A formális módszerek szerepe

dr. Majzik István
dr. Bartha Tamás
dr. Patariczza András

BME Méréstechnika és Információs Rendszerek Tanszék

Mik a formális módszerek?

Definíciók, formalizmusok

Mik azok a formális módszerek?

- Matematikai technikák használata,
  - elsősorban diszkrét matematika
  - és matematikai logika,

  hogy hardver és szoftver rendszerek
  - specifikációját,
  - terveit (modelljeit),
  - implementációját,
  - dokumentációját

  elkészítsük és ellenőrizzük.

Első lépés: Leírás (modellezés) formális nyelven

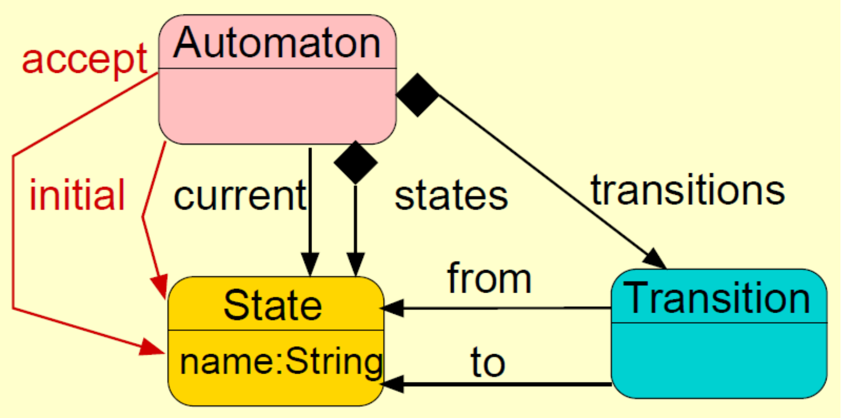
- A formalizálás célja: Matematikai precizitású megadás
  - Tervek: modellek, tervezői döntések (modellezési nyelv)
  - Követelmények: elvárt tulajdonságok (követelmény leíró nyelv)
- Formális nyelvek felépítése
  - Formális szintaxis
    - Jelölésmód: milyen nyelvi elemek és kapcsolatok vannak?
  - Formális szemantika
    - A jelölésmód interpretációja: mit értek alatta?
- Milyen lehet a formális leírás?
  - Tipikusan absztrakt (implementáció-független)
  - Sokféle szempontú: struktúra, funkció, teljesítmény, biztonság, …
- A formális nyelv előnyei
  - Precizitás: Egyértelműség, ellenőrizhetőség
  - Automatikus feldolgozás: Ellenőrzés, leképezés, (kód)generálás

Formális szintaxis (áttekintés)

- Matematikai megadás:

$$KS = (S, R, L) \text{ és AP, ahol}$$
$$\text{AP} = \{\text{P,Q,R,...}\}$$
$$\text{S} = \{\text{s}_1, \text{s}_2, \text{s}_3, \ldots \text{s}_n\}$$
$$\text{R} \subseteq \text{S} \times \text{S}$$
$$\text{L: S} \rightarrow 2^{AP}$$

- BNF: BL ::= true | false | p∧q | p∨q
- Metamodell:



- Absztrakt szintaxis: nyelvtani szabályok
- Konkrét szintaxis: megjelenítés

Formális szemantika (áttekintés)

A szintaxis alapján felírt modellek jelentése (mit értünk alatta):

- Műveleti (operációs) szemantika: „Programozóknak”
  - Megadja, mi történik a végrehajtás (számítások) során
  - Egyszerű elemekre épít: pl. állapotok, események, akciók
  - Pl. ellenőrzéshez az állapottér felvételéhez
- Axiomatikus szemantika: „Helyességbizonyításhoz”
  - Állítás nyelv + axiómakészlet + következtetési szabályok
  - Pl. automatikus tételbizonyító rendszerekhez
- Denotációs szemantika: „Fordítóprogramokhoz”
  - Szintaxis által meghatározott leképzés egy ismert doménre
    - Ismert matematikai domén, pl. számítási szekvencia, vezérlési gráf, állapothalmaz, … és ezeken definiált műveletek (összefűzés, unió, …)
    - A modellek vizsgálata: a mögöttes matematikai domén vizsgálata
  - Pl. kódgeneráláshoz is

Második lépés: A formális modell használata

- A formális modell végrehajtása
  - Szimuláció
- A formális modell ellenőrzése: Formális verifikáció
  - A modell „önmagában való” vizsgálata
    - Konzisztencia, ellentmondás-mentesség
    - Teljesség, zártság
  - A modell és a követelmények összevetése
    - Elvárt tulajdonságok teljesülnek (terv ↔ specifikáció)
    - A modellfinomítás megtartja a tulajdonságot (eredeti ↔ finomított terv)
- A formális modell alapján történő szintézis:
  - Szoftver (programkód, konfiguráció) generálása
  - Hardver implementáció generálása
  - Dokumentáció generálása

Mire jók a formális módszerek egy informatikus mérnök számára?

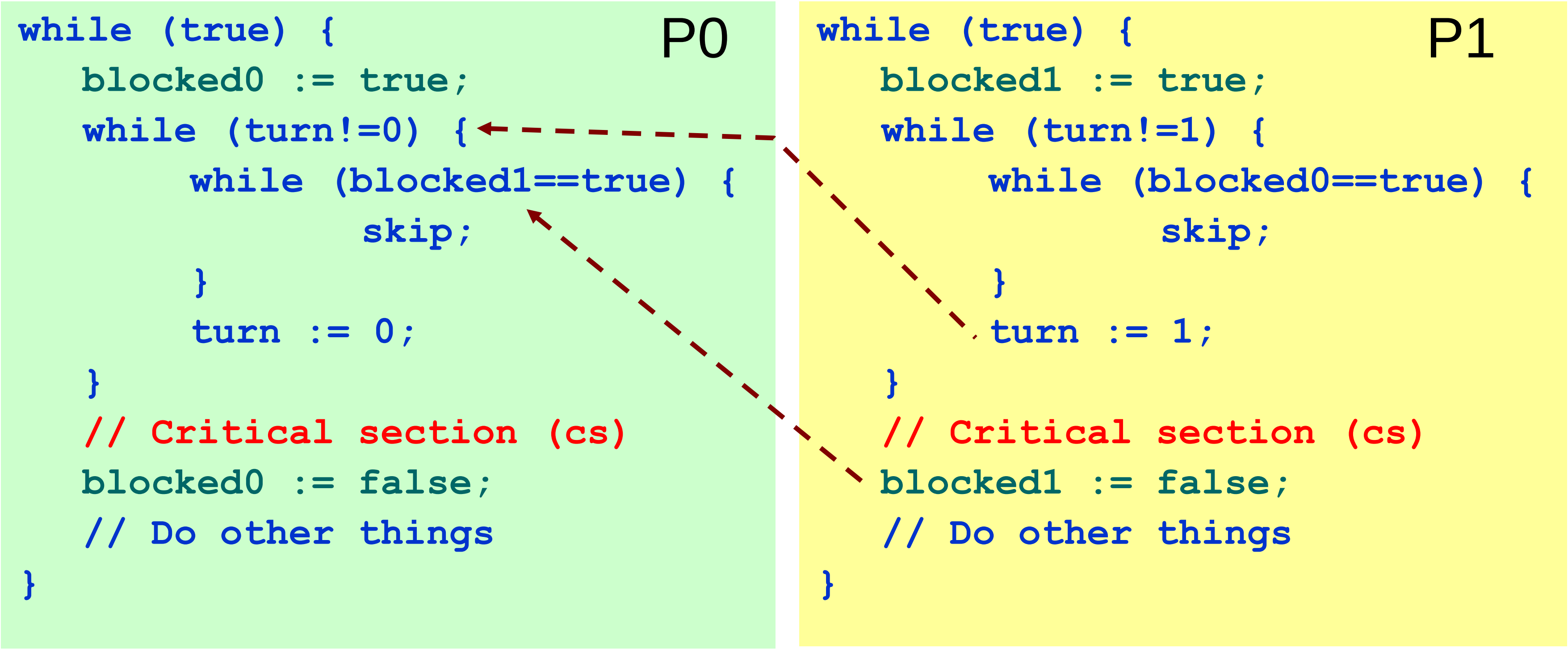
Egy jellegzetes példa:
Algoritmus helyességének ellenőrzése

Egy mérnöki feladat

- Több processzből álló alkalmazás
- Cél: Egy hardver erőforráshoz egyszerre csak egy processz férhessen hozzá (kölcsönös kizárás kell)
  - Példa: Kommunikációs csatorna használata
  - Ez védendő „kritikus szakasz” a programban
  - A platform (OS, futtató rendszer) nem ad ehhez támogatást: nincs szemafor, monitor, stb.
  - Csak megosztott változók (egy művelettel olvashatók vagy írhatók) használhatók a kritikus szakasz védelmére
- Hogyan valósítsuk meg?
  - Klasszikus megoldások
  - Saját algoritmus

Egy kölcsönös kizárás algoritmus

- 2 résztvevőre, 3 megosztott változóval (H. Hyman, 1966)
  - **blocked0**: Első résztvevő (P0) be akar lépni
  - **blocked1**: Második résztvevő (P1) be akar lépni
  - **turn**: Ki következik belépni (0 esetén P0, 1 esetén P1)



P0

```
while (true) {
   blocked0 := true;
   while (turn!=0) {
        while (blocked1==true) {
                skip;
        }
        turn := 0;
   }
   // Critical section (cs)
   blocked0 := false;
   // Do other things
}
```

P1

```
while (true) {
   blocked1 := true;
   while (turn!=1) {
        while (blocked0==true) {
                skip;
        }
        turn := 1;
   }
   // Critical section (cs)
   blocked1 := false;
   // Do other things
}
```

Helyes-e ez az algoritmus?

