

# Korlátos modellellenőrzés

dr. Majzik István

BME Méréstechnika és Információs Rendszerek Tanszék

# Hol tartunk most?

- Alacsony szintű formalizmusok (KS, LTS, KTS)
- Magasabb szintű formalizmusok

Temporális logikák:  
PLTL, CTL, CTL\*

Rendszer modellje

Követelmény megadása

Alapszintű  
algorithmus

Hogyan lesz  
hatékony?

i

Automatikus  
modellellenőrző

n

OK

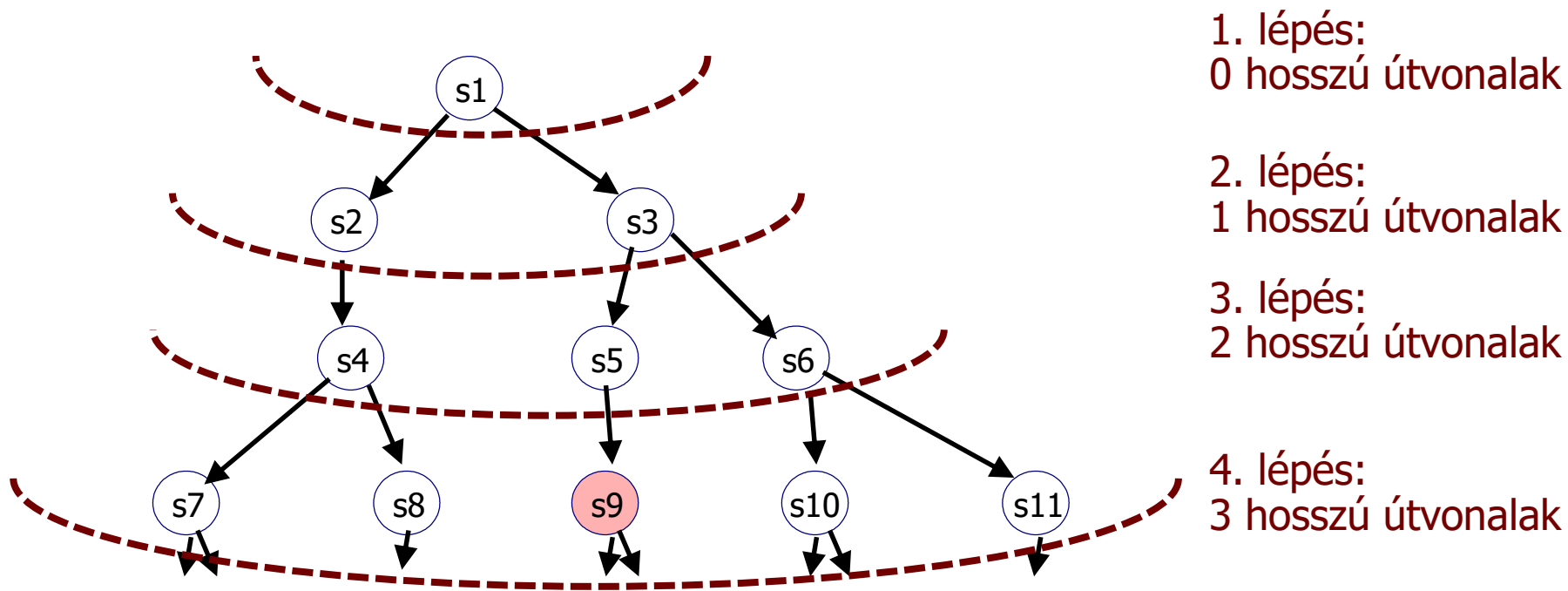
Ellenpélda

# A korlátozott modellellenőrzés alapja

- Az állapotteret nem „egyben” kezeljük
- Az útvonalak hosszát korlátozva végezzük az ellenőrzést
  - Állapottér bejárás egyszerre csak adott útvonalhossz korlátig történik: részleges ellenőrzés
  - Az útvonalhossz iteratívan növelhető
  - Egyes esetekben van „átmérője” az állapottérnek: a leghosszabb útvonal, ami bejárható
- A leghosszabb útvonal néha becsülhető
  - Intuíció a probléma méretéről
  - Worst case végrehajtási idő alapján

