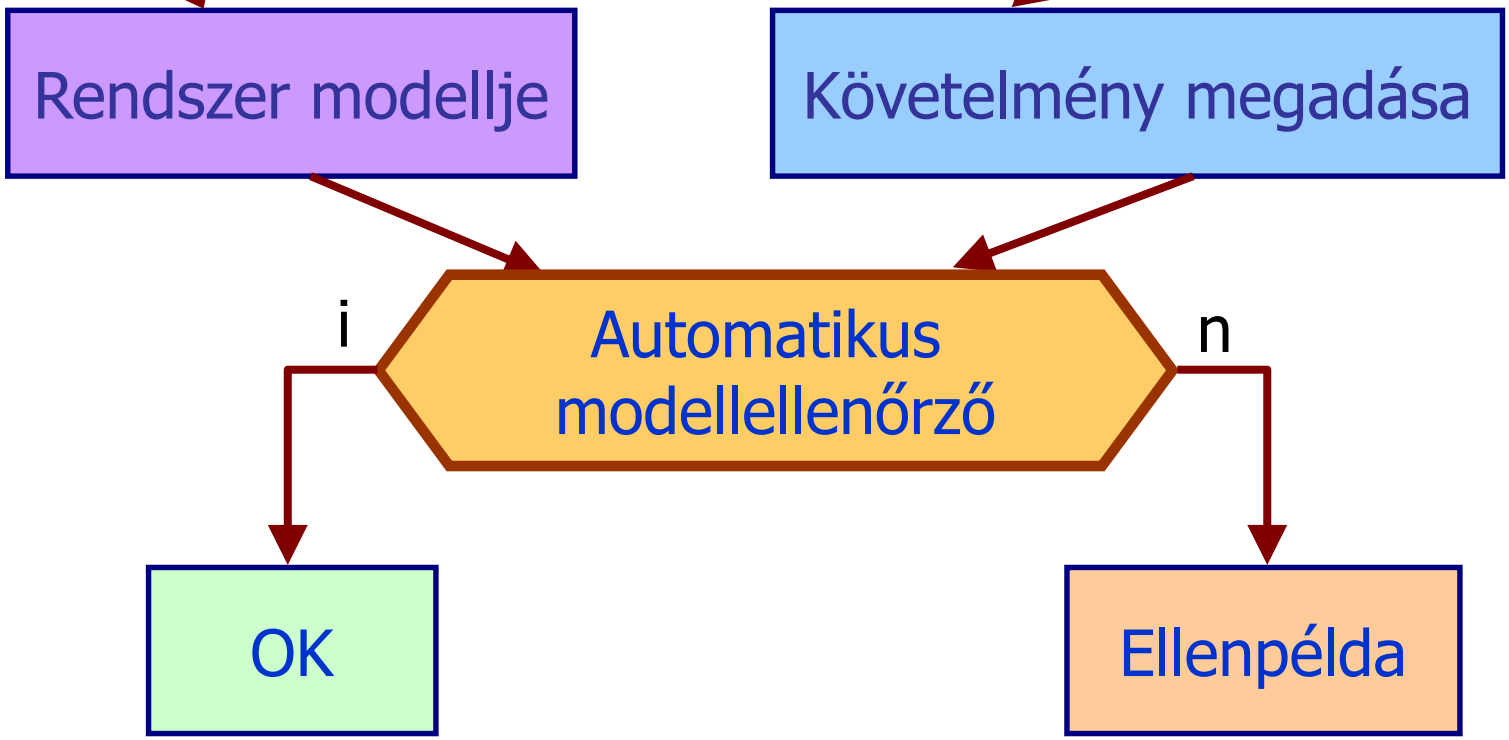
A modellellenőrzési feladat (bevezető)

dr. Majzik István
BME Méréstechnika és Információs Rendszerek Tanszék

Ismétlés: Mit szeretnénk elérni?

