A formális módszerek szerepe

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

BME Méréstechnika és Információs Rendszerek Tanszék

Formális módszerek

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

  használata arra, hogy elkészítsük és ellenőrizzük hardver és szoftver rendszerek
  - specifikációját (követelményeit),
  - terveit (modelljeit),
  - implementációját (megvalósítását),
  - dokumentációját.

Mire jók a formális módszerek?

Egy jellegzetes példa:
Algoritmus ellenőrzése

Egy mérnöki feladat

- Többprocesszes alkalmazás
- 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
- Hogyan valósítsuk meg?
  - Klasszikus megoldások
  - Saját algoritmus

Kölcsönös kizárás algoritmus (pszeudo-kód)

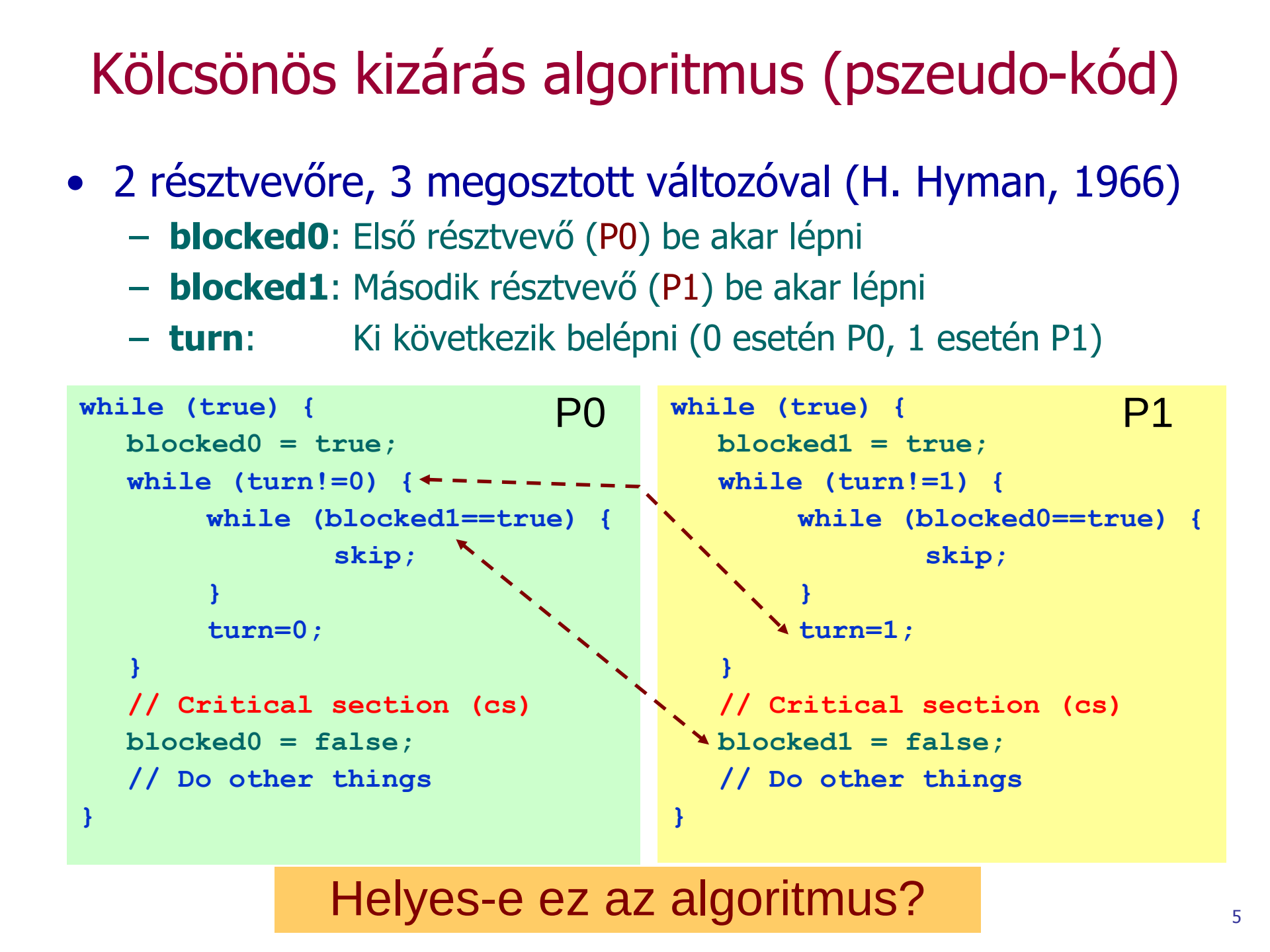
- 2 résztvevőre, 3 megosztott változóval (H. Hyman, 1966)
  - **blocked0**: Első résztvevő (P0) be akar lépni
  - **blocked1**: Második résztvevő (P1) be akar lépni
  - **turn**: Ki következik belépni (0 esetén P0, 1 esetén P1)

