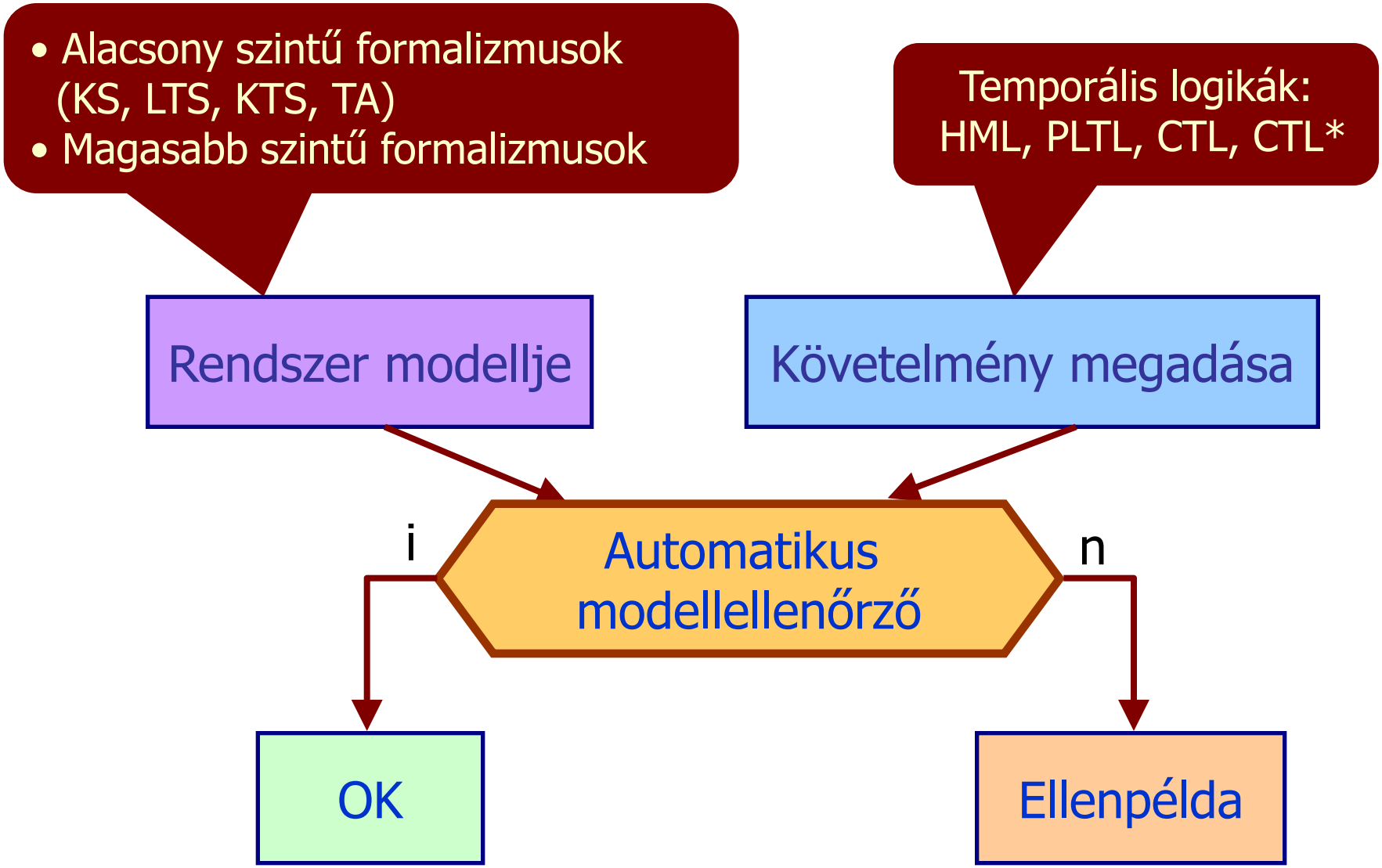
A modellellenőrzés hatékony technikái: Az állapottér kezelése

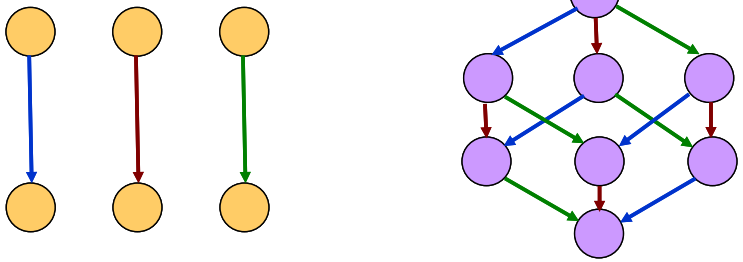
Majzik István
BME Méréstechnika és Információs Rendszerek Tanszék

Ismétlés: Mit szeretnénk elérni?



Ismétlés: A modellellenőrzés tanult technikái

- PLTL modell ellenőrzés:
  - Automata alapú megközelítés:
    - Szinkron szorzat automata konstruálása, elfogadó állapotok keresése
- CTL modell ellenőrzés:
  - Szemantika alapú módszer:
    - Állapotok iteratív címkézése (fixpont iteráció)
- Állapottér robbanás:
  - Egyszerűek az alapszintű modellek (állapot és átmenet)
  - Nagy méretű állapottér adódik (pl. elosztott rendszerek)



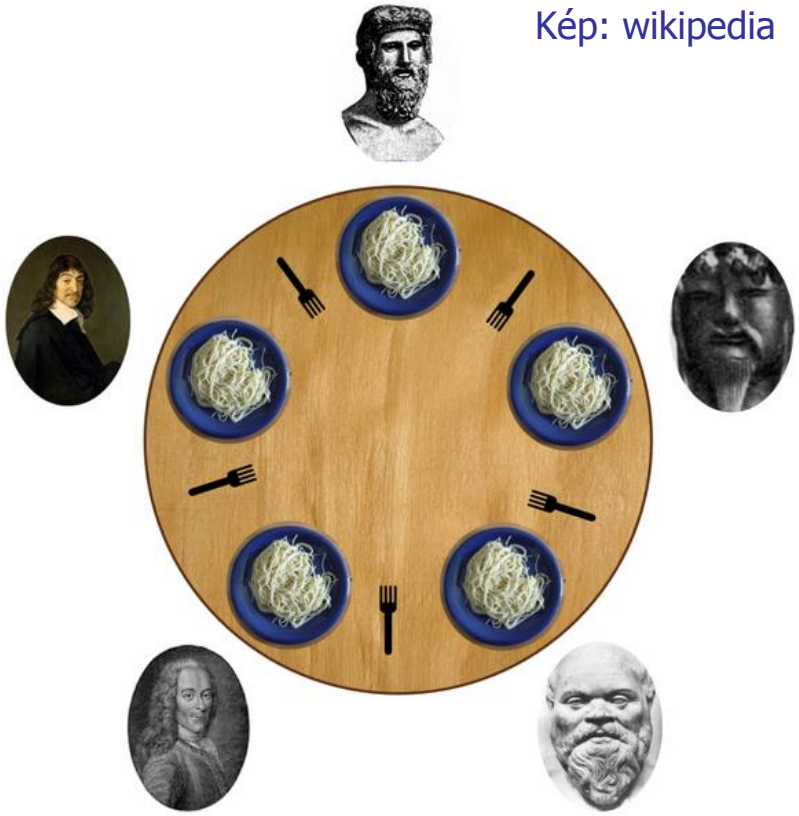
- Hogyan lehet nagy méretű modelleket ellenőrizni?
  - Ígéret: CTL modell ellenőrzés: 10^{20} állapot (egyes esetekben $>10^{100}$ állapot)

