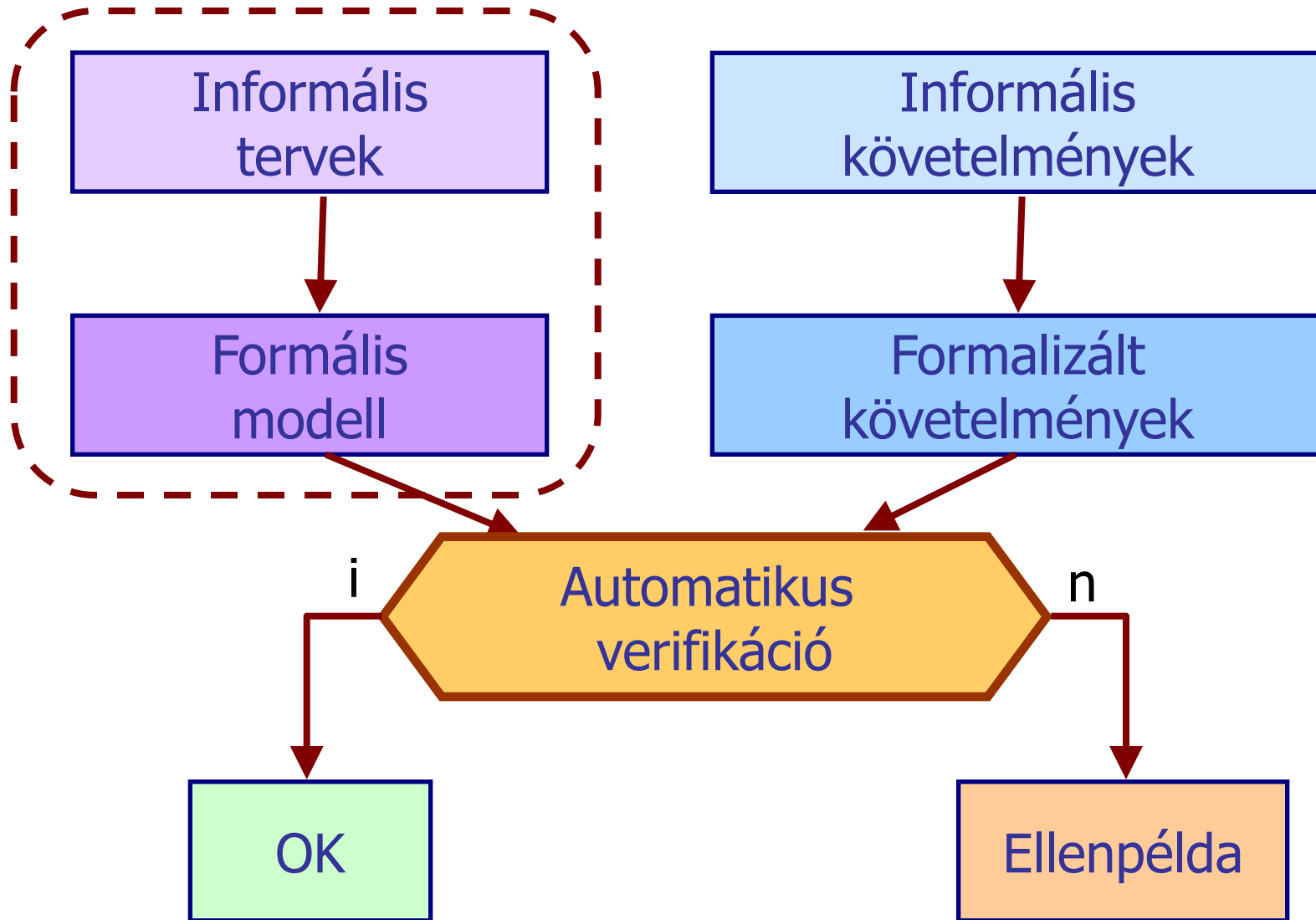


Alapszintű formalizmusok

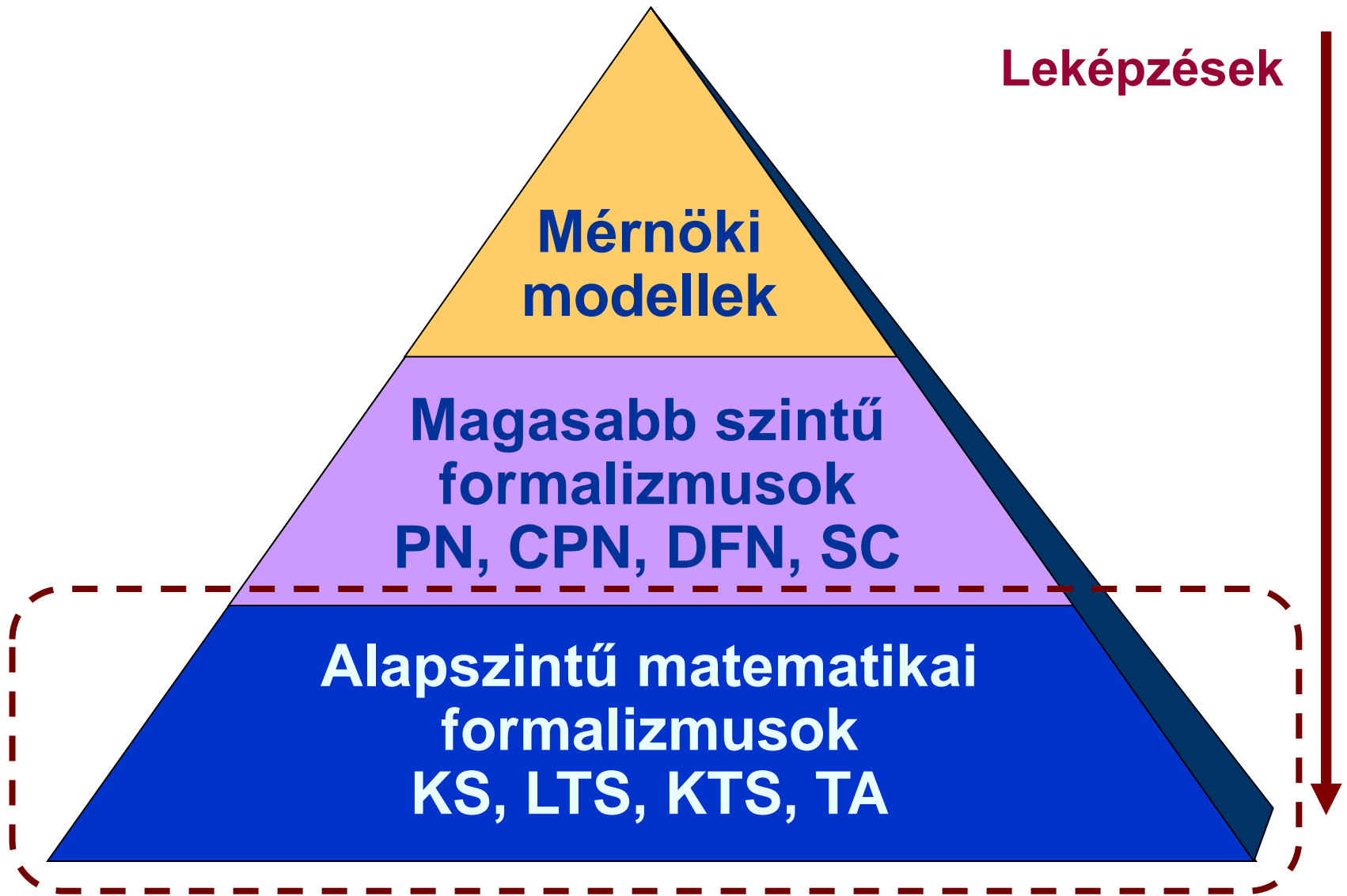
dr. Majzik István

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
 - Kiterjesztések: Változók, óraváltozók, szinkronizáció

1. Kripke-struktúra

KS, Kripke-structure:

- Állapotok, állapotátmenetek, állapotok címkézése
- **Állapotok** tulajdonságait fejezzük ki: címkézés **atomi kijelentésekkel**
- Egy állapothoz több címke is rendelhető

Alkalmazás: Viselkedés, algoritmus leírása

$KS = (S, R, L)$ és AP , ahol

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

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

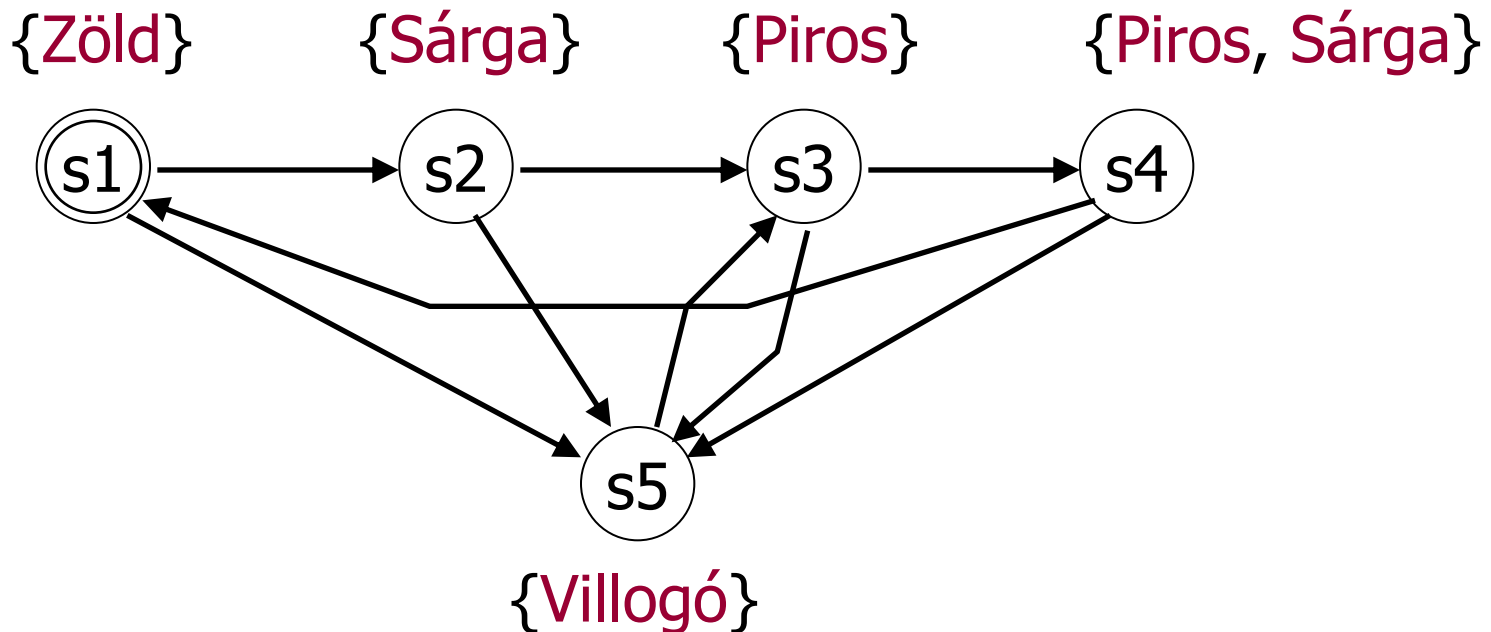
$R \subseteq S \times S$: állapotátmeneti reláció

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

Kripke-struktúra példa

Közlekedési lámpa viselkedése

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



2. Címkezett tranzíciós rendszer

LTS, Labeled Transition System:

- Állapotok, állapotátmenetek, állapotátmenetek címkézése
- **Állapotátmenetek** tulajdonságait fejezzük ki: címkézés **akciókkal**
- Egy állapotátmeneten csak egy akció szerepelhet (konvenció)

Alkalmazás: Kommunikáció, protokollok modellezése

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

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

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

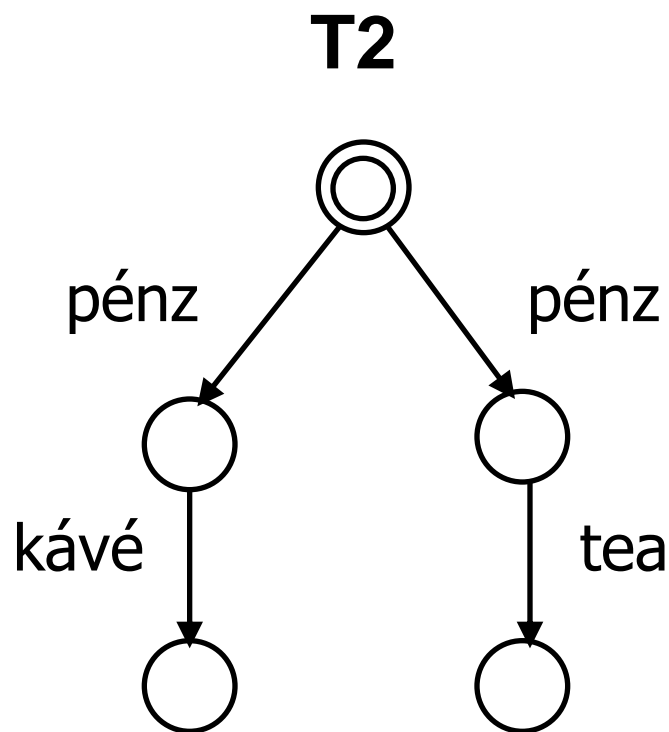
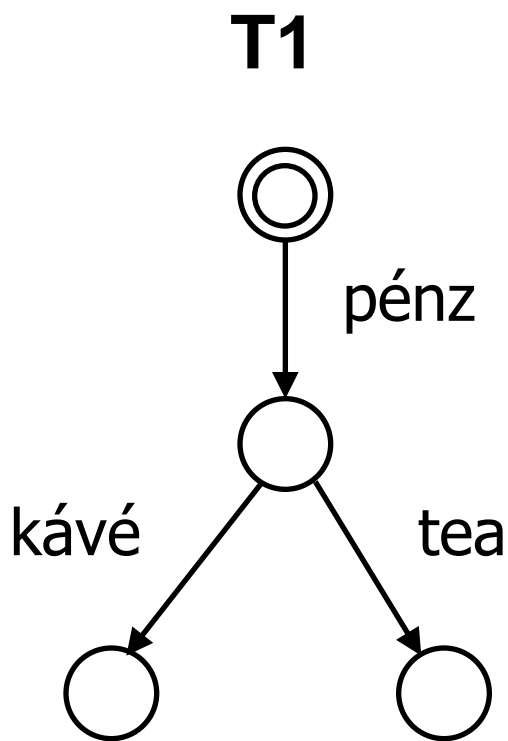
$\rightarrow \subseteq S \times Act \times S$ címkezett állapotátmenetek

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, állapotátmenetek, címkézések
- Állapotok és állapotátmenetek 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

$AP = \{P, Q, R, \dots\}$ atomi kijelentések halmaza

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

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

$\rightarrow \subseteq S \times Act \times S$ állapotátmeneti reláció (címkékkel)