Mikor helyes az algoritmus?

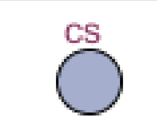
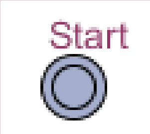
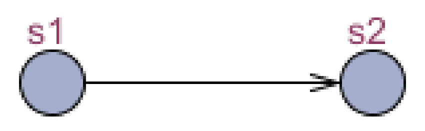
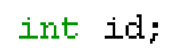
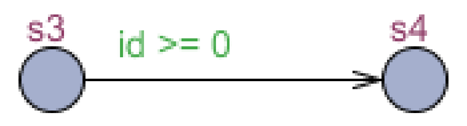
- Kölcsönös kizárás biztosított:
  - Egyszerre csak az egyik processz (P0 vagy P1) lehet a kritikus szakaszban
- Lehetséges az elvárt viselkedés:
  - P0 valamikor beléphet a kritikus szakaszba
  - P1 valamikor beléphet a kritikus szakaszba
- Nincs kiéheztetés:
  - P0 mindenképpen be fog lépni a kritikus szakaszba
  - P1 mindenképpen be fog lépni a kritikus szakaszba
- Holtpontmentesség:
  - Nem alakul ki kölcsönös várakozás (leállás)

Hogyan ellenőrizhetjük a helyességet?

- Megvalósítással és annak tesztelésével
  - Probléma: A processzek utasításai konkurensek (sokféle ütemezés szerint írhatják, olvashatják a megosztott változókat)
  - Létre tudunk-e hozni minden lehetséges konkurens végrehajtást lefedő tesztkészletet?
  - A problémás eset figyeléséhez külön ellenőrző kell
  - A hiba drágán javítható (csak az implementáció után derül ki)
- Modellezéssel és a modell szimulációjával
  - Tudunk-e szimulálni minden lehetséges konkurens végrehajtást?
  - A problémás esetek detektálása odafigyelést igényel
  - A hibák viszont olcsóbban javíthatók modell szinten
- Itt javasolt: Formális modell készítése és ellenőrzése
  - Automatikus ellenőrzés minden lehetséges konkurens végrehajtásra
  - Problémás esetek automatikus felderítése és bemutatása
  - A modellben követhetők (és javíthatók) az algoritmus hibái

Készítsünk formális modellt!

- Formális nyelv: Véges automata, változókkal

Szintaxis elemek	**Megjelenítés**
Állapot névvel, kezdőállapot	 
Állapotátmenet	
Változó	 `int id;`
Feltétel átmeneten	
Akció átmeneten	

A P0 processz formális modellje

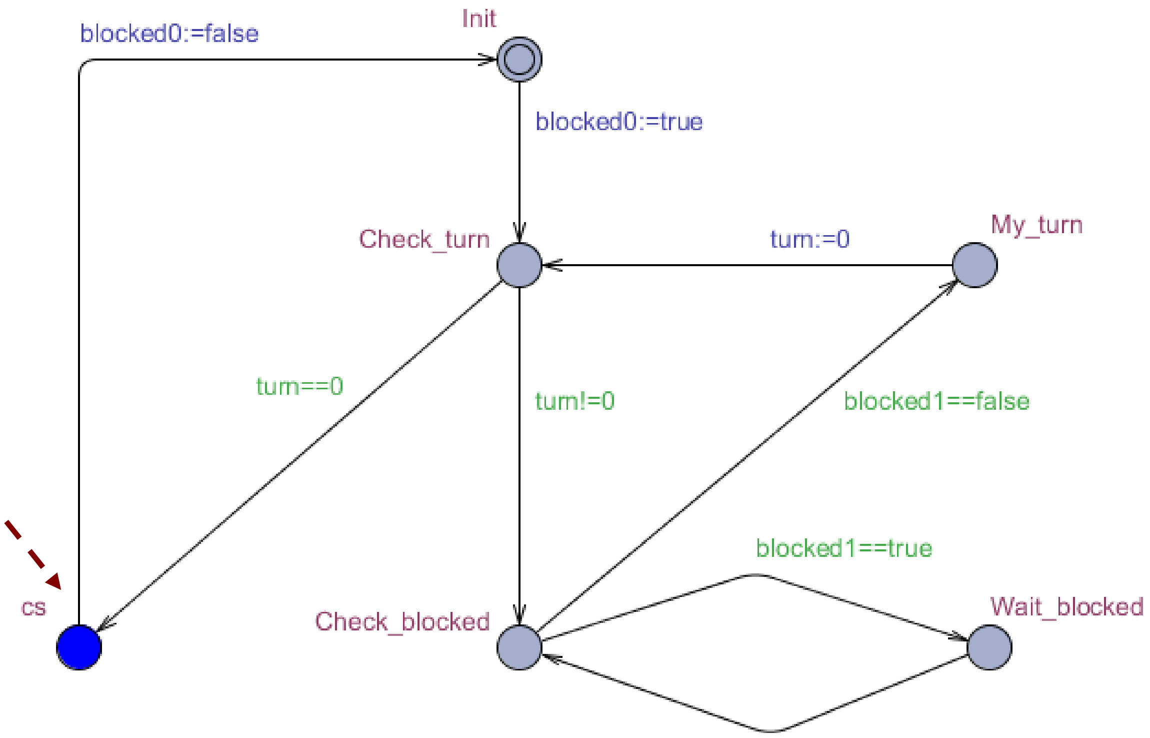
Változók deklarációi:

bool blocked0=false;
bool blocked1=false;
int[0,1] turn=0;
system P0, P1;

A P0 processz pszeudokód és automata modell:

P0

```
while (true) {
   blocked0 := true;
   while (turn!=0) {
        while (blocked1==true) {
                skip;
        }
        turn := 0;
   }
   // Critical section (cs)
   blocked0 := false;
   // Do other things
}
```



A P1 processz formális modellje

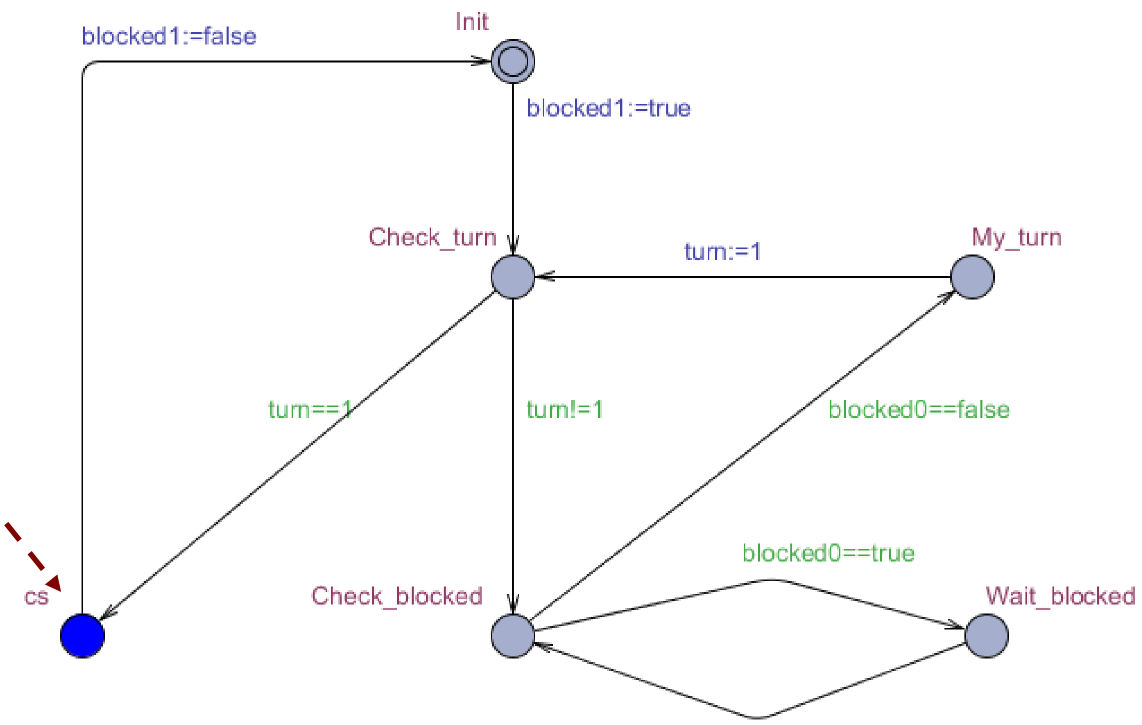
Változók deklarációi:

bool blocked0=false;
bool blocked1=false;
int[0,1] turn=0;
system P0, P1;

