

**Absztrakció:  
Absztrakt interpretáció.  
Predikátum absztrakció**

Majzik István

Budapesti Műszaki és Gazdaságtudományi Egyetem  
Méréstechnika és Információs Rendszerek Tanszék

# Bevezető

- A szoftverek bonyolultak: Hatalmas állapottér
  - Nagy értékészletű változók (int, float, ...)
  - Ciklusok
  - Függvényhívások, rekurzió
  - Objektumok dinamikus létrehozása és törlése
  - ...
- Hogyan illeszthető az ellenőrzés a megismert „egyszerű” technológiákhoz?
  - Korlátozások bevezetése, pl.:
    - Cikluslefutások számának korlátozása (és így kihajtogatás)
    - Korlátos modellellenőrzés (pl. CBMC)
  - Absztrakció, pl.:
    - Absztrakt interpretáció: Statikus analízishez
    - Predikátum absztrakció: Program modellellenőrzéshez

**Absztrakt interpretáció:  
absztrakt statikus analízis**

# Statikus analízis (ismétlés)

- Alapötlet:
  - Változók lehetséges értékeinek terjesztése
  - Iteratív (amíg nem változik az értékkészlet)
- Felhasználási példák: Hibák detektálása
  - Tömbindexek túlcímzése
  - Nullával való osztás
  - Függvények értelmezési tartományának túllépése
  - ...
- Absztrakció:
  - A program konkrét interpretációja helyett absztrakt interpretációja (kisebb bonyolultságú elemzési feladat)

# Konkrét interpretáció

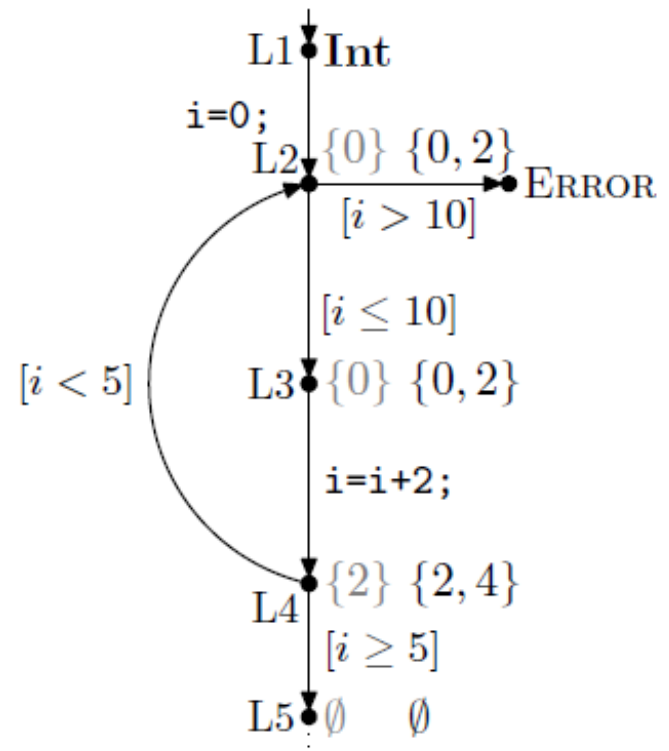
- Program példa:

```
int i=0;
do {
    assert (i<=10);
    i = i+2;
} while (i<5);
```

## 1. Konkrét interpretáció:

CFG jelölése:  $i$  értékei

- 1. lépés: szürke
- 2. lépés: fekete
- L2 esetén unió (L1, L4-től)
- Folytatás fixpontig

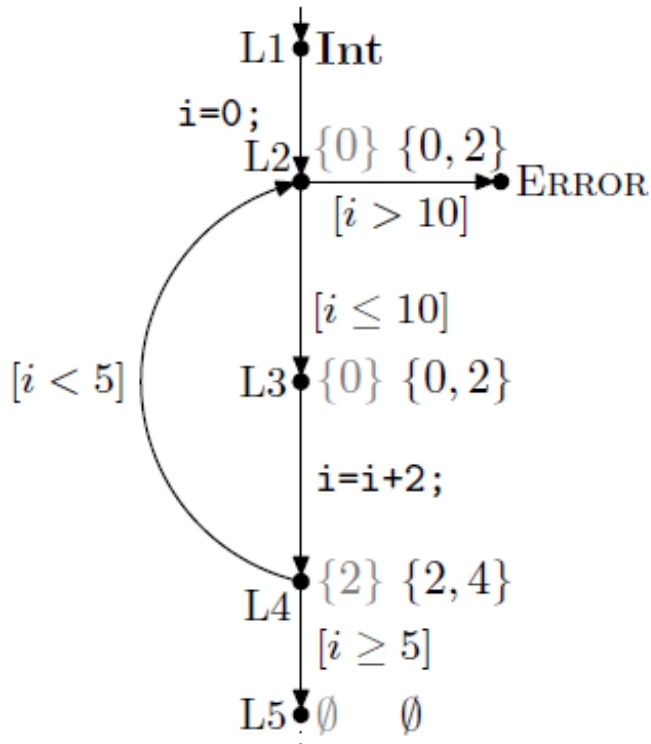


# Absztrakció

- Konkrét interpretáció:
  - Változók **érték**halmazainak iteratív terjesztése
  - Hátrány: Nagy méretű halmazok lehetnek
- Absztrakt interpretáció:
  - A konkrét domén leképezése **absztrakt doménre**
    - Pl. tartományra képezni le az adatértékeket (érték halmazt)
    - Pontos értékek „elvesznek”
  - Az absztrakt domén terjesztése az interpretáció során
    - Pl. tartományok uniójára való leképezés
- Analízis az absztrakt domén segítségével
  - Cél: **Biztonságos** konkrét-absztrakt domén leképezés
    - Az absztrakt domén lefedi a konkrét domént („érték nem vész el”)
    - De az absztrakt domén reprezentálhat olyan értékeket, amik nem fordulhatnak elő a konkrét esetben

