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 kockajáték feladat

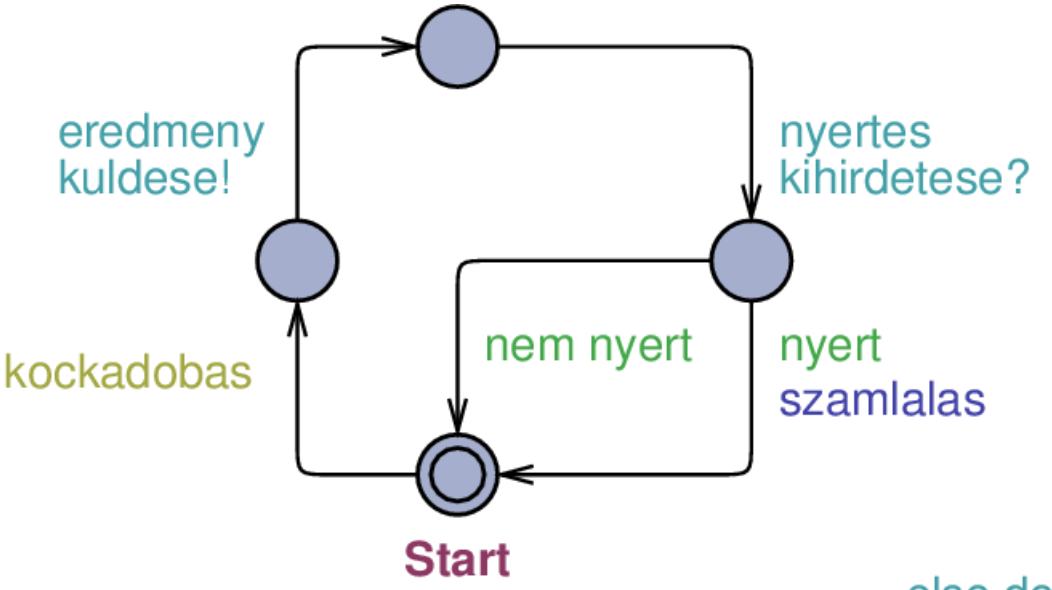
A feladat

- A 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 kör nyertesét
  - A játékosok számlálják a nyeréseiket
  - Győztes: 10 körben nyer

Mit kell megoldanunk?

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

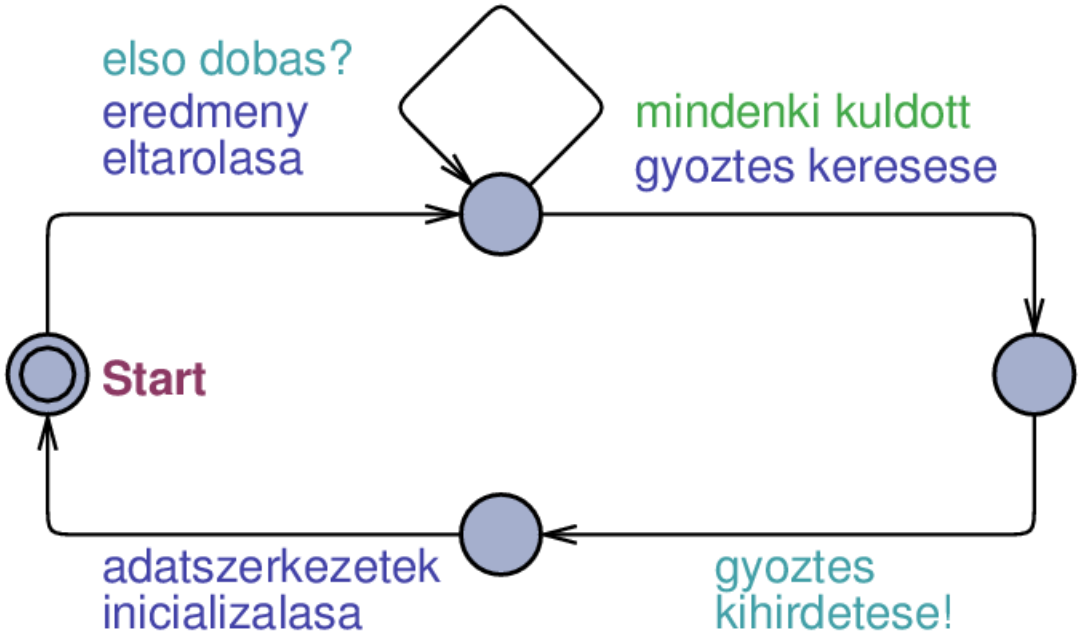