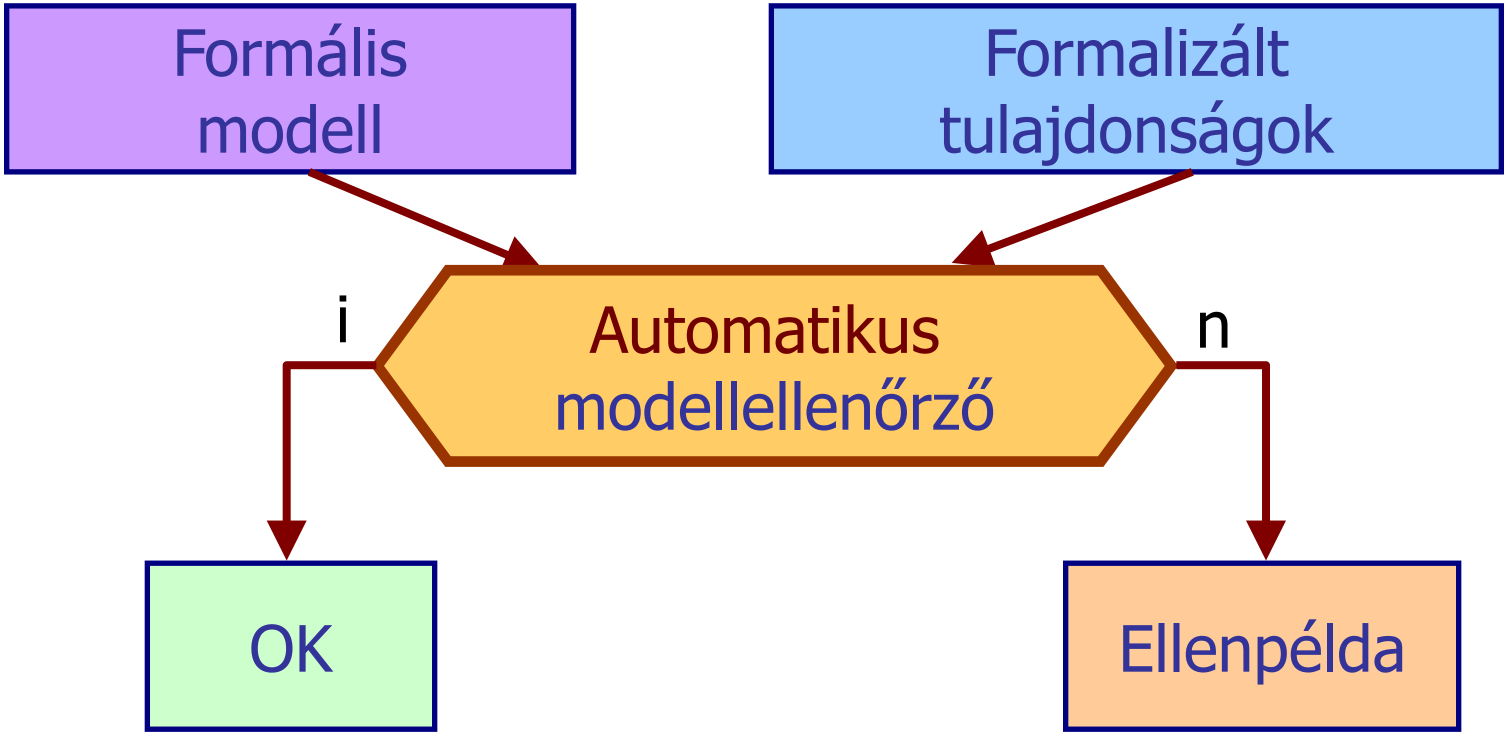
A P1 processz pszeudokód és automata modell:

P1

```
while (true) {
   blocked1 := true;
   while (turn!=1) {
        while (blocked0==true) {
                skip;
        }
        turn := 1;
   }
   // Critical section (cs)
   blocked1 := false;
   // Do other things
}
```

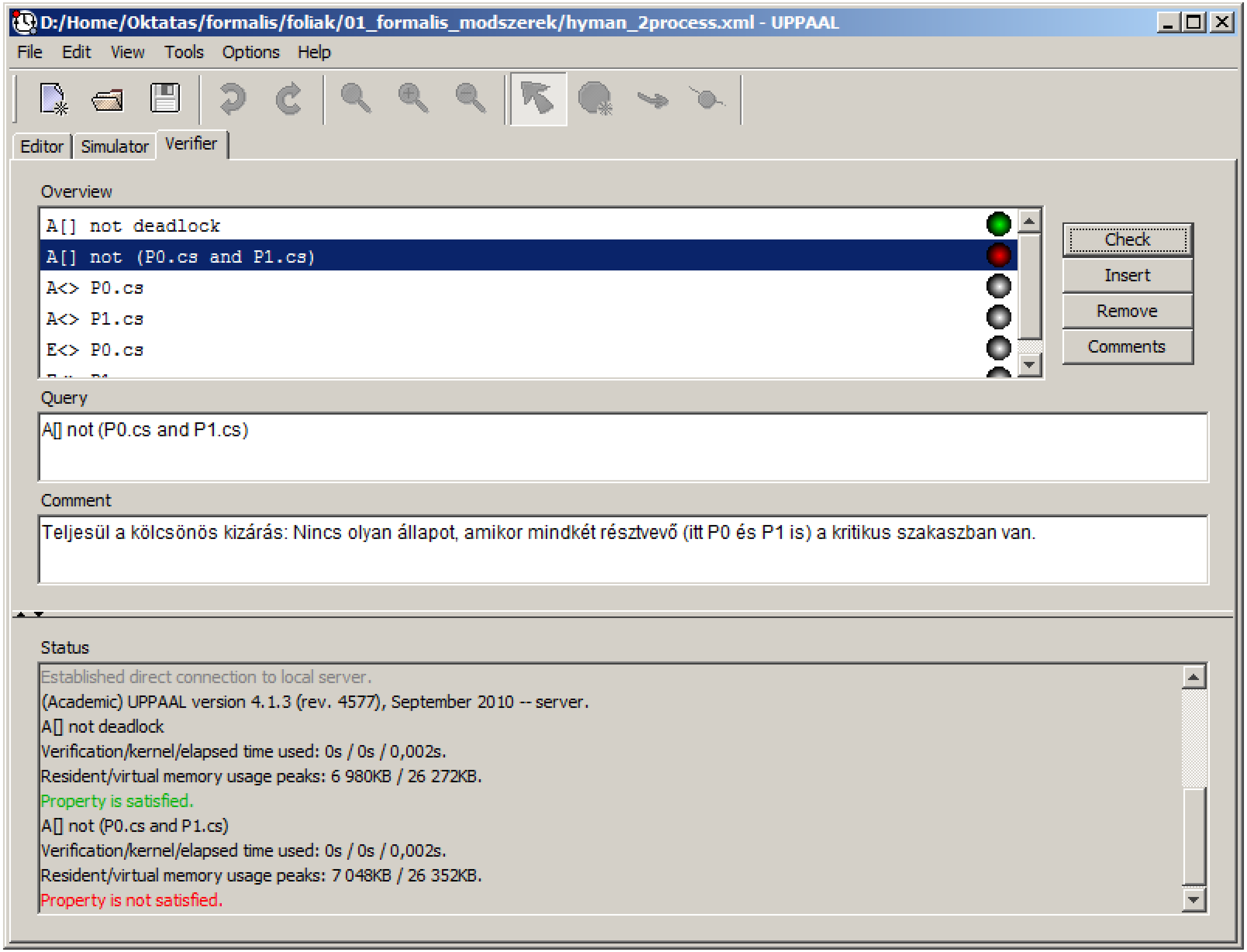


Egy jellegzetes formális ellenőrzés (verifikáció)



- Modellellenőrzés: A formális modell állapotterének teljes ellenőrzése a tulajdonságokat sértő viselkedés kereséséhez

Ellenőrzés egy gombnyomásra



Overview

```
A[] not deadlock                 ●
A[] not (P0.cs and P1.cs)        ●
A<> P0.cs                        ○
A<> P1.cs                        ○
E<> P0.cs                        ○
```

Query

A[] not (P0.cs and P1.cs)

Comment

Teljesül a kölcsönös kizárás: Nincs olyan állapot, amikor mindkét résztvevő (itt P0 és P1 is) a kritikus szakaszban van.