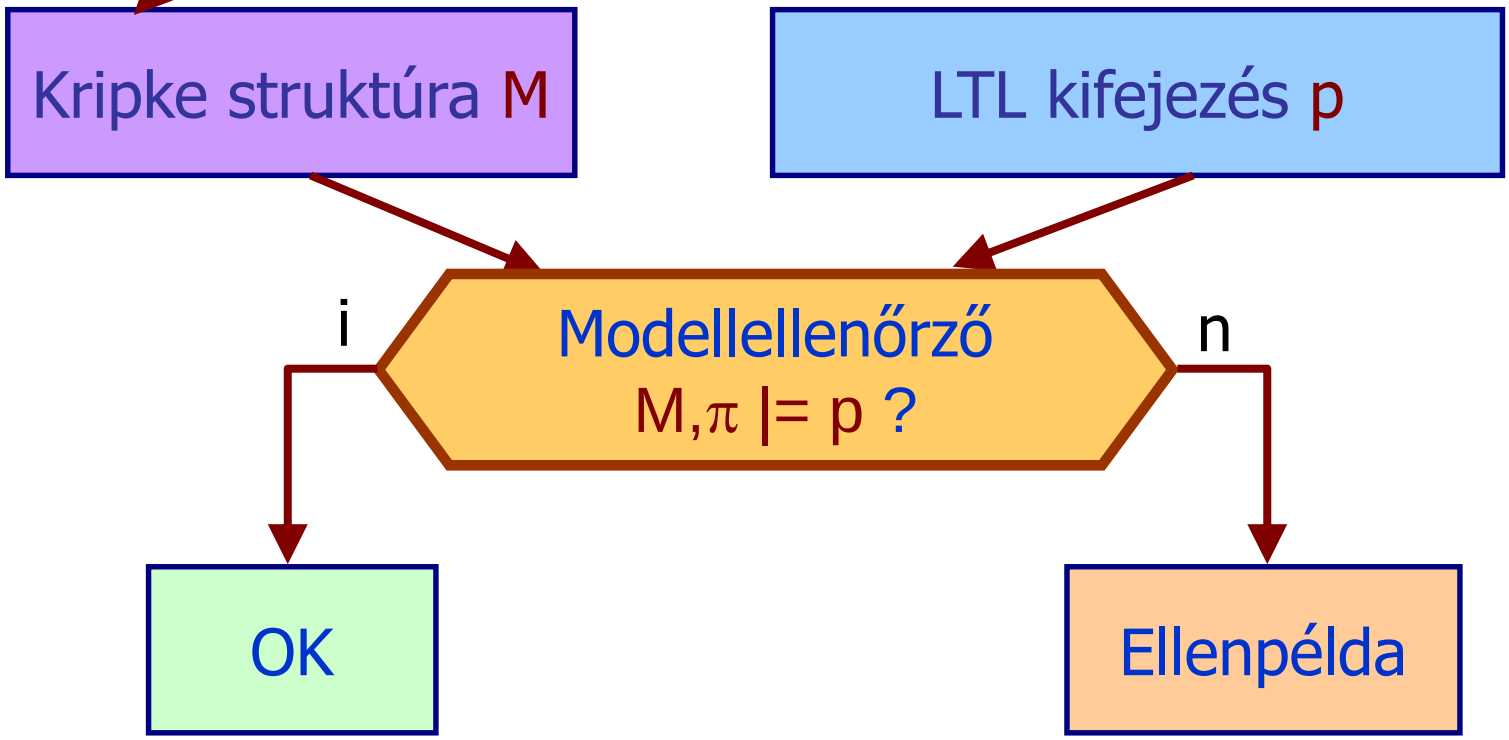
Alacsony szintű modellek:
- KS, LTS, KTS
- Időzített automata

Állapot elérhetőségi követelmények:
- Temporális logikák: lineáris / elágazó idejű