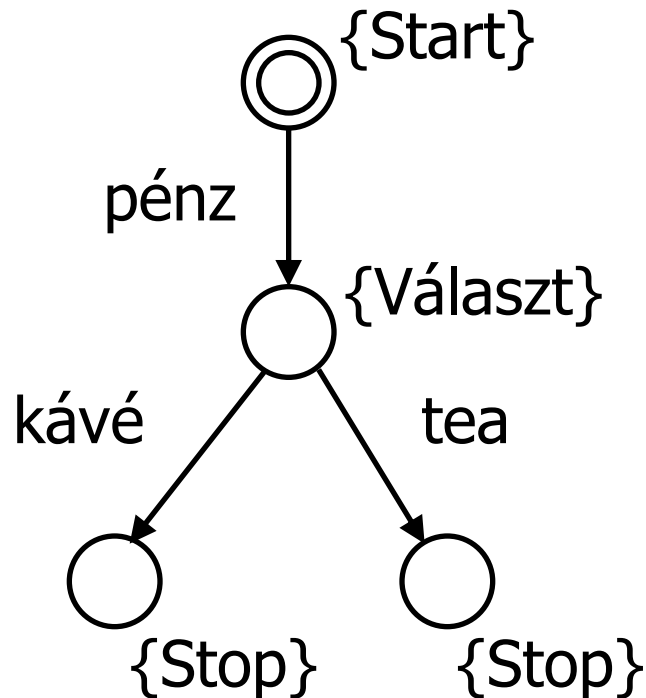
$L: S \rightarrow 2^{AP}$ állapotok címkézése atomi kijelentésekkel

KTS példa

- Italautomata modellje állapot címkékkel

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

AP = {Start, Választ, Stop} állapotok kijelzése



Időzített automaták
(az UPPAAL eszközben)

Time 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 deklarálnak típusal (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

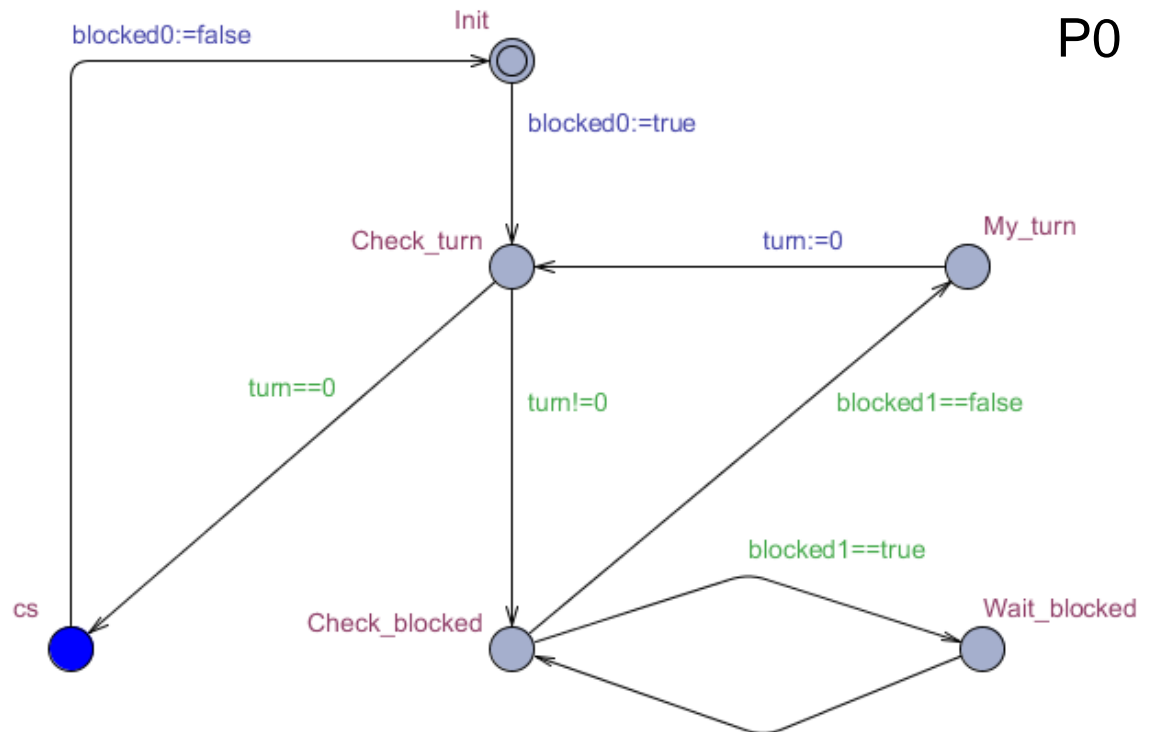
Deklarációk:

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

A P0 automata pseudokódja és modellje:

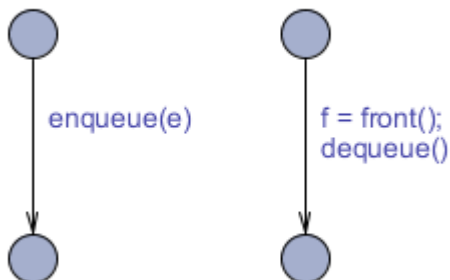
```
while (true) {  
    blocked0 := true;  
    while (turn!=0) {  
        while (blocked1==true) {  
            skip;  
        }  
        turn := 0;  
    }  
    // Critical section (cs)  
    blocked0 := false;  
    // Do other things  
}
```

