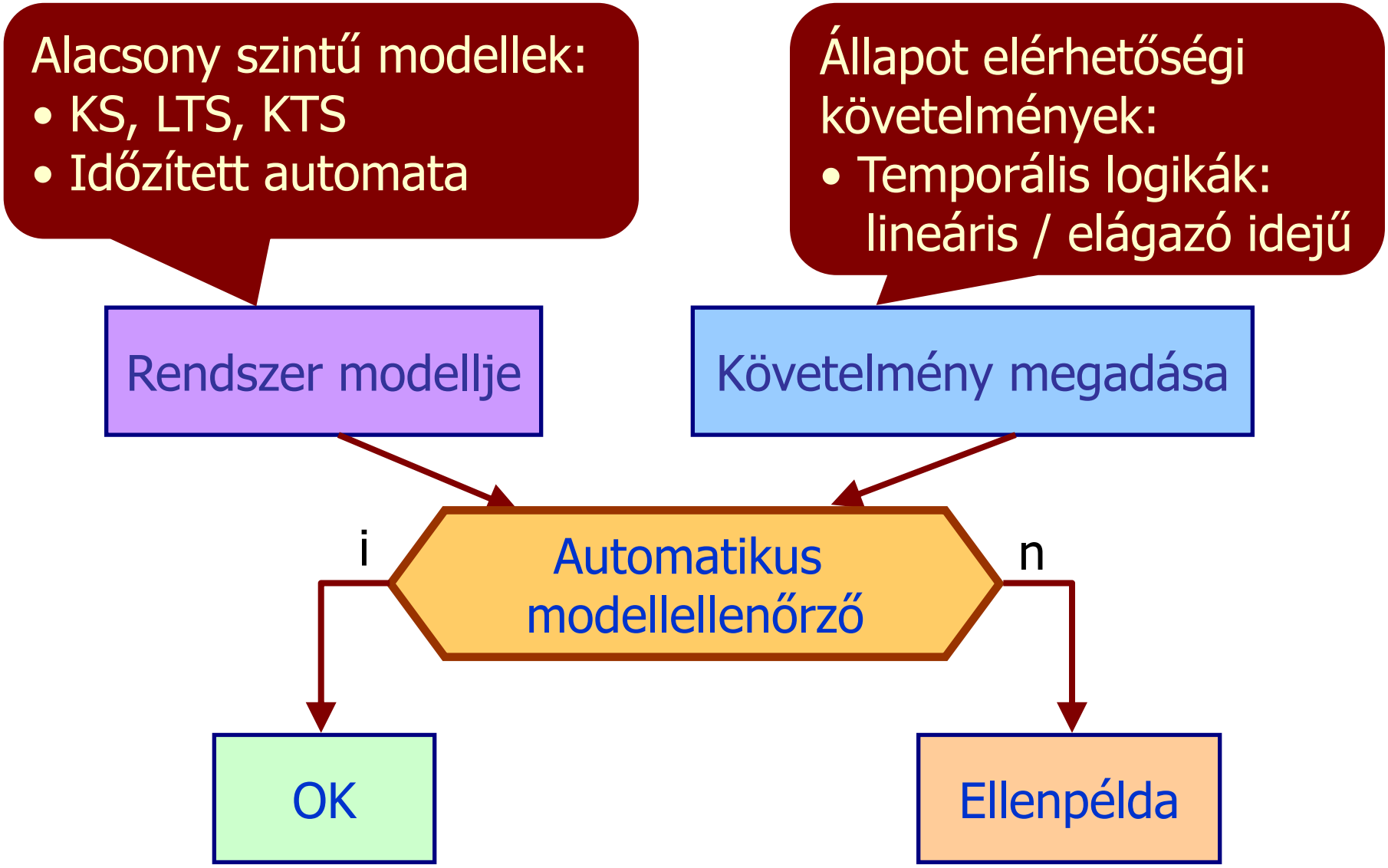