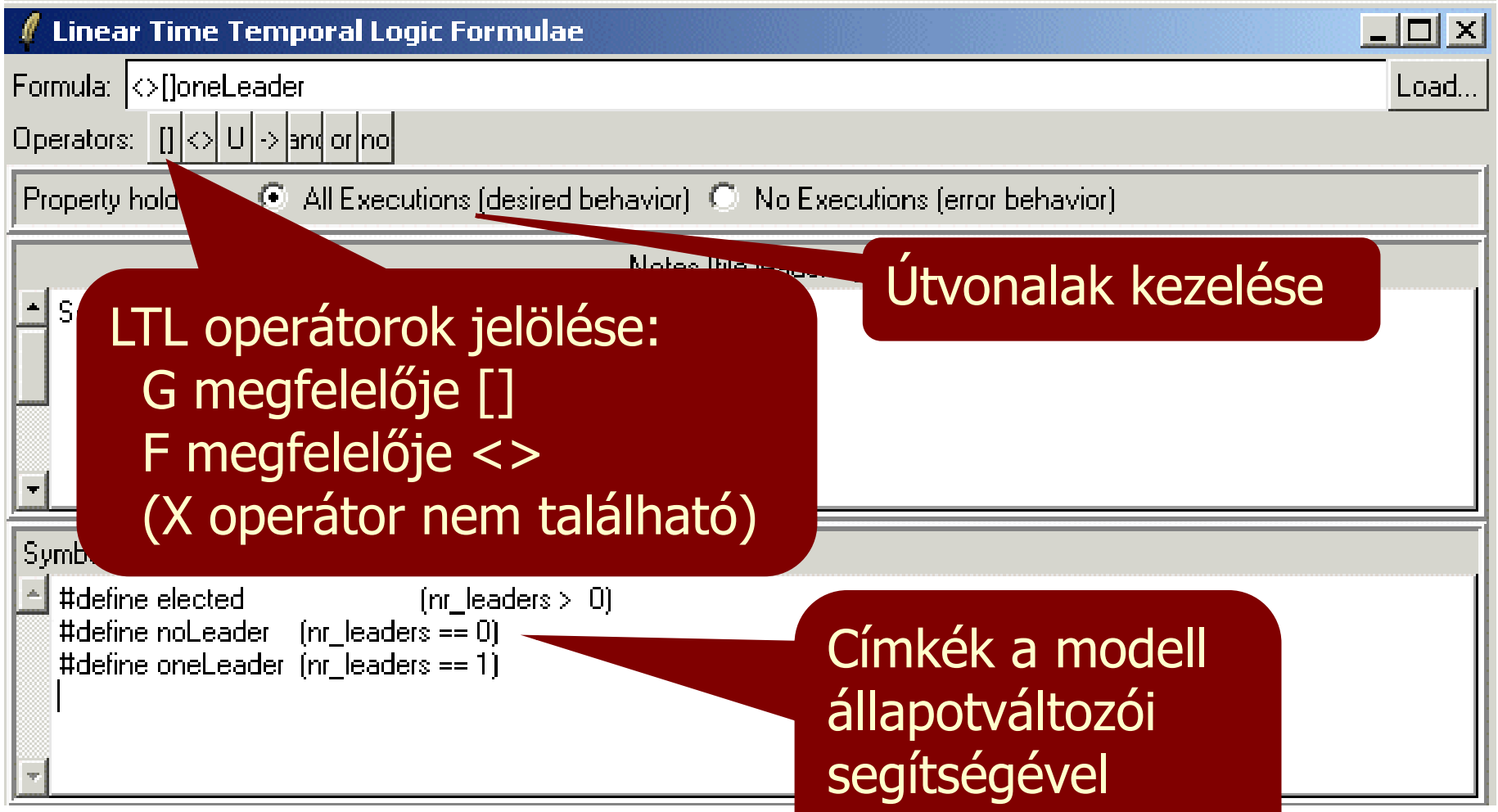


LTL modellellenőrzés

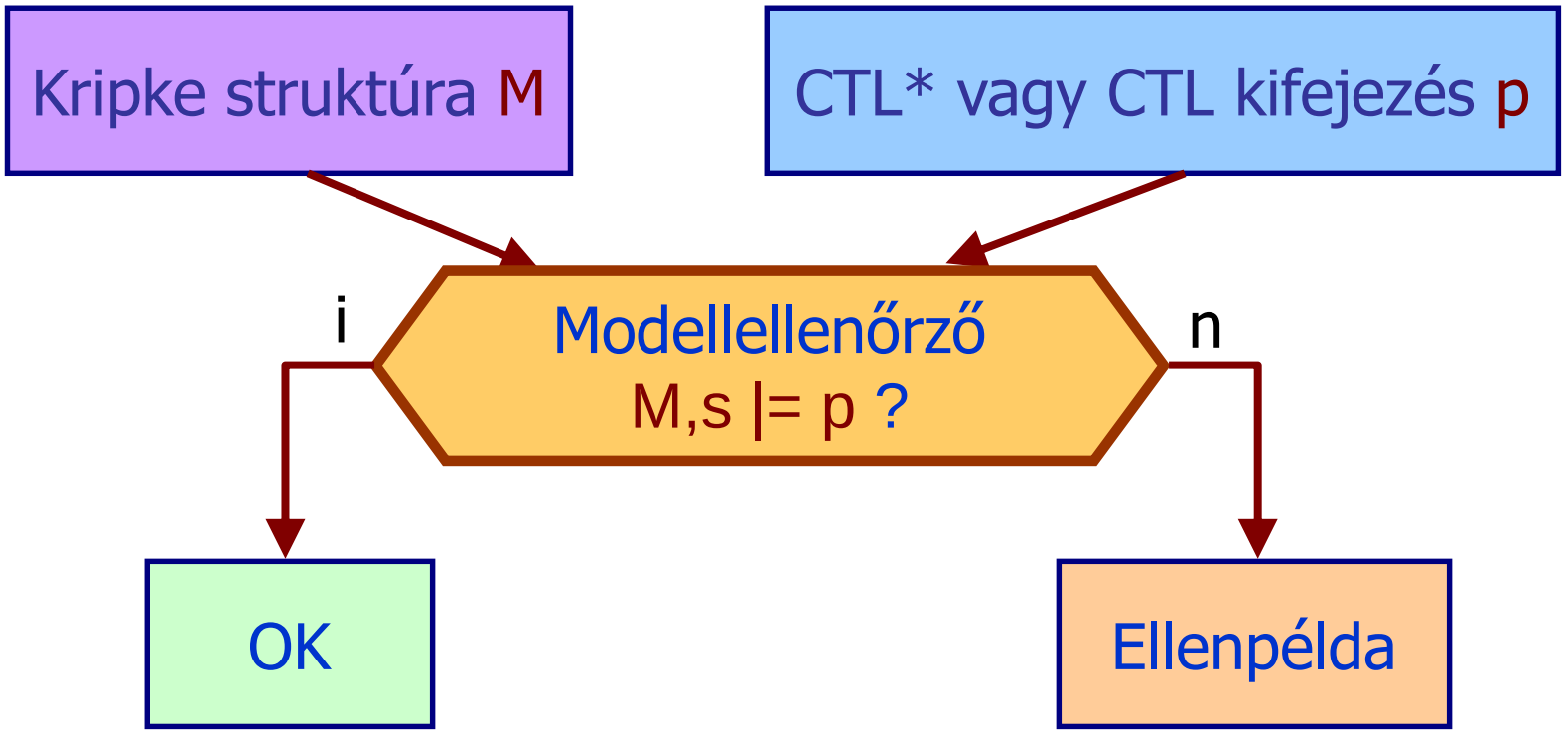
Ha nincs útvonal megadva, akkor a kezdőállapotból induló minden útra ellenőriz!



A SPIN modellellenőrző (régi felület)