Status

```
Established direct connection to local server.
(Academic) UPPAAL version 4.1.3 (rev. 4577), September 2010 -- server.
A[] not deadlock
Verification/kernel/elapsed time used: 0s / 0s / 0,002s.
Resident/virtual memory usage peaks: 6 980KB / 26 272KB.
Property is satisfied.
A[] not (P0.cs and P1.cs)
Verification/kernel/elapsed time used: 0s / 0s / 0,002s.
Resident/virtual memory usage peaks: 7 048KB / 26 352KB.
Property is not satisfied.
```

Összefoglalás: Mit ígérnek a formális módszerek?

- Tervek, algoritmusok precíz leírása
  - Egyértelmű, konzisztens
  - Ellenőrzés, implementáció alapja lehet
- Tervek, algoritmusok helyességének ellenőrzése
  - Teljes körű ellenőrzés (helyesség igazolás)
  - Problémás esetek (hibás végrehajtás) felderítése
  - Korai hibajavítás
- A használathoz automatikus eszközök
  - Szintaxis ellenőrzése
  - Formális verifikáció (helyesség igazolás)
  - Kapcsolódás a fejlesztés további fázisaihoz: forráskód generálás, tesztgenerálás, dokumentáció

Formális módszerek értékelése

Mikor használjuk?

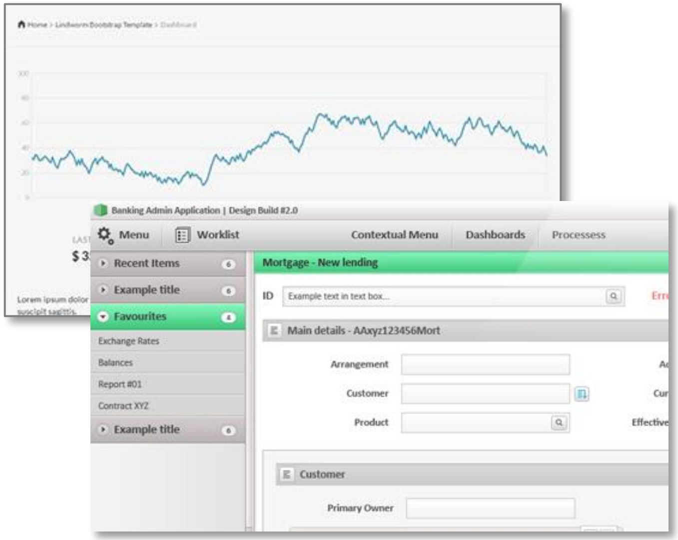
Mik a korlátok?

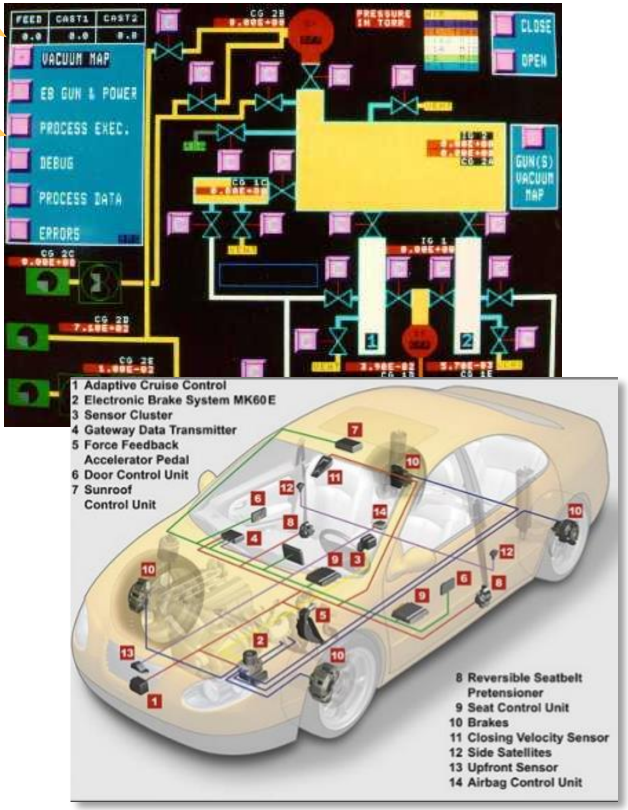
Mikor lehet sikeres?

Kiemelt szerep: Kritikus rendszerek tervezése

Bennmaradó hibák kockázata nagy

Konkurens, aszinkron működés





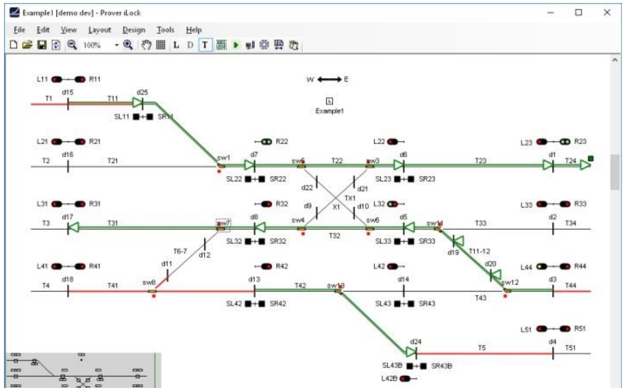
- Minimális tervezés
- Implicit folyamat
- Egyszerű eszközök

- Alapos tervezés
- Jól definiált folyamat
- Hatékony eszközök

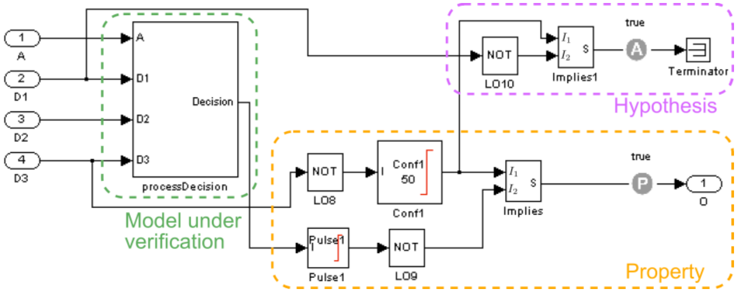
- Igazolt helyességű tervek
- Meghatározott folyamat
- Automatikus eszközök

Formális módszerek kritikus rendszerekhez

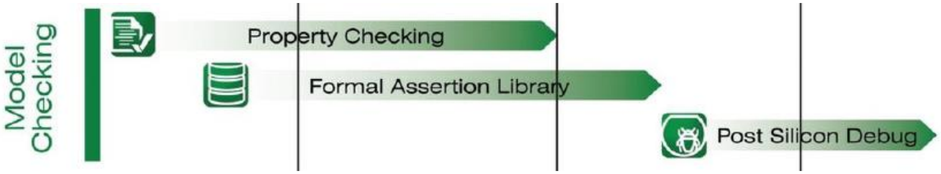
- iLock Tool Suite (Prover)
  - Formális nyelv (HLL) vasúti biztosítóberendezésekhez
  - Formális verifikáció és kódgenerálás



- SCADE Suite (Esterel Tech.)
  - Biztonságigazolt „szoftvergyár” beágyazott vezérlőkhöz



- Questa Formal Verification Tool (Mentor Graphics)
  - RTL szintű hardver tervek ellenőrzése



