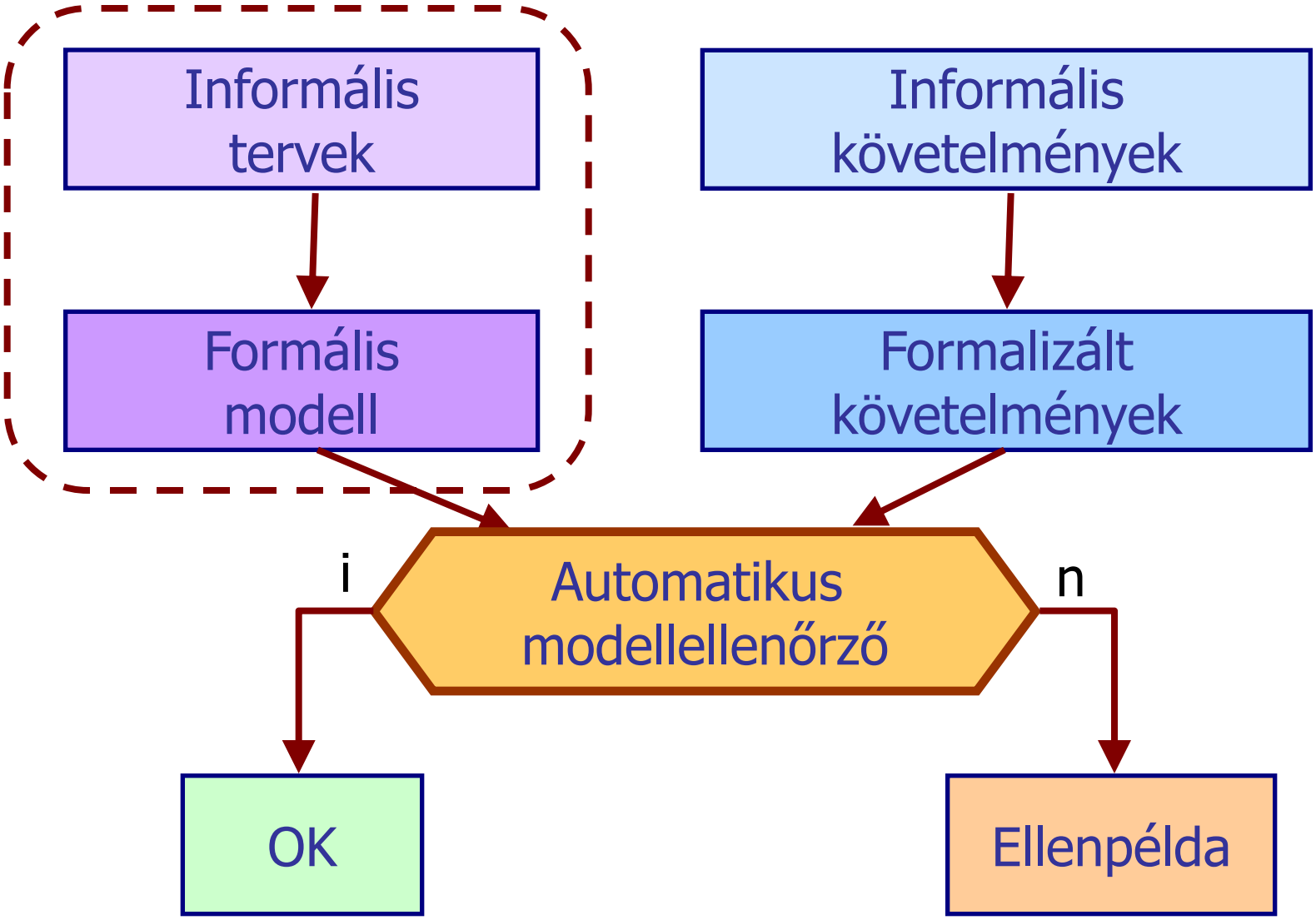
Alapszintű formalizmusok

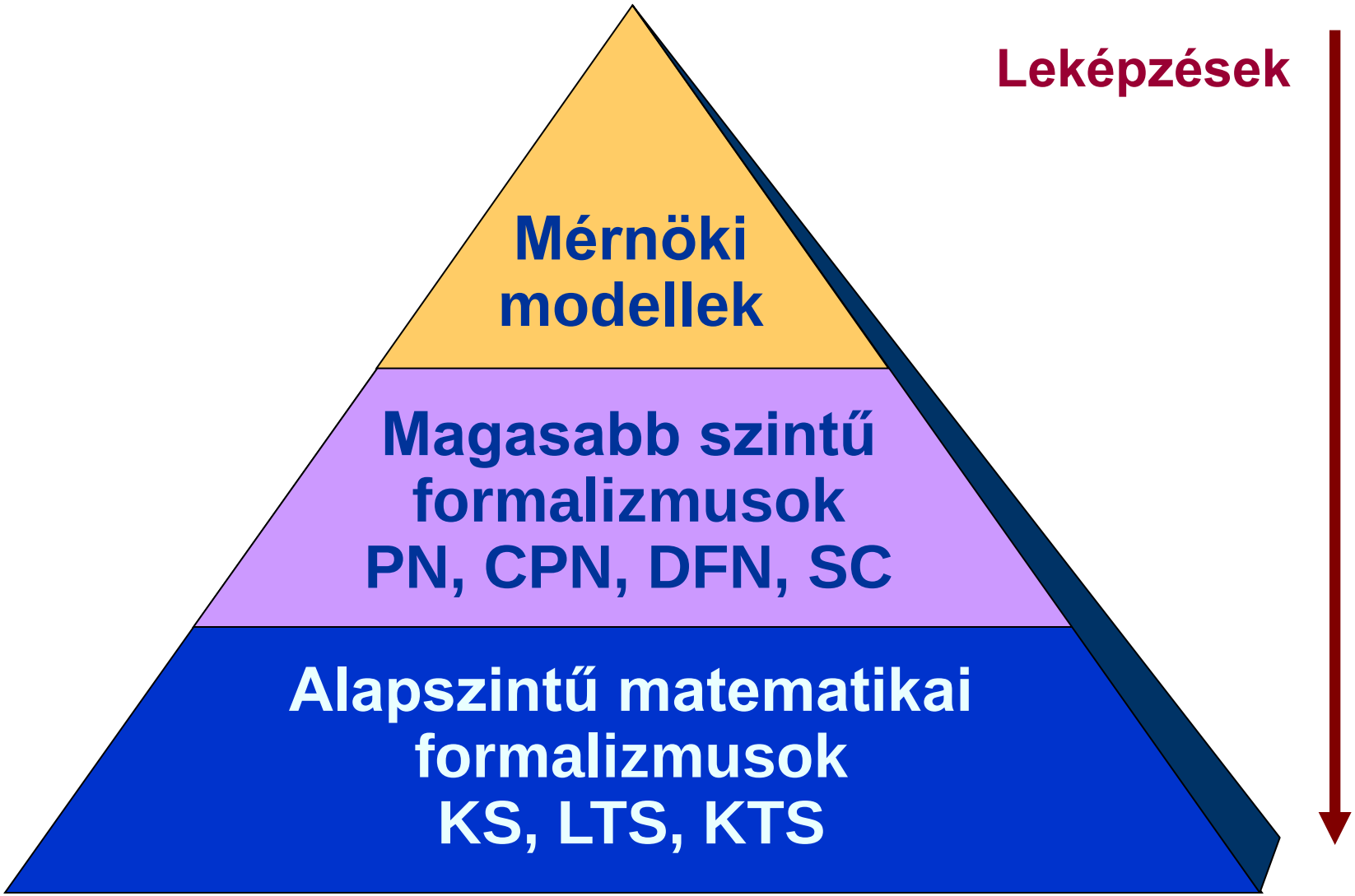
dr. Majzik István

BME Méréstechnika és Információs Rendszerek Tanszék

Mit szeretnénk elérni?