CTL* vagy CTL modellellenőrzés



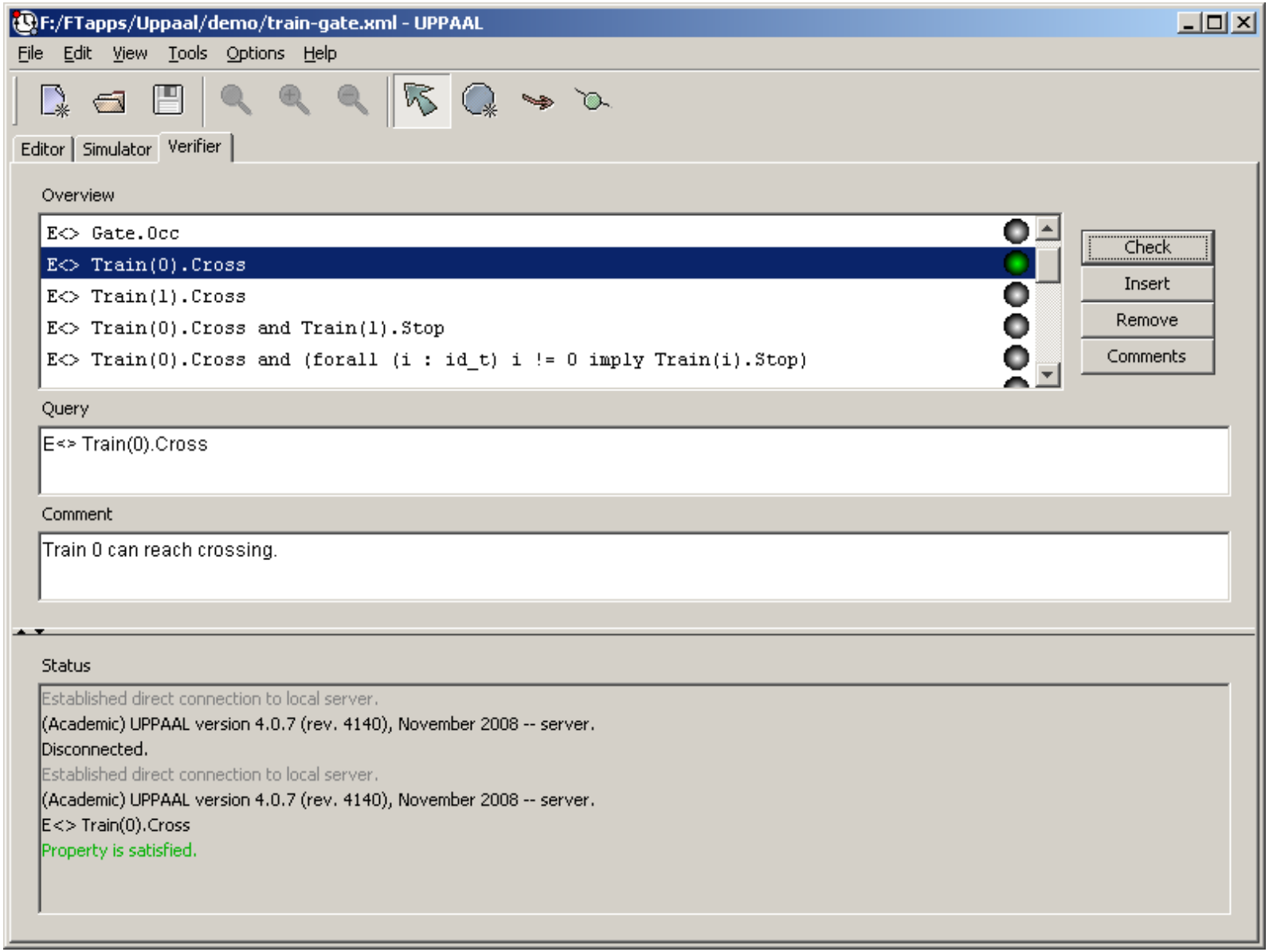
Az UPPAAL modellellenőrző lehetőségei

- Atomi kijelentések:
  - Processz vezérlési hely hivatkozható: pl. Train.cross
  - Változók értéke hivatkozható: pl. a!=1
    - Egész aritmetika és bitenkénti műveletek használhatók
  - Modell paraméterekre: forall, exists operátorok
    - Pl. forall (i : int[0,4]) Train(i).cross minden i paraméterre
  - Holtpont (nincs átmenet): deadlock kijelentéssel megadható
- Boole logikai operátorok:
  - and, or, imply, not
- Temporális operátorok: Korlátozott CTL
  - Kifejezés elején szerepelhet egy temporális operátor
    - Speciális lehetőség: p-->q, jelentése AG (p imply AF q)
  - Az U és X operátorok nem használhatók
  - Jelölés: [] szerepel G helyett, <> szerepel F helyett
    - Így lesz: A[], A<>, E[], E<>