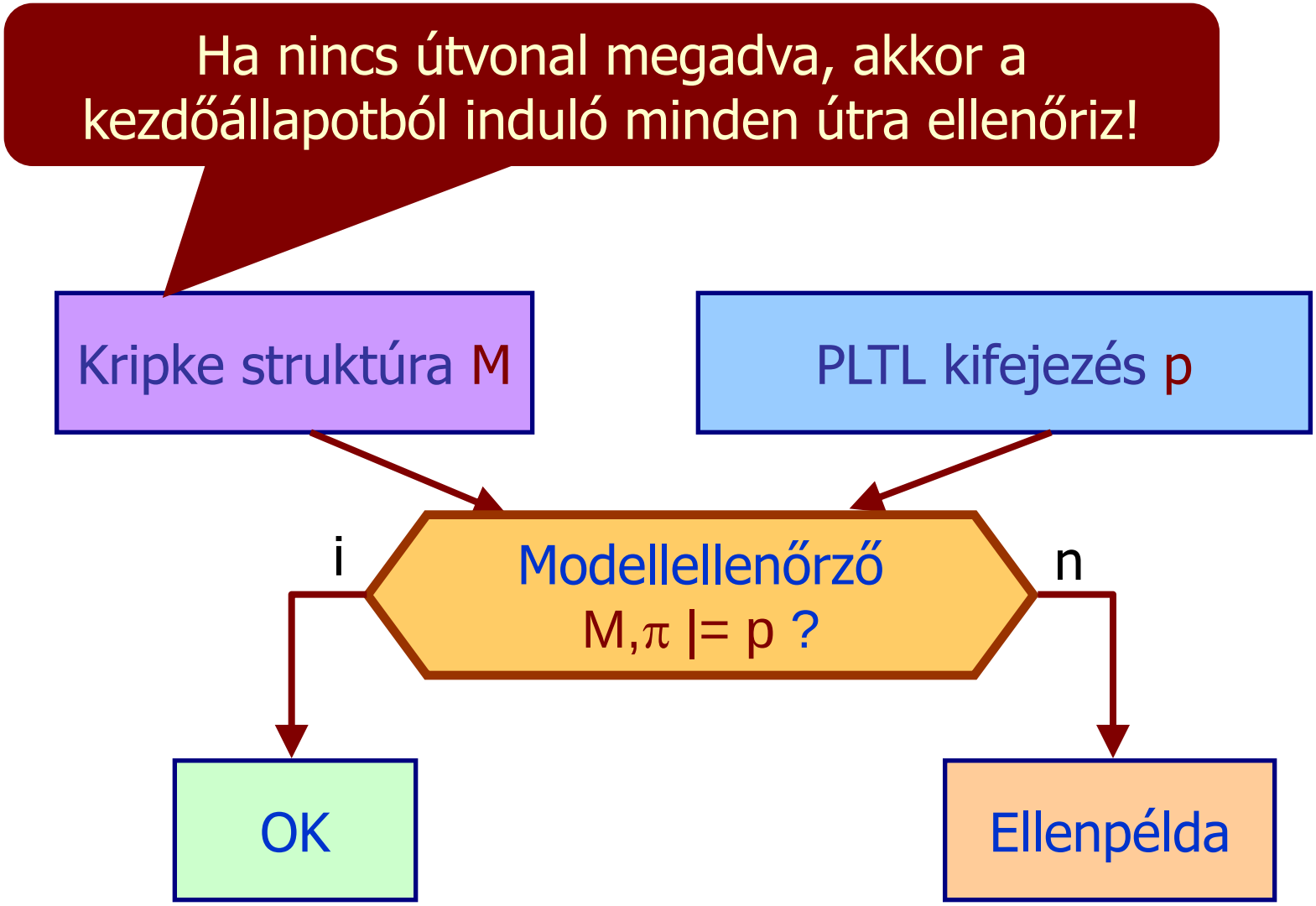
A modellellenőrzési feladat (bevezető)