# A korlátos modellellenőrzés alapötlete

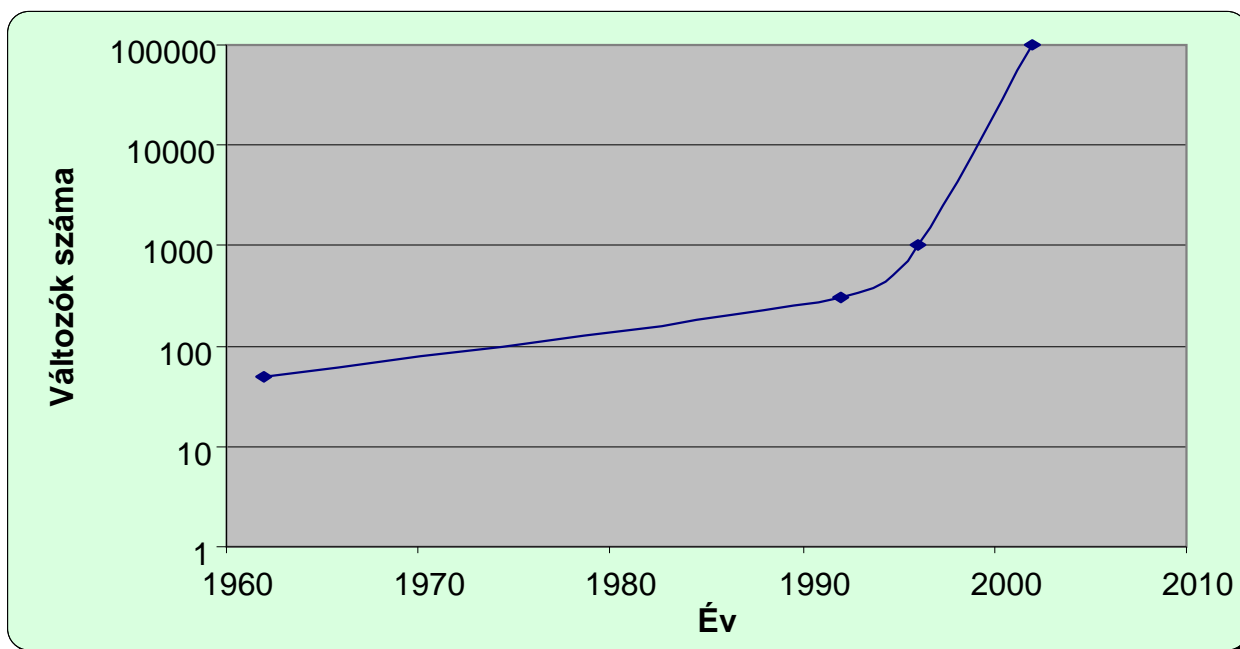
Ellenpélda keresés a kezdőállapotból egyre hosszabb útvonalakon



Hamarabb ellenpéldát találhatunk, a teljes állapottér bejárása nélkül

# Felhasználható eredmény: SAT megoldók

- SAT megoldó:
  - Adott logikai függvényhez olyan változó-értékeket (változó-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 vannak
  - zChaff, MiniSAT, ...

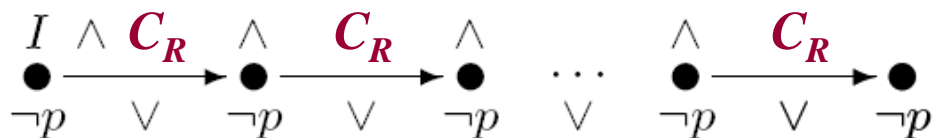


# Célkitűzés

- A probléma alkalmas leképezése logikai függvényre
  - Modell és logikai követelmény együttesen
  - Tipikusan invariáns követelmény ellenőrzésére:  
Minden állapotra előírt tulajdonság
- Cél: SAT megoldó használata ellenpélda keresésre
  - Ha a követelmény nem teljesül, a SAT megoldó által adott behelyettesítés legyen egy ellenpélda
    - Az ellenpélda használható a hibakereséshez
    - Invariáns tulajdonságok esetén jól használható módszer
  - Ha a követelmény teljesül, a SAT megoldó nem talál behelyettesítést a függvényhez