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?

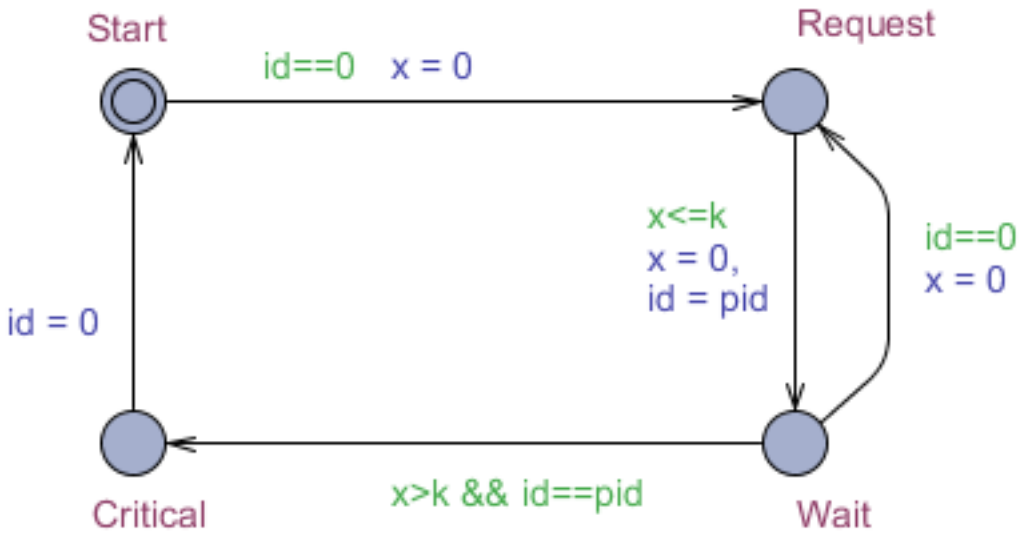
- Kölcsönös kizárás biztosított:
  - Egyszerre csak az egyik processz (P0 vagy P1) lehet a kritikus szakaszban
- Lehetséges az elvárt viselkedés:
  - P0 egyáltalán beléphet a kritikus szakaszba
  - P1 egyáltalán beléphet a kritikus szakaszba
- Nincs „kiéheztetés”:
  - P0 mindenképpen be fog lépni a kritikus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

Hogyan ellenőrizhetjük a követelményeket?

- Megvalósítással és annak tesztelésével
  - Létre tudunk-e hozni minden lehetséges végrehajtást (azaz lehetséges átlapoló utasítássorrendet) lefedő teszt eseteket?
  - A problémás esetek 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

Készítsünk formális modellt!

- Automata formalizmus:
  - Állapotok és állapotátmenetek
  - Változók, konstansok
  - Változókon kiértékelhető feltételek az átmenetek végrehajtásához
  - Értékadás akciók az átmenetek végrehajtása során
- Szintaxis példa (diagram):