dr. Majzik István
BME Méréstechnika és Információs Rendszerek Tanszék

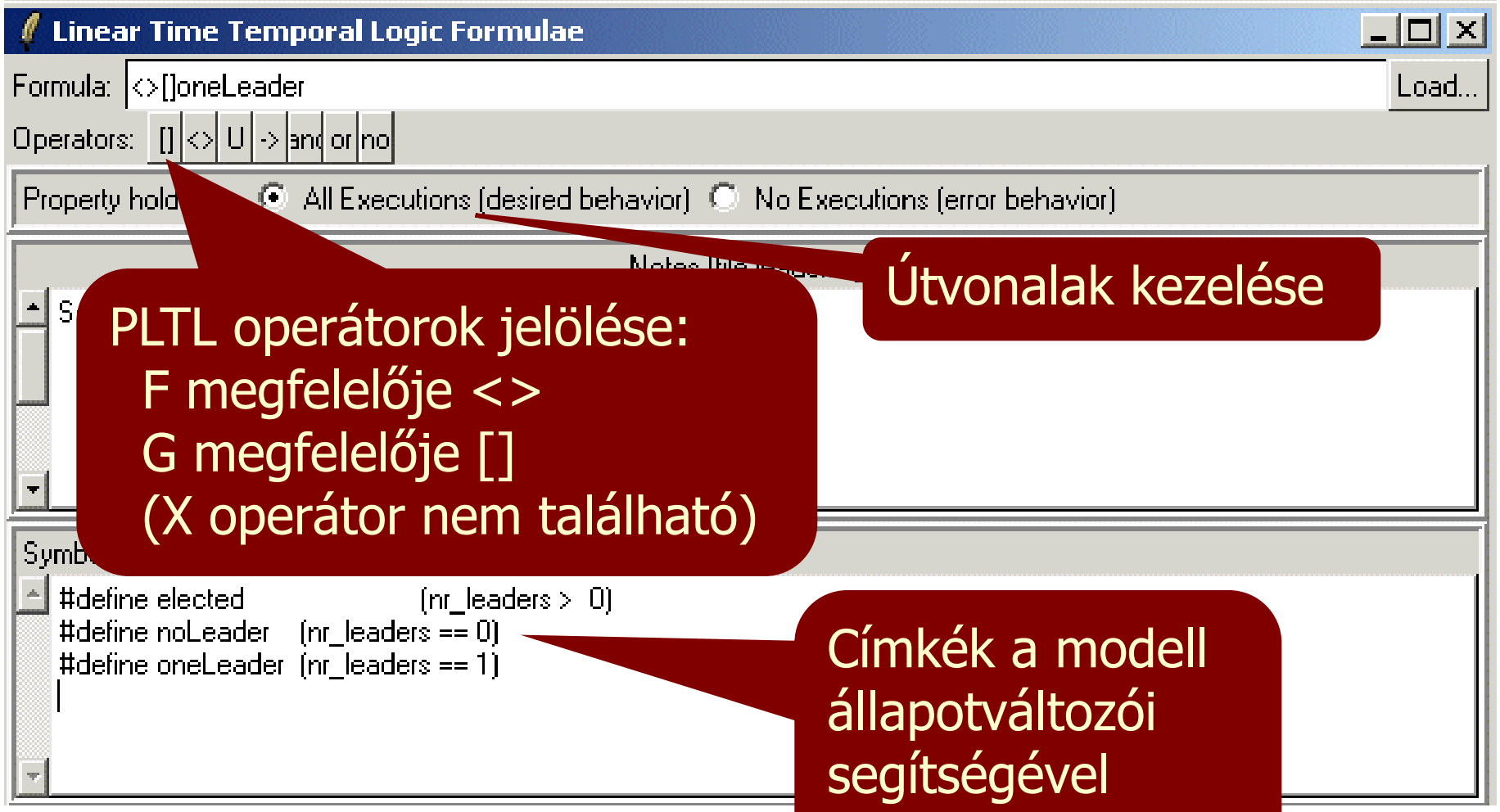
Ismétlés: Mit szeretnénk elérni?



