

A formális módszerek szerepe

dr. Majzik István
dr. Bartha Tamás
dr. Pataricza András

BME Méréstechnika és Információs Rendszerek Tanszék

Mik a formális módszerek?

Definíciók, formalizmusok

Formális módszerek az informatikában

- Matematikai technikák
 - modellek, formalizmusok,
 - algoritmusok

használata hardver és szoftver rendszerek

- specifikálására,
- tervezésére,
- ellenőrzésére,
- szintézisére,
- tesztelésére,
- dokumentálására

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?
- A formális nyelv előnyei
 - Precizitás: Egyértelműség, ellenőrizhetőség
 - Automatikus feldolgozás: Ellenőrzés, leképezés, (kód)generálás
- Milyen lehet a formális leírás?
 - Tipikusan absztrakt (implementáció-független)
 - Sokféle szempontú: struktúra, funkció, teljesítmény, biztonság, ...

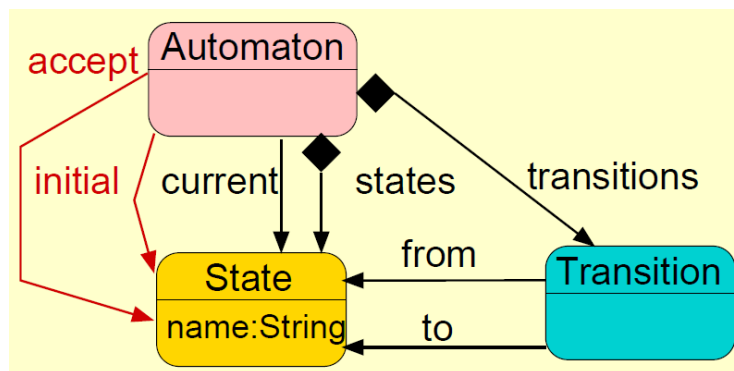
Formális szintaxis (áttekintés)

- Matematikai megadás:

$KS = (S, R, L)$ és AP , ahol

$$AP = \{P, Q, R, \dots\}$$
$$S = \{s_1, s_2, s_3, \dots, s_n\}$$
$$R \subseteq S \times S$$
$$L: S \rightarrow 2^{AP}$$

- BNF: $BL ::= \text{true} \mid \text{false} \mid p \wedge q \mid p \vee q$
- 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
 - Használható az állapottér felvételéhez és ellenőrzéséhez
- **Axiomatikus szemantika: „Helyességbizonyításhoz”**
 - Megadja, hogyan „vezethetők le” a tulajdonságok
 - Állítás nyelv + axiómakészlet + következtetési szabályok
 - Használható automatikus tételbizonyító rendszerekhez
- **Denotációs szemantika: „Fordítóprogramokhoz”**
 - Szintaxis által meghatározott **leképzés** egy ismert doménre
 - Ismert matematikai domén, pl. vezérlési gráf, számítási szekvencia, állapothalmaz, ... és ezeken definiált műveletek (összefűzés, unió, ...)
 - Használható kódgeneráláshoz is

Második lépés: A formális modell használata

- A formális modell végrehajtása
 - Szimuláció, animáció
- 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 \leftrightarrow specifikáció)
 - A modellek összevetése:
 - Modellfinomítás vizsgálata: tulajdonságok megtartása, kiterjesztése
- A formális modell alapján történő szintézis:
 - Szoftver (programkód, konfiguráció), hardver leírás generálása
 - Tesztek generálása
 - Dokumentáció generálása

Mire jók a formális módszerek
egy mérnökinformatikus számára?

Egy jellegzetes példa:
Algoritmus helyességének ellenőrzése

Egy mérnöki feladat

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

```
while (true) {  
    blocked0 := true;  
    while (turn!=0) {  
        while (blocked1==true) {  
            skip;  
        }  
        turn := 0;  
    }  
    // Start of critical section  
    // End of critical section  
    blocked0 := false;  
    // Do other things  
}
```

P0

```
while (true) {  
    blocked1 := true;  
    while (turn!=1) {  
        while (blocked0==true) {  
            skip;  
        }  
        turn := 1;  
    }  
    // Start of critical section  
    // End of critical section  
    blocked1 := false;  
    // Do other things  
}
```

P1

Helyes-e ez az algoritmus?

Mikor helyes az algoritmus?

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


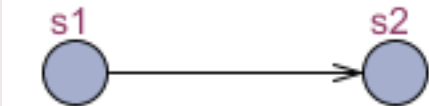
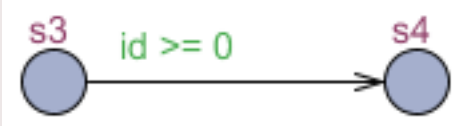
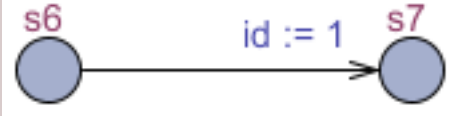
Hogyan ellenőrizhetjük a helyességet?

- Megvalósítással és annak tesztelésével
 - Probléma: Konkurens processzek – utasításaik sokféle ütemezés szerint végrehajthatók: írhatják, olvashatják a megosztott változókat
 - Létre tudunk-e hozni olyan tesztkörnyezetet és tesztkészletet, ami minden lehetséges konkurens végrehajtást lefed?
 - A problémás eset figyeléséhez külön ellenőrző kell
 - A hiba drágán javítható (csak az implementáció után derül ki)
- Modellezéssel és a modell szimulációjával
 - A hibák olcsóbban javíthatók modell szinten
 - Tudunk-e szimulálni minden lehetséges konkurens végrehajtást?
- 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:

Szintaxis elemek	Megjelenítés
Állapot névvel, kezdőállapot	
Állapotátmenet	
Változó deklarálás	<pre>int id;</pre>
Feltétel átmeneten	
Akció átmeneten	

- Szemantika: Aktív állapotból, igazra kiértékelhető feltétel mellett váltás az állapotátmenet célállapotába, az akció végrehajtásával

A P0 processz formális modellje

Változók deklarációi:

bool blocked0=false;

bool blocked1=false;