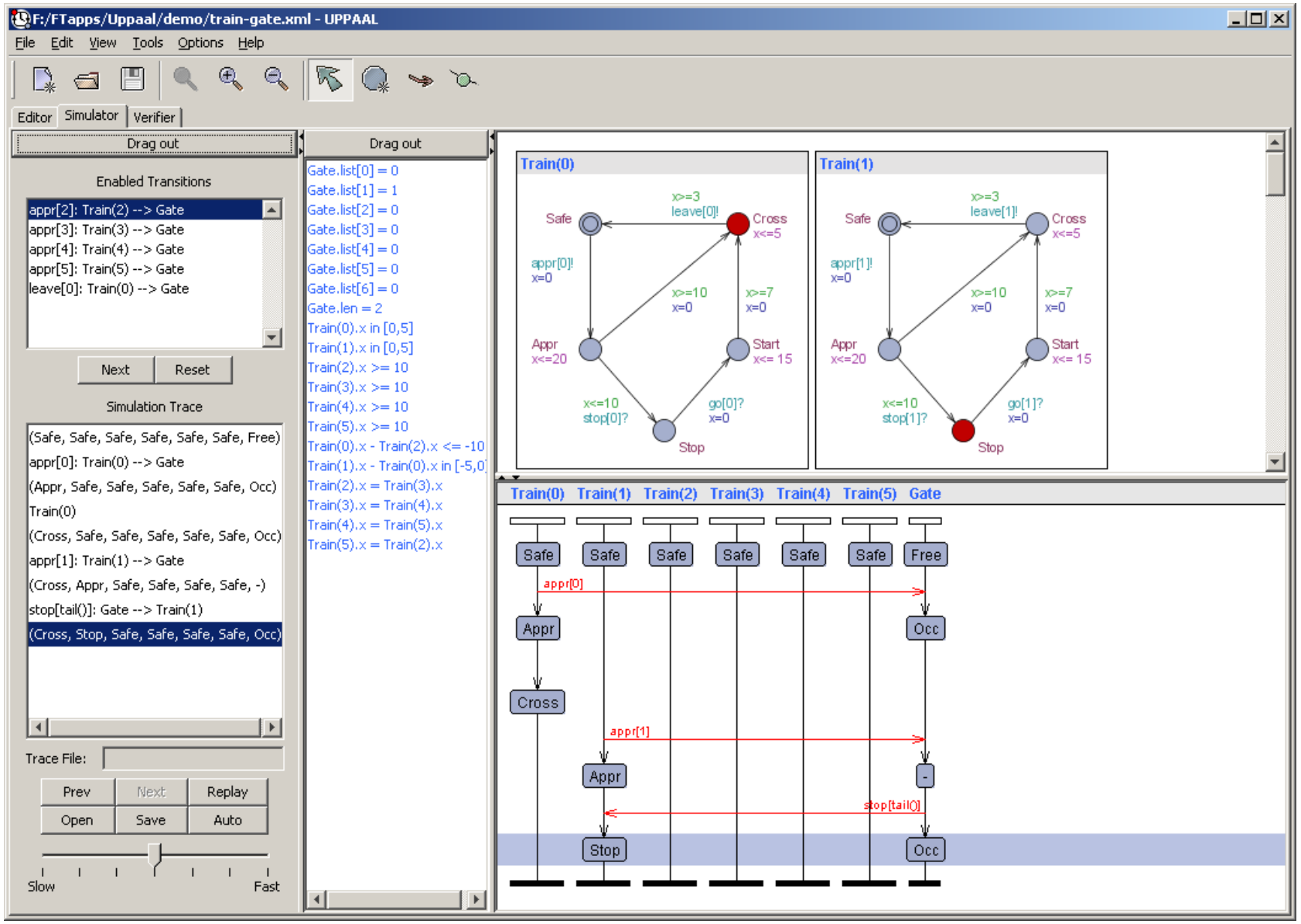
Követelmények ellenőrzése az UPPAAL-ban

- Követelmények halmaza összeállítható
- Modellellenőrzés ezekre egyenként is indítható
- Diagnosztikai trace generálható
  - Ellenpélda (nem teljesülésre) vagy ún. tanú (teljesülésre)
  - Legrövidebb, leggyorsabb, vagy akármilyen kérhető
  - Betölthető a szimulátorba (végigjátszható)
- Keresés az állapottérben
  - Mélységi vagy szélességi és lehet
- Állapottárolás különféle opciókkal
  - Redukció
  - Közelítő állapottér tárolás (alul- illetve felülbecslés)

Az UPPAAL modellellenőrző ablaka



Ellenpélda az UPPAAL szimulátorban



További részletek: UPPAAL Tutorial

Önálló feldolgozásra előírt tananyag (házi feladathoz is)!

- A Tutorial on Uppaal 4.0
  - Írta: G. Behrmann, A. David, K. G. Larsen
  - https://uppaal.org/documentation/

    https://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf

  Tartalom:
  - Timed Automata in Uppaal
  - Overview of the Uppaal Toolkit
  - Example 1, 2, 3
  - Modelling Patterns: Value passing, multicast, atomicity, …
- További forrás: UPPAAL Help (eszközben) és Online Help
  - https://docs.uppaal.org/

A motivációs mintapélda befejezése

Mintapélda: Kölcsönös kizárás

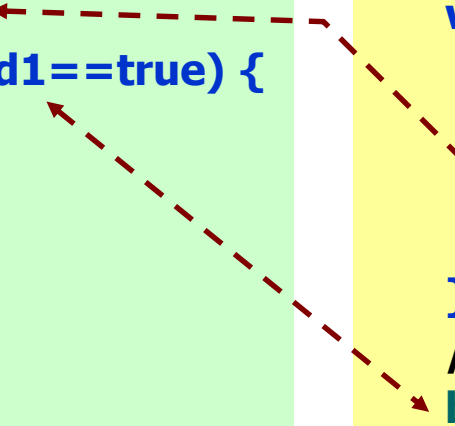
- 2 résztvevőre, 3 megosztott változóval (H. Hyman, 1966)
  - **blocked0**: Első résztvevő (P0) be akar lépni
  - **blocked1**: Második résztvevő (P1) be akar lépni
  - **turn**: Ki következik belépni (0 esetén P0, 1 esetén P1)

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: Két processz

Deklarációk:

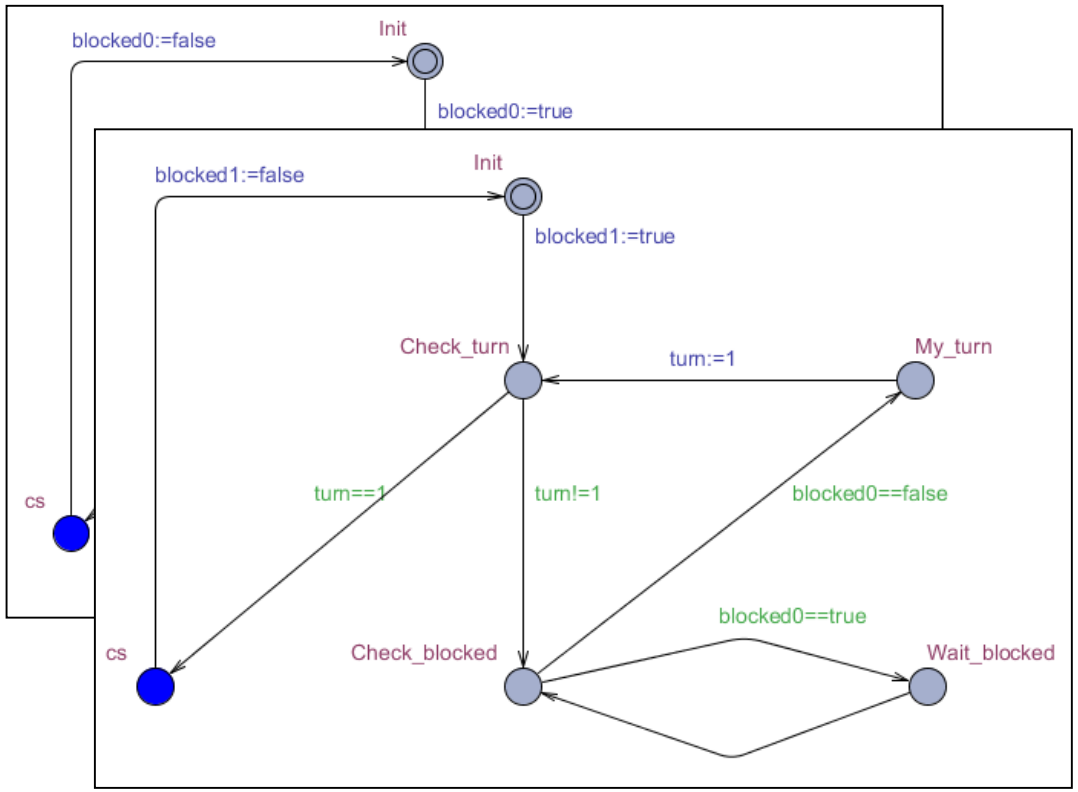
```
bool blocked0=false;
bool blocked1=false;
int[0,1] turn=0;
system P0, P1;
```

Kihasznált modellezési lehetőségek:

- Közös változók rendszerszintű deklarálása
- Korlátozott értékkészletű változók

Azonos struktúra, csak egyes változókban, adatértékekben különbözik

A P0 és P1 automata:



P0

```
while (true) {
   blocked0 := true;
   while (turn!=0) {
        while (blocked1==true) {
   }
   //
   bl
   //
}
```

P1

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

A modell UPPAAL-ban: Paraméterezett processz

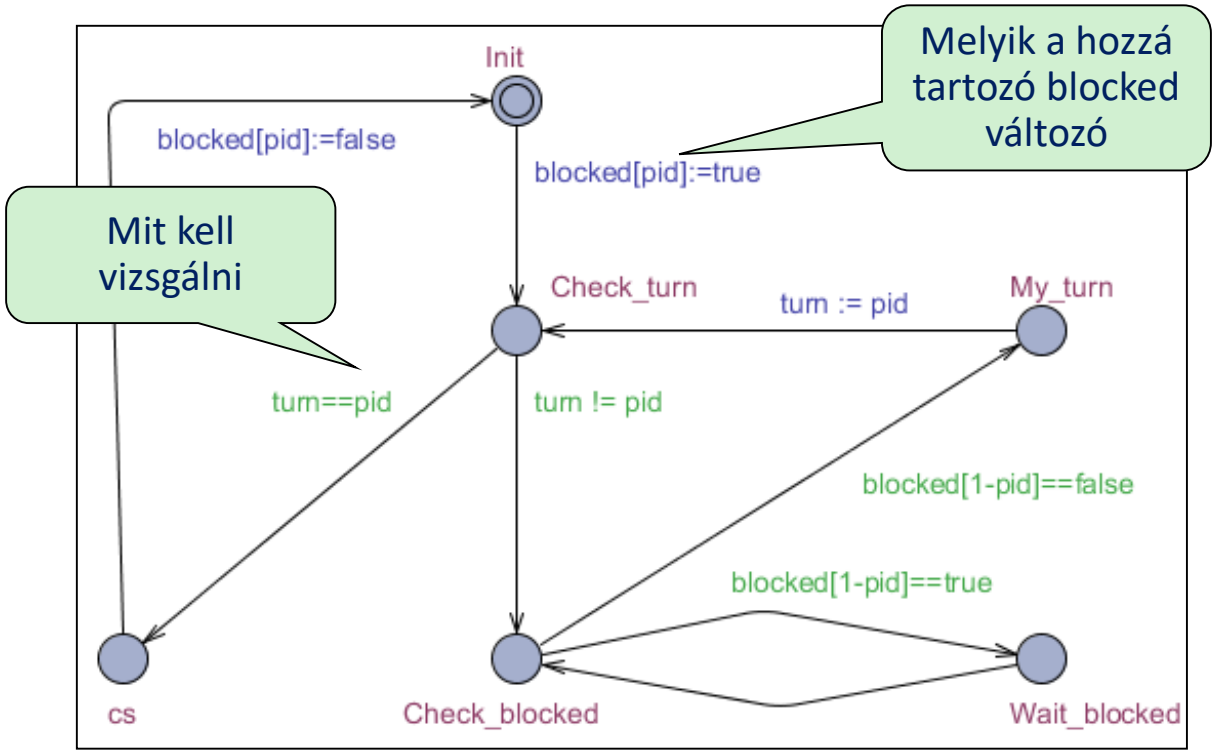
Deklarációk:

```
bool blocked[2];
int[0,1] turn=0;
```