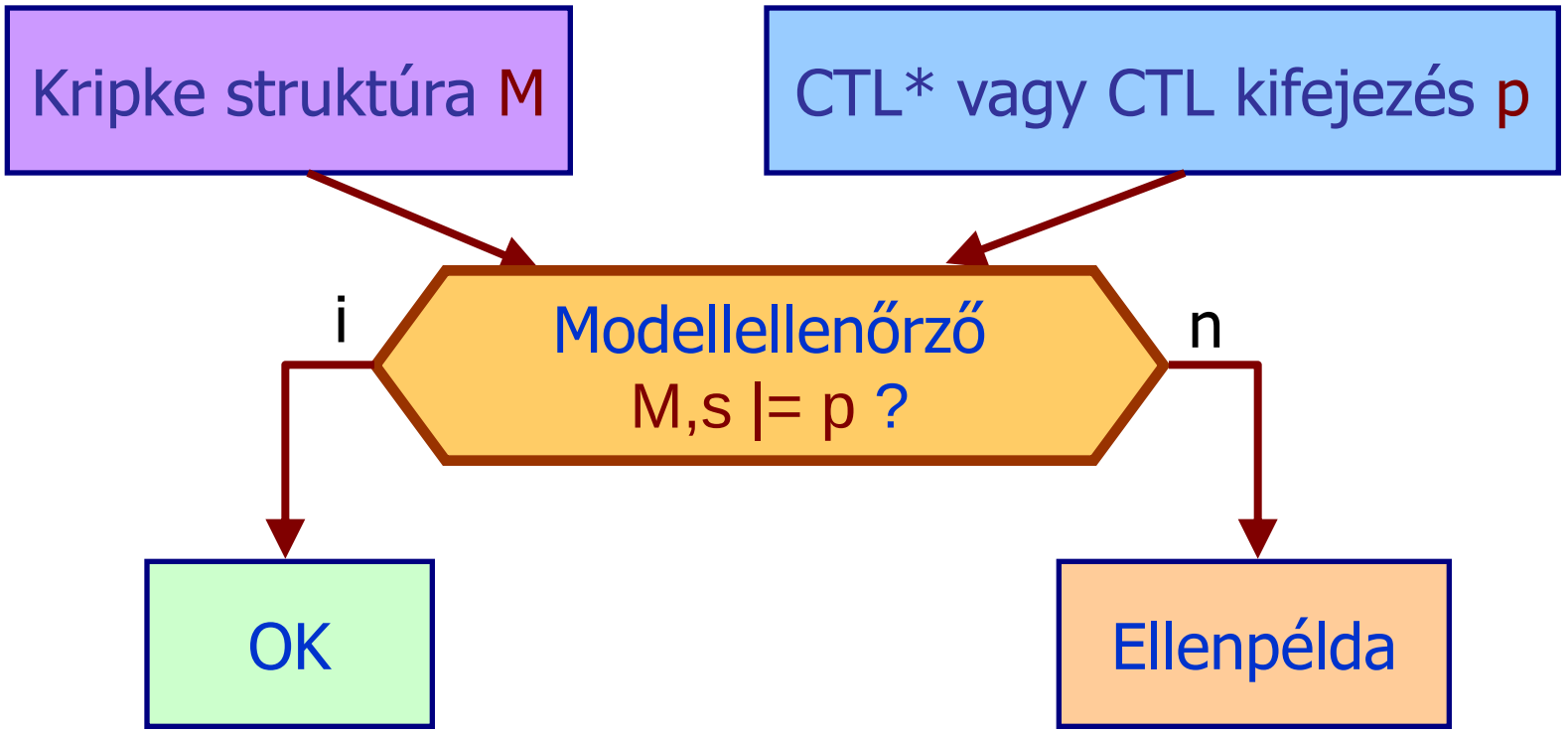
PLTL modellellenőrzés



A SPIN modellellenőrző (régi felület)