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	$> 10^{200}$
…	…

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



Kép: wikipedia

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

Állapottér kezelési technikák: Előzetes áttekintés

- CTL modellellenőrzés: Szimbolikus technika
  - (Címkézett) állapothalmazok tárolása és manipulálása helyett Boole függvényeken végzett műveletek
  - A Boole függvények hatékony tárolása ROBDD alkalmazásával
  - Első ilyen modellellenőrzők: SMV, nuSMV
- Invariánsok modellellenőrzése: Korlátos modellellenőrzés
  - Logikai függvények igazságának keresése SAT technikával
  - Adott mélységig folytatható modell ellenőrzés: Korlátos hosszúságú ellenpéldák keresése
- LTL modellellenőrzés: Részleges rendezés
  - A lehetséges útvonalak közül reprezentatív útvonalak kiválasztása az adott követelmény ellenőrzéséhez
  - Bemutatott technika: SPIN modellellenőrző alapja
- Általános módszer problémaméret csökkentésre: Absztrakció

Szimbolikus modellellenőrzés a CTL esetén

Ismétlés: CTL modellellenőrzés állapot címkézéssel

- Az iteráció halmazműveletekkel történik
  - Kezdőhalmaz: Előző lépések alapján
  - Inkrementális bővítés a Z halmazhoz tartozó (már címkézett) állapotok megelőző állapotain:

    $pre_E(Z)=\{s \in S \mid$ 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: $X_0 = \{s \mid Q \in L(s)\}$
  - Iteráció: $X_{i+1} = X_i \cup (pre_E(X_i) \cap \{s \mid P \in L(s)\})$

Eddig címkézettek és ..

.. megelőző állapotaik közül amelyek …

… P-vel címkézettek

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

Ismétlés: CTL modellellenőrzés fixpont kereséssel

- Sat(EF p) számítása: lfp $\tau(z)$
  - ahol $\tau(z)$= Sat(p) $\cup$ $pre_E(z)$
  - ahol $pre_E(z)$={s | $\exists$t: (s,s')$\in$R és s'$\in$z}

- Kleene tétele alapján lfp $\tau(z)$ számítása:
  - $z_0 = \emptyset$
  - $z_1 = \tau(z_0) = \tau(\emptyset)$ = Sat(p) $\cup$ $pre_E(\emptyset)$
  - …
  - $z_{i+1} = \tau(z_i)$= Sat(p) $\cup$ $pre_E(z_i)$
  - …
  - folytatás amíg $z_{i+1} = z_i$ és itt z_i = lfp $\tau(z)$ = Sat(EF p)

A szimbolikus modellellenőrzés alapötlete

- Állapothalmazok tárolása állapot felsorolás helyett logikai függvények formájában
  - Állapotok „kódolása” Boole logikai vektorokkal
    - S állapottér kódolásához $n=\lceil \log_2|S| \rceil$ bit elég, azaz ha teljesül $2^n \geq |S|$
  - Állapothalmaz „kódolása” n-változós Boole függvényekkel
    - Akkor és csak akkor 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
- Elméleti alap: Stone-tétel
  - Minden véges Boole-algebra izomorf egy véges S halmaz részhalmazainak algebrájával
  - $(2^S, \cap, \cup, \varnothing, S)$ izomorf $(F_n, \vee, \wedge, 0, 1)$
    - F_n az n-változós logikai függvények {0,1} felett
    - 2^{2^n} féle n-változós logikai függvény van, $2^{|S|}$ a részhalmazok száma

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

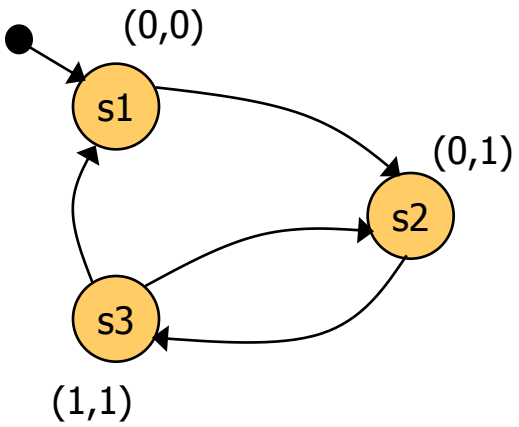
  $C_Y(x_1, x_2, \ldots, x_n)$ a.cs.a. legyen igaz egy $(u_1, u_2, \ldots, u_n)$ behelyettesítésre, ha $(u_1,u_2,\ldots, u_n) \in Y$

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

- Állapothalmazokra általában:

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

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:

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

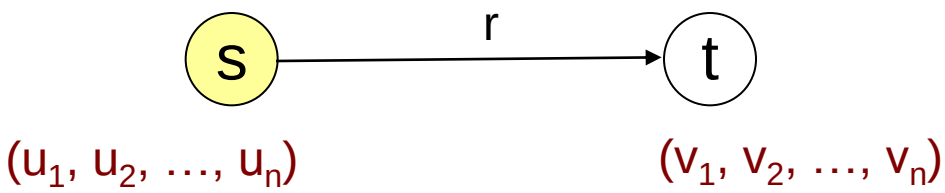
Állapothalmaz karakterisztikus függvénye:

{s1,s2} állapothalmaz:

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

Karakterisztikus függvények (folytatás)

- Állapotátmenetekre: C_r



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

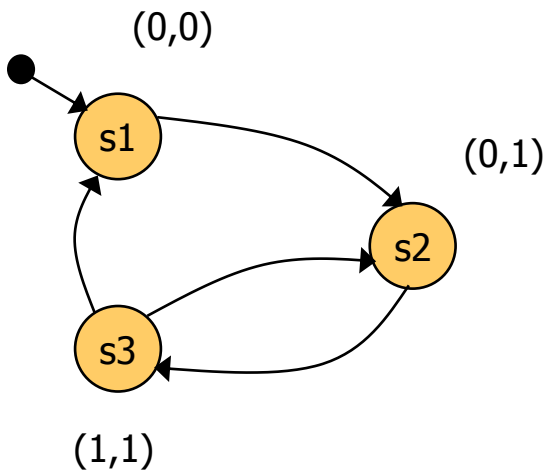
Karakterisztikus függvény $C_r(x_1, x_2, \ldots, x_n, x'_1, x'_2, \ldots, x'_n)$ alakban;
„vesszős” változók jelzik a cél állapotot

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

C_r konstruálása:

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

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:

