Formális modellezés és verifikáció

Rendszertervezés és -integráció előadás

dr. Majzik István

Méréstechnika és
Információs Rendszerek
Tanszék

Célkitűzések

Egy mérnöki feladat

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

Egy lehetséges algoritmus

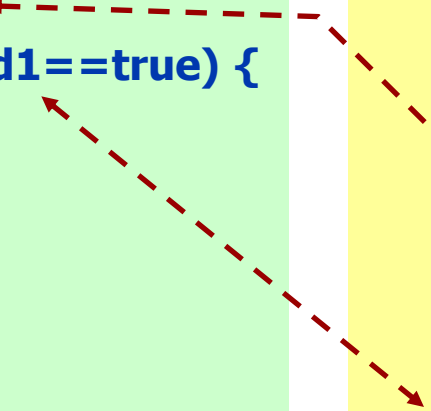
- Kölcsönös kizárás 2 résztvevőre, 3 megosztott változóval (Hyman, 1966)
  - **blocked0**: Első résztvevő (P0) be akar lépni
  - **blocked1**: Második résztvevő (P1) be akar lépni
  - **turn**: Ki következik belépni (0 esetén P0, 1 esetén P1)

P0

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

P1

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



Helyes-e ez az algoritmus?

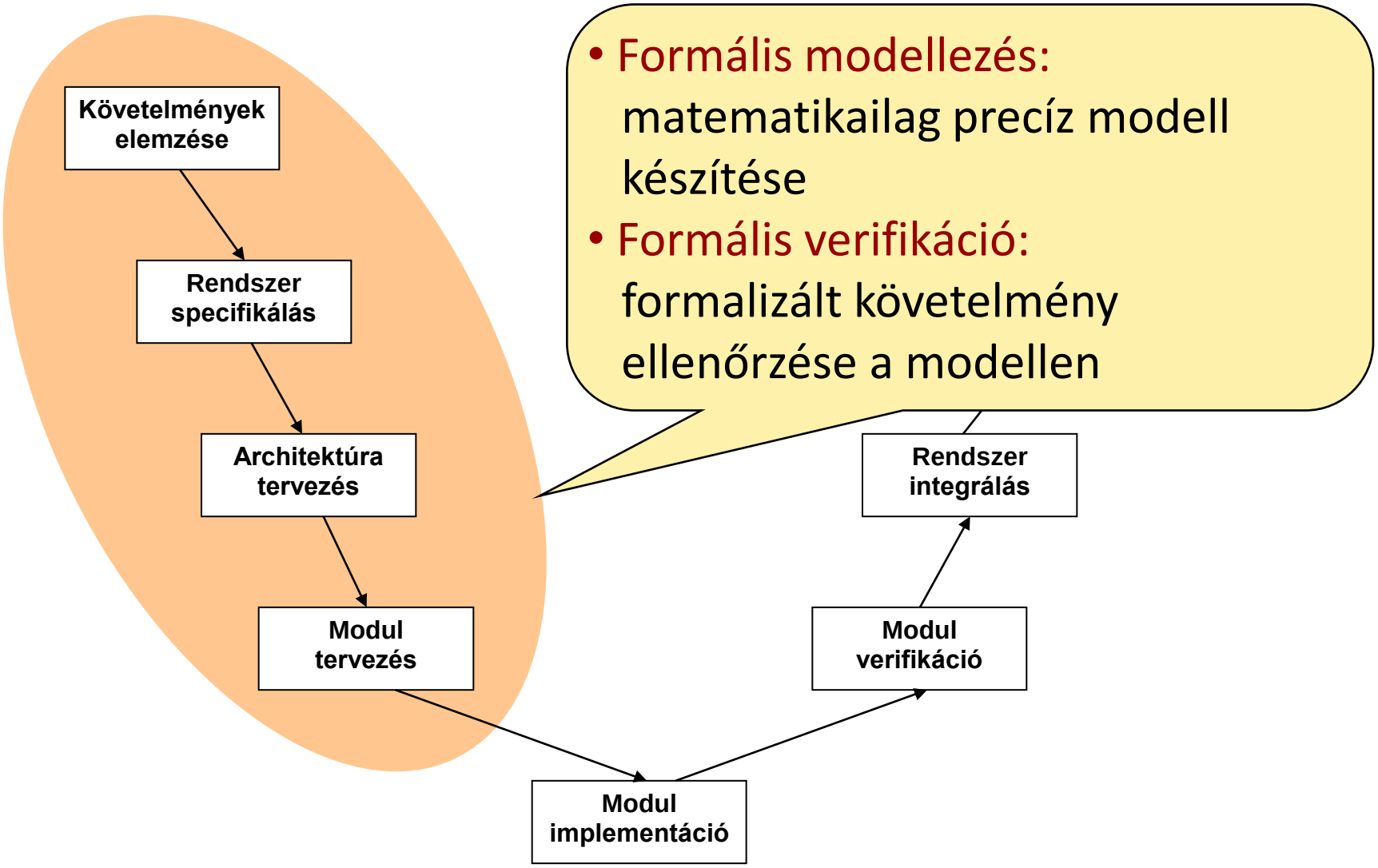
Mit jelent a helyesség?

- Kölcsönös kizárás:
  - Egyszerre csak az egyik résztvevő lehet a kritikus szakaszban
- Lehetséges az elvárt viselkedés:
  - P0 egyáltalán be tud lépni a kritikus szakaszba
  - P1 egyáltalán be tud lépni 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?

- Implementációval majd teszteléssel
  - Létre tudunk-e hozni minden lehetséges végrehajtást lefedő teszteseteket? (Itt konkurens processzekről van szó!)
  - A problémás esetek figyeléséhez külön ellenőrző kell
  - A hiba csak az implementáció után derül ki, drágán javítható
- Modellezéssel és szimulációval
  - 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
  - Szisztematikus algoritmus az állapottér teljes felvételéhez
  - Automatikus a követelmények teljesülésének figyelése, – ha ehhez a helyességi követelményeket formalizáljuk

Ellenőrzések a tervezési fázisban



Módszerek és intézkedések

- IEC 61508:
  Functional safety in electrical / electronic / programmable electronic safety-related systems
- Példa:
  Szoftver architektúra tervezés

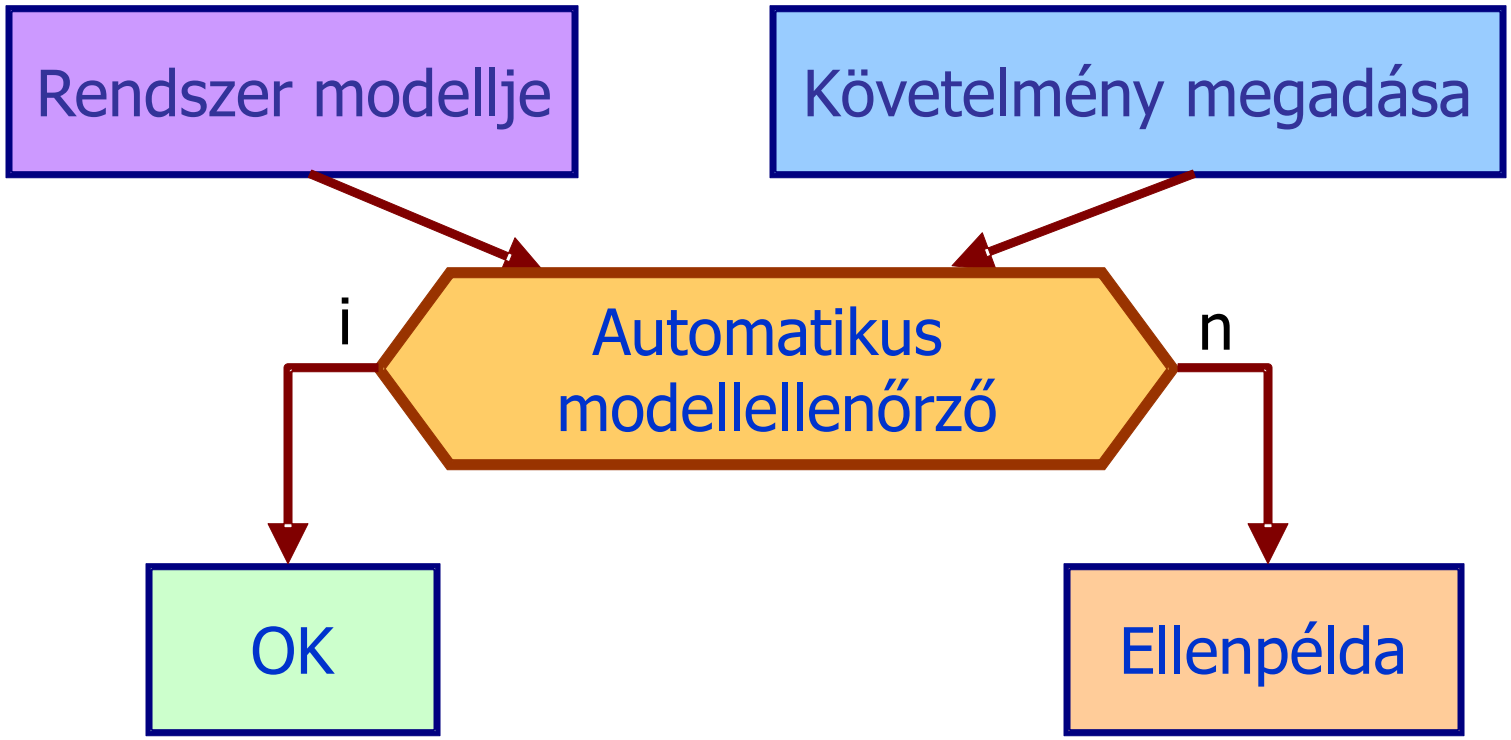
Table A.2 – Software design and development: software architecture design (see 7.4.3)

	Technique/Measure*	Ref	SIL1	SIL2	SIL3	SIL4
1	Fault detection and diagnosis	C.3.1	---	R	HR	HR
2	Error detecting and correcting codes	C.3.2	R	R	R	HR
3a	Failure assertion programming	C.3.3	R	R	R	HR
3b	Safety bag techniques	C.3.4	---	R	R	R
3c	Diverse programming	C.3.5	R	R	R	HR
3d	Recovery block	C.3.6	R	R	R	R
3e	Backward recovery	C.3.7	R	R	R	R
3f	Forward recovery	C.3.8	R	R	R	R
3g	Re-try fault recovery mechanisms	C.3.9	R	R	R	HR
3h	Memorising executed cases	C.3.10	---	R	R	HR
4	Graceful degradation	C.3.11	R	R	HR	HR
5	Artificial intelligence - fault correction	C.3.12	---	NR	NR	NR
6	Dynamic reconfiguration	C.3.13	---	NR	NR	NR
7a	Structured methods including for example, JSD, MASCOT, SADT and Yourdon.	C.2.1	HR	HR	HR	HR
7b	Semi-formal methods	Table B.7	R	R	HR	HR
7c	Formal methods including for example, CCS, CSP, HOL, LOTOS, OBJ, temporal logic, VDM and Z	C.2.4	---	R	R	HR
8	Computer-aided specification tools	B.2.4	R	R	HR	HR

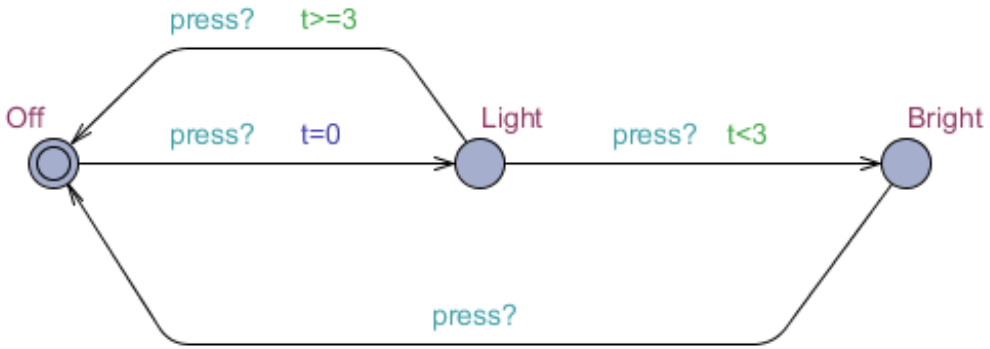
NOTE – The measures in this table concerning fault tolerance (control of failures) should be considered with the requirements for architecture and control of failures for the hardware of the programmable electronics in IEC 61508-2.

