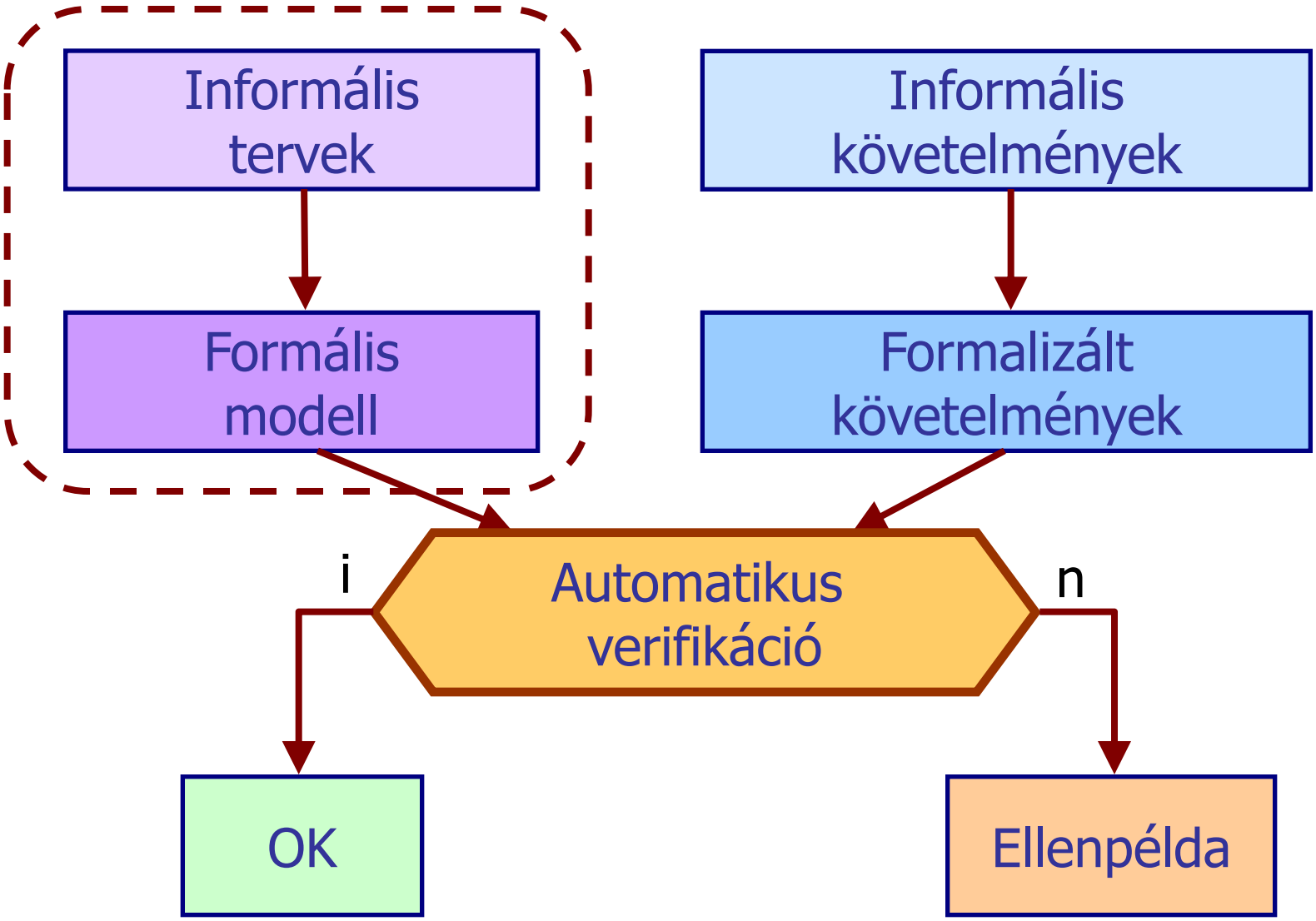
Alapszintű formalizmusok

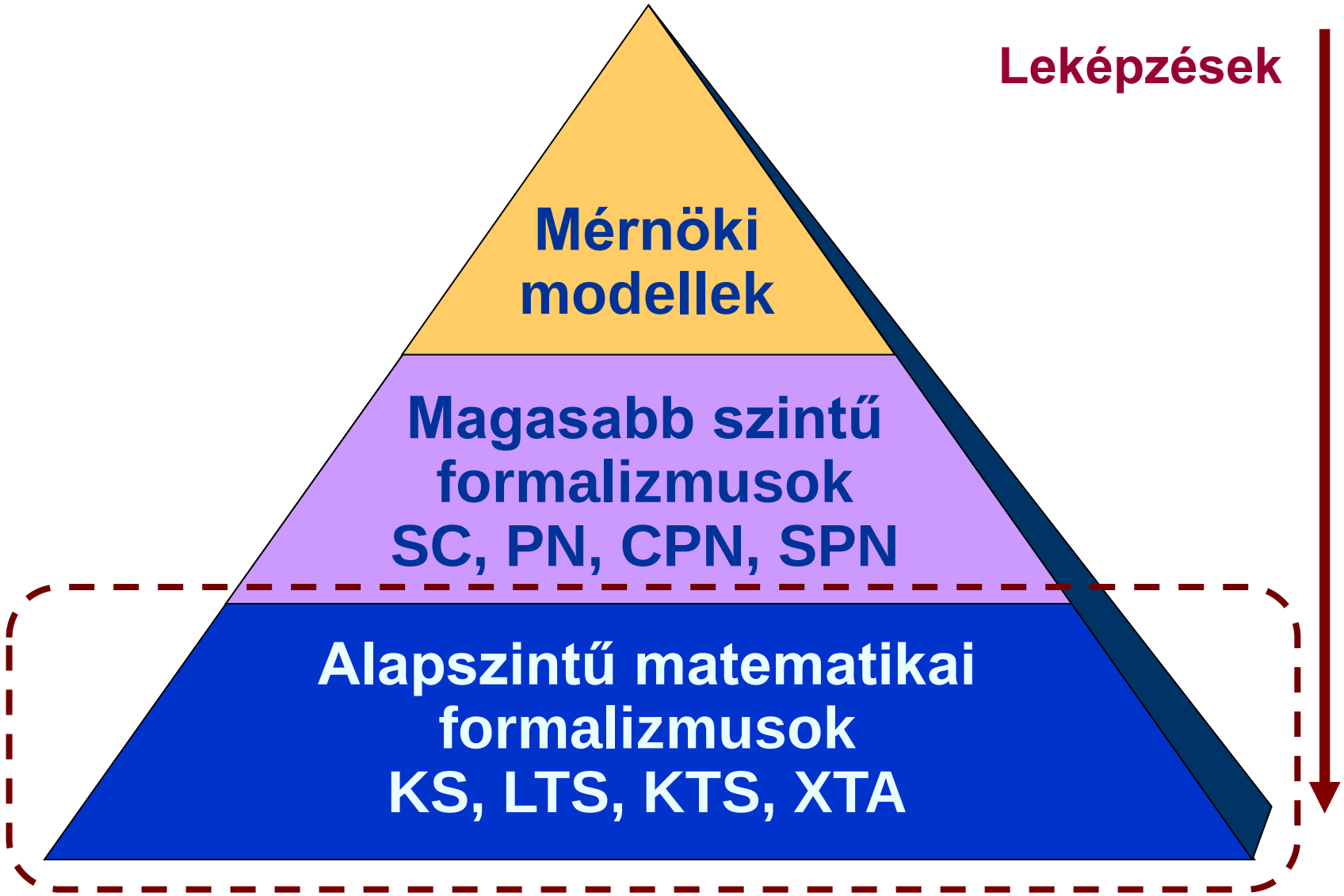
dr. Majzik István

BME Méréstechnika és Információs Rendszerek Tanszék

Mit szeretnénk elérni?



Modellek a formális ellenőrzéshez



Alapszintű formalizmusok (áttekintés)

- Kripke-struktúrák (KS)
  - Állapotok tulajdonságokkal (címkékkel)
  - Állapotátmenetek
- Címkézett tranzíciós rendszerek (LTS)
  - Állapotok
  - Állapotátmenetek tulajdonságokkal
- Kripke tranzíciós rendszerek (KTS)
  - Állapotok tulajdonságokkal
  - Állapotátmenetek tulajdonságokkal
- Kiterjesztett időzített automaták (XTA)
  - Automaták változókkal, óraváltozókkal

1. Kripke-struktúra

- KS (Kripke Structure) használat célja:
  - Viselkedés, algoritmus állapotainak leírása
  - Állapotok tulajdonságai: címkézés atomi kijelentésekkel (többel is)
- Szintaxis:

$KS = (S, R, L)$ és AP, ahol

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

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

$R \subseteq S \times S$: állapotátmeneti reláció

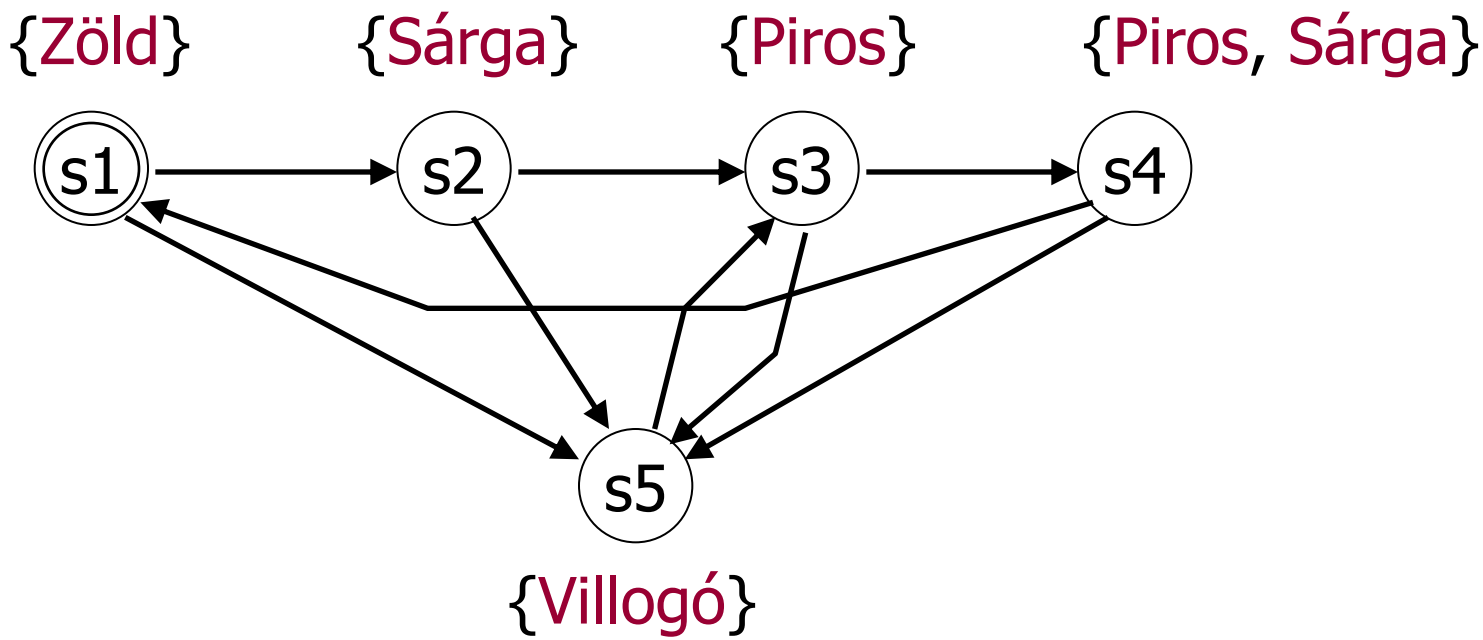
$L: S \to 2^{AP}$ állapotok címkézése atomi kijelentésekkel

- Szemantika:
  - Kezdetben a kezdőállapot aktív
  - Az aktív állapot változhat az állapotátmenetek mentén

Kripke-struktúra példa

Közlekedési lámpa viselkedése

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



2. Címkézett tranzíciós rendszer (LTS)

- LTS (Labeled Transition System) használat célja:
  - Viselkedés, kommunikáció állapotátmeneteinek leírása
  - Állapotátmenetek tulajdonságai: címkézés akciókkal (egyet eggyel)
- Szintaxis:

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

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

$Act = \{a,b,c,...\}$ akciók halmaza (domén-specifikus)

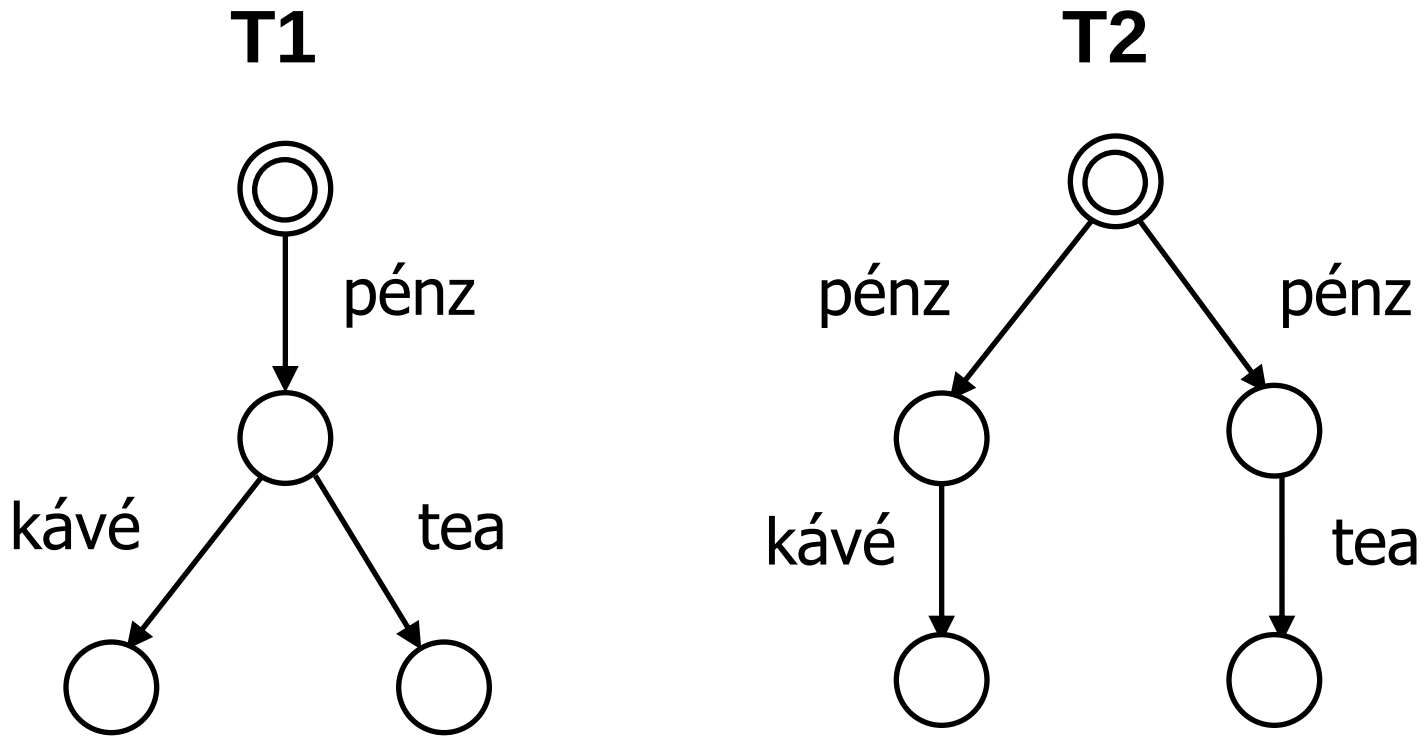
$\rightarrow \subseteq S \times Act \times S$ címkézett állapotátmenetek, pl. $s_1 \overset{a}{\rightarrow} s_2$

- Szemantika:
  - Kezdetben a kezdőállapot aktív
  - Az aktív állapot változhat az állapotátmenetek mentén

LTS példák

- Italautomata modelljei

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



3. Kripke tranzíciós rendszer (KTS)

- KTS (Kripke Transition System) használat célja:
  - Viselkedés, protokoll állapotainak és állapotátmeneteinek leírása
  - Állapotok tulajdonságai: címkézés atomi kijelentésekkel (többel is)
  - Állapotátmenetek tulajdonságai: címkézés akciókkal (egyet eggyel)
- Szintaxis:

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

$AP = \{P, Q, R, ...\}$ atomi kijelentések és $Act = \{a, b, c, ...\}$ akciók halmaza

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

$\rightarrow \subseteq S \times Act \times S$ állapotátmeneti reláció (akciókkal címkézve)

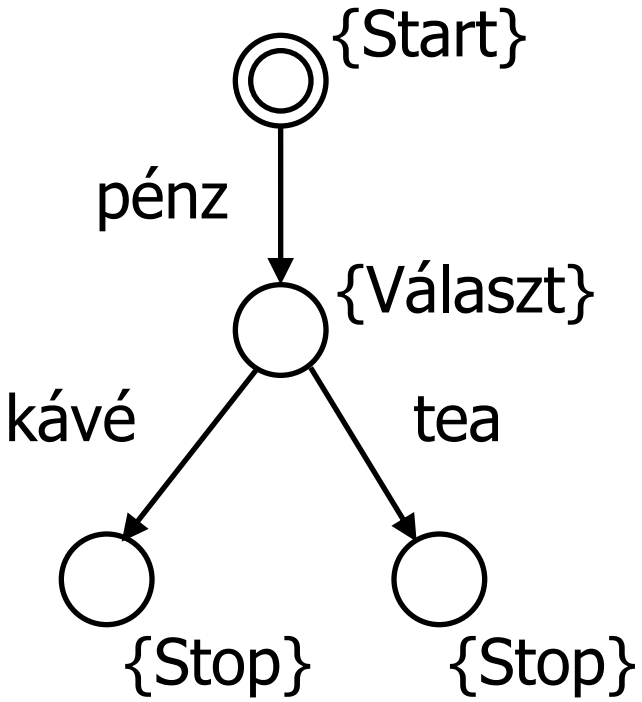
$L: S \rightarrow 2^{AP}$ állapotok címkézése atomi kijelentésekkel

Szemantika:

- Kezdetben a kezdőállapot aktív
- Az aktív állapot változhat az állapotátmenetek mentén

KTS példa

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



Kiterjesztett időzített automaták és használatuk az UPPAAL eszközben

Extended Timed Automata
Network of Timed Automata

Automaták és változók

- Cél: Állapot alapú viselkedés modellezése
- Alap formalizmus: Véges állapotú automata (FSM)
  - Állapotok (névvel hivatkozhatók)
  - Átmenetek
- Nyelvi kiterjesztés: Egész értékű változók használata
  - Változók deklarálhatók típussal (értéktartománnyal)
  - Konstansok definiálhatók
  - Egész aritmetika használható
  - Az FSM „állapot” itt csak „vezérlési hely” (változó is az állapot része)
- Változók használata állapotátmeneteken:
  - Őrfeltétel hozzárendelése: A változókon kiértékelhető predikátum
    - Az átmenet bekövetkezéséhez igaz kell legyen
  - Akció hozzárendelése: Értékadás változóknak
    - Az átmenet bekövetkezésekor végrehajtódik