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

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

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

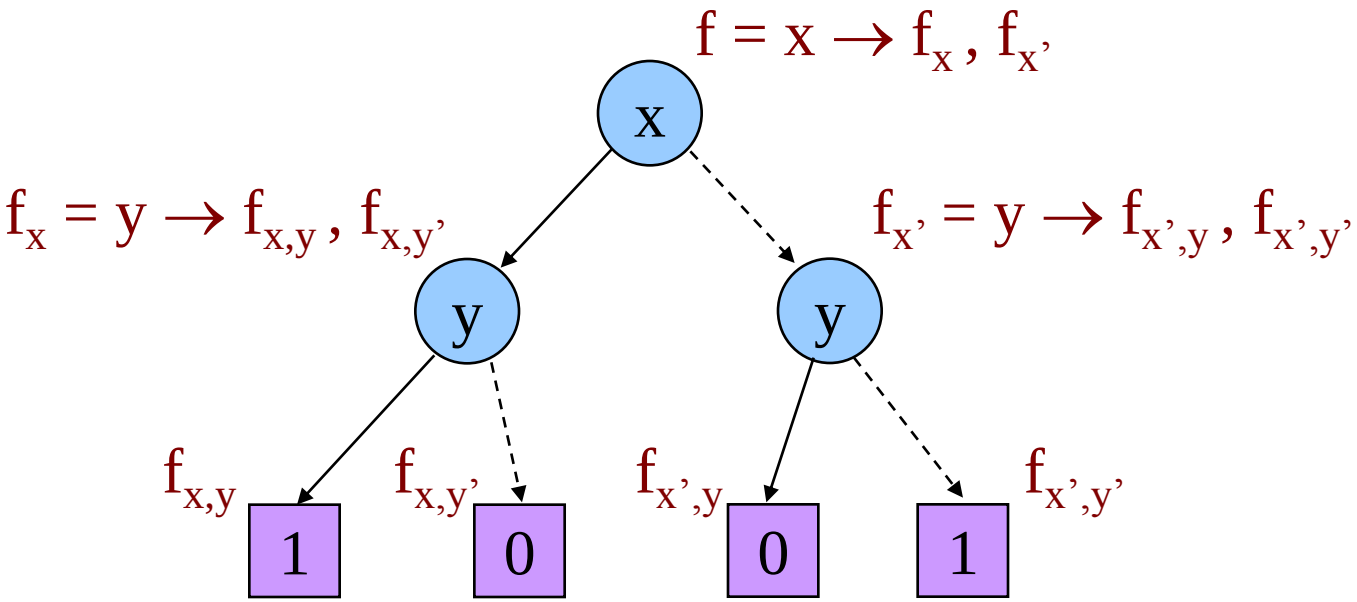
  ahol $\exists_x C = C[1/x] \vee C[0/x]$ „egzisztenciális absztrakció”

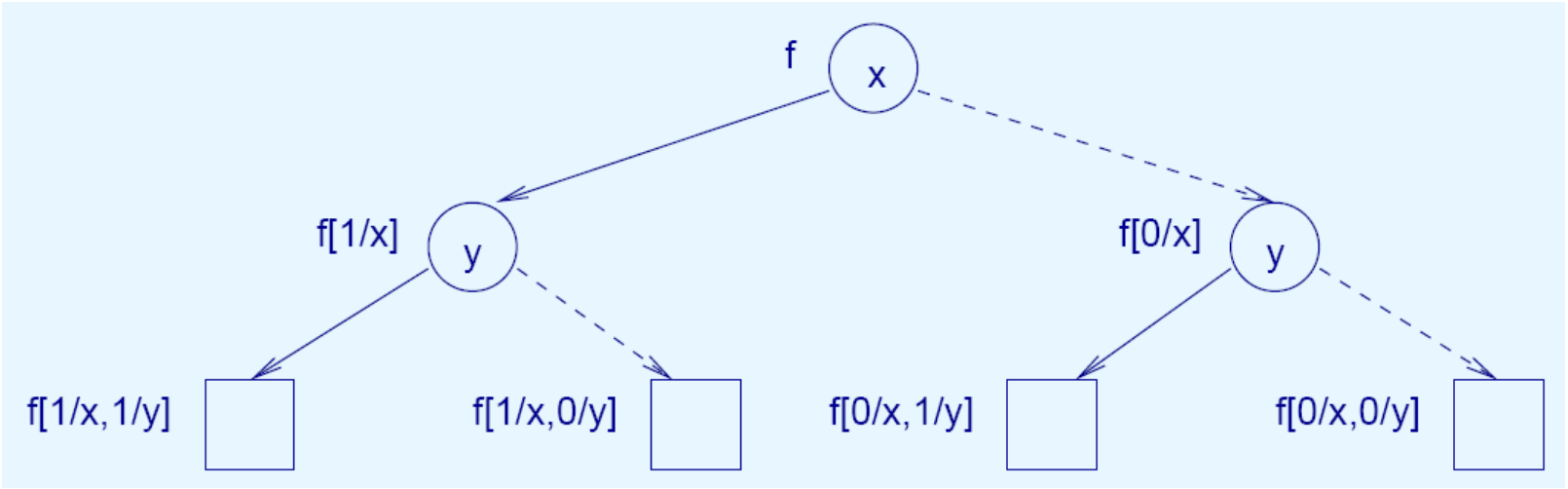
- Eddig: Modellellenőrzés állapothalmaz műveletekkel: visszavezetve logikai függvényeken végzett műveletekre
  - Halmazok uniója: Függvények $\vee$ kapcsolata
  - Halmazok metszete: Függvények $\wedge$ kapcsolata
  - $pre_E(Z)$ képzése: Összetett művelet (egzisztenciális absztrakció)

Logikai függvények reprezentációja

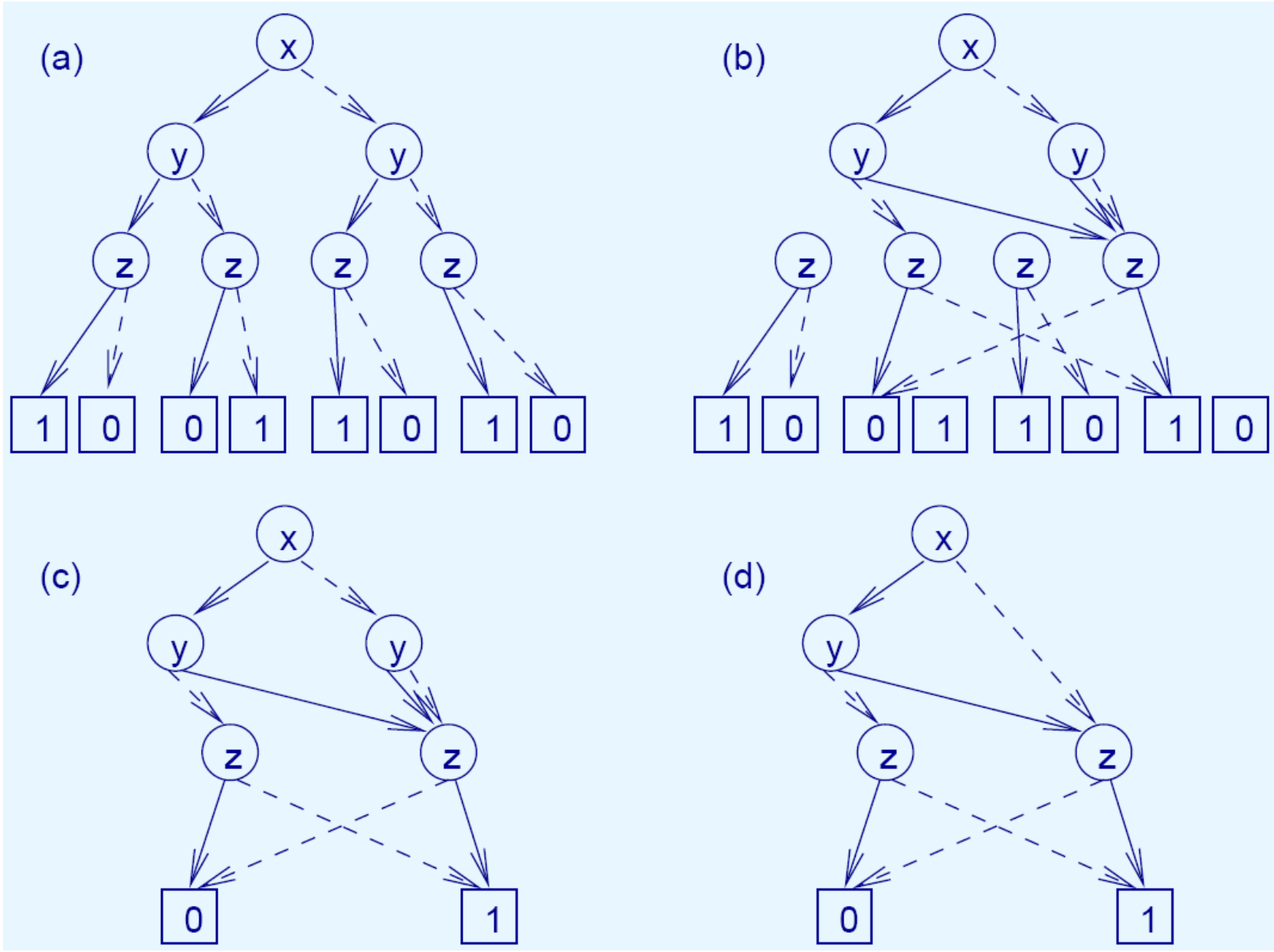
- Kanonikus forma: ROBDD
  - Redukált, rendezett, bináris döntési diagram
- Lépések: Bináris döntési fa, BDD, OBDD, ROBDD
  - Bináris döntési fa: Shannon-felbontás
    - „if-then-else” operátor: $x \rightarrow x_1, x_0 = (x \wedge x_1) \vee (\neg x \wedge x_0)$
    - felbontás: $f = x \rightarrow f[1/x], f[0/x]$
    - más jelölés: $f(x,y,z,\ldots) = x \rightarrow f_x(y,z,\ldots), f_{x'}(y,z,\ldots)$
  - BDD: azonos részfák összevonva
  - OBDD: minden ágon azonos változó sorrendezés
  - ROBDD: szükségtelen csomópontok redukálása
    - Ha mindkét ág ugyanahhoz a csomóponthoz vezet

