

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

Mik azok a formális módszerek?

- **Matematikai technikák használata,**
 - elsősorban diszkrét matematika
 - és matematikai logika,**hogyan hardver és szoftver rendszerek**
 - specifikációját,
 - terveit (modelljeit),
 - implementációját,
 - dokumentációját**elkészítjük 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:

$KS = (S, R, L)$ és AP, ahol

$AP = \{P, Q, R, \dots\}$

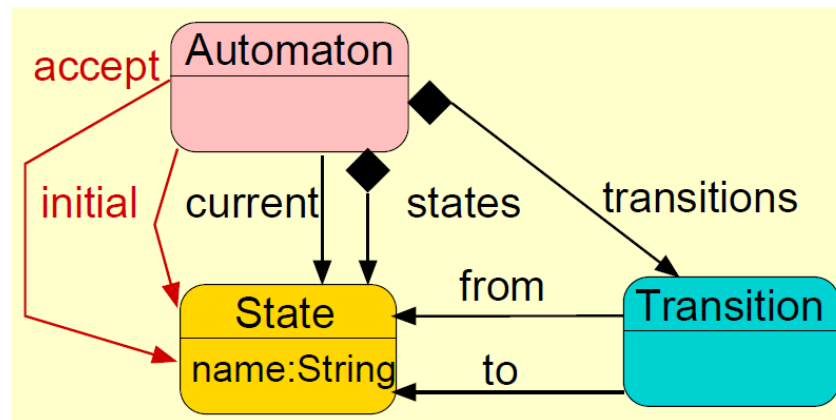
$S = \{s_1, s_2, s_3, \dots, s_n\}$

$R \subseteq S \times S$

$L: S \rightarrow 2^{AP}$

- 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:

- **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 leképezés 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 \leftrightarrow specifikáció)
 - A modellfinomítás megtartja a tulajdonságot (eredeti \leftrightarrow finomított tervek)
- A formális modell alapján történő **szintézis:**
 - Szoftver (programkód, konfiguráció) generálása
 - Hardver implementáció 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
 - 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)

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

P0

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

P1

Helyes-e ez az algoritmus?

Mikor helyes az algoritmus?



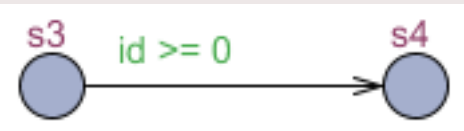
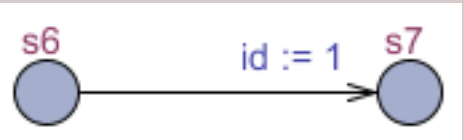
- Kölcsonös kizárás biztosított:
 - Egyszerre csak az egyik processz (P0 vagy P1) lehet a kritikus szakaszban
- Lehetséges az elvárt viselkedés:
 - P0 egyáltalán **beléphet** a kritikus szakaszba
 - P1 egyáltalán **beléphet** a kritikus szakaszba
- Nincs kiéheztetés:
 - P0 **mindenképpen** be fog lépni 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 konkurens végrehajtása (utasításaik sokféle ütemezés szerint írhatják, olvashatják a megosztott változókat)
 - Létre tudunk-e hozni minden lehetséges végrehajtást lefedő teszt eseteket?
 - 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 végrehajtást?
 - A problémás esetek detektálása nagy odafigyelést igényel
 - A hibák viszont olcsóbban javíthatók modell szinten
- **Itt javasolt: Formális modell készítése és ellenőrzése**
 - Minden lehetséges végrehajtási sorrend automatikus ellenőrzése
 - Problémás esetek automatikus felderítése és bemutatása
 - A modellben követhetők (és javíthatók) az algoritmus hibái

Készítsünk formális modellt!

- Formális nyelv: Véges automata, változókkal

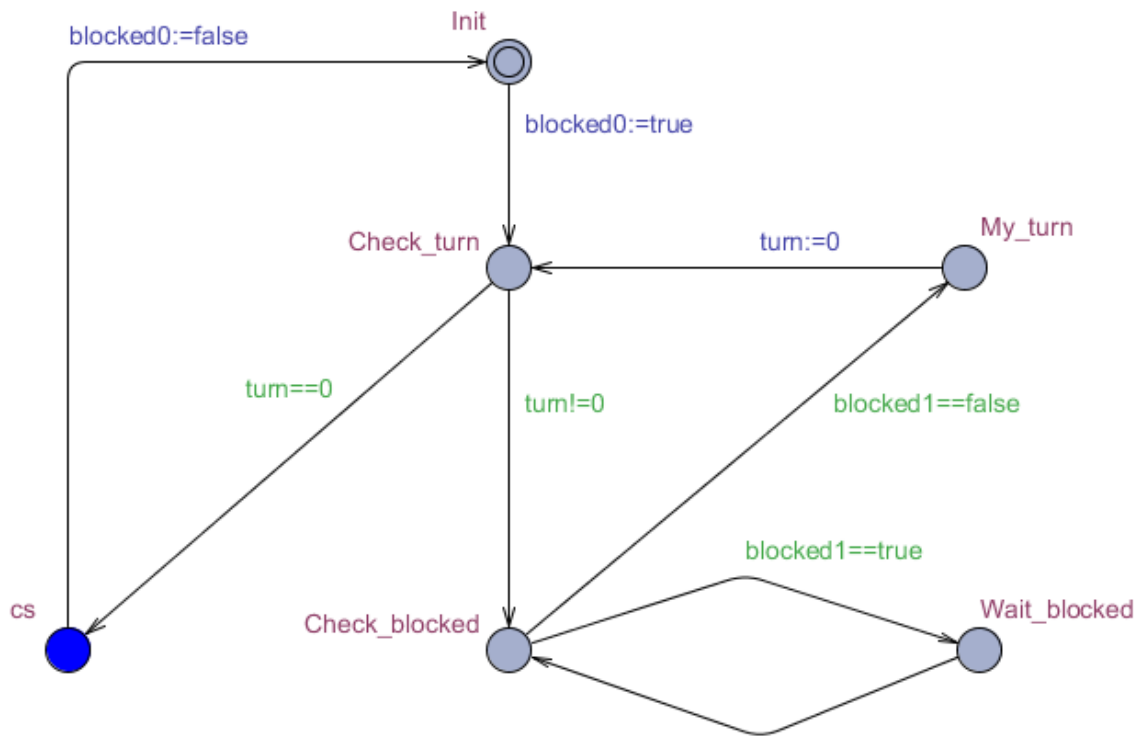
Szintaxis elemek	Megjelenítés
Állapot névvel, kezdőállapot	
Állapotátmenet	
Változó	<pre>int id;</pre>
Feltétel átmeneten	
Akció átmeneten	