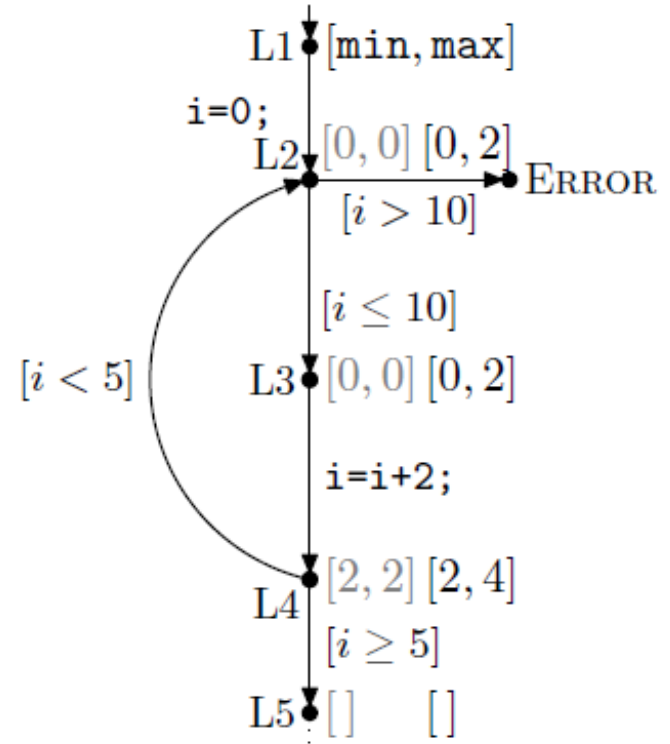
# Absztrakt interpretáció példa: Intervallumok

## 1. Konkrét interpretáció:



1. lépés: szürke
2. lépés: fekete

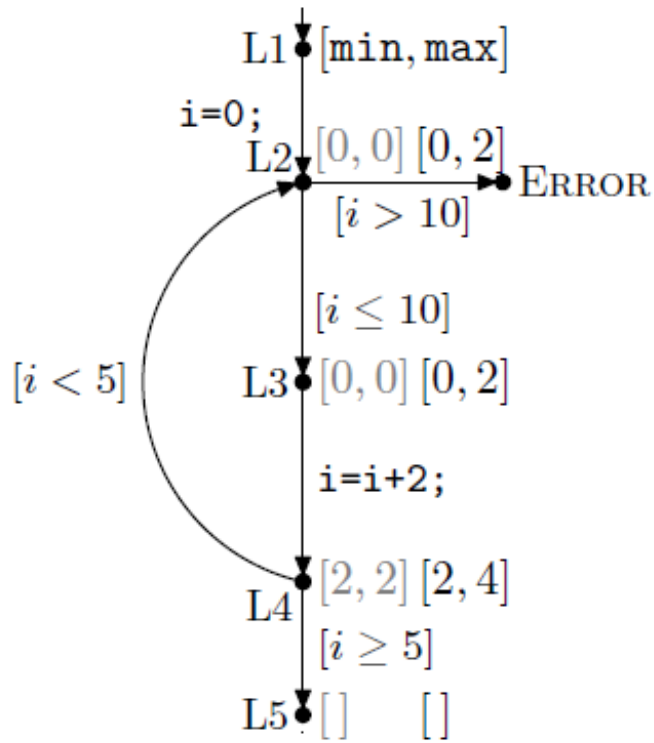
## 2. Absztrakt interpretáció:



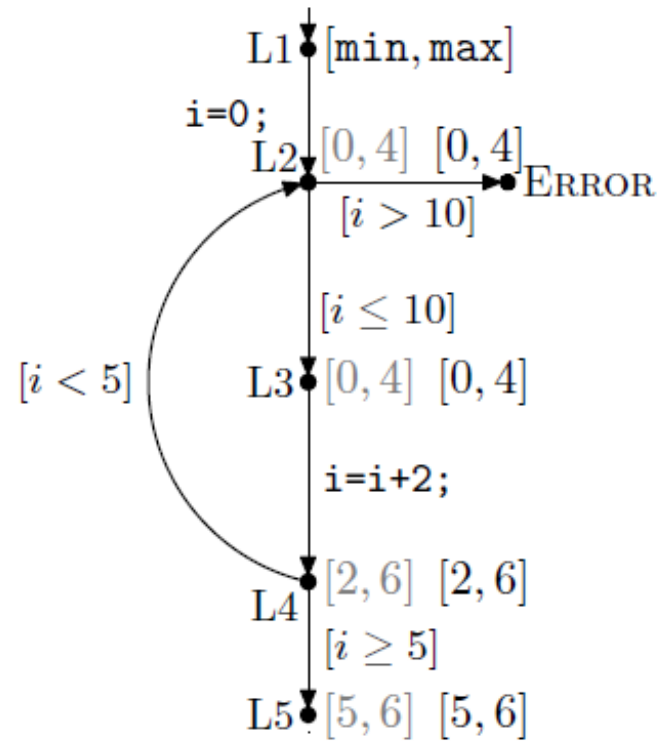
- L2-nél intervallum egyesítés:  
[0,0] és [2,2] esetén [0,2]  
Így **elveszett** az az információ,  
hogy  $i$  értéke nem lehet 1

# Absztrakt fixpont elérése

## 2. Absztrakt interpretáció:



## 3. Absztrakt fixpont:



Intervallumok tovább nem nőnek



# Absztrakt domének meghatározása

- Cél: Kijelentéseket tudjunk tenni numerikus változókra
  - Precízebb domén: Kevesebb információvesztés
  - Sokféle absztrakciót használhatunk (alkalmazásfüggő)
- Nem-relációs absztrakt domének:  
Változók közötti **relációk** nem őrizhetők meg  
(pl.  $x < y$  nem őrizhető meg az absztrakt doménben)
  - Előjelek: {Pos, Neg, Zero}
  - Intervallumok: Finomabb, mint az előjelek doménje
  - Paritás: Páros, páratlan
  - Kongruenciák:  $(v \bmod k)$  értékei alkotják
    - Egyenlőtlenségek kimutathatók
    - Pl. ha  $(x \bmod k) \neq (y \bmod k)$  kimutatható, akkor az  $1/(x-y)$  művelet nem vezet nullával való osztási hibához