Példa: Bináris döntési fa f(x,y) függvényre



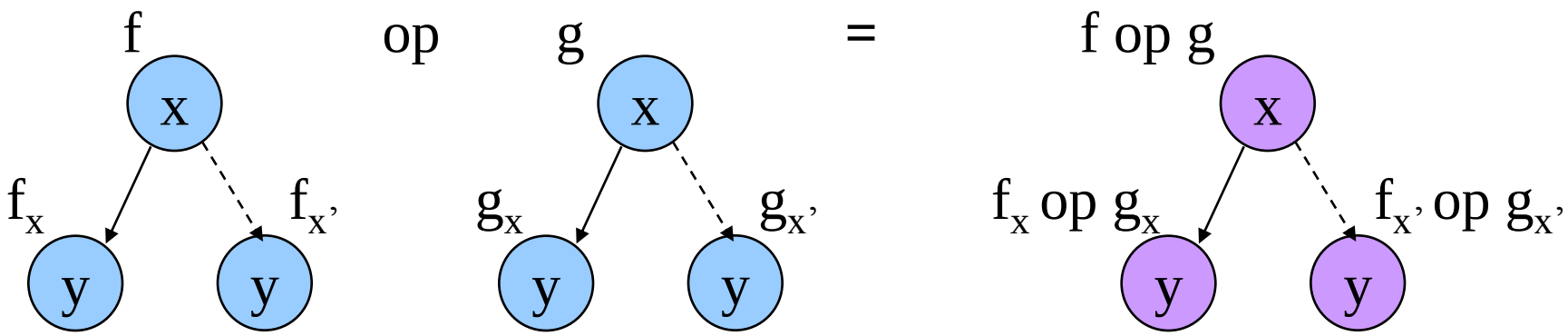


Bináris döntési fa redukálása



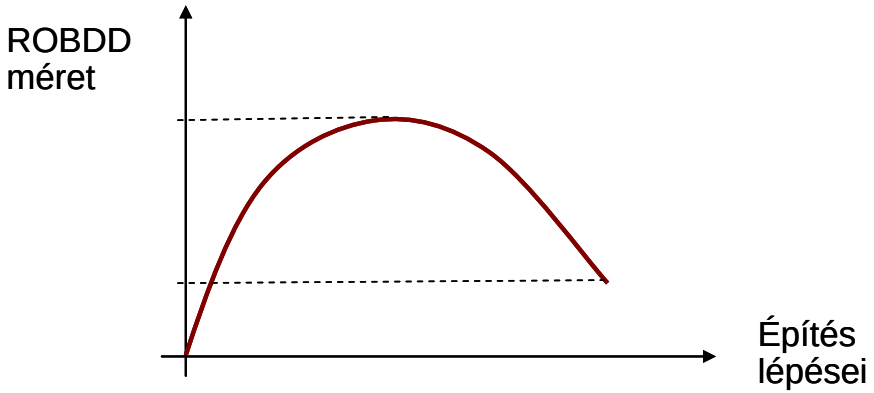
ROBDD használat

- Generálás: ld. jegyzet (+ Formális módszerek tárgy)
  - Rekurzív felbontás és azonos részfák figyelése
- Műveletek ROBDD által reprezentált fv-ekkel:
  - f op g = $(x \rightarrow f_x, f_{x'})$ op $(x \rightarrow g_x, g_{x'}) = x \rightarrow (f_x \text{ op } g_x), (f_{x'} \text{ op } g_{x'})$
  - f op g = $(x \rightarrow f_x, f_{x'})$ op g $= x \rightarrow (f_x \text{ op } g), (f_{x'} \text{ op } g)$
  - f op g = f op $(x \rightarrow g_x, g_{x'}) = x \rightarrow (f \text{ op } g_x), (f \text{ op } g_{x'})$



ROBDD mérete

- Tömör ábrázolásmód, pl. étkező filozófusok problémája:
  - 16 filozófus: $4{,}7 \cdot 10^{10}$ állapot, 747 ROBDD csomópont
  - 28 filozófus: $4{,}8 \cdot 10^{18}$ állapot, 1347 ROBDD csomópont
    itt 63 bites állapottér címzés helyett ~21kB elég!
- Fontos a helyes sorrend megválasztása
  - Más sorrend nagyságrendbeli méretkülönbséget is jelenthet
  - Optimális sorrend „próba-szerencse” alapon derülhet ki
- Tárigény:
  - Ha egy konkurens rendszer modellje alapján folyamatosan építjük a ROBDD-t, akkor az építés közben sok ideiglenes csomópontot kell tárolnunk, amit később lehet redukálni



ROBDD és modellellenőrzés (összefoglalás)

- Modellellenőrzés: Állapothalmazokon műveletek
  - Iteratív Sat(p) számítás
- Halmazműveletek leképzése karakterisztikus függvények műveleteire
  - „Állapotok kódolása”: Boole-függvények
- Karakterisztikus függvény műveleteinek leképzése ROBDD-ken végzett műveletekre
  - Közvetlenül az ROBDD reprezentáción

Korlátos modellellenőrzés (Bounded Model Checking)