Szereplők és állapotok absztrakt modellje



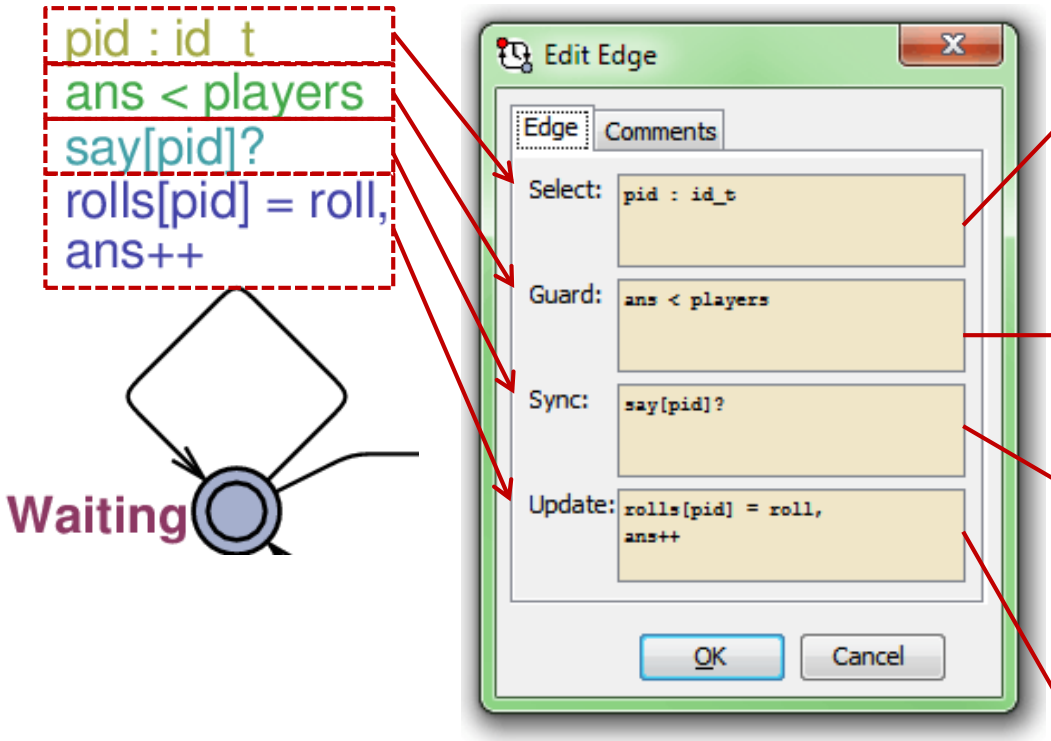
Bíró

amig mindenki el nem kuldi:
kovetkezo dobas?
eredmeny eltarolasa

Játékos



Modellezési kitérő: az élekhez rendelt kifejezések



- 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

Globális deklarációk:

```
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;
```

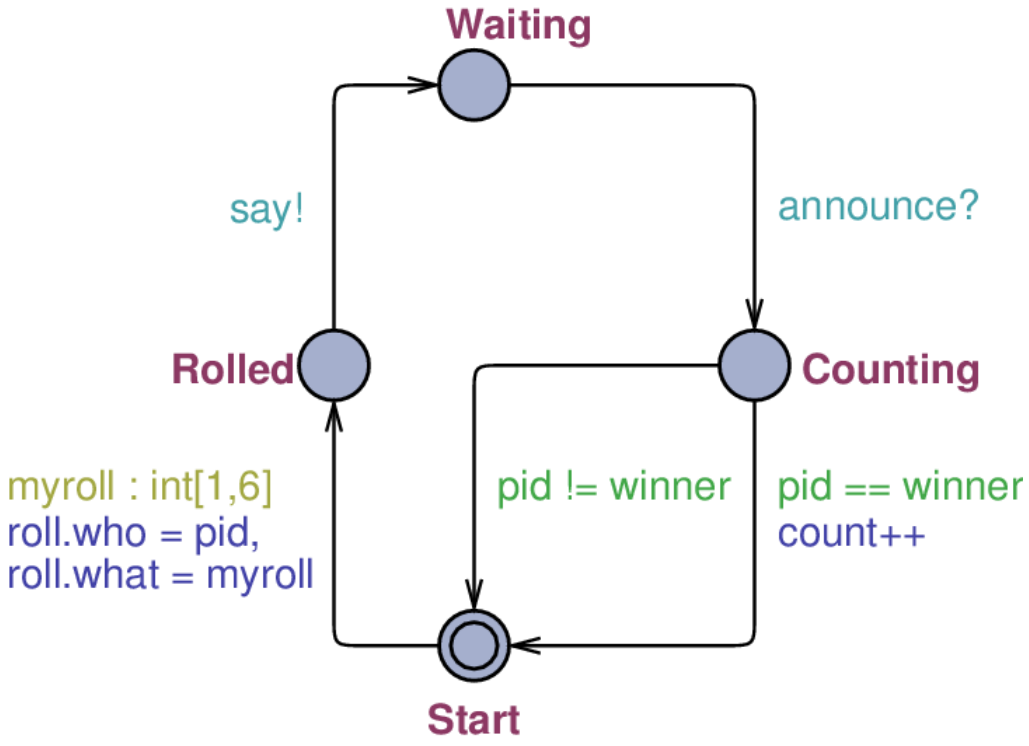