CTL* vagy CTL modellellenőrzés



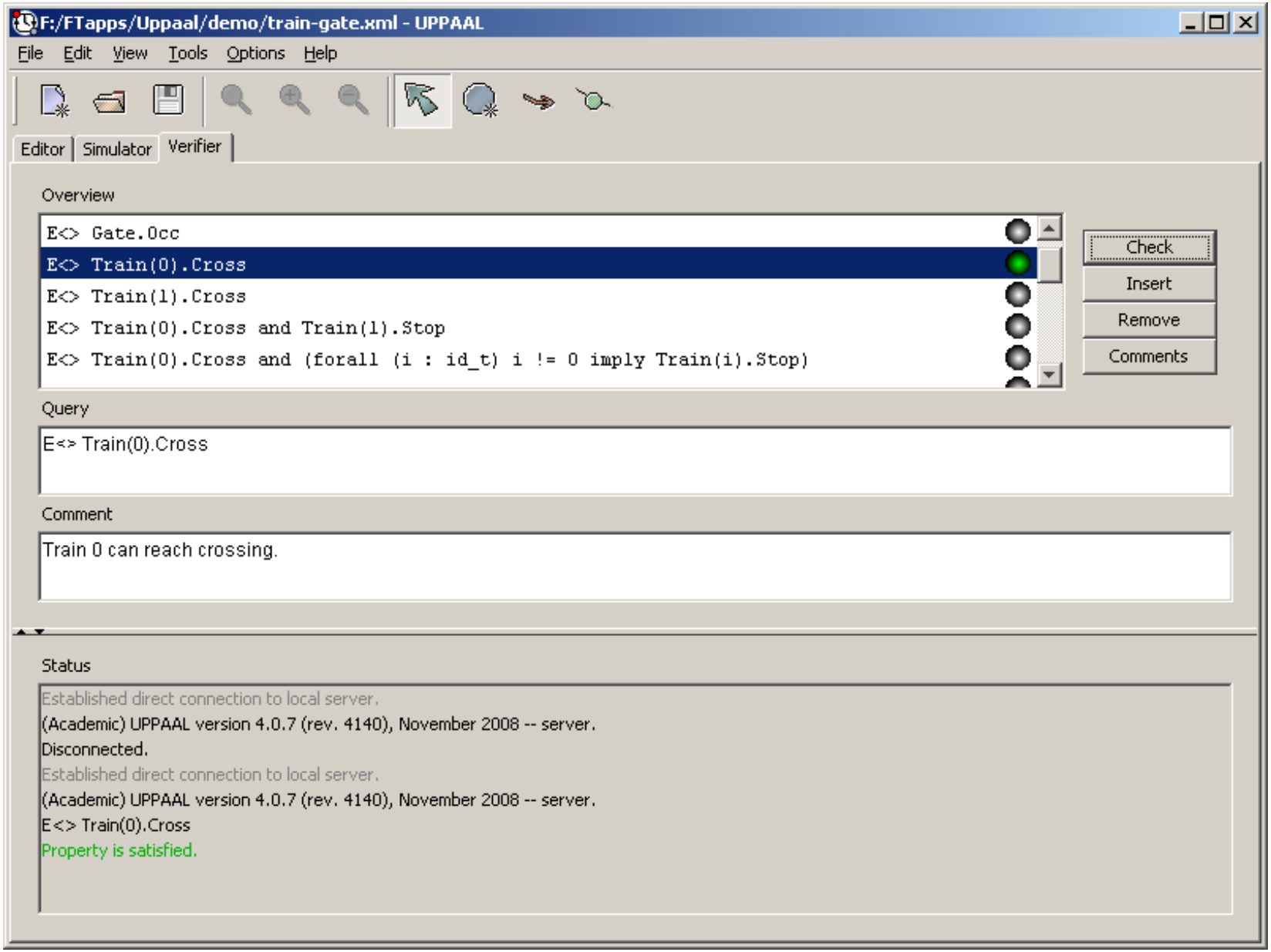
Az UPPAAL modellellenőrző lehetőségei

- Atomi kijelentések:
  - Változók értéke hivatkozható: pl. a!=1
    - Egész aritmetika és bitenkénti műveletek használhatók
  - Állapot hivatkozható: pl. Train(0).cross
    - Paraméterezett processzekre: forall, exists operátorok
  - Holtpont: deadlock kijelentéssel megadható (nincs akció)
- Boole logikai operátorok:
  - and, or, imply, not, ? : (ez utóbbi az if-then-else)
- Temporális operátorok: Korlátozott CTL
  - Kifejezés elején szerepelhet egy temporális operátor
    - Speciális lehetőség: p-->q, jelentése AG (p imply AF q)
  - Jelölés: [] szerepel G helyett, <> szerepel F helyett
    - Így lesz: A[], A<>, E[], E<>
    - E[] esetén véges útvonalon is értelmezett (végállapotig)

