

Korlátos modellellenőrzés (Bounded Model Checking)

Majzik István

BME Méréstechnika és Információs Rendszerek Tanszék

Ismétlés: Az állapottér robbanás kezelése

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

Lehetőség: A SAT megoldók fejlődése

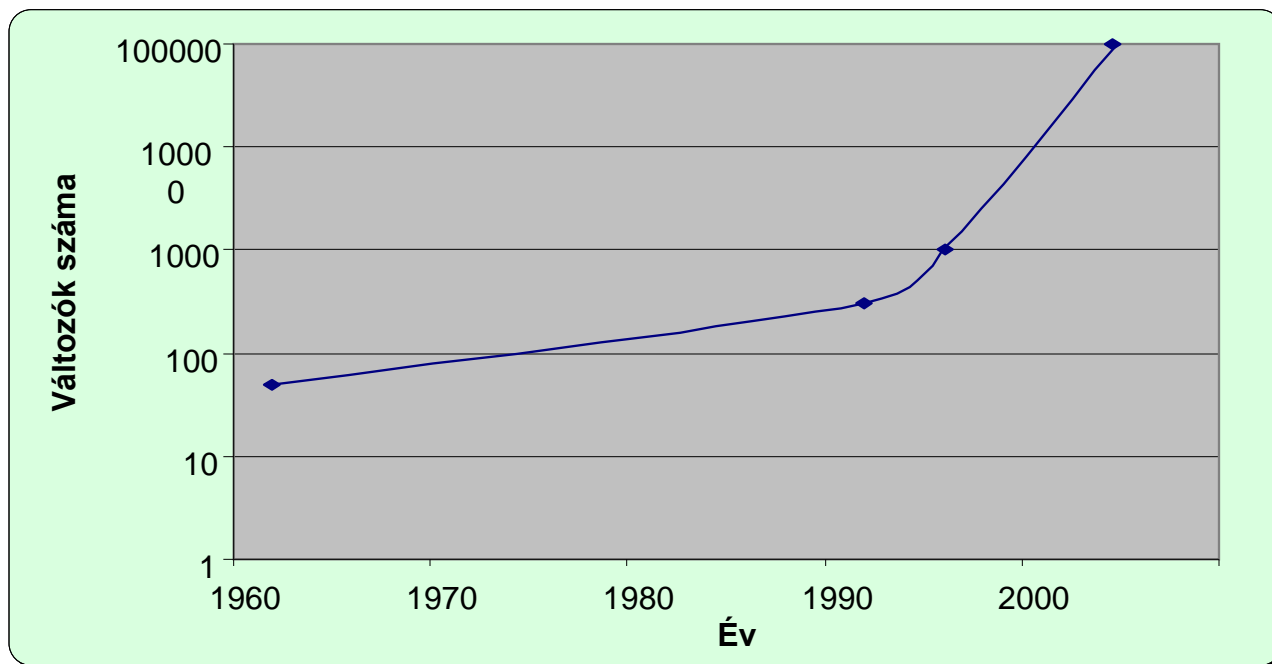
- SAT megoldó:
 - Adott logikai függvényhez olyan **változó érték behelyettesítést** keres, amelyekkel a függvény értéke **igaz**
- NP-teljes probléma, de hatékony algoritmusok léteznek
 - zChaff, MiniSAT