* Appropriate techniques/measures shall be selected according to the safety integrity level. Alternate or equivalent techniques/measures are indicated by a letter following the number. Only one of the alternate or equivalent techniques/measures has to be satisfied.

Mit szeretnénk elérni?

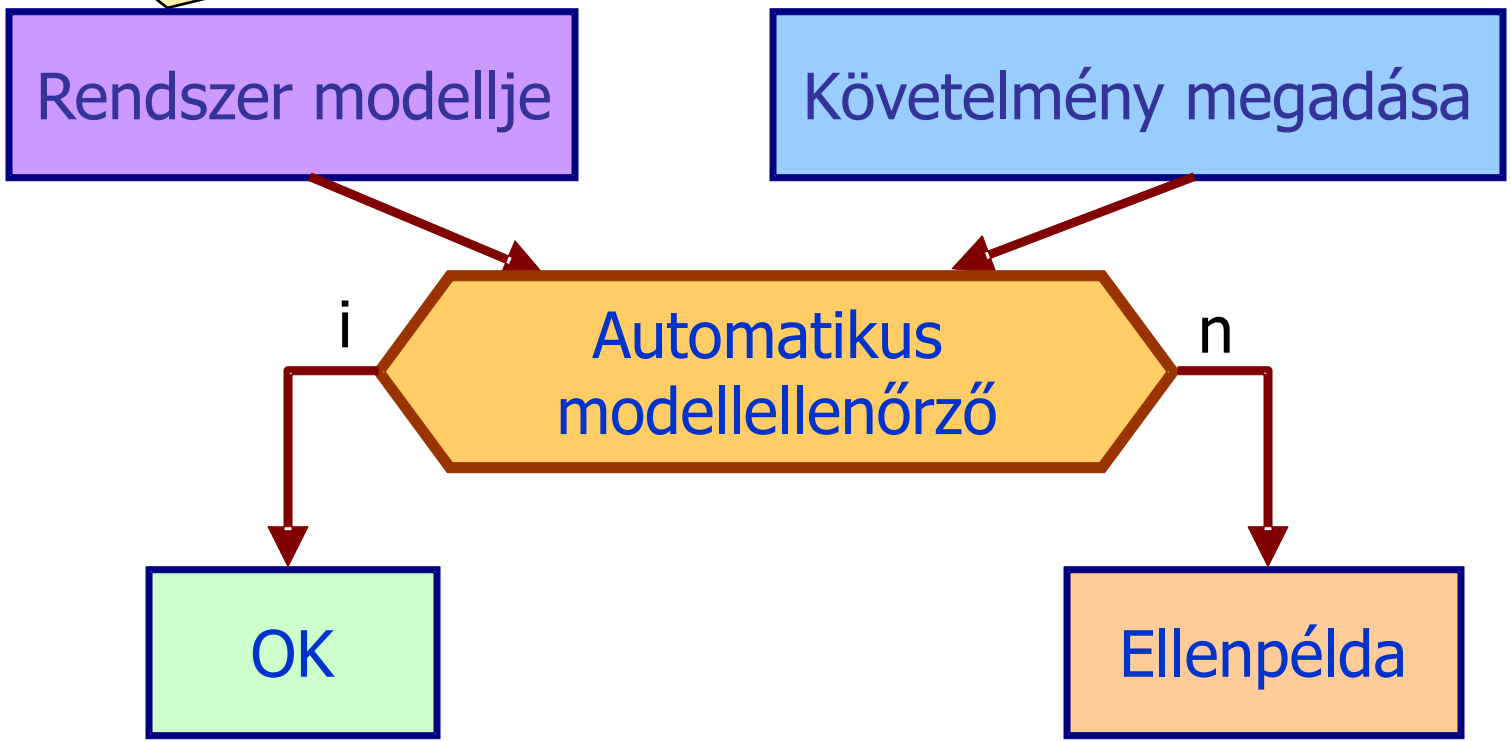


Modellezés: Időzített automaták



Mit szeretnénk elérni?

- Modellezés időzített automatákkal
- Magasabb szintű modellekből automatikus transzformációval származtatható



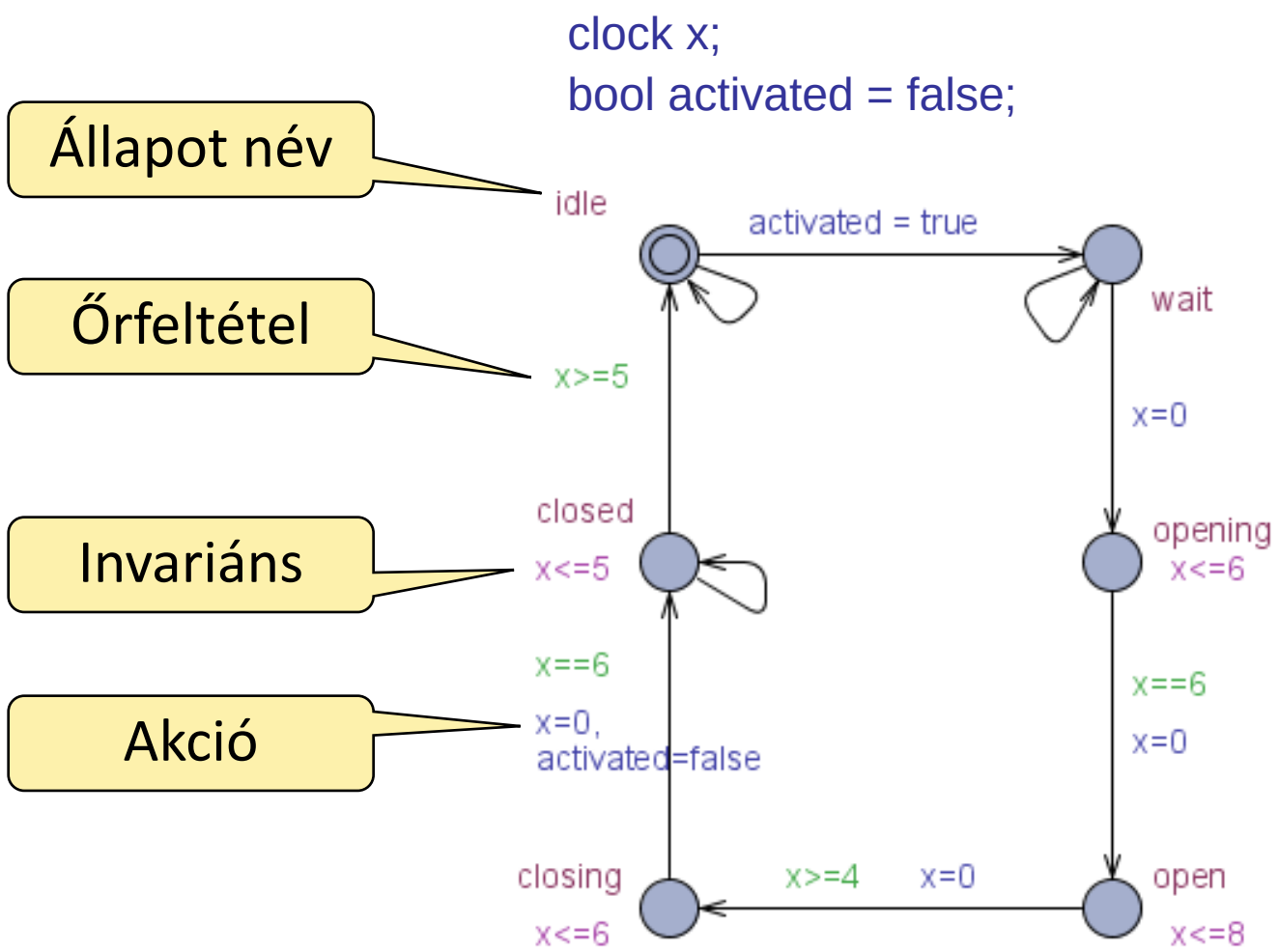
Automaták és változók

- Cél: Állapot alapú viselkedés modellezése
- Alap formalizmus: Véges állapotú automata (FSM)
  - Állapotok (más néven vezérlési helyek, névvel hivatkozhatók)
  - Állapotátmenetek
- Kiterjesztés: Egész értékű változók használata
  - Számítások, adatmanipuláció modellezése (egész aritmetikával)
  - Változók típusa, értéktartománya megadható
  - Konstansok definiálhatók
- Állapotátmenetek kiterjesztése:
  - Őrfeltétel hozzárendelése: A változókon kiértékelhető predikátum, az átmenet bekövetkezéséhez igaz kell legyen
  - Akció hozzárendelése: Értékadás változóknak