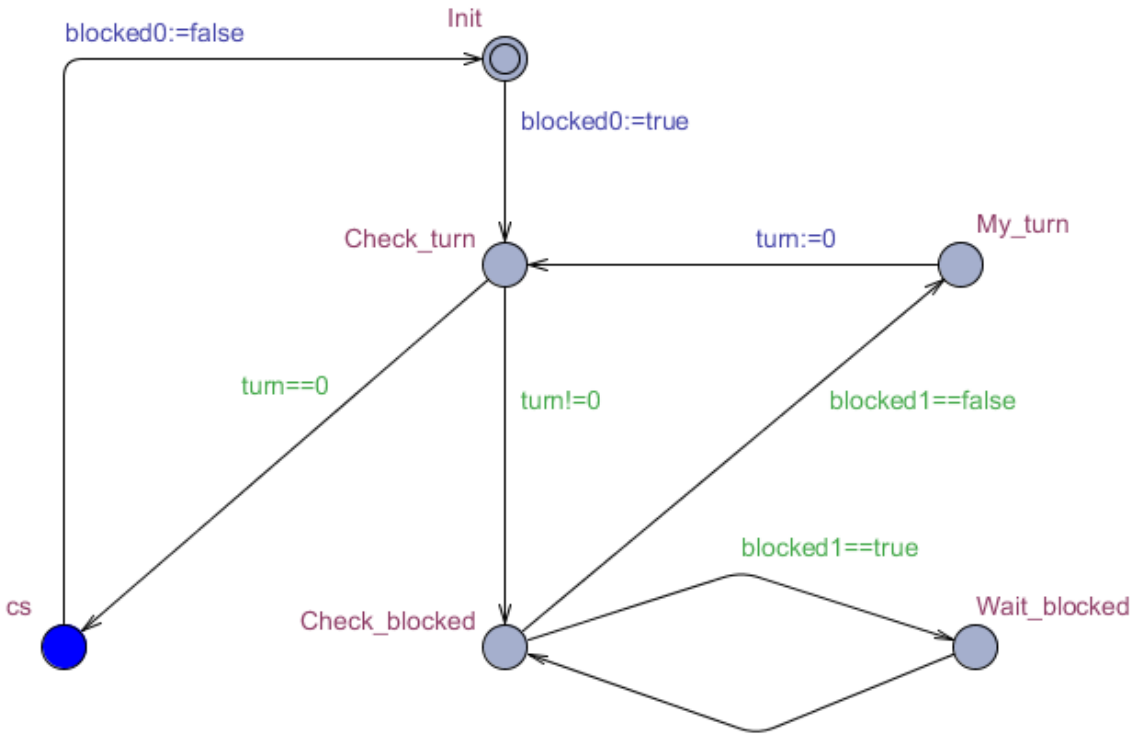


A P0 processz formális modellje

Deklarációk:

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

A P0 automata:



P0

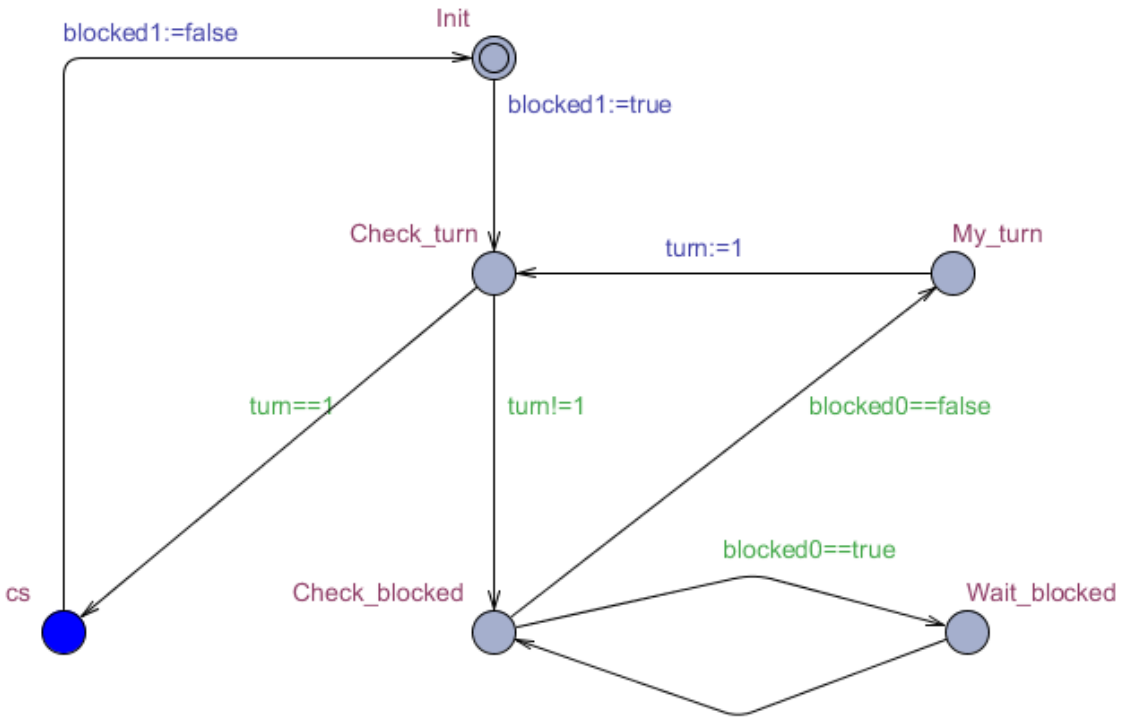
```
while (true) {
    blocked0 = true;
    while (turn!=0) {
        while (blocked1==true) {
            skip;
        }
        turn=0;
    }
    // Critical section
    blocked0 = false;
    // Do other things
}
```