Rendszer:

```
system Referee, Player;
```

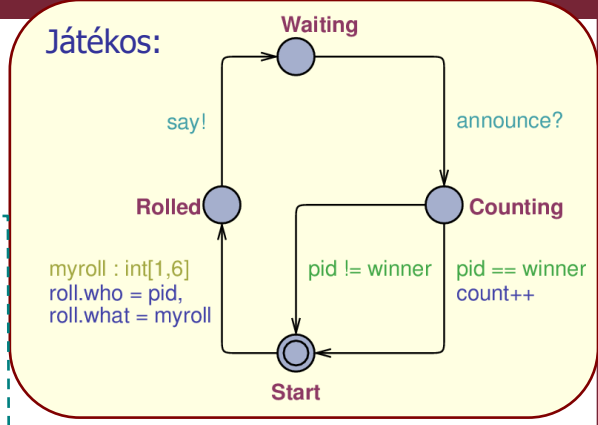
Játékos (Player):

```
Player(id_t  pid)

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



Modellezés: A bíró



Bíró (Referee):

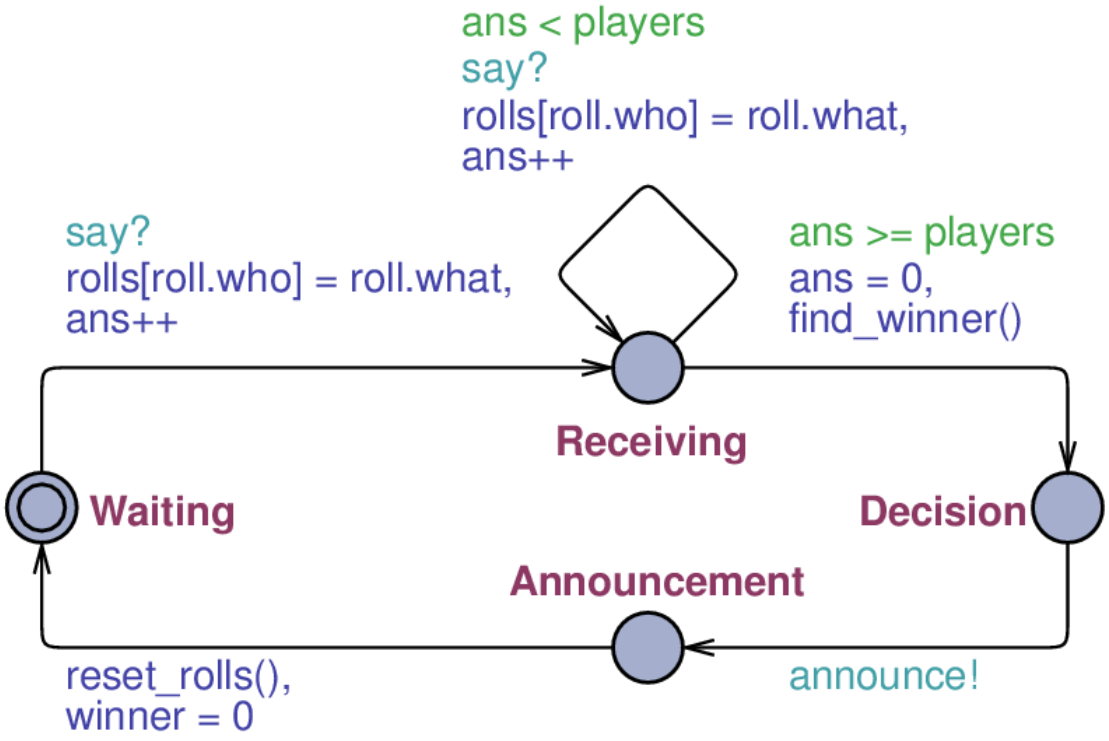
```
int [0,players] ans = 0;
dice_t rolls[id_t];
dice_t best = 0;

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;
}
```