Kiterjesztések óraváltozókkal

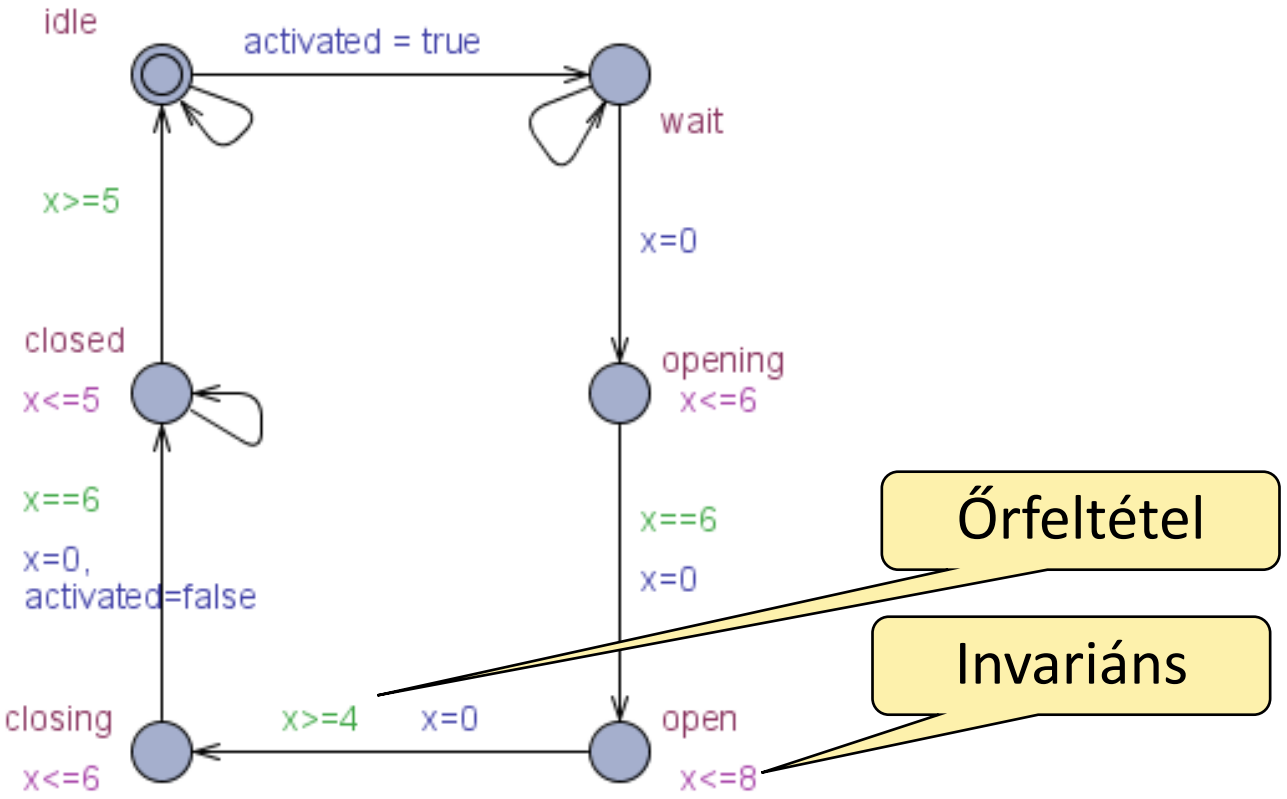
- Cél: Idő függvényében változó viselkedés vizsgálata
  - Idő telik az állapotokban
  - Eltelt időtől (nem csak az állapottól) függő viselkedés
  - Gyakorlati megvalósítás: időzítők (számlálók)
- Modellezési kiterjesztés: Óraváltozók
  - Azonos lépésben „haladó” konkurens órák (időzítők modellje)
  - Relatív időmérés (pl. time-out): Időzítők resetelése és leolvasása
- Használat állapotátmenetekben:
  - Akciók: Óraváltozók nullázása (resetelés), egymástól függetlenül
  - Őrfeltételek: Óraváltozók és konstansok a feltételekben
- Használat állapotokban:
  - Állapot invariánsok: Óraváltozók és konstansok segítségével megadja, meddig állhat fenn az adott állapot

Időzített automata



Az invariánsok és őrfeltételek szerepe

```
clock x;
bool activated = false;
```

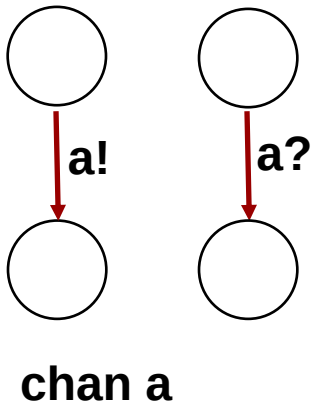


Az open állapot elhagyásakor a [4, 8] tartományban lehet x értéke

Kiterjesztések elosztott rendszerekhez

- Cél: Együttműködő automaták hálózatának modellezése
  - Itt: Különálló automaták átmeneteinek együttes végrehajtása
  - Együttlépő átmenetek (randevú): szinkron kommunikáció
    - Üzenet küldés és fogadás csak együtt valósulhat meg (küldő vár)
    - Ezzel aszinkron kommunikáció is leírható

- Kiterjesztés: Szinkronizált akciók
  - Csatornák definiálása (szinkron csatorna)
  - Üzenetküldés: ! operátor a csatornára
    Üzenetfogadás: ? operátor a csatornára
    - Pl: az a nevű csatorna esetén a! és a? akciók
  - Csatornatömbök is használhatók
    - Pl. a[id] csatorna az id változóval indexelve

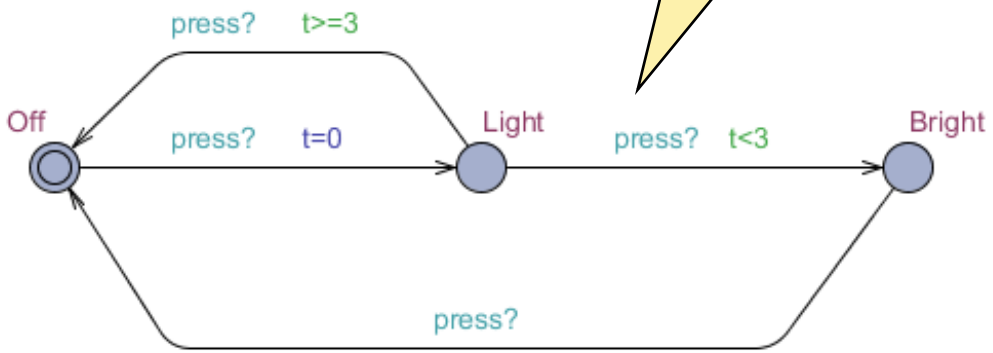


Példa óraváltozókra és szinkronizálásra

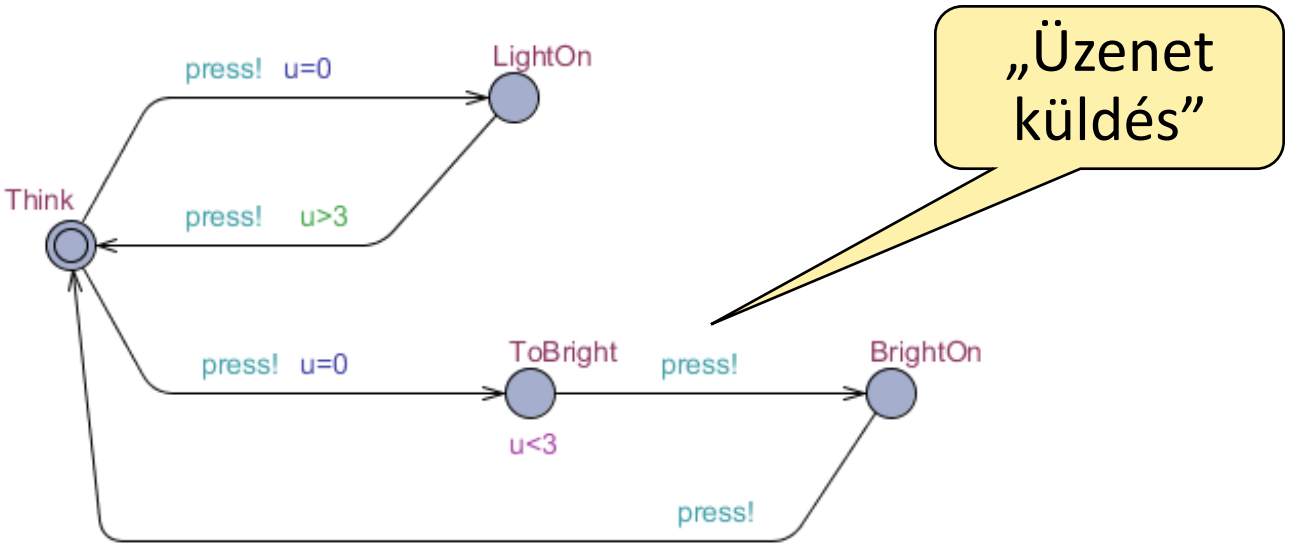
Deklarációk:

```
clock t, u;
chan press;
```

Kapcsoló:

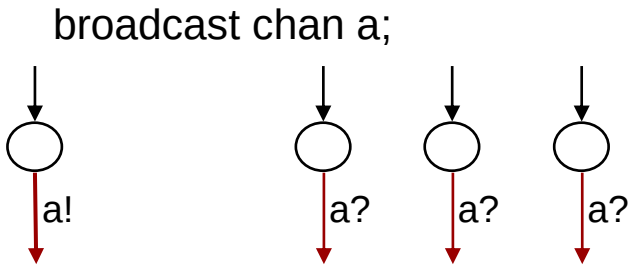


Felhasználó:

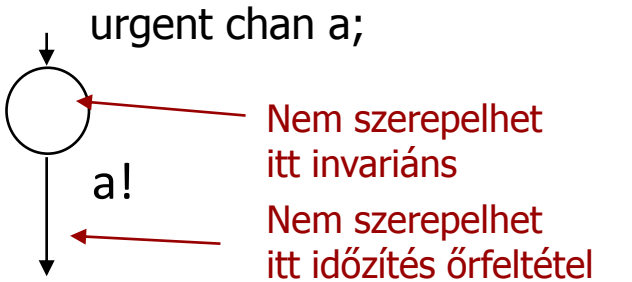


További lehetőségek

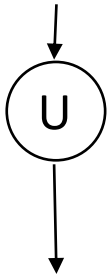
- Broadcast csatorna
  - Egy küldő (mindig tud küldeni)
  - Több fogadó (ha éppen tud, akkor szinkronizál a küldővel)



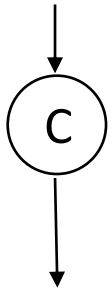
- Urgent csatorna: késleltetés korlátozása
  - Késleltetés nélkül, azonnal végrehajtandó szinkronizáció (de előtte átlapolva azonnali végrehajtás lehet)



- Urgent állapot: késleltetés korlátozása
  - Nem telhet idő az adott állapotban (különben hiba)