# További absztrakt domének

- Relációs domének: Relációk megtartása
  - Difference Bound Matrices (DBM):
    - $x-y \leq c$ ,  $x \leq c$  és  $-x \leq c$  formájú egyenlőtlenségek konjunkciója jelöli ki a doméneket ( $c$  egész)
    - Valósídejű rendszerek ellenőrzéséhez alkalmazzák (órák különbségei)
  - Octagon:
    - $ax+by \leq c$ , alakú egyenlőtlenségek, ahol  $a$  és  $b$   $-1,0,1$  lehet,  $c$  egész
  - Octahedra: Egyenlőtlenségek több mint 2 változóra
  - Polyhedra:
    - $a_1x_1 + a_2x_2 + \dots + a_nx_n \leq c$ , alakú egyenlőtlenségek ( $a_i$  és  $c$  is egészek)
- Hierarchia a precizitás szempontjából:
  - Előjelek, Intervallumok, DBM, Octagons, Octahedra, Polyhedra
  - Használhatók egyenlőtlenségek vizsgálatára (pl. ciklus feltételek, valósídejű beágyazott rendszerek esetén)
- Ellipszoid domén:  $ax^2+bx+cy^2 \leq n$ 
  - Digitális szűrőkhöz használják
- Új absztrakt domének is definiálhatók (alkalmazásfüggő)

# Predikátum absztrakció

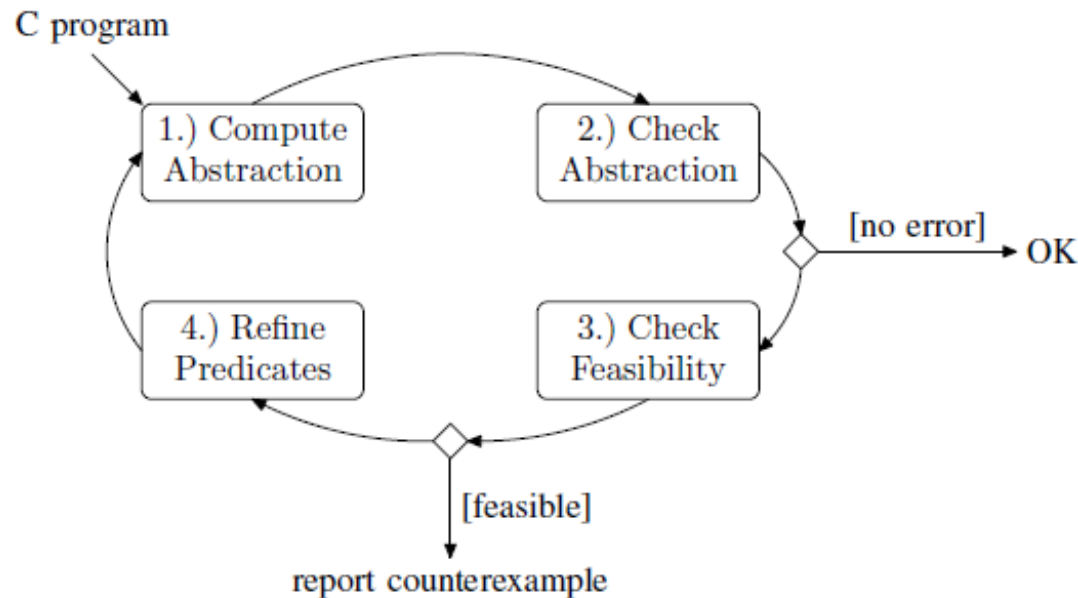
# Bevezetés

- Legelterjedtebb absztrakciós technika a szoftver modellellenőrzésben
  - Ld. a SLAM eszközkészlet sikere
- Absztrakció:
  - Logikai predikátumok a program állapottér particionálására
  - Különbség az absztrakt interpretációhoz képest: Itt a **programra specifikus** lehet az absztrakció
  - Kihívás: A megfelelő predikátumok megtalálása
- Absztrakció finomítás:
  - Modellellenőrzés az absztrakt állapottéren: **Hamis ellenpélda** adódhat
  - Hamis ellenpélda alapján újabb predikátumok: Precízebb absztrakció

# Ellenpélda által irányított absztrakció finomítás

## Counterexample Guided Abstraction Refinement

1. Absztrakció
2. Verifikáció: Ellenőrzés az absztrakt programon
3. Szimuláció: Az ellenpélda vizsgálata
4. Predikátumfinomítás, ha hamis ellenpélda volt