P0



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];
}

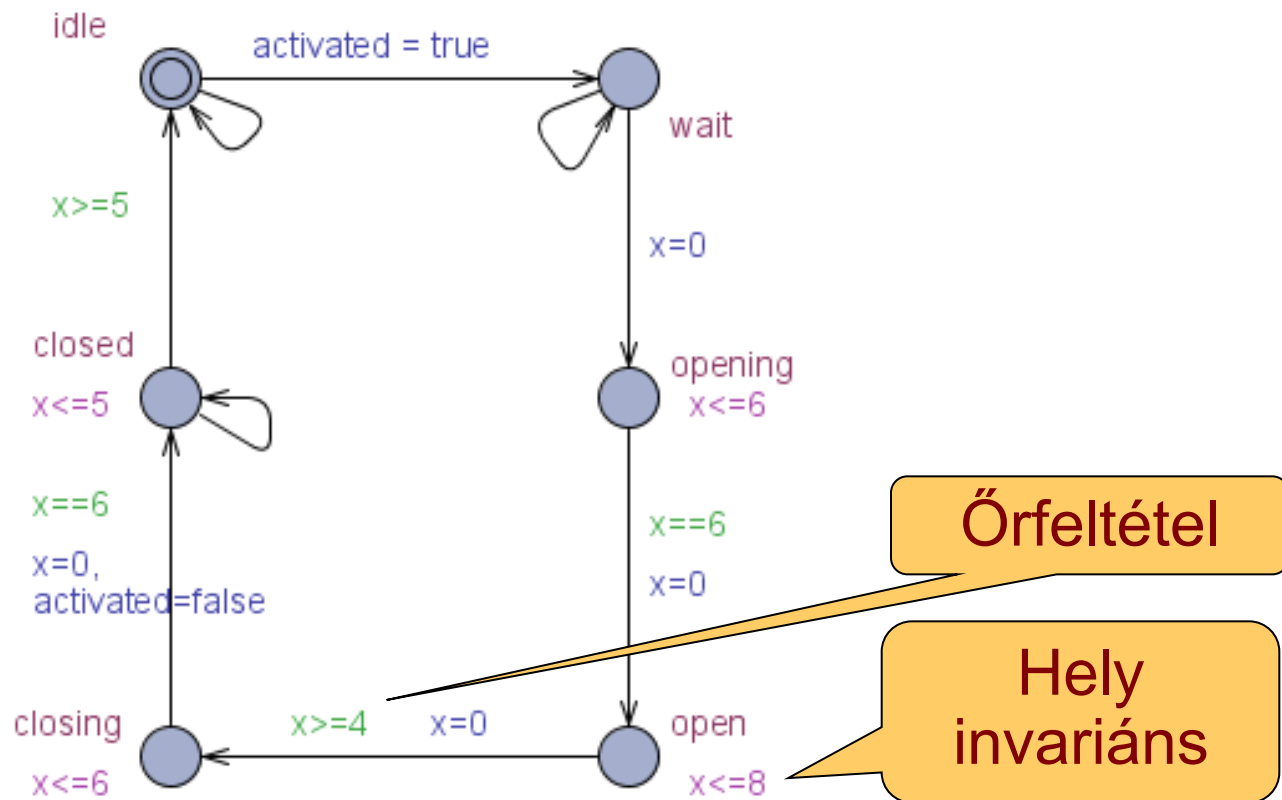
// Remove the front element of the list
void dequeue()
{
    int i := 0;
    while (i < last)
    {
        list[i] := list[i + 1];
        i++;
    }
    list[i] := 0;
    last := last-1;
}
```

Kiterjesztések óraváltozókkal

- Cél: Valószerű 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)
- Óraváltozók használata átmeneteken:
 - Akciók: Óraváltozók nullázása (resetelés), egymástól függetlenül
 - Örfeltételek: Óraváltozók és konstansok használhatók a predikátumokban
- Óraváltozók használata vezérlési helyeken:
 - Hely invariáns: Predikátum óraváltozókon és konstansokon; megadja, meddig állhat fenn az adott állapot

Az invariánsok és őrfeltételek szerepe

clock x;

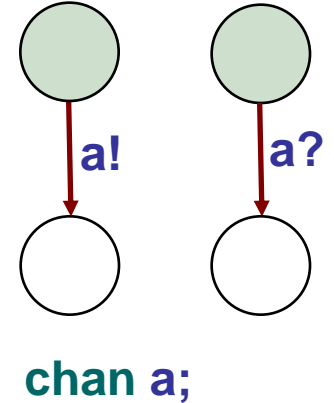


Az **open** állapot elhagyásakor a $[4, 8]$ tartományban lehet az x óra értéke



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

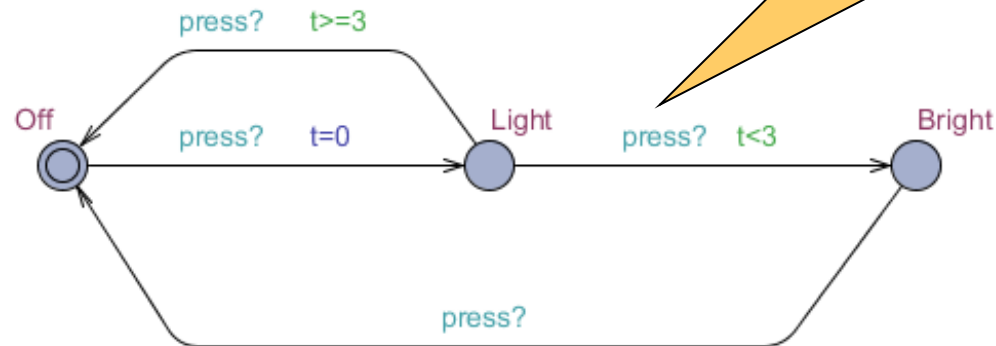


Példa óraváltozókra és szinkronizálásra

Deklarációk:

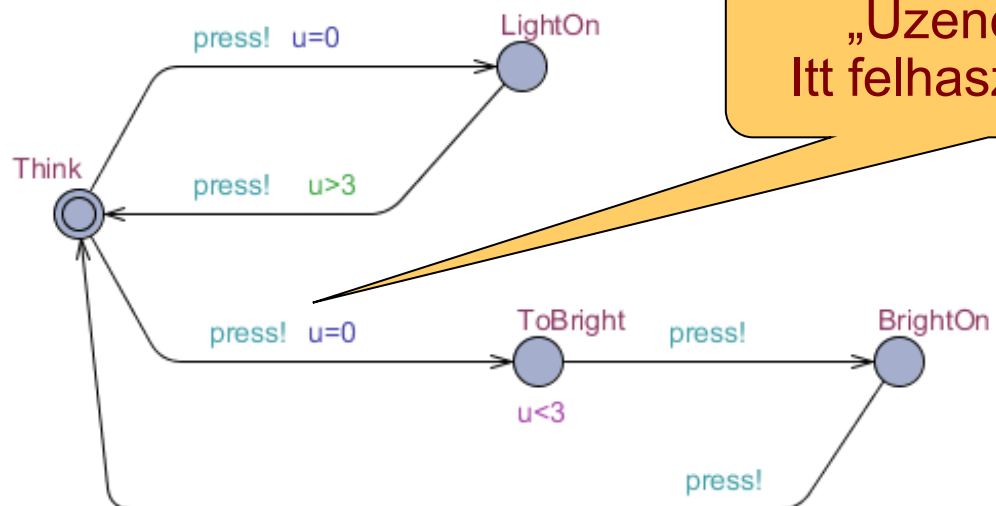
```
clock t, u;  
chan press;
```

Kapcsoló:



„Üzenet fogadás”:
Itt felhasználói input

Felhasználó:



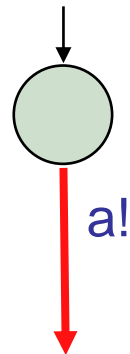
„Üzenet küldés”:
Itt felhasználói output

További lehetőségek: Broadcast csatorna

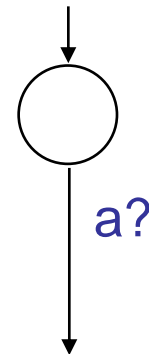
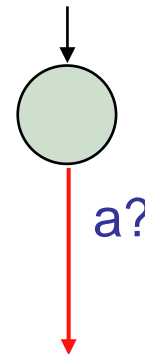
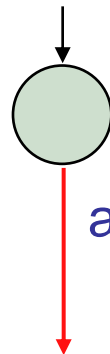
- Broadcast csatorna: $1 \rightarrow N$ kommunikáció
 - „Üzenetküldés” feltétel nélkül megtörténik
 - Nem kell fogadó készenlétére (randevúra) várni
 - Minden „üzenetfogadásra kész” partner erre szinkronizálódik
 - Üzenetfogadáshoz szükséges az üzenetküldés
 - Használati feltétel: Nem szerepelhet őrfeltétel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

broadcast chan a;

Üzenetküldő:



Üzenet fogadók:



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:

urgent chan a;



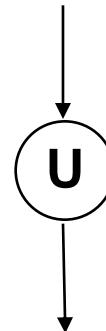
További lehetőségek: Speciális állapotok

- **Urgent** állapot: késleltetés korlátozása

- Nem telhet idő az adott állapotban

- Ekvivalens modell:

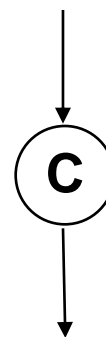
- Óraváltó bevezetése: `clock x;`
- Minden bemenő élen resetelve: `x:=0`
- Állapot invariáns hozzárendelése: `x<=0`



- **Committed** állapot: átmenetek egybefogása

- Bemenő és kimenő átmenet egy atomi műveletként végrehajtva

- A bemenő és kimenő átmenetek végrehajtása között más automata normál átmenete nem lehet végrehajtva



Adatértékek véletlen választása: szelekció

- Véletlen adatválasztás modellezése: **select**

- Változó megadása típussal

Pl. deklarációk:

```
typedef int[1,6] dice_type;
```

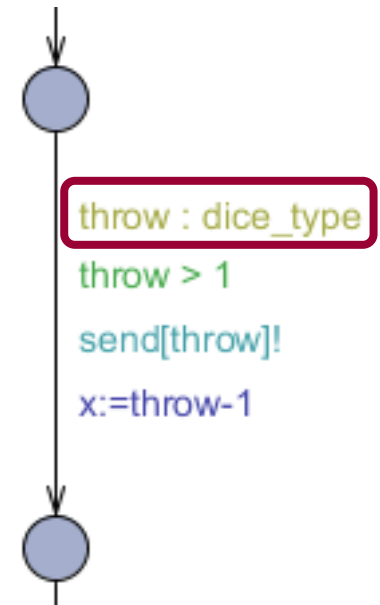
select konstrukció az átmeneten:

```
throw: dice_type
```

- A változót **véletlenszerűen** adatértékhez köti
a változó **típusa szerinti tartományból**

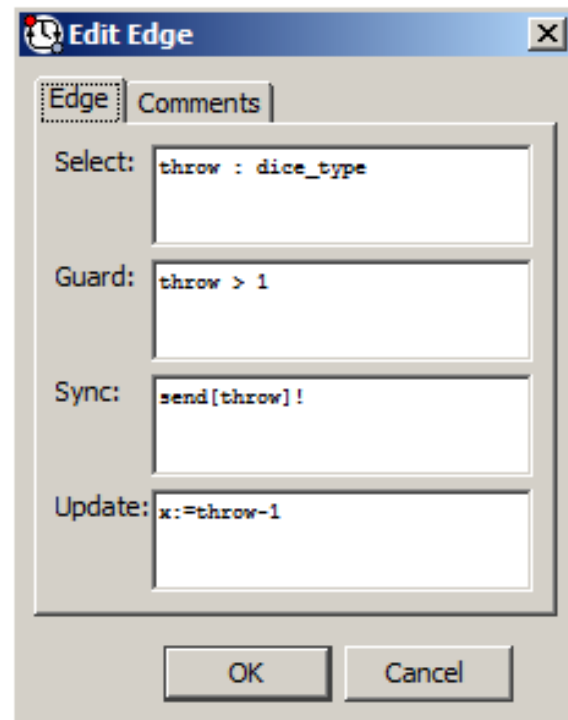
Pl. a throw változó értéke 1 és 6 között lesz

- 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 automataéhoz 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

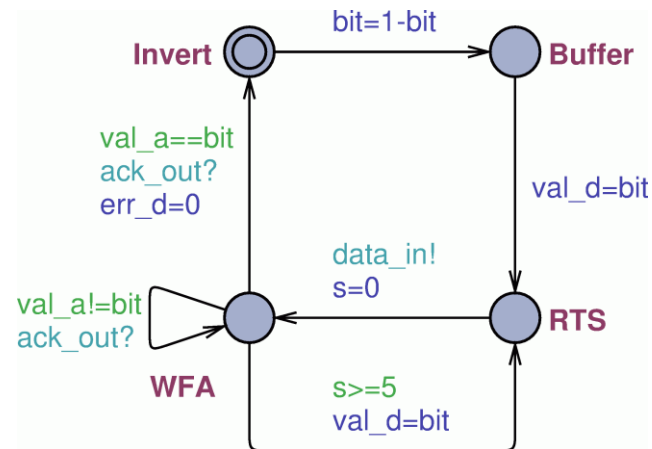
$G(C,V)$ predikátumok órákon vagy változókon

- Hely invariánsok:

$$Inv: N \rightarrow G(C)$$

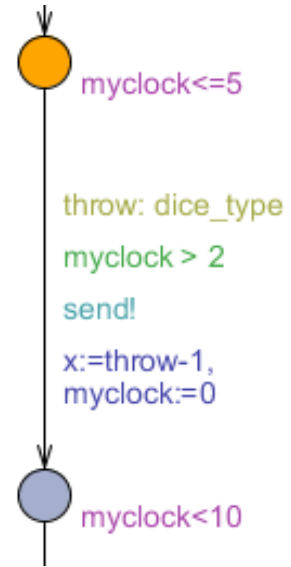
- Vezérlési helyek nevei:

$$L: N \rightarrow A \text{ itt } A \text{ névhalmoz}$$



Szemantika egy automata esetén

- Kezdeti állapot:
 - Kezdő vezérlési hely aktív, minden óra 0
- 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
 - Örfeltétel teljesül
 - Szinkronizáció bekövetkezhet
 - Órák nullázása teljesíti a cél hely 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 átment célhelye válik aktívvá



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

Automaták példányosítása: Motiváció

Deklarációk:

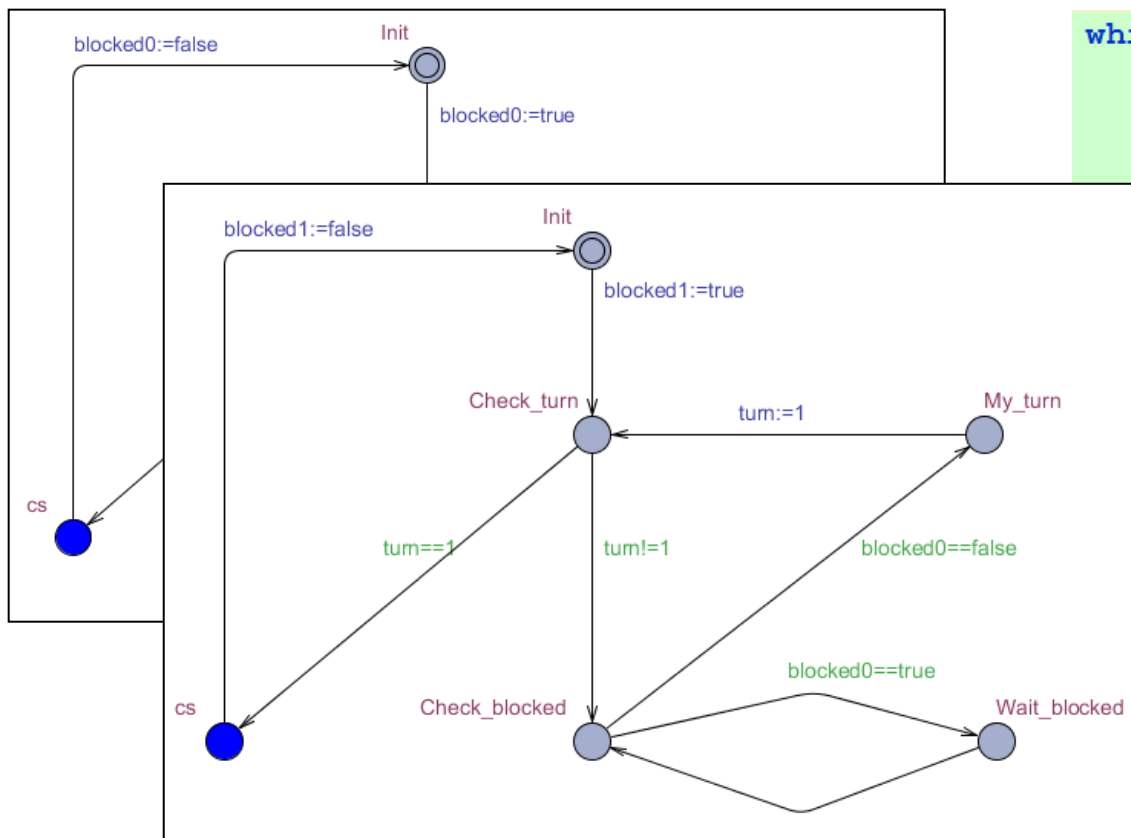
```
bool blocked0=false;
```

```
bool blocked1=false;
```

```
int[0,1] turn=0;
```

```
system P0, P1;
```

A P0 és P1 automata:



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

P0

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

P1

Automaták példányosítása

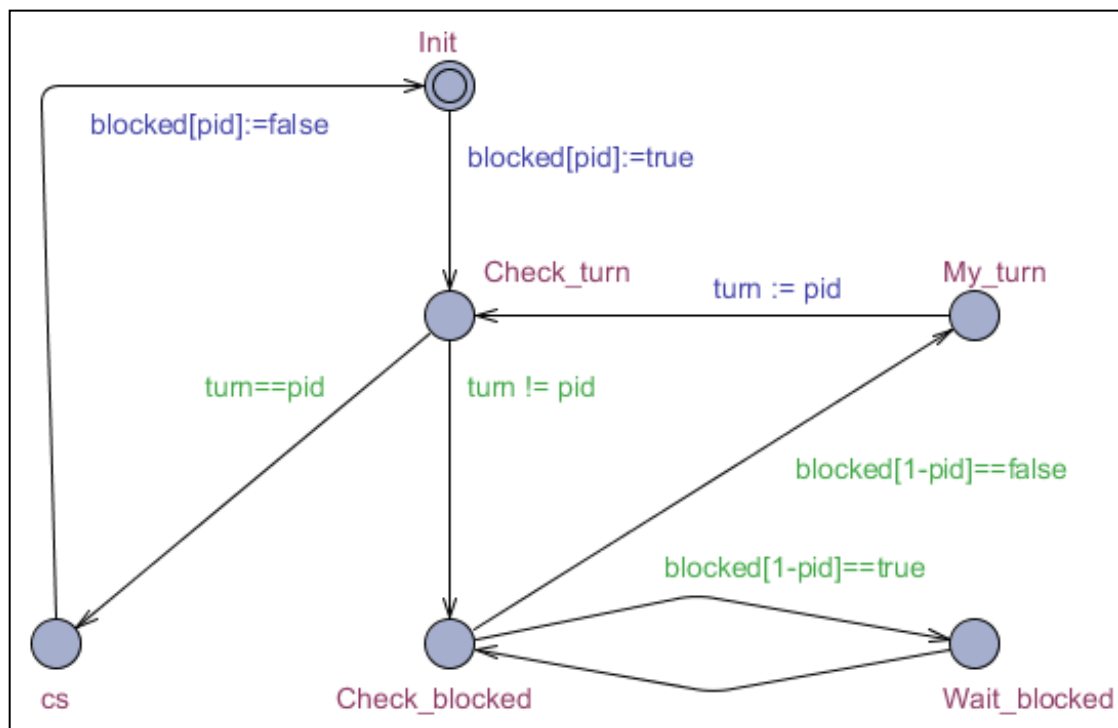
Deklarációk:

```
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:



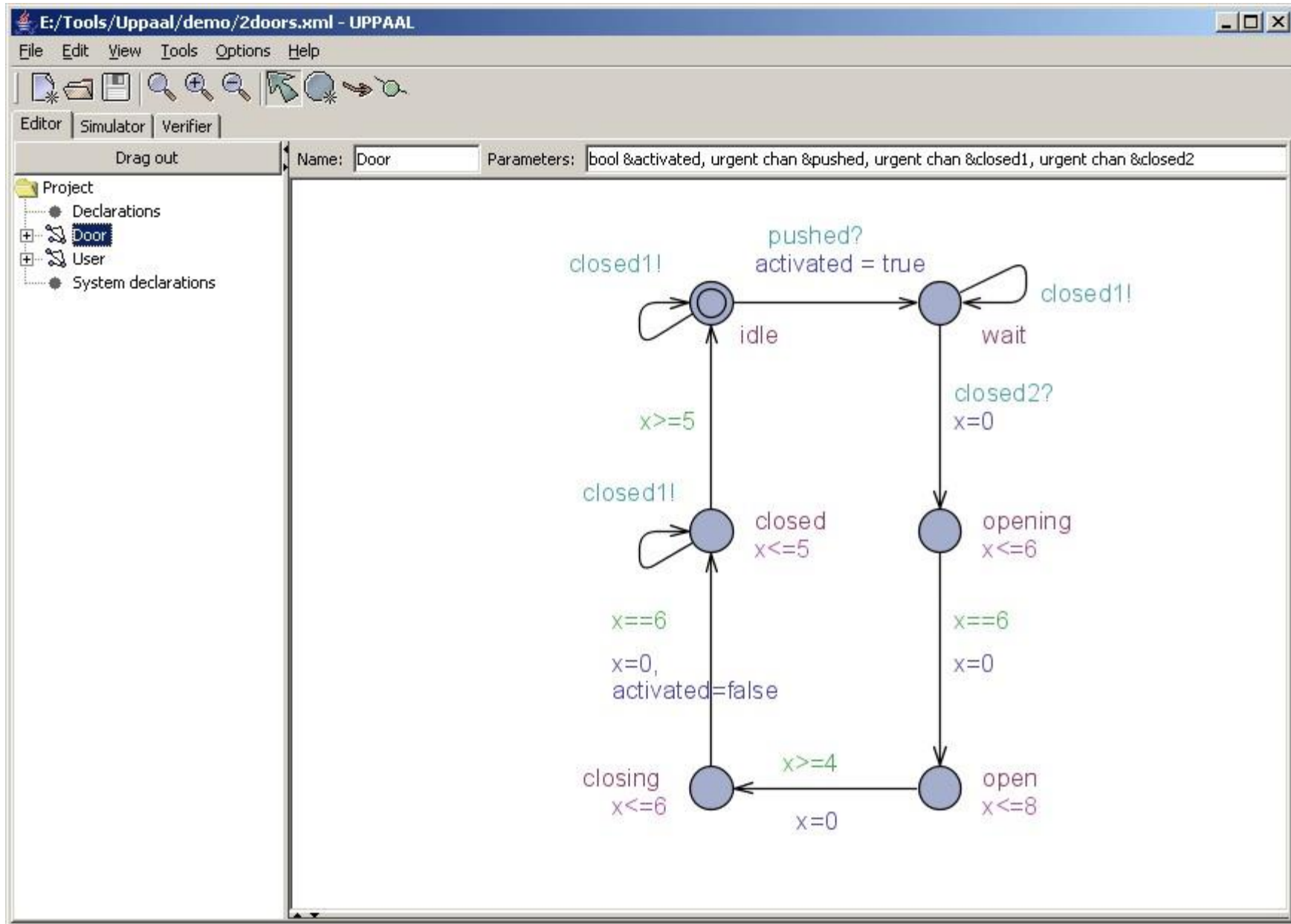
P0 példány pid=0 mellett:

```
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

Door2

User1

User2

Door1 Door2 User1 User2

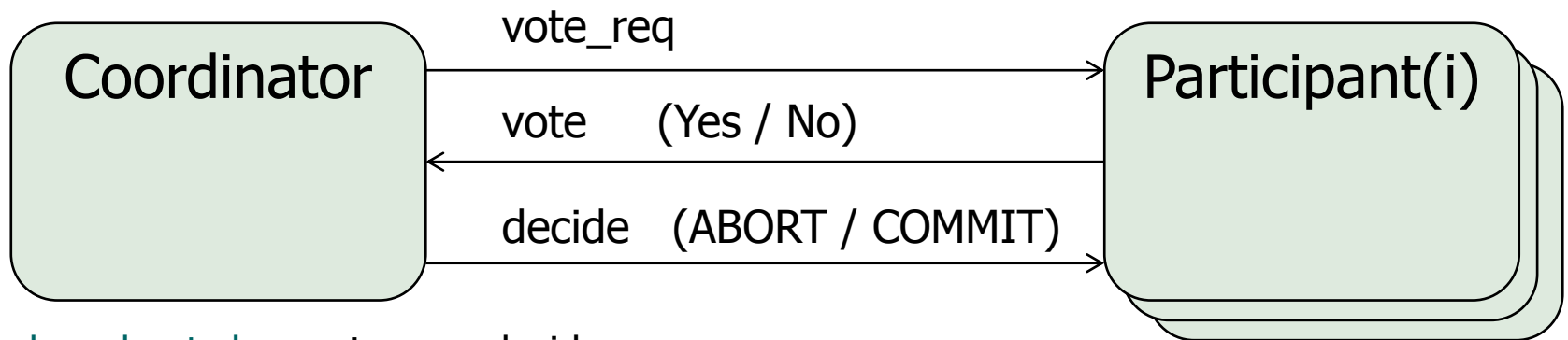
Verifikáció

The screenshot shows the UPPAAL software interface with the following components:

- File:** FTapps/Uppaal/demo/2doors.xml - UPPAAL
- Menu:** File, Edit, View, Tools, Options, Help
- Toolbar:** Includes icons for file operations (new, open, save), search, and navigation.
- Editor:** Contains the main verification report.
- Overview:** A list of properties with status indicators (green for satisfied, grey for not checked).
 - `A[] not (Door1.open and Door2.open)` (Green)
 - `A[] (Door1.opening imply User1.w<=31) and (Door2.opening imply User2.w<=31)` (Green)
 - `E<> Door1.open` (Grey)
 - `E<> Door2.open` (Green)
- Query:** `A[] not (Door1.open and Door2.open)`
- Comment:** Mutex: The two doors are never open at the same time.
- Status:** A log of verification results.
 - Established direct connection to local server.
 - (Academic) UPPAAL version 4.0.7 (rev. 4140), November 2008 -- server.
 - Disconnected.
 - Established direct connection to local server.
 - (Academic) UPPAAL version 4.0.7 (rev. 4140), November 2008 -- server.
 - `A[] not (Door1.open and Door2.open)`
Property is satisfied.
 - `A[] (Door1.opening imply User1.w<=31) and (Door2.opening imply User2.w<=31)`
Property is satisfied.
 - `E<> Door2.open`
Property is satisfied.
 - `A[] not deadlock`
Property is satisfied.
 - `Door2.wait --> Door2.open`
Property is satisfied.
 - `Door1.wait --> Door1.open`
Property is satisfied.

Mintapélda:
A kétfázisú commit protokoll

A modell struktúrája



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

// Csatornák