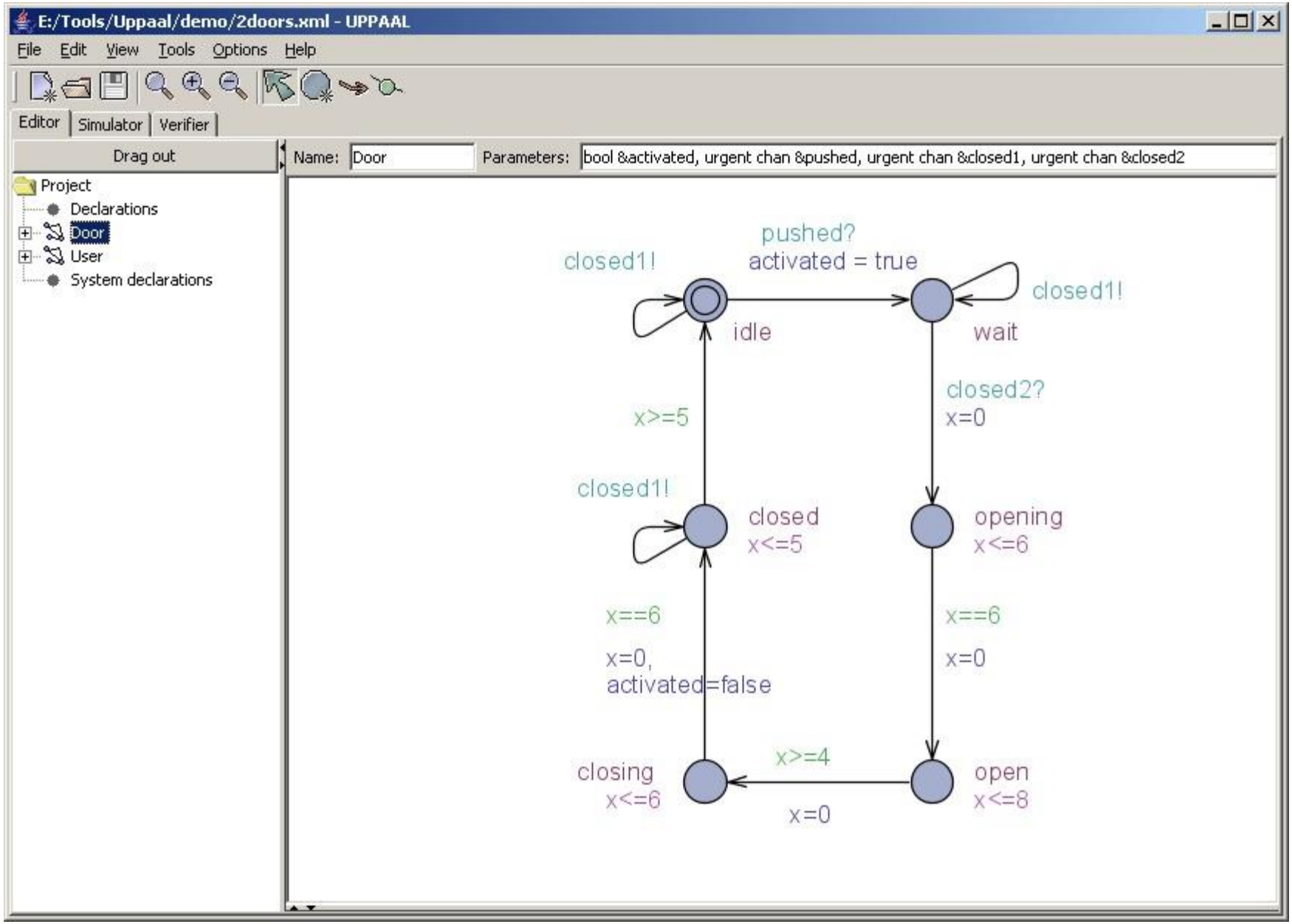
- Committed állapot: Atomi végrehajtást ír elő
  - A kimenő átmenet végrehajtása előtt más automata átmenete nem lehet végrehajtva: bemenő és kimenő átmenet egy atomi művelet



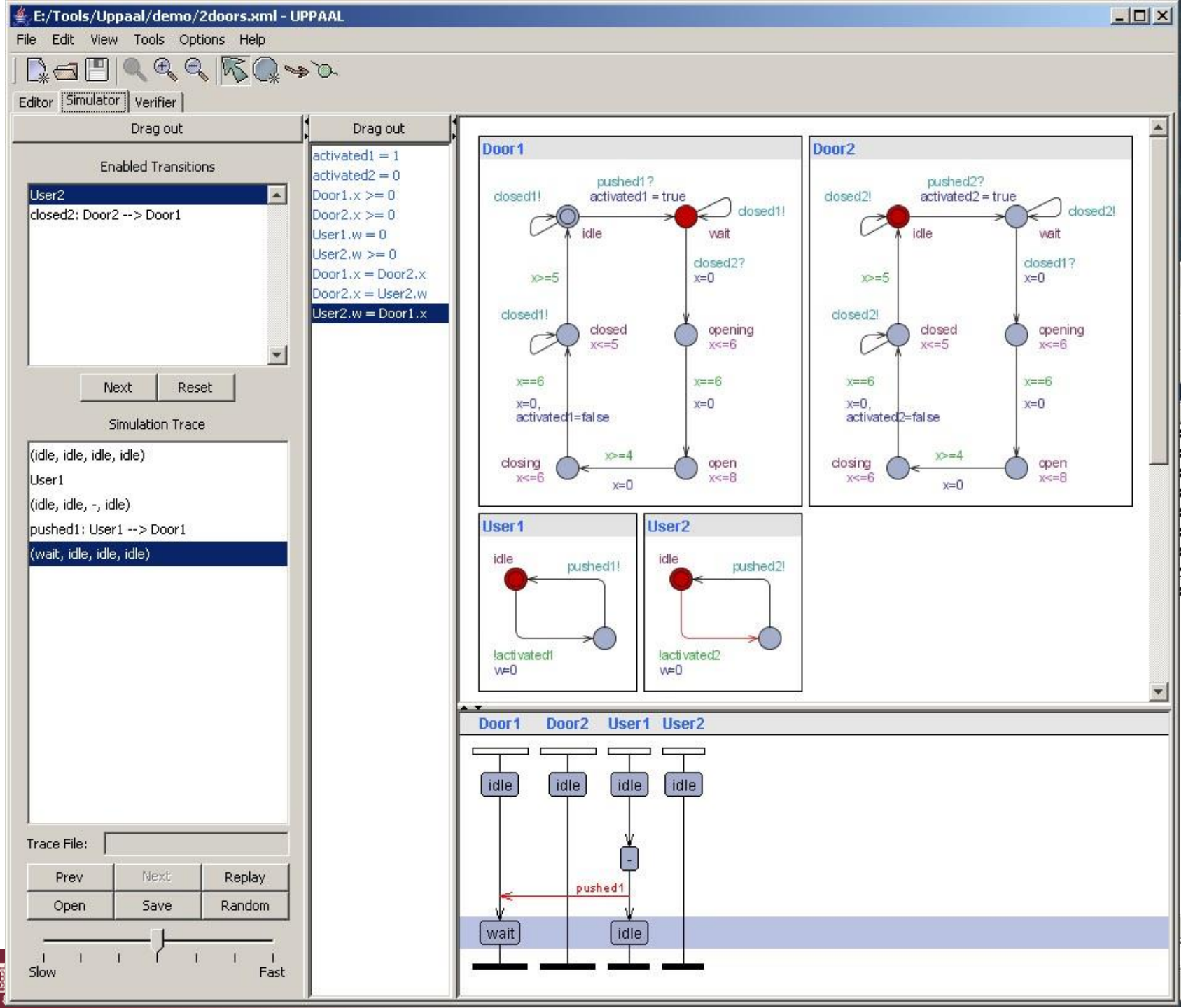
Az UPPAAL eszköz

- Fejlesztése (1999-):
  - Uppsala University, Svédország
  - Aalborg University, Dánia
- Web lap (információk, letöltés, példák):
  http://www.uppaal.org/
- Kapcsolódó eszközök:
  - UPPAAL CoVer: Tesztgenerálás
  - UPPAAL TRON: On-line tesztelés
  - UPPAAL PORT: Komponens alapú rendszerek tervezése
  - …
- Kereskedelmi verzió:
  http://www.uppaal.com/

Automata modell

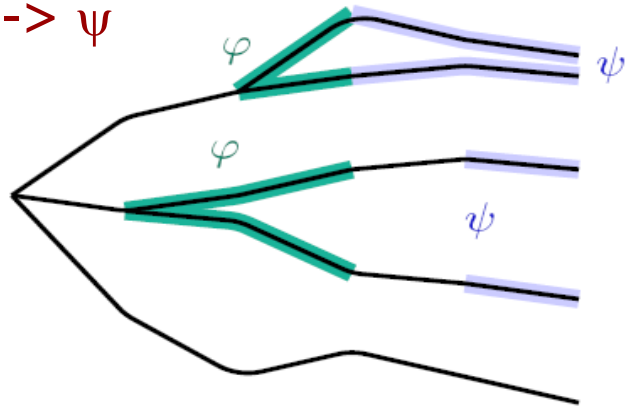


Szimulátor

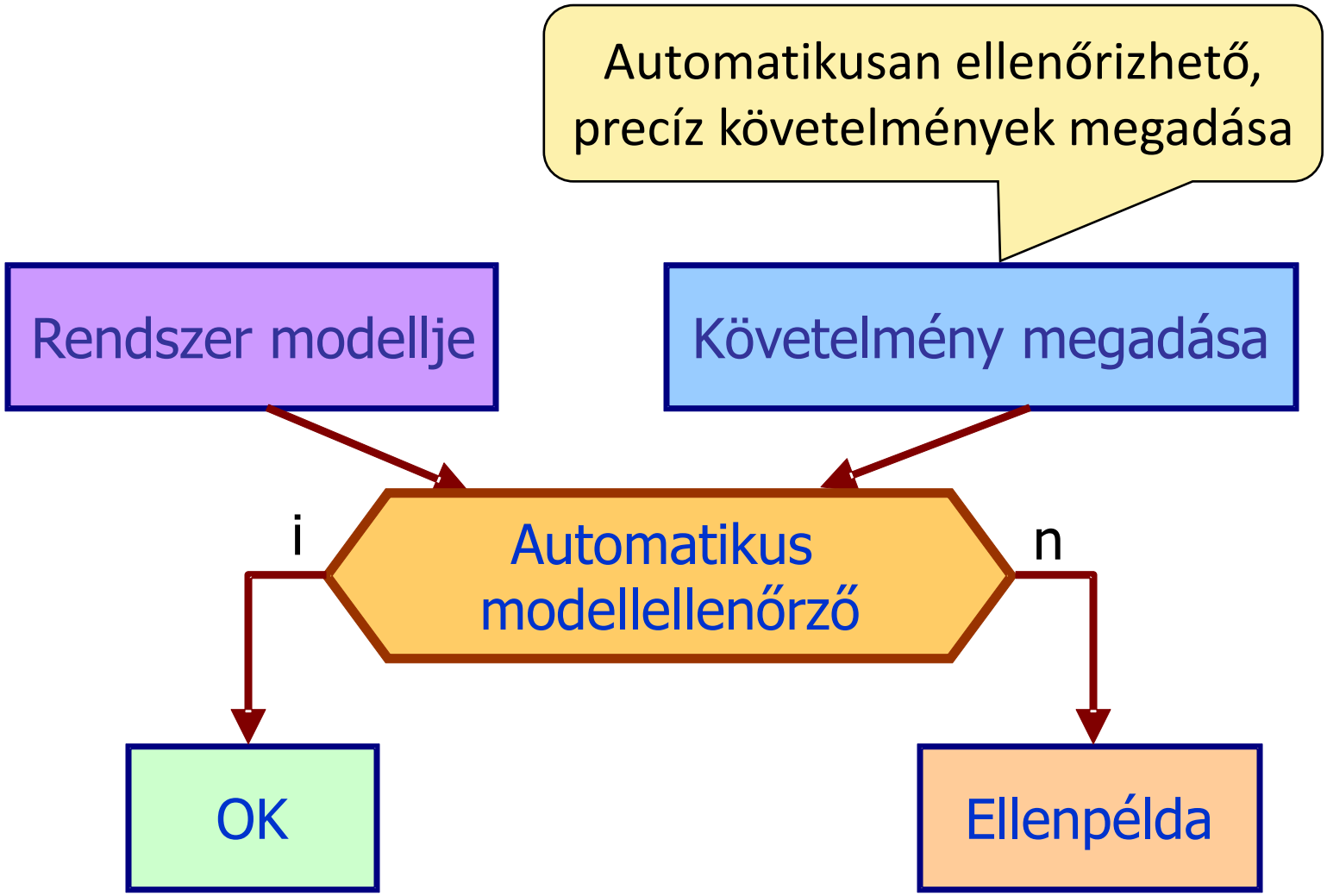

E:/Tools/Uppaal/demo/2doors.xml - UPPAAL
File Edit View Tools Options Help
Editor Simulator Verifier
Drag out
Enabled Transitions
User2
closed2: Door2 --> Door1
Next Reset
Simulation Trace
(idle, idle, idle, idle)
User1
(idle, idle, -, idle)
pushed1: User1 --> Door1
(wait, idle, idle, idle)
Trace File:
Prev Next Replay
Open Save Random
Slow Fast
Drag out
activated1 = 1
activated2 = 0
Door1.x >= 0
Door2.x >= 0
User1.w = 0
User2.w >= 0
Door1.x = Door2.x
Door2.x = User2.w
User2.w = Door1.x
Door1
pushed1?
activated1 = true
closed1!
idle
wait
closed2?
x=0
x>=5
closed1!
closed
x<=5
opening
x<=6
x==6
x=0,
activated1=false
x==6
x=0
closing
x<=6
x>=4
x=0
open
x<=8
Door2
pushed2?
activated2 = true
closed2!
idle
wait
closed1?
x=0
x>=5
closed2!
closed
x<=5
opening
x<=6
x==6
x=0,
activated2=false
x==6
x=0
closing
x<=6
x>=4
x=0
open
x<=8
User1
idle
pushed1!
!activated1
w=0
User2
idle
pushed2!
!activated2
w=0
Door1 Door2 User1 User2
idle
idle
idle
idle
-
pushed1
wait
idle

Követelmények formalizálása: Temporális logikák

φ --> ψ



Mit szeretnénk elérni?