Példa: Automata változókkal

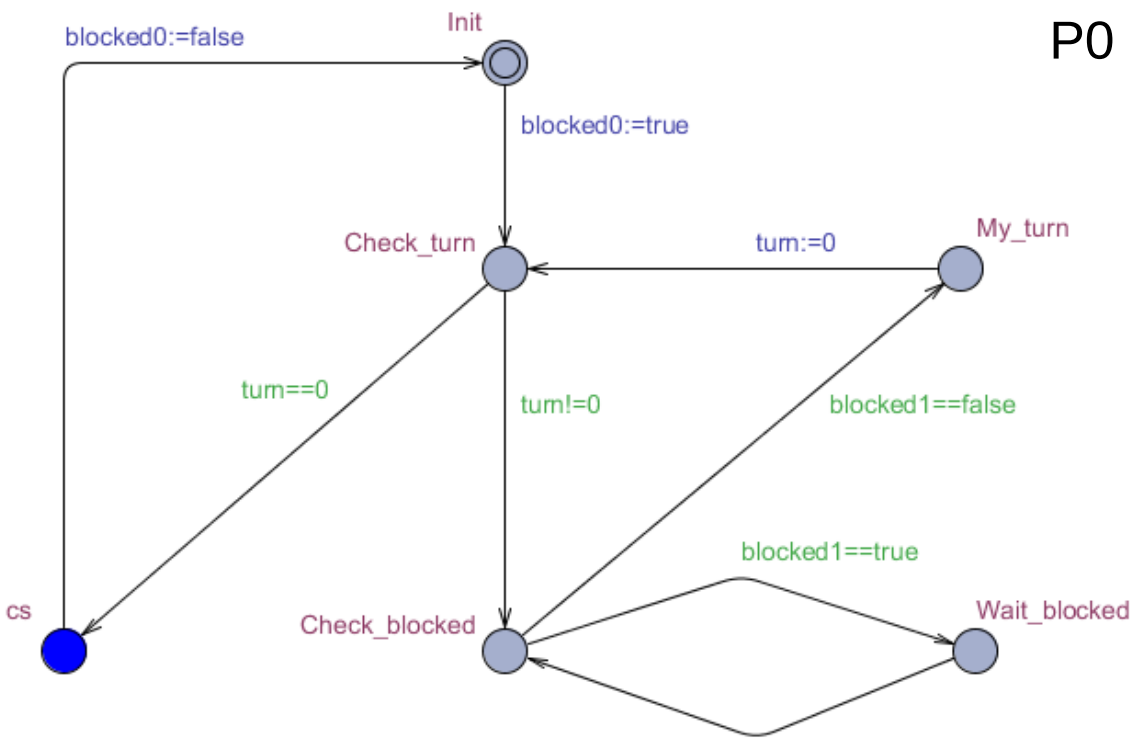
Deklarációk:

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

A P0 automata pszeudokódja és modellje:

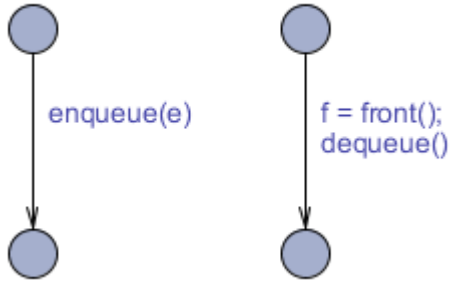
P0

```
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;
int list[N];
int[0,N-1] last;

// Put an element at the end of the list
// (overflow not checked)
void enqueue(int element)
{
        list[last++] := element;
}
```

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

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

Kiterjesztések óraváltozókkal

- Cél: Valósidejű viselkedés modellezése
  - Idő telik az állapotokban
  - Relatív időmérés (pl. timeout-hoz): Időzítő resetelése és leolvasása
  - Az idő függvényében változó a viselkedés
- Nyelvi kiterjesztés: Óraváltozók
  - Azonos rátával automatikusan „haladó” konkurens órák (időzítők)
- Óraváltozók használata átmeneteken:
  - Akciók: Óraváltozók nullázása (resetelés), egymástól függetlenül
  - Őrfeltételek: Óraváltozók és konstansok használhatók a predikátumokban
- Óraváltozók használata vezérlési helyeken:
  - Hely invariáns: Predikátum óraváltozókon és konstansokon; megadja, meddig állhat fenn az adott állapot

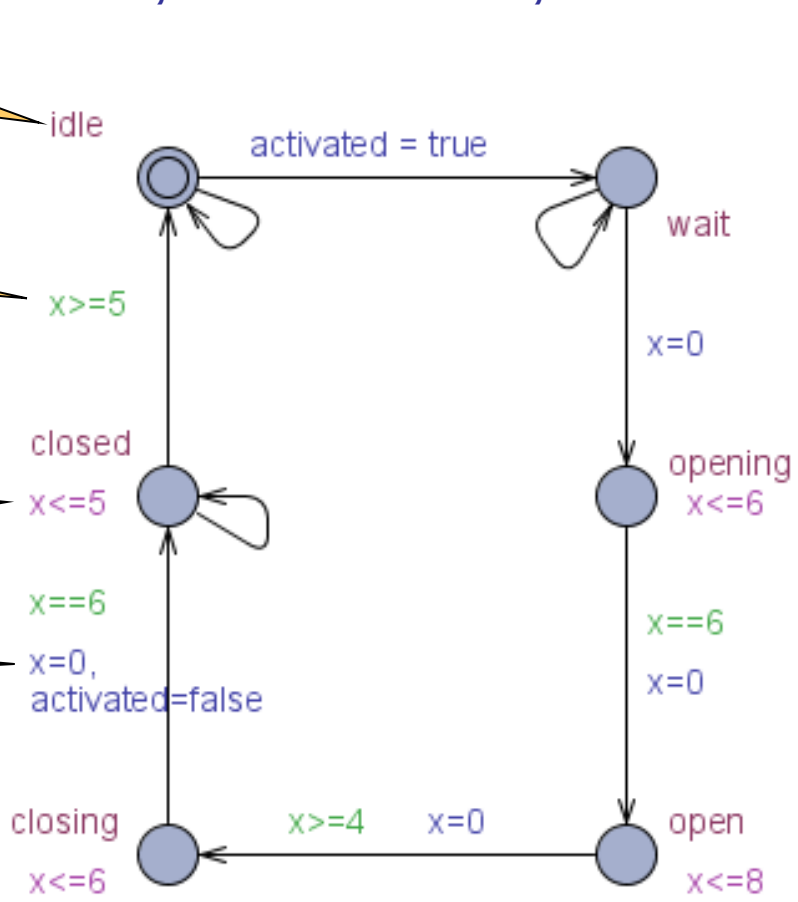
Példa: Időzített automata (az UPPAAL eszközben)

Vezérlési hely (névvel)

Őrfeltétel

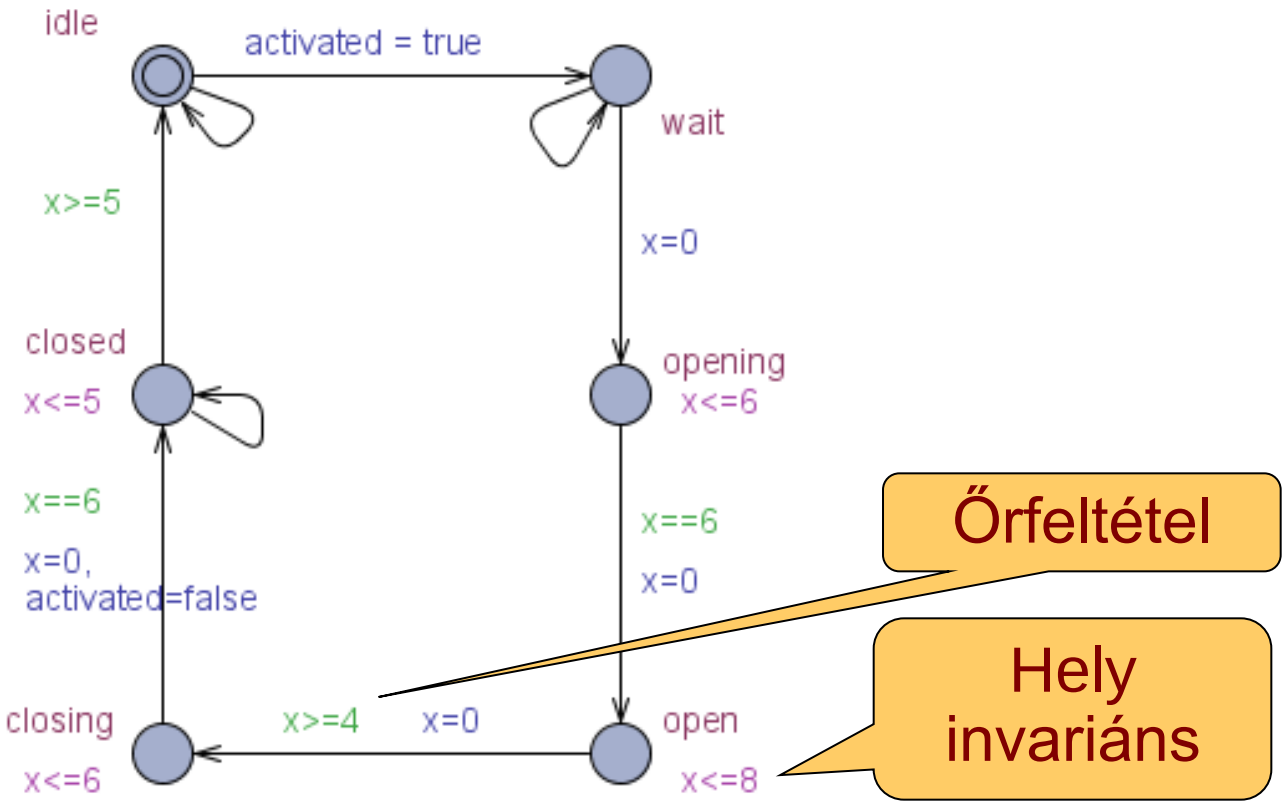
Hely invariáns

Akció



Az invariánsok és őrfeltételek szerepe

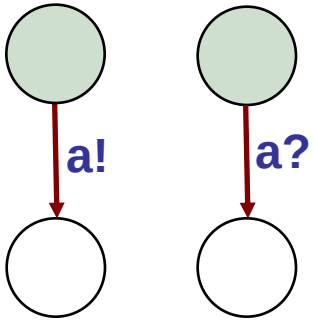
clock x;



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

Kiterjesztések elosztott rendszerekhez

- Cél: Együttműködő automaták hálózatának modellezése
  - Interakció: Szinkronizált, együttlépő átmenetek (randevú)
    - Szinkronizáció kezdeményezője: szinkron üzenet küldője (fogadóra vár)
    - Szinkronizáció résztvevője: szinkron üzenet fogadója
  - Ezzel az alapelemmel más jellegű interakciók is leírhatók
- Nyelvi kiterjesztés: Szinkronizált akciók
  - Csatornák definiálása (szinkron csatorna: chan)
  - Üzenetküldés: ! operátor a csatornára
    Üzenetfogadás: ? operátor a csatornára
    - Pl. az a nevű csatorna esetén a! és a? akciók



chan a;