Modellek a formális ellenőrzéshez



Alapszintű formalizmusok (áttekintés)

- Kripke-struktúrák (KS)
  - Állapotok, állapotátmenetek
  - Állapotok lokális tulajdonságai mint címkék
- Címkézett tranzíciós rendszerek (LTS)
  - Állapotok, állapotátmenetek
  - Állapotok lokális tulajdonságai mint címkék
- Kripke tranzíciós rendszerek (KTS)
  - Állapotok, állapotátmenetek
  - Állapotok és lokális tulajdonságai mint címkék
- Véges állapotú automaták (időkezeléssel)
  - Kiterjesztések: Változók, óraváltozók, szinkronizáció

1. Kripke-struktúra

KS, Kripke-structure:

- Állapotok tulajdonságait fejezzük ki: címkézés atomi kijelentésekkel
- Egy állapothoz sok címke rendelhető

Alkalmazás: Viselkedés, algoritmus leírása

$KS = (S, R, L)$ és AP, ahol

$\mathrm{AP} = \{\mathrm{P}, \mathrm{Q}, \mathrm{R}, \ldots\}$ atomi kijelentések halmaza (domén-specifikus)

$\mathrm{S} = \{\mathrm{s}_1, \mathrm{s}_2, \mathrm{s}_3, \ldots \mathrm{s}_n\}$ állapotok halmaza

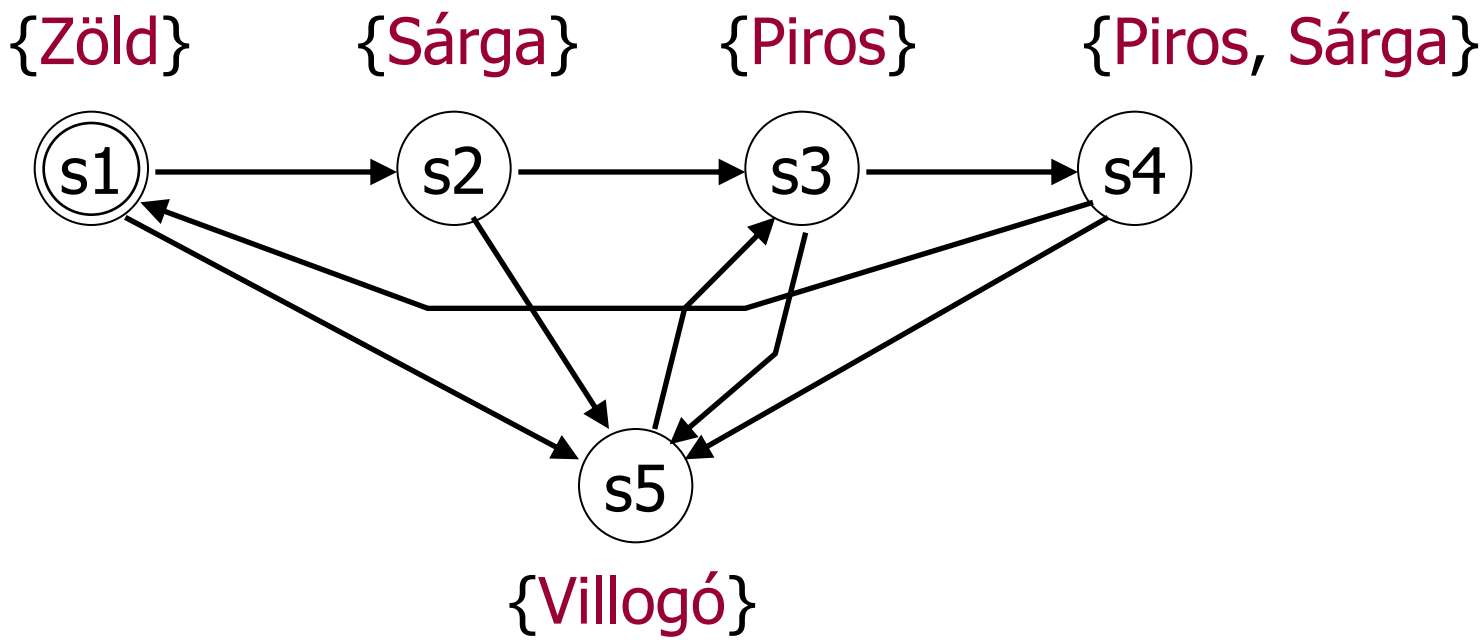
$\mathrm{R} \subseteq \mathrm{S} \times \mathrm{S}$: állapotátmeneti reláció

$\mathrm{L}: \mathrm{S} \to 2^{AP}$ állapotok címkézése atomi kijelentésekkel

Kripke-struktúra példa

Közlekedési lámpa viselkedése

- AP={Zöld, Sárga, Piros, Villogó} lámpa képe
- S = {s1, s2, s3, s4, s5}



2. Címkézett tranzíciós rendszer

LTS, Labeled Transition System:

- Állapotátmenetek tulajdonságait fejezzük ki: címkézés akciókkal
- Egy átmeneten csak egy akció szerepelhet