Ellenőrizendő követelmények (példa)

Egy példa az ellenőrizendő követelmények jellegére:

- Egy klímaberendezés állapotai:
  - Kikapcsolva, bekapcsolva, elromlott, gyengén hűt, erősen hűt, fűt, szellőztet
- A klíma néhány működési követelménye:
  - Bekapcsolás után szellőztetést kezd
  - Erős hűtés csak gyenge hűtés után következhet
  - Amíg a fűtés be nem fejeződik, addig szellőztetni kell
  - Ha a klíma elromlott, nem fűthet
  - ...

Állapotokra vonatkozó követelmények

- Lokális: Egy-egy állapotra vonatkozó
  - Az adott állapotban eldönthető a teljesítése (pl. az állapotváltozók értékei alapján)
  - Példa: „A kezdeti állapotban legyen szellőztetés”
- Elérhetőségi: Több állapot bekövetkezési sorrendjére vonatkozó követelmények
  - Az állapottér vizsgálatával dönthető el a teljesítése
  - Példa: „A fűtési fázis befejezése után szellőztetni kell”
  - Folyamatosan működő rendszerekre is alkalmas
  - A tipikus követelmények jellege:
    - A rendszer „biztonsága”
    - A rendszer „élő” jellege

A követelmények típusai

„Biztonsági” jellegű követelmények

- Veszélyes, nemkívánatos helyzetek elkerülését írják elő:
  - „Minden állapotban kisebb a nyomás a kritikusnál.”
  - „A présgép csak becsukott ajtó mellett üzemel.”
  - „A processzek között mindig teljesül a kölcsönös kizárás.”
- Formálisan: Invariáns tulajdonság
  - „Minden elérhető állapotban igaz, hogy …”

„Élő” jellegű követelmények

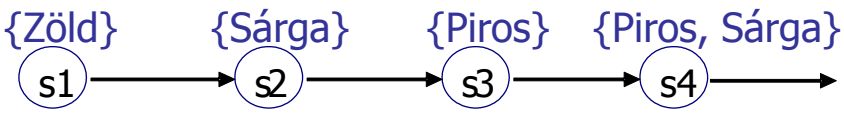
- Kívánatos helyzetek elérését írják elő
  - „Az indítás után a présgép kiadja az elkészült terméket.”
  - „A zavarás után a folyamat visszakerül stabil állapotba.”
  - „Az elküldött üzenet feldolgozása megtörténik a fogadónál.”
- Formálisan: Állapotok elérését fogalmazzák meg:
  - „Elérhető olyan állapot, hogy …”

Követelmények leíró nyelve

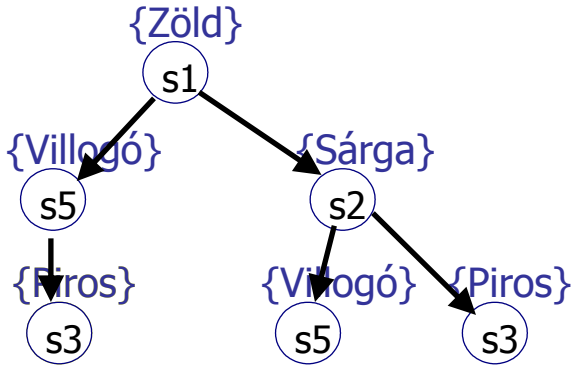
- Elérhetőség: Több állapot bekövetkezési sorrendjére vonatkozó követelmények
  - Állapotok egymásutánisága mint logikai idő vehető figyelembe
    - Jelen időpillanat: Aktuális állapot
    - Következő időpillanat: Rákövetkező állapot(ok) s.í.t.
  - Temporális (logikai időbeli, sorrendi) operátorok használhatók a követelmények kifejezésére
    - Temporális operátorok: „mindig”, „valamikor”, „mielőtt”, „addig, amíg”, „azelőtt, hogy”, …
- Temporális logikák:
  - Formális nyelv arra, hogy kijelentések igazságának logikai időbeli változását vizsgálhassuk

Temporális logikák

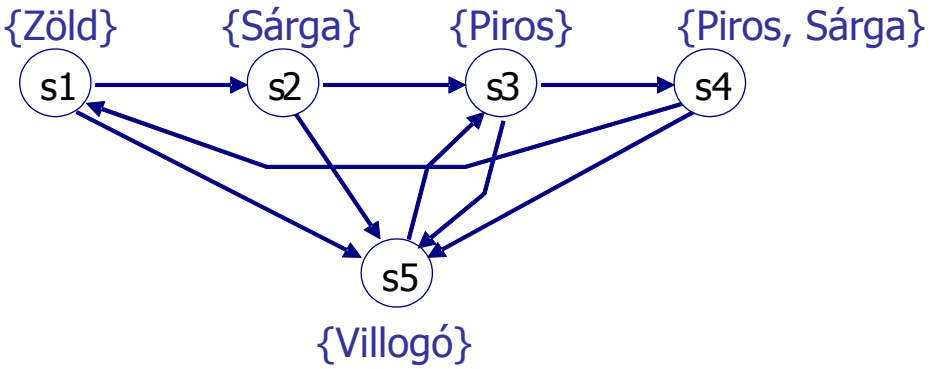
- Lineáris:
  Egymás utáni állapotok egy sorozatot alkotnak: minden állapotnak egy rákövetkezője van
  → logikai idő mint idővonal



- Elágazó:
  Egymás utáni lehetséges állapotok egy fa struktúrában: minden állapotnak több rákövetkezője lehet
  → logikai idő elágazó idővonalak mentén



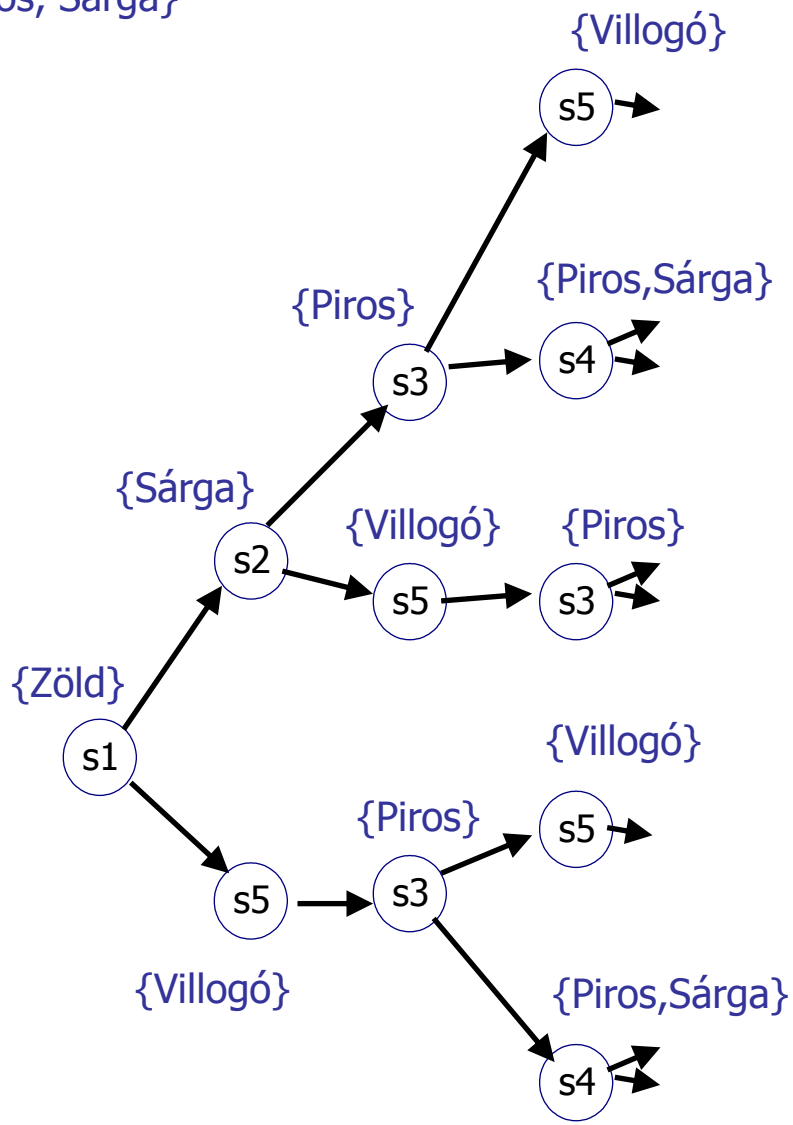
A rendszer működése mint számítási fa



↑

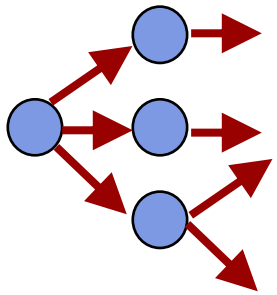
Automata címkézett állapotokkal

Számítási fa: Lehetséges elágazások

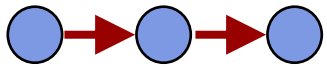


Az utak és állapotok „leszámolása”

- Adott állapotból (kezdőállapotból) induló utakat megadó operátorok
  - A: minden útvonalra („for All”)
  - E: legalább egy útvonalra („Exists”)



- Adott úton az egymás után bejárt állapotokat megadó operátorok
  - G: minden állapotra („Globally)
  - F: egy jövőbeli állapotra („Future”)



- Együtt kell használni ezeket
  - Utakat mindenképpen meg kell adni az állapotokhoz
  - AG, AF, EG, EF összetett operátorok

Temporális operátorok (UPPAAL jelölés)