# Informális bevezetés

- Hogyan képezzük az állapotteret?
    - Kiindulás a kezdőállapotból:  $I(s)$  karakterisztikus függvénnyel megadható
    - „Bejárás”: 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” az állapotátmeneti reláció mentén:  $C_R(s, s')$  sorozatos alkalmazása
    - Ellenpélda (valahol  $p(s)$  nem teljesül):  $\neg p(s)$  vagylagosan a bejárt állapotokra
- Ezt a függvényt igazzá tevő behelyettesítés adja az ellenpéldát!



# Jelölések

- $M=(S,R,L)$  Kripke-struktúra
- Logikai függvények:
  - $I(s)$ : kezdőállapotok  $n$  változós karakterisztikus függvénye
    - Háttér: Állapotok „kódolása”  $n$  hosszú bitvektorokkal
  - $C_R(s,s')$ : állapotátmenetek  $2n$  változós karakterisztikus függvénye
    - Minden egyes állapotátmenet karakterisztikus függvénye  $\vee$  operátorral
  - $\text{path}()$ :  $k$  hosszú útvonalak  $(k+1)n$  változós karakt. függvénye

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

Vesszős változók helyett  
felső index az  $s$   
állapotok kódolásában

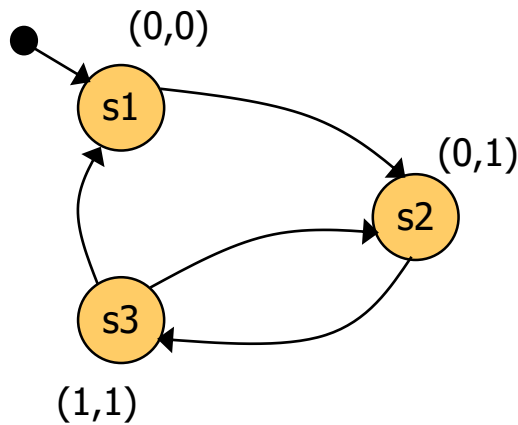
- Kezdőállapotból induló,  $k$  hosszú útvonalak karakt. függvénye

$$I(s^0) \wedge \text{path}(s^0, s^1, \dots, s^k)$$

- $p(s)$ : címkézett állapotok halmazának karakterisztikus függvénye
  - Atomi kijelentésekre pl.  $P(s)$ ,  $Q(s)$ : Az  $L$  címkézés határozza meg
  - Általánosan: Állapotváltozók alapján konstruálható



# Példa: A modell leképzése logikai függvénybe

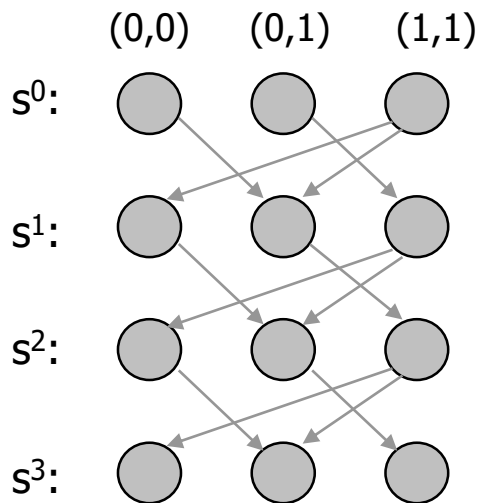


Kezdőállapot predikátum:

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

Állapotátmeneti reláció:

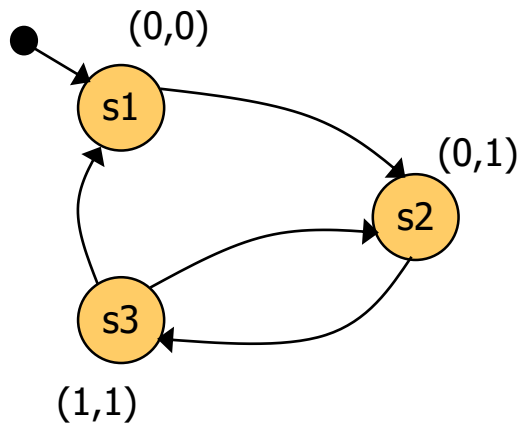
$$\begin{aligned}
 C_R(x,y,x',y') = & (\neg x \wedge \neg y \wedge \neg x' \wedge y') \vee \\
 & \vee (\neg x \wedge y \wedge x' \wedge y') \vee \\
 & \vee (x \wedge y \wedge \neg x' \wedge y') \vee \\
 & \vee (x \wedge y \wedge \neg x' \wedge \neg y')
 \end{aligned}$$



3 lépéses kibontás:

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

# Példa: A modell leképzése logikai függvénybe

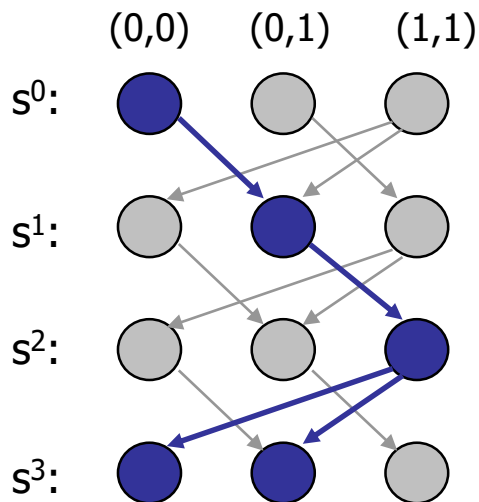


Kezdőállapot predikátum:

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

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 kibontás 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

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

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

A fenti függvény igaz értékéhez tartozó behelyettesítést tud keresni a SAT megoldó, ez lesz az ellenpélda

- Azaz az  $(s^0, s^1, \dots, s^i)$  útvonalat meghatározó  $(i+1)*n$  változó értéket ( $i+1$  állapot, állapotonként  $n$  bitérték) keres
- Ötlet:  $i=0,1,2,\dots$ -ra rendre megvizsgálni, hogy  $i$  hosszú útvonalon igaz lehet-e a következő függvény (ha igaz, akkor van ellenpélda):

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

# Az algoritmus elemei

- Iteráció:  $i=0,1,2,\dots$  az útvonalak hosszára
- Ciklusmentes utakat vizsgálunk:  $\text{lfpath}$

Kifejthető az állapotváltozókkal

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

- Megállási feltétel az iteráció során:
  - Kezdőállapotból nincs  $i$  hosszú ciklusmentes út, azaz nem lehet igaz

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

- A „rossz” állapothoz (ahol  $p(s)$  nem igaz) nem létezik  $i$  hosszú ciklusmentes út (kezdőállapottól függetlenül), azaz nem lehet igaz