A P0 processz formális modellje

Deklarációk:

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

A P0 automata:



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

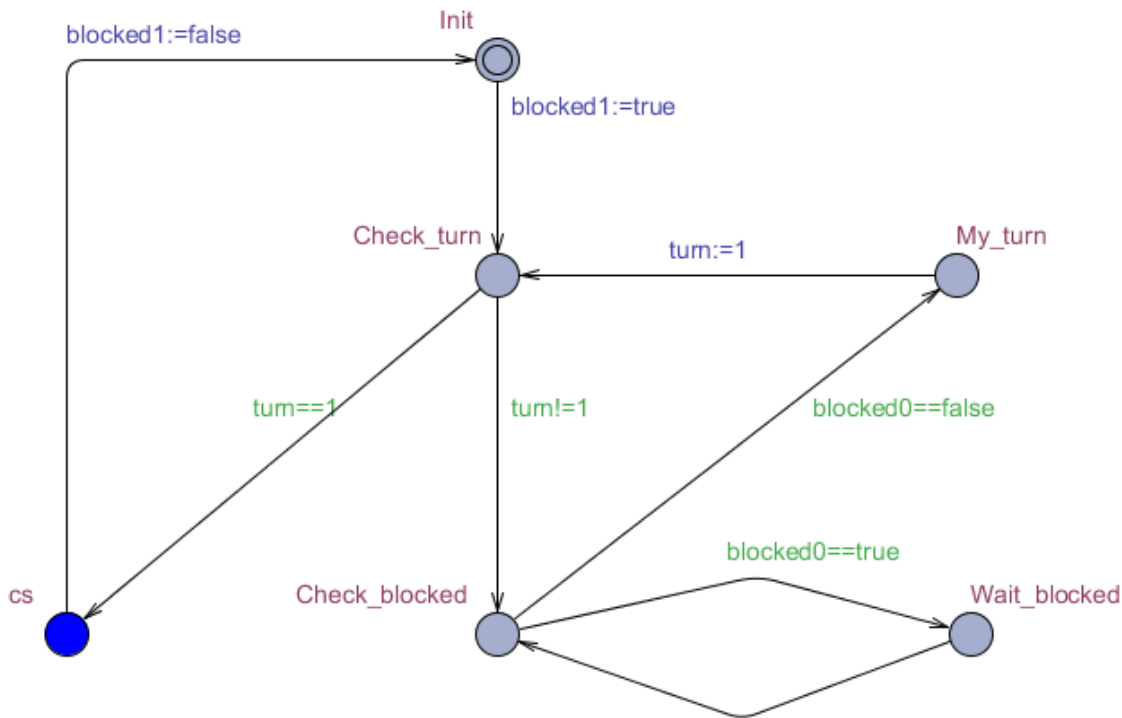
P0

A P1 processz formális modellje

Deklarációk:

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

A P1 automata:



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

P1

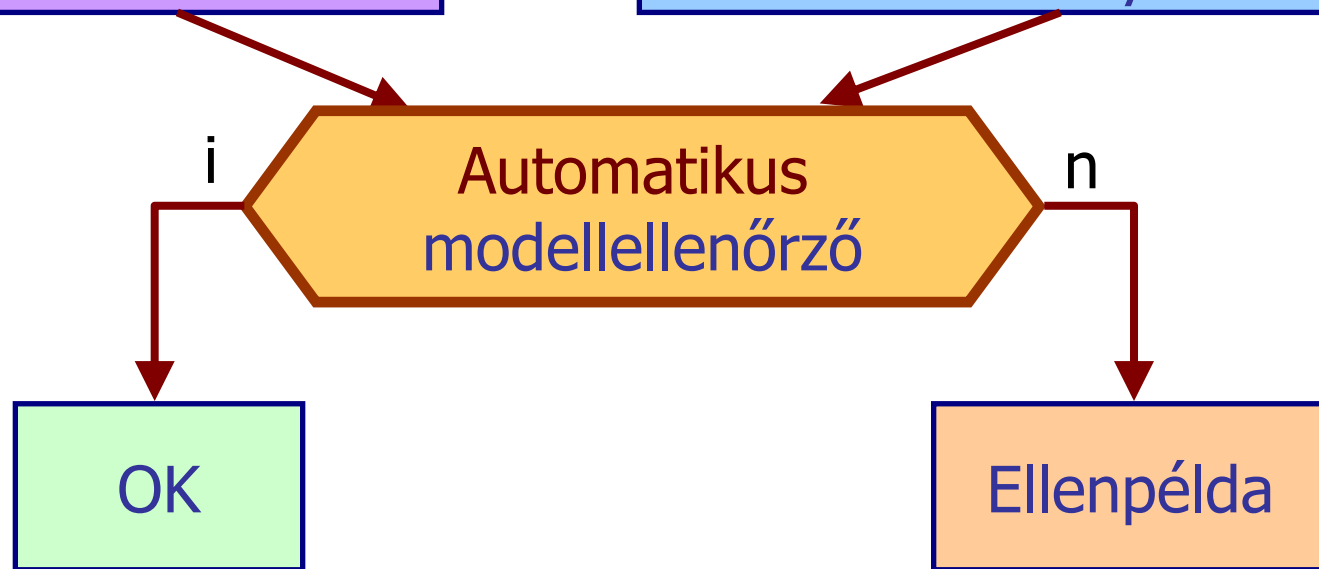
Egy jellegzetes formális ellenőrzés (verifikáció)

- Alacsony szintű, vagy
- magasabb szintű, vagy
- mérnöki modell

Automatikusan ellenőrizhető, precíz követelmények

Formális modell

Formalizált követelmények



Mit adnak 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?

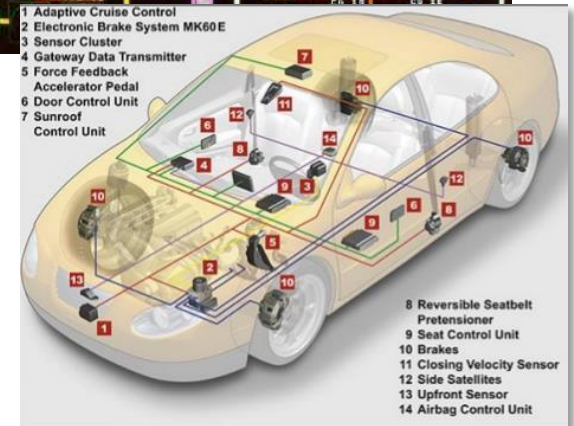
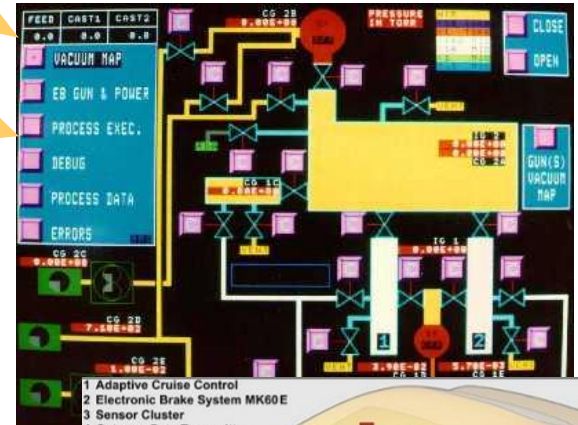
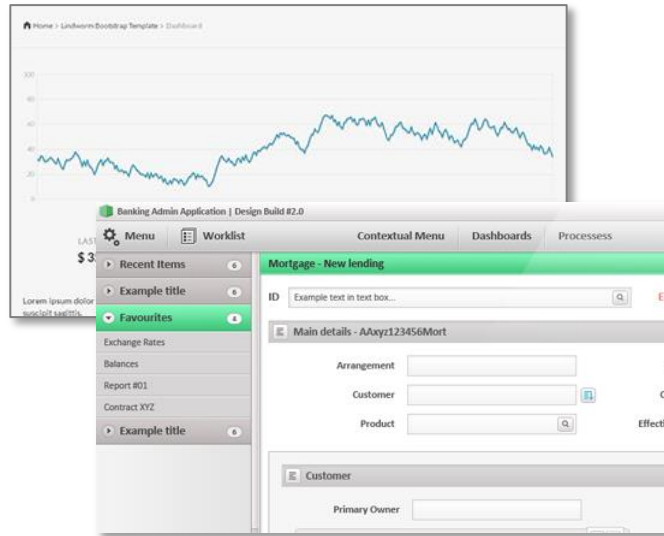
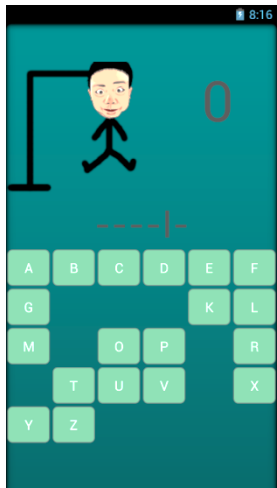
Mik a korlátok?

Mikor lehet sikeres?