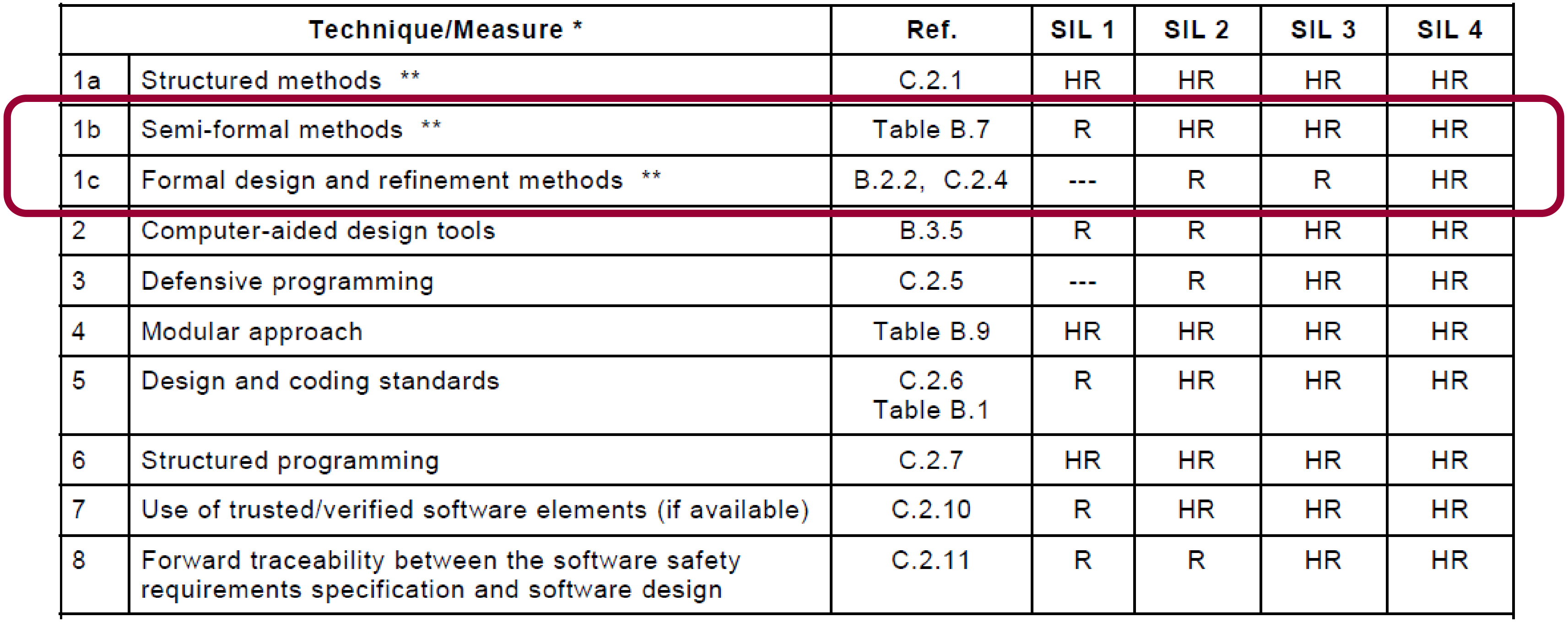
- LDRA Tool Suite (LDRA)
  - Szoftver adatfolyam analízis

Szabvány előírások biztonságkritikus szoftverekhez

- Biztonságintegritási szintek (SIL 1…4)
- IEC 61508: Szabvány előírások a fejlesztésre

Functional safety in electrical / electronic / programmable electronic safety-related systems

Software design and development - detailed design:



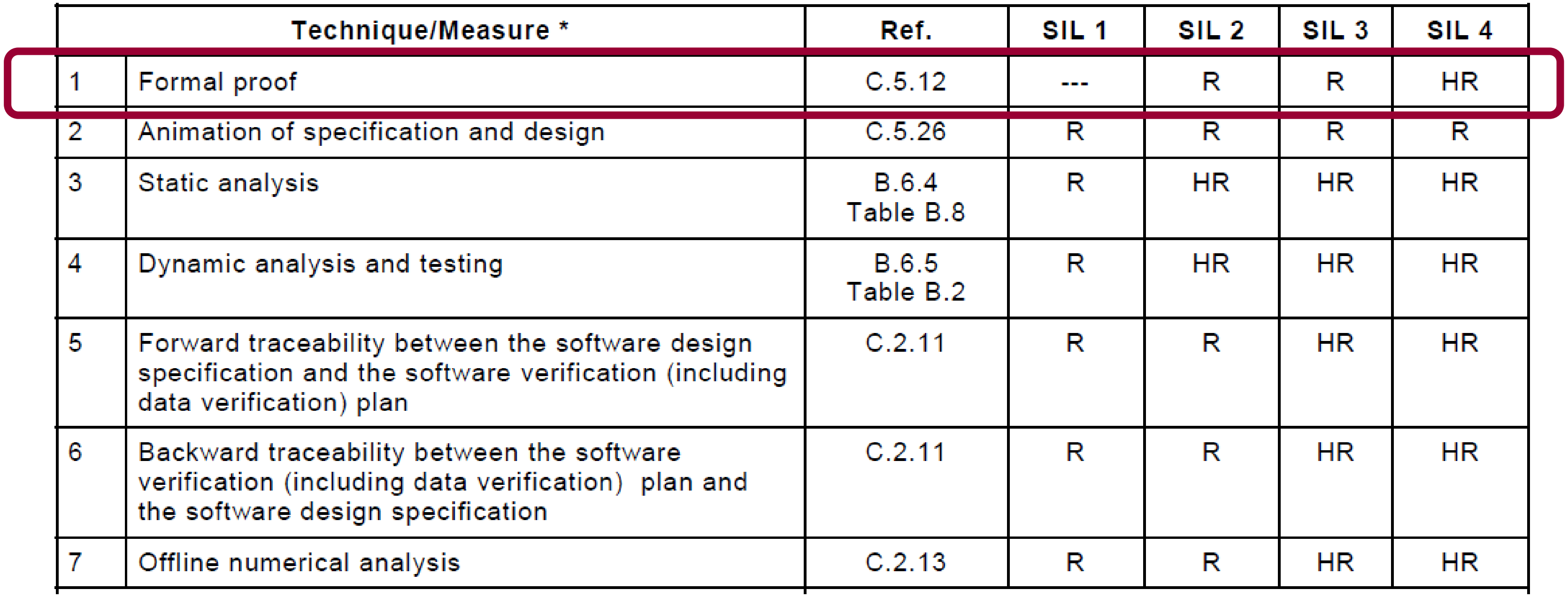
Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1a	Structured methods **	C.2.1	HR	HR	HR	HR
1b	Semi-formal methods **	Table B.7	R	HR	HR	HR
1c	Formal design and refinement methods **	B.2.2, C.2.4	---	R	R	HR
2	Computer-aided design tools	B.3.5	R	R	HR	HR
3	Defensive programming	C.2.5	---	R	HR	HR
4	Modular approach	Table B.9	HR	HR	HR	HR
5	Design and coding standards	C.2.6 Table B.1	R	HR	HR	HR
6	Structured programming	C.2.7	HR	HR	HR	HR
7	Use of trusted/verified software elements (if available)	C.2.10	R	HR	HR	HR
8	Forward traceability between the software safety requirements specification and software design	C.2.11	R	R	HR	HR

Szabvány előírások biztonságkritikus szoftverekhez

- Biztonságintegritási szintek (SIL 1…4)
- IEC 61508: Szabvány előírások a fejlesztésre

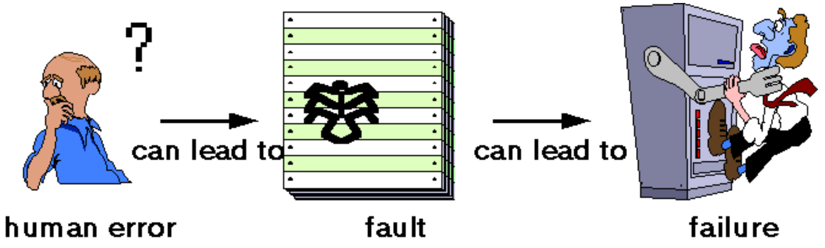
Functional safety in electrical / electronic / programmable electronic safety-related systems

Software verification:



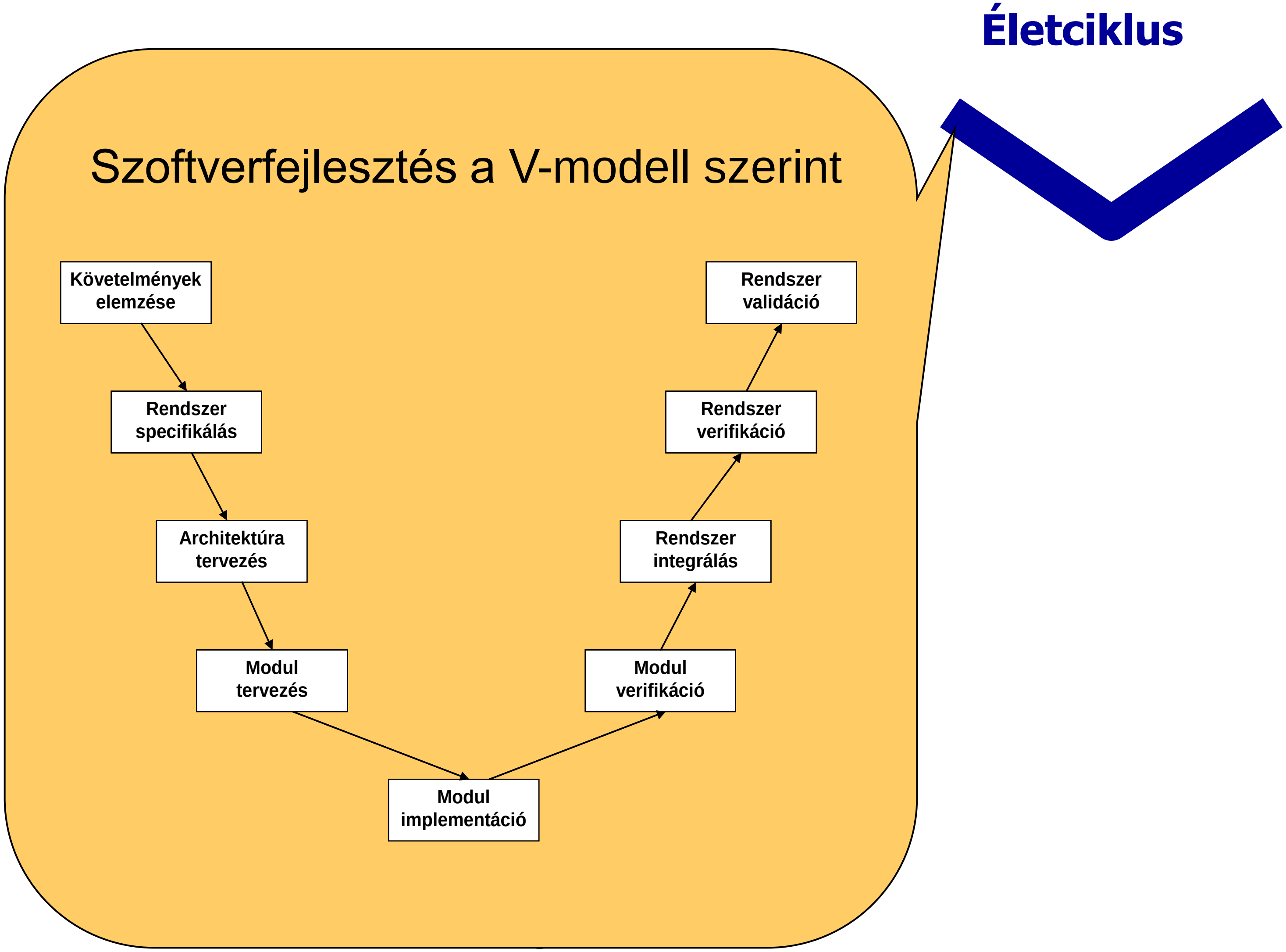
	Technique/Measure *	Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1	Formal proof	C.5.12	---	R	R	HR
2	Animation of specification and design	C.5.26	R	R	R	R
3	Static analysis	B.6.4 Table B.8	R	HR	HR	HR
4	Dynamic analysis and testing	B.6.5 Table B.2	R	HR	HR	HR
5	Forward traceability between the software design specification and the software verification (including data verification) plan	C.2.11	R	R	HR	HR
6	Backward traceability between the software verification (including data verification) plan and the software design specification	C.2.11	R	R	HR	HR
7	Offline numerical analysis	C.2.13	R	R	HR	HR

Cél: Kézi ellenőrzés költségeinek csökkentése



- Tipikus kódméret:
  - 10 kLOC … 1000 kLOC
- Fejlesztési ráfordítás:
  - Nagyméretű szoftver: 0,1 - 0,5 mérnökév / kLOC
  - Kritikus szoftver: 5-10 mérnökév / kLOC
- Hiba eltávolítás (ellenőrzés, tesztelés, javítás):
  - 45 - 75% ráfordítás a fejlesztési költségekből
  - Minél korábban detektálható egy hiba, annál olcsóbban javítható
  - Cél: Ellenőrzés már a specifikálás, tervezés során; automatikus eszközök használata
  - Formális módszerek ezt lehetővé teszik

V-től az Y fejlesztési modellig



Életciklus

Szoftverfejlesztés a V-modell szerint

- Követelmények elemzése
- Rendszer specifikálás
- Architektúra tervezés
- Modul tervezés
- Modul implementáció
- Modul verifikáció
- Rendszer integrálás
- Rendszer verifikáció
- Rendszer validáció

V-től az Y fejlesztési modellig

Életciklus

Ráfordítás megtakarítás

Kézi kódolás — 0%

Modell alapú automatikus kódgenerátor használata — -20%

Minősített automatikus kódgenerátor használata — -50%

Formális verifikációval kiegészített tervezés — -60%

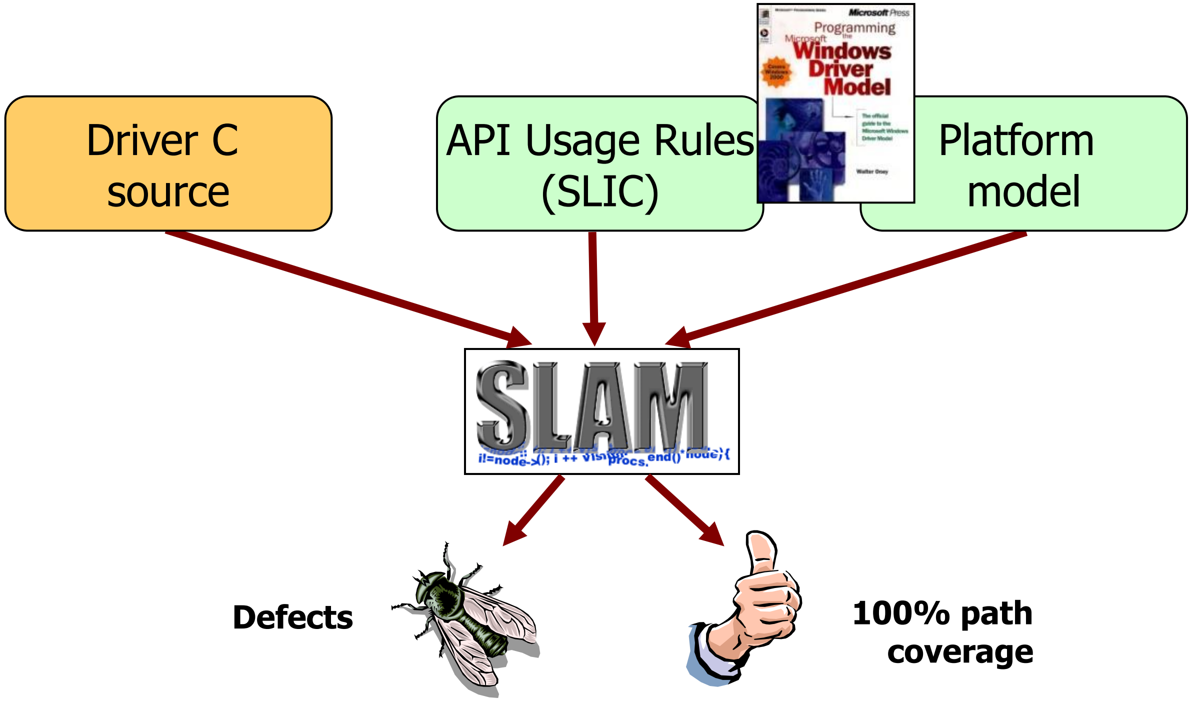
* Adatok: Esterel Technologies (Ansys)

(Formális) verifikáció nem váltja ki a validációt

Verifikáció (igazolás)	**Validáció (érvényesítés)**
„Jól építjük-e a rendszert?”	„Jó rendszert építettünk-e?”
Fejlesztési fázisok ellenőrzése: • A fejlesztési lépések eredménye (tervek, modellek, forráskód) megfelel a specifikációnak és az előző lépéseknek	Fejlesztés eredményének ellenőrzése • A kész rendszer megfelel a felhasználói elvárásoknak, kielégíti a felhasználói igényeket
Objektív folyamat; formalizálható, automatizálható	Szubjektív elvárások lehetnek; elfogadhatósági ellenőrzés
Felderíthető hibák: Tervezési, implementációs hibák	Felderíthető hibák: Követelmények hiányosságai is
Nincs rá szükség, ha automatikus a leképzés a specifikáció és az implementáció között	Nincs rá szükség, ha a specifikáció tökéletes (elég egyszerű)

Példa: OS komponensek fejlesztése SLAM eszközzel

- Motiváció: Hibás meghajtóprogramok megzavarhatják az OS működését
  - Pl.: 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
  - Megtalált hibás (szabályokhoz nem illeszkedő) utakat vissza kell ellenőrizni