```
const int N=2;
typedef int[1,N] pid_type;
typedef int[0,1] vote_type;
```

// Résztevők száma
// Résztevők azonosítóinak típusa
// Szavazatok típusa (0: No, 1: Yes)

```
int decision = 0;
const int ABORT = 0;
const int COMMIT = 1;
```

// A koordinátor döntése

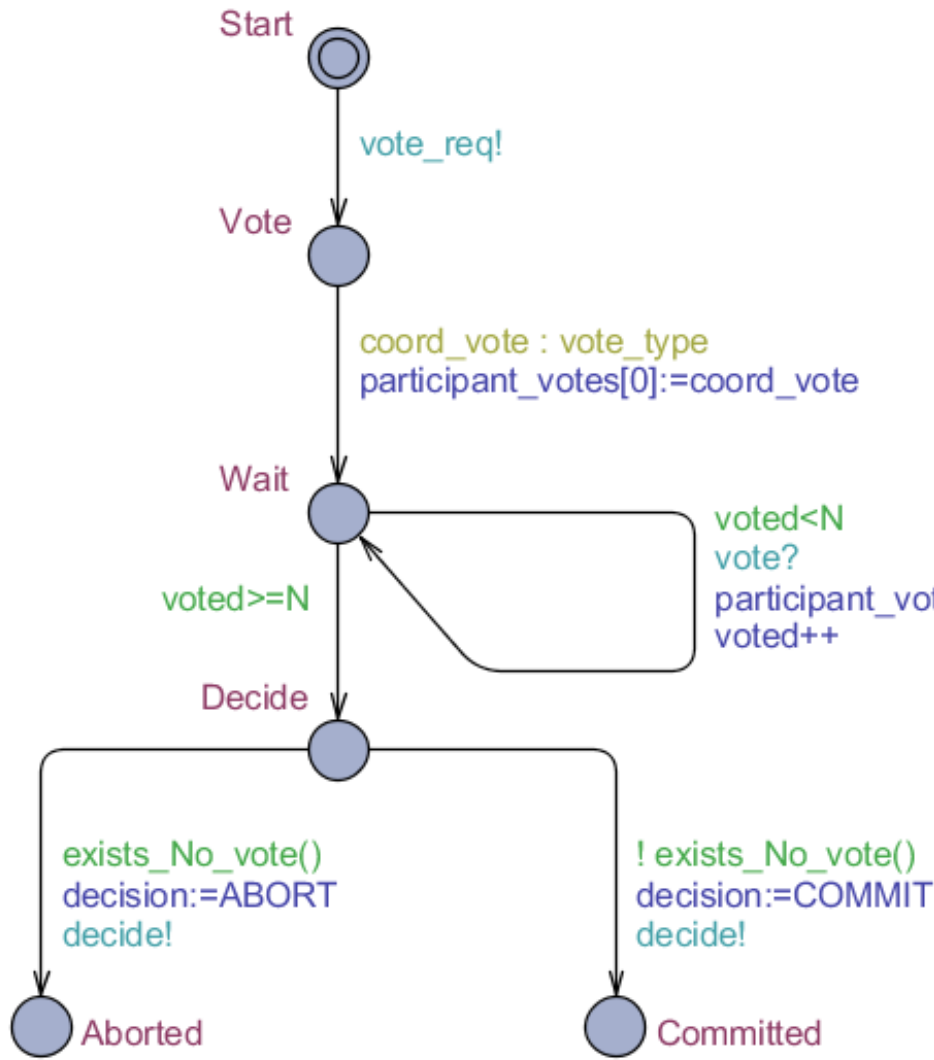
```
vote_type my_vote;
int my_pid;
```

// Szavazat küldve a résztvevőtől
// Azonosító küldve a résztvevőtől