A P1 processz formális modellje

Deklarációk:

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

A P1 automata:



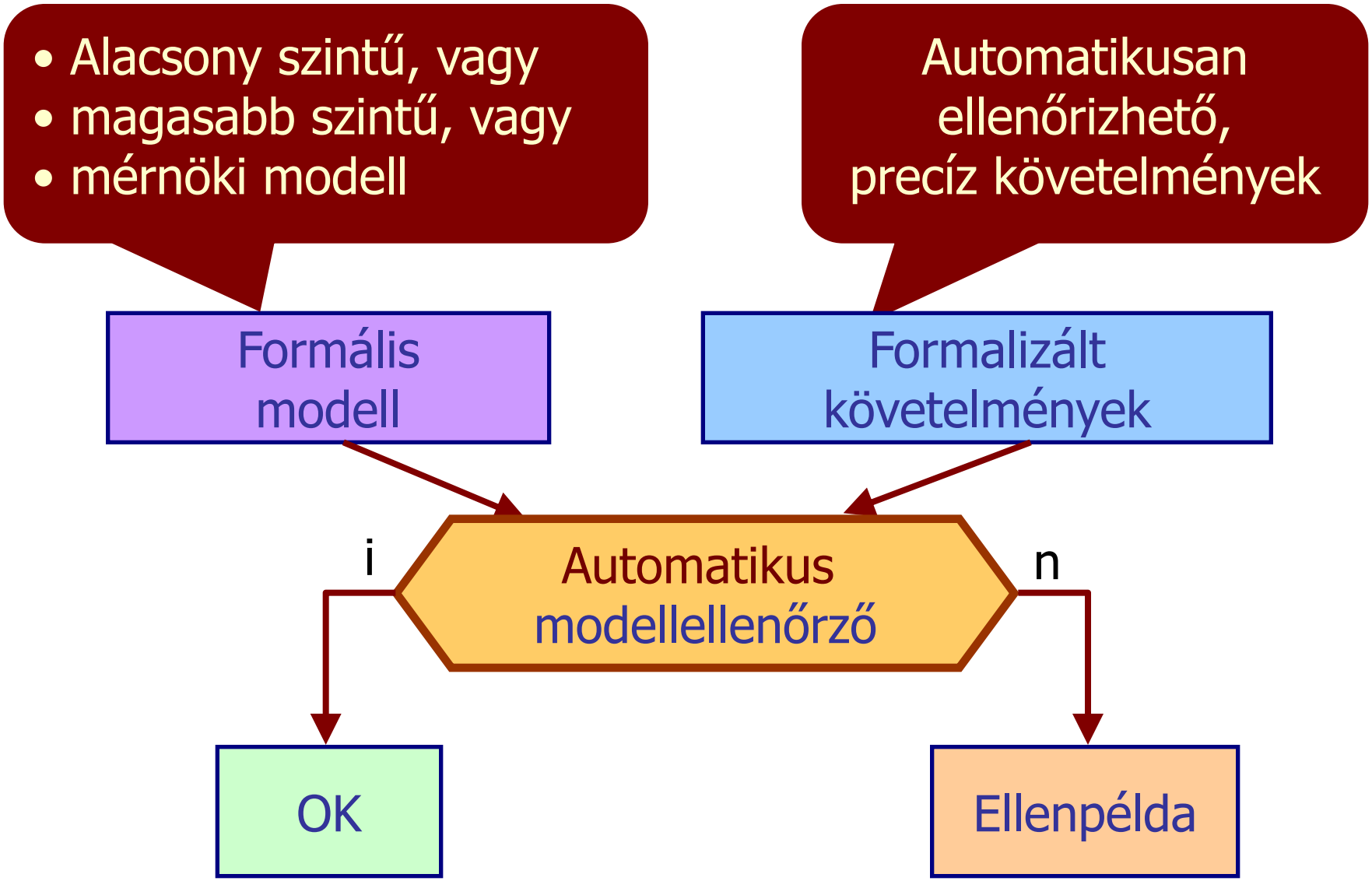
P1

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

Hogyan ellenőrizhetjük a követelményeket?

- Megvalósítással és annak tesztelésével
  - Létre tudunk-e hozni minden lehetséges végrehajtást (azaz lehetséges átlapoló utasítássorrendet) lefedő teszt eseteket?
  - A problémás esetek 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
- Modellezéssel és az állapottér teljes ellenőrzésével
  - Minden lehetséges végrehajtást ellenőriz: Modell állapottér bejárás szisztematikus algoritmussal, „gombnyomásra”
  - Automatikusan ellenőrzött követelmények: Leíró nyelv és hatékony ellenőrző algoritmus kell
  - Modell szinten javítható hibák: Hiba esetén (ellen)példát ad

Egy jellegzetes formális ellenőrzés (verifikáció)



Az algoritmus javítása

Peterson algoritmusa:

- P0 résztvevőre (P1 hasonlóan):

Hyman:

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

Peterson:

```
while (true) {
    blocked0 = true;
    turn=1;
    while (blocked1==true &&
        turn!=0) {
            skip;
    }

    // Critical section
    blocked0 = false;
    // Do other things
}
```

A formális módszerek szerepe (áttekintés)

Mit szeretnénk elérni?

- Valós probléma formális vizsgálata:
  1. Probléma formalizálása, formális modell készítése <- feltételezések, absztrakció
  2. Formális modell analízise <- automatikus eszközök is
  3. Eredmények értelmezése, felhasználása <- alkalmazhatóság

- Mi kell ehhez?