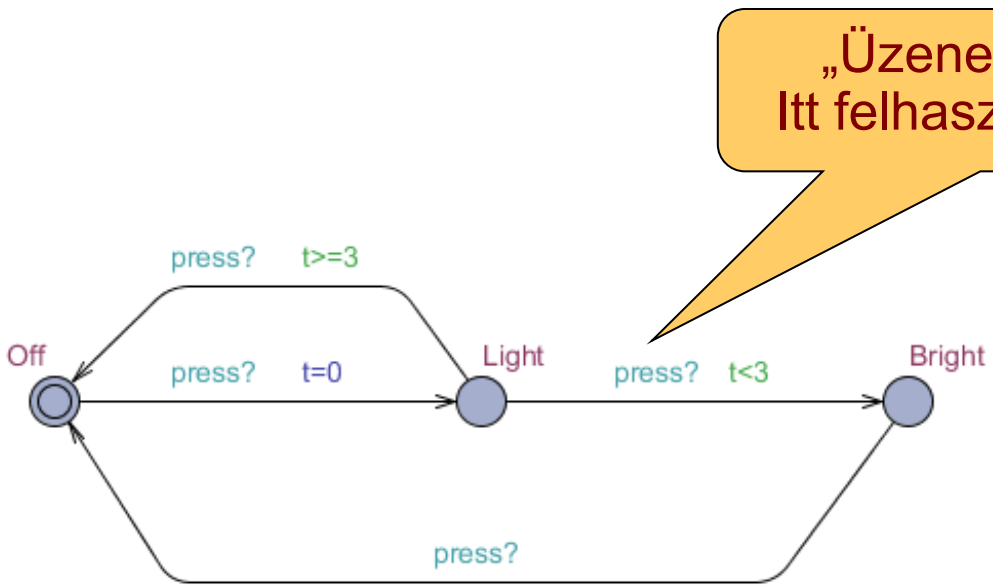
- Kiterjesztés: Csatornatömbök kezelése
  - Csatornatömb deklarálás: chan a[]
  - Műveletek csatornatömb elemein id változóval indexelve: a[id]! illetve a[id]? akciók

Példa óraváltozókra és szinkronizálásra

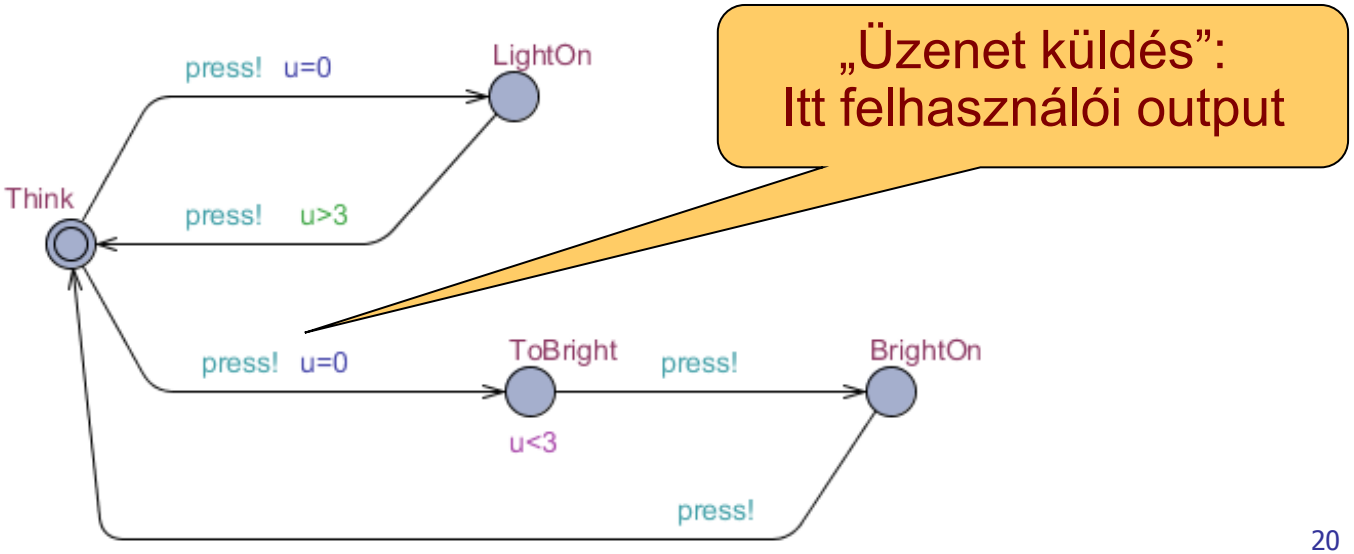
Deklarációk:

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

Kapcsoló:



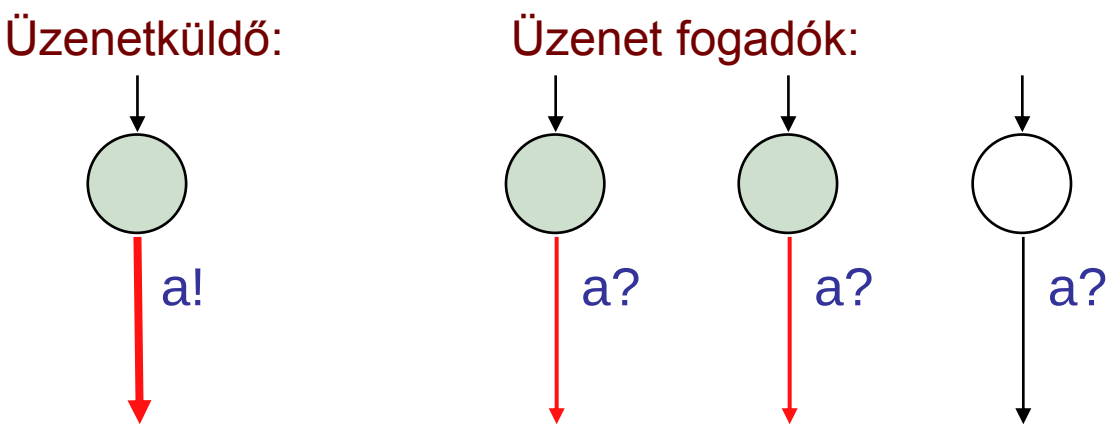
Felhasználó:



További lehetőségek: Broadcast csatorna

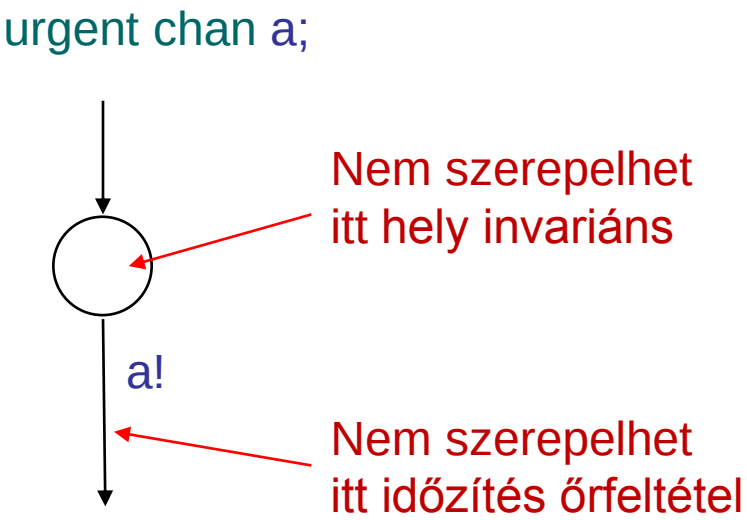
- Broadcast csatorna: 1→N kommunikáció
  - „Üzenetküldés” feltétel nélkül megtörténik
    - Nem kell fogadó készenlétére (randevúra) várni
  - Minden „üzenetfogadásra kész” partner erre szinkronizálódik
    - Üzenetfogadáshoz szükséges az üzenetküldés
  - Használati feltétel: Nem szerepelhet őrfeltétel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

broadcast chan a;



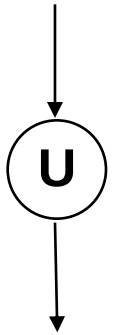
További lehetőségek: Urgent csatorna

- Urgent csatorna: Nem enged késleltetést
  - Késleltetés nélkül, azonnal végrehajtandó szinkronizáció
    - Előtte más átmenetek azonnali végrehajtása történhet
  - Korlátozott az őrfeltételek és invariánsok használata:

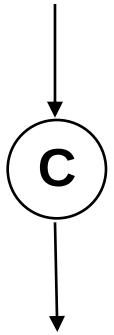


További lehetőségek: Speciális állapotok

- Urgent állapot: késleltetés korlátozása
  - Nem telhet idő az adott állapotban
  - Ekvivalens modell:
    - Óraváltozó bevezetése: clock x;
    - Minden bemenő élen resetelve: x:=0
    - Hely 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 normál átmenete nem lehet végrehajtva
    (legfeljebb committed állapotból induló)



Adatértékek véletlen választása: szelekció

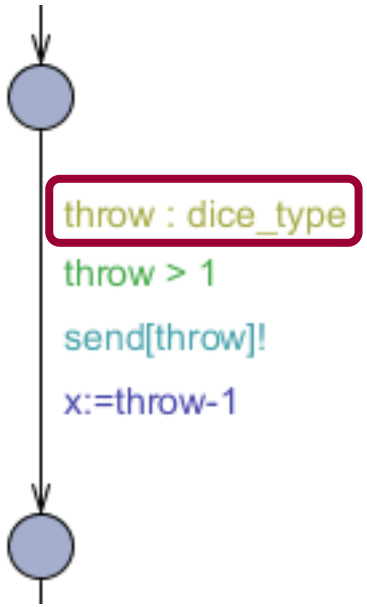
- Véletlen adatválasztás modellezése: select
  - Változó megadása típussal