$$\text{lfpath}(s^0, s^1, \dots, s^i) \wedge \neg p(s^i)$$

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

# Az algoritmus

$i = 0$

while True do

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

then return True

if 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

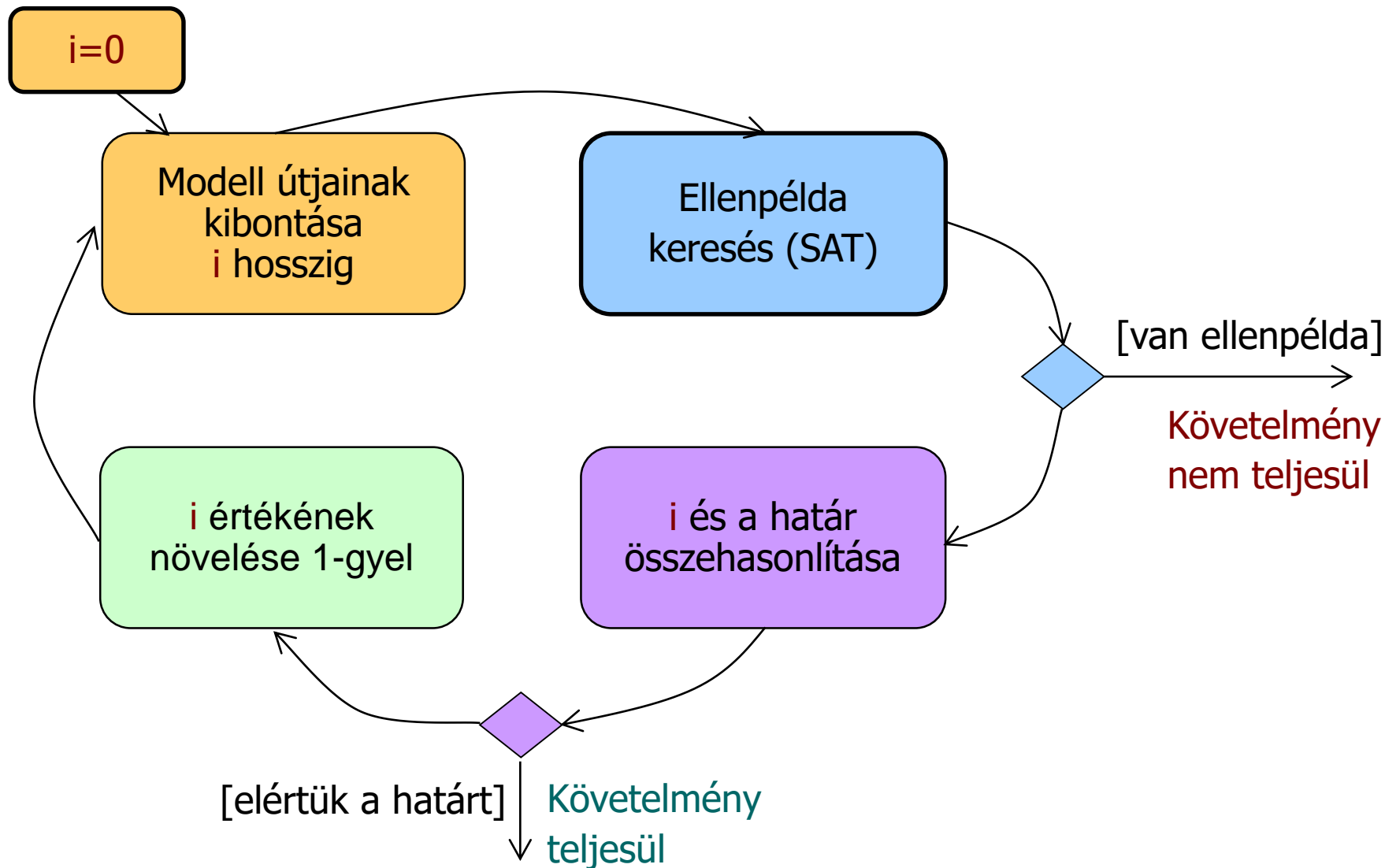
Nincs  $i$  hosszú  
ciklusmentes út  
„rossz” állapotra

Van  $i$  hosszú út  
kezdőállapotból  
„rossz” állapotba

Iteráció

- 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

# Korlátos modellellenőrzés iterációval



# Összefoglalás: A BMC használata

- Invariánsok vizsgálatára hatékony
- Helyes módszer ciklusmentes utakat használva
  - Ha van ellenpélda az adott úthossz korlátig, azt megtalálja
  - Ha ellenpéldát talál, akkor az valódi ellenpélda
- Állapottér kezelés
  - SAT megoldó: Szimbolikus technika, logikai függvényeken
  - Adott úthossz korlátig részleges eredmény kapható
- Legrövidebb ellenpélda kereshető
  - Tesztgeneráláshoz használható
- Automatikus módszer
  - A korlát kijelölése lehet heurisztikus (az állapottér várt „átmérője”)
- Eszközök:
  - Pl. Symbolic Analysis Laboratory (SAL): sal-bmc, sal-atg

