

Modellezés UPPAAL-ban

Kockajáték mintafeladat és megoldása

dr. Bartha Tamás

BME Méréstechnika és Információs Rendszerek Tanszék

Tartalom

- A mintafeladat demonstrál néhány hasznos UPPAAL modellezési fogást:
 - Véletlen érték generálása és felhasználása
 - Atomi műveletek modellezése
 - Szinkron kommunikáció modellezése
 - Globális osztott változó használatával
 - Dedikált csatornatömbök alkalmazásával
 - Adatstruktúrák és függvények használata
 - Változók kezelése (állapottér csökkentése)
 - Temporális kifejezések írása modellellenőrzéshez

A feladat

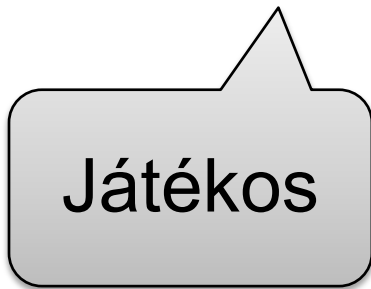
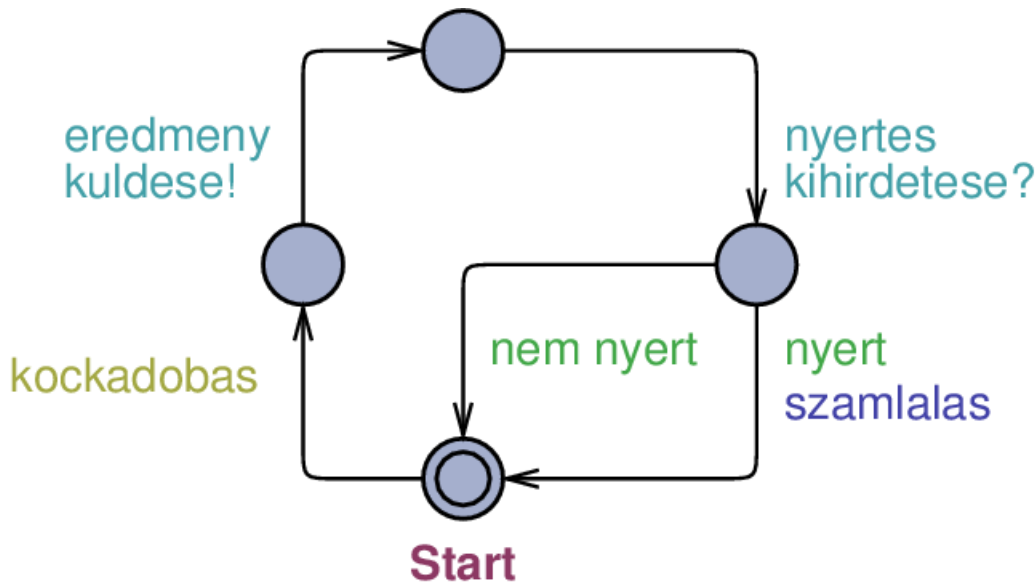
A feladat

- Kockadobálós játék
 - Szereplők: n játékos, 1 bíró
 - Minden játékos egy kockával egyszer dob
 - Közlik az eredményt a bíróval
 - A bíró
 - feljegyzi az eredményeket,
 - megkeresi a legnagyobbat,
 - kihirdeti a nyertest
 - A játékosok számlálják a nyeréseket

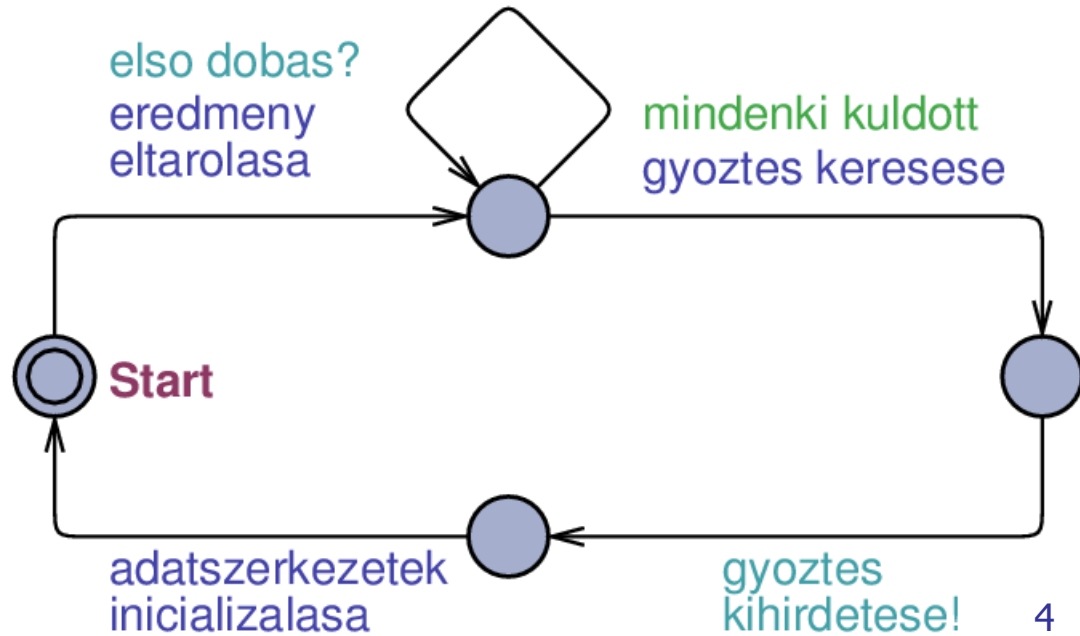
Mit kell megoldanunk?