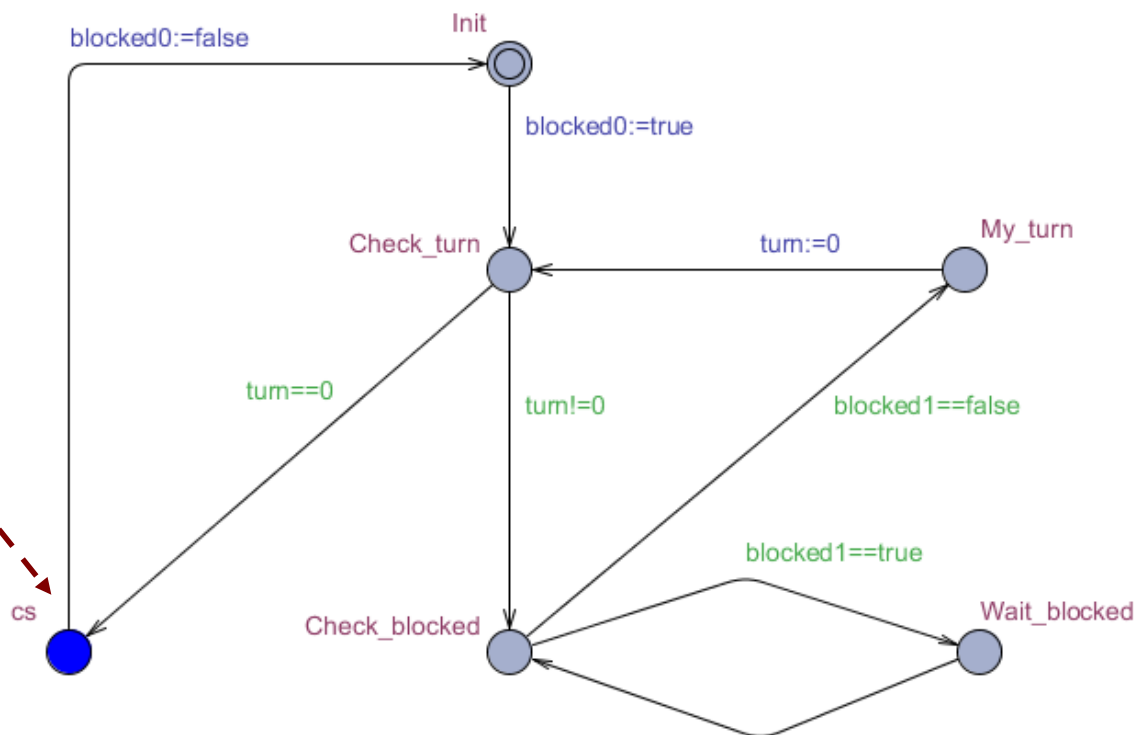
int[0,1] turn=0;

system P0, P1;

A P0 processz pszeudokód és automata modellje:

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

P0



A P1 processz formális modellje

Változók deklarációi:

bool blocked0=false;

bool blocked1=false;

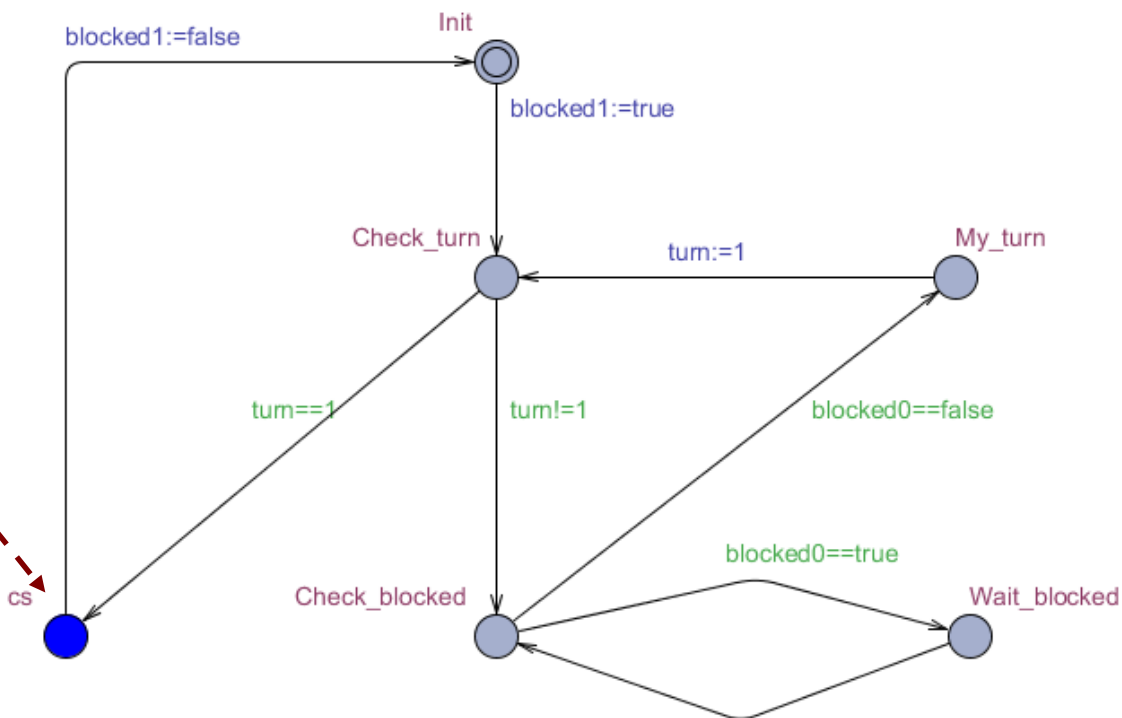
int[0,1] turn=0;

system P0, P1;