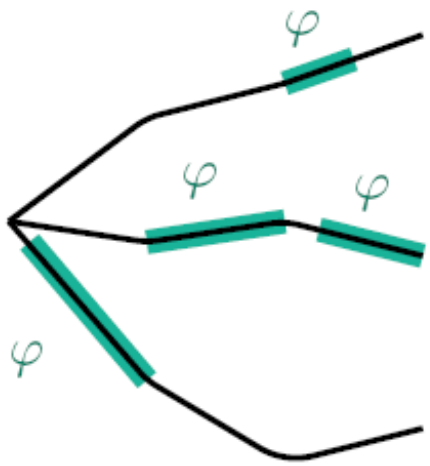
Operátor	Informális jelentés	UPPAAL jelölés
AG φ	Minden útvonalon minden állapotra φ	A[] φ
AF φ	Minden útvonalon a jövőben (előbb-utóbb) φ	A<> φ
EG φ	Egy létező útvonalon minden állapotra φ	E[] φ
EF φ	Egy létező útvonalon a jövőben (előbb-utóbb) φ	E<> φ
AG(φ => AF ψ)	φ-t követően mindig ψ	φ --> ψ
	Sehol sincs holtpont	A[] not deadlock

φ és ψ Boole-logikai kifejezések órákon, változókon és állapotokon

Minden útvonalra vonatkozó operátorok

AG φ

AF φ



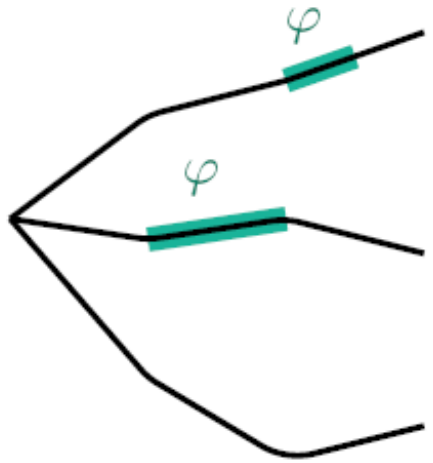
AG φ: Minden útvonalon minden állapotra igaz φ

AF φ: Minden útvonalon a jövőben előbb-utóbb elérünk olyan állapotba, ahol igaz φ

Egy-egy útvonalra vonatkozó operátorok

EG φ

EF φ

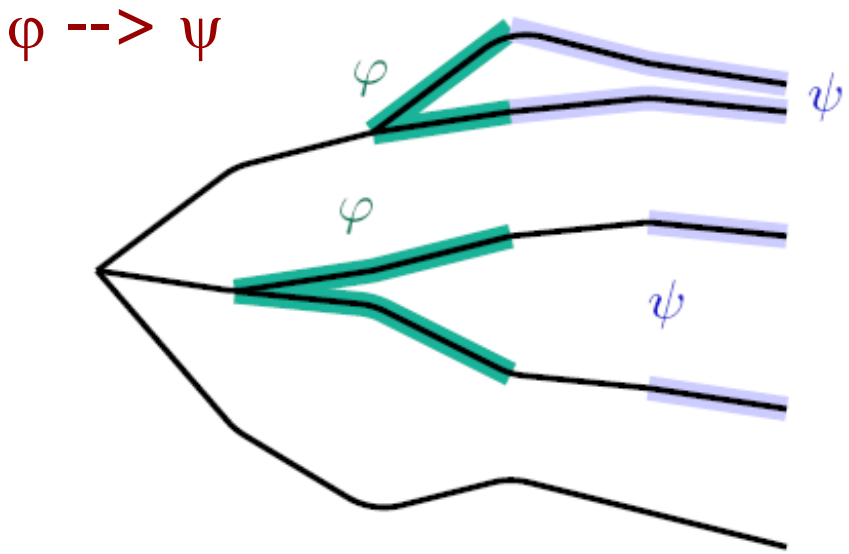


EG φ: Létezik egy útvonal, ahol minden állapotra igaz φ

EF φ: Létezik egy útvonal, ahol a jövőben előbb-utóbb elérünk olyan állapotba, ahol igaz φ

- Van-e kapcsolat AG és EF között?
- Van-e kapcsolat AF és EG között?

Feltételes elérhetőség



- AG(φ => AF ψ) ≡ φ --> ψ
  Minden útvonalra és ezeken minden állapotra igaz, hogy φ bekövetkezése esetén előbb-utóbb mindig elérünk olyan állapotba, ahol ψ teljesül
- Időzítéssel együtt: φ --> (ψ and x <= t) ahol x egy óraváltozó, amit akkor reseteltünk, amikor φ igaz lett

Néhány példa követelmények formalizálására

Adott egy klímaberendezés, aminek a következő üzemmódokat kell biztosítania:

{Kikapcsolva, Bekapcsolva, Elromlott, GyengénHűt, ErősenHűt, Fűt, Szellőztet}

- Ezekre a tulajdonságokra (állapot címkékre) hivatkozhatunk a követelmények formalizálása során
- Egy állapotban több tulajdonság is fennállhat
- A követelményeket a kezdőállapotra fogalmazzuk meg
- A követelmény formalizáláshoz a modellt még nem feltétlenül ismerjük

Néhány példa követelmények formalizálására

A klímaberendezés üzemmódjai:

{Kikapcsolva, Bekapcsolva, Elromlott, GyengénHűt, ErősenHűt, Fűt, Szellőztet}

Példák formalizált követelményekre:

- A bekapcsolt klíma előbb-utóbb mindig szellőztet:

  AF (Bekapcsolva $\wedge$ Szellőztet)

- Ha a klíma elromlott, nem fűthet (nem lehet ilyen hibamód):

  AG ($\neg$(Elromlott $\wedge$ Fűt))

- Fűtés után mindig szellőztetni kell:

  AG(Fűt => AF (Szellőztet)) vagy Fűt --> Szellőztet

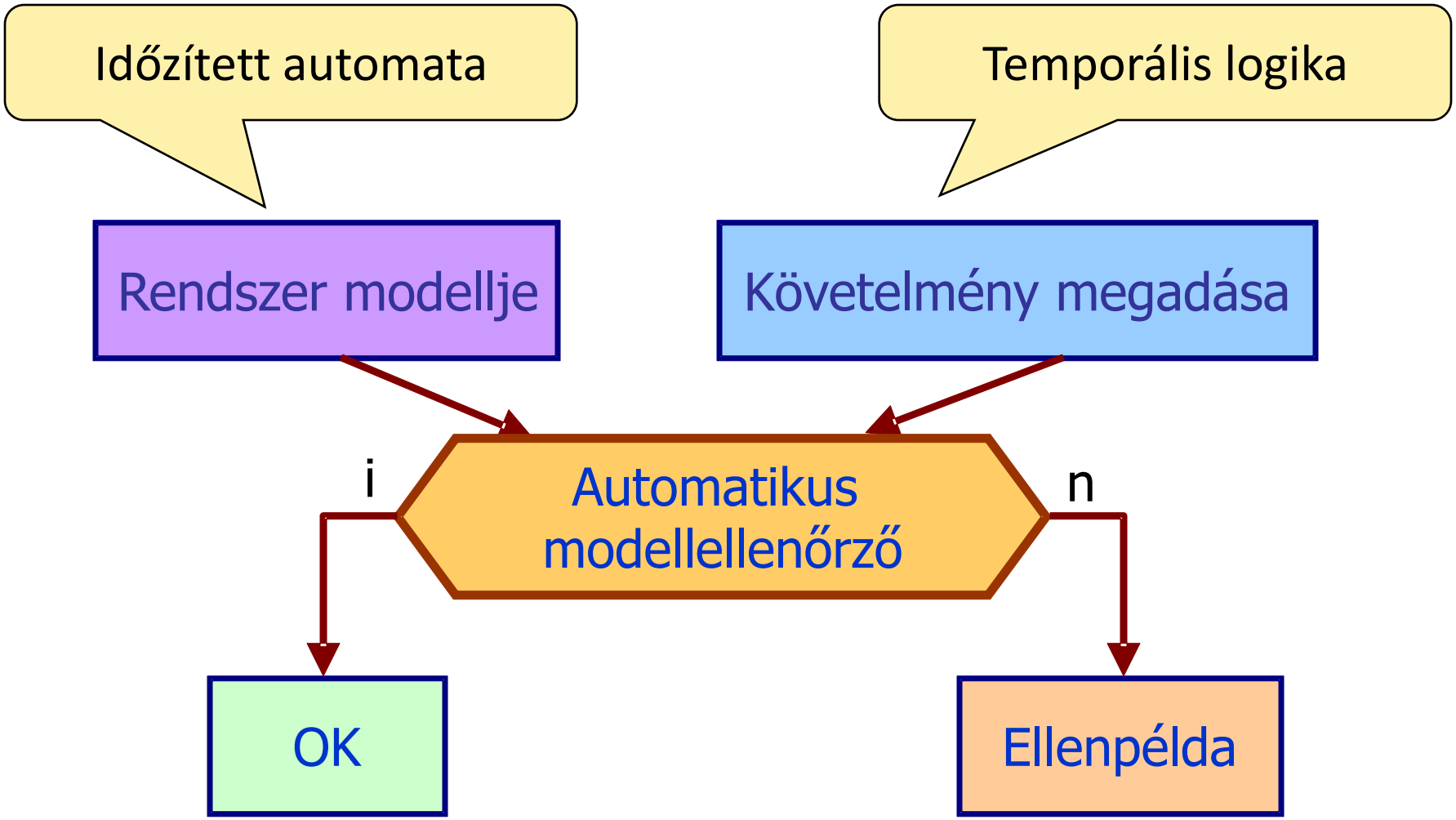
- A gyenge hűtést lehetőség van bekapcsolni:

  EF (GyengénHűt)

- Lehetséges olyan működtetés, hogy soha sincs erős hűtés:

  EG ($\neg$ErősenHűt)