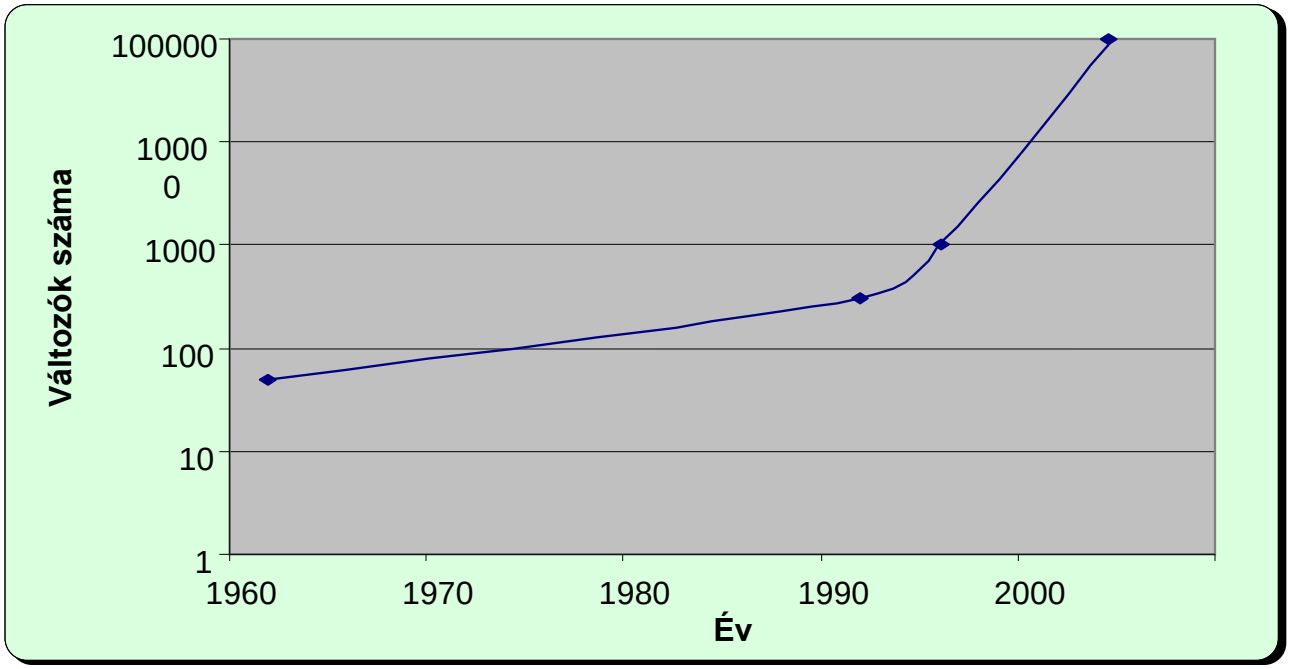
Majzik István
BME Méréstechnika és Információs Rendszerek Tanszék

Állapottér kezelési technikák: Előzetes áttekintés

- CTL modellellenőrzés: Szimbolikus technika
  - (Címkézett) állapothalmazok tárolása és manipulálása helyett Boole függvényeken végzett műveletek
  - A Boole függvények hatékony tárolása ROBDD alkalmazásával
  - Első ilyen modellellenőrzők: SMV, nuSMV
- Invariánsok modellellenőrzése: Korlátos modellellenőrzés
  - Logikai függvények igazságának keresése SAT technikával
  - Adott mélységig folytatható modell ellenőrzés: Korlátos hosszúságú ellenpéldák keresése
- LTL modellellenőrzés: Részleges rendezés
  - A lehetséges útvonalak közül reprezentatív útvonalak kiválasztása az adott követelmény ellenőrzéséhez
  - Bemutatott technika: SPIN modellellenőrző alapja
- Általános módszer problémaméret csökkentésre: Absztrakció

Lehetőség: A SAT megoldók fejlődése

- SAT megoldó:
  - Adott logikai függvényhez olyan változó érték behelyettesítést keres, amelyekkel a függvény értéke igaz
  - Példa: $f(x_1,x_2,x_3)=x_1 \wedge x_2 \wedge \neg x_3$ függvény esetén (1,1,0) bitvektor
- NP-teljes probléma, de hatékony algoritmusok léteznek
  - Pl. zChaff, MiniSAT, …



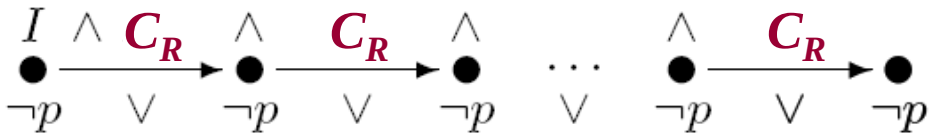
Célkitűzés

- A modellellenőrzési probléma leképzése logikai függvény helyettesítési értékének keresésére
  - Modell + temporális követelmény együttes megadása
    - Tipikusan invariáns követelményekhez!
- 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 egy ellenpéldát jelöl ki
    - Az ellenpélda használható a hibakereséshez
    - Invariáns tulajdonságok esetén jól használható módszer

Informális bevezetés

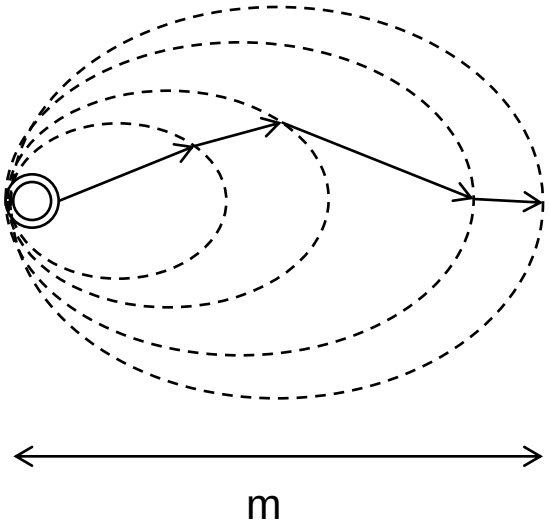
- Hogyan képezzük az állapotteret?
  - Kiindulás a kezdőállapotból: I(s) karakterisztikus függvénnyel megadható
  - „Kibontás”: Lehetséges továbblépések az állapotátmeneti reláció mentén
    - Állapotátmeneti reláció (hová léphetünk tovább): $C_R(s,s')$ karakterisztikus függvénnyel
    - Ha s-ben vagyunk, $C_R(s,s')$ alkalmazása adja meg a lehetséges s’ rákövetkezőket, majd s’ esetén $C_R(s',s'')$ alkalmazása adja meg a lehetséges s’’ rákövetkezőket …
    - Egyszerűbb jelölés: Vesszők használata helyett felső index: $C_R(s^0,s^1)$ majd $C_R(s^1,s^2)$ …
- Hogyan adjuk meg a követelményt?
  - Invariáns: Minden állapotra előírt kritérium: általánosan egy p(s) predikátum
- Az ellenpéldát kijelölő logikai függvény részei (konjunkcióval):
  - Kezdőállapotból indulunk: I(s)
  - „Lépegetünk” (kibontunk) az állapotátmeneti reláció mentén: $C_R(s,s')$
  - Ellenpélda (valahol p(s) nem teljesül): ¬p(s) diszjunkció a bejárt állapotokra

Ezt a függvényt igazzá tevő behelyettesítés adja az ellenpéldát!