    Pl. deklaráció:

    typedef int[1,6] dice_type;

    select konstrukció az átmeneten egy throw változóval:

    throw: dice_type



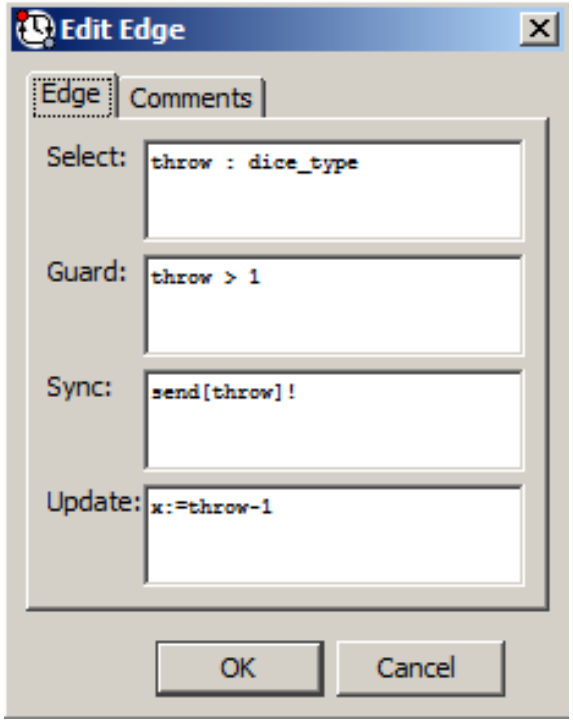
  - A változót véletlenszerűen adatértékhez köti a változó típusa szerinti tartományból

    Pl. a throw változó értéke 1 és 6 között lesz

  - A kötött változó ezután az átmenethez tartozó szinkronizációban, őrfeltételben, akcióban használható
  - A modell ellenőrzése: Minden lehetséges választást bejár

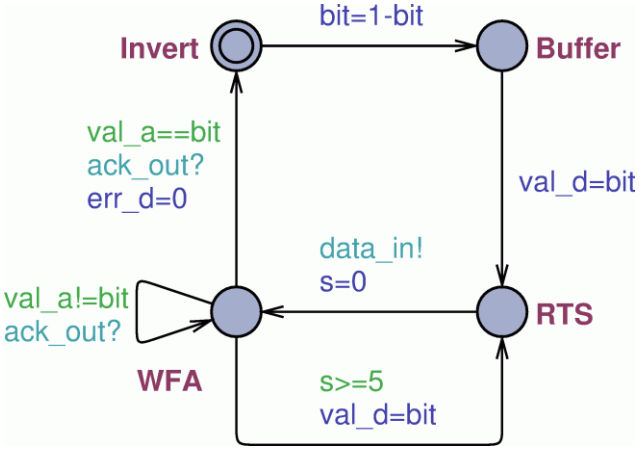
A kiértékelés és végrehajtás sorrendje

- 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 átmenetek esetén a küldő akciója a fogadóé előtt fut le



Formális szintaxis egy automata esetén

TA=(N, n_0, E, Inv, L) és C, V, Act, A



- Vezérlési helyek halmaza: N
- Kezdő vezérlési hely: n_0
- Élek őrfeltételekkel, akciókkal és nullázott órákkal:

  $E \subseteq N \times G(C,V) \times Act \times 2^C \times N$

  itt órák: C, változók: V, akciók: Act

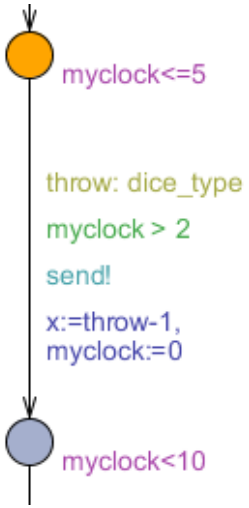
  G(C,V) predikátumok órákon vagy változókon
- Hely invariánsok:

  Inv: $N \rightarrow G(C)$ predikátumok órákon
- Vezérlési helyek nevei:

  L: $N \rightarrow A$ itt A névhalmaz

Szemantika egy automata esetén

- Kezdeti állapot:
  - Kezdő vezérlési hely aktív, minden óra 0, változók inicializálva
- Aktív állapotban: késleltetés vagy átmenet végrehajtás
- Késleltetés: Az órák értéke azonosan növekszik
  - Addig telhet egy adott vezérlési helyen az idő, amíg a hely invariáns teljesül
- Átmenet végrehajtása:
  - Átmenet engedélyezett (adott szelekció mellett), ha
    - Kiindulási helye aktív
    - Őrfeltétel teljesül
    - Szinkronizáció bekövetkezhet
    - Órák nullázása teljesíti a cél helyének invariánsát
  - Engedélyezett átmenet végrehajtódik (ha több van: véletlen választás)
    - Szinkronizáció és akció megtörténik, a nullázott órák értéke 0 lesz (küldő akciója a fogadóé előtt fut le)
    - Az átmenet célhelye válik aktívvá



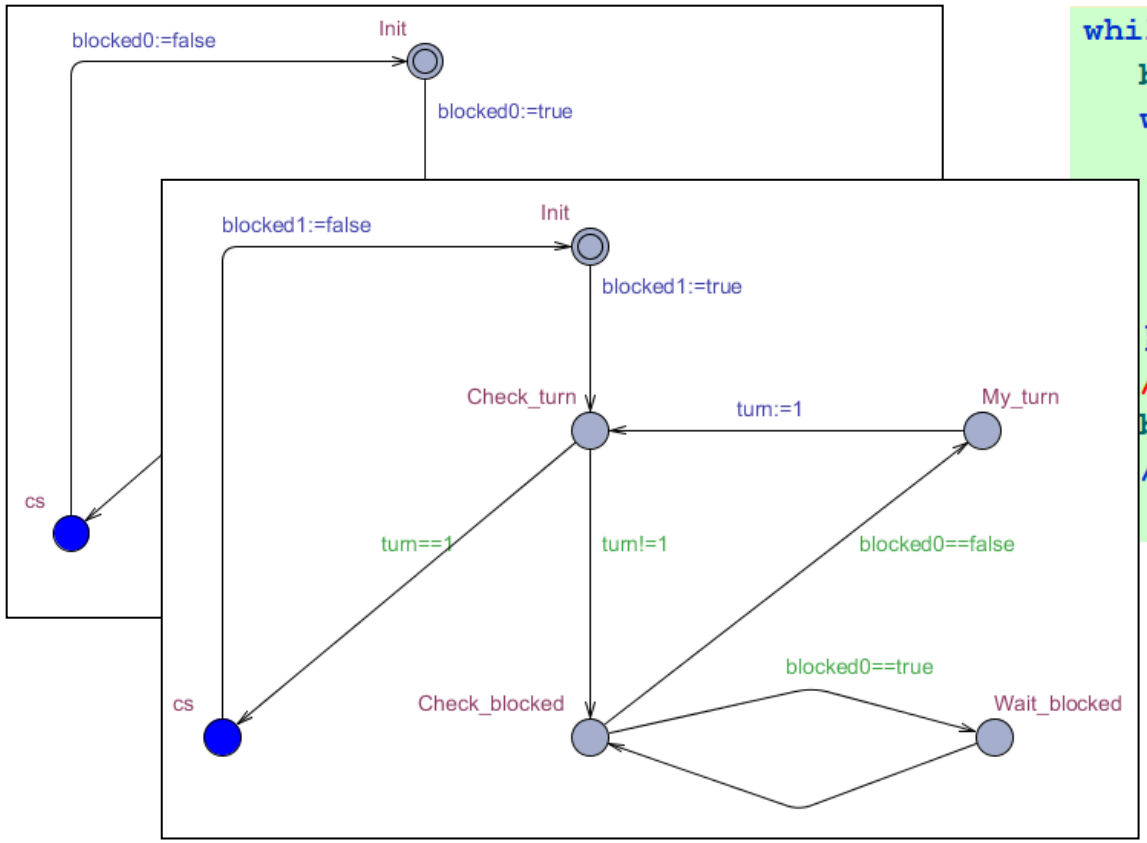

```
typedef int[1,6] dice_type;
int x=0;
chan send;
clock myclock;
```