  1. Formális nyelv a probléma leírására
  2. Eljárás a formális modell analízisére
  3. Eredmények érvényessége (ld. feltételezések, eszközök)

Első lépés: Formális nyelv

- A formalizálás célja: Matematikai precizitással megadni
  - Modelleket: terveket, tervezői döntéseket (modellezési nyelv)
  - Követelményeket: elvárt tulajdonságokat (követelmény leíró nyelv)
- Formális nyelvek felépítése
  - Formális szintaxis
    - Jelölésmód: milyen nyelvi elemek és kapcsolatok vannak
  - Formális szemantika
    - A jelölésmód interpretációja: mit értek alatta
- Mit szeretnénk leírni formális nyelvekkel?
  - Funkcionalitás (viselkedés, feltételek, elvárások, …)
  - Struktúra, interfészek
  - Extra-funkcionális aspektusok is: Teljesítmény, megbízhatóság, …
- A formális nyelv használatának előnyei
  - Egyértelműség, ellenőrizhetőség
  - Automatikus feldolgozhatóság

Továbblépés: Formális módszerek

Formális módszer:

- A formális modellről ismeretet adó matematikai eljárás
- Eszközökkel támogatható

- A formális modell végrehajtása
  - Szimuláció
- A formális modell ellenőrzése: Formális verifikáció
  - „Önmagában való” vizsgálat
    - Konzisztencia, ellentmondás-mentesség
    - Teljesség, zártság
  - „Megfelelés” vizsgálata
    - Modellek között
    - Modellek és elvárt tulajdonságok között (implementáció ↔ specifikáció)
- A formális modell alapján történő szintézis:
  - Szoftver (programkód, konfiguráció) generálása
  - Hardver tervek generálása

Mire szeretnénk használni a formális módszereket?

Milyen problémákkal nézünk szembe?

Miben segíthetnek a formális módszerek?