$$\omega_1 = (x_2 \vee x_3)$$

$$\omega_2 = (\neg x_1 \vee \neg x_4)$$

$$\omega_3 = (\neg x_2 \vee x_4)$$

$$A = \{x_1=0, x_2=1, x_3=0, x_4=1\}$$

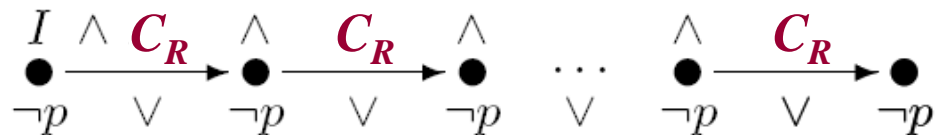


Célkitűzés

- 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

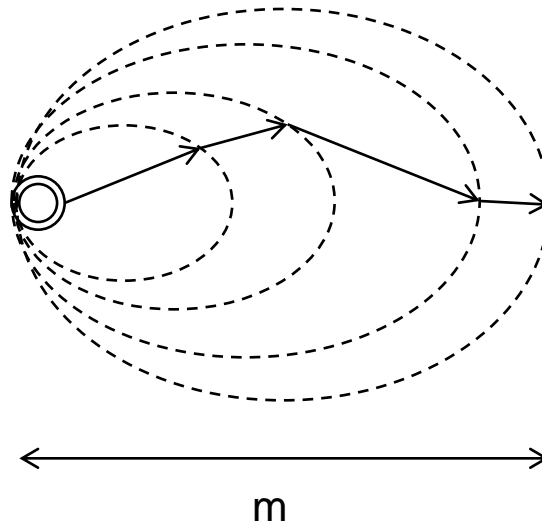
Informális bevezetés

- Hogyan képezzük az állapotteret?
 - Kiindulás a kezdőállapot(ok)ból: $I(s)$ 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, $C_R(s,s')$ adja meg a lehetséges s' rákövetkezőket, és így tovább
- Hogyan adjuk meg a követelményt?
 - Invariáns: Minden állapotra előírt kritérium: $p(s)$ predikátum
- Az ellenpéldát kijelölő logikai függvény: A következő konjunkció (\wedge kapcsolat):
 - Kezdőállapotból indulunk
 - Kibontjuk az állapotátmeneti reláció mentén egymás után
 - Valahol $p(s)$ nem teljesül: $\neg p(s)$ diszjunkciója (\vee kapcsolat) minden állapotraEzt 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

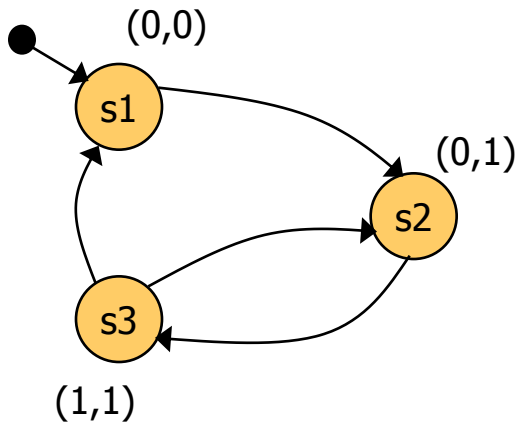
Vesszős változók helyett felső index

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

- Adott végpontok között k hosszúságú útvonal létezése

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

Példa: A modell leképzése logikai függvénybe

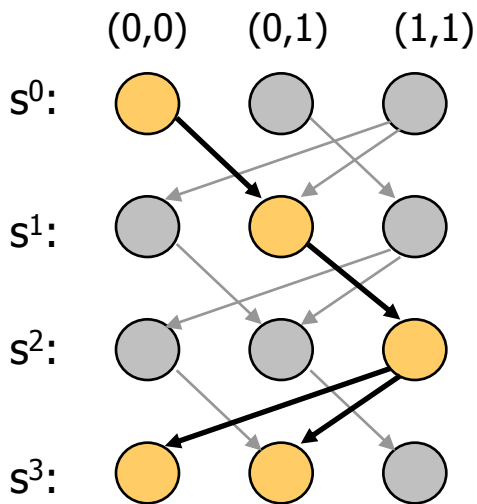


Kezdőállapot predikátum:

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

Állapotátmeneti reláció:

$$\begin{aligned}
 C_R(x,y,x',y') = & (\neg x \wedge \neg y \wedge \neg x' \wedge y') \vee \\
 & \vee (\neg x \wedge y \wedge x' \wedge y') \vee \\
 & \vee (x \wedge y \wedge \neg x' \wedge y') \vee \\
 & \vee (x \wedge y \wedge \neg x' \wedge \neg y')
 \end{aligned}$$



3 lépéses kihajtogatás a kezdőállapotból:

$$\begin{aligned}
 I(x^0,y^0) \wedge \text{path}(s^0,s^1,s^2,s^3) = \\
 = I(x^0,y^0) \wedge \\
 C_R(x^0,y^0, x^1,y^1) \wedge \\
 C_R(x^1,y^1, x^2,y^2) \wedge \\
 C_R(x^2,y^2, x^3,y^3)
 \end{aligned}$$

A probléma formalizálása

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

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

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

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

- A fv. igaz értékéhez tartozó behelyettesítést ad a SAT megoldó!
 - Azaz az (s^0, s^1, \dots, s^i) útvonalat meghatározó $(i+1)n$ változó értéket
- Első ötlet: $i=0, 1, 2, \dots$ -ra rendre megvizsgálni, hogy i hosszú útvonalon igaz lehet-e a következő függvény (ha igaz, akkor van ellenpélda):