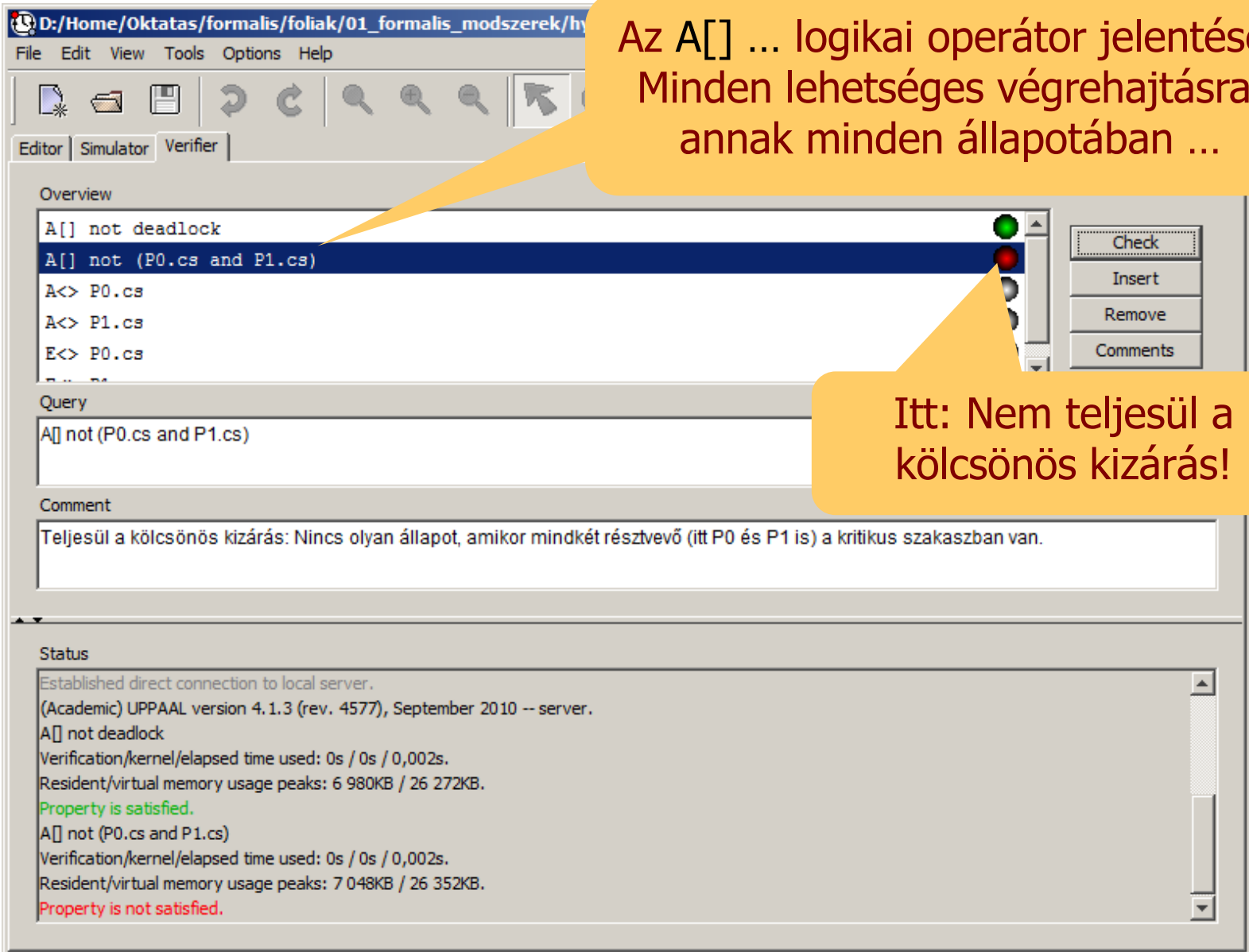
A P1 processz pszeudokód és automata modellje:

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

P1



Tulajdonságok ellenőrzése egy gombnyomásra

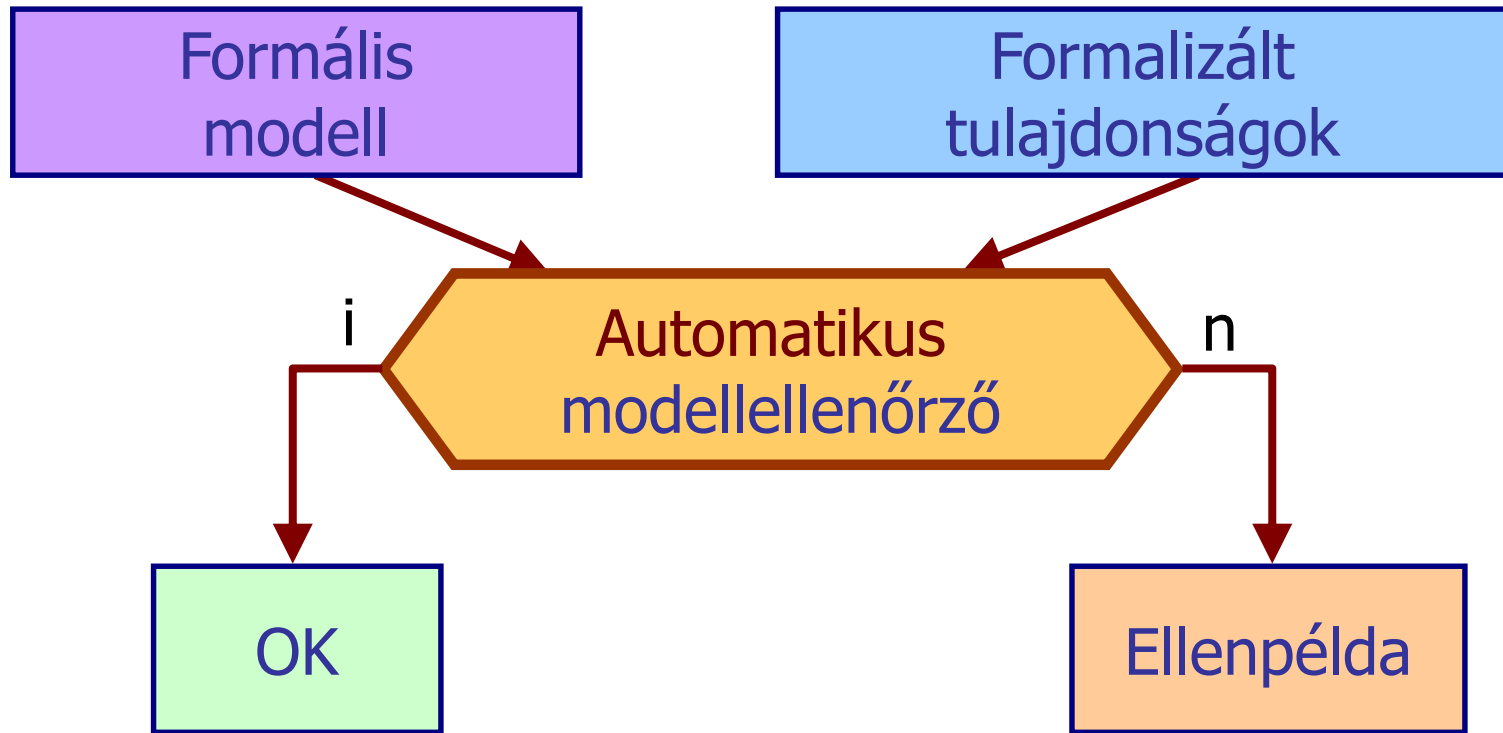


The screenshot shows the UPPAAL model checker interface. The 'Overview' tab is active, displaying a list of properties. The property `A[] not (P0.cs and P1.cs)` is highlighted in blue. To the right of this property, a traffic light icon shows a green light at the top and a red light at the bottom. To the right of the traffic light are four buttons: 'Check', 'Insert', 'Remove', and 'Comments'. Below the 'Overview' tab is the 'Query' section, which contains the same property `A[] not (P0.cs and P1.cs)`. Below the 'Query' section is the 'Comment' section, which contains the text: 'Teljesül a kölcsönös kizárás: Nincs olyan állapot, amikor mindkét résztvevő (itt P0 és P1 is) a kritikus szakaszban van.' Below the 'Comment' section is the 'Status' section, which contains the following text: '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.'