Alkalmazás: Kommunikáció, protokollok modellezése

$LTS = (S, Act, \rightarrow)$, ahol

$S = \{s_1, s_2, \ldots s_n\}$ állapotok halmaza

$Act = \{a, b, c, \ldots\}$ akciók (címkék) halmaza

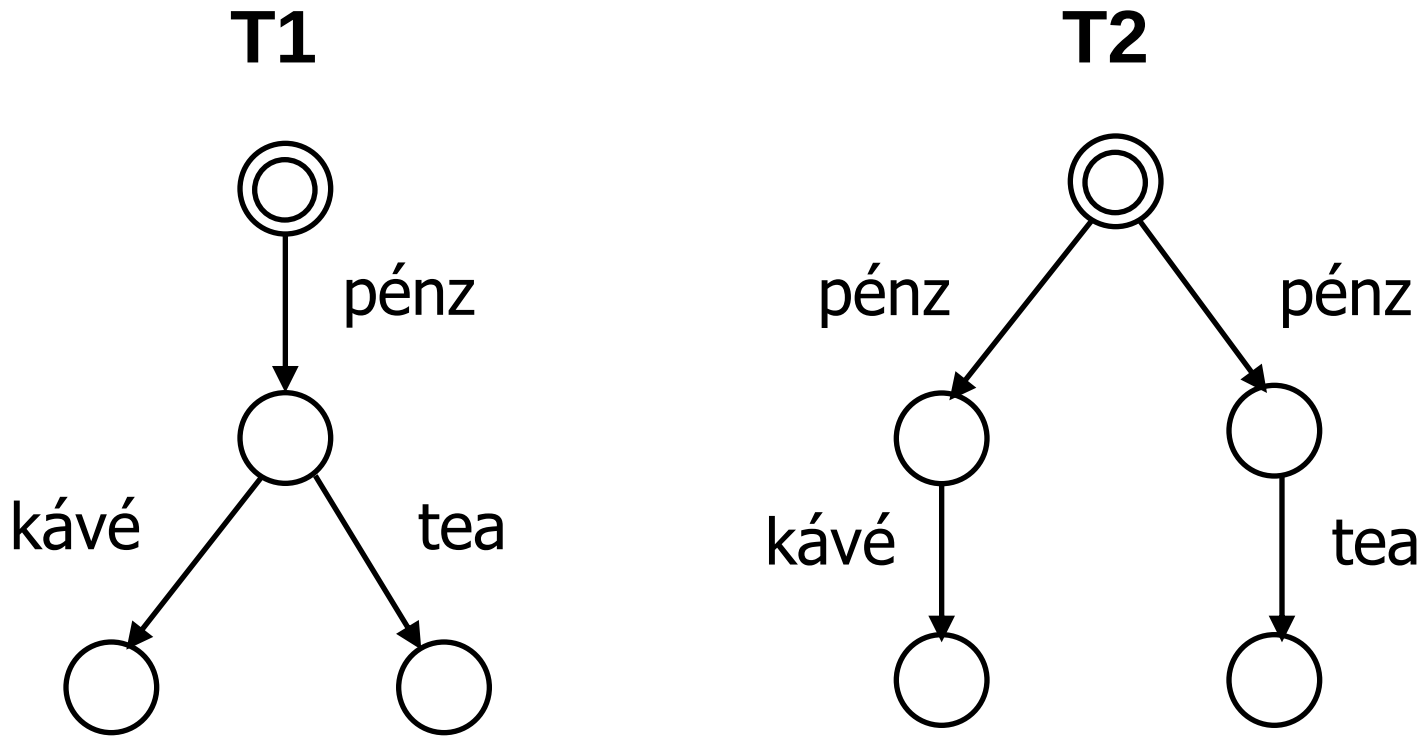
$\rightarrow \subseteq S \times Act \times S$ címkézett állapotátmenetek

Állapotátmenetek szokásos jelölése: $s_1 \xrightarrow{a} s_2$

LTS példák

- Italautomata modelljei

  Act = {pénz, kávé, tea} interakciók a felhasználóval



3. Kripke tranzíciós rendszer

KTS, Kripke Transition System:

- Állapotok és átmenetek tulajdonságait is kifejezzük: címkézés atomi kijelentésekkel és akciókkal
- Egy állapothoz sok címke rendelhető, egy átmenethez egy címke rendelhető

$KTS = (S, \rightarrow, L)$ és AP, Act, ahol

$AP = \{P, Q, R, \ldots\}$ atomi kijelentések halmaza (domén-specifikus)

$Act = \{a, b, c, \ldots\}$ akciók halmaza

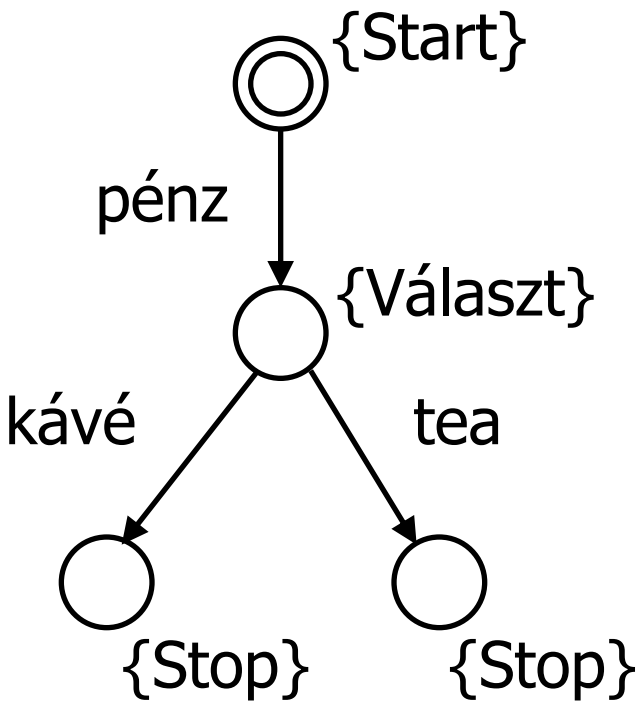
$S = \{s_1, s_2, s_3, \ldots s_n\}$ állapotok halmaza

$\rightarrow \subseteq S \times Act \times S$ állapotátmeneti reláció

$L: S \rightarrow 2^{AP}$ állapotok címkézése atomi kijelentésekkel

KTS példa

- Italautomata modellje állapot címkékkel
  - Act = {pénz, kávé, tea} interakciók a felhasználóval
  - AP = {Start, Választ, Stop} állapotok kijelzése



Időzített automaták és az UPPAAL eszköz

Automaták és változók

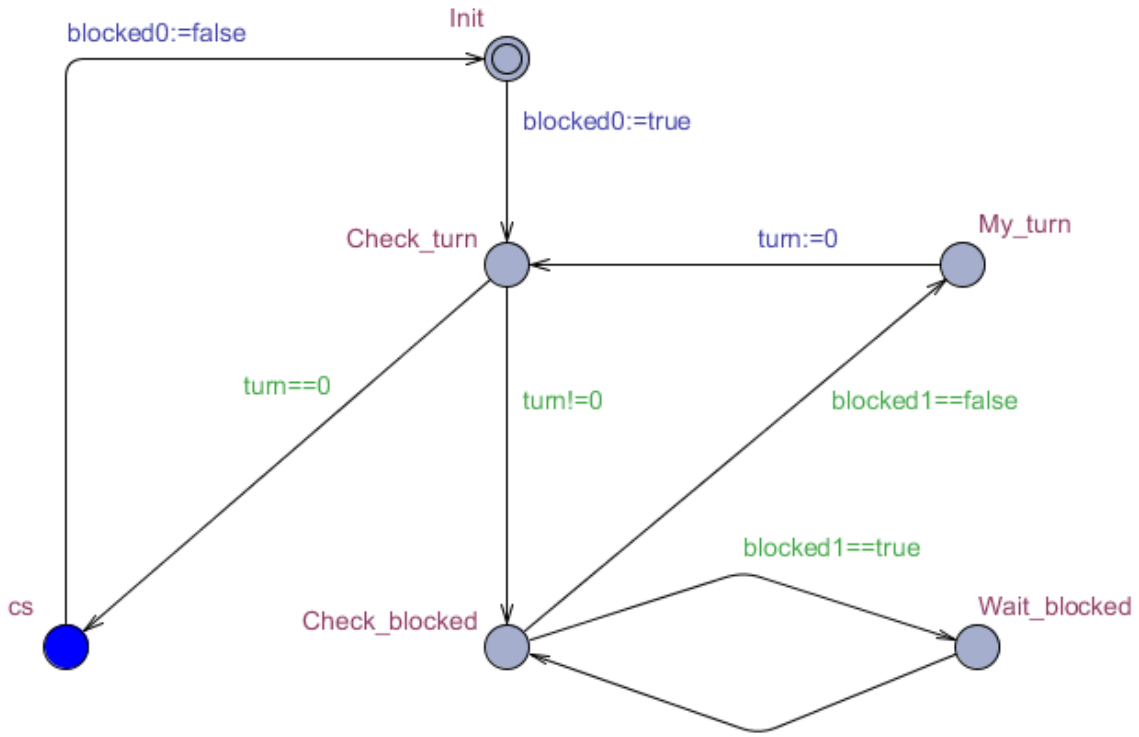
- Cél: Állapot alapú viselkedés modellezése
- Alap formalizmus: Véges állapotú automata (FSM)
  - Állapotok (névvel hivatkozhatók)
  - Állapotátmenetek
- Nyelvi kiterjesztés: Egész értékű változók használata
  - Változók deklarálhatók típussal (értéktartománnyal)
  - Konstansok definiálhatók
  - Egész aritmetika használható
- Használat állapotátmeneteken:
  - Őrfeltétel hozzárendelése: A változókon kiértékelhető predikátum
    - Az átmenet bekövetkezéséhez igaz kell legyen
  - Akció hozzárendelése: Értékadás változóknak
    - Az átmenet bekövetkezésekor végrehajtódik