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

Az algoritmus elemei

- Iteráció: $i=0,1,2,\dots$ az útvonalak hosszára
- Ciklusmentes utakat vizsgálunk: *lfp*ath

$$lfp\text{ath}(s^0, s^1, \dots, s^k) = \text{path}(s^0, s^1, \dots, s^k) \wedge \bigwedge_{0 \leq i < j \leq k} s^i \neq s^j$$

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

$$I(s^0) \wedge lfp\text{ath}(s^0, s^1, \dots, s^i)$$

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

$$lfp\text{ath}(s^0, s^1, \dots, s^i) \wedge \neg P(s^i)$$

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

Az algoritmus

$i = 0$

while True do

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

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

then return True

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

then return (s^0, s^1, \dots, s^i)

$i = i + 1$

end

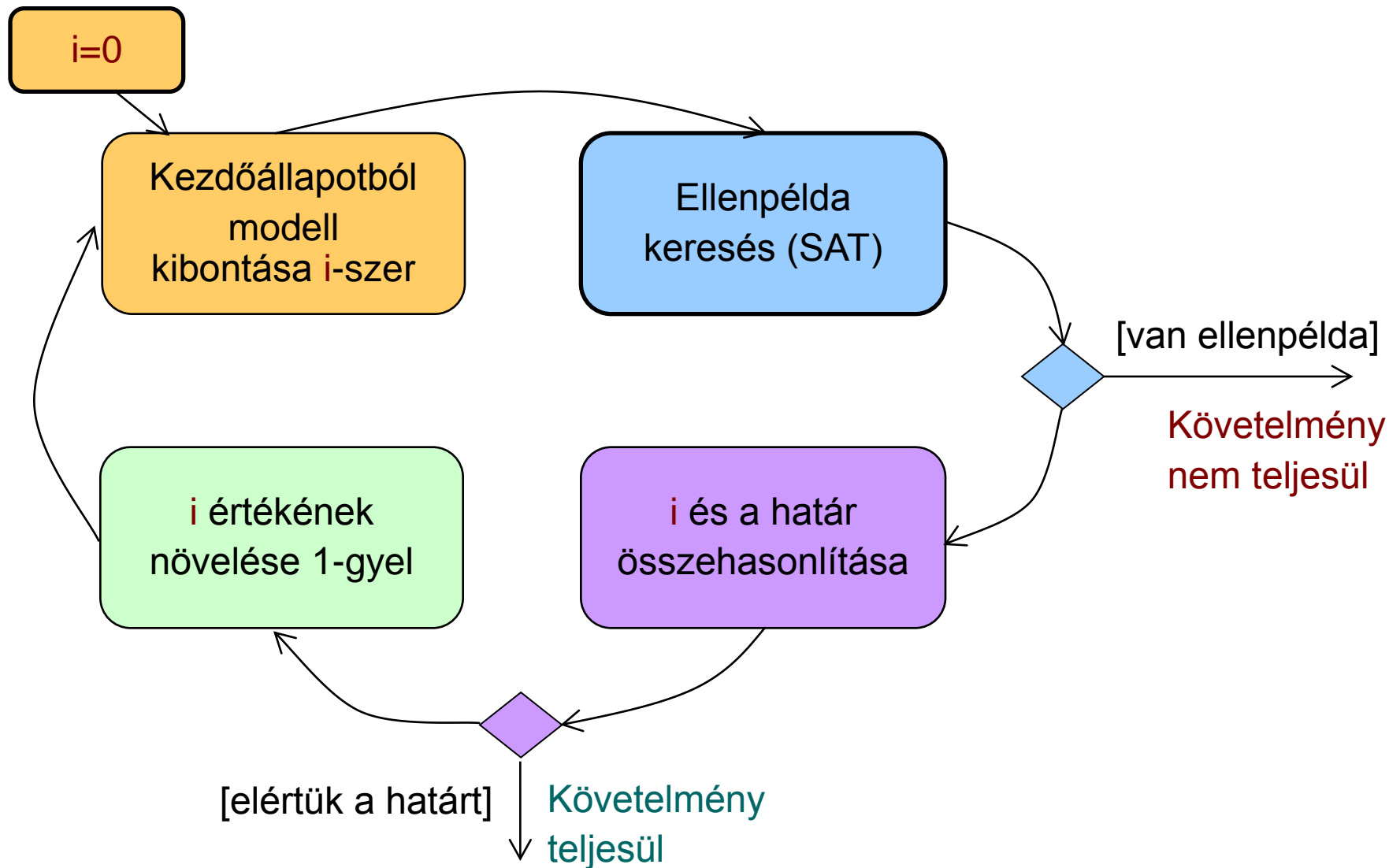
Nincs már i hosszú ciklusmentes út kezdőállapotból