Az `A[]` ... logikai operátor jelentése:
Minden lehetséges végrehajtásra,
annak minden állapotában ...

Itt: Nem teljesül a
kölcsönös kizárás!

Egy jellegzetes formális ellenőrzés (verifikáció)



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

Formális módszerek használata

Mit ígérnek a formális módszerek?
Mikor és hogyan használjuk?

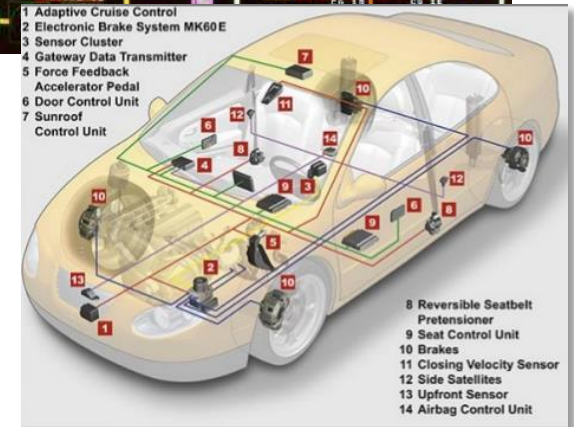
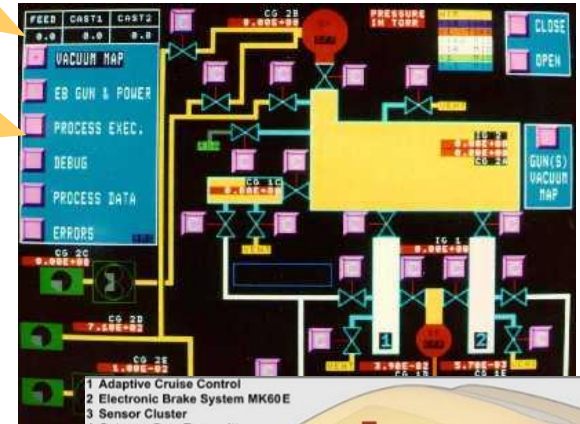
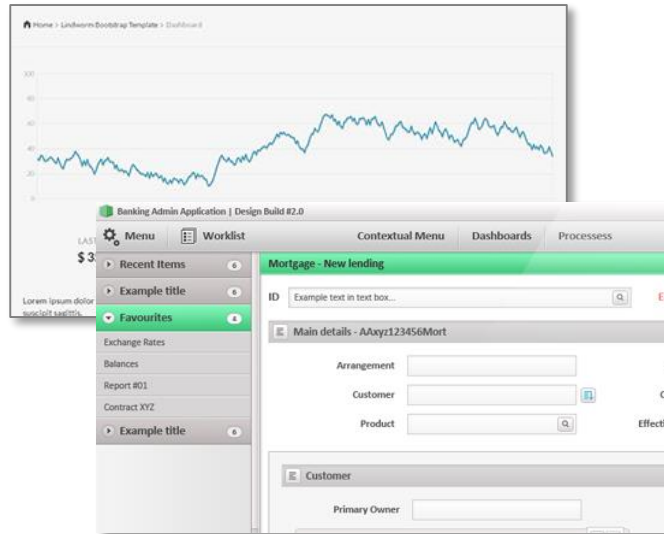
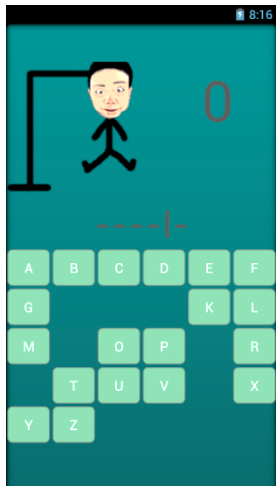
Mit ígérnek a formális módszerek?

- 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

Kiemelt szerep: Kritikus rendszerek tervezése

Bennmaradó hibák
kockázata nagy

Konkurens / aszinkron /
időfüggő működés



- Minimális tervezés
- Implicit folyamat
- Egyszerű eszközök

- Alapos tervezés
- Jól definiált folyamat
- Hatékony eszközök

- Igazolt helyességű tervek
- Meghatározott folyamat
- Automatikus eszközök

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	R
3	Defensive programming	C.2.5	---	R	HR	---
4	Modular approach	Table B.9	---	---	---	---
5	Design and coding standards	C.2.6 Table B.1	---	---	---	---
6	Structured programming	C.2.7	---	---	---	---
7	Use of trusted/verified software elements (if available)	C.2.10	---	---	---	---
8	Forward traceability between the software safety requirements specification and software design	C.2.11	R	R	HR	HR

HR: Highly Recommended
Nyomatékosan ajánlott
(elhagyását indokolni kell)

Szabvány előírások biztonságkritikus szoftverekhez

- Biztonságintegritási szintek (SIL 1...4)
- IEC 61508 szabvány: Előírások a fejlesztésre

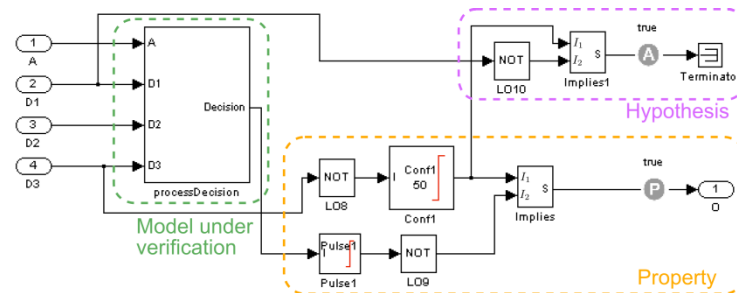
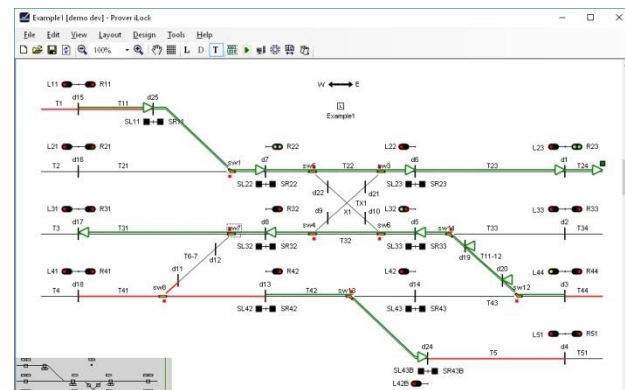
Functional safety in electrical / electronic / programmable electronic safety-related systems