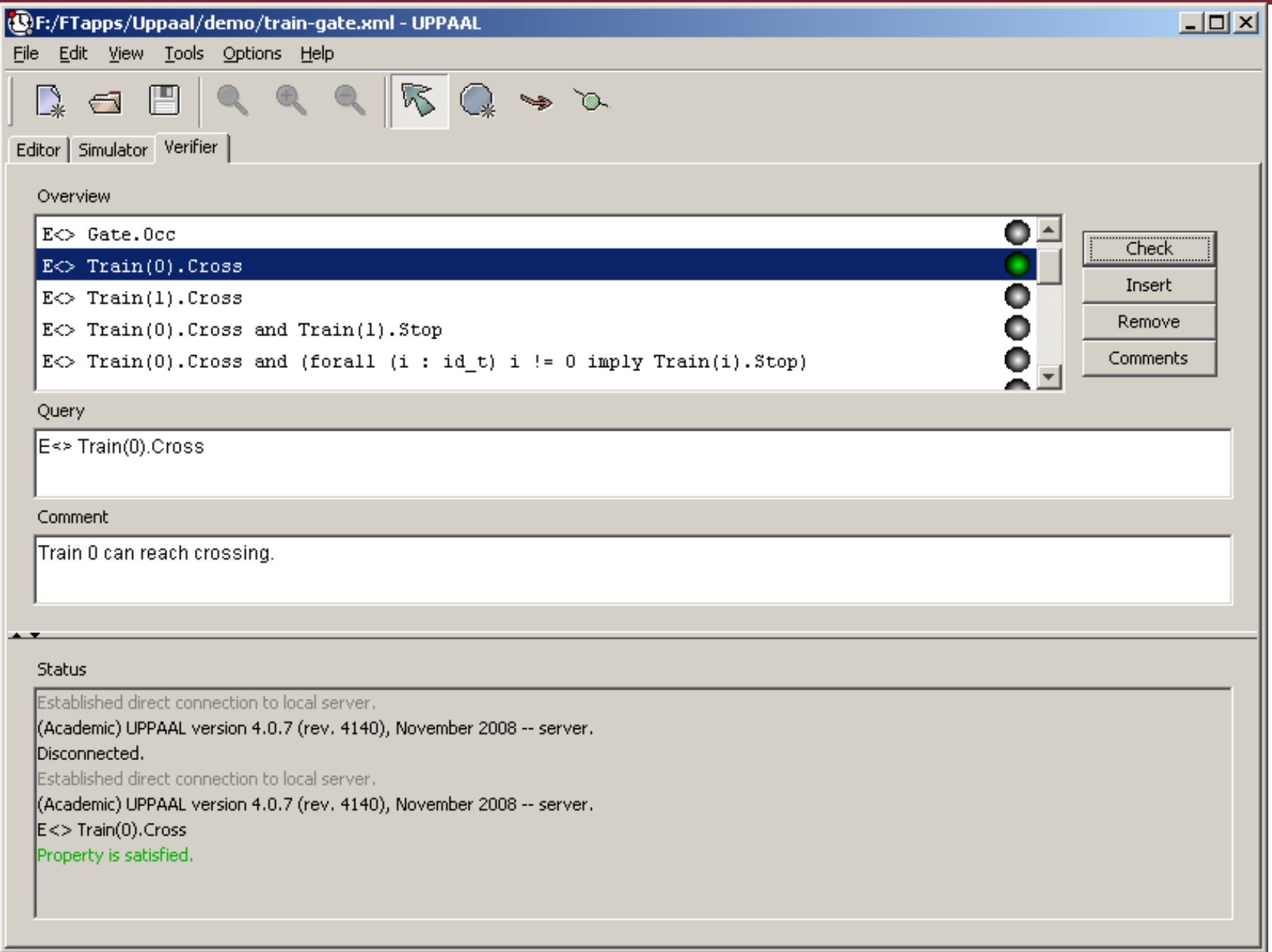
A modellellenőrzés működése



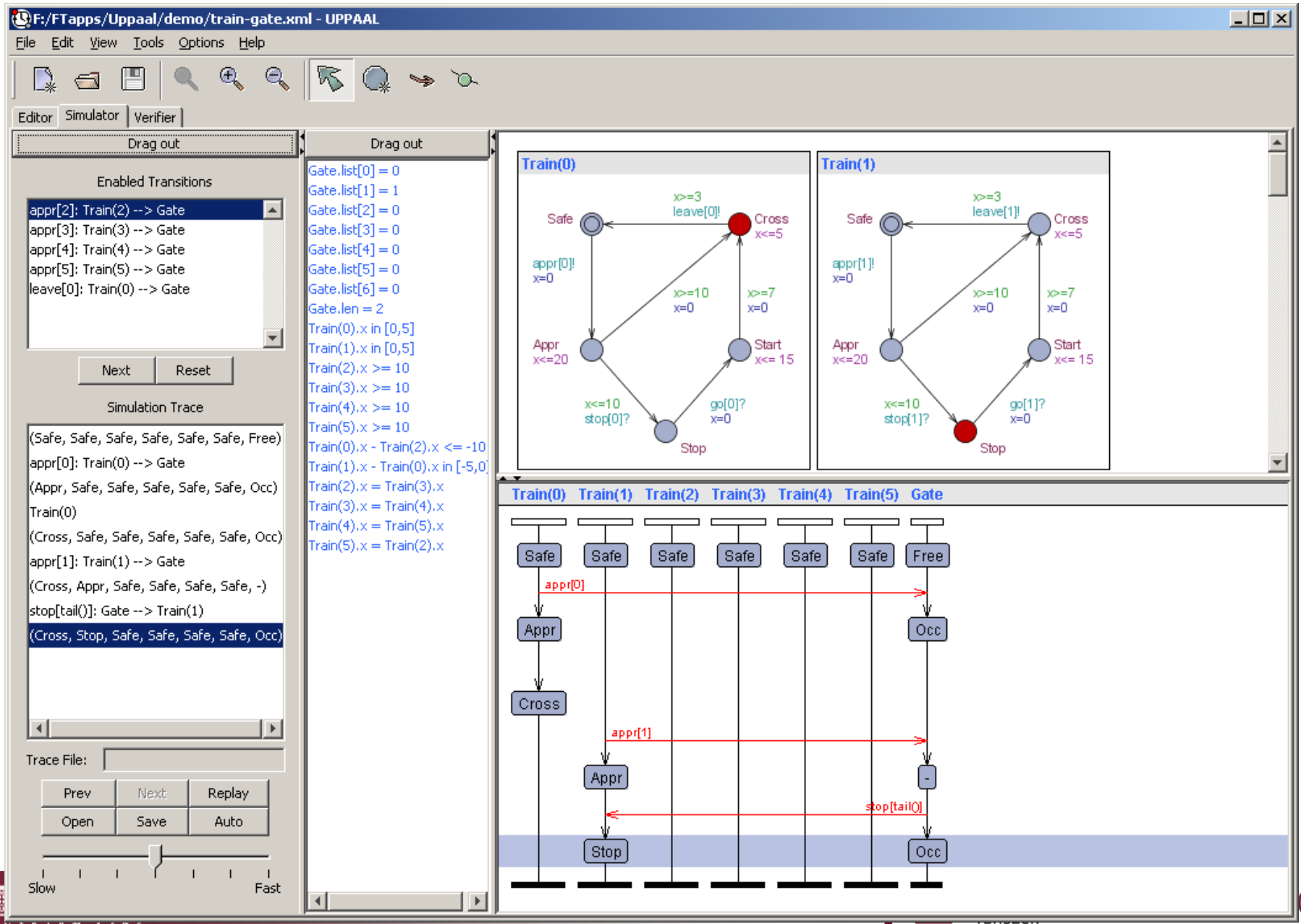
Az UPPAAL modellellenőrző

- Követelmények halmaza szerkeszthető
- Modell ellenőrzés egyenként is indítható
- Keresés az állapottérben beállítható:
  - Mélységi vagy szélességi
- Állapottárolás különféle opciókkal:
  - Redukció
  - Alul- illetve felülbecslés
- Ellenpélda generálható invariánsokhoz:
  - Legrövidebb, leggyorsabb, vagy akármilyen
  - Betölthető a szimulátorba (végigjátszható)

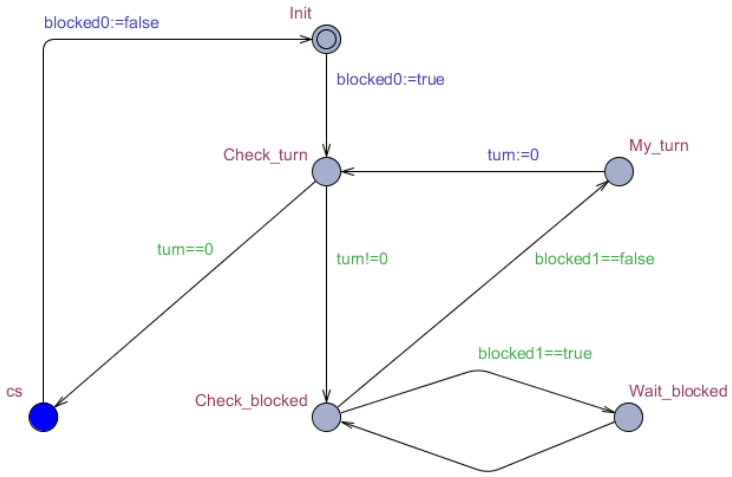
Az UPPAAL modellellenőrző ablaka



Ellenpélda a szimulátorban



Mintapélda


blocked0:=false
Init
blocked0:=true
Check_turn
turn:=0
My_turn
turn==0
turn!=0
blocked1==false
blocked1==true
cs
Check_blocked
Wait_blocked

Mintapélda: Kölcsönös kizárás

- 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
    blocked0 = false;
    // Do other things
}
```

P1

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

Helyes-e ez az algoritmus?

A modell UPPAAL-ban (első változat)

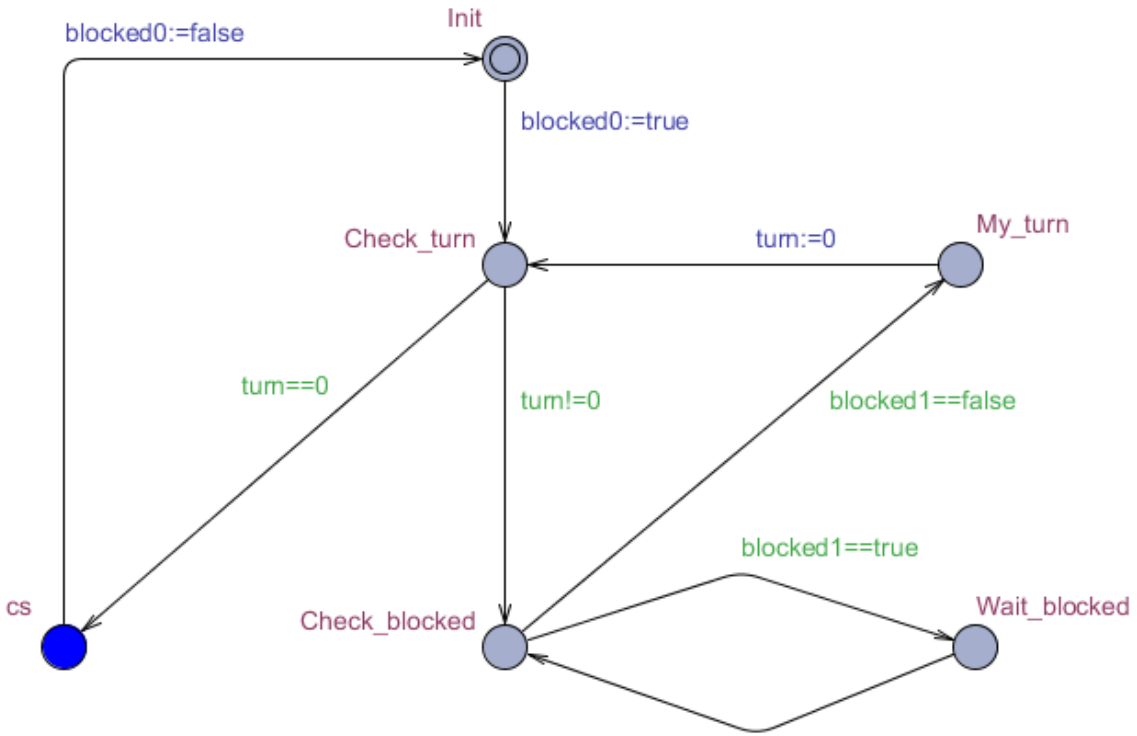
Deklarációk:

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

Kihasznált modellezési lehetőségek:

- Közös változók rendszerszintű deklarálása
- Korlátozott értékkészletű változók

A P0 automata:



P0

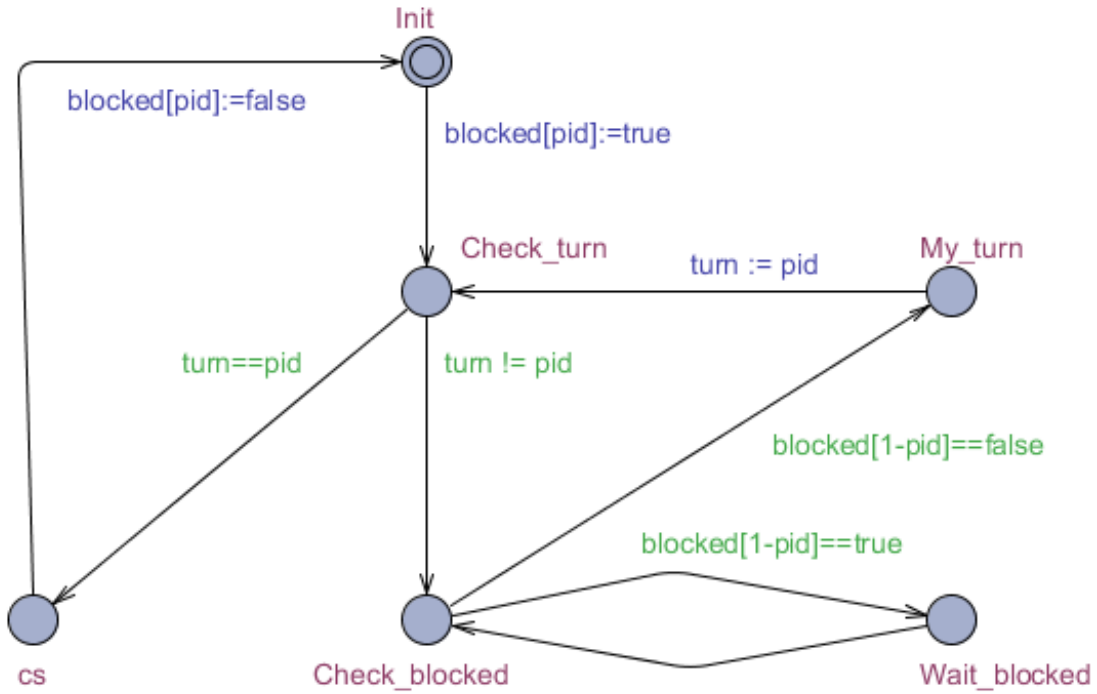
```
while (true) {
    blocked0 = true;
    while (turn!=0) {
        while (blocked1==true) {
            skip;
        }
        turn=0;
    }
    // Critical section
    blocked0 = false;
    // Do other things
}
```

A modell UPPAAL-ban (második változat)

Deklarációk:

```
bool blocked[2];
int[0,1] turn;
P0 = P(0);
P1 = P(1);
system P0,P1;
```

A P automata (template) pid paraméterrel:



Kihasznált modellezési lehetőségek:

- Közös változók rendszerszintű deklarálása
- Korlátozott értékkészletű változók
- Azonos viselkedésű résztvevők azonos automata template alapján
- Példányosítás paraméterezéssel
- Változó tömbök (résztvevőkhöz)

P0

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

Ellenőrizendő követelmények

- Kölcsönös kizárás:
  - Egyszerre csak az egyik résztvevő lehet a kritikus szakaszban
- Lehetséges az elvárt viselkedés:
  - P0 egyáltalán be tud lépni a kritikus szakaszba
  - P1 egyáltalán be tud lépni 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)

UPPAAL: A követelmények formalizálása

- Kölcsönös kizárás:
  - Egyszerre csak az egyik résztvevő lehet a kritikus szakaszban: A[] not (P0.cs and P1.cs)
- Holtpontmentesség:
  - Nem alakul ki kölcsönös várakozás (leállás): A[] not deadlock
- Lehetséges az elvárt viselkedés:
  - P0 egyáltalán beléphet a kritikus szakaszba: E<>(P0.cs)
  - P1 egyáltalán beléphet a kritikus szakaszba: E<>(P1.cs)
- Nincs kiéheztetés:
  - P0 mindenképpen be fog lépni a kritikus szakaszba: A<>(P0.cs)
  - P1 mindenképpen be fog lépni a kritikus szakaszba: A<>(P1.cs)

UPPAAL: A modellellenőrzés eredménye

- Nincs holtpont
- Az élő jellegű követelmények teljesülnek
  - Mindkét processz be tud lépni a kritikus szakaszba (lehetőség)
- A kölcsönös kizárás nem teljesül!
  - Ellenpélda: Egy átlapolt végrehajtás, a szimulátorban követhető
- Kiéheztetés elkerülése (szükségszerű belépés) nem garantálható
  - Triviális ellenpélda: végtelen sok ideig vár egy állapotban
  - Ennek elkerülése: pl. urgent állapot (ha ez érvényes viselkedés), vagy időfüggő feltételek és állapot invariánsok
  - Itt: ezek után is van olyan (ciklikus) viselkedés, ahol az egyik processz kiéhezteti a másikat

A kölcsönös kizárás hibájának javítása

Peterson algoritmusa

- P0 résztvevőre (P1 értelemszerű):

Hyman:

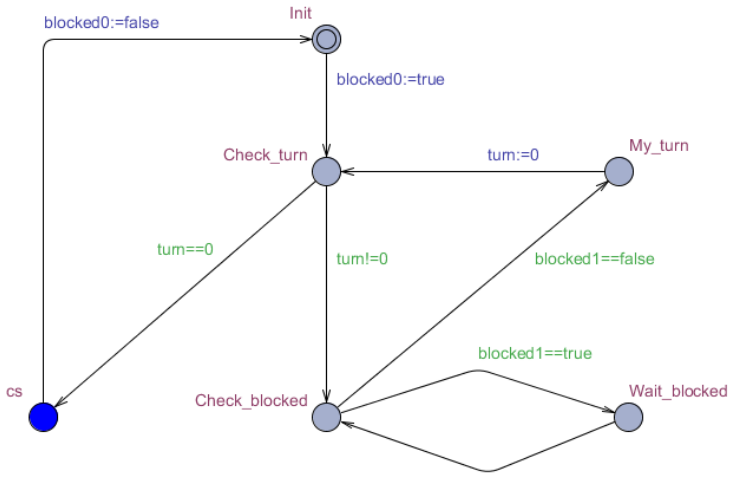
```
while (true) {
    blocked0 = true;
    while (turn!=0) {
        while (blocked1==true) {
            skip;
        }
        turn=0;
    }
    // Critical section
    blocked0 = false;
    // Do other things
}
```

Peterson:

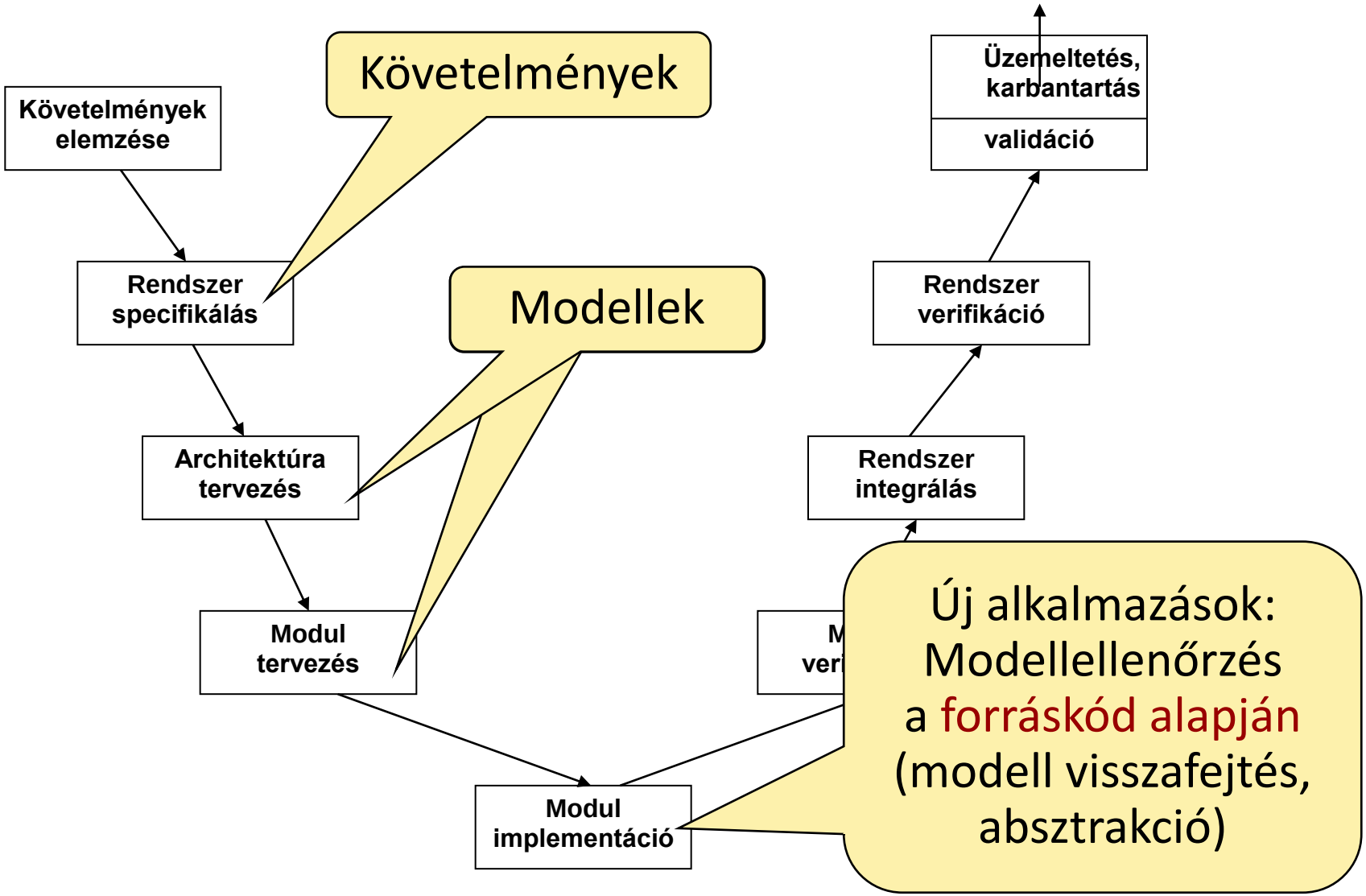
```
while (true) {
    blocked0 = true;
    turn=1;
    while (blocked1==true &&
           turn!=0) {
        skip;
    }

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

A modellellenőrzés tulajdonságai



A modellellenőrzés használata



A modellellenőrzés tulajdonságai

- Kedvező tulajdonságok:
  - Garantált eredményt ad: Teljes állapottér bejárás
  - Lehetséges nagy modell állapotteret is kezelni (kompakt tárolással)
    - Akár 10^{20}, de példa van 10^{100} méretű állapottér ellenőrzésére is
  - Teljesen automatikus eszköz, nem szükséges tervezői intuíció és erős matematikai háttérismeret
  - Ellenpéldát generál, ami segít a hibák javításában
- Problémák:
  - Skálázhatóság nem biztosított (explicit állapottér bejárás)
  - Elsősorban vezérlés-orientált alkalmazásokra hatékony
    - Komplex adatstruktúrák hatalmas állapotteret jelentenek
  - Nehéz az eredményeket általánosítani
    - Ha egy protokoll helyes 2 résztvevő esetén, akkor helyes-e n esetén?
  - A követelmények formalizálása komplikált
    - „Nyelvjárások” alakultak ki különböző alkalmazási területeken

Továbblépés

Az ellenőrzött modellek felhasználása

- Forráskód generálás
- Monitor szintézis futásidőbeli verifikációhoz
- Dokumentáció készítés