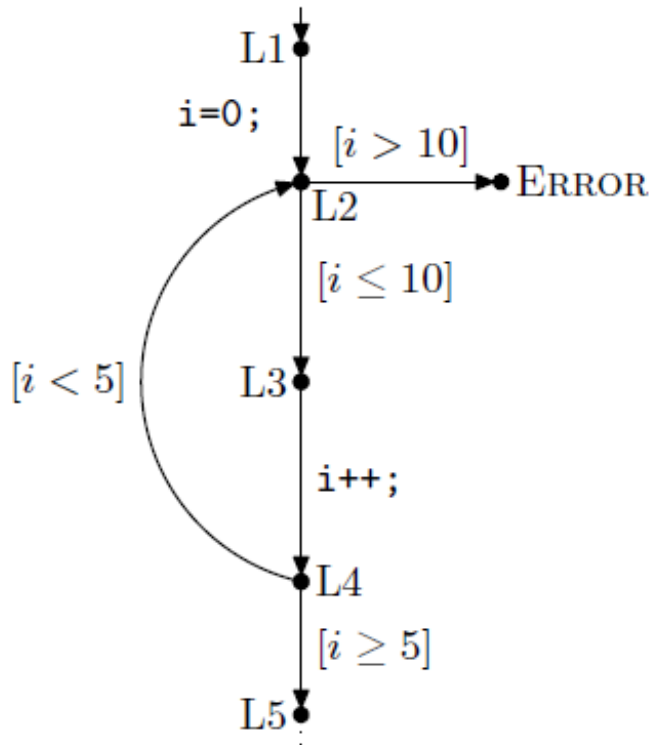


# 1. lépés: Predikátum absztrakció

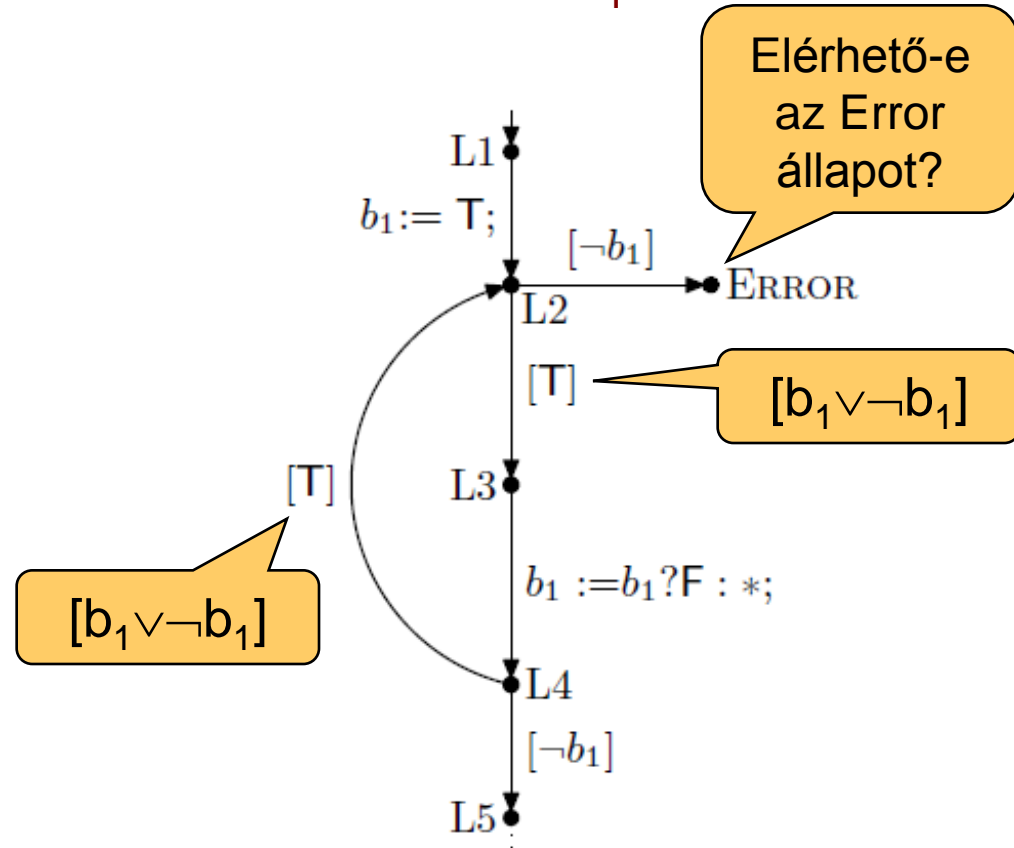
- Program (programgráf):
  - L1, L2, ... programhelyek utasításokkal
  - $R_{L_1}, R_{L_2}, \dots$  tranzíció relációk programállapotok között  $L_i$  végrehajtásával
  - Ezek uniója: R tranzíció reláció programállapotok között
- Predikátum absztrakció: Egy  $R'$  konstruálása
  - Predikátumok felírása a program változóira (pl.  $i==0$ )
  - Állapotok particionálása: Predikátumok igaz/hamis értéke alapján
    - A partíciók definiálják az absztrakt állapotokat
    - Átmenetek az absztrakt állapotok között:
      - Csak ha van átmenet a megfelelő konkrét állapotok között
    - Eredmény: Boole program (predikátum értéke Boole változóba)
      - Csak Boole változók, de eredeti vezérlési utasítások (függvények is)
  - Egzisztenciális jellegű absztrakció a programon, biztonságos az elérhetőségi tulajdonságokra
    - Ami a konkrét programban elérhető, az az absztraktban is elérhető (de az absztrakt programban több útvonal lehetséges)

# 1. Predikátum absztrakció

Konkrét program (int i):



Absztrakció  $b_1$  változóra:



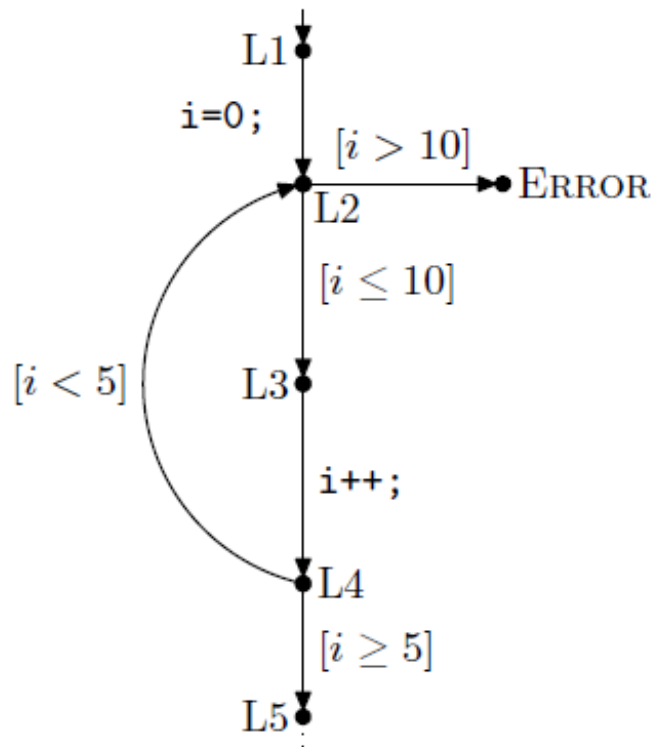
Predikátum:  $(i==0)$

A predikátum értéke  $b_1$  változó értéke

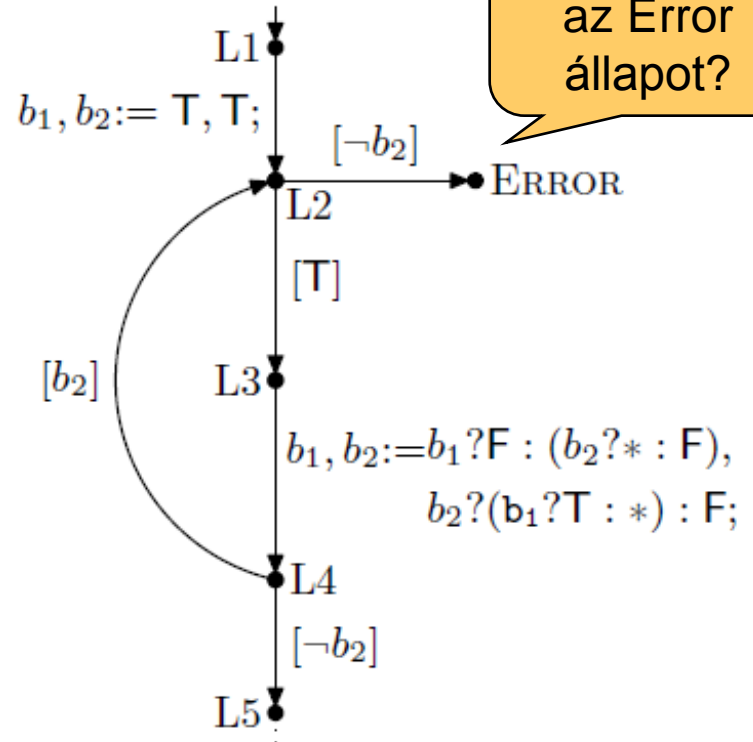
\* jelöli az ismeretlen értéket

# Finomabb absztrakció

Konkrét program (int i):



Absztrakció  $b_1$ ,  $b_2$  változóira:



Elérhető-e az Error állapot?

Predikátum  $b_1$ -hez:  $(i==0)$

Predikátum  $b_2$ -höz:  $(i<5)$



# Absztrakt program származtatása

- **Automatikus absztrakció: Tranzíció reláció számítása**
  - El kell dönteni, hogy egy absztrakt állapotból az adott utasítással milyen absztrakt állapotba juthatunk el
    - Ez határozza meg a Boole változók értékének változását
    - Pl.  $L3 \rightarrow L4$  esetén ( $i++$  utasítás):  
 $\neg(i==0) \wedge \neg(i<5)$  –ből nem lehet tranzíció  $\neg(i==0) \wedge (i<5)$  állapotba
- **Hátrány:**
  - $n$  predikátum esetén  $2^n$  absztrakt állapot lehet
  - $(2^n)^2$  döntés kell az absztrakt állapotok közötti absztrakt tranzíció reláció számításához
- **Közelítés: Cartesian abstraction**
  - Minden predikátumra külön-külön absztrakció számítás
  - Az eredményül kapott absztrakt relációk szorzata
- **Döntések számítása:**
  - Elsőrendű logikai tételbizonyítók, aritmetikával kombinálva
    - Pl. ZAPATO, Simplify
  - SAT (SMT) megoldók bit-szintű formula kezeléshez

## 2. lépés: Verifikáció

- Elérhetőségi probléma: Boole programokra eldönthető
  - Technológia: Szimbolikus modellellenőrzők
    - BDD alapú megvalósítások
    - SAT alapú eszközök: Csak korlátos ellenőrzésre
    - Speciális eszközök:
      - Quantified Boolean Formulas (QBF) megoldók
  - Példa: Error nem elérhető az előbbi konkrét programban
- Modellellenőrzés az absztrakt programokon:
- $P=\{b_1\}$  absztrakció esetén, ahol  $b_1: (i==0)$ 
    - L2 állapot esetén fennálló absztrakt állapotok:  $b_1, \neg b_1$
    - Így Error állapot elérhető (ez **hamis ellenpélda**)
  - $P=\{b_1, b_2\}$  absztrakció esetén, ahol  $b_1: (i==0), b_2: (i<5)$ 
    - L2 állapot esetén fennálló absztrakt állapotok:  $b_1 \wedge b_2, \neg b_1 \wedge b_2$
    - Így Error állapot nem érhető el

### 3. lépés: Szimuláció

- Modellellenőrzés eredménye:
  - Nincs ellenpélda: OK (konkrét programban sincs)
  - Van ellenpélda: **Hamis** lehet
- Ellenpélda (útvonal) ellenőrzése
  - Szimuláció a konkrét programon
  - Hamis ellenpélda: Nem bejárható útvonal
- Példa az útvonal ellenőrzésére
  - $P=\{b_1\}$  esetén (L1, L2, L3, L4, L2, Error) útvonal
    - Az absztrakt programban bejárható (Error elérhető)
    - Konkrét programban nem járható be
- Technológia: Szimbolikus szimuláció
  - Absztrakt állapot „terjesztése” az ellenpélda alapján bejárva a program állapotokat

## 4. lépés: Absztrakció finomítás

- Hamis ellenpéldák forrásai:
  - Hamis **útvonal**: A predikátumok nem elégségesek az állapotok megfelelő megkülönböztetéséhez
  - Hamis **átmenetek**: Közelítő absztrakció (Cartesian)
- Finomítás kivitelezése:
  - Hamis útvonalra:
    - További predikátumok (heurisztikus) hozzáadása
  - Hamis átmenetekre:
    - Kényszerek hozzáadása az absztrakt átmenet relációhoz
- Példa a hamis útvonalra:
  - $P=\{b_1\}$  absztrakció esetén (L1, L2, L3, L4, L2, Error) hamis ellenpélda
    - Itt ( $i=0$ ) és ( $i<5$ ) predikátumok elégségesek, hogy az absztrakt programban elkerülhető legyen ( $b_1$ ,  $b_2$  absztrakció)

# Absztrakciót támogató eszközök

## Statikus analízis, modellellenőrzés, BMC

Tool name	Tool developer						Languages
		Symbolic analysis	Abstraction	Counterexample	BMC	Concurrency	
ASTRÉE	École Normale Supérieure	×	×				C (subset)
CODESONAR	Grammtech Inc.	×	×				C, C++, ADA
PolySpace	PolySpace Technologies	×	×			×	C, C++, ADA, UML
PREVENT	Coverity	×	×			×	C, C++, Java
BLAST	UC Berkeley/EPF Lausanne	×	×	×		×	C
F-SOFT (abs)	NEC	×	×	×			C
Java PathFind.	NASA	×		×	×	×	Java
MAGIC	Carnegie Mellon University	×	×	×		× <sup>1</sup>	C
SATABS	Oxford University	×	×	×		×	C, C++, SpecC, SystemC
SLAM	Microsoft	×	×	×		×	C
SPIN	Bell Labs <sup>2</sup>			×	×	×	PROMELA, C <sup>3</sup>
ZING	Microsoft Research			×	×	×	ZING (object oriented)
CBMC	CMU/Oxford University	×		×	×		C, C++, SpecC, SystemC
F-SOFT (bmc)	NEC	×		×	×		C
EXE	Stanford University	×		×	×		C
SATURN	Stanford University	×		×	×		C