- Véletlen érték generálása
- Kommunikáció
 - Értékek „továbbítása”
 - „Broadcast” kommunikáció
 - Csatornatömbök kezelése
- Adatszerkezetek
- Függvények
- Konkurencia és időzítés
- Modellellenőrzés

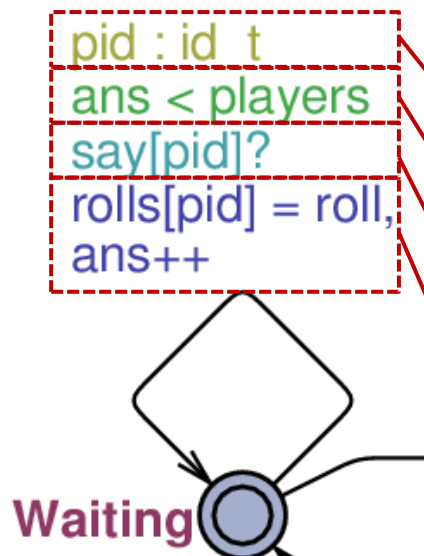
Szereplők és állapotok absztrakt modellje



amíg mindenki el nem kuli:
következő dobás?
eredmény eltarolasa



Modellezési kitérő: az élekhez rendelt kifejezések



Edit Edge

Edge Comments

Select: pid : id t

Guard: ans < players

Sync: say[pid]?

Update: rolls[pid] = roll,
ans++

OK Cancel

- Az élek esetén a szekciók kiértékelési sorrendje:
Select » Sync » Guard » Update

- Selection
 - Nemdeterminisztikus választás egy változó értékészletéből
- Guard
 - Engedélyező feltétel (logikai kifejezés)
- Synchronization
 - Szinkronizáció adott csatornán megfelelő processz „párok” között
- Update
 - Állapotváltás esetén kiértékelt kifejezés (mellékhatása is lehet)

Modellezés: A rendszer és egy játékos

Rendszer:

```
system Player, Referee;
```

```
const int players = 3;
```

```
const int wins = 10;
```

```
typedef int[0,players-1] id_t;
```

```
typedef int[0,6] dice_t;
```

```
struct {  
    id_t who;  
    dice_t what;  
} roll;
```

```
id_t winner;
```

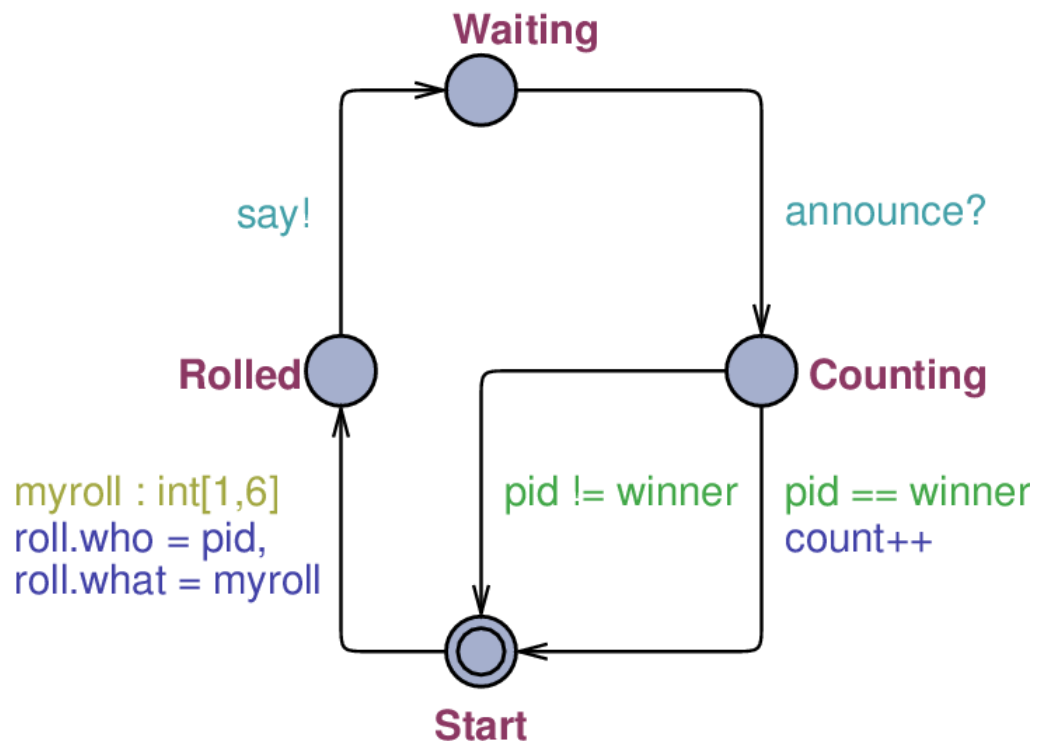
```
chan say;
```

```
broadcast chan announce;
```