Példa: Automata változókkal

Deklarációk:

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

A P0 automata:

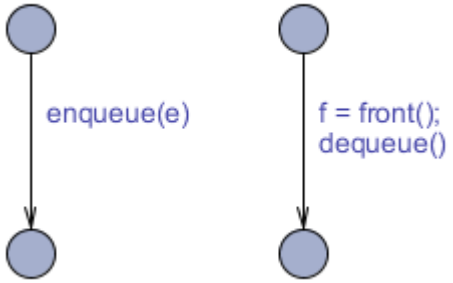


P0

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

Függvények alkalmazása átmenetek akcióiban

Példa: Lista kezelése




```
const int N = 6;
typedef int[0,N-1] id_t;
id_t list[N+1];
int[0,N] len;

// Put an element at the end of the list
void enqueue(id_t element)
{
        list[len++] = element;
}
```

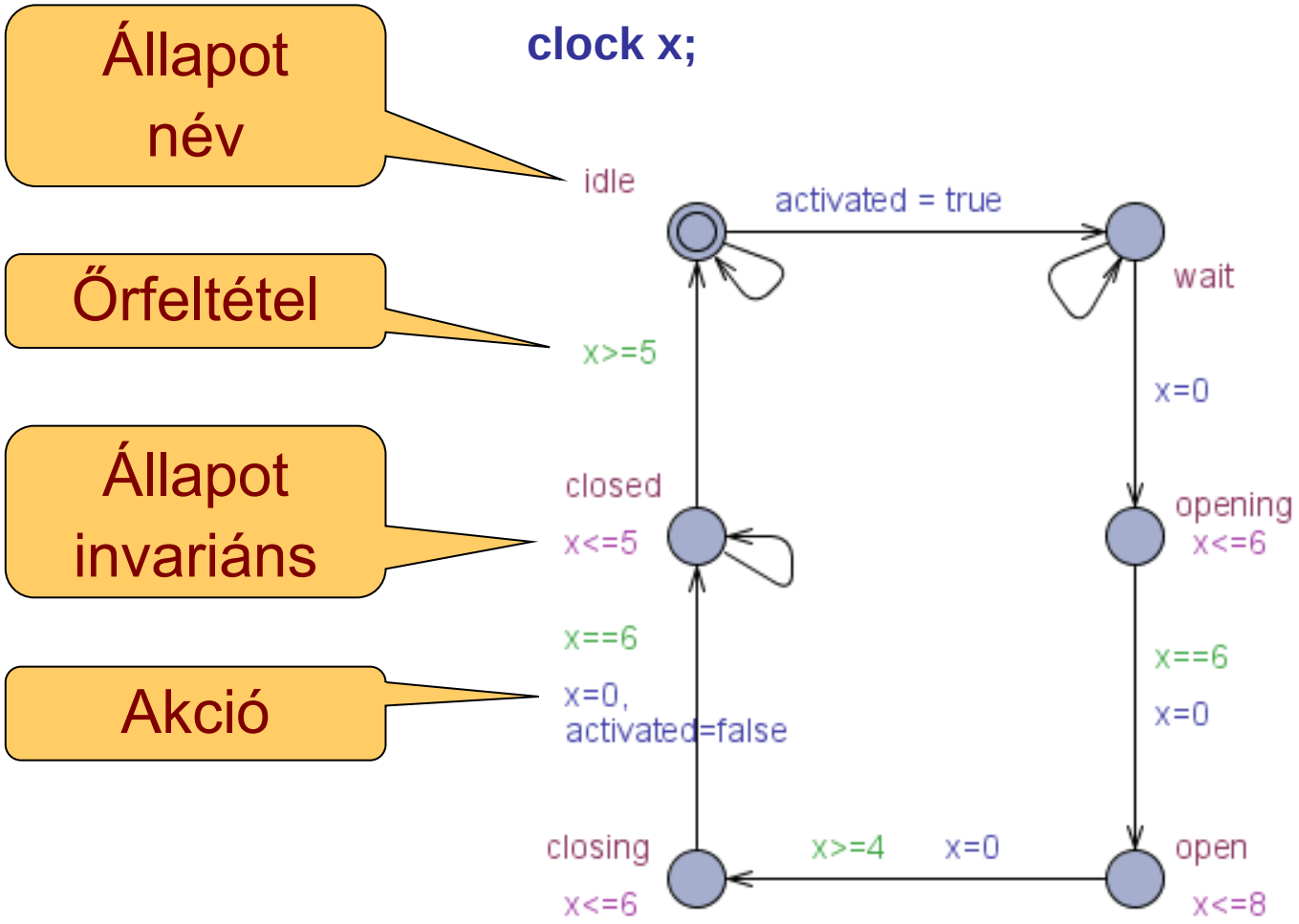
```
// Returns the front element of the list
id_t front()
{
   return list[0];
}

// Remove the front element of the list
void dequeue()
{
        int i = 0;
        len -= 1;
        while (i < len)
        {
                list[i] = list[i + 1];
                i++;
        }
        list[i] = 0;
}
```