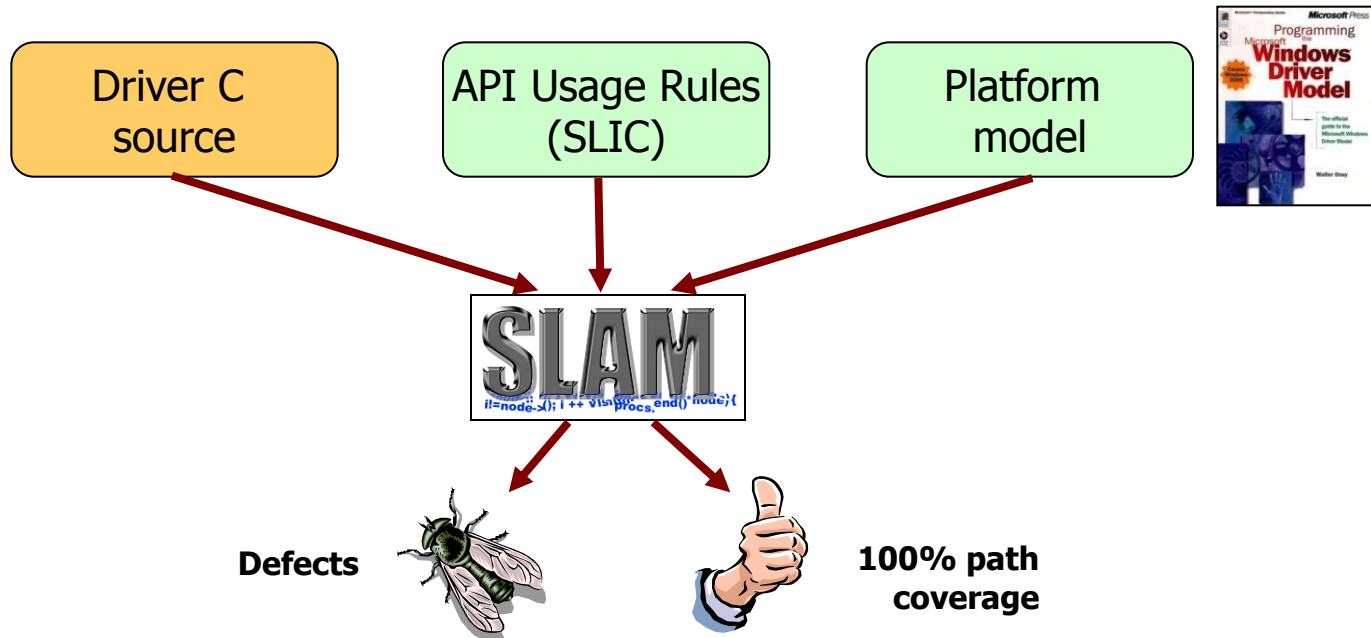
# Eszköz példa: SLAM

Software, Languages, Analysis and Model checking project  
SDV 1.0 (Static Driver Verifier, 2002)  
SLAM 2 (WDK kiegészítéseként)

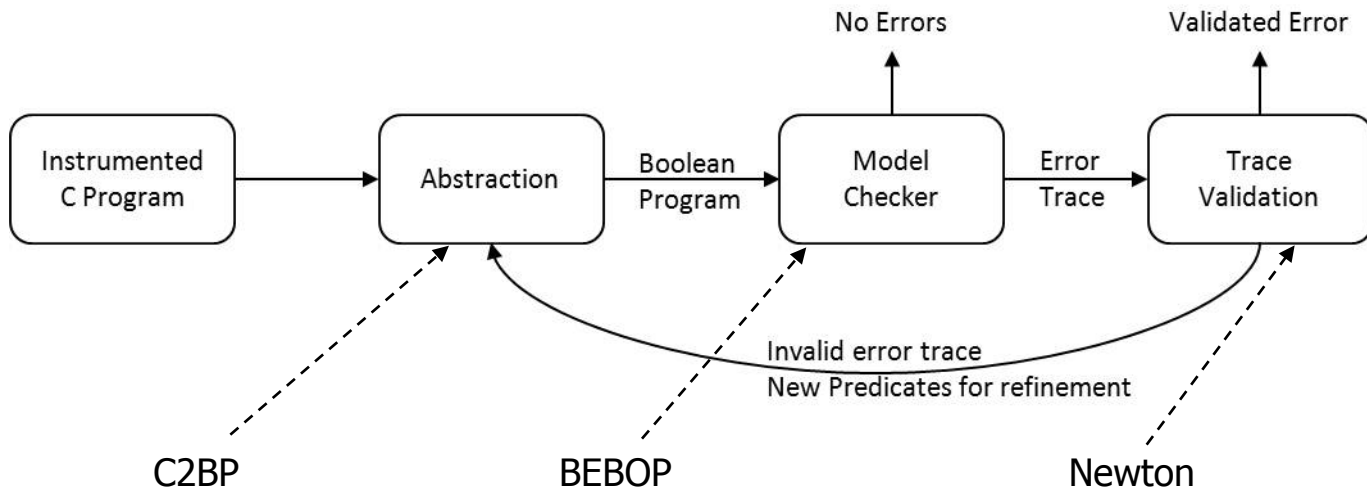
Ld: Kiss Ákos referátuma, 2010.

# A SLAM célkitűzései

- Motiváció: Hibás driverek megzavarhatják az OS működését
  - PI: hibás zárolások erőforrásokon (nincs zárolva, nincs felszabadítva, ...)
- Megoldás:
  - Szabályokkal leírható a helyes használat (pl. zárhasználat „állapotgépe”)
  - Ennek teljesülését ellenőrzi a SLAM a forráskódon
  - Forráskód absztrakciót használ: **Boole-program** állapotainak vizsgálata
  - Hibás (szabályokhoz nem illeszkedő) utakat vissza kell ellenőrizni!