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

Költséges ellenőrzés és hibajavítás

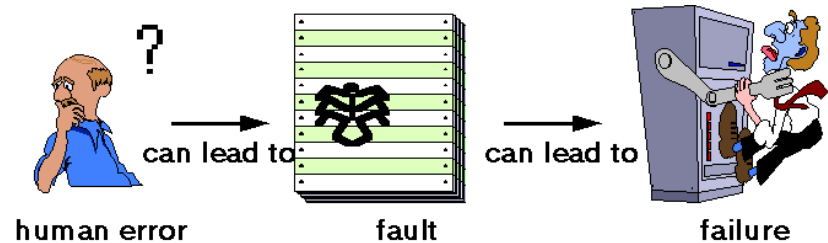
- Tipikus kódméret:
 - 10 kLOC ... 1000 kLOC

- 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



Biztonságkritikus szoftverek fejlesztése

- 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

Biztonságkritikus szoftverek fejlesztése

- 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) verifikáció nem váltja ki a validációt

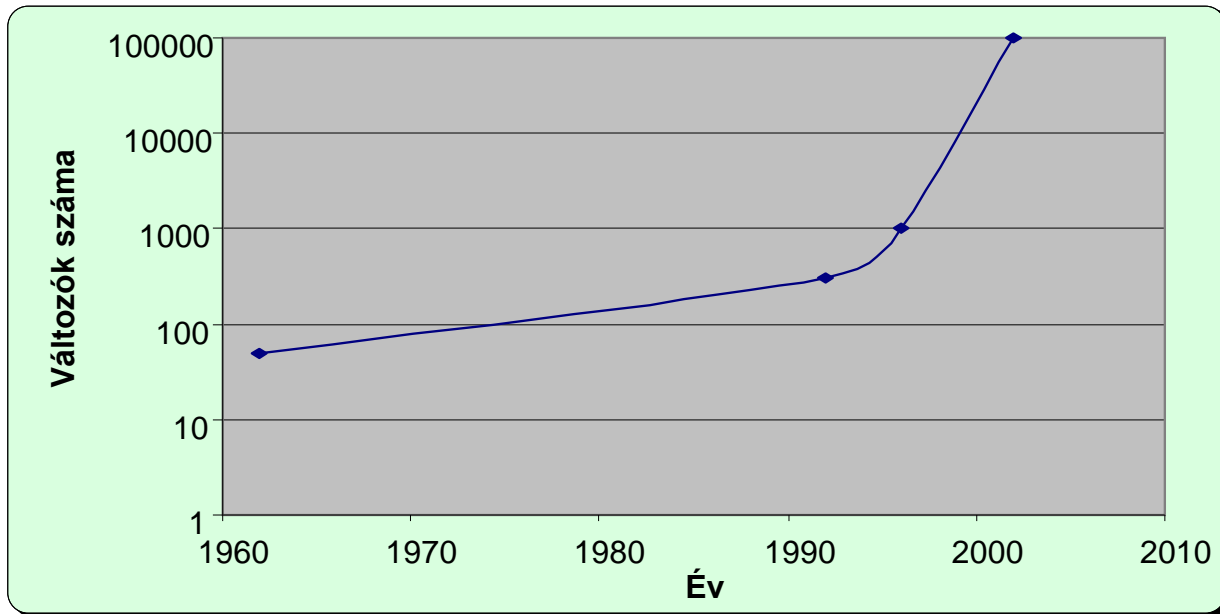
Verifikáció (igazolás)	Validáció (érvényesítés)
„Jól építjük-e a rendszert?”	„Jó rendszert építettünk-e?”
Fejlesztési fázisok ellenőrzése (eredmények, összhang)	A fejlesztés eredményének ellenőrzése
Fejlesztési lépések során használt tervek, modellek, dokumentáció, forráskód ellenőrzése	A kész rendszer és a felhasználói elvárások közötti megfelelés ellenőrzése
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 követelmények és az implementáció között	Nincs rá szükség, ha a specifikáció tökéletes (elég egyszerű)

Amik a formális módszereket nehézé teszik...

- Valóságghű 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

- Példa: A SAT eszközök (logikai kielégíthetőség)



- 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^{62900} méretű állapottérre is volt már példa :-O