Kiterjesztések óraváltozókkal

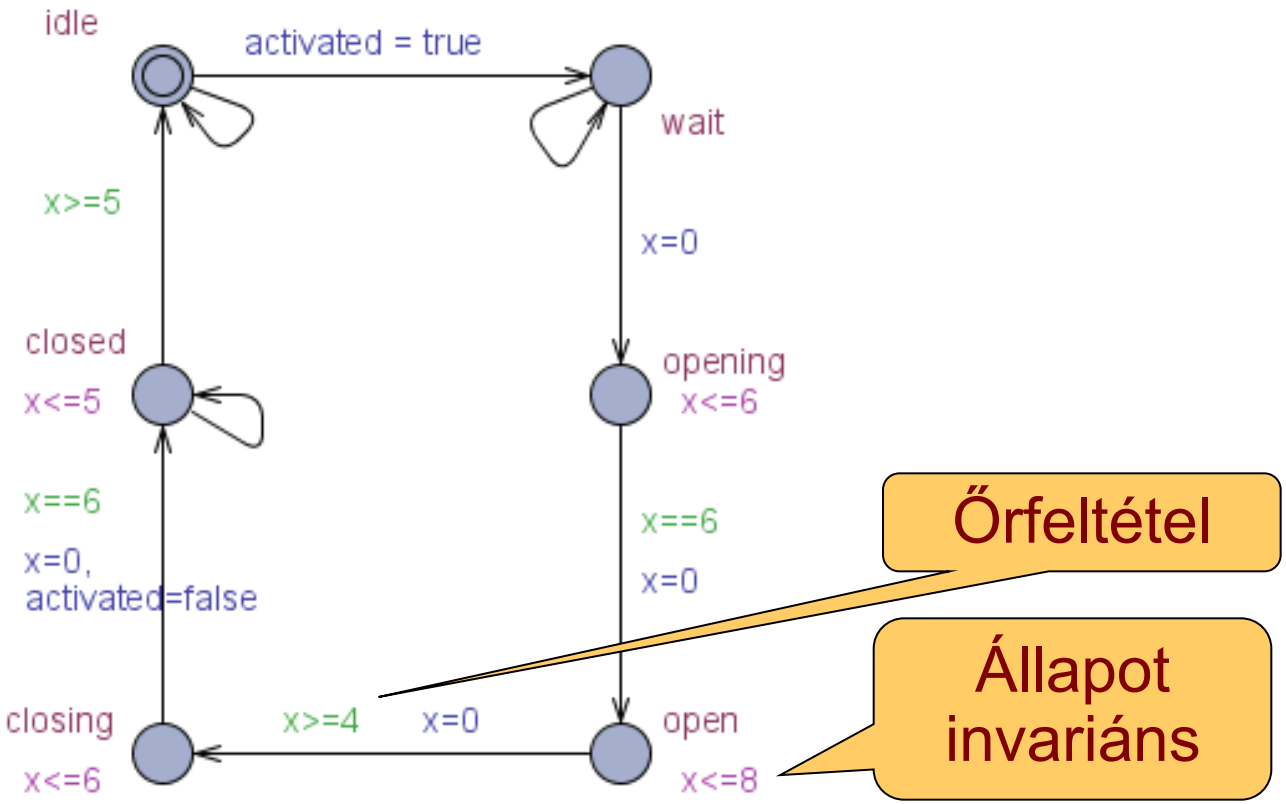
- Cél: Valósidejű viselkedés modellezése
  - Idő telik az állapotokban
  - Relatív időmérés (pl. time-out): Időzítő resetelése és leolvasása
  - Az idő függvényében változó a viselkedés
- Nyelvi kiterjesztés: Óraváltozók
  - Azonos rátával automatikusan „haladó” konkurens órák (időzítők)
- Használat állapotátmenetekben:
  - Akciók: Óraváltozók nullázása (resetelés), egymástól függetlenül
  - Őrfeltételek: Óraváltozók és konstansok használhatók a predikátumokban
- Használat állapotokban:
  - Állapot invariánsok: Predikátum óraváltozókon és konstansokon, megadja, meddig állhat fenn az adott állapot

Példa: Időzített automata (az UPPAAL eszközben)



Az invariánsok és őrfeltételek szerepe

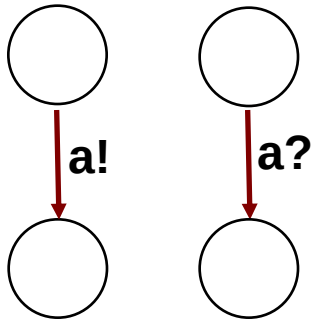
clock x;



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

Kiterjesztések elosztott rendszerekhez

- Cél: Együttműködő automaták hálózatának modellezése
  - Szinkronizáció az egyes automaták között
  - Együttlépő átmenetek (randevú): szinkron kommunikáció
    - Üzenet küldés és fogadás csak együtt valósulhat meg (küldő vár)
    - Ezzel aszinkron kommunikáció is leírható
- Nyelvi 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



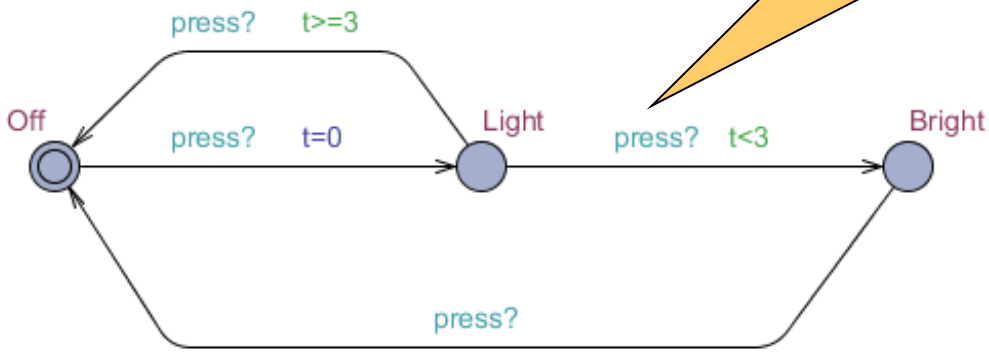
- Csatornák kezelése
  - Csatornatömb: a[]
  - Csatorna megadás: Pl. a[id] csatorna egy id változó esetén, a[id]! illetve a[id]? akciók

Példa óraváltozókra és szinkronizálásra

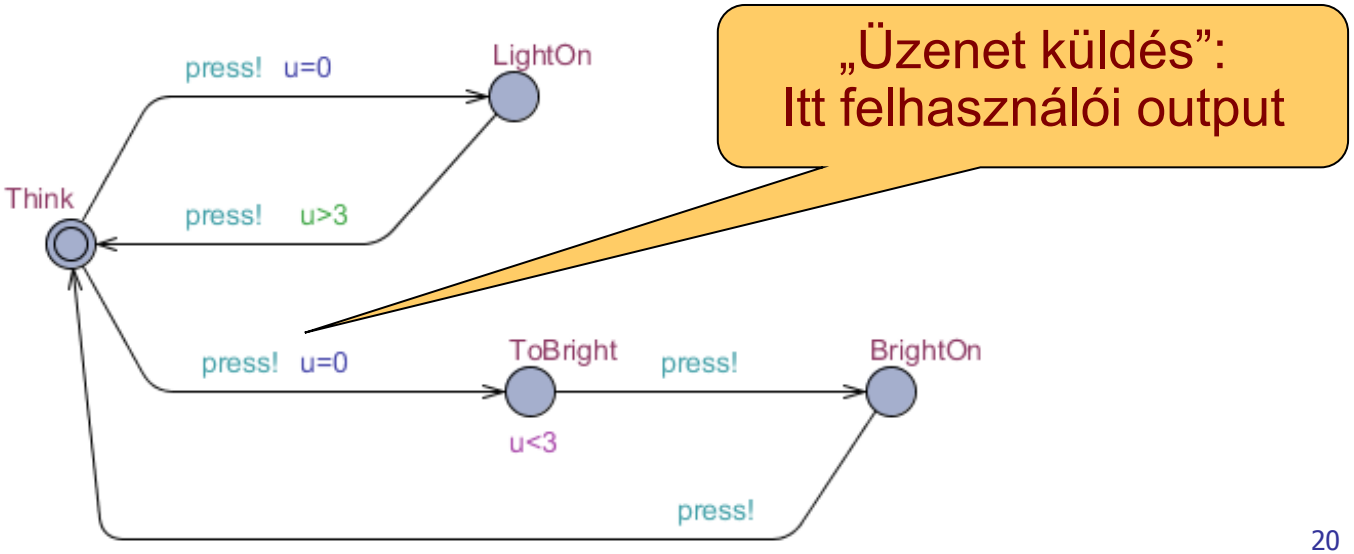
Deklarációk:

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

Kapcsoló:



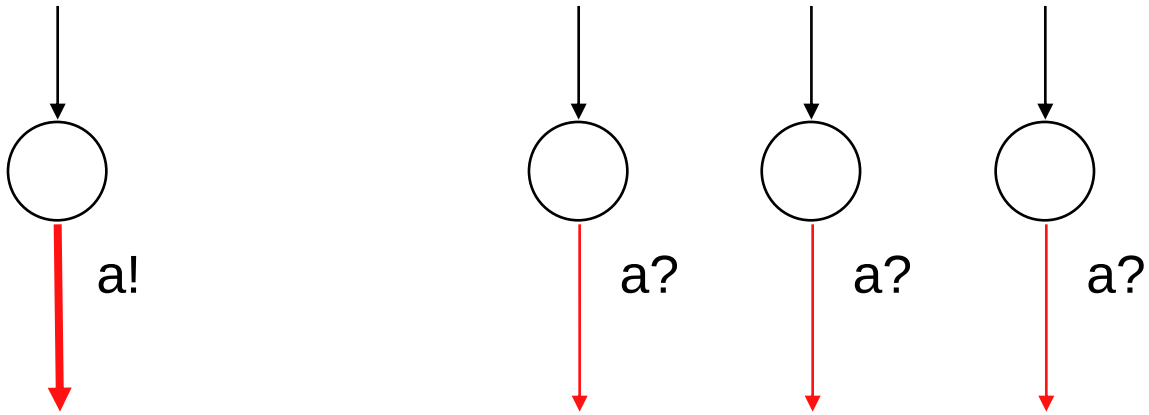
Felhasználó:



További lehetőségek: Broadcast csatorna

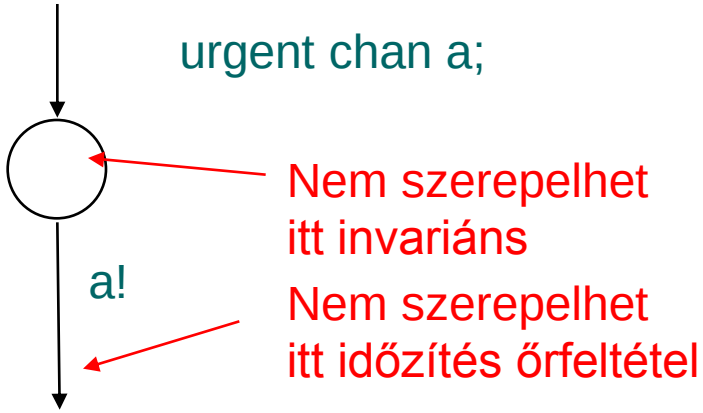
- Broadcast csatorna: 1->N kommunikáció
  - „Üzenetküldés” feltétel nélkül megtörténik
    - Nem kell fogadó készenlétére (randevúra) várni
  - Minden „üzenetfogadásra kész” partner erre szinkronizálódik
    - Üzenetfogadáshoz szükséges az üzenetküldés
  - Használati feltétel: Nem szerepelhet őrfeltétel a broadcast csatornára hivatkozó üzenetfogadó átmeneten

broadcast chan a;



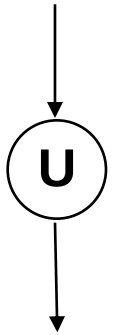
További lehetőségek: Urgent csatorna

- Urgent csatorna: Nem enged késleltetést
  - Késleltetés nélkül, azonnal végrehajtandó szinkronizáció (de előtte más átmenetek azonnali végrehajtása lehet)
  - Használati feltételek:
    - Nem szerepelhet időzítés őrfeltétel azon az átmeneten, ami ilyen csatornára hivatkozó akcióval van címkézve
    - Nem szerepelhet invariáns azon az állapoton, ahonnan olyan átmenet indul, ami ilyen csatornára hivatkozó akcióval van címkézve

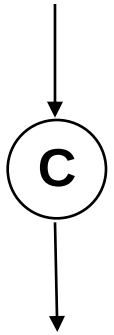


További lehetőségek: Speciális állapotok

- Urgent állapot: késleltetés korlátozása
  - Nem telhet idő az adott állapotban
  - Ekvivalens modell:
    - Óraváltozó bevezetése: clock x;
    - Minden bemenő élen resetelve: x:=0
    - Állapot invariáns hozzárendelése: x<=0

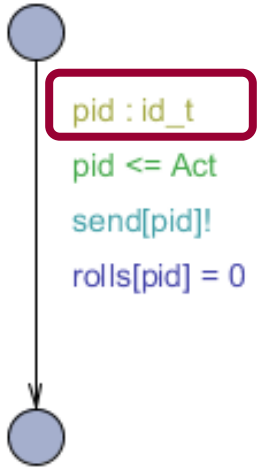


- Committed állapot: átmenetek egybefogása
  - Bemenő és kimenő átmenet egy atomi műveletként végrehajtva
  - A bemenő és kimenő átmenetek végrehajtása között más automata átmenete nem lehet végrehajtva

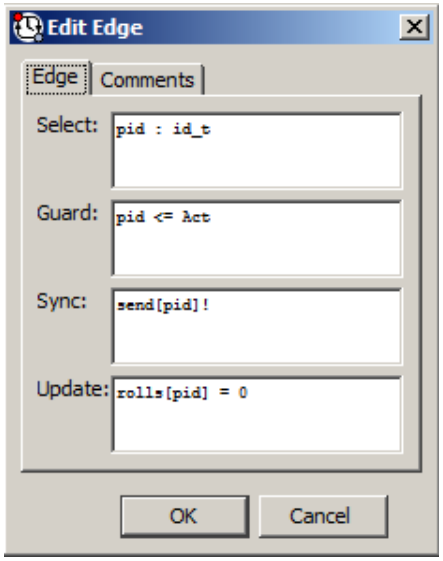


Véletlen választás és kiértékelés

- Véletlen választás modellezése
  - select konstrukció: egy változó adható meg, típussal
  - A változót véletlenszerűen adatértékhez köti, a változó megadott típusa szerinti tartományból
  - Az átmenethez tartozó szinkronizációban, őrfeltételben, akcióban használható a kötött változó
  - Minden lehetséges választást bejár az ellenőrzés során