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

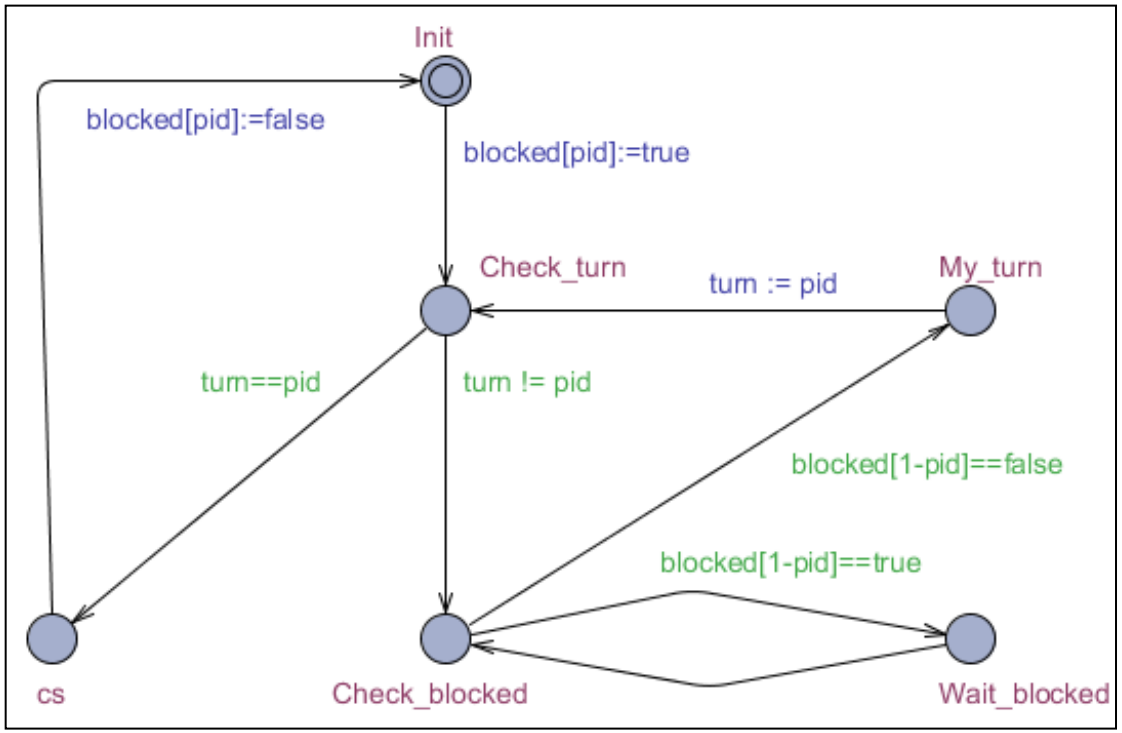
Deklarációk:

```
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:

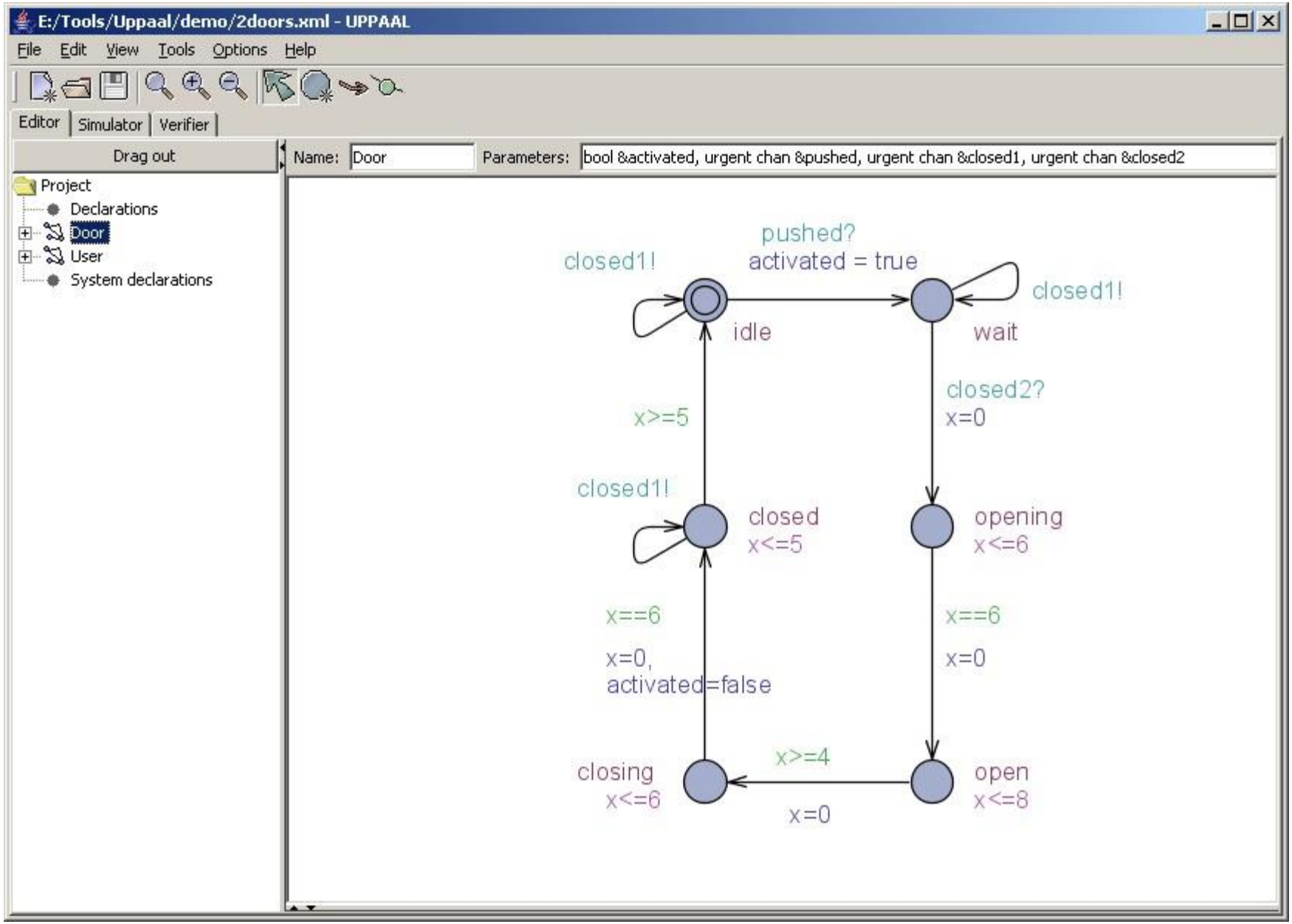
P0

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

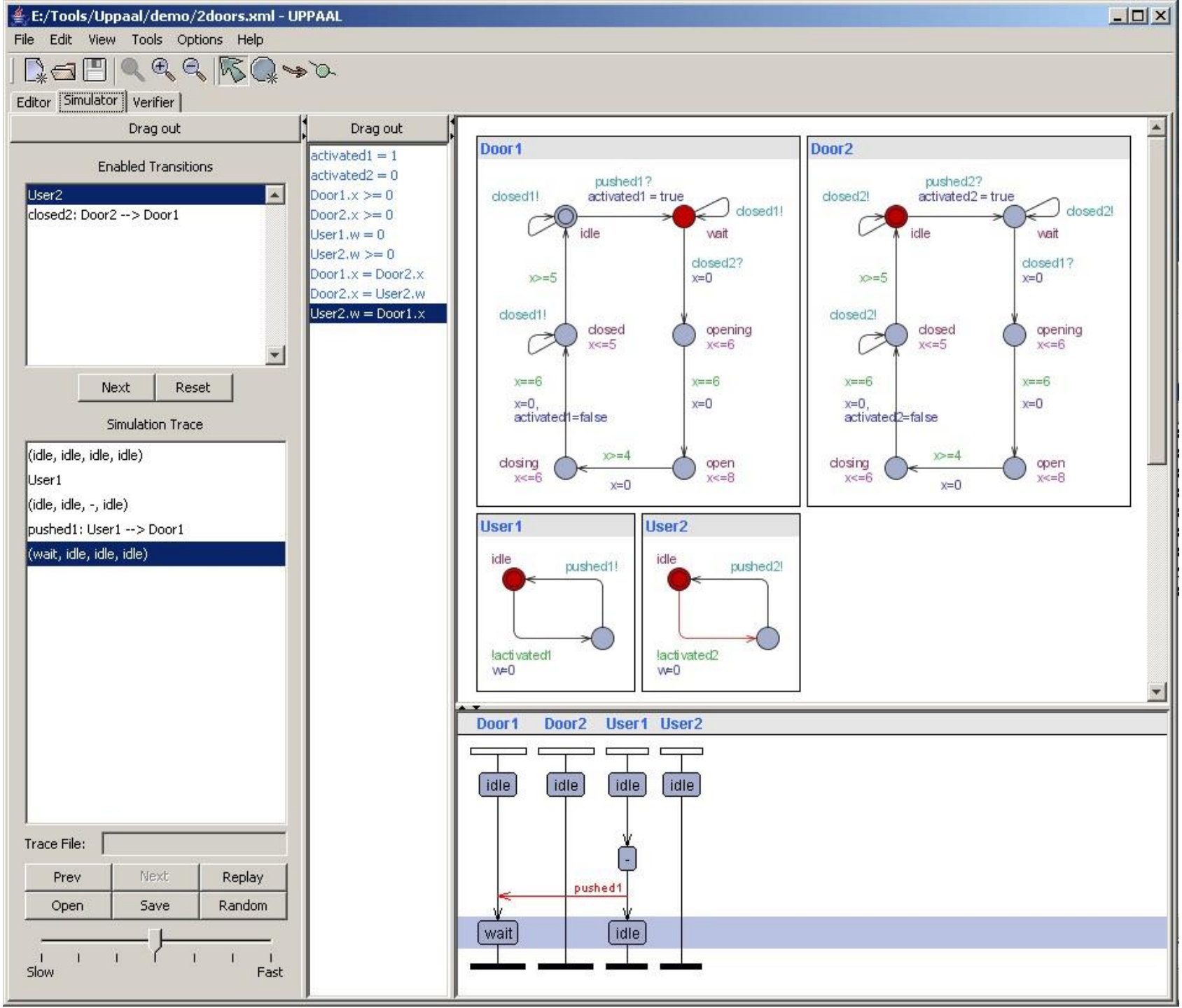
Az UPPAAL eszköz

- 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 integráció
  - …
- Kereskedelmi verzió:
  http://www.uppaal.com/

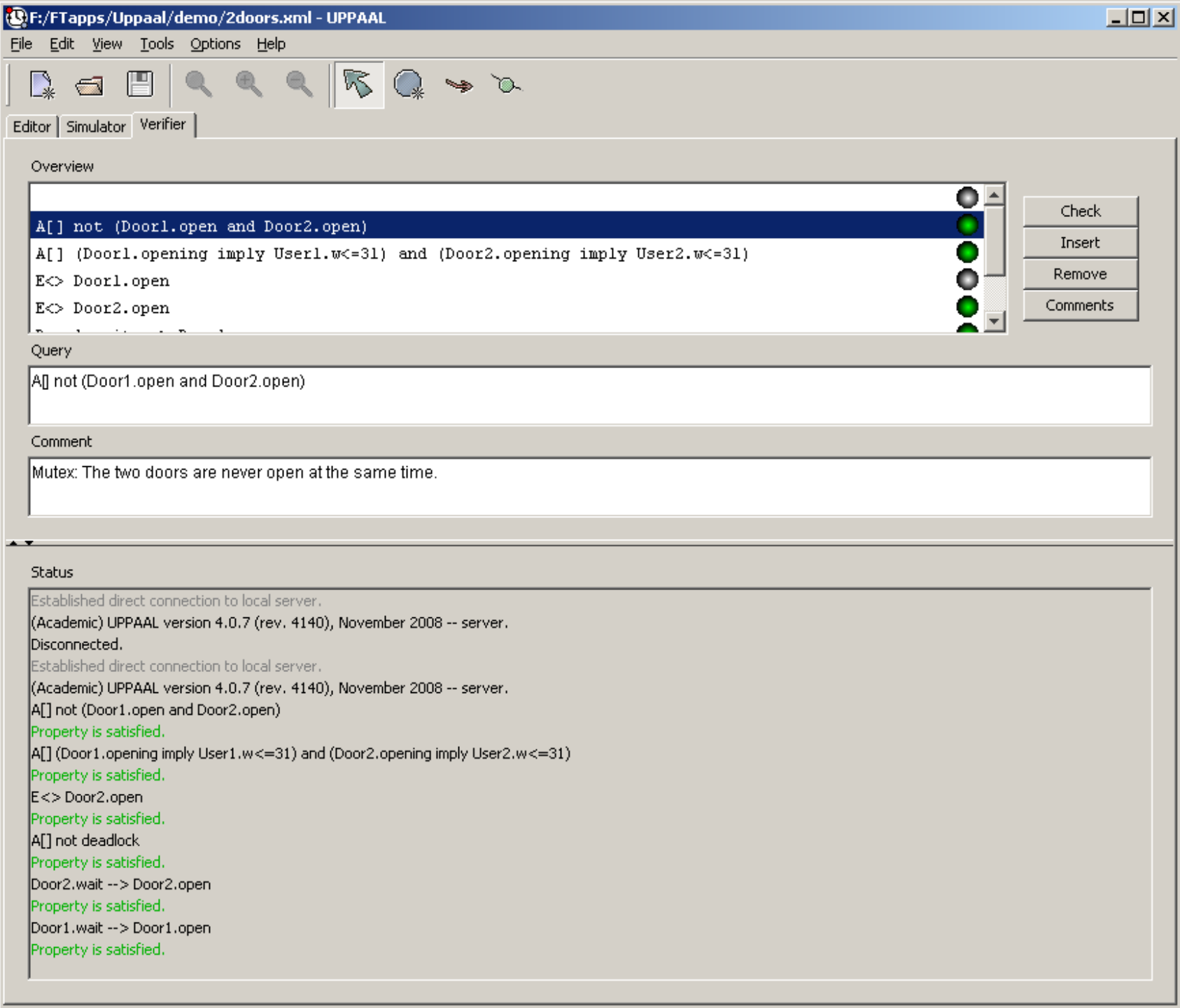
Automata modell



Szimulátor