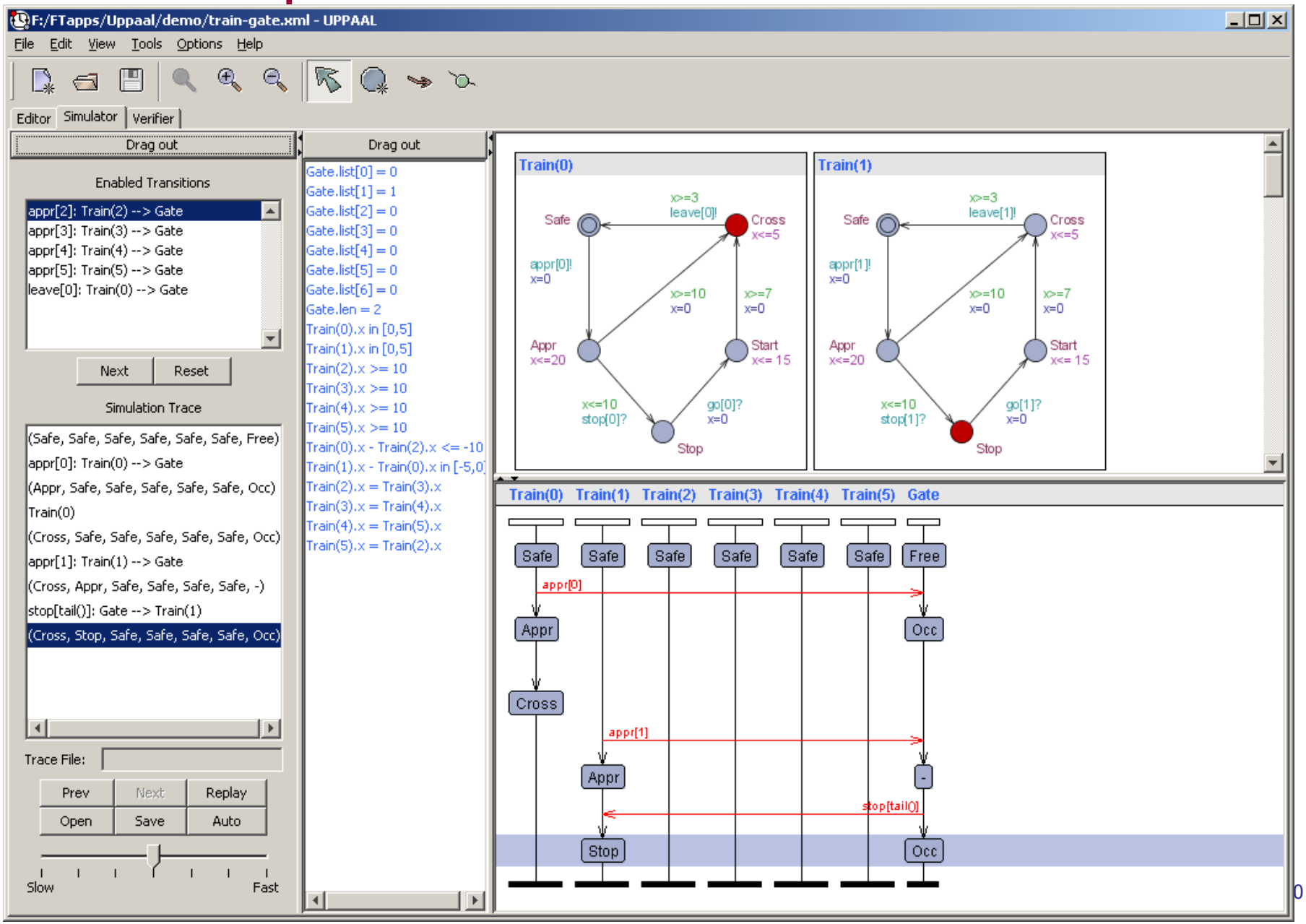
Követelmények ellenőrzése az UPPAAL-ban

- Követelmények halmaza szerkeszthető
- Modellellenőrzés egyenként is indítható
- Ellenpélda generálható
  - Legrövidebb, leggyorsabb, vagy akármi
  - Betölthető a szimulátorba (végigjátszható)
- Keresés az állapottérben
  - Mélységi
  - Szélességi
- Állapottárolás különféle opciókkal
  - Redukció
  - Közelítő állapottér tárolás (alul- illetve felülbecslés)
  - Hash tábla mérete megadható

Az UPPAAL modellellenőrző ablaka



Ellenpélda az UPPAAL szimulátorban



A 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
}
```

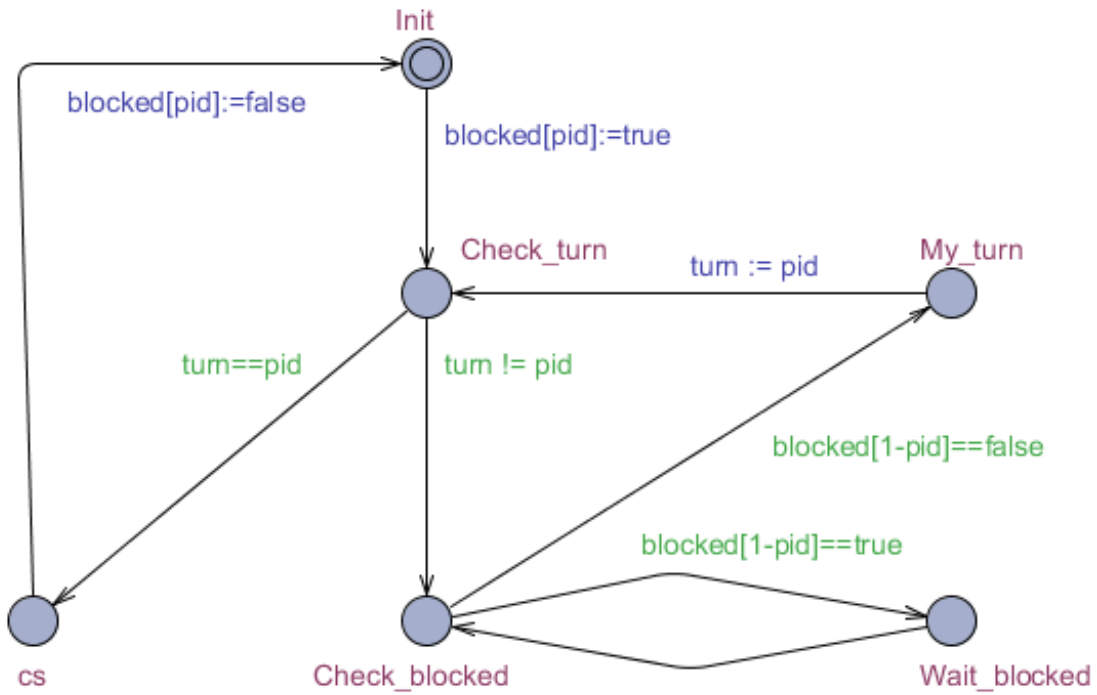
Helyes-e ez az algoritmus?

A modell UPPAAL-ban

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
}
```