# A SLAM működése



```
do {  
  
    AcquireLock (&devices->writeListLock);  
  
    deviceNoOld = deviceNo;  
    more = devices->Next;  
  
    if (more) {  
        ReleaseLock (&devices->writeListLock);  
        ...  
        deviceNo++;  
    }  
} while (deviceNoOld != deviceNo);  
  
ReleaseLock (&devices->writeListLock);
```

## Predikátum absztrakció:

```
do {  
    deviceNoOld = deviceNo;  
    more = devices->Next;  
    if (more) {  
        deviceNo++;  
    }  
} while (deviceNoOld != deviceNo);
```

```
do {  
    b = true;  
  
    if (*) {  
        b = b ? false : *;  
    }  
} while (!b);
```



# Eszköz példa: BLAST

Berkeley Lazy Abstraction Software Verification Tool  
Open Source (Apache License 2.0)

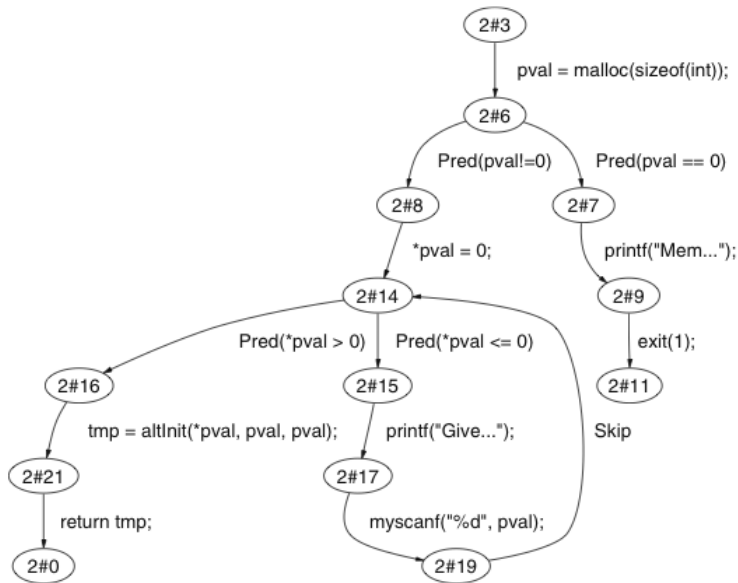
Ld: Marosi Attila Csaba referátuma, 2010.

# A BLAST célkitűzése

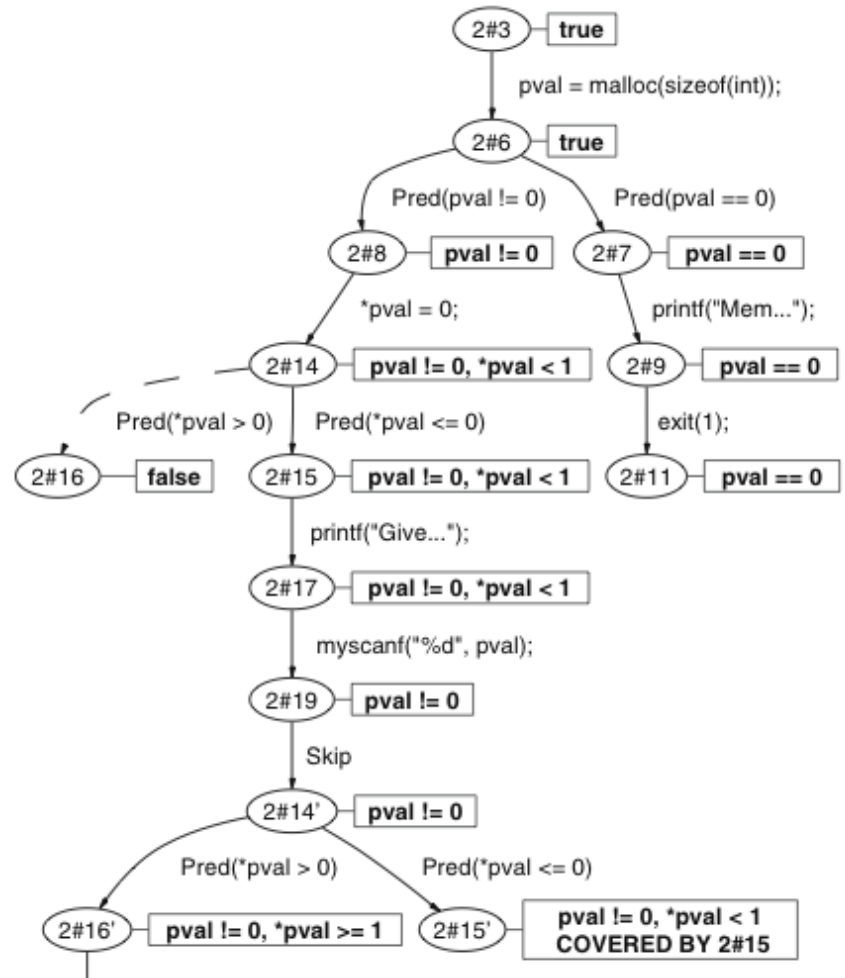
- C programok elérhetőségi tulajdonságának automatikus ellenőrzése
  - C forráskódból modell generálása
  - Megadott követelmények alapján modellellenőrzés
- Bemenet: Címkézett C forráskód
  - Blast Query Language monitorkifejezés
- Kimenet:
  - Jelentés a követelmény teljesüléséről
  - Ellenpélda (trace)
  - Sorozatos finomítás (korlát: memória, futásidő)

# A BLAST működése

## Control flow automata



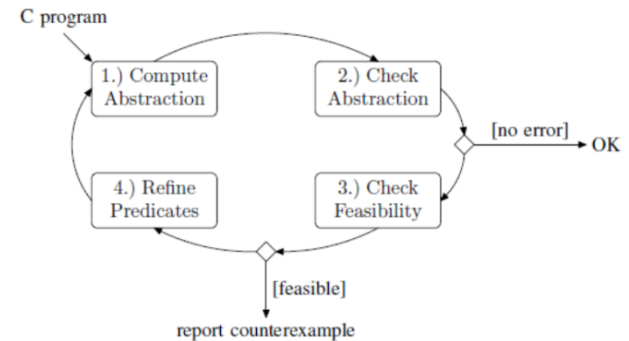
## Abstract reachability tree



Boole kifejezések

# A BLAST működése

- CEGAR: Counterexample Guided Abstraction Refinement



- Lusta predikátum absztrakció:

- Csak a hibához vezető úton található csúcsoknál történik az Abstract Reachability Tree finomítása

- Korlátozások:

- Szorzás és bit-műveleteket figyelmen kívül hagyja (hamis hibát jelezhet)
- Integer overflow-t nem figyel
- (Függvény)mutatókat figyelmen kívül hagyja
- Feltételezi, hogy minden mutató aritmetika biztonságos
- Nem támogat rekurzív függvényeket

# Eszköz példa: ZING

Systematic State Space Exploration Infrastructure for Software  
Microsoft

Ld: Dudás Ákos referátuma, 2010.