Korlátos modellellenőrzés

- Az útvonalak hosszát korlátozva végezzük az ellenőrzést
  - Teljes állapottér bejárás csak m útvonalhossz korlátig
  - Egyes esetekben van „átmérője” az állapottérnek: ez a leghosszabb útvonal, ami bejárható
- Az m korlát becslése
  - Intuíció a probléma méretéről
  - WCET becslés használata



Jelölések

- M=(S,R,L) Kripke-struktúra, $R \subseteq S \times S$
- Logikai függvények:
  - $C_s(s)$ állapotok „kódolása” n hosszú bitvektorokkal
  - I(s) a kezdőállapotok predikátuma (több is lehet)
  - $C_R(s,s')$ állapotátmenetek 2n változós karakterisztikus függvénye
    - „Vesszős” állapotváltozók a cél állapot számára
  - P(s) invariáns: Állapotok halmazának n változós karakterisztikus függvénye (L alapján)
  - Útvonal: k hosszú útvonal (k+1)n változóval

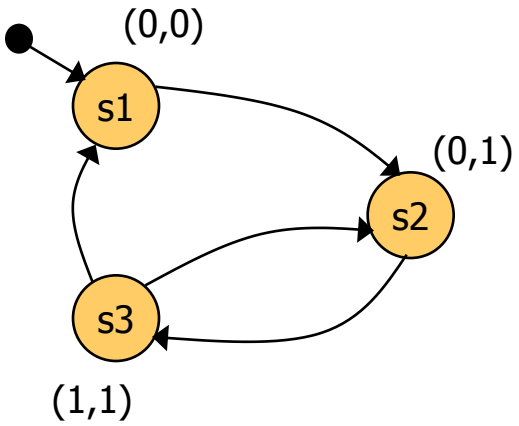
$$path(s^0, s^1, \ldots, s^k) = \bigwedge_{0 \le i < k} C_R(s^i, s^{i+1})$$

Vesszős változók helyett felső index

  - Adott végpontok között k hosszúságú útvonal létezése

$$path_k(s^0, s^k) = \exists_{s_1, \ldots, s_{k-1}} path(s^0, s^1, \ldots, s^k)$$

Példa: A modell leképzése logikai függvénybe

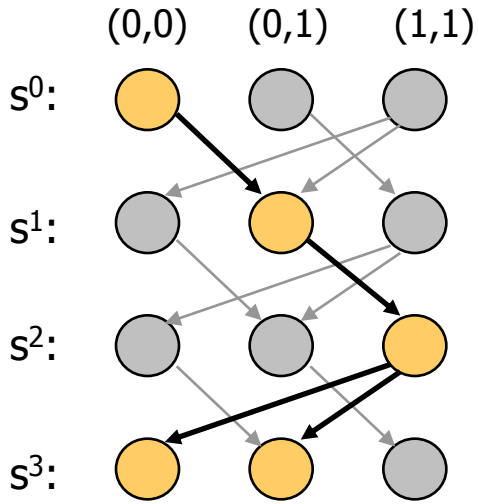


Kezdőállapot predikátum:

$I(x,y) = (\neg x \wedge \neg y)$

Állapotátmeneti reláció:

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



3 lépéses kihajtogatás a kezdőállapotból:

$$I(x^0,y^0) \wedge path(s^0,s^1,s^2,s^3) =$$
$$= I(x^0,y^0) \wedge$$
$$C_R(x^0,y^0, x^1,y^1) \wedge$$
$$C_R(x^1,y^1, x^2,y^2) \wedge$$
$$C_R(x^2,y^2, x^3,y^3)$$

A probléma formalizálása

- Bizonyítandó P(s) invariáns: Minden útvonal, ami a kezdőállapotból indul, olyan állapotba 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 függvény igaz értékéhez tartozó behelyettesítést ad 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):

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

Az algoritmus elemei

- Iteráció: i=0,1,2,... az útvonalak hosszára
- Ciklusmentes utakat vizsgálunk: *lfpath*

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

- Megállási feltétel az iteráció során (i-1-ről i-re lépéskor):
  - Nincs i hosszú ciklusmentes út kezdőállapotból, azaz nem lehet igaz

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

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

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

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

Az algoritmus

$i = 0$

while True do

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

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

    then return True

  if 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

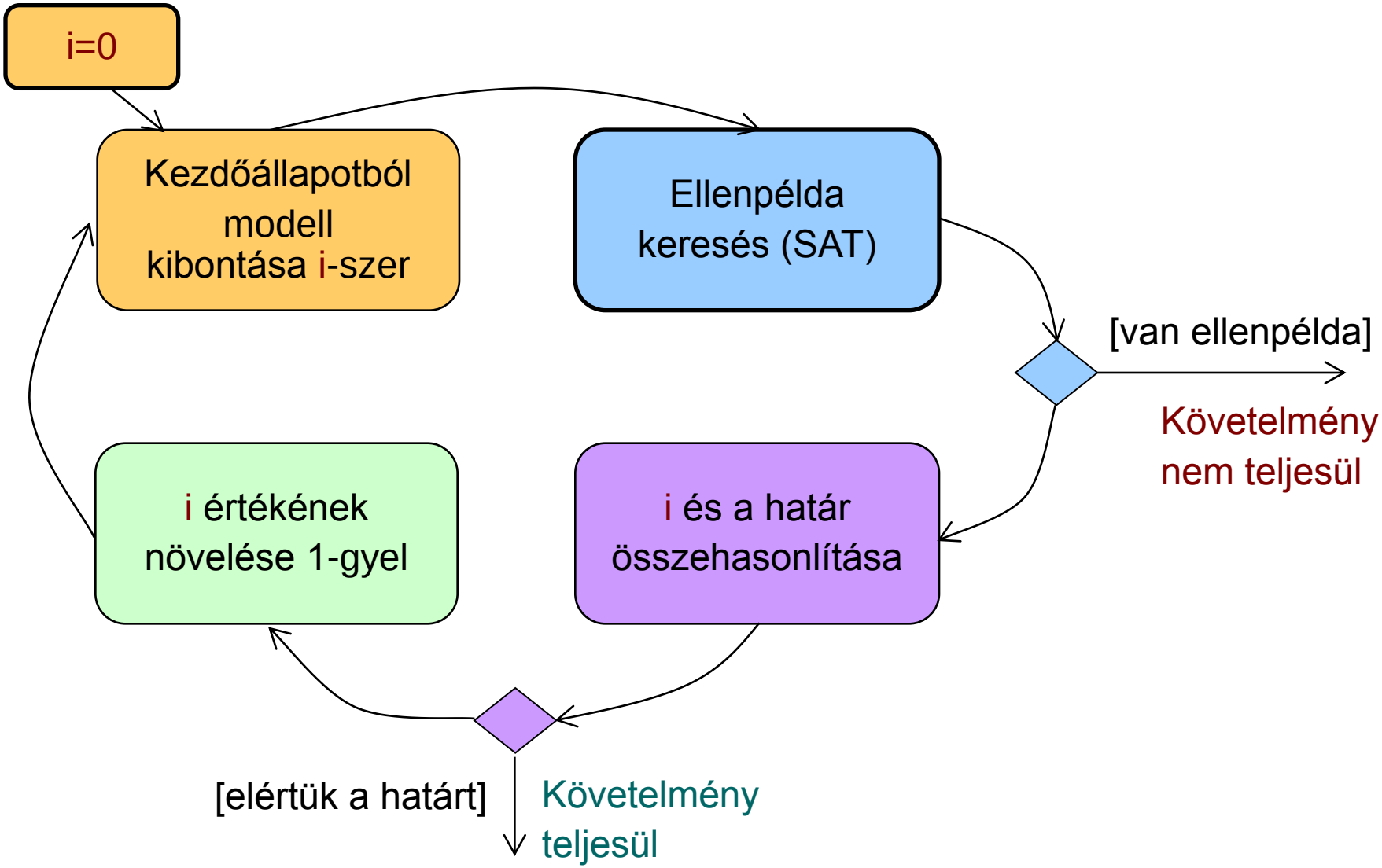
Nincs már i hosszú ciklusmentes út kezdőállapotból

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

Eredmény: Modellellenőrzés iterációval



Az algoritmus első 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
- Nem garantált, hogy az eredményül kapott ellenpélda (útvonal) minimális hosszúságú
  - Nem 0-ról kezdtük az iterációt; ha k nagy, akkor túllőhetünk a célon
  - Itt k kezdőértékére 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 már korábbi iteráció ellenpéldaként adná)