Driver C source

API Usage Rules (SLIC)

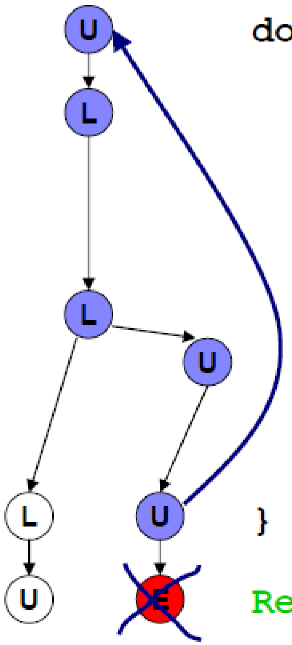
Platform model

Defects

100% path coverage

Példa: OS komponensek fejlesztése SLAM eszközzel

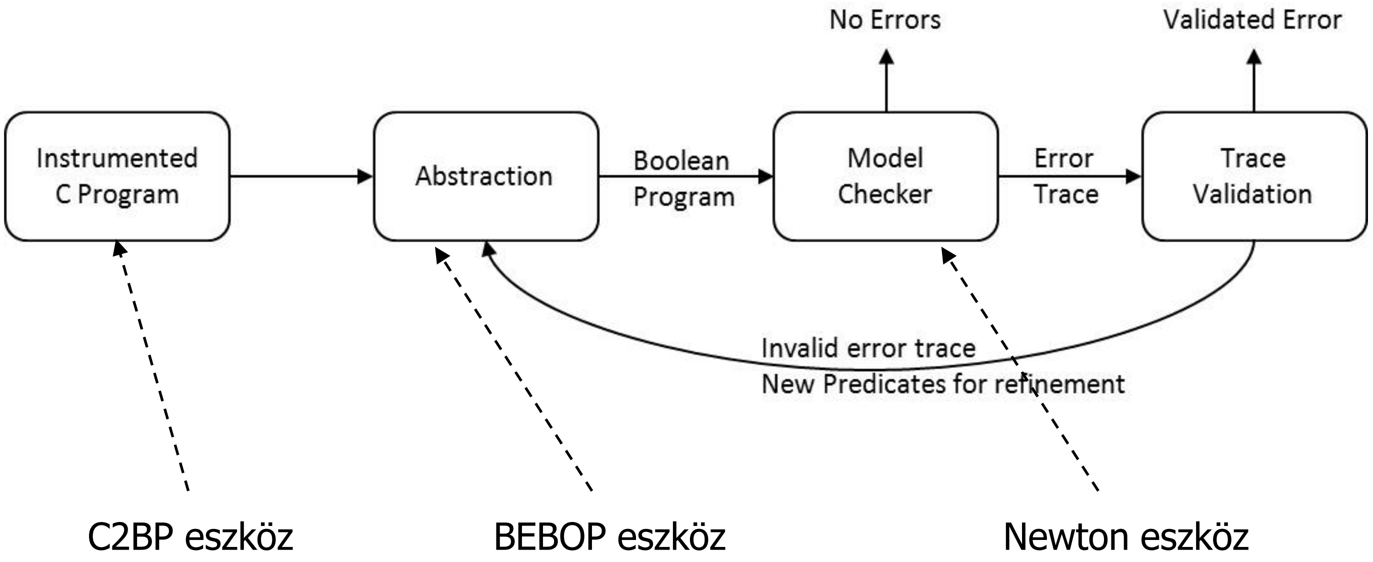
- Motiváció: Hibás meghajtóprogramok megzavarhatják az OS működését
  - Pl.: hibás zárolások erőforrásokon (nincs zárolva, nincs felszabadítva, …)
- Megoldás:
  - Szabályokkal leírható a helyes használat (pl. zárhasználat „állapotgépe”)
  - Ennek teljesülését ellenőrzi a SLAM a forráskódon
  - Forráskód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



```
do {

   AcquireLock(&devices>writeListLock);

   deviceNoOld = deviceNo;
   more = devices->Next;

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

ReleaseLock(&devices->writeListLock);
```

Példa: OS komponensek fejlesztése SLAM eszközzel

- Motiváció: Hibás meghajtóprogramok megzavarhatják az OS működését
  - Pl.: 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
  - Megtalált hibás (szabályokhoz nem illeszkedő) utakat vissza kell ellenőrizni


No Errors

Validated Error

Instrumented C Program

Abstraction

Boolean Program

Model Checker

Error Trace

Trace Validation

Invalid error trace

New Predicates for refinement

C2BP eszköz

BEBOP eszköz

Newton eszköz

Példák: Sikeres alkalmazások

- USA TCAS-II forgalomirányító rendszer
  - RSML nyelven specifikált; teljesség és ellentmondás-mentesség ellenőrzése
- Philips Audio Protocol
  - 1994: manuális verifikáció, majd 1996: automatikus ellenőrzés (HyTech)
- Lockheed C130J repülési szoftvere
  - Programfejlesztés helyességbizonyítással (CORE nyelv + Ada)
  - Költség nem nőtt a tesztelés egyszerűsödése miatt
- IEEE Futurebus+ szabvány
  - Carnegie Mellon SMV: cache koherencia protokoll hibájának kiderítése
- Hardver projektek: Motorola DSP, AMD 5K86
  - Motorola DSP Complex Arithmetic Processor mag (250 regiszter): DSP algoritmusok ellenőrzése (ACL2 automatikus tételbizonyító)
- Intel Core i7 processzor
  - *„For the recent Intel CoreTM i7 design we used formal verification as the primary validation vehicle for the core execution cluster"*
  - Szimbolikus szimuláció az adatutak teljes vizsgálatára (2700 mikroutasítás, 20 mérnökévnyi munka) – Binary Decision Diagram alkalmazása
- Modell alapú szoftverfejlesztéshez kapcsolódó eszközök
  - IBM (Telelogic), Ansys (Esterel), Prover, Mentor, Verum, Conformiq, …

Forráskódhoz illeszkedő formális verifikáció

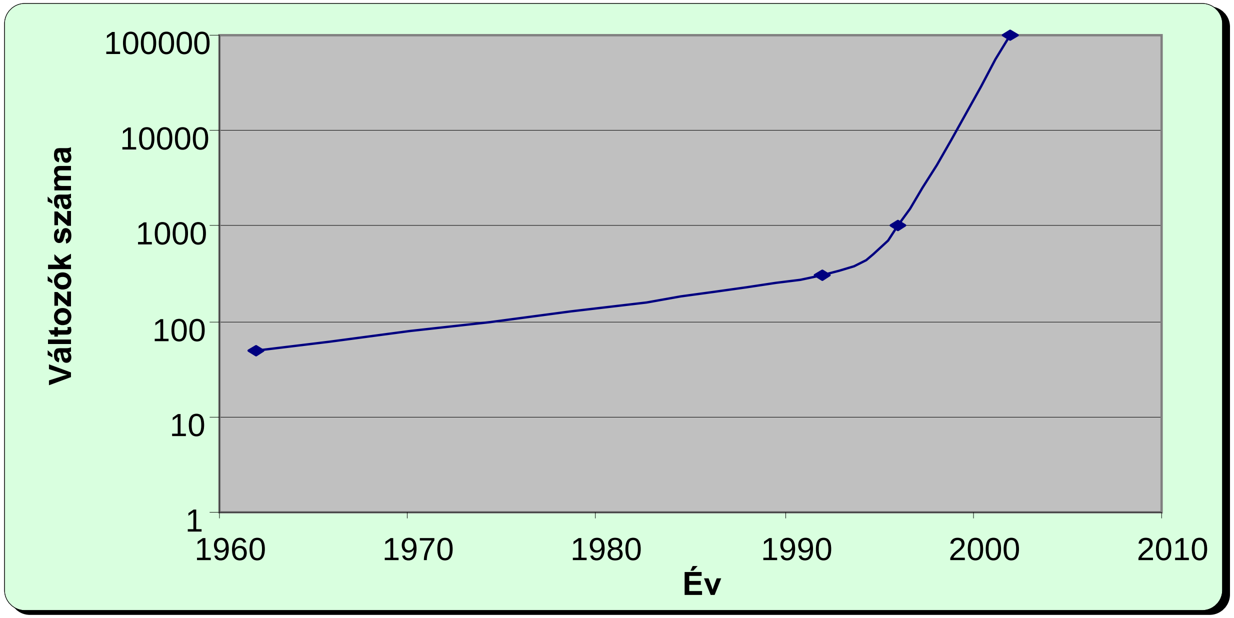
- Java
  - PathFinder: modell absztrakció
  - Java VM formalizálása: Abstract State Machine
- Ada
  - SPARK Ada verification condition generator: tételbizonyítóhoz
- C
  - BLAST: Szoftver modellellenőrző C programokhoz (absztrakció)
  - CBMC: C alapú korlátos modellellenőrző
- C#, Visual Basic .Net
  - Zing (MS Visual Studio-hoz): Konkurens OO szoftver ellenőrzése
- Spec# (C# superset)
  - MS Research Boogie 2: Specifikációs nyelvi kiterjesztések
  - Helyességi kritériumok ellenőrzése: program absztrakcióval és tételbizonyítóval (Z3)
- Microsoft Windows Driver Kit (WDK)
  - Static Driver Verifier Platform, SLAM 2 eszköz
  - Windows API használati feltételeinek statikus ellenőrzése