Ellenőrizzük a működést! (dice_roll_1)

- Mindig van győztes a játékban
  - Mindig van olyan játékos, aki maximális számú alkalommal nyer

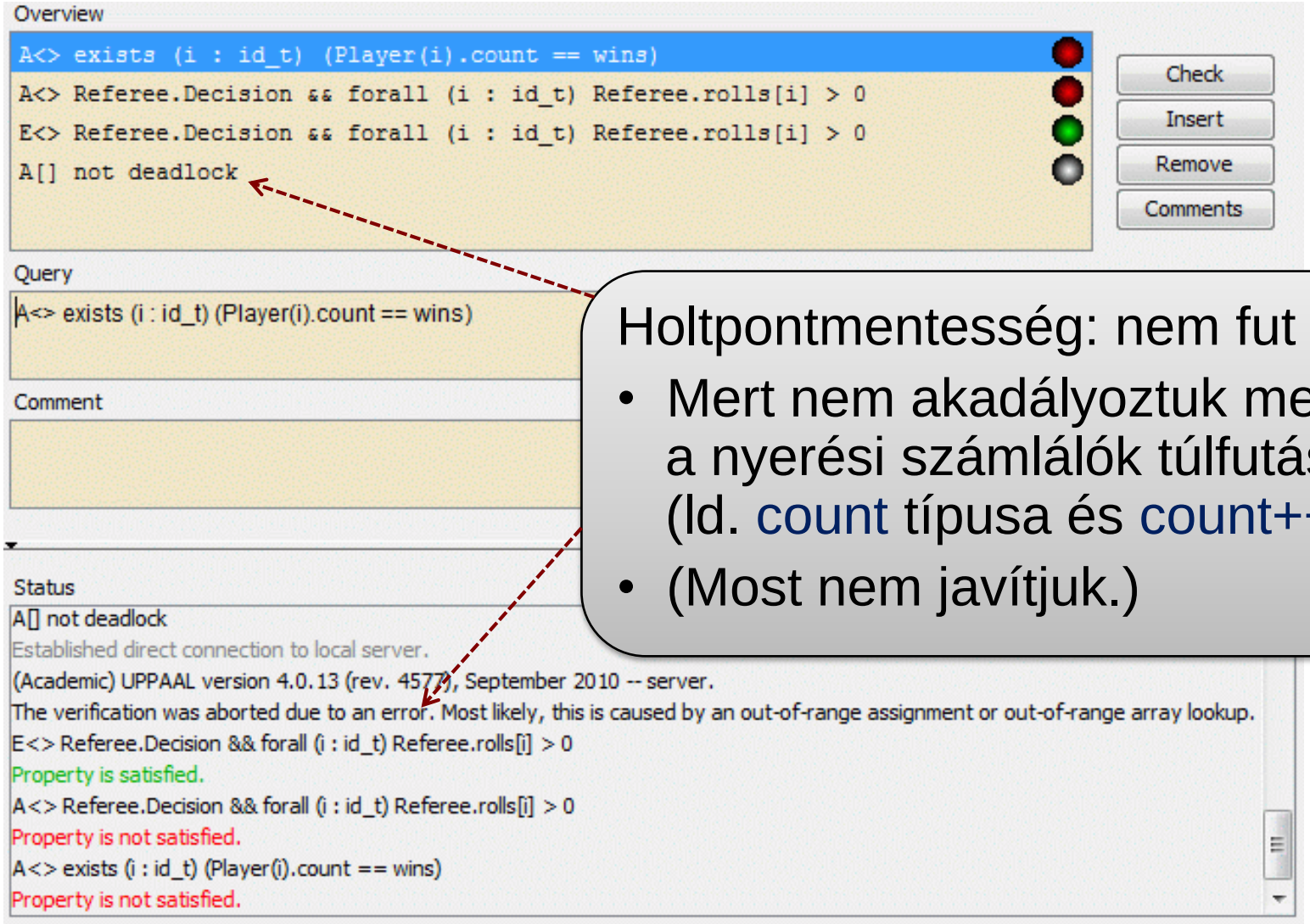
    A<> exists (i : id_t) (Player(i).count == wins)
- A bíró akkor hoz döntést, ha minden játékos dobott
  - Ez legalább egyszer megtörténik:

    E<> Referee.Decision && forall (i : id_t) (Referee.rolls[i] > 0)