# Az Intel eredményei (hardver verifikáció)

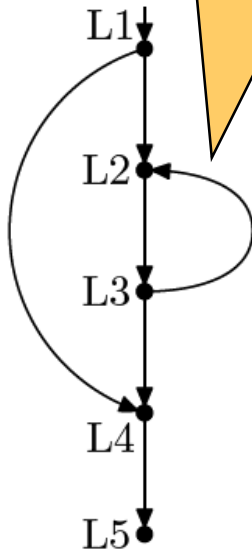
Model	$k$	Forecast (BDD)	Thunder (SAT)
Circuit 1	5	114	2.4
Circuit 2	7	2	0.8
Circuit 3	7	106	2
Circuit 4	11	6189	1.9
Circuit 5	11	4196	10
Circuit 6	10	2354	5.5
Circuit 7	20	2795	236
Circuit 8	28	—	45.6
Circuit 9	28	—	39.9
Circuit 10	8	2487	5
Circuit 11	8	2940	5
Circuit 12	10	5524	378
Circuit 13	37	—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



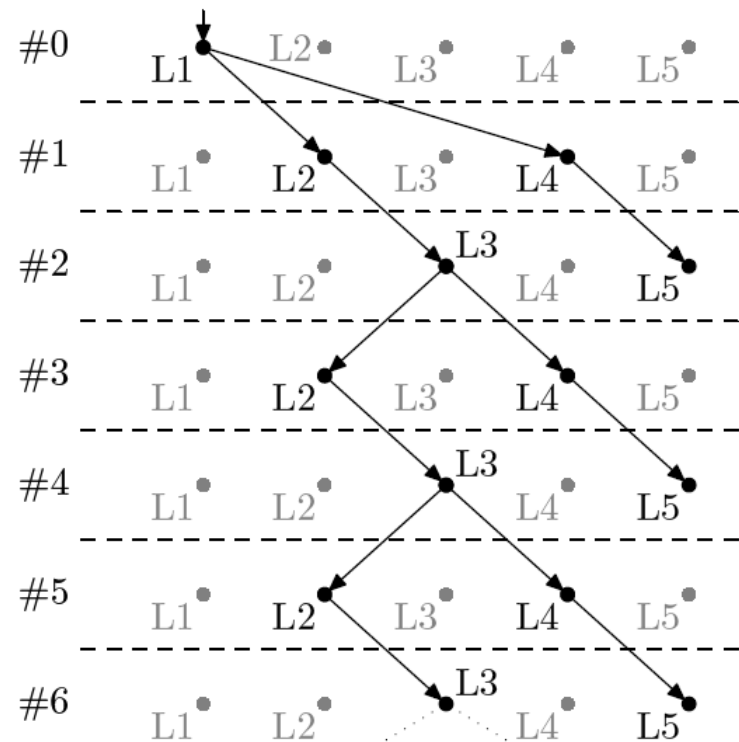
# Állapottér növekedés szoftverek esetén: A ciklusok

A ciklusok bejárása új állapotokat eredményez!

Ciklus a szoftverben:  
Az állapotváltozók  
módosulnak!