```
int voted = 0;
vote_type participant_votes[N+1];
```

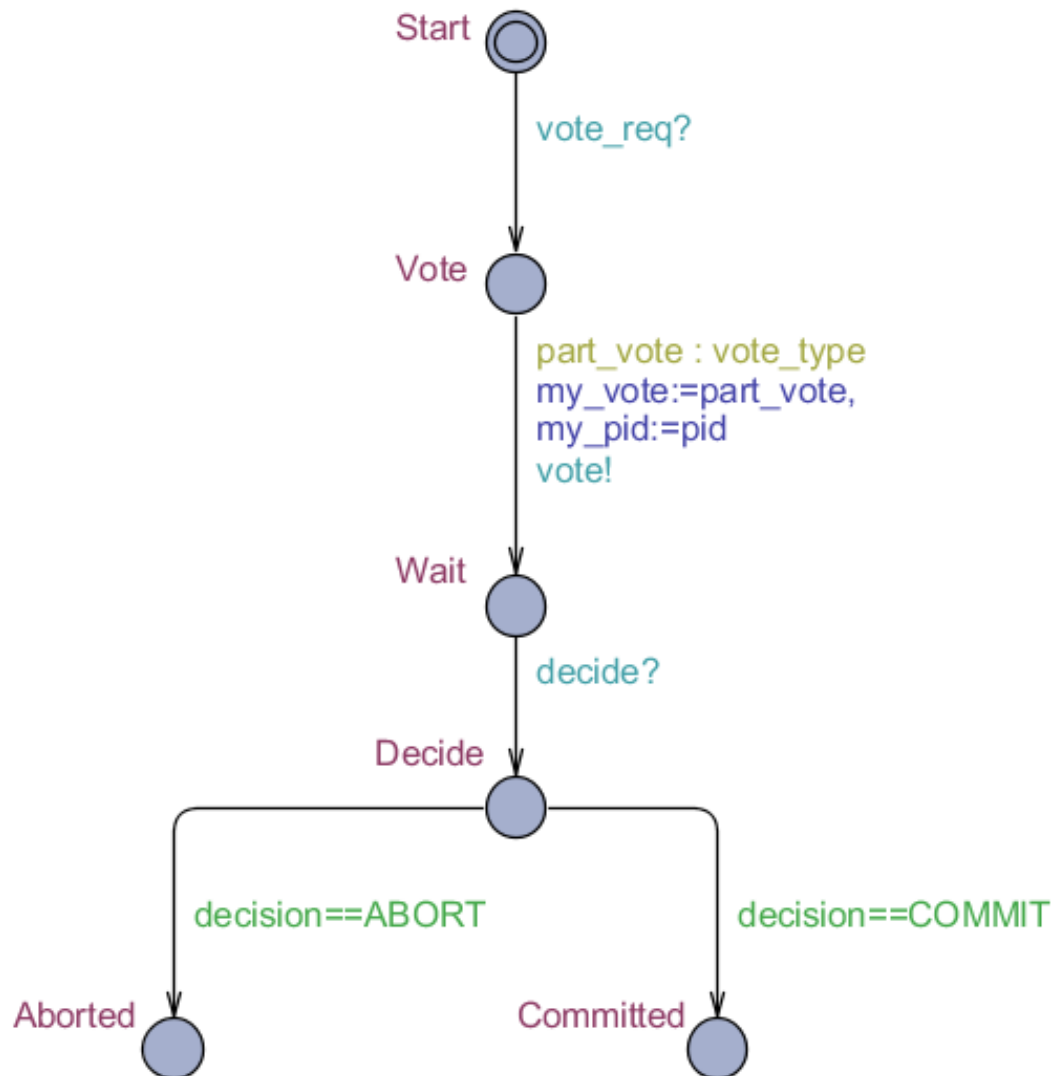
// A szavazott résztvevők száma
// A koordinátor és résztvevők szavazatai

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



Ellenőrizendő tulajdonságok (példák)

$A[]$ logikai operátor: Minden végrehajtási út 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:

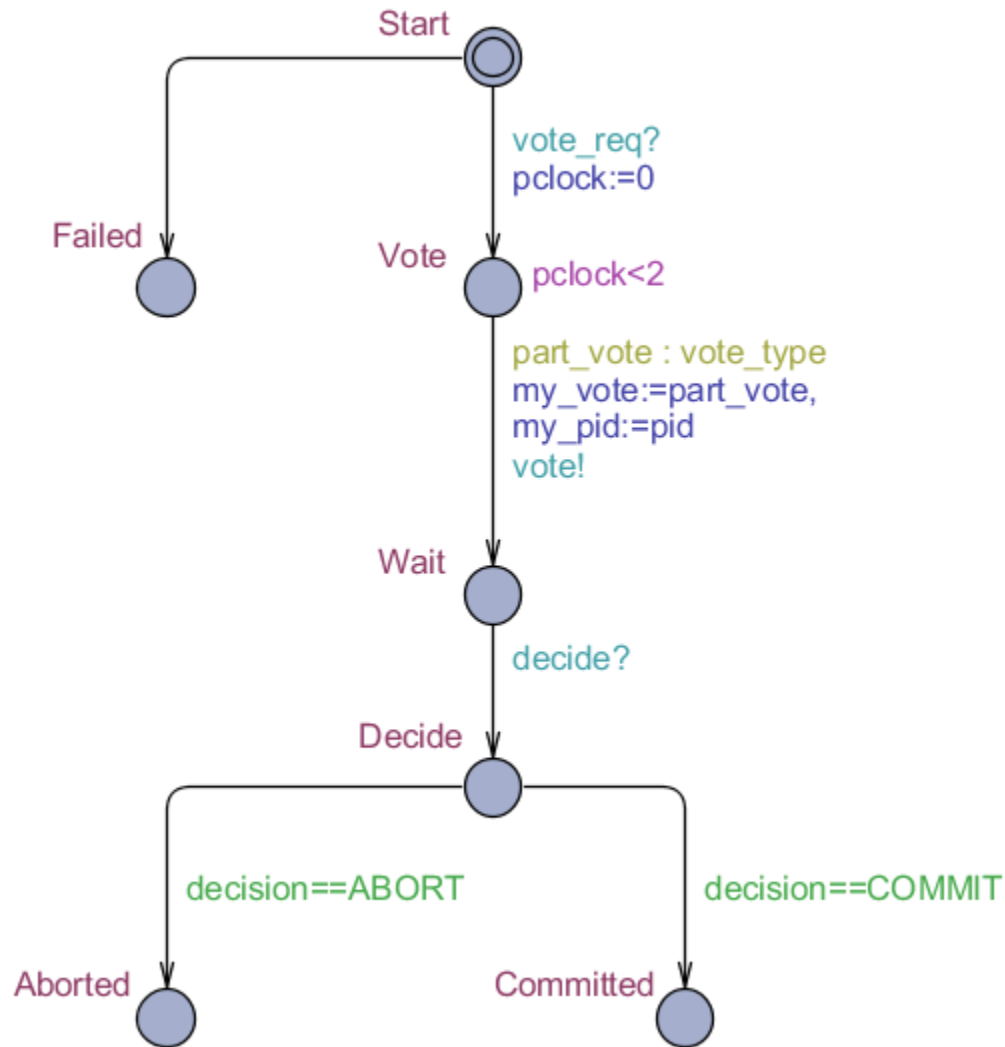
$A[]$ (Coordinator.Committed imply forall (i : int[0,N])
Coordinator.participant_votes[i]==1)

- 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)

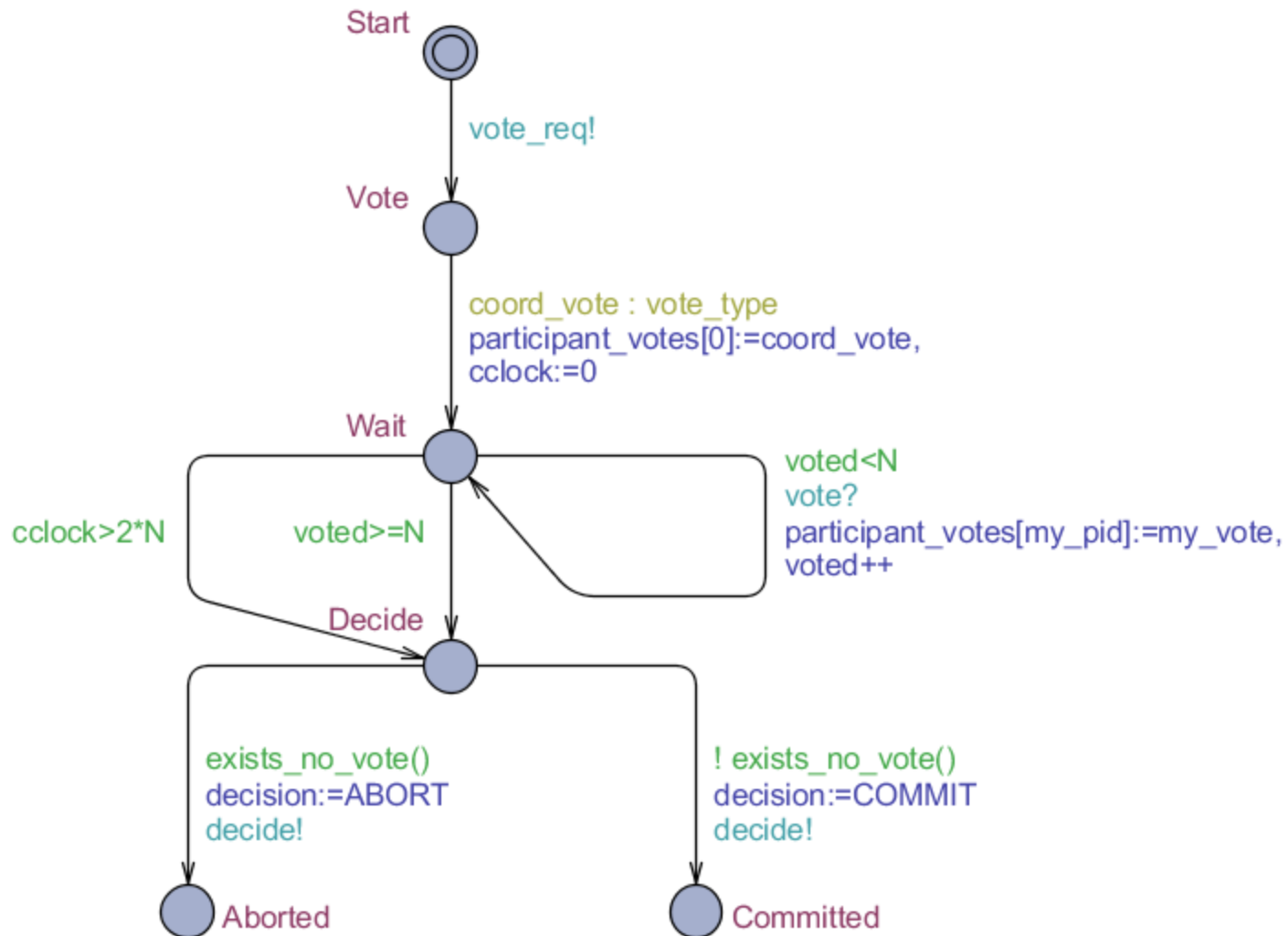
Résztevő: Kiesés és időkorlát a szavazásra



`clock pclock;`

`// Résztevő lokális órája`

Koordinátor: Timeout a szavazáskor



```
clock cclock;           // Koordinátor lokális órája
```