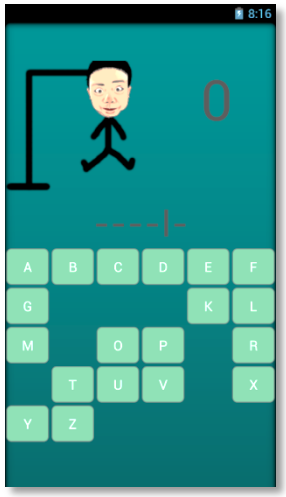
Egy tanulságos történet…

- Vasa svéd hadihajó, 1628:
  Elsüllyedt közvetlenül
  a vízrebocsátás után

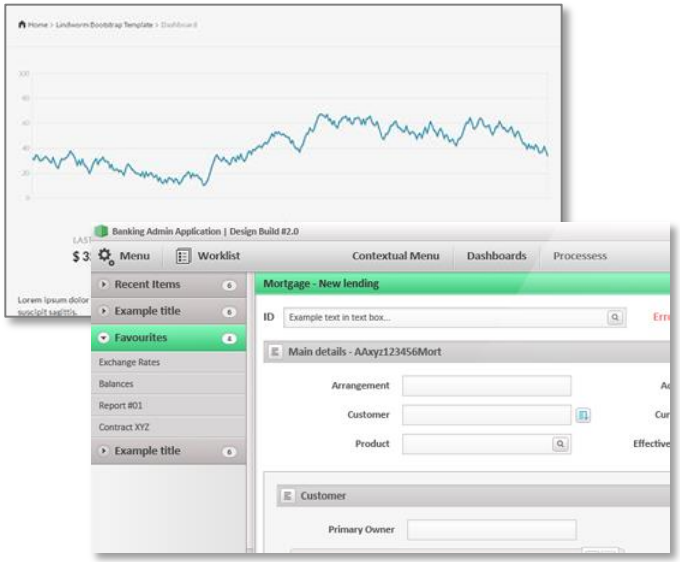
- Problémák:
  - Változó követelmények
    (II. Gusztáv Adolf király)
  - Hiányzó pontos specifikáció
    (Henrik Hybertsson építő)
  - Ellenőrizetlen tervek
    (Johan Isbrandsson alvállalkozó)
  - Figyelmeztetések figyelmen kívül hagyása
    (Fleming admirális)

- Dokumentáció:
  - The Vasa: A Disaster Story with Software Analogies. By Linda Rising. The Software Practitioner, January-February 2001.
  - Why the Vasa Sank: 10 Problems and Some Antidotes for Software Projects. By Richard E. Fairley and Mary Jane Willshire. IEEE Software, March-April 2003.