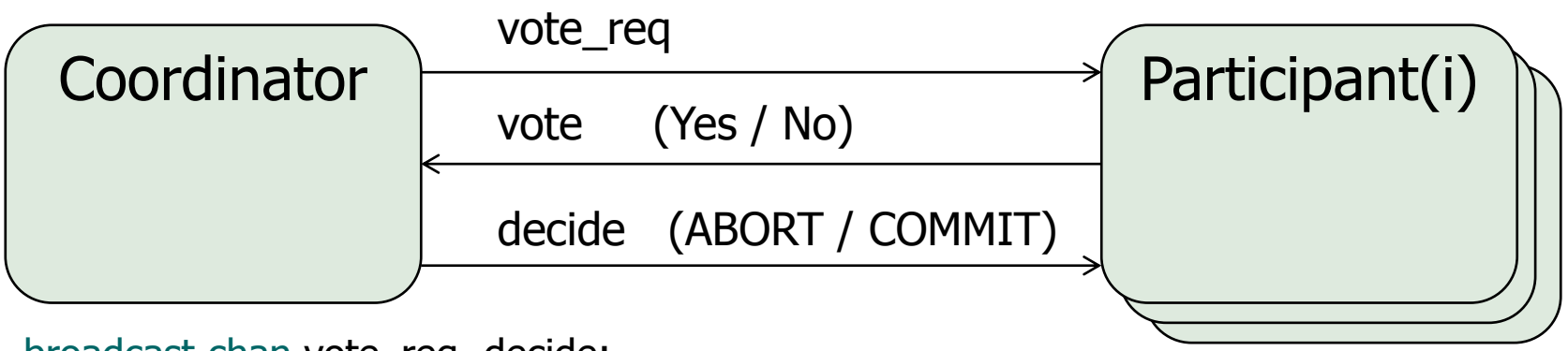

E:/Tools/Uppaal/demo/2doors.xml - UPPAAL
File Edit View Tools Options Help
Editor Simulator Verifier
Drag out
Enabled Transitions
User2
closed2: Door2 --> Door1
Next Reset
Simulation Trace
(idle, idle, idle, idle)
User1
(idle, idle, -, idle)
pushed1: User1 --> Door1
(wait, idle, idle, idle)
Trace File:
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



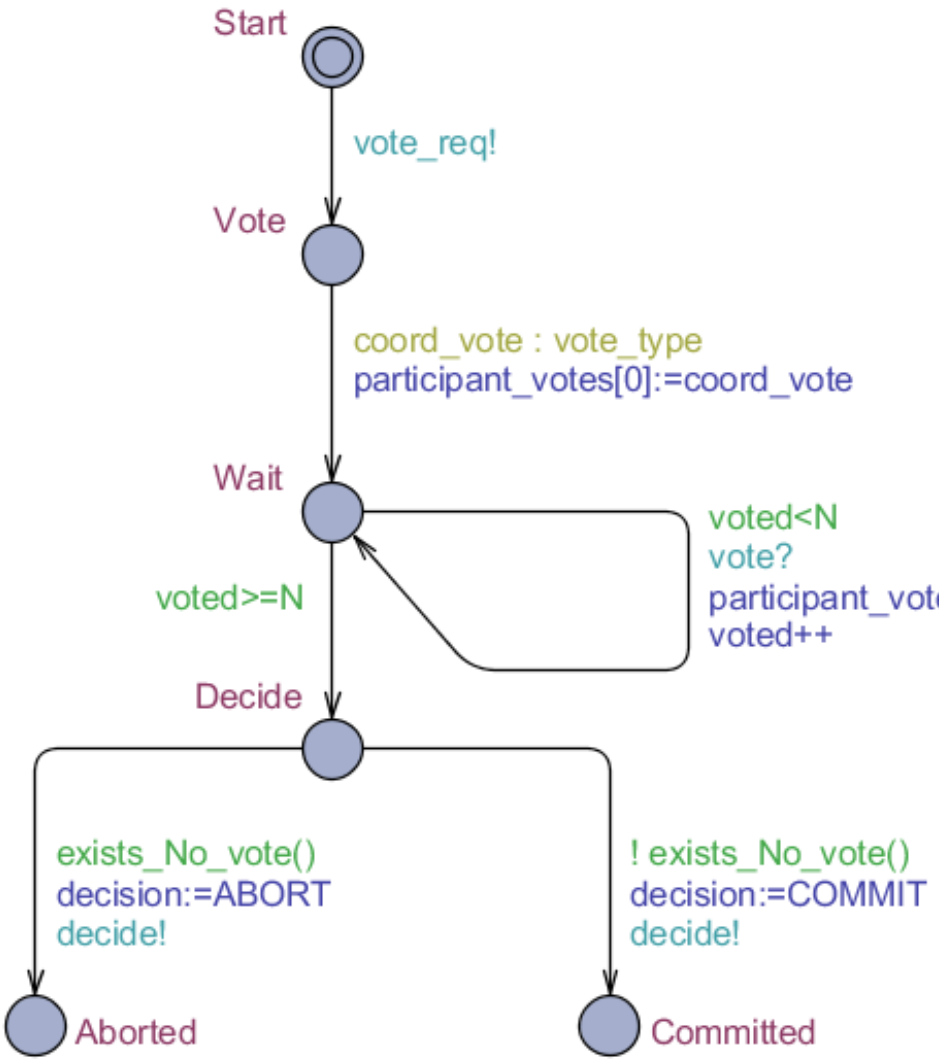
Mintapélda:
A kétfázisú commit protokoll

A modell struktúrája



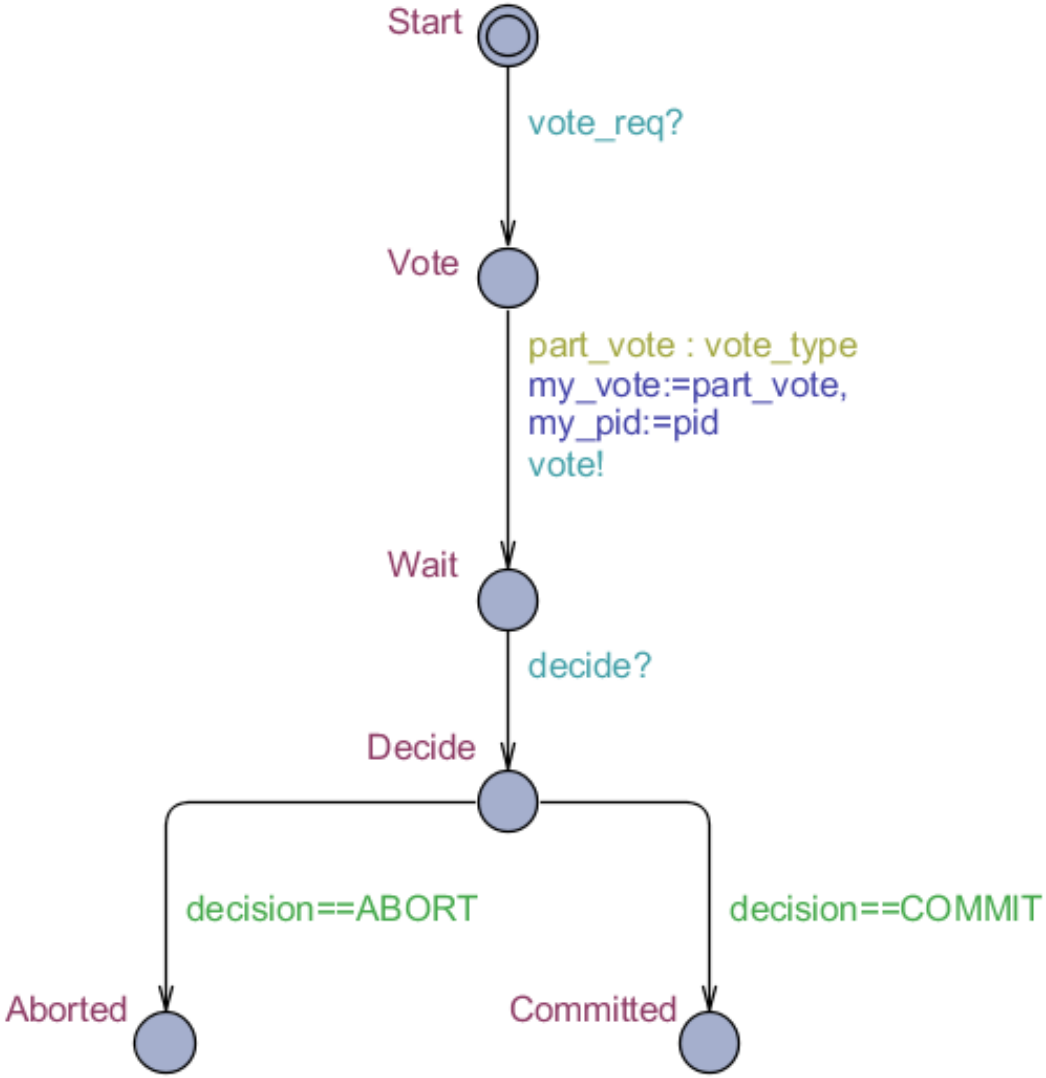