Kihasznált modellezési lehetőségek:

- Azonos viselkedésű résztvevők azonos automata template alapján
- Példányosítás paraméterezéssel
- Változó tömbök (résztvevőkhöz)

A P(int pid) automata template a pid paraméterrel:



Teljes rendszer megadása:

```
P0 = P(0);
P1 = P(1);
system P0, P1;
```

Másik lehetőség:

```
system P;
```

Ha a template paramétere korlátos értékkészletű, pl. int[0,1], akkor minden értékével példányosít

UPPAAL: A követelmények formalizálása

- Kölcsönös kizárás:
  - Soha nem lehet egyszerre P0 és P1 a kritikus szakaszban:
    A[] not (P0.cs and P1.cs)
- Holtpontmentesség:
  - Soha nem alakul ki kölcsönös várakozás (leállás): A[] not deadlock
- Lehetséges az elvárt viselkedés:
  - P0 valamikor be fog lépni a kritikus szakaszba: E<> (P0.cs)
  - P1 valamikor be for lépni 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

- A kölcsönös kizárás nem teljesül!
  - Ellenpélda: Átlapolt végrehajtás (ütemezés) a két résztvevő között, végigjátszható a szimulátorban
  - Javítás: Pl. Peterson, Dekker algoritmusa

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
}
```

UPPAAL: A modellellenőrzés eredménye

- A kölcsönös kizárás nem teljesül!
  - Ellenpélda: Átlapolt végrehajtás (ütemezés) a két résztvevő között, végigjátszható a szimulátorban
  - Javítás: Pl. Peterson, Dekker algoritmusa
- Nincs holtpont
- Lehetséges az elvárt viselkedés
- A kiéheztetés elkerülése nem teljesül
  - Triviális ellenpélda: A kiinduló állapotban marad
    - Ez megengedett (ld. időfüggő viselkedés modellezése)
    - Korlátozni kell egy-egy vezérlési helyen tölthető időt
  - Ezután is lehet kiéheztetés?
    - Van rá példa (az egyik processz ciklikus működése)
    - Az algoritmus önmagában nem garantálja a kiéheztetés elkerülését, fair ütemezés is szükséges

Demó: Dekker algoritmusa

Algoritmus két processz közötti kölcsönös kizárásra megosztott változókkal:

- bool wants_to_enter[0] = false; // P0 processz akar-e belépni
- bool wants_to_enter[1] = false; // P1 processz akar-e belépni
- int [0,1] turn = 0; // P0 fog belépni (0 esetén) vagy P1 (1 esetén)

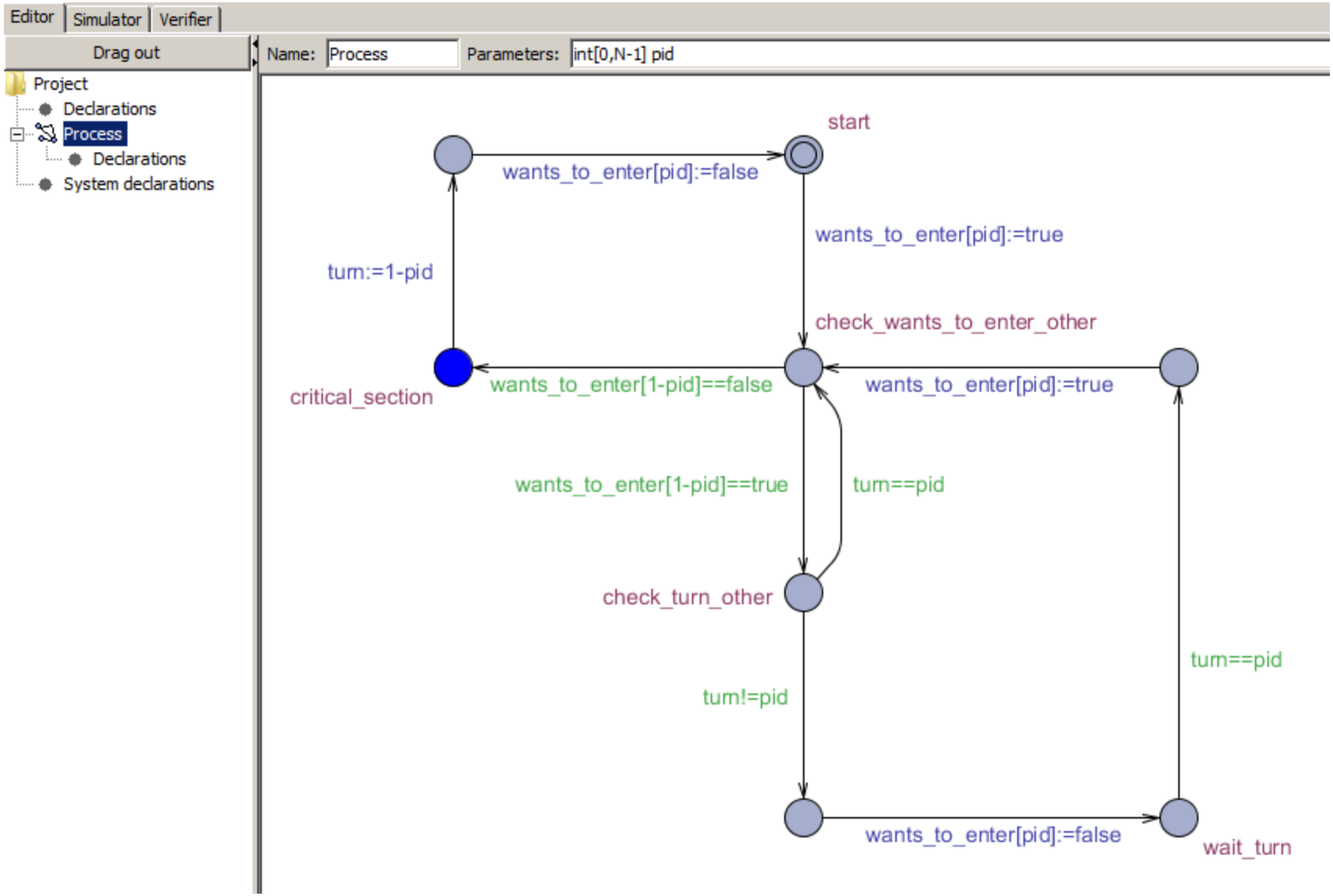
P0