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

Formális módszerek kritikus rendszerekhez

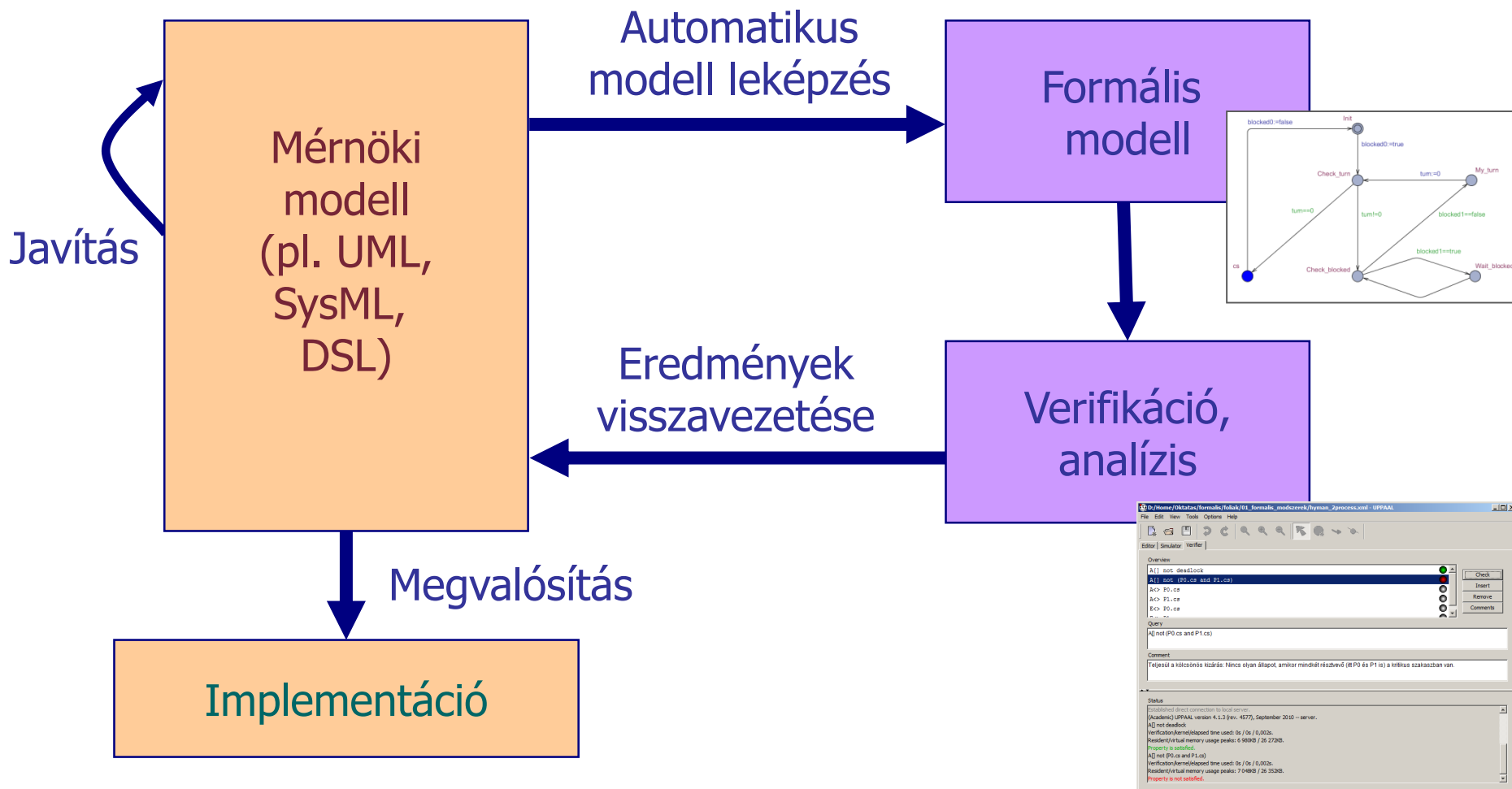
- iLock Tool Suite (Prover)
 - Formális nyelv (HLL) vasúti biztosítóberendezésekhez
 - Formális verifikáció és kódgenerálás
- SCADE Suite (Ansys)
 - Biztonságigazolt „szoftvergyár” beágyazott vezérlőkhöz
- Questa Formal Verification Tool (Mentor Graphics)
 - RTL szintű hardver tervek ellenőrzése
- LDRA Tool Suite (LDRA)
 - Szoftver adatfolyam analízis



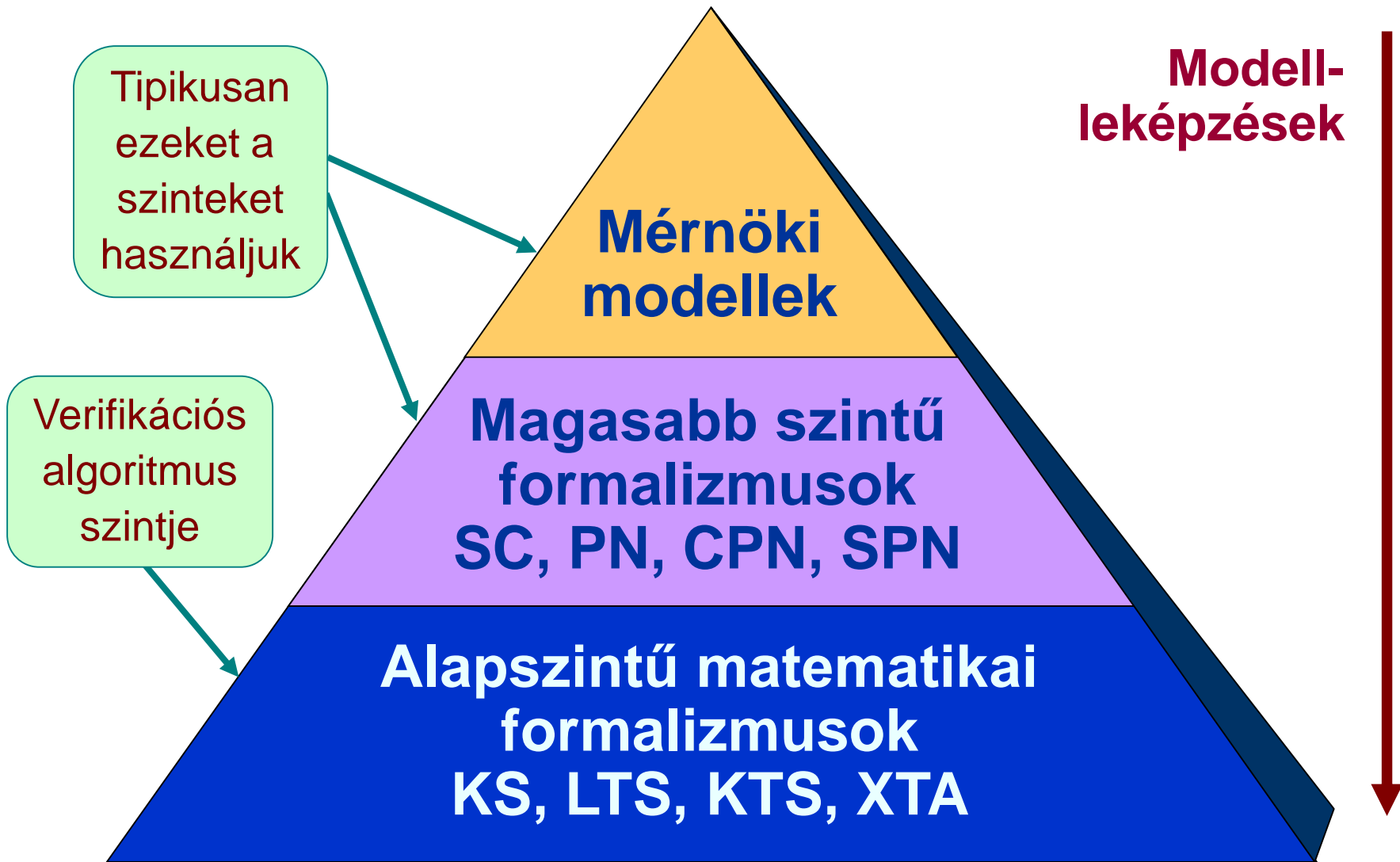
Jellegzetes alkalmazás a mérnöki tervezésben

Inf. rendszer tervezése

Formális ellenőrzés



Visszaautalás: A tárgy felépítése

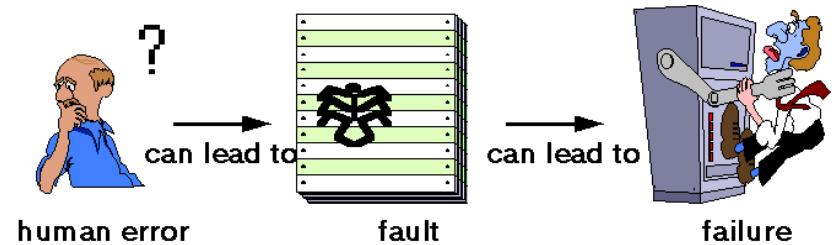


A formális módszerek használatának előnyei

Mi a használat haszna?
Vannak-e sikertörténetek?

Cél: Kézi ellenőrzés költségeinek csökkentése

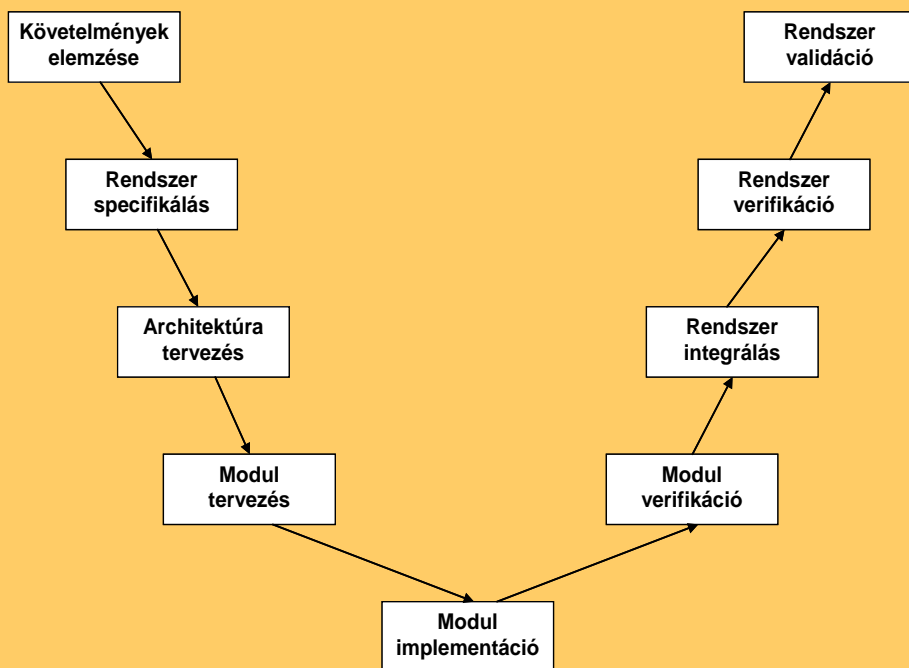
- Tipikus kódméret:
 - 10 kLOC ... 1000 kLOC
- Fejlesztési ráfordítás:
 - Nagyméretű szoftver: 0,1 - 0,5 mérnökév / kLOC
 - Kritikus szoftver: 5-10 mérnökév / kLOC ráfordítás
- Hiba eltávolítás (ellenőrzés, tesztelés, javítás):
 - 45 - 75% ráfordítás a fejlesztési költségekből
 - Minél korábban detektálható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



V-től az Y fejlesztési modellig

Életciklus

Szoftverfejlesztés a V-modell szerint



ráfordítás

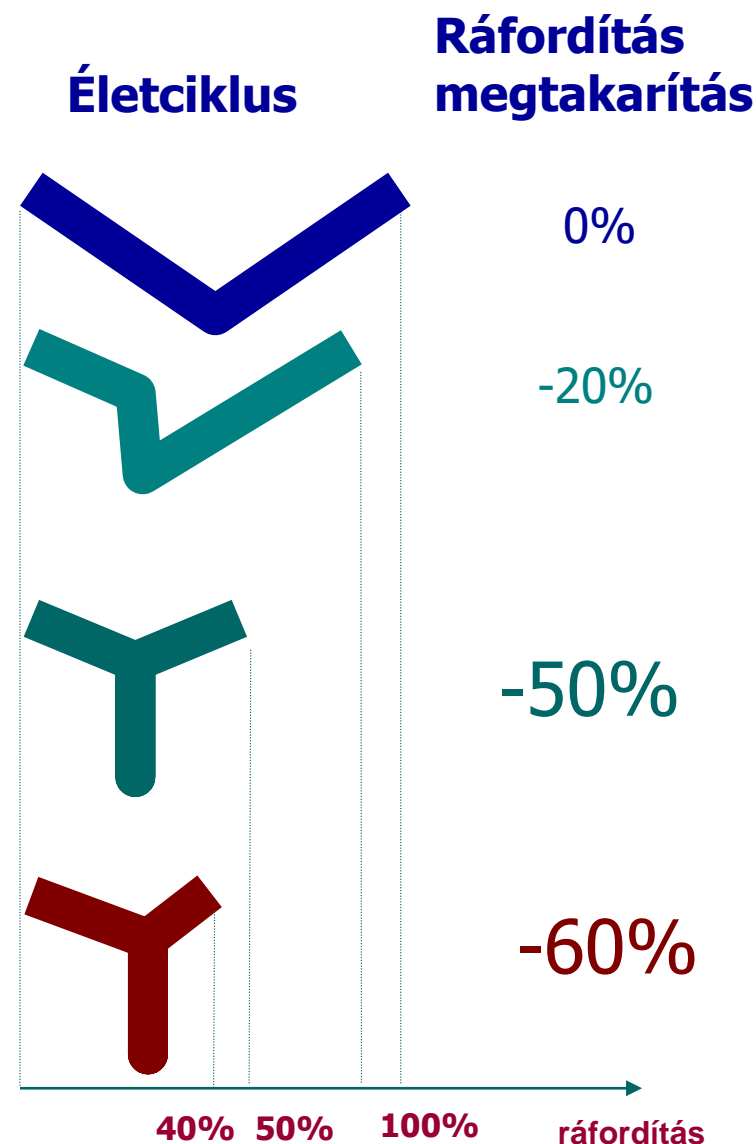
V-től az Y fejlesztési modellig

Kézi kódolás

Modell alapú automatikus
kódgenerátor használata

Minősített automatikus
kódgenerátor használata

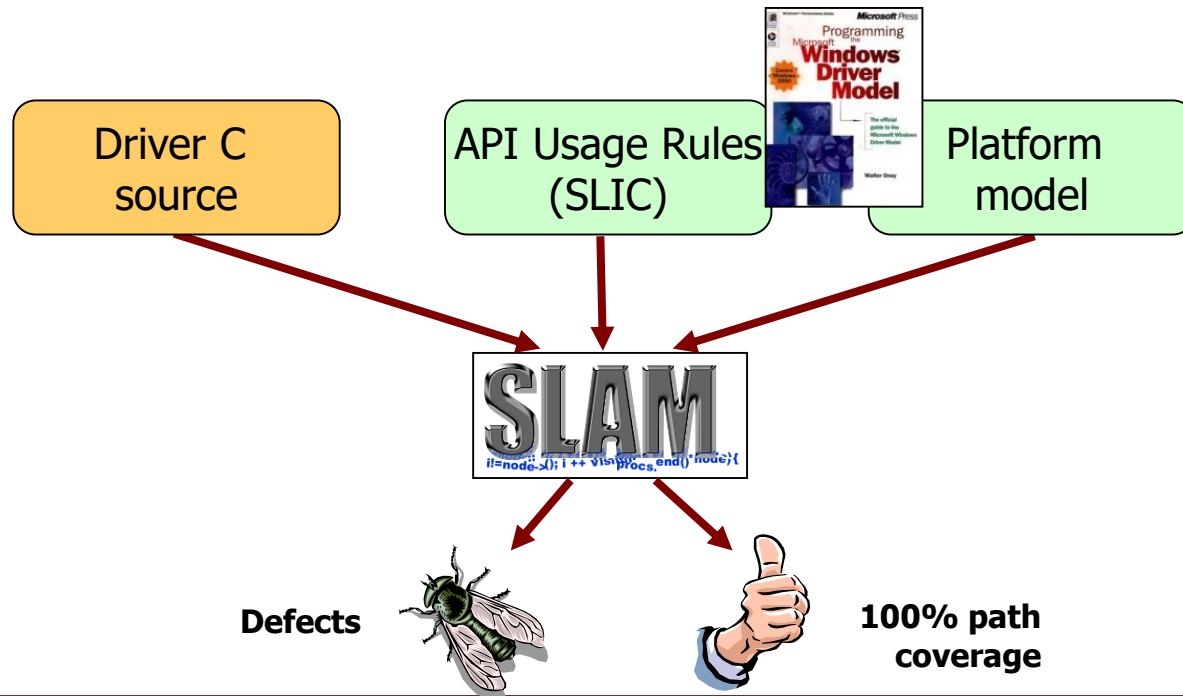
Formális verifikációval
kiegészített tervezés



* Adatok: Esterel Technologies (Ansys)

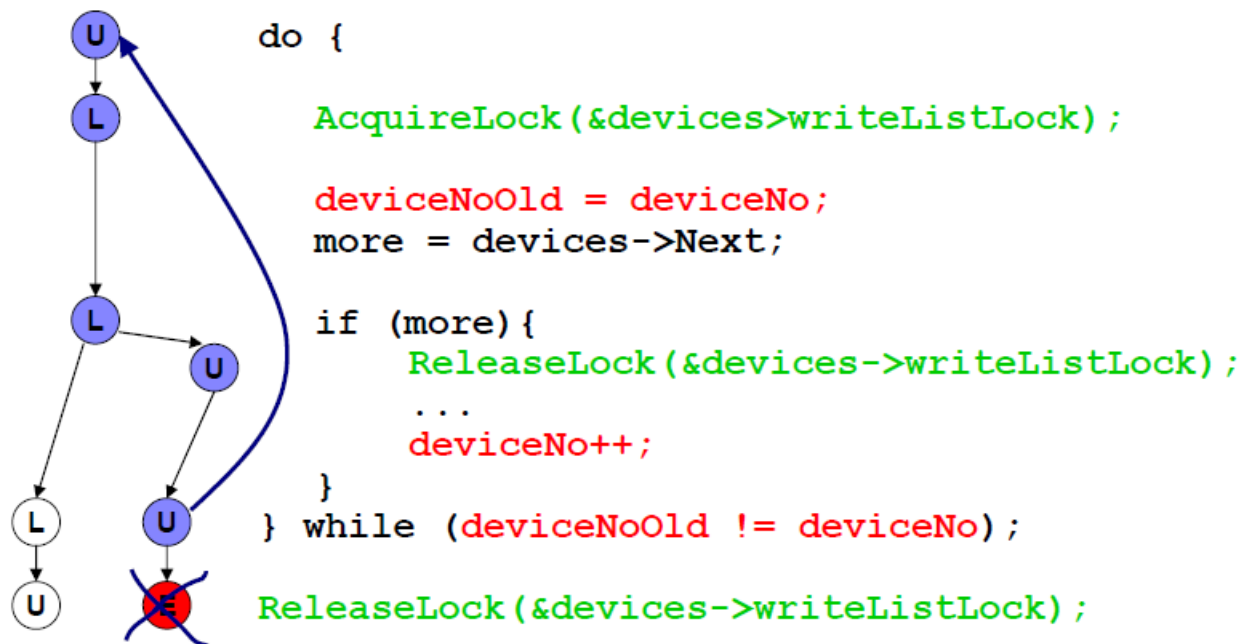
Példa: OS komponensek fejlesztése SLAM eszközzel

- **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



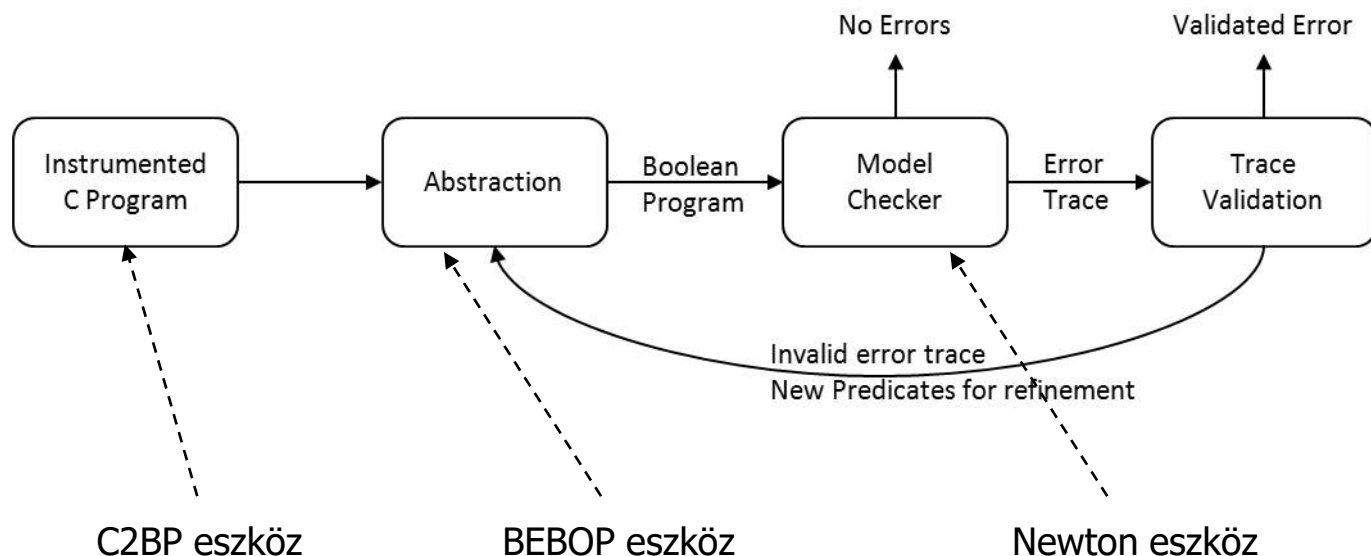
Példa: OS komponensek fejlesztése SLAM eszközzel

- 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



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



Példák: Sikeres alkalmazások

- USA TCAS-II forgalomirányító rendszer
 - RSMML 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 Core™ 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, ...

Példa: 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

A formális módszerek használatának korlátai

Mik a korlátok?

Mik a lehetőségek ezek feloldására?

Formális verifikáció nem váltja ki a validációt

Verifikáció (igazolás)

„Jól építjük-e a rendszert?”

Fejlesztési fázisok ellenőrzése:

- A fejlesztési lépések eredménye (tervek, modellek, forráskód) megfelel a specifikációnak és az előző lépéseknek

Objektív folyamat;
formalizálható, automatizálható

Felderíthető hibák: Tervezési, implementációs hibák

Nincs rá szükség, ha automatikus a leképzés a specifikáció és az implementáció között

Validáció (érvényesítés)

„Jó rendszert építettünk-e?”

Fejlesztés eredményének ellenőrzése

- A kész rendszer megfelel a felhasználói elvárásoknak, kielégíti a felhasználói igényeket

Szubjektív elvárások lehetnek; elfogadhatósági ellenőrzés

Felderíthető hibák: Követelmények hiányosságai is

Nincs rá szükség, ha a specifikáció tökéletes (elég egyszerű)

Amik a formális módszereket nehézé teszik

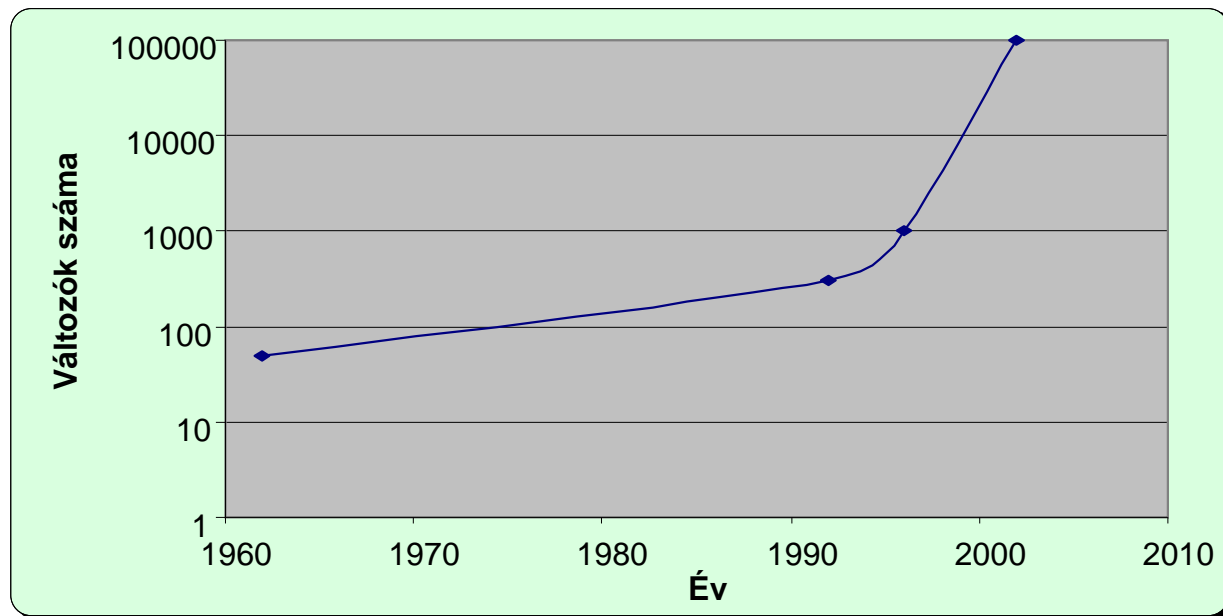
- Valósághű modellezés
 - Ismeretek hiánya, feltételezések (pl. a környezetről)
 - De: Formális módszerek használatától független ez a probléma
- Speciális ismereteket igényel a felhasználótól
 - Matematikai modellek, jelölésrendszer
 - De: Mérnöki modellezési nyelvek eltakarhatják
- Bonyolultak az ellenőrzés 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

Hatékony használat, de korlátok is



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

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?
 - Előnyök, sikertörténetek
 - Korlátok és feloldásuk