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, \dots, 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

Van i hosszú útvonal kezdőállapotból „rossz” állapoton át

while True do

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

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

then return (s^0, s^1, \dots, s^i)

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

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

then return True

$i = i + 1$

end

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

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*

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

- Az algoritmus átírása: *lfp* helyett *shp*
- 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 kezdőállapottól 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 $\text{SAT}(I(s^0) \wedge \text{path}(s^0, s^1, \dots, s^i) \wedge \neg \bigwedge_{j=0}^i (P(s^j)))$

then return (s^0, s^1, \dots, s^i)

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

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

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

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

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

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

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

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

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

A harmadik finomított algoritmus

$i = k$

while True do

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

then return (s^0, s^1, \dots, s^i)

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

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

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: k-indukció

- Bevezetés: P_i legyen tulajdonságok sorozata

- Hagyományos matematikai indukció:

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

- 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
 - Kiinduló lépés: k hosszú, a kezdőállapotból induló útvonalakon teljesül az invariáns
 - Korlátos modellellenőrzéssel ellenőrizhető a kezdőállapotból
 - Indukciós lépés: ha k hosszú, tetszőleges állapotból induló útvonalra teljesül az invariáns, akkor az útvonal utolsó állapotának rákövetkező állapotára is teljesül
 - Csak az állapotátmeneti reláció alapján vizsgálandó
 - Útvonalak vizsgálata: Több megkötést ad

Kitekintés: A k-indukció állapottereken

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

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

Megfelelője:

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

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

Megfelelője:

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

Kitekintés: A k-indukció használata

- Esetek az invariáns kiértékelésére:
 - A kiinduló lépés (korlátos modellellenőrzés) során adódik ellenpélda: Az invariáns nem teljesül
 - A kiinduló lépés során sem és az indukciós lépés során sem adódik ellenpélda: Az invariáns teljesül
 - Egyébként: Az invariáns teljesülése nem ismert
 - Az indukciós ellenpéldában nem lehet bízni (nem biztos, hogy létezik)
- Továbbblépés, 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 lépés 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

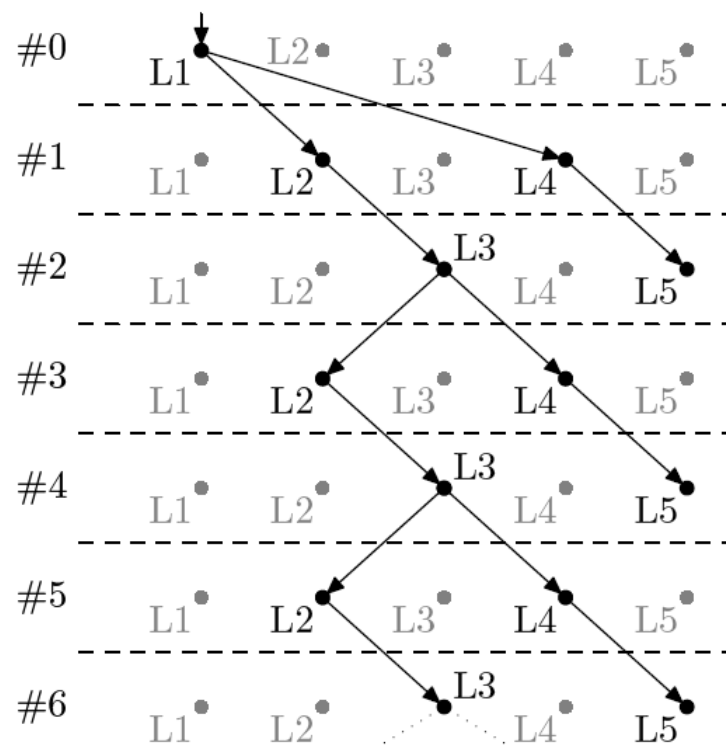
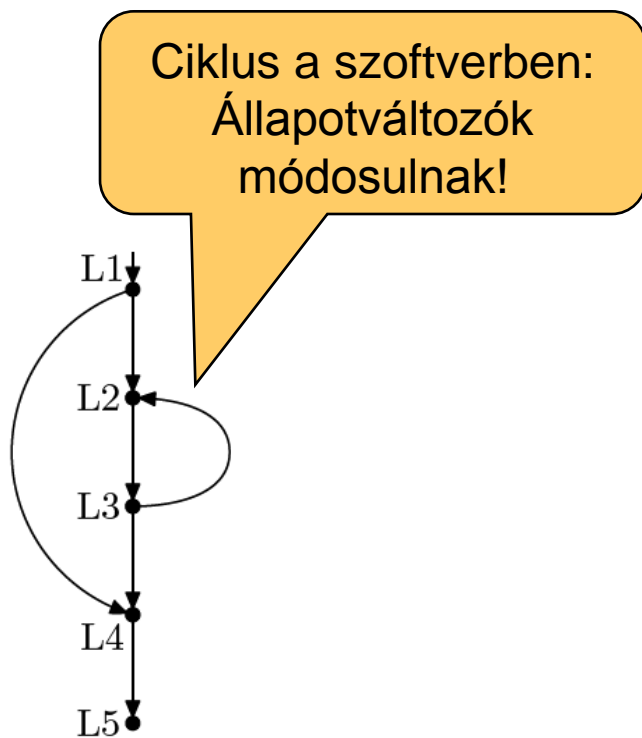
- Invariánsok vizsgálatára hatékony
 - Nem az általános modell ellenőrzési feladat megoldása!
- Helyes és teljes módszer az adott korlát mellett
 - Ha van ellenpélda, azt megtalálja (egyébként az invariáns igaz)
 - Ha ellenpéldát talál, akkor az valódi ellenpélda
 - A ciklusmentes utak használata bonyolultabbá 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!