Amik a formális módszereket nehézzé teszik…

- Valósághű modellezés
  - Ismeretek hiánya, feltételezések (pl. a környezetről)
  - De: Formális módszerek használatától független ez a probléma
- Speciális ismereteket igényel a felhasználótól
  - Matematikai modellek, jelölésrendszer
  - De: Mérnöki modellezési nyelvek eltakarhatják
- Bonyolultak az ellenőrzés és szintézis módszerei
  - Algoritmusok, technikák korlátait ismerni kell
  - Kézi beavatkozásra lehet szükség (pl. tételbizonyító rendszerek)
  - De: Terjednek a „gombnyomásra működő” eszközök
- Csak „kisméretű” problémákra alkalmazható
  - Nagy modell, állapottér kezelhető-e a meglévő erőforrásokkal?
  - De: Eszközök hatékonysága folyamatosan nő

A formális verifikáció fejlődése (példák)

- SAT eszközök (logikai függvények kielégíthetősége)



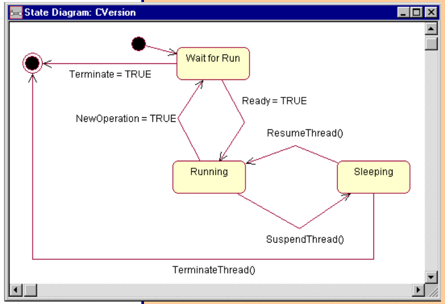
- Modellellenőrző eszközök képességei:
  - – $10^{20} \approx 2^{66}$ méretű állapottér ellenőrzése (ROBDD, 1990)
  - – $10^{100} \approx 2^{328}$ méretű állapottér is elérhető (konkrét eset)
  - – $10^{62\,900}$ méretű állapottérre is volt már példa :-O

Egy sikeres megközelítés

Inf. rendszer tervezése

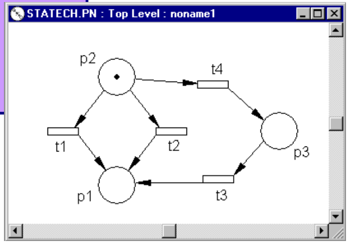
Formális ellenőrzés

Mérnöki modell (pl. DSL)


State Diagram: CVersion

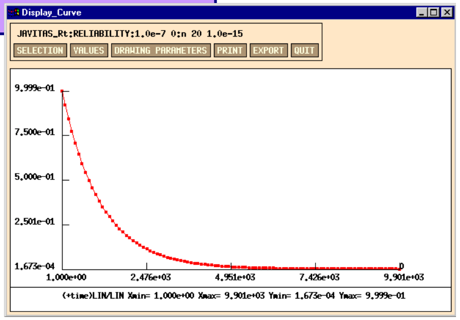
Automatikus modell leképzés

Formális modell


STATECH.PN : Top Level : noname1

Verifikáció, analízis

Eredmények visszavezetése


Display_Curve

```
JAVITAS_Rt:RELIABILITY:1.0e-7 0:n 20 1.0e-15
```

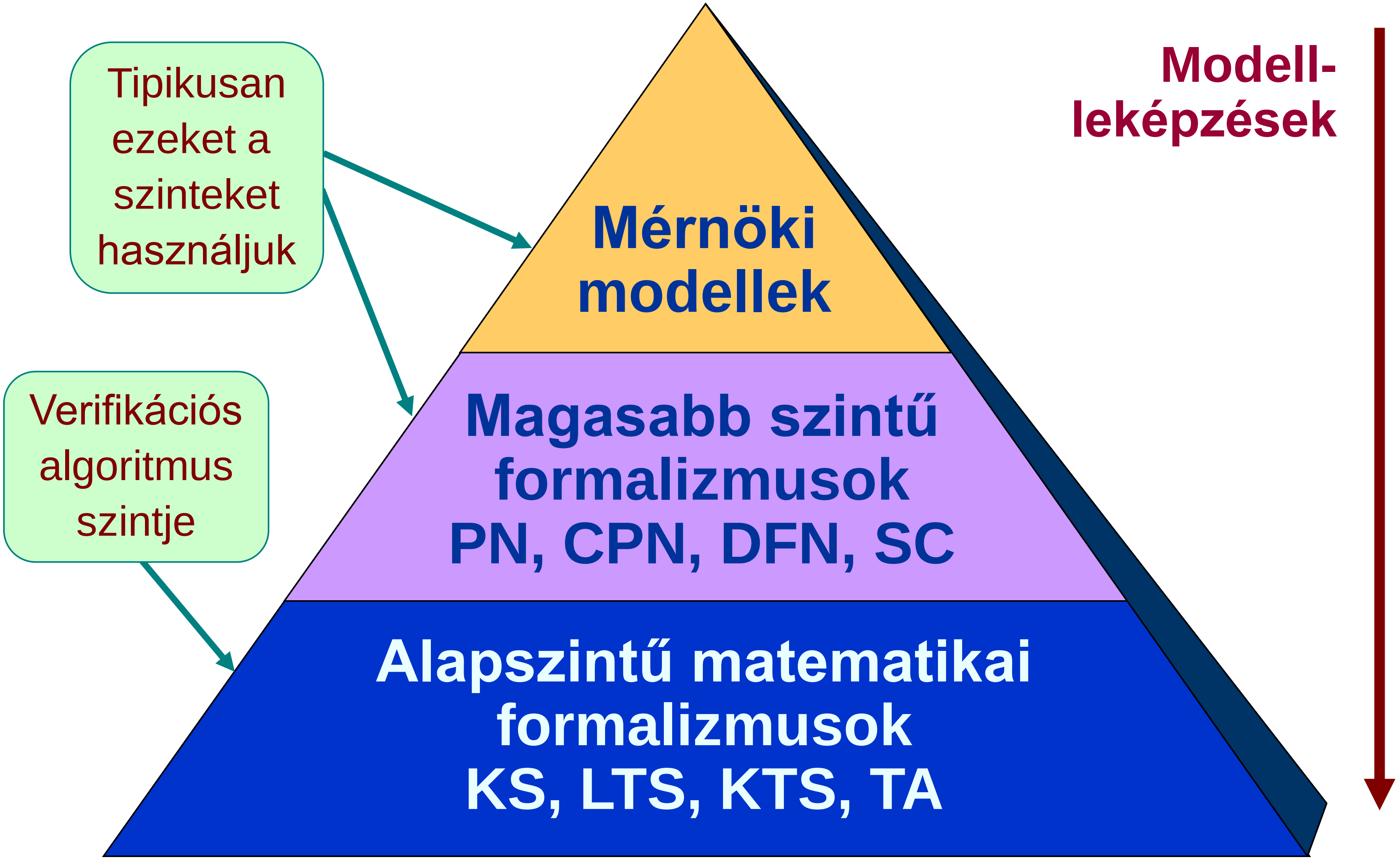
```
(+time)LIN/LIN Xmin= 1.000e+00 Xmax= 9.901e+03 Ymin= 1.673e-04 Ymax= 9.999e-01
```

Megvalósítás

Implementáció

```
int main() {
  while (i<z){
  a=x+x*b[i++];
  }
}
```

Visszautalás: A tárgy felépítése



Tipikusan ezeket a szinteket használjuk

Verifikációs algoritmus szintje

Mérnöki modellek

Magasabb szintű formalizmusok PN, CPN, DFN, SC

Alapszintű matematikai formalizmusok KS, LTS, KTS, TA

Modell-leképzések

Összefoglalás

- Mik a formális módszerek?
  - Formális nyelv (formalizmus):
    Tervek és tulajdonságok leírása
  - Eljárások, technikák a formális leíráshoz:
    Szimuláció, formális verifikáció, szintézis
- Mire használhatók?
  - Tipikus alkalmazási terület: Kritikus rendszerek
  - A formális módszerek lehetőségei
- Mit várhatunk?
  - Korlátok
  - Sikertörténetek