  - Ez minden lehetséges úton bekövetkezik:

    A<> Referee.Decision && forall (i : id_t) (Referee.rolls[i] > 0)
- A rendszerben nincs holtpont
  - Nincs olyan állapot, amelyből ne vezetne ki olyan engedélyezett állapotátmenet, amellyel egy következő állapotba léphetünk

    A[] not deadlock

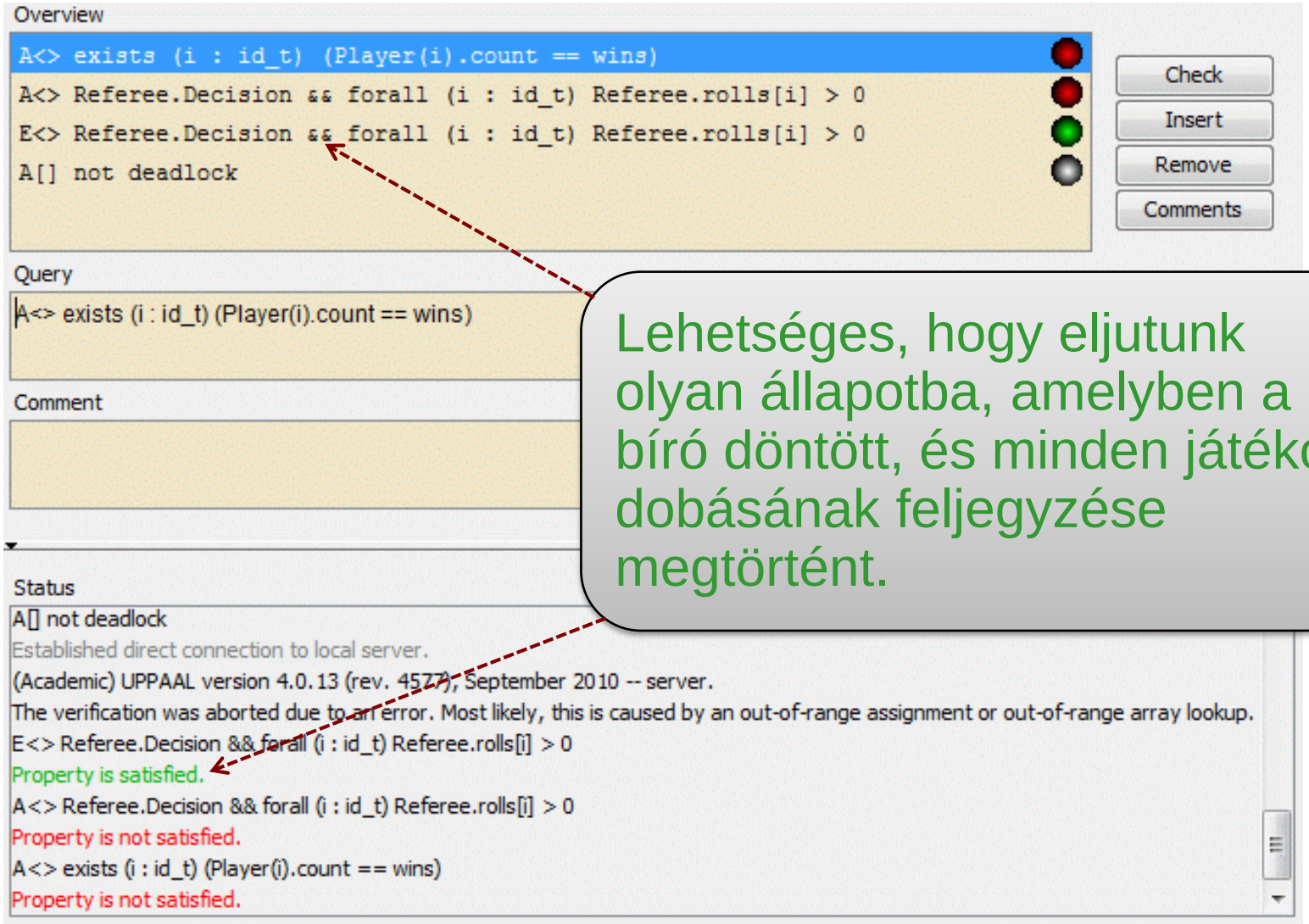
Ellenőrizzük a működést! (dice_roll_1)



Holtpontmentesség: nem fut le.