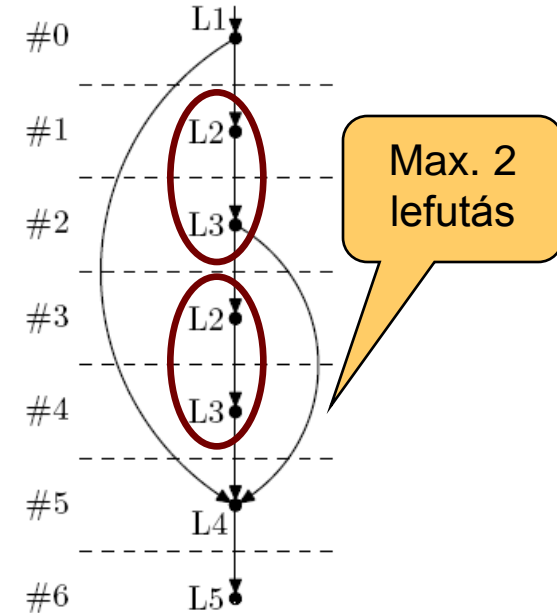
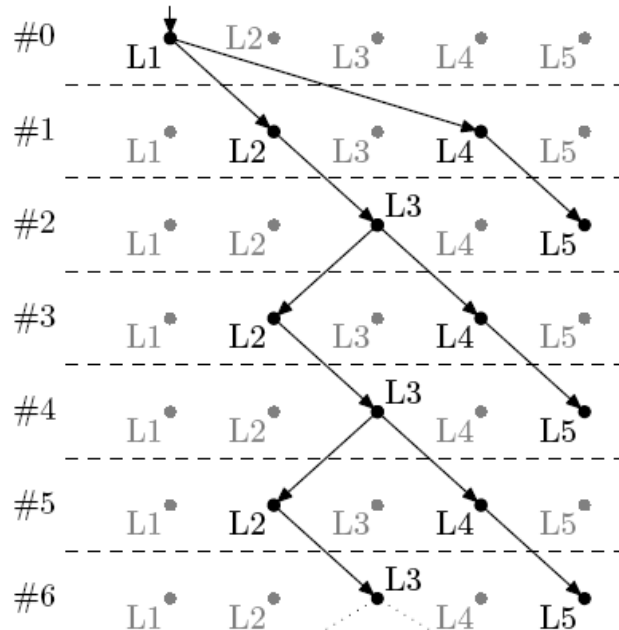
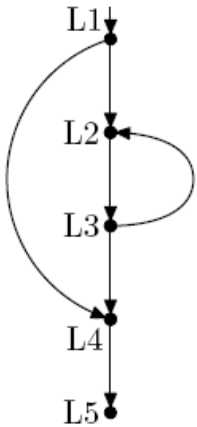


Vezérlési gráf (CFG) példa

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**: A SAT megoldó kiterjesztése különböző domének kezelésére (pl. integer aritmetika)
- SATURN:
 - Korlátozott ciklusbejárás: max. 2 lefutás
 - Teljes Linux kernel ellenőrizhető: Null pointer hivatkozásokra