság f, g függvényekre (itt op Boole operátor):
  1. $f \text{ op } g = (x \rightarrow f_x, f_{\underline{x}}) \text{ op } (x \rightarrow g_x, g_{\underline{x}}) = x \rightarrow (f_x \text{ op } g_x), (f_{\underline{x}} \text{ op } g_{\underline{x}})$
- További szabályok (redukálás miatt hiányzó változó):
  2. $f \text{ op } g = (x \rightarrow f_x, f_{\underline{x}}) \text{ op } g = x \rightarrow (f_x \text{ op } g), (f_{\underline{x}} \text{ op } g)$
  3. $f \text{ op } g = f \text{ op } (x \rightarrow g_x, g_{\underline{x}}) = x \rightarrow (f \text{ op } g_x), (f \text{ op } g_{\underline{x}})$

- 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

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,g){
  init(G);
  return App(op,f,g);
}

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;
}
```

Behelyettesítés ROBDD alakjának előállítása

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

Itt az u alatti 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