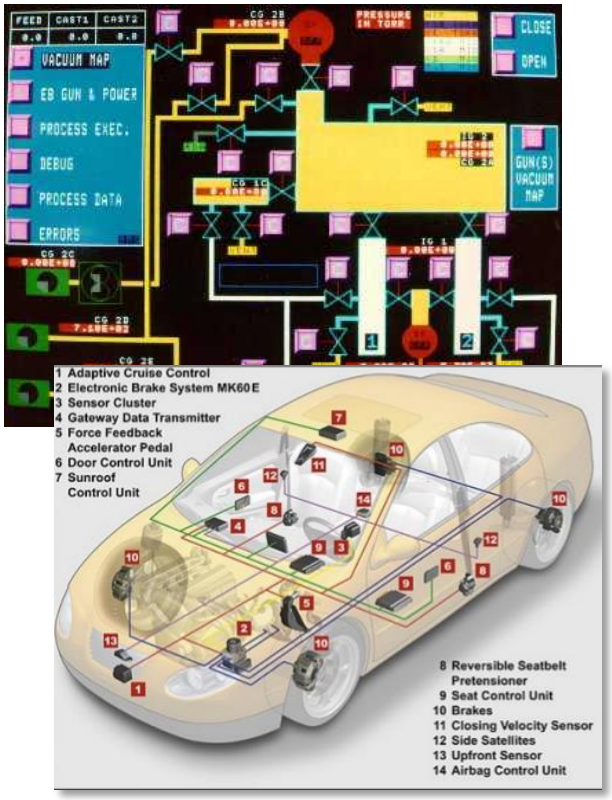
Komplex rendszerek tervezése



- Minimális tervezés
- Implicit folyamat
- Egyszerű eszközök



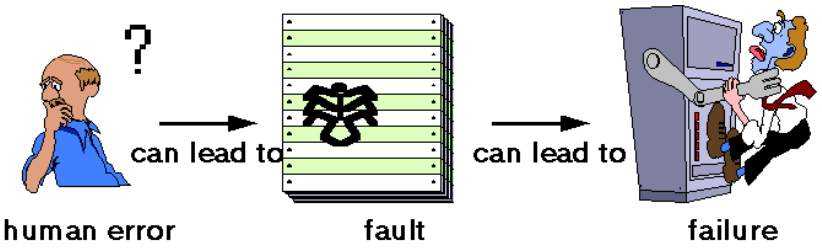
- Alapos tervezés
- Definiált folyamat
- Hatékony eszközök



- Ellenőrzött tervek
- Meghatározott folyamat
- Automatikus eszközök

Szoftver minőségi krízis

- Tipikus kódméret:
  - 10 kLOC … 1000 kLOC
- Fejlesztési ráfordítás:
  - 0,1 - 0,5 mérnökév / kLOC (nagyméretű szoftver)
  - 5-10 mérnökév / kLOC (kritikus szoftver)
- Hiba eltávolítás (ellenőrzés, tesztelés, javítás):
  - 45 - 75% ráfordítás
- Hibasűrűség változása:
  - 10 - 200 hiba / kLOC jön létre a fejlesztés során



Ellenőrzés, debuggolás, javítás

  - 0,01 - 10 hiba / kLOC maradhat az üzembe helyezésig

IT alkalmazások fejlesztésének kihívásai

- Jó minőségű specifikáció és tervek készítése
  - Teljes
  - Ellentmondás-mentes, egyértelmű
  - Ellenőrizhető
- Tervek ellenőrzése
  - Tervezői döntések igazolása
    - Bizonyítottan helyes tervek a továbblépés alapjai
  - Hibák elkerülése vagy korai felderítése
    - Minőség ↔ költség ↔ fejlesztési idő optimalizálás
- Bizonyított helyességű tervezői eszközök használata
  - Forráskód, konfiguráció, teszt és monitor szintézis

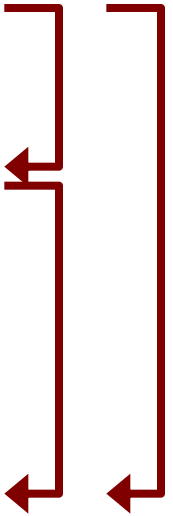
Ezek alapjait a formális módszerek adhatják!