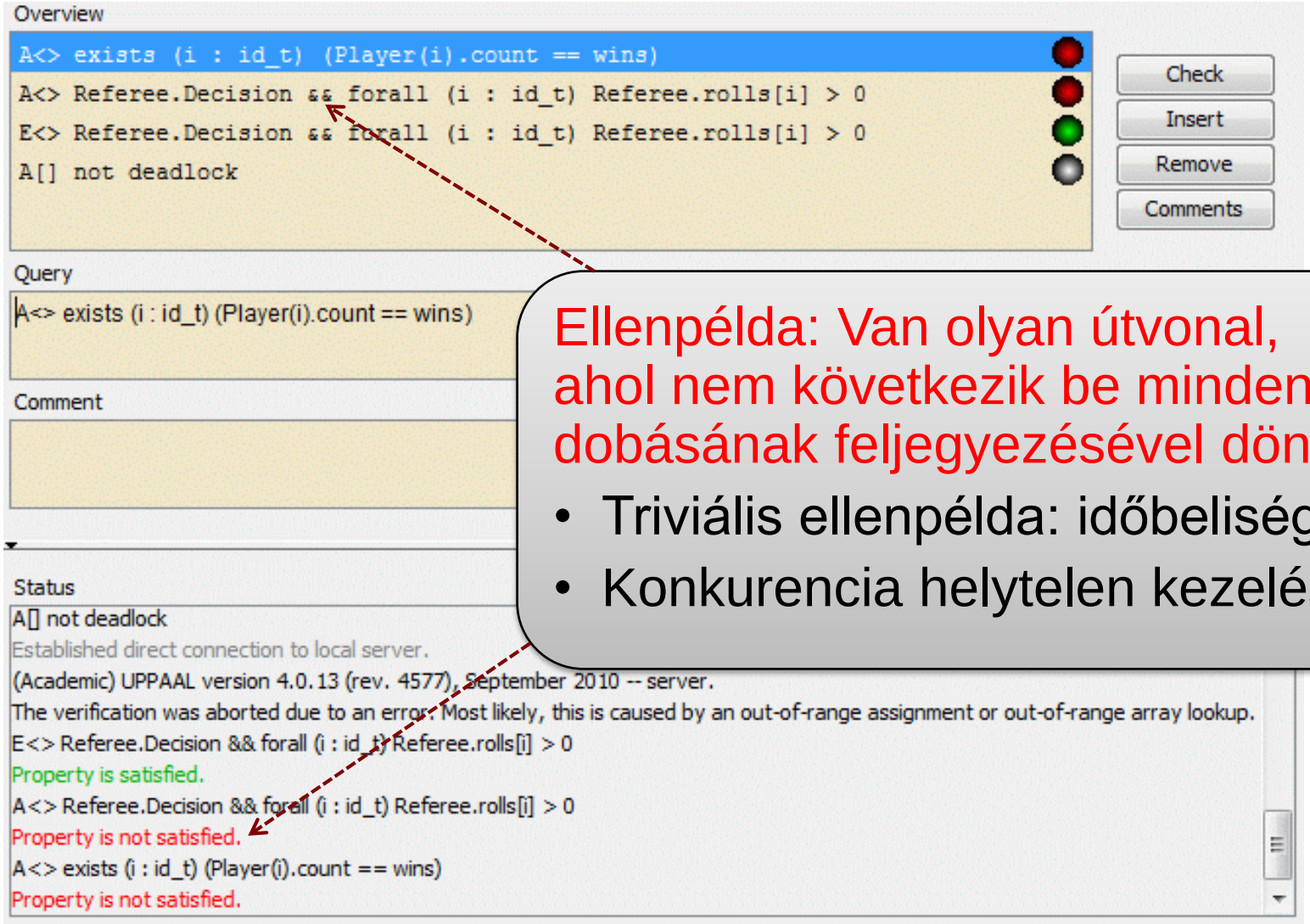
- Mert nem akadályoztuk meg a nyerési számlálók túlfutását (ld. count típusa és count++).
- (Most nem javítjuk.)

Ellenőrizzük a működést! (dice_roll_1)



Lehetséges, hogy eljutunk olyan állapotba, amelyben a bíró döntött, és minden játékos dobásának feljegyzése megtörtént.