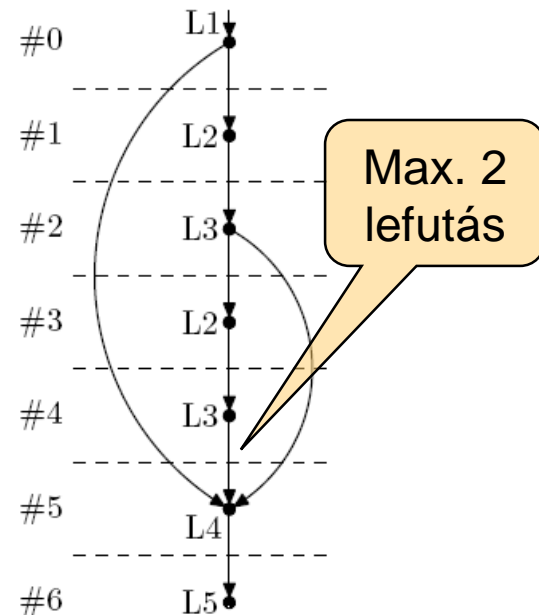
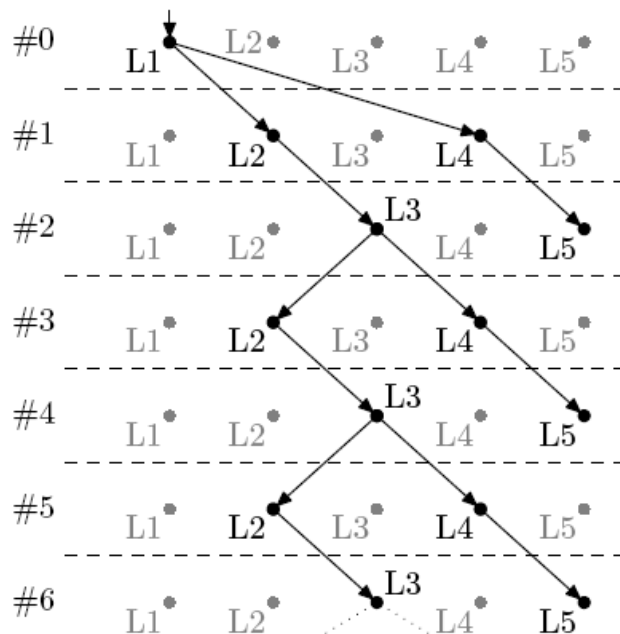
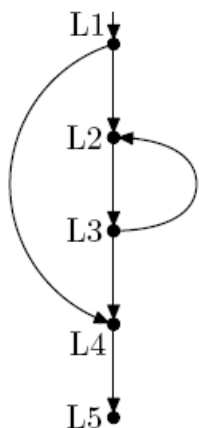
Vezérlési gráf (CFG) példa



Ciklus bejárással új  
állapotok jönnek létre

# Ciklusok kezelése a modellellenőrzőkben

- Ciklusok kezelésének lehetőségei
  - Alapeset: Teljes kibontás (path enumeration)
    - Szisztematikusan új állapotok felvétele minden lehetséges ágon
  - Közelítés: Korlátozott ciklusbejárás (loop unrolling)
    - Ciklusokra egyenként lefutási korlátot adni és úgy kibontani