Egy sikeres megközelítés

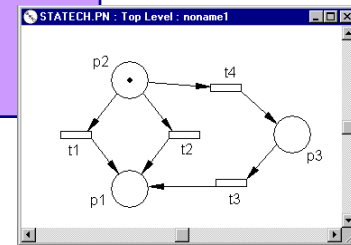
Inf. rendszer tervezése

Formális ellenőrzés

**Mérnöki modell
(pl. DSL)**

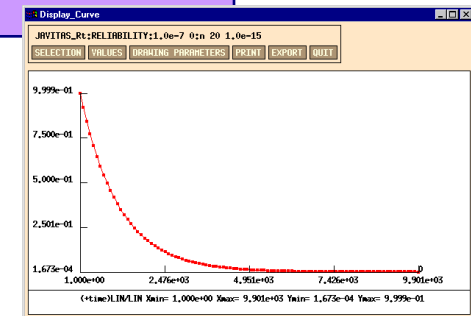
Automatikus
modell leképezés

**Formális
modell**



Eredmények
visszavezetése

**Verifikáció,
analízis**



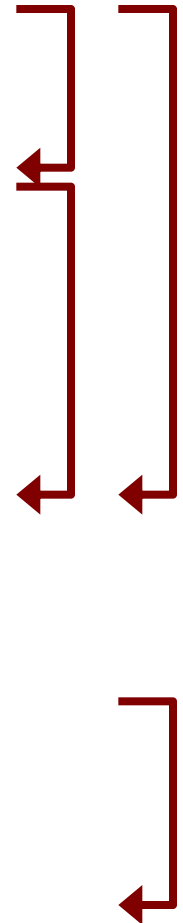
Megvalósítás

Implementáció

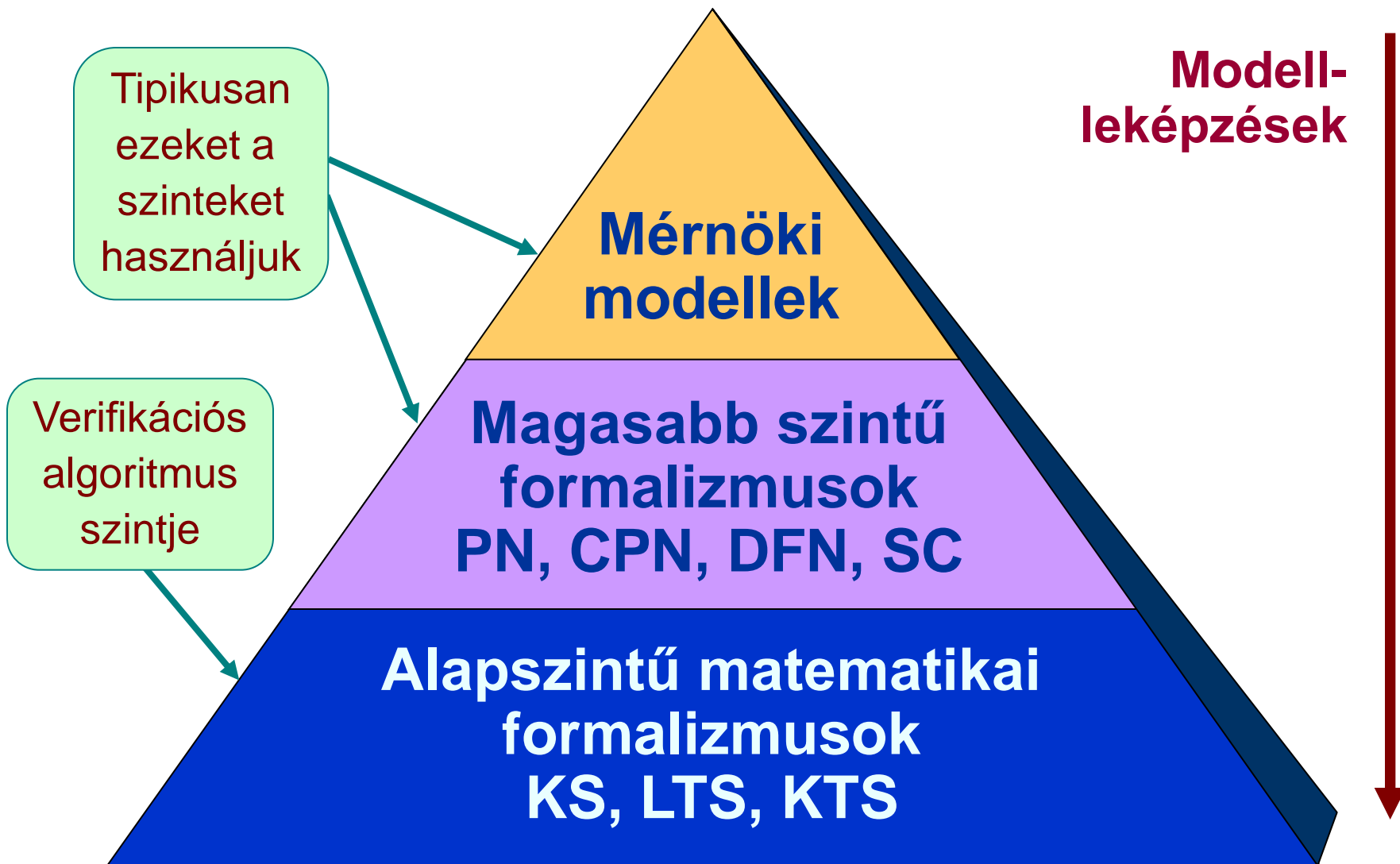
```
int main() {  
  while (!<z){  
    a=x*x*b[i++];  
  }  
}
```

Modellek a formális ellenőrzéshez

- Tervek leírása (tervezői modellek)
 - Mérnöki modellek:
 - Pl. DSL, UML, SysML (fél-)formális szemantikával
 - Magasabb szintű formális modellek:
 - Vezérlés orientált: Automata, Petri-háló, ...
 - Adatfeldolgozás orientált: Adatfolyam háló, ...
 - Kommunikáció orientált: Processz algebra, ...
 - Alapszintű formális modellek:
 - KS, LTS, KTS, automaták, Büchi automaták
- Tulajdonság leírások
 - Magasabb szintű formalizálás:
 - Idődiagram, üzenet szekvencia diagram (MSC)
 - Alapszintű formalizálás:
 - Elsőrendű logika, temporális logika, referencia automata