```
broadcast chan vote_req, decide;
chan vote;                                  // Csatornák

const int N=2;                              // Résztvevők száma
typedef int[1,N] pid_type;                  // Résztvevők azonosítóinak típusa
typedef int[0,1] vote_type;                 // Szavazatok típusa (0: No, 1: Yes)

int decision = 0;                           // A koordinátor döntése
const int ABORT  = 0;
const int COMMIT = 1;

vote_type my_vote;                          // Szavazat küldve a résztvevőtől
int my_pid;                                 // Azonosító küldve a résztvevőtől

int voted = 0;                              // A szavazott résztvevők száma
vote_type participant_votes[N+1];           // A koordinátor és résztvevők szavazatai
```

A koordinátor




```
bool exists_No_vote() {
   int[0,N+1] i;
   for (i=0; i<N+1; i++) {
      if (participant_votes[i]==0) {
         return true;
      }
   }
   return false;
}
```

A résztvevő


Start
vote_req?
Vote
part_vote : vote_type
my_vote:=part_vote,
my_pid:=pid
vote!
Wait
decide?
Decide
decision==ABORT
decision==COMMIT
Aborted
Committed

Ellenőrizendő tulajdonságok (példák)

A[] logikai operátor: Minden végrehajtás minden állapotára

- Ha a koordinátor Aborted, akkor volt No szavazat:

  A[] (Coordinator.Aborted imply
  exists (i : int[0,N]) Coordinator.participant_votes[i]==0)

- Ha a koordinátor Committed, minden szavazat Yes volt:

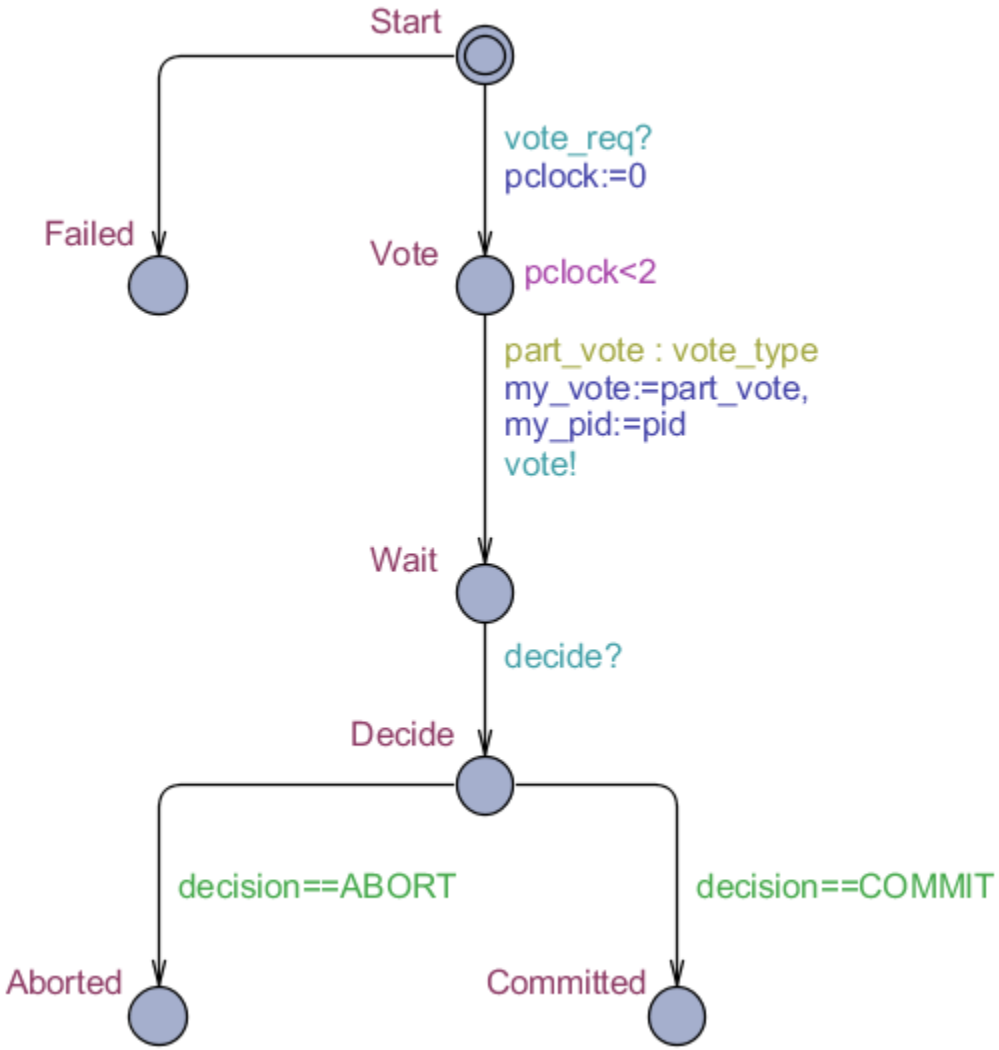
  A[] (Coordinator.Committed imply
  forall (i : int[0,N]) Coordinator.participant_votes[i]==1)

- Nincs olyan helyzet, hogy eltérő a döntés:

  A[] not (Coordinator.Aborted and
  exists (i : int[1,N]) Participant(i).Committed)

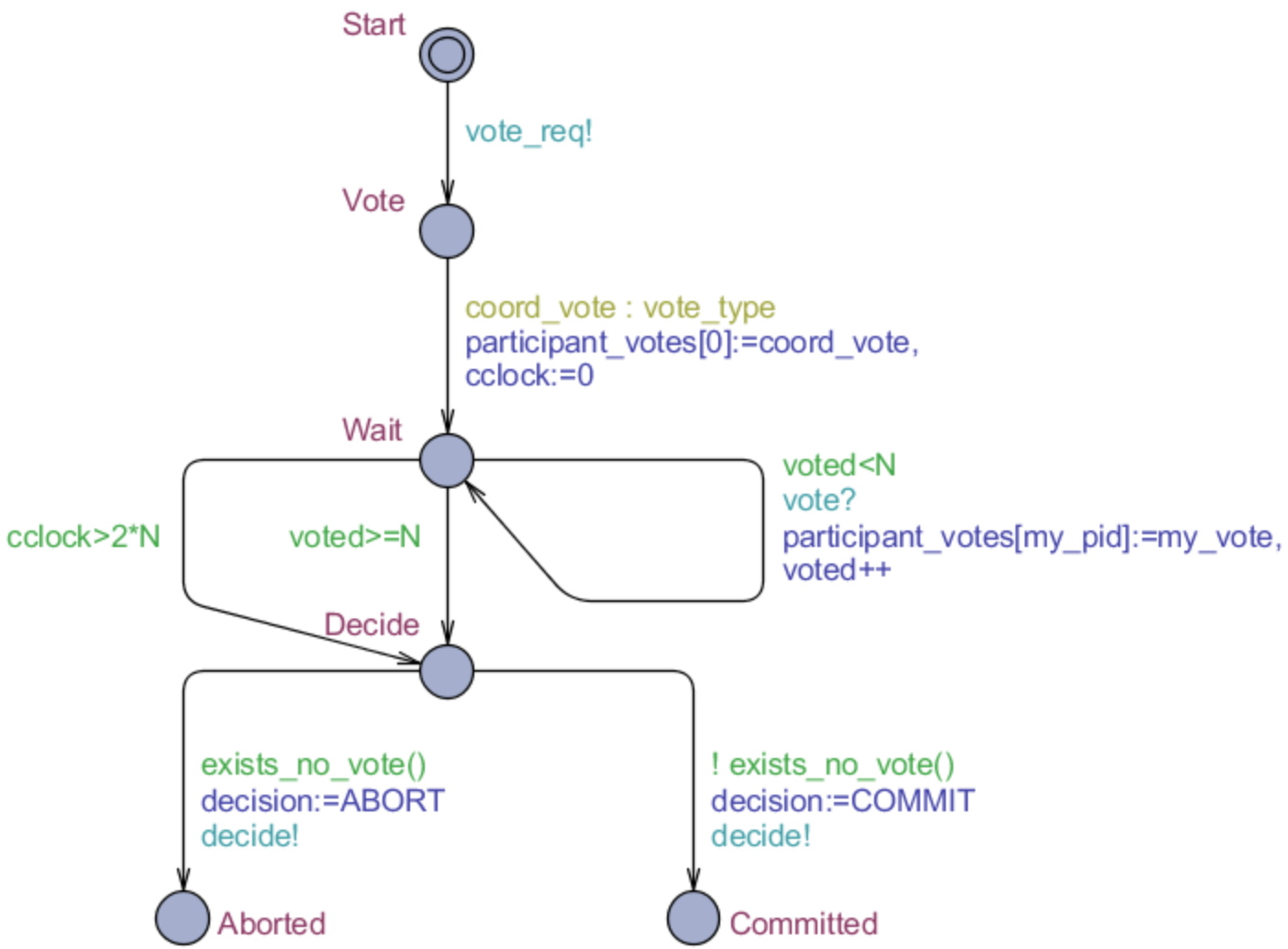
  A[] not (Coordinator.Committed and
  exists (i : int[1,N]) Participant(i).Aborted)

Résztvevő: Kiesés és időkorlát a szavazásra




```
clock pclock;          // Résztvevő lokális órája
```

Koordinátor: Timeout a szavazáskor



clock cclock; // Koordinátor lokális órája