```
while (true) {
   wants_to_enter[0] := true
   while (wants_to_enter[1]) {
      if (turn != 0) {
         wants_to_enter[0] := false
         while (turn != 0) {
           // Busy wait
         }
         wants_to_enter[0] := true
      }
   }
   // Critical section
   turn := 1
   wants_to_enter[0] := false
   // Do other things
}
```

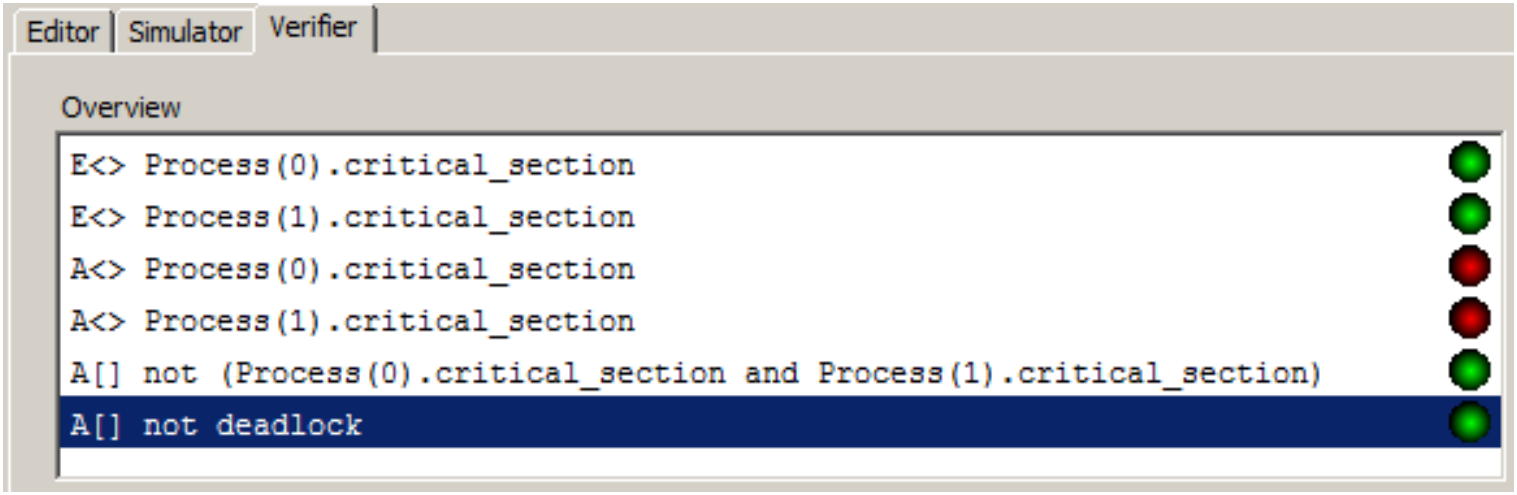
P1

```
while (true) {
   wants_to_enter[1] := true
   while (wants_to_enter[0]) {
      if (turn != 1) {
         wants_to_enter[1] := false
         while (turn != 1) {
           // Busy wait
         }
         wants_to_enter[1] := true
      }
   }
   // Critical section
   turn := 0
   wants_to_enter[1] := false
   // Do other things
}
```

Demó: Dekker algoritmusának modellje



Demó: Dekker algoritmusának ellenőrzése



- Teljesülő követelmények:
  - Belépés lehetősége: E<>Process(i).critical_section
  - Kölcsönös kizárás: A[] not (Process(0).critical_section and Process(1).critical_section)
  - Holtpontmentesség: A[] not deadlock
- Nem teljesülő követelmények:
  - Kiéheztetés-mentesség: A<>Process(i).critical_section

Demó: Dekker ellenpéldák és modell módosítások

- A processzek végtelen ideig várakozhatnak adott vezérlési helyeken
  - Ha nem releváns az időfüggő viselkedés: Várakozás korlátozása Urgent vezérlési helyekkel
  - Ha releváns az időfüggő viselkedés: Várakozás korlátozása óraváltozók és vezérlési hely invariánsok használatával
- Az egyik processz ciklikusan működik, míg a másik processz nem lép előre (nem fair az ütemezés)
  - Ha nem releváns az időfüggő viselkedés: Ütemező (scheduler) modellezése (pl. szinkronizációval), ami garantálja a processzek fair végrehajtását
  - Ha releváns az időfüggő viselkedés: Ciklusok végrehajtási idejének modellezése (időfüggő őrfeltétel a továbblépésre), ami esélyt ad közben a másik processz haladására