Visszautalás: A tárgy felépítése



Sikeres alkalmazások

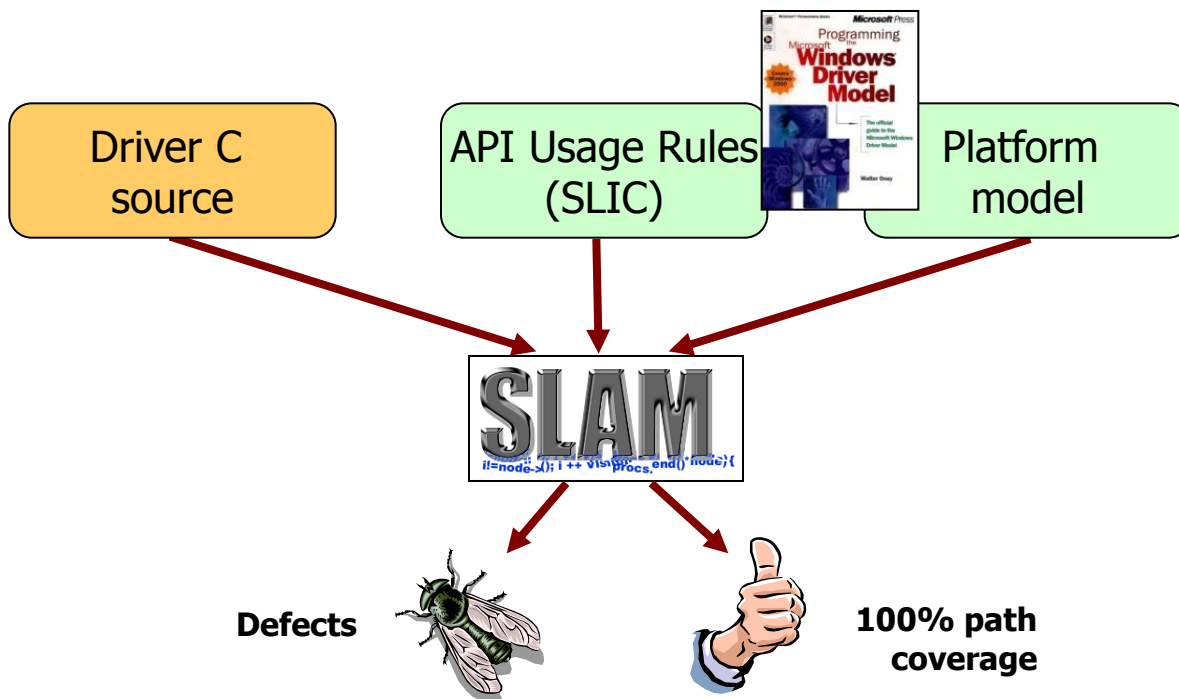
- USA TCAS-II forgalomirányító rendszer
 - RSMIL nyelven **specifikált**; teljesség és ellentmondás-mentesség ellenőrzése
- Philips Audio Protocol
 - 1994: manuális verifikáció, majd 1996: **automatikus ellenőrzés** (HyTech)
- Lockheed C130J repülési szoftvere
 - Programfejlesztés **helyességbizonyítással** (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)
 - AMD 5K86 processzor: Lebegőpontos osztás algoritmusának ellenőrzése
- **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 mérnökvényi 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

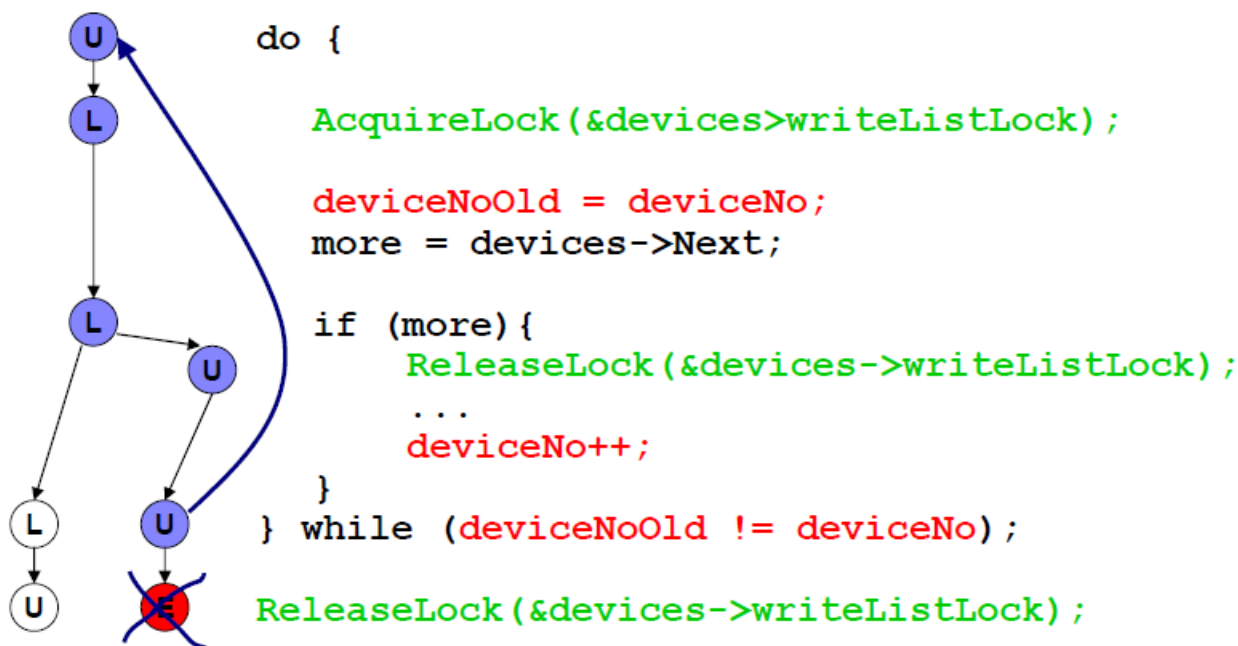
Példa: SLAM

- Motiváció: Hibás meghajtók 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