# Szoftver modellellenőrzés

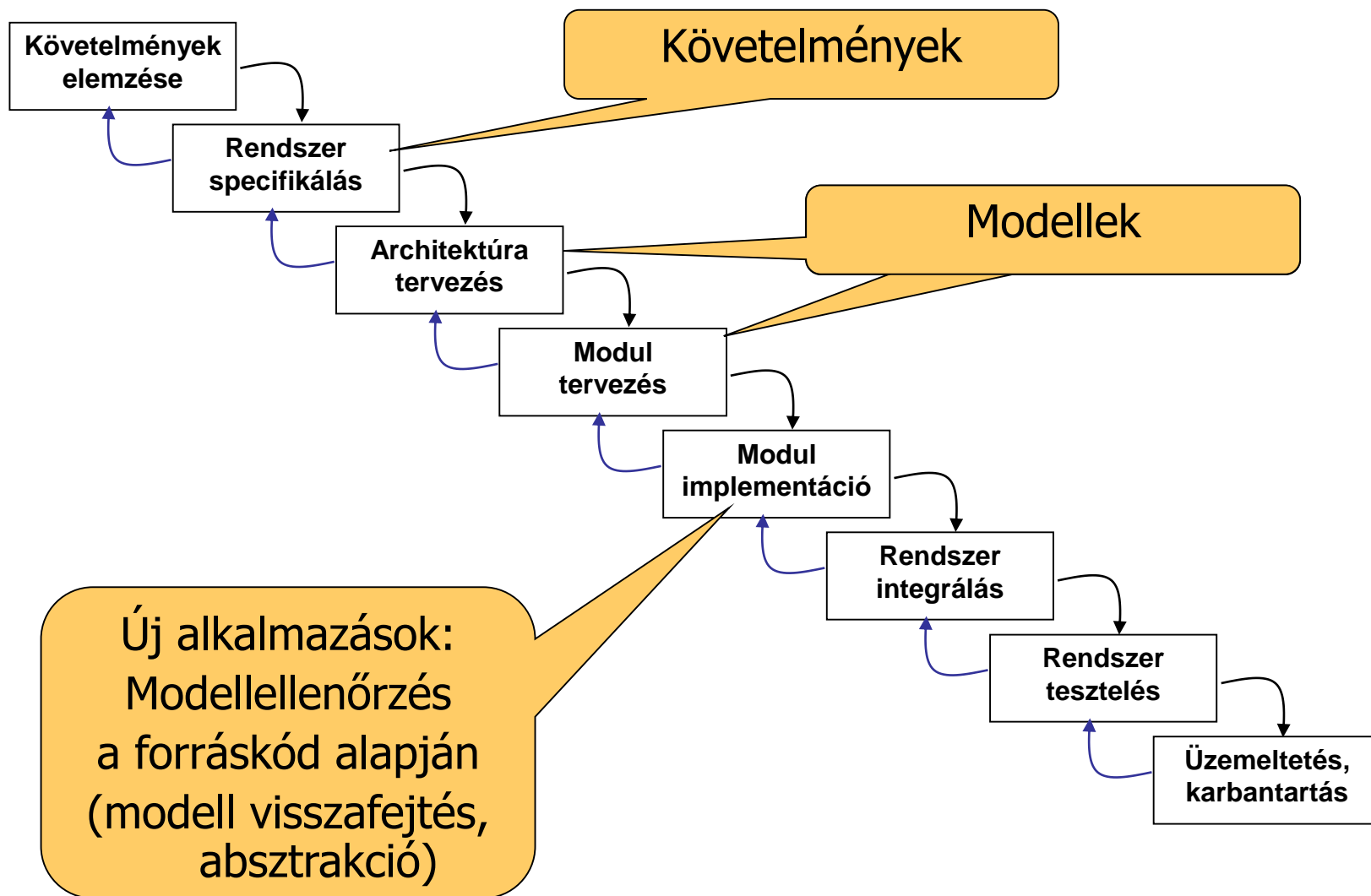
- F-SOFT (NEC):
  - Hagyományos teljes kibontás (path enumeration)
  - Unix rendszerprogramokra alkalmazták (pl. pppd)
- CBMC (CMU, Oxford University):
  - C, SystemC támogatása
  - Korlátozott ciklusbejárás (loop unrolling)
  - Egyes Linux, Windows, MacOS rendszerkönyvtárak támogatása
  - Integer aritmetikai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

# Összefoglalás: Technikák modellellenőrzéshez

- **Szimbolikus modellellenőrzés**
  - ROBDD alakban kezelt karakterisztikus függvények
  - „Jól strukturált” problémák esetén hatékony
    - Pl. azonos viselkedésű résztvevők protollokban
  - 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 ellenpélda keresése szükséges)
  - Tesztgenerálásra jól alkalmazható
- **További lehetőség: Absztrakció**

# Összefoglalás: 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 esetén)
  - Hatékony technikák: Szimbolikus, SAT alapú (korlátos), absztrakció
- 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 még mindig korlátos
  - Explicit állapotter bejárást alkalmaz
  - Hatékony technikák vannak ugyan erre, de nem garantált a jó skálázhatóság
- Elsősorban vezérlés-orientált alkalmazásokra
  - Komplex adatstruktúrák nagy állapotteret jelenthetnek
- Nehéz az eredményeket általánosítani
  - Ha egy protokoll helyes **2** résztvevő esetén, akkor helyes-e **N** résztvevő esetén?
- A követelmények formalizálása nem egyszerű
  - Temporális logika „nyelvjárások” alakultak ki különböző alkalmazási területeken
  - Példa: PSL (Property Specification Language, IEEE szabvány)