Az első finomított algoritmus

$i = k$

Kezdő érték

while True do

Van i hosszú útvonal kezdőállapotból „rossz” állapoton át

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

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

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

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

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

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

then return True

$i = i + 1$

end

Az algoritmus második finomítása

- Eddig az iteráció hosszát a leghosszabb ciklusmentes útvonal határozta meg az állapottérben
  - (Az állapottér hamarabb bejárható, ha) az állapotpárok között a legrövidebb útvonalat keressük: *shpath*

$$shpath(s^0, s^1, ..., s^k) = path(s^0, s^1, ..., s^k) \wedge \neg(\bigvee_{0 \le i < k} path_i(s^0, s^k))$$

  - Az algoritmus átírása: *lfpath* helyett *shpath*
- Hátrány: Sok egzisztenciális absztrakció szükséges ($path_i$)
  - Speciális algoritmusok szükségesek (ún. kvantor eliminálás)
- Iterációk száma: A minimális érték a következők közül:
  - Előre haladó átmérő: Leghosszabb az állapotpárokat összekötő legrövidebb útvonalak közül, a kezdőállapotból indulva, nem kezdőállapotokon át
  - Visszafelé haladó átmérő: Leghosszabb az állapotpárokat összekötő legrövidebb útvonalak közül, olyan állapotból, ahol P nem igaz, olyan állapotokon át, ahol P igaz

A második finomított algoritmus

$i = k$

while True do

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

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

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

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

then return True

$i = i + 1$

end

Van i hosszú útvonal kezdőállapotból „rossz” állapoton át

i+1-re vizsgált terminálás, shpath szerepel

Az algoritmus harmadik finomítása

- A legrövidebb utak keresése során az összes kezdőállapotot figyelembe vesszük
  - Olyan utak elkerülhetők, amelyek esetén a végállapot egy másik kezdőállapotból rövidebben elérhető
  - A kezdőállapotokat tehát együttesen vesszük figyelembe

$$shpath'_j(I, s^k) = \exists s^0 : (I(s^0) \wedge path_j(s^0, s^k)) \wedge$$

$$\neg \exists s^0 : (I(s^0) \wedge \bigvee_{0 \le i < j} path_i(s^0, s^k))$$

Legrövidebb út s^k-ba bármelyik kezdőállapotból, j hosszúságú

Nem igaz, hogy van olyan kezdőállapot, amiből j-nél rövidebb út vezet s^k-ba

- Hasonlóan, a „rossz” állapotokba vezető legrövidebb utak is definiálhatók

$$shpath'_j(s, \neg P)$$

A harmadik finomított algoritmus

$i = k$

while True do

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

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

if not SAT$(shpath'_{i+1}(I, s^{i+1}))$

or not SAT$(shpath'_{i+1}(s, \neg P))$

then return True

$i = i + 1$

end

- A fixpont iterációs algoritmushoz közelít...

A finomított algoritmusok összefoglalása

- Első algoritmus: Egyre hosszabb utak vizsgálata
  - SAT megoldóval ellenpélda keresés:
    - Előrefelé haladó 0,1,2,... hosszú ciklusmentes utak keresése kezdőállapotból
- Nem 0-ról kezdett iteráció
  - Gyorsabb eredmény
  - Legrövidebb ellenpélda megtalálása nem garantált
- Legrövidebb utak figyelembe vétele
  - Kisebb lehet az állapottér bejáráshoz szükséges iterációk száma
  - Nagyobb kihívás a SAT megoldónak
- Állapothalmazok figyelembe vétele
  - Kezdőállapotok, „rossz” állapotok együttesen kezelve

Kitekintés: A k-indukció

- Bevezetés: P_i legyen tulajdonságok sorozata
  - Hagyományos matematikai indukció:

$$P_0 \wedge \forall i : \left(P_i \Rightarrow P_{i+1} \right) \Rightarrow \forall n : P_n$$

  - k-indukció:

$$\bigwedge_{j=0}^{k-1} P_j \wedge \forall i : \left(\left(\bigwedge_{j=0}^{k-1} P_{i+j} \right) \Rightarrow P_{i+k} \right) \Rightarrow \forall n : P_n$$

- Ötlet: Alkalmazás állapottereken invariáns ellenőrzésre
  - Kiindulás: k hosszú, a kezdőállapotból induló útvonalakon teljesül az invariáns (ez korlátos modellellenőrzéssel kiderül)
  - Indukciós feltétel: Ha k hosszú, tetszőleges állapotból induló útvonalra teljesül az invariáns, akkor az útvonal utolsó állapotának rákövetkező állapotára is teljesül
    - Útvonalak vizsgálata: Több megkötést ad
    - Egy állapotból nem feltétlenül adódik a következő állapotra, de k lépésből lehet, hogy adódik a k+1-edikre a tulajdonság

Kitekintés: A k-indukció állapottereken

- Alapképlet: $\bigwedge_{j=0}^{k-1} P_j \wedge \forall i : \left(\left(\bigwedge_{j=0}^{k-1} P_{i+j} \right) \Rightarrow P_{i+k} \right) \Rightarrow \forall n : P_n$

- Kiindulás: $\bigwedge_{j=0}^{k-1} P_j$

  Megfelelője:

$$\forall s^0, s^1, \ldots, s^{k-1} : \left(I(s^0) \wedge path(s^0, s^1, \ldots, s^{k-1}) \right) \Rightarrow \left(\forall 0 \leq j < k-1 : \; P(s^j) \right)$$

- Indukciós feltétel: $\forall i : \left(\left(\bigwedge_{j=0}^{k-1} P_{i+j} \right) \Rightarrow P_{i+k} \right)$

  Megfelelője:

$$\forall i : \forall s^i, s^{i+1}, \ldots, s^{i+k} : \left(path(s^i, s^{i+1}, \ldots, s^{i+k}) \wedge \bigwedge_{j=i}^{i+k-1} P(s^i) \right) \Rightarrow P(s^{i+k})$$

Kitekintés: A k-indukció használata

- Esetek az invariáns kiértékelésére:
  - Ha a kiinduló lépés (korlátos modellellenőrzés) során adódik ellenpélda: Az invariáns nem teljesül
  - Ha a kiinduló lépés során sem és az indukciós feltétel ellenőrzése során sem adódik ellenpélda: Az invariáns teljesül
  - Egyébként: Az invariáns teljesülése nem ismert
    - Az indukciós ellenpéldában nem lehet bízni (nem biztos, hogy létezik az adott kezdőállapot mellett)
- Továbblépés, ha nem ismert az eredmény:
  - Az indukció mélységének növelése
    - Hátha hosszabb útvonalakkal adódik a következmény
  - Az invariáns erősítése: P helyett P' ellenőrzése, ahol P' => P
  - Kiegészítő invariáns használata
    - Ha van már ismert (másik) L invariáns, ezzel korlátozhatók az utak
    - Az indukciós feltétel módosítása:

$$\bigwedge_{j=0}^{k-1} P_j \wedge \forall i : \left(\left(\bigwedge_{j=0}^{k-1} (P_{i+j} \wedge L_{i+j}) \right) \Rightarrow P_{i+k} \right) \Rightarrow \forall n : P_n$$