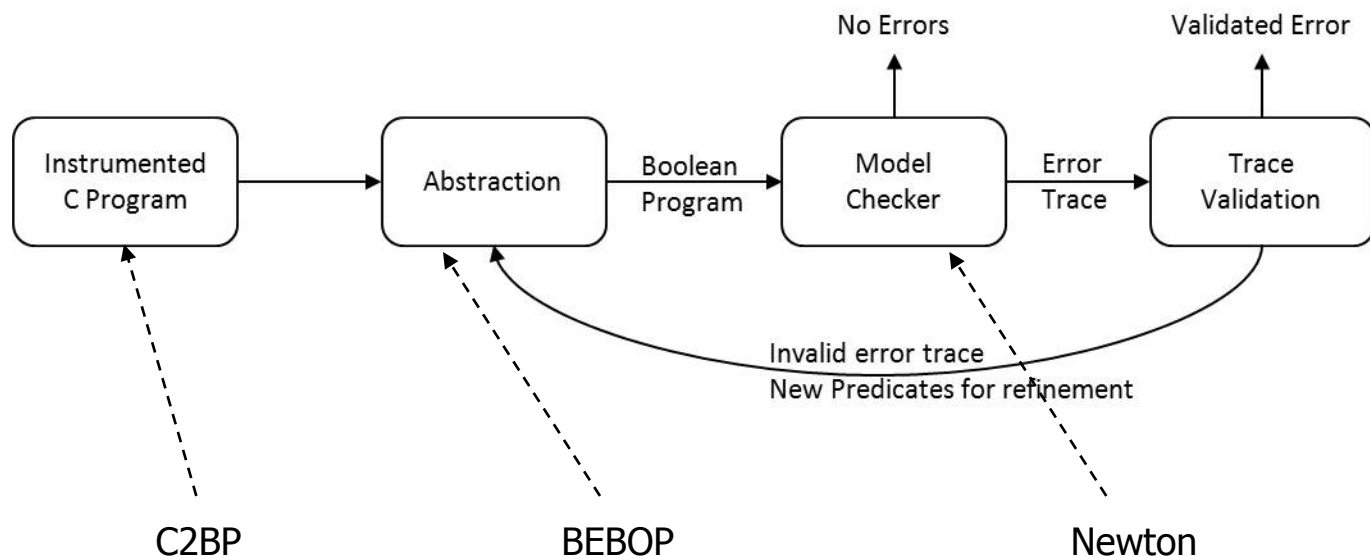
Példa: SLAM

- Motiváció: Hibás meghajtók 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



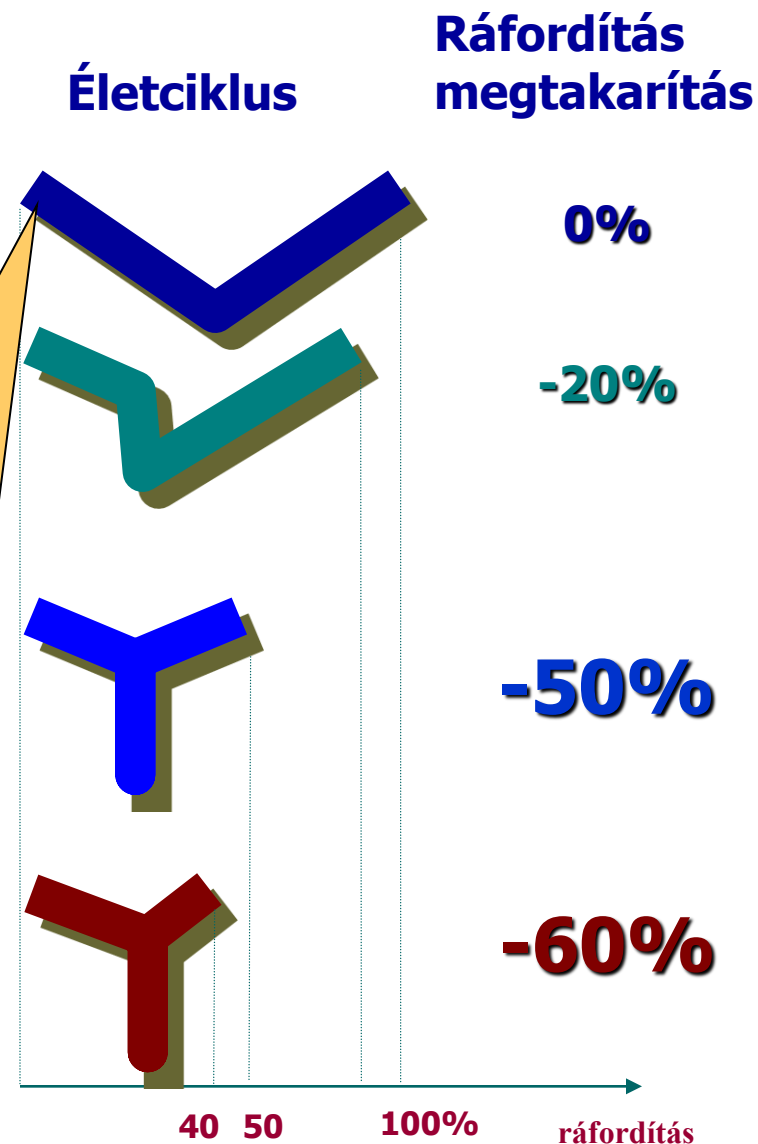
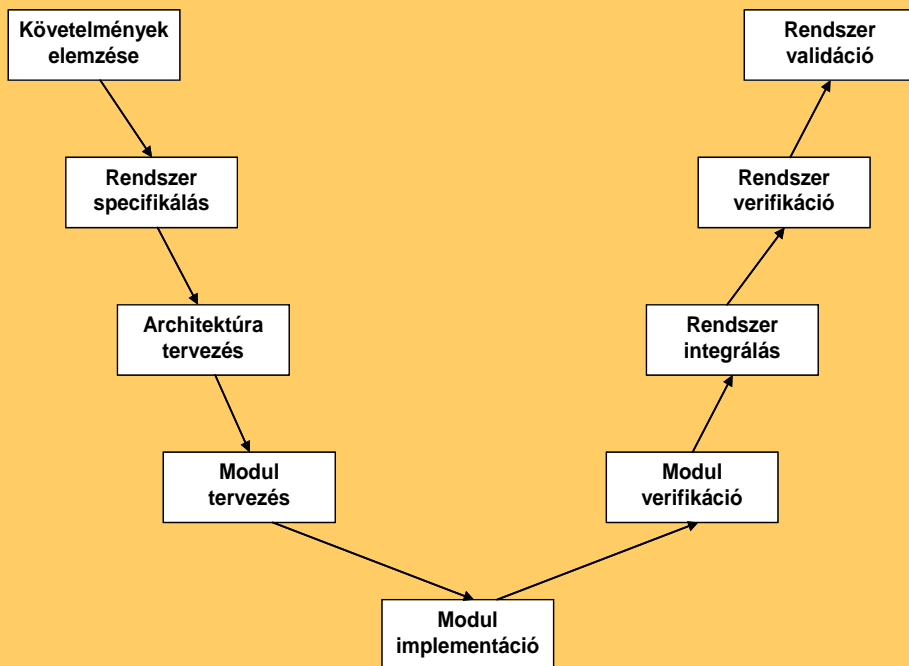
Példa: SLAM