Ellenőrizzük a működést! (dice_roll_1)

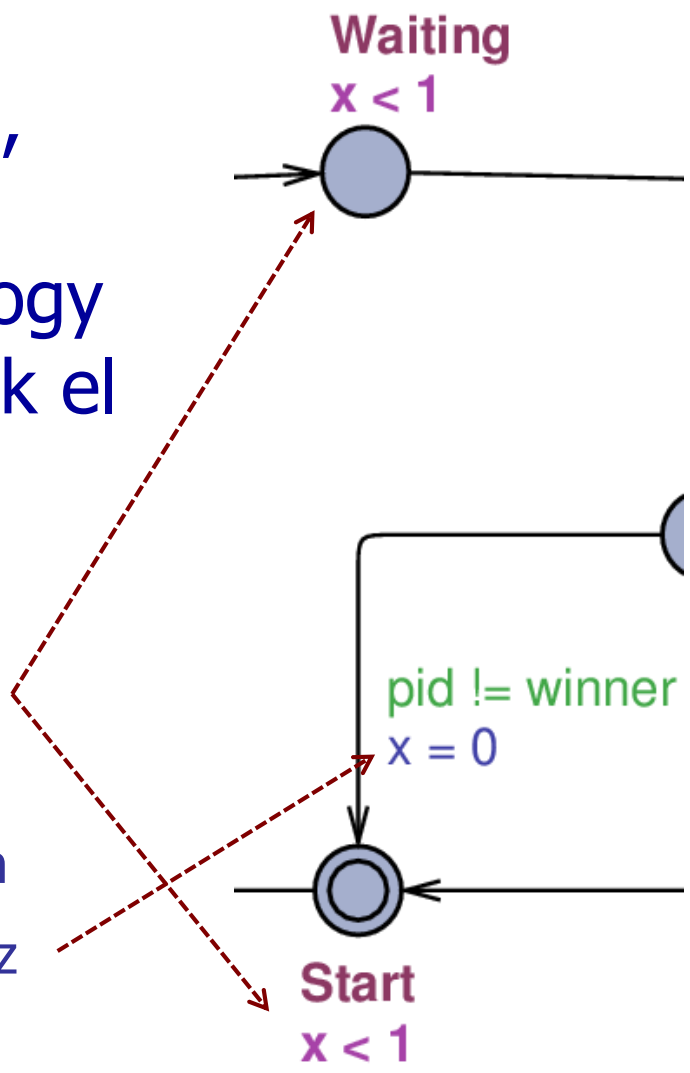


Ellenpélda: Van olyan útvonal, ahol nem következik be mindenki dobásának feljegyezésével döntés

- Triviális ellenpélda: időbeliség
- Konkurencia helytelen kezelése

Triviális ellenpélda kiküszöbölése: időzítés

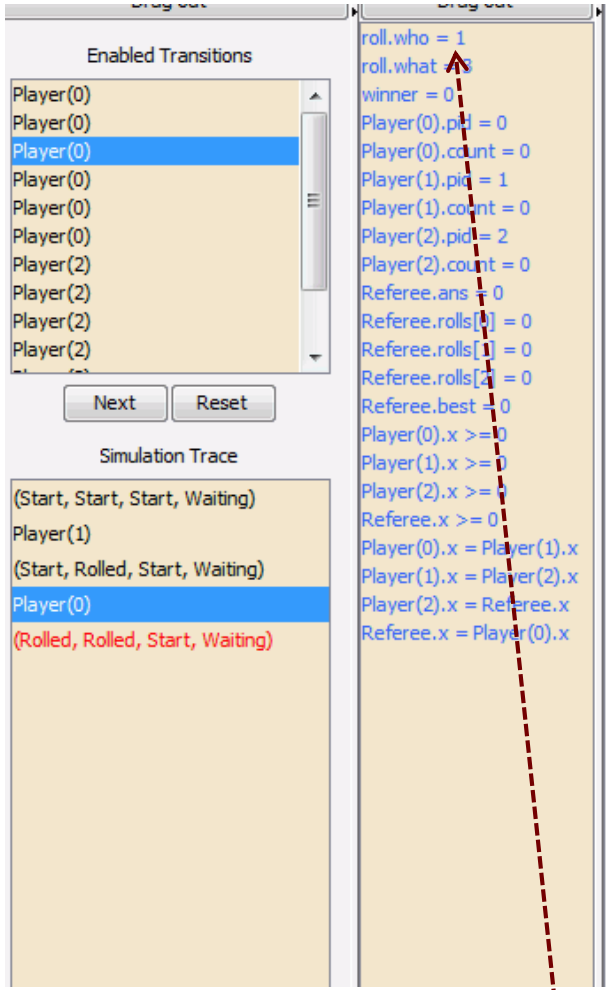
- Ha az összes lehetséges lefutást vizsgáljuk (pl. A<>), akkor az UPPAAL figyelembe veszi azt a lehetőséget is, hogy egy állapotot sosem hagyunk el