Verifikáció és validáció összehasonlítása

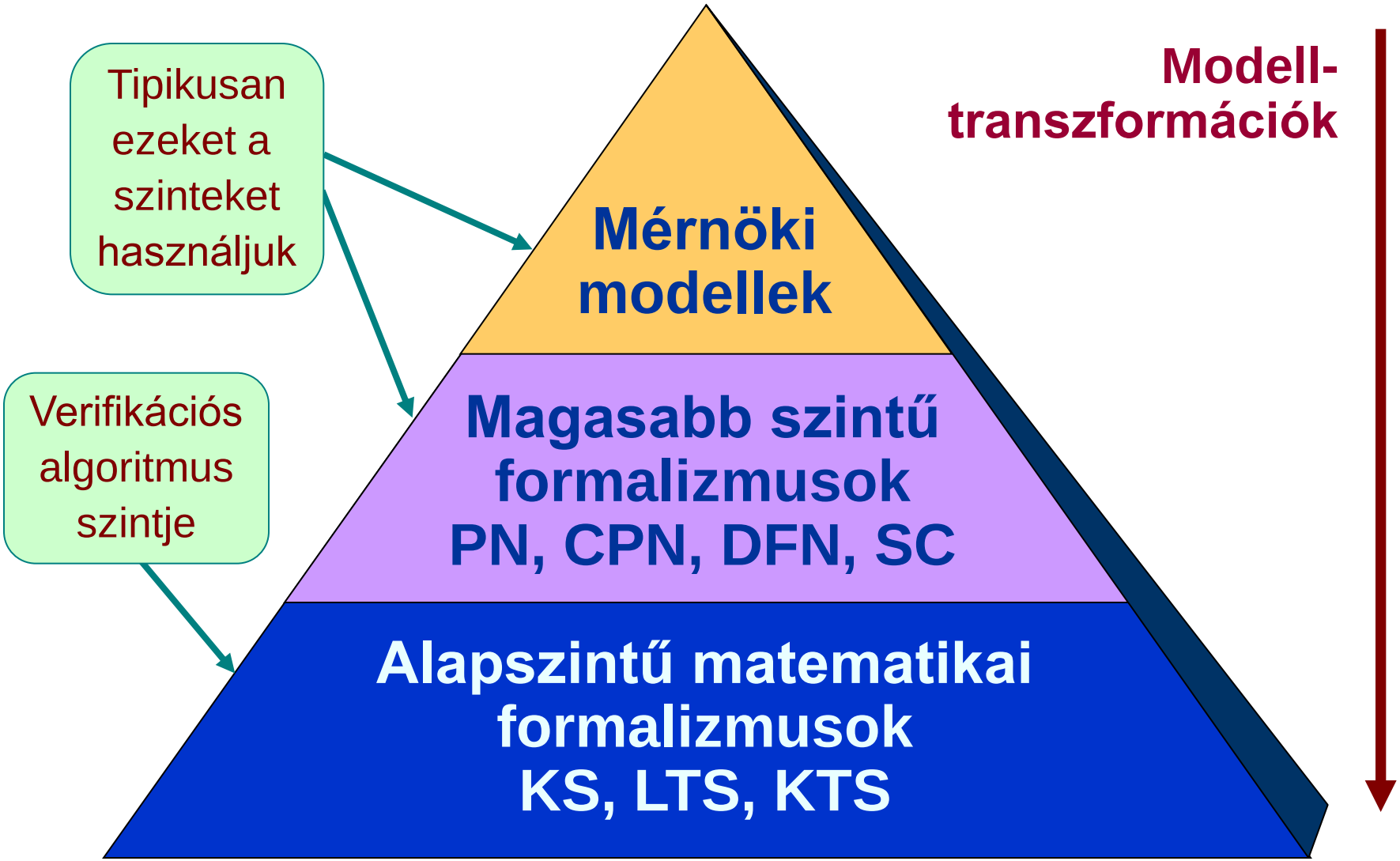
Verifikáció (igazolás)	Validáció (érvényesítés)
„Jól építjük-e a rendszert?”	„Jó rendszert építettünk-e?”
Összhang ellenőrzése a fejlesztési fázisokban, illetve ezek között	A fejlesztés eredményének ellenőrzése
Fejlesztési lépések során használt tervek (modellek) és specifikációjuk közötti megfelelés 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 követelmény és implementáció között	Nincs rá szükség, ha a specifikáció tökéletes (elég egyszerű)

Modellek a formális ellenőrzéshez

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



A tárgy felépítése



Klasszikus 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 spec. nyelv + Ada)
  - Költség nem nőtt a tesztelés egyszerűsödése miatt
- IEEE Futurebus+ szabvány
  - Carnegie Mellon SMV: cache koherencia protokoll hibájának kiderítése
- Hardver projektek: ACL2 automatikus tételbizonyító
  - Motorola DSP Complex Arithmetic Processor mag (250 regiszter): DSP algoritmusok ellenőrzése
  - AMD 5K86 processzor: Lebegőpontos osztás algoritmusának ellenőrzése
- Intel 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, Esterel, Prover, Mentor, Telelogic, …

Forráskódhoz illeszkedő formális verifikáció

- Java
  - Bandera, 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 szoftver modellellenőrzése
- 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 Research Platform, SLAM 2 eszköz
  - Windows API használati feltételeinek statikus ellenőrzése

Összefoglalás

- Mik a formális módszerek?
  - Formalizmus, formális nyelv
  - Formális módszerek és eszközök: Szimuláció, formális verifikáció, szintézis
- Mire használhatók?
  - Motiváció: Szoftver minőségi kihívások
  - A formális módszerek lehetőségei
- Mit várhatunk?
  - Korlátok
  - Sikertörténetek