Összefoglalás: A BMC használata

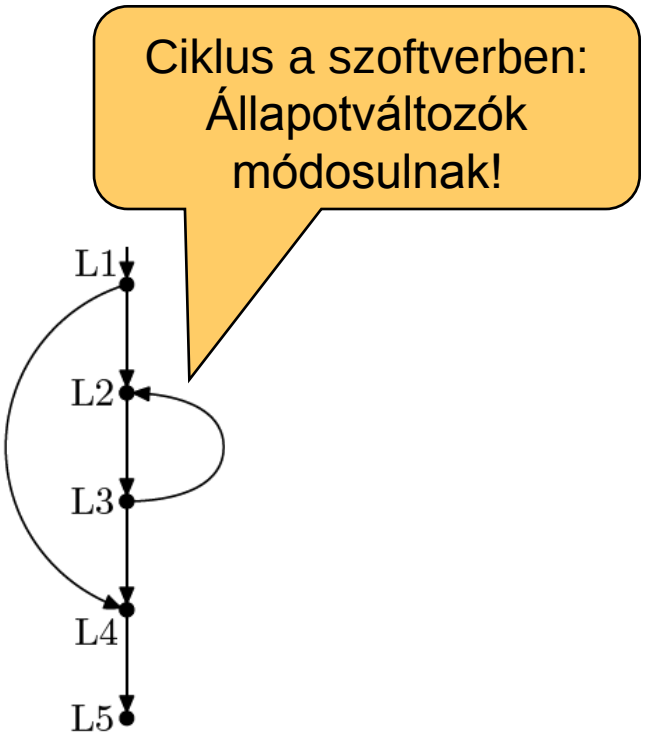
- Invariánsok vizsgálatára hatékony
  - Nem az általános modell ellenőrzési feladat megoldása!
- Helyes és teljes módszer az adott korlát mellett
  - Ha van ellenpélda, azt megtalálja (egyébként az invariáns igaz)
  - Ha ellenpéldát talál, akkor az valódi ellenpélda
  - A ciklusmentes utak használata bonyolultabbá tesz
- Állapottér robbanásának „elkerülése”
  - Adott számú iteráció vizsgálatával részleges eredmény kapható
- Legrövidebb ellenpélda keresése
  - Tesztgeneráláshoz használható
- Automatikus módszer
  - A korlát kijelölése lehet heurisztikus (az állapottér „átmérője”)
- Eszközök:
  - SAL: sal-smc, sal-bmc, sal-atg

Az Intel eredményei (hardver verifikáció)

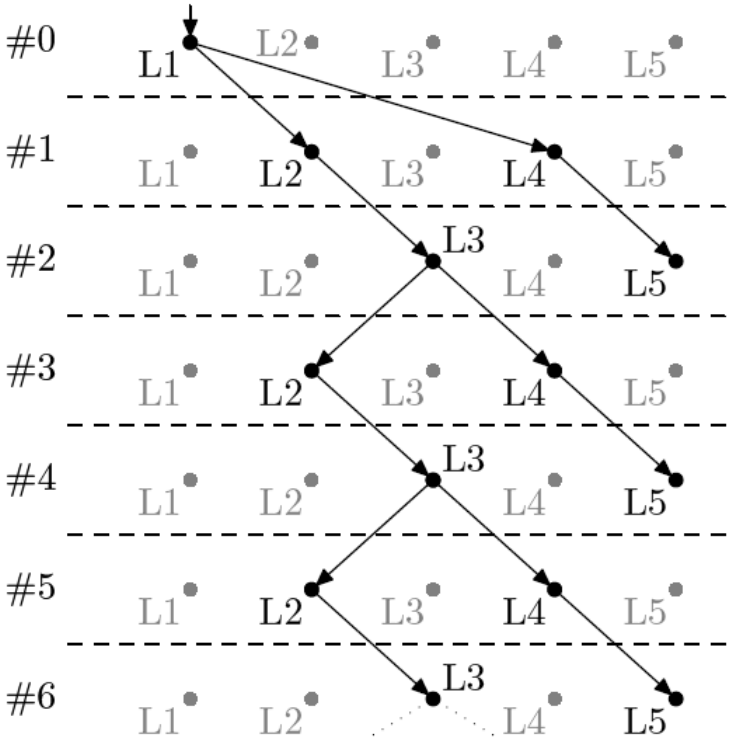
Model	k	Forecast (BDD)	Thunder (SAT)
Circuit 1	5	114	2.4
Circuit 2	7	2	0.8
Circuit 3	7	106	2
Circuit 4	11	6189	1.9
Circuit 5	11	4196	10
Circuit 6	10	2354	5.5
Circuit 7	20	2795	236
Circuit 8	28	—	45.6
Circuit 9	28	—	39.9
Circuit 10	8	2487	5
Circuit 11	8	2940	5
Circuit 12	10	5524	378
Circuit 13	37	—	195.1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

Alkalmazás szoftverek esetén: A ciklusok problémája

A ciklusok bejárása új állapotokat eredményez!



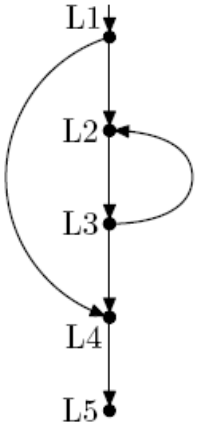
Vezérlési gráf (CFG) példa

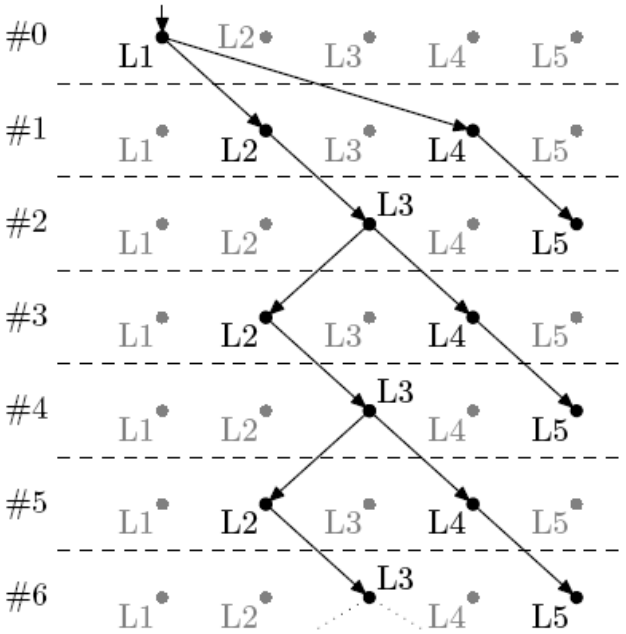


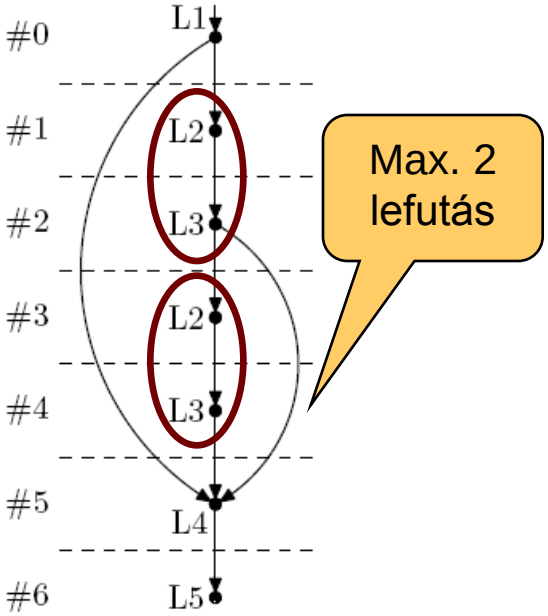
Széthajtogatás

Korlátozott ciklusbejárás

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







Eszközök

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