- Megoldás:
  - Óraváltozó bevezetése
    - Vezérlési hely invariáns előírása
    - Legfeljebb 1 időegységig tartózkodhat az adott állapotban
    - A vezérlési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

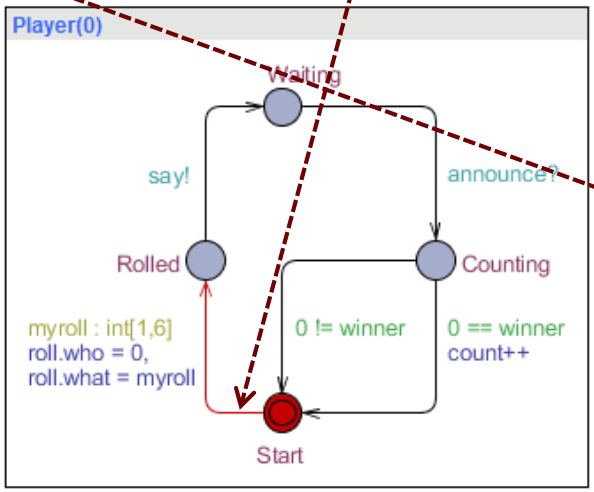


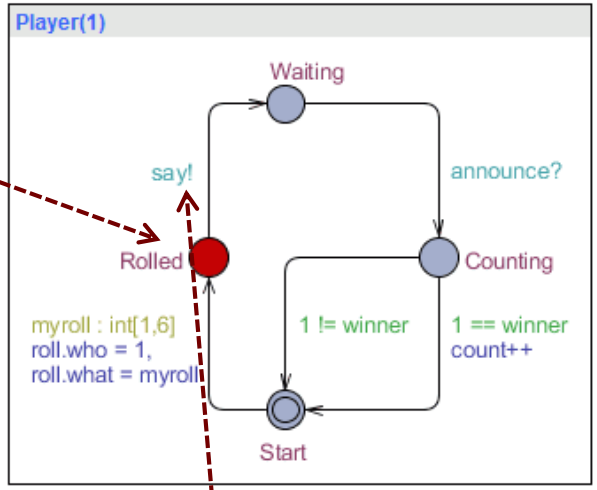
Konkurens működés - mi a baj?

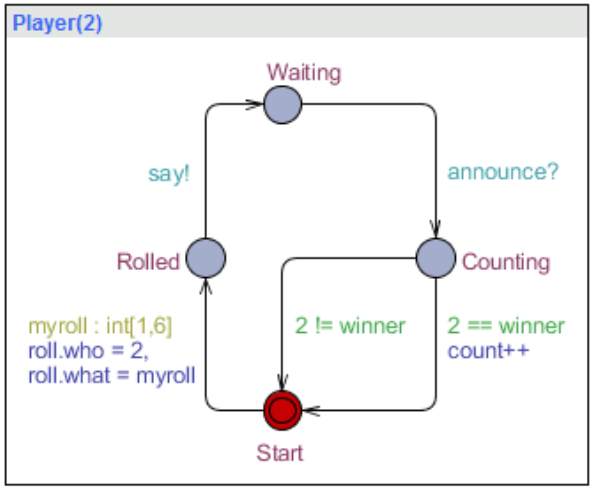
Player(1) kockával dobott

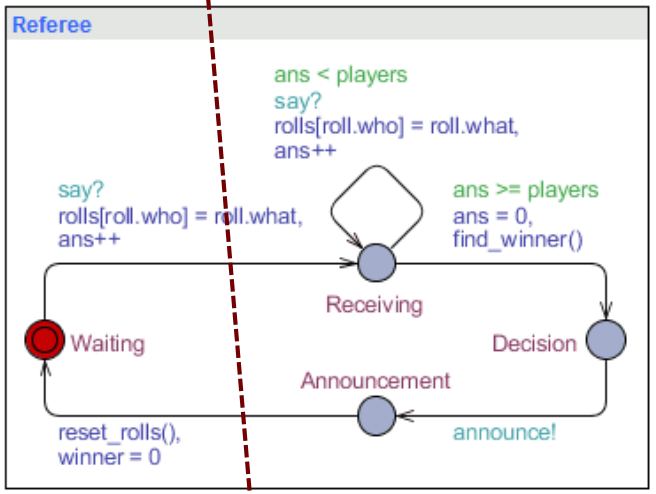
Player(0) most fog dobni