- Motiváció: Hibás meghajtók 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



V-től az Y fejlesztési modellig

Szoftverfejlesztés a V-modell szerint



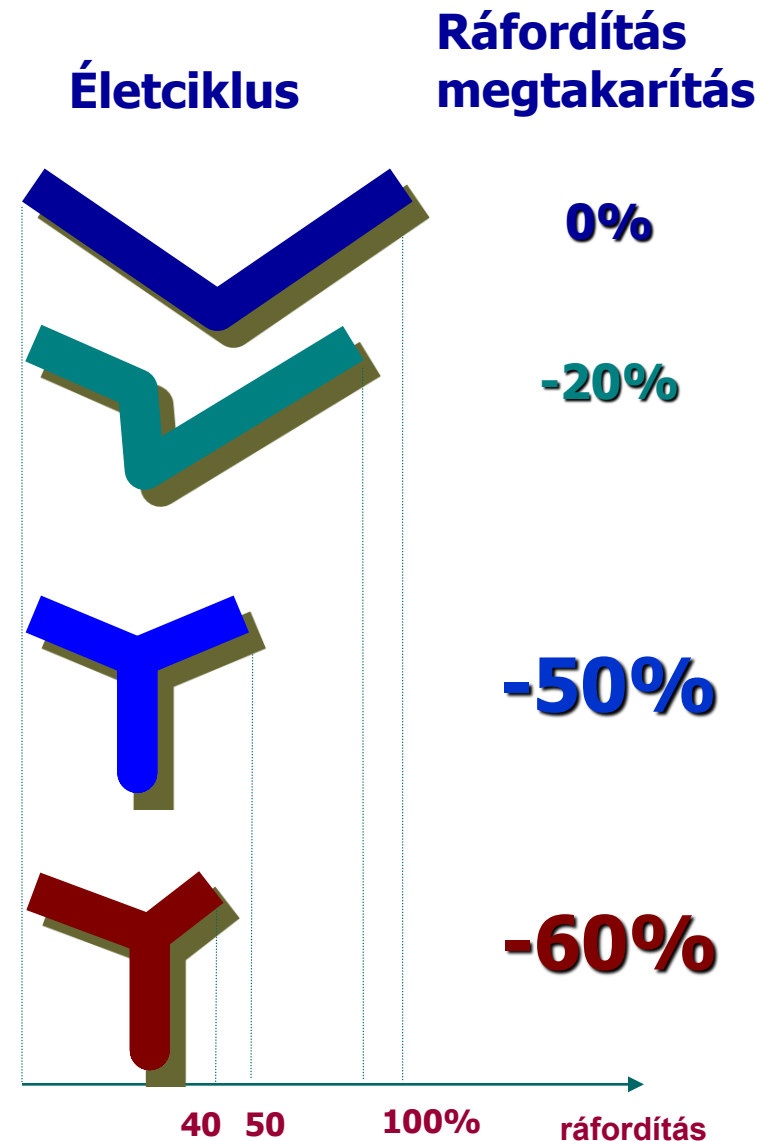
V-től az Y fejlesztési modellig

Kézi kódolás

“Közönséges” automatikus kódgenerátor használata

Minősített automatikus kódgenerátor használata

Formális verifikációval kiegészített tervezés



* Adatok: Esterel Technologies (Ansys)

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