# A ZING célkitűzése

- Állapottér vizsgálat korlátos modellellenőrzéssel magas szintű nyelven leírt konkurens OO programokra
  - Holtpont ellenőrzése
  - Assertion ellenőrzés
  - Exception ellenőrzés
  - Ekvivalencia ellenőrzés (szimuláció reláció két modell között)
- ZING nyelvű modell
  - Függvényhívások (stack)
  - Dinamikusan létrejövő objektumok
  - Dinamikusan létrejövő folyamatok
    - Üzenetküldés vagy megosztott memória használata

# A ZING nyelv: Példa az étkező filozófusokra

```
class Fork {
  Philosopher holder;

  void PickUp(Philosopher eater){
    atomic {
      select {
        wait( holder==null ) ->
          holder = eater; }
    }
  }

  void PutDown(){
    holder = null;
  };
}
```

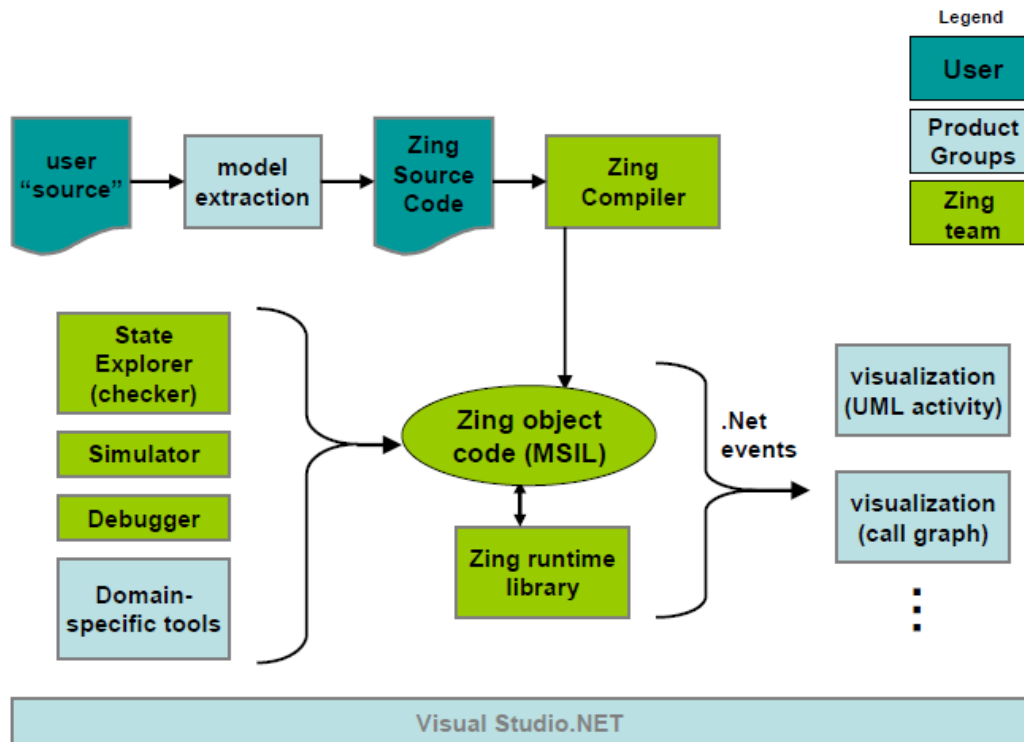
```
class Philosopher {
  Fork leftFork;
  Fork rightFork;

  void Run() {
    while (true) {
      leftFork.PickUp(this);
      rightFork.PickUp(this);

      leftFork.PutDown();
      rightFork.PutDown();
    };
  }
}
```

# A ZING architektúrája

1. (Forráskód →) Zing modell
2. Zing compiler: MSIL kód áll elő
  - LTS reprezentáció (Zing Object Model, ZOM)
3. Állapottér bejárás ezen a modellen





# A ZING eszköz

The screenshot shows the Zing Model Viewer interface for the 'Philosophers' model. The 'Trace' window on the left lists process identifiers (p3, p1, p2, p4). The 'State detail' window is open to the 'General' tab, displaying the following information:

- State Type: Error
- Fingerprint: 402307eb:50fb8828:63c440b2:7feef37a
- Error: (Deadlock: n processes are runnable and one or more processes is blocked in an invalid end state)

A red box highlights the word 'Deadlock' in the error message, with a red arrow pointing to it from a larger red box containing the word 'Deadlock'. The bottom status bar indicates 'Verification failed'.

The screenshot shows the Zing Model Viewer interface for the 'PhilosophersJav' model. A 'State exploration... complete' dialog box is displayed with the following statistics:

Statistics			
State transitions:	1219359	Elapsed time:	00:00:56
Distinct states:	710670	States / sec :	21591
Maximum depth:	155	Current depth:	0

An 'OK' button is visible at the bottom of the dialog box. The bottom status bar indicates 'Verification passed'.

# A ZING állapottér kezelésének érdekes elemei

- **Állapotok hash kódolása (lenyomata)**
  - Könnyebb egyezés vizsgálat
- **Atomi utasítás blokkok**
  - Közben más folyamat állapotváltására nem kell figyelni
- **Függvények gyorsítárasa**
  - Bemenet és előidézett változások megjegyzése
  - Ugyanilyen bemenetre nem kell ismétetni az elemzést
- **Állapotváltozás gyorsítárasa**
  - Állapotváltáskor elmenti a változást két állapot között
  - Ugyanilyen esetben nem kell generálni az állapotvektort
- **Kézi „assume” állítások**
  - Állapottér vágás