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 leállás (amikor nincs továbblépés): A[] not deadlock
- Lehetséges az elvárt viselkedés:
  - P0 beléphet a kritikus szakaszba: E<> (P0.cs)
  - P1 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

- A kölcsönös kizárás nem teljesül!
  - Ellenpélda: Átlapolódás 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: Átlapolódás 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

Dekker algoritmusa (demo)

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)

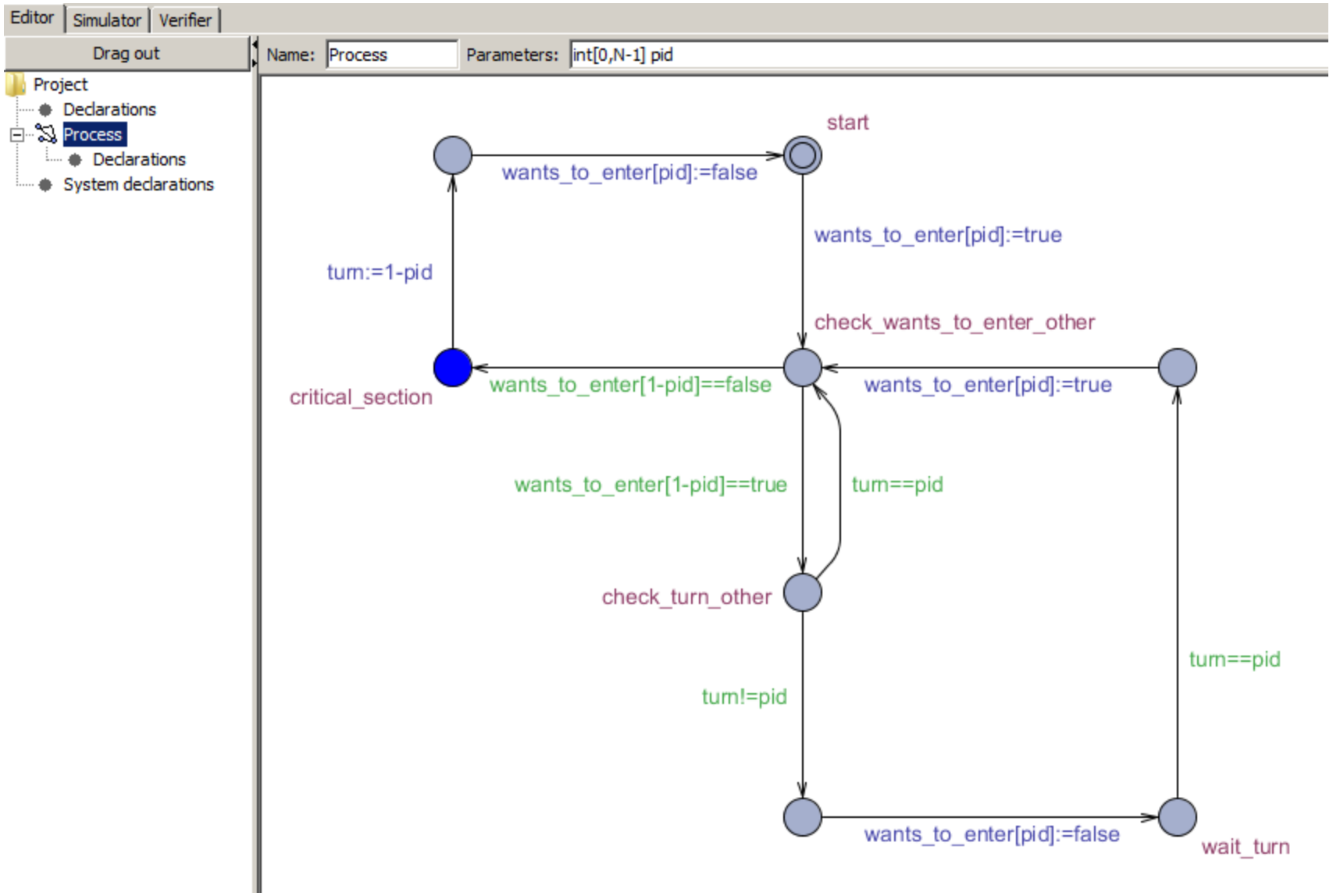
P0

```
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
}
```