Játékos:

```
Player(id_t pid)
```

```
int[0,wins] count = 0;
```



Modellezés: A bíró

Bíró:

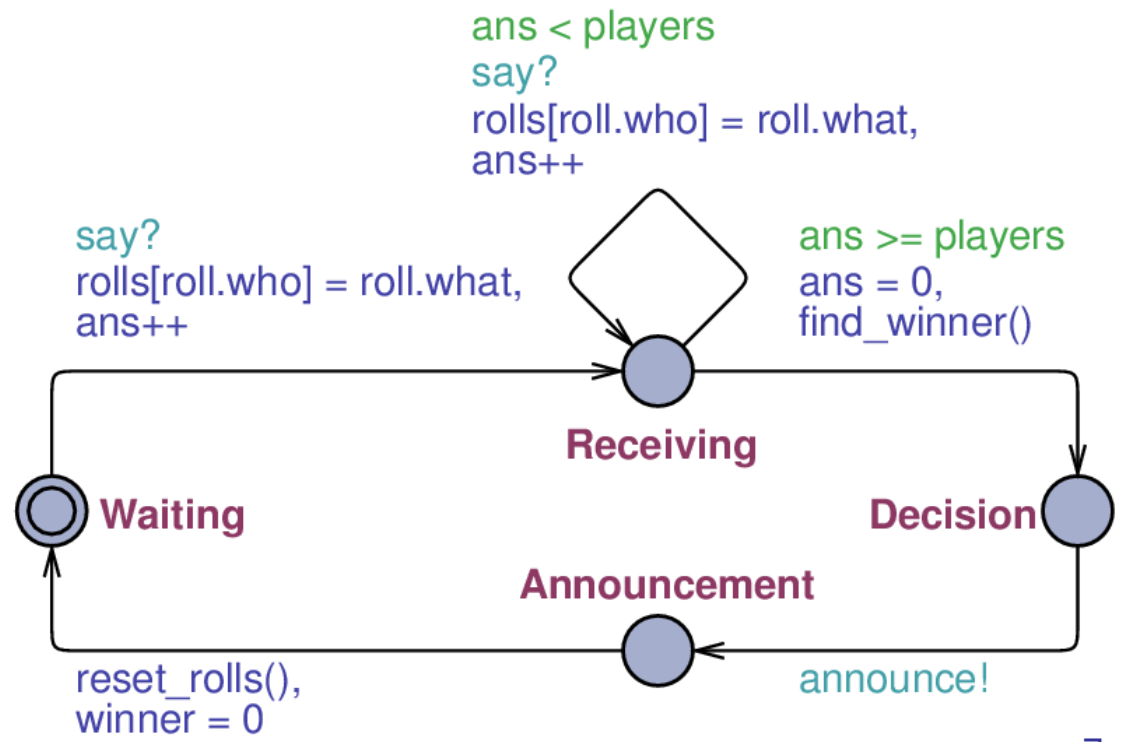
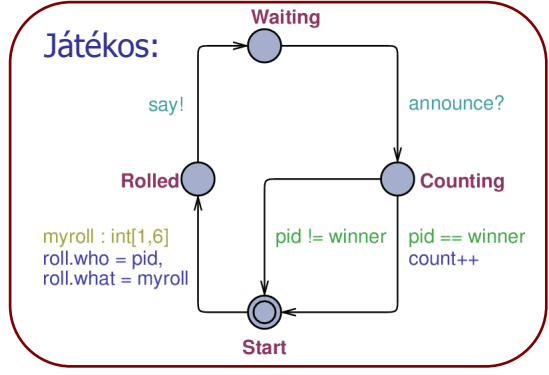
```
int [0,players] ans = 0;
dice_t rolls[id_t];
dice_t best = 0;
```

```
void find_winner() {
    int[0,players] i;

    for (i = 0; i < players; i++) {
        if (rolls[i] > best) {
            best = rolls[i];
            winner = i;
        }
    }
    best = 0;
}
```

```
void reset_rolls() {
    int[0,players] i;

    for (i = 0; i < players; i++) rolls[i] = 0;
}
```



Ellenőrizzük a működést! (dice_roll_1)

- Mindig van győztes a játékban
 - Mindig van olyan játékos, aki maximális számú alkalommal nyer
 $A \langle \rangle \text{ exists } (i : \text{id_t}) (\text{Player}(i).\text{count} == \text{wins})$
- A bíró csak akkor hoz döntést, ha minden játékos dobott
 - Ez legalább egyszer megtörténik:
 $E \langle \rangle \text{ Referee.Decision} \ \&\& \ \text{forall } (i : \text{id_t}) (\text{Referee.rolls}[i] > 0)$
 - Ez minden lehetséges úton bekövetkezik:
 $A \langle \rangle \text{ Referee.Decision} \ \&\& \ \text{forall } (i : \text{id_t}) (\text{Referee.rolls}[i] > 0)$
- A rendszerben nincs holtpont
 - Nincs olyan állapot, amelyből ne vezetne ki olyan engedélyezett állapotátmenet, amellyel egy következő állapotba léphetünk
 $A[] \text{ not deadlock}$

Ellenőrizzük a működést! (dice_roll_1)

Overview

```
A<> exists (i : id_t) (Player(i).count == wins)
A<> Referee.Decision && forall (i : id_t) Referee.rolls[i] > 0
E<> Referee.Decision && forall (i : id_t) Referee.rolls[i] > 0
A[] not deadlock
```

Check
Insert
Remove
Comments

Query

```
A<> exists (i : id_t) (Player(i).count == wins)
```

Comment

Status

```
A[] not deadlock
Established direct connection to local server.
(Academic) UPPAAL version 4.0.13 (rev. 4577), September 2010 -- server.
The verification was aborted due to an error. Most likely, this is caused by an out-of-range assignment or out-of-range array lookup.
E<> Referee.Decision && forall (i : id_t) Referee.rolls[i] > 0
Property is satisfied.
A<> Referee.Decision && forall (i : id_t) Referee.rolls[i] > 0
Property is not satisfied.
A<> exists (i : id_t) (Player(i).count == wins)
Property is not satisfied.
```

Holtpontmentesség: nem fut le.

- Mert nem akadályoztuk meg a nyerési számlálók túlfutását (ld. `count` típusa és `count++`).
- (Most nem javítjuk.)

Ellenőrizzük a működést! (dice_roll_1)

The screenshot shows a software interface with the following sections:

- Overview:** A list of queries with status indicators (red, green, grey) and buttons for 'Check', 'Insert', 'Remove', and 'Comments'.
 - Query 1: `A<> exists (i : id_t) (Player(i).count == wins)` (Red indicator)
 - Query 2: `A<> Referee.Decision && forall (i : id_t) Referee.rolls[i] > 0` (Red indicator)
 - Query 3: `E<> Referee.Decision && forall (i : id_t) Referee.rolls[i] > 0` (Green indicator)
 - Query 4: `A[] not deadlock` (Grey indicator)
- Query:** `A<> exists (i : id_t) (Player(i).count == wins)`
- Comment:** (Empty)
- Status:** A log of execution results.
 - `A[] not deadlock`
 - Established direct connection to local server.
 - (Academic) UPPAAL version 4.0.13 (rev. 4577), September 2010 -- server.
 - The verification was aborted due to an error. Most likely, this is caused by an out-of-range assignment or out-of-range array lookup.
 - `E<> Referee.Decision && forall (i : id_t) Referee.rolls[i] > 0`
 - `Property is satisfied.` (Green text)
 - `A<> Referee.Decision && forall (i : id_t) Referee.rolls[i] > 0`
 - `Property is not satisfied.` (Red text)
 - `A<> exists (i : id_t) (Player(i).count == wins)`
 - `Property is not satisfied.` (Red text)

A callout box with green text states: "Lehetséges, hogy eljutunk olyan állapotba, amelyben a bíró döntött, és minden játékos dobásának feljegyzése megtörtént." (It is possible that we reach a state where the referee has decided and all player dice rolls are recorded.)

Ellenőrizzük a működést! (dice_roll_1)

The screenshot shows a software verification tool interface. At the top, there is an 'Overview' section with a list of queries and their status indicators (red, green, or grey circles). The first query, 'A<> exists (i : id_t) (Player(i).count == wins)', is highlighted in blue and has a red status indicator. Below this is a 'Query' section containing the same query. A 'Comment' section is empty. The 'Status' section shows the execution results for the queries. The first query is marked as 'Property is not satisfied.' in red. The second query, 'A<> Referee.Decision && forall (i : id_t) Referee.rolls[i] > 0', is marked as 'Property is satisfied.' in green. The third query, 'E<> Referee.Decision && forall (i : id_t) Referee.rolls[i] > 0', is marked as 'Property is not satisfied.' in red. The fourth query, 'A[] not deadlock', is marked as 'Property is not satisfied.' in red. A red dashed arrow points from the first query in the Overview section to the 'Property is not satisfied.' message in the Status section. Another red dashed arrow points from the 'Property is not satisfied.' message in the Status section to a callout box on the right.

Overview

```
A<> exists (i : id_t) (Player(i).count == wins)
A<> Referee.Decision && forall (i : id_t) Referee.rolls[i] > 0
E<> Referee.Decision && forall (i : id_t) Referee.rolls[i] > 0
A[] not deadlock
```

Query

```
A<> exists (i : id_t) (Player(i).count == wins)
```

Comment

Status

```
A[] not deadlock
Established direct connection to local server.
(Academic) UPPAAL version 4.0.13 (rev. 4577), September 2010 -- server.
The verification was aborted due to an error. Most likely, this is caused by an out-of-range assignment or out-of-range array lookup.
E<> Referee.Decision && forall (i : id_t) Referee.rolls[i] > 0
Property is satisfied.
A<> Referee.Decision && forall (i : id_t) Referee.rolls[i] > 0
Property is not satisfied.
A<> exists (i : id_t) (Player(i).count == wins)
Property is not satisfied.
```

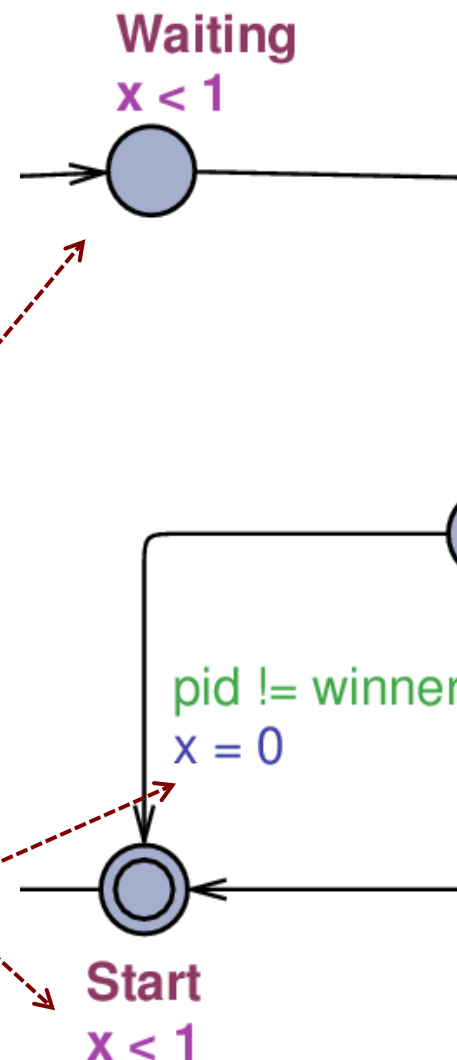
Check
Insert
Remove
Comments

Ellenpélda: Van olyan útvonal, ahol nem következnek be döntések és mindenki dobásának feljegyzése

- Triviális ellenpélda: időbeliség
- Konkurencia helytelen kezelése

Triviális ellenpélda kiküszöbölése: időzítés

- Ha az összes lehetséges lefutást vizsgáljuk (pl. $A \leftrightarrow$), akkor az UPPAAL figyelembe veszi azt a lehetőséget is, hogy egy állapotot sosem hagyunk el
- Megoldás:
 - Óraváltozó bevezetése
 - Invariáns előírása
 - Legfeljebb 1 időegységig tartózkodhatunk az adott állapotban
 - Gondoskodunk az óraváltozó inicializálásáról



Konkurens működés - mi a baj?

Player(1) kockával dobott

Player(0) kockával dob

Enabled Transitions

- Player(0)
- Player(0)
- Player(0)
- Player(0)
- Player(0)
- Player(0)
- Player(2)
- Player(2)
- Player(2)
- Player(2)

Next Reset

Simulation Trace

(Start, Start, Start, Waiting)

Player(1)

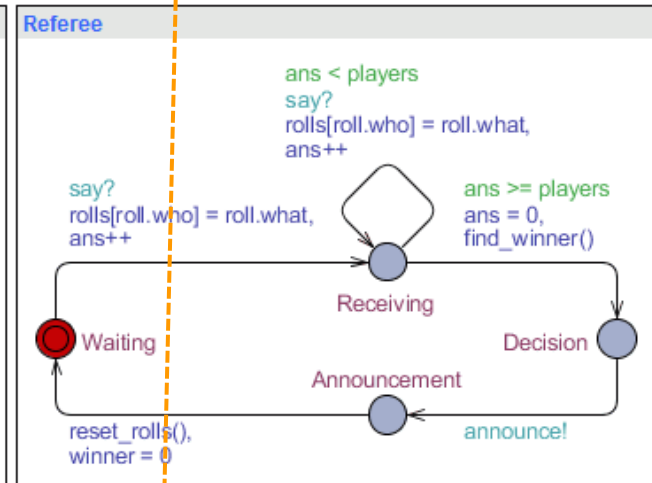
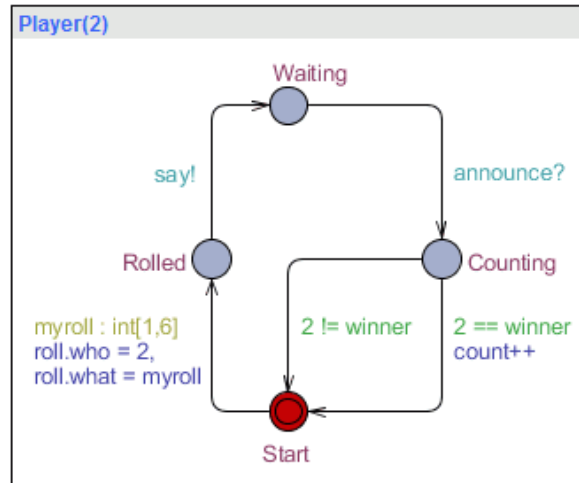
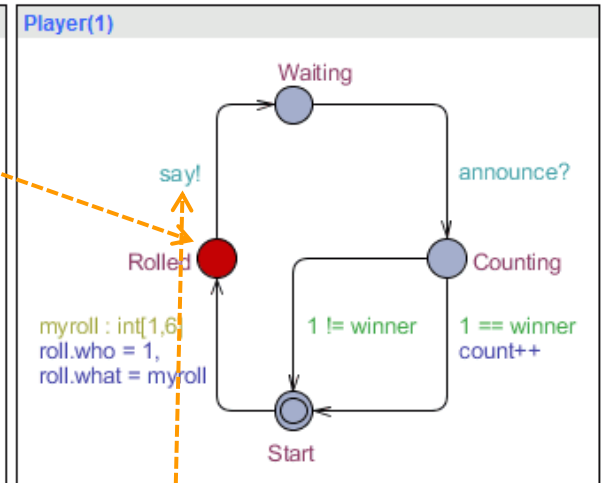
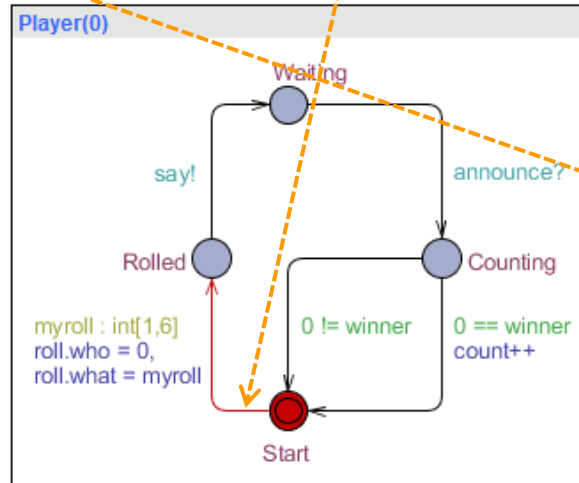
(Start, Rolled, Start, Waiting)

Player(0)

(Rolled, Rolled, Start, Waiting)

```

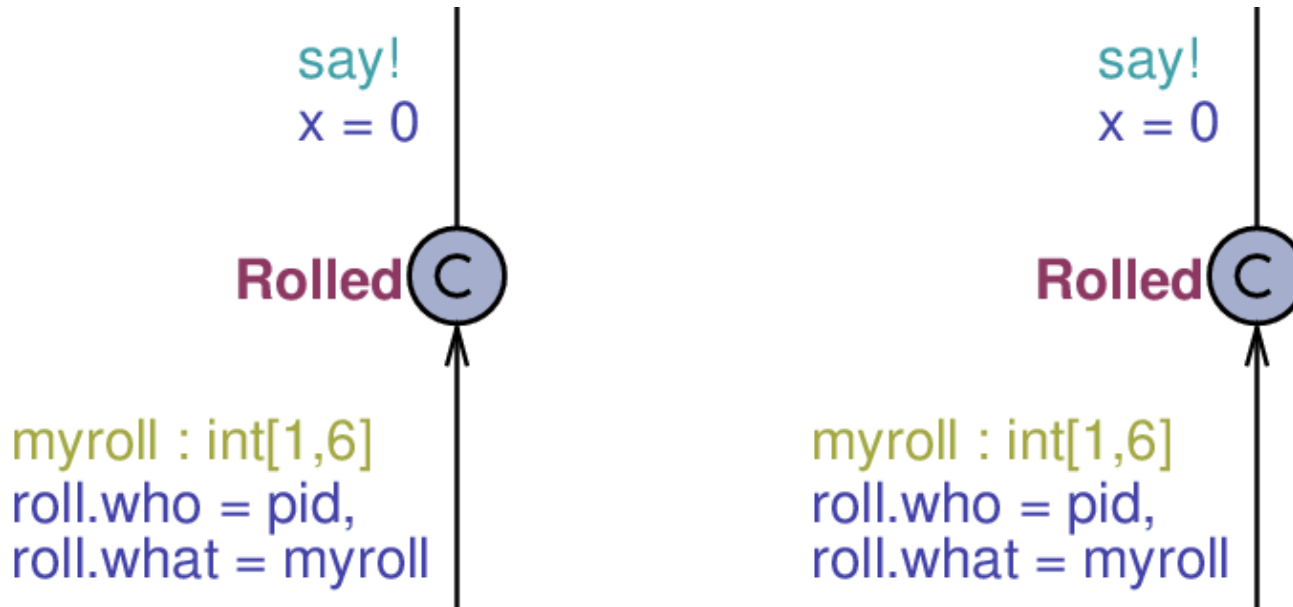
roll.who = 1
roll.what = 3
winner = 0
Player(0).pid = 0
Player(0).count = 0
Player(1).pid = 1
Player(1).count = 0
Player(2).pid = 2
Player(2).count = 0
Referee.ans = 0
Referee.rolls[0] = 0
Referee.rolls[1] = 0
Referee.rolls[2] = 0
Referee.best = 0
Player(0).x >= 0
Player(1).x >= 0
Player(2).x >= 0
Referee.x >= 0
Player(0).x = Player(1).x
Player(1).x = Player(2).x
Player(2).x = Referee.x
Referee.x = Player(0).x
                    
```



Player(0) felülírja a megosztott változót

Player(1) hibásan „küldi el”

Nem kívánt konkurencia elkerülése (dice_roll_1.1)



- **Probléma: Átlapolódhat (konkurens) az egyes játékosokra:**
 - A dobás eredményének rögzítése (**roll** közös változó feltöltése)
 - A bíróval való közlés (**say!** átmenet)
- **Megoldás:**
 - Konkurencia megszüntetése: „**committed**” állapot bevezetése (a „**committed**” állapotból azonnal tovább kell lépnünk)

Speciális lehetőségek (dice_roll_2)

- Csatornatömbök használata

- A fogadó folyamat egy **Select** konstrukcióval „egyszerre” az összes csatornát figyeli
- A csatorna azonosító felhasználható az **Update** szekcióban!

```
pid : id_t  
ans < players  
say[pid]?  
rolls[pid] = roll,  
ans++
```



```
myroll : int[1,6]  
say[pid]!  
roll = myroll
```