- Az átmenethez rendelt kifejezések kiértékelése:
  - A Select választás köt először
  - A Guard őrfeltétel igaz kell legyen az átmenet engedélyezéséhez
  - A Sync szinkronizáció másik automatáéhoz köti az átmenet végrehajtását
  - Az Update akció az átmenet végrehajtása során következik be
  - Szinkronizáló élek esetén a küldő akciója a fogadóé előtt fut le

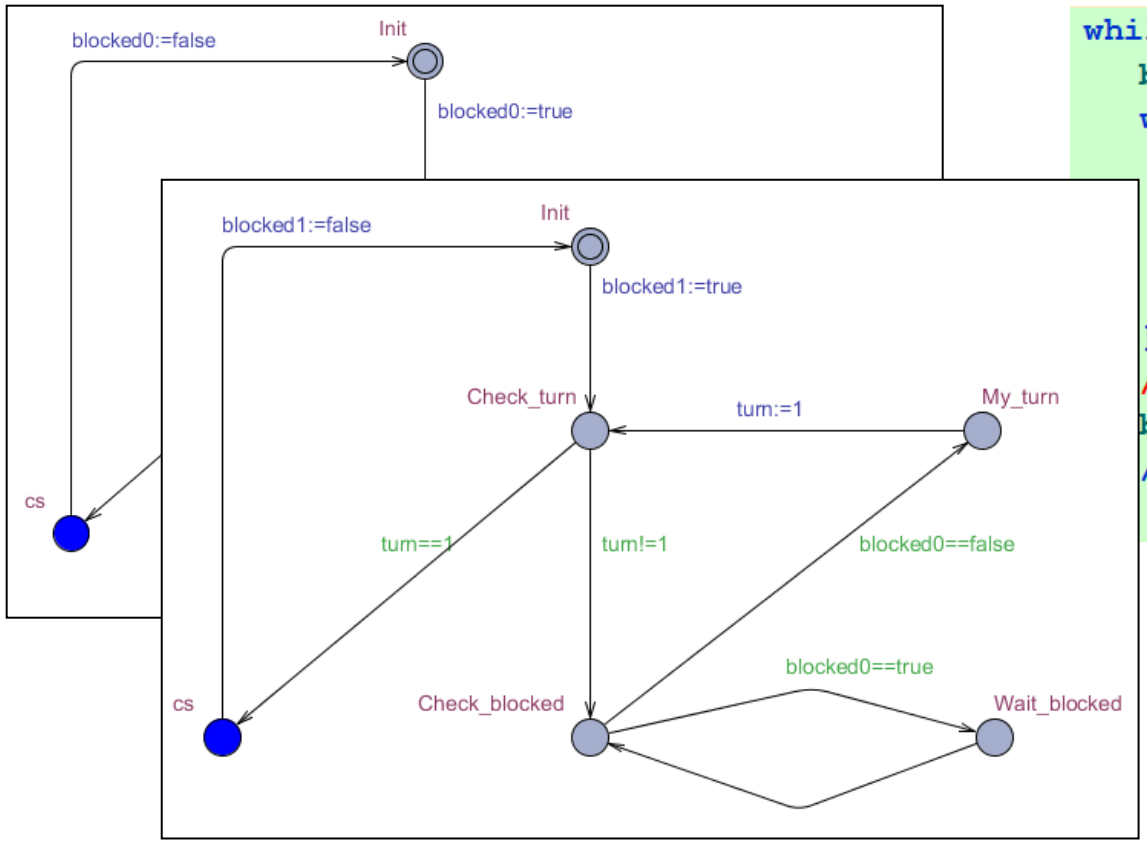


Automaták példányosítása: Motiváció

Deklarációk:

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

A P0 és P1 automata:



P0

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

P1

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

Automaták példányosítása

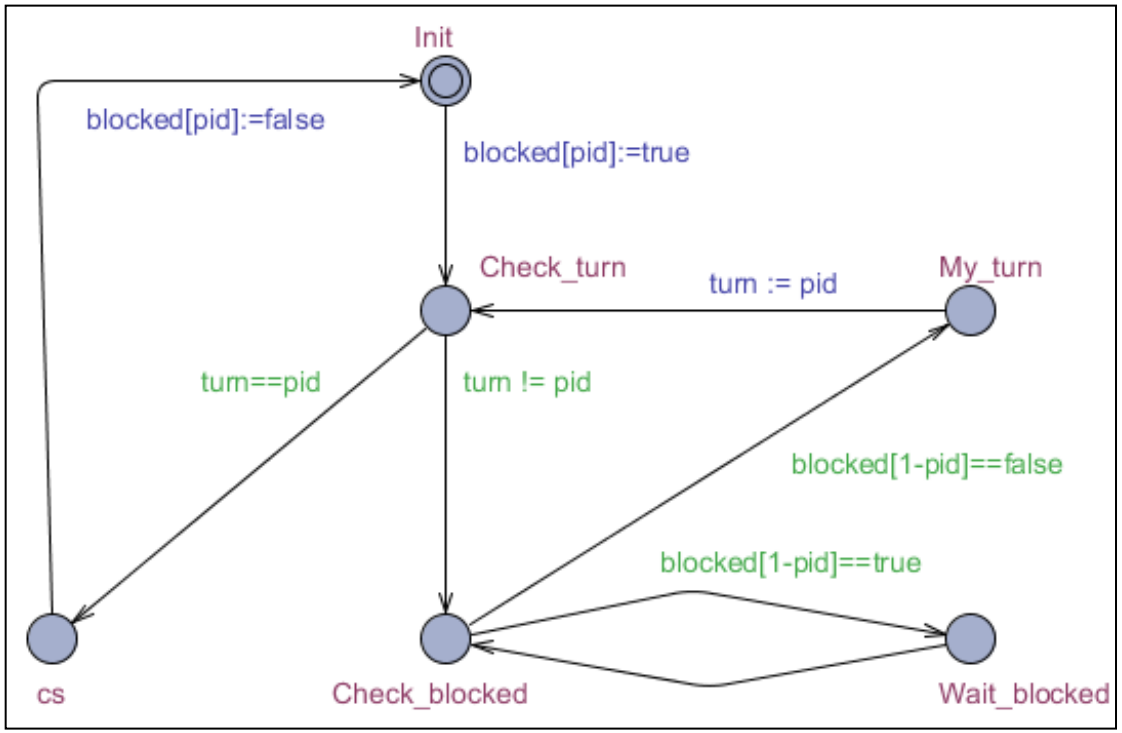
Deklarációk:

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

Kihasznált modellezési lehetőségek:

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

A P automata template pid paraméterrel:



P0 példány pid=0 mellett:

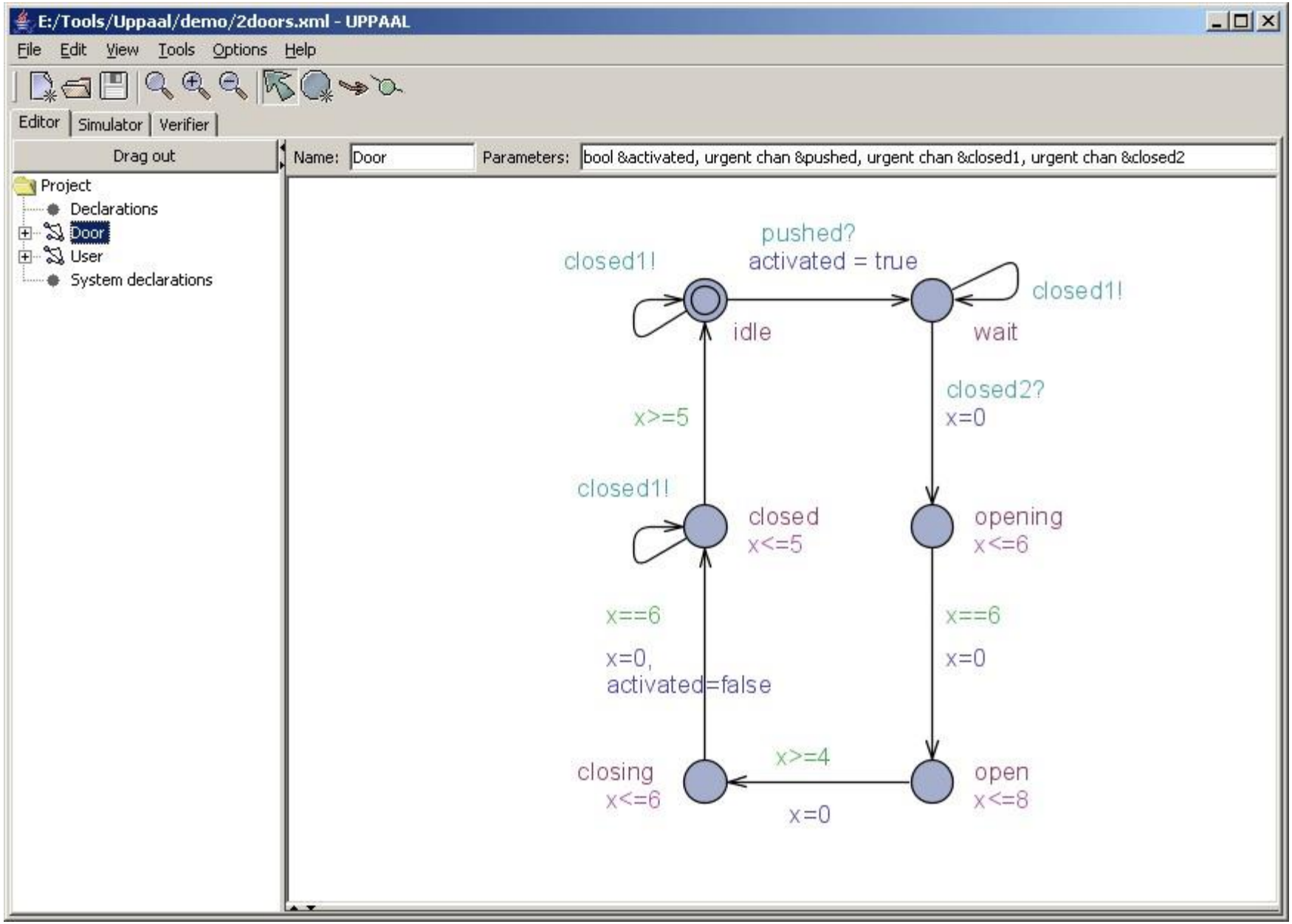
P0

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

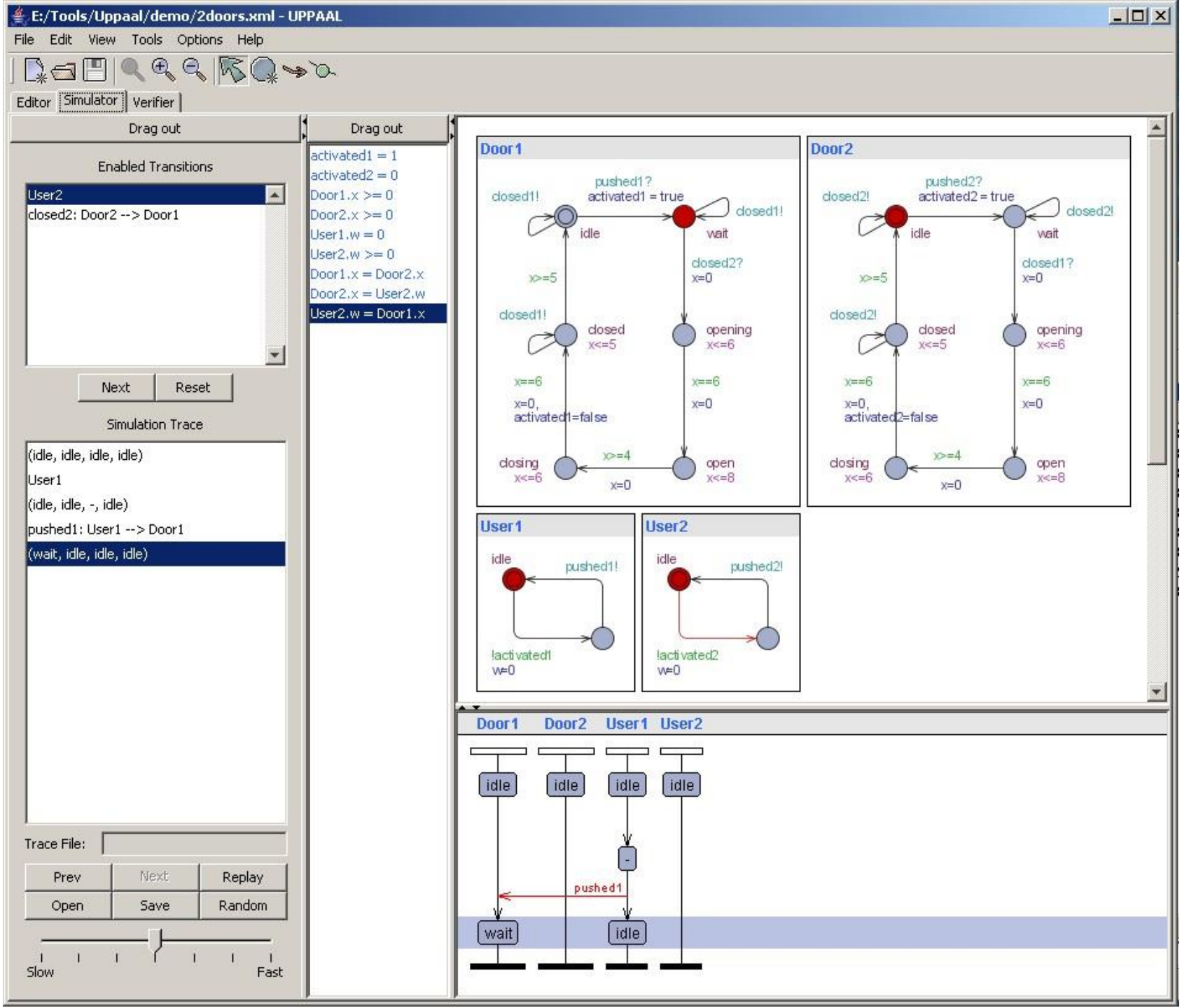
Az UPPAAL eszköz

- Fejlesztése (1999-):
  - Uppsala University, Svédország
  - Aalborg University, Dánia
- Akadémiai verzió (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

Verifikáció