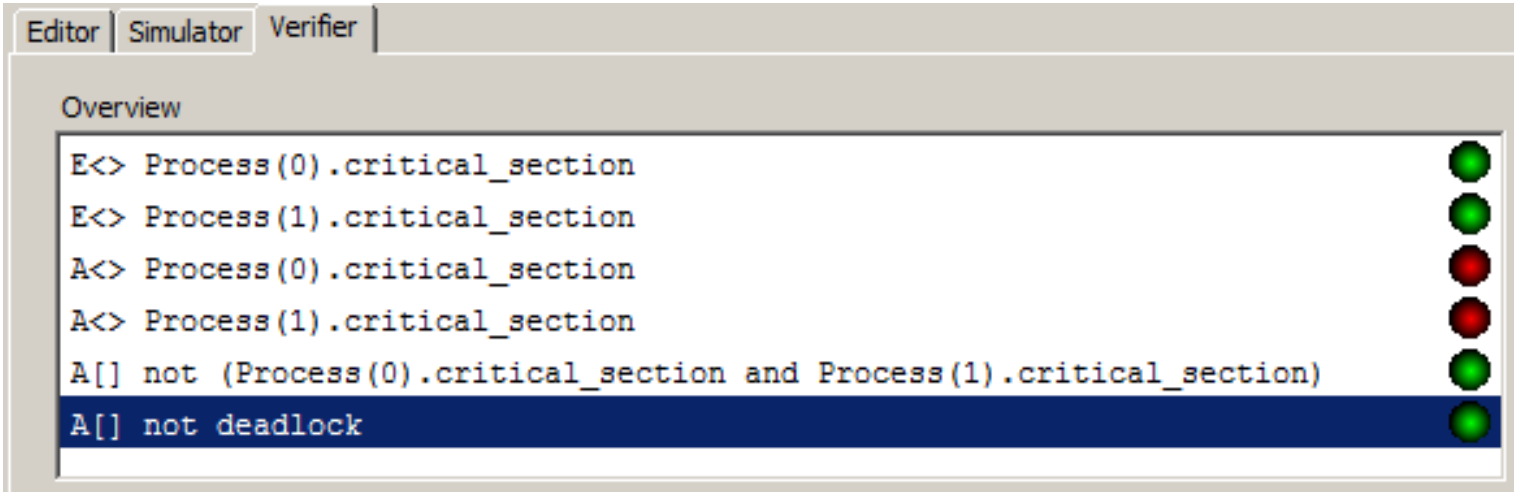
P1

```
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
}
```

Dekker algoritmusának modellje (demo)



Dekker algoritmusának ellenőrzése (demo)



- 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

Dekker algoritmus modelljének módosítása (demo)

- Ellenpéldák a kiéheztetés-mentességhez
  - A processzek végtelen ideig várakoznak egy állapotban
  - Az egyik processz ciklikusan működik, míg a másik processz nem lép előre
- Modellezési lehetőségek az ellenpéldák elkerülésére
  - Ha nem fontos az időfüggő viselkedés:
    - Adott helyen való várakozás korlátozása Urgent helyekkel és
    - ütemező (scheduler) modellezése, ami garantálja a fair végrehajtást (egyszerű példa: szinkronizációval)
  - Ha fontos az időfüggő viselkedés:
    - Adott helyen eltöltött idő korlátozása hely invariánsokkal és
    - késleltetés („időtöltés”) a ciklusokban, ami esélyt ad a másik processz haladására,
      vagy ütemező (scheduler) modellezése, ami garantálja a fair végrehajtást (egyszerű példa: szinkronizációval)