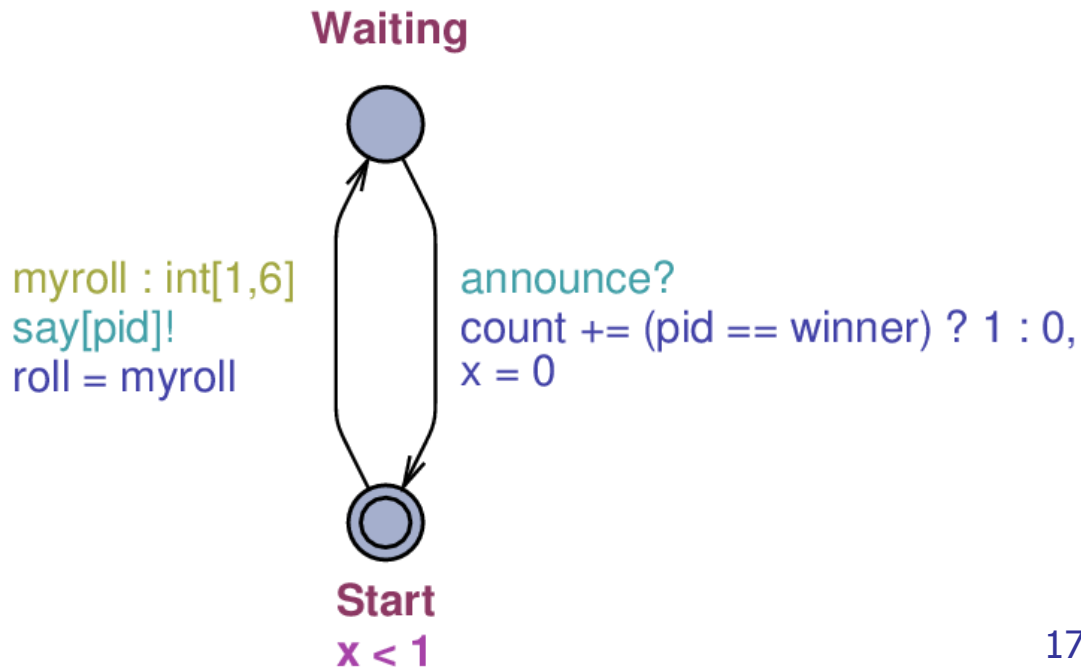
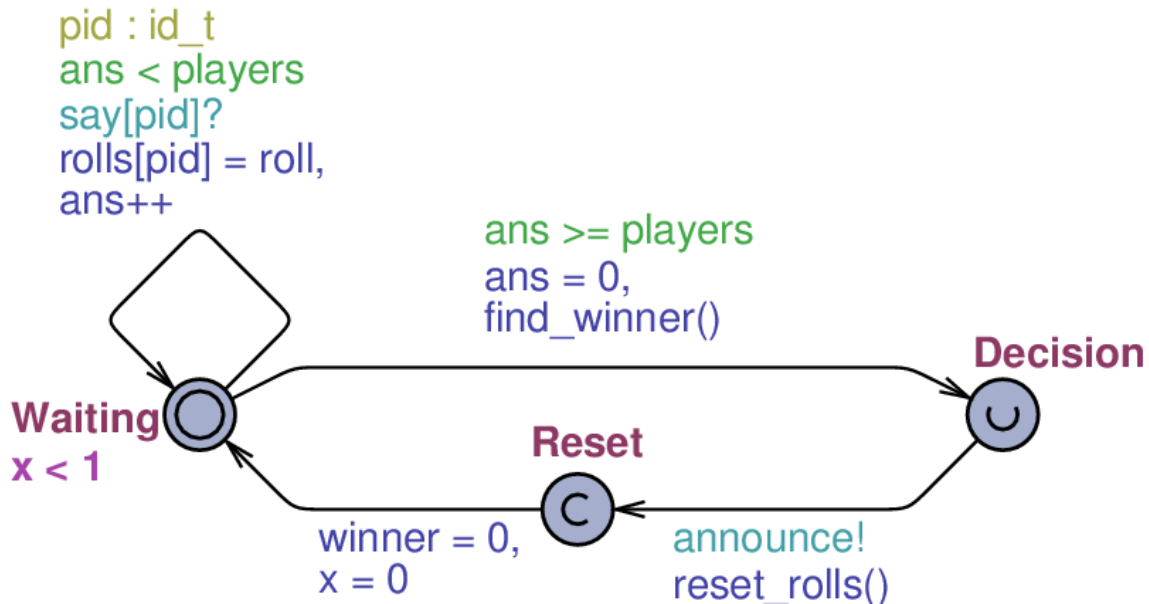
- Iterátorok alkalmazása

```
void reset_rolls() {  
    for (i : id_t) rolls[i] = 0;  
}
```

```
void find_winner() {  
    for (i : id_t) {  
        if (rolls[i] > best) {  
            best = rolls[i];  
            winner = i;  
        }  
    }  
    best = 0;  
}
```

További egyszerűsítési lehetőségek

- Csatornatömbök használata: válaszok gyűjtése egy állapotban
- „?” operátor alkalmazása
- Reset állapot „committed”



További modellezési tippek, tanácsok

- Az élek esetén a szekciók kiértékelési sorrendje:
Select » Sync » Guard » Update
 - Szinkronizáló élek esetén a küldő Update-ja a fogadóé előtt fut le
 - Nem tesztelhetünk a fogadóban egy szinkronizáló él által beállított globális változót (a küldőnél előbb kell beállítanunk, pl. előző élen)
 - Nem „védhetünk meg” Guard-dal egy Sync-ben levő változót (pl. csatornatömb esetén)
- A függvények működésének ellenőrzése nehézkes
 - Nincs lehetőség nyomkövetésre (a belső működés szimulációjával)
 - Próbáljunk a fejlesztés során kis lépésekben haladni, és szimulációval, verifikációval gyakran ellenőrizni

További modellezési tippek, tanácsok

- Az $A \leftrightarrow q$ tulajdonság használata esetén ki kell zárni a triviális ellenpéldát
 - Óraváltozók használatával
 - „Urgent” állapot beállításával
 - Emlékezzünk a „leads to” $p \rightarrow q$ szemantikájára: $A[]$ ($p \text{ imply } A \leftrightarrow q$)
- Ne felejtsük el az óraváltozókat megfelelően inicializálni
- A csatorna-, vagy automataszintű prioritások használata
 - Az UPPAAL modellellenőrzője nem támogatja (pl. a holtponthoz sem tudja ellenőrizni)
 - Ezek a modellezési elemek lehetőleg kerülendők