

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

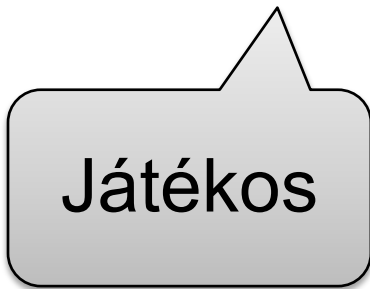
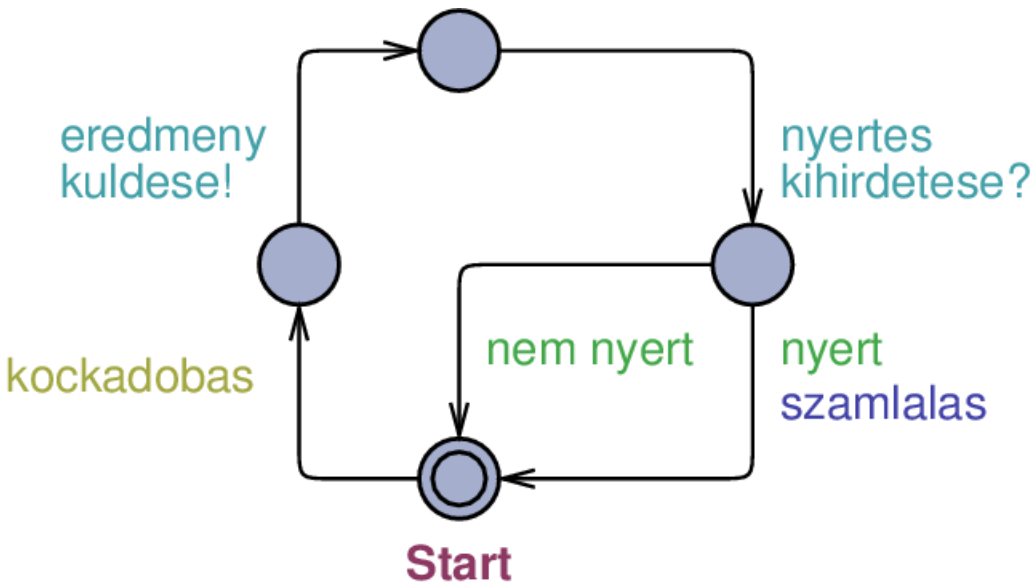
A feladat

- 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

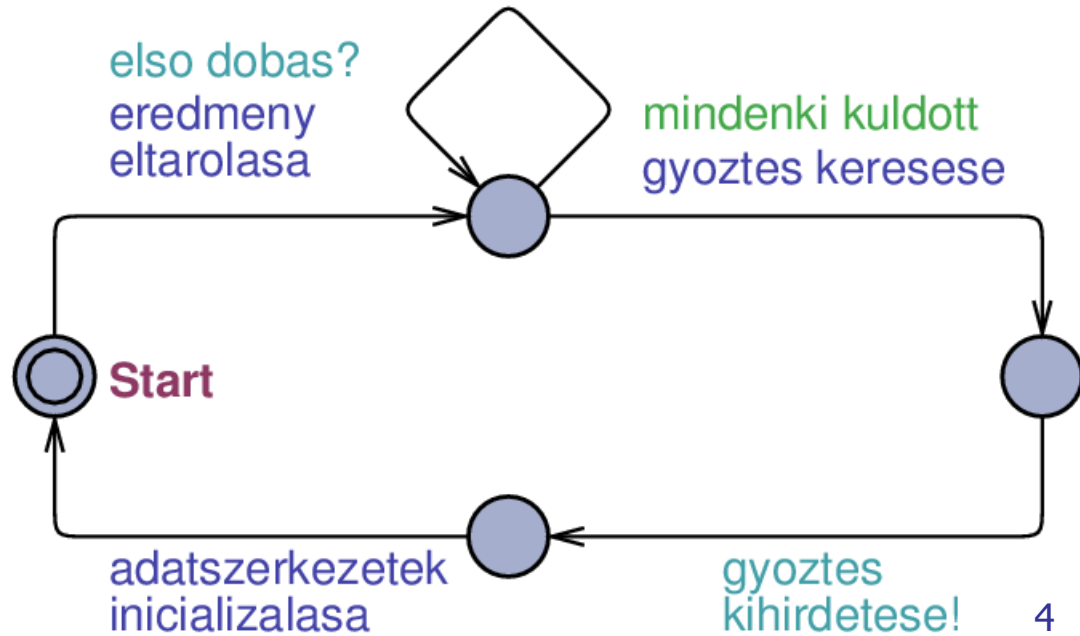
Mit kell megoldanunk?

- Kockadobás
 - Véletlen érték generálása
- Eredményközlés és hirdetés
 - Értékek „továbbítása”
 - „Broadcast” kommunikáció
 - Csatornatömbök kezelése
- Eredmények feljegyzése
 - Adatszerkezetek
- Győztes keresése
 - Függvények

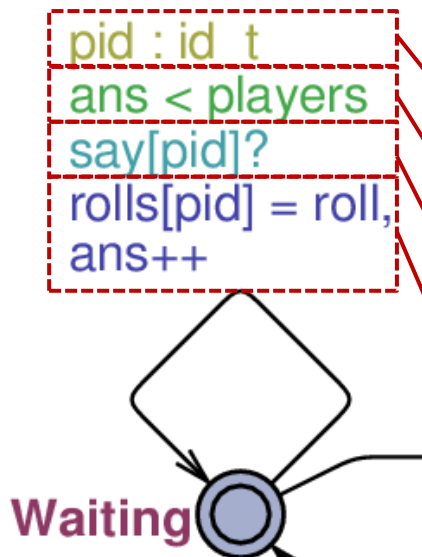
Szereplők és állapotok absztrakt modellje



amíg mindenki el nem kuli:
következő dobás?
eredmény eltarolasa



Modellezési kitérő: az élekhez rendelt kifejezések



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

The 'Edit Edge' dialog box has two tabs: 'Edge' and 'Comments'. The 'Edge' tab is active and contains four text input fields:

- Select:** pid : id t
- Guard:** ans < players
- Sync:** say[pid]?
- Update:** rolls[pid] = roll,
ans++

At the bottom of the dialog are 'OK' and 'Cancel' buttons.

- Selection

- Nemdeterminisztikus választás egy változó értékkészletéből

- 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;
```

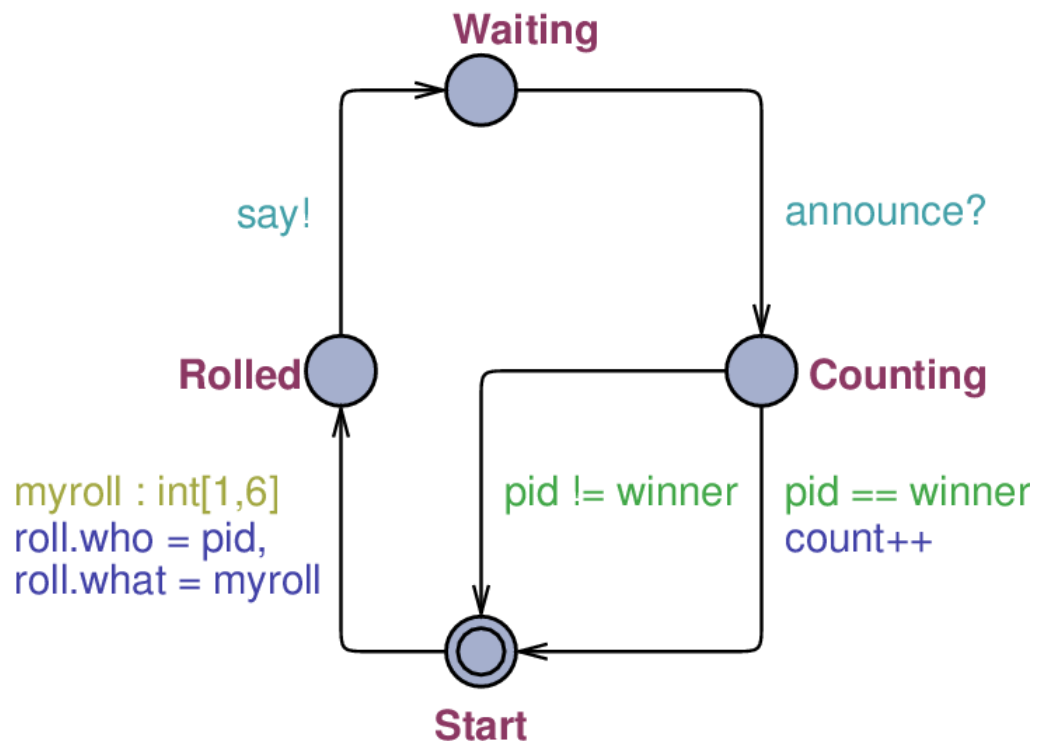
```
chan say;
```

```
broadcast chan announce;
```

Játékos (Player):

```
Player(id_t pid)
```

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



Modellezés: A bíró

Bíró (Referee):

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

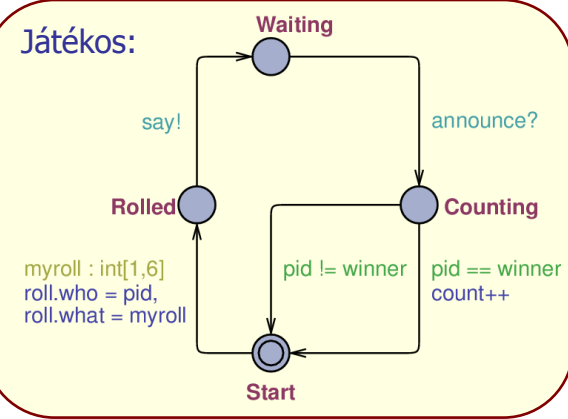
```
void find_winner() {
    int[0,players] i;

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

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

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

Játékos:



ans < players

say?

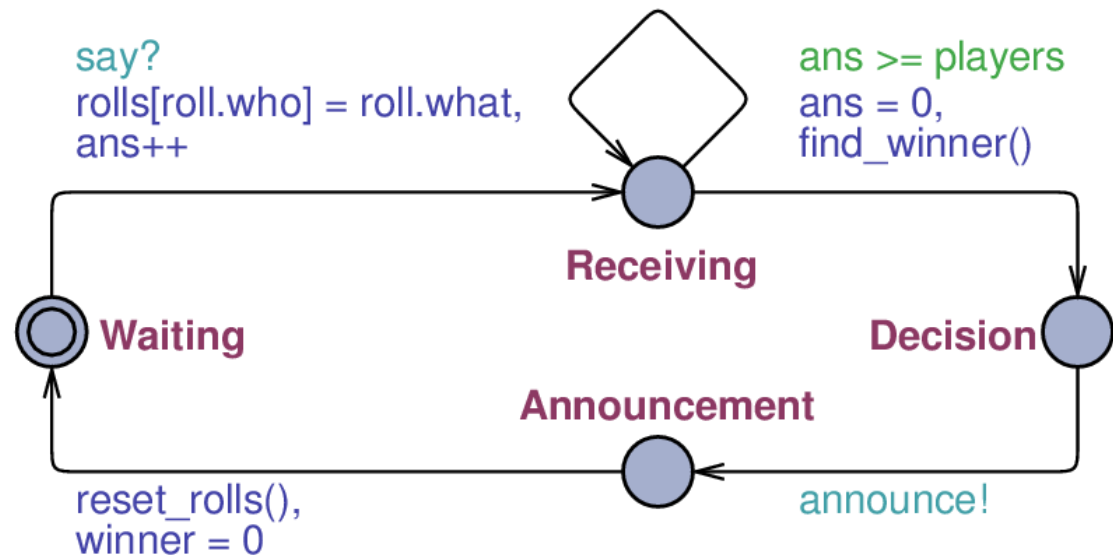
rolls[roll.who] = roll.what,
ans++

say?

rolls[roll.who] = roll.what,
ans++

ans >= players

ans = 0,
find_winner()



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 holtpon t
 - 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)

The screenshot shows the UPPAAL model checker interface. The main window displays a list of queries and their status. The first query, `A<> exists (i : id_t) (Player(i).count == wins)`, is highlighted in blue and has a red lightbulb icon next to it, indicating it is not satisfied. Below it, two other queries are shown with green lightbulb icons, indicating they are satisfied. The status bar at the bottom shows `A[] not deadlock`. A callout box on the right contains the text: "Holtpontmentesség: nem fut le. • Mert nem akadályoztuk meg a nyerési számlálók túlfutását (ld. count típusa és count++). • (Most nem javítjuk.)". A red dashed arrow points from the callout box to the first query in the list.

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

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 with the following sections:

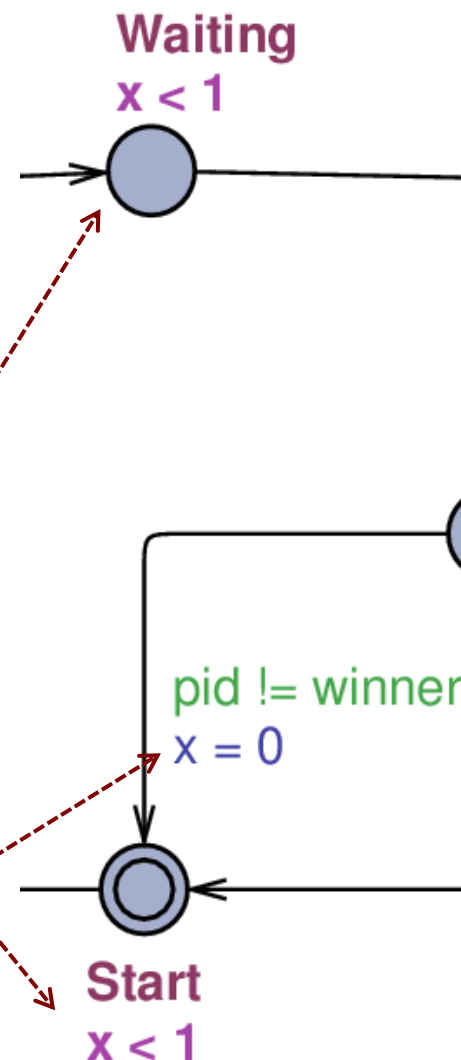
- Overview:** A list of queries with status indicators (red, green, grey) and control buttons (Check, Insert, Remove, Comments). The first query is highlighted in blue: `A<> exists (i : id_t) (Player(i).count == wins)`. A red dashed arrow points from this query to the status section below.
- Query:** The selected query: `A<> exists (i : id_t) (Player(i).count == wins)`.
- Comment:** An empty text area.
- Status:** A log of verification results. A red dashed arrow points from the status section back to the query in the Overview section. The log contains:
 - `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.

Ellenpélda: Van olyan útvonal, ahol nem következnek be mindenki dobásának feljegyezésével döntés

- Triviális ellenpélda: időbeliség
- Konkurencia helytelen kezelése

Triviális ellenpélda kiküszöbölése: időzítés

- Ha az összes lehetséges lefutást vizsgáljuk (pl. $A \langle \rangle$), akkor az UPPAAL figyelembe veszi azt a lehetőséget is, hogy egy állapotot sosem hagyunk el
- Megoldás:
 - Óraváltozó bevezetése
 - Vezérlési hely invariáns előírása
 - Legfeljebb 1 időegységig tartózkodhat az adott állapotban
 - A vezérlési helybe lépő éleken az óraváltozót nullázni kell



Konkurens működés - mi a baj?

Player(1) kockával dobott

Player(0) most fog dobni

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)

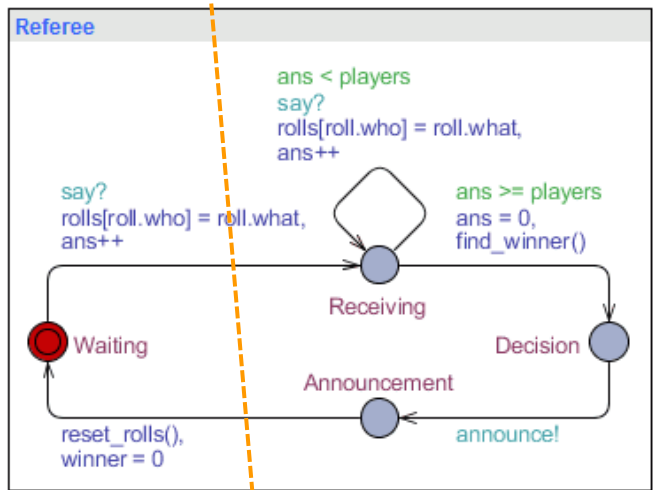
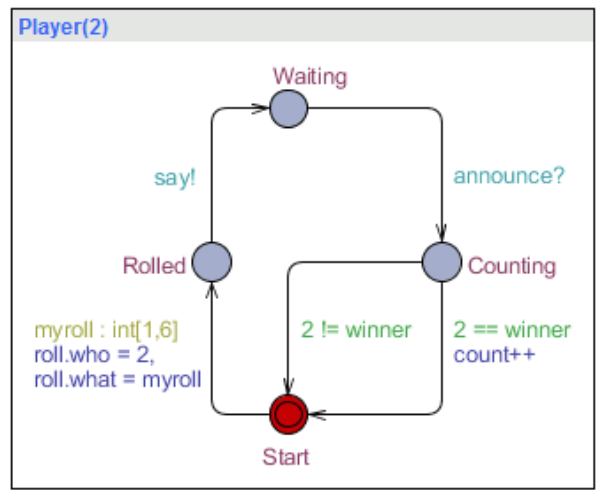
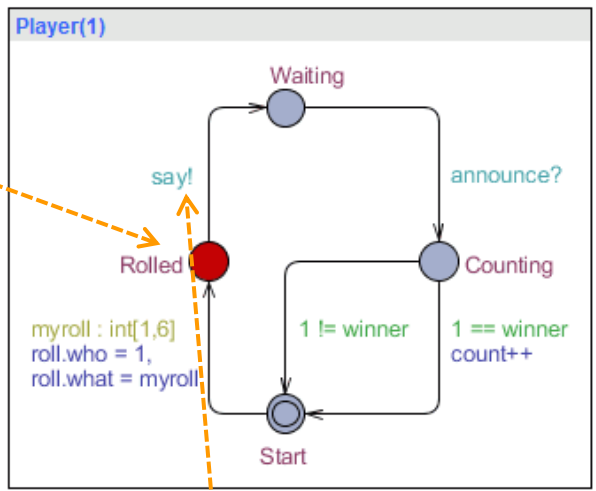
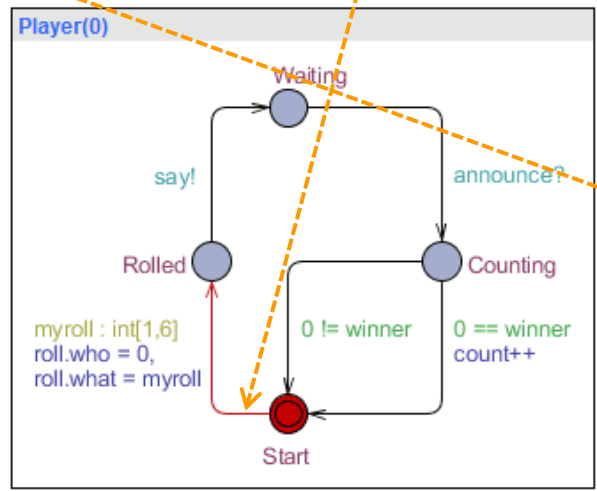
(Start, Rolled, Start, Waiting)

Player(0)

(Rolled, Rolled, Start, Waiting)

```

roll.who = 1
roll.what = 5
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
                    
```

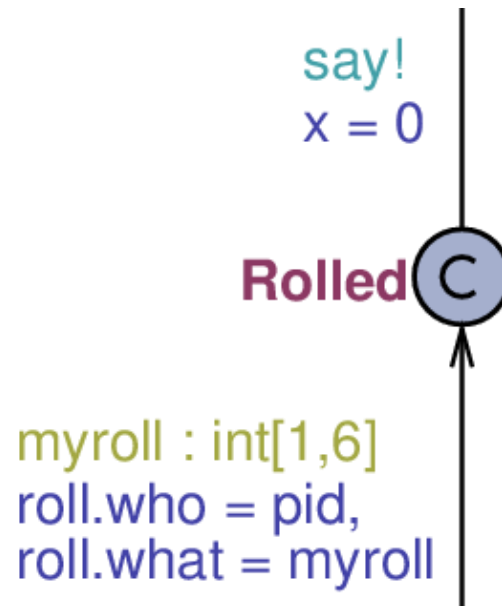
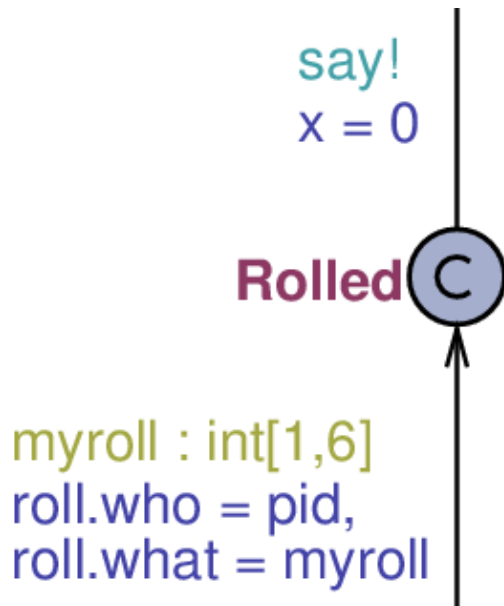


Player(0) felülírja majd a megosztott változót

Player(1) másét „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 az automata azonnal továbblép: itt az eredmény rögzítése és a közlés egyben történik



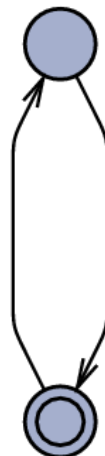
Speciális lehetőségek (dice_roll_2)

- Csatornatömbök használata a közös **say** csatorna helyett
 - A bíró egy **Select** konstrukcióval választja ki a csatornát (modellellenőrzés: az összes lehetőséget 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
```



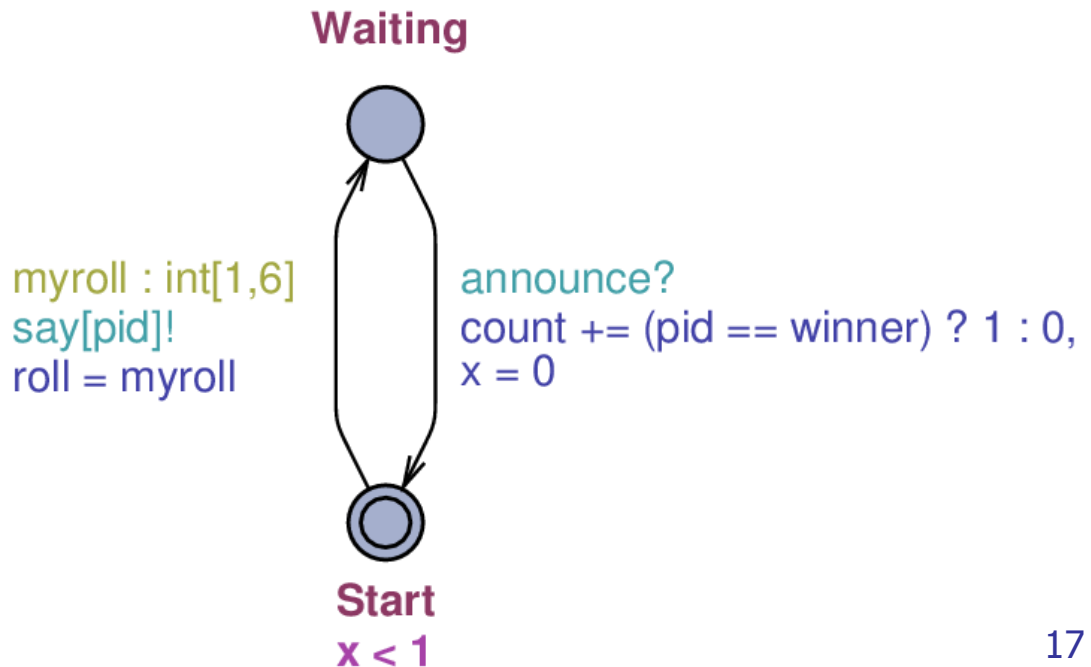
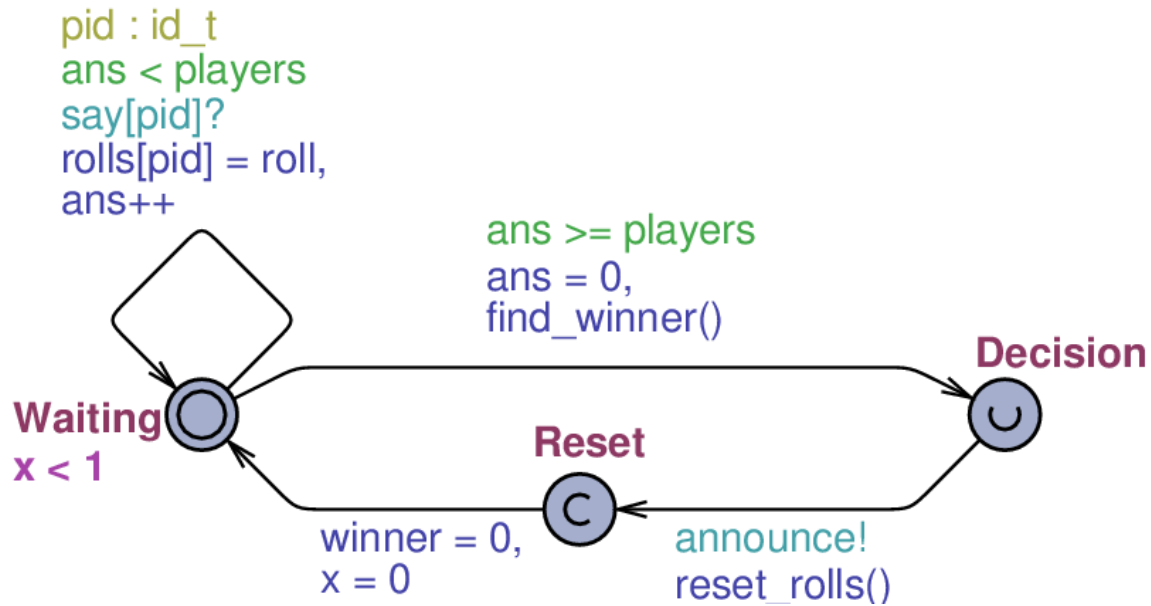
- Iterátorok alkalmazása `for (i = 0; i < players; i++)` helyett (rövidebb)

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

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

További egyszerűsítési lehetőségek

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



További modellezési tippek, tanácsok

- Az élek esetén a szekciók kiértékelési sorrendje
 - Szinkronizáló élek esetén a küldő **Update**-ja a fogadóé előtt fut le, de a fogadó **Guard** feltételének kiértékelése után
 - Nem vizsgálhatunk a fogadó **Guard** feltételében egy szinkronizáló él **Update**-ja által beállított globális változót (a küldőnél előbb kell beállítanunk, pl. előző élen)
- 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 $A \langle \rangle q$ tulajdonság használata esetén ki kell zárnunk a triviális ellenpéldát
 - Óraváltozók használatával
 - „Urgent” állapotok beállításával
 - $A \ p \ \rightarrow \ q$ „leads to” része az $A \langle \rangle$, ld. $A[]$ ($p \ \text{imply} \ A \langle \rangle \ q$)
- Vezérlési hely invariánsok:
 - Belépő éleken ne felejtsük el az óraváltozókat inicializálni (nullázni)
- A csatorna-, vagy automataszintű prioritások használata
 - Az UPPAAL modellellenőrzője nem támogatja (pl. a holtponthoz sem tudja ellenőrizni)
 - Ezek a modellezési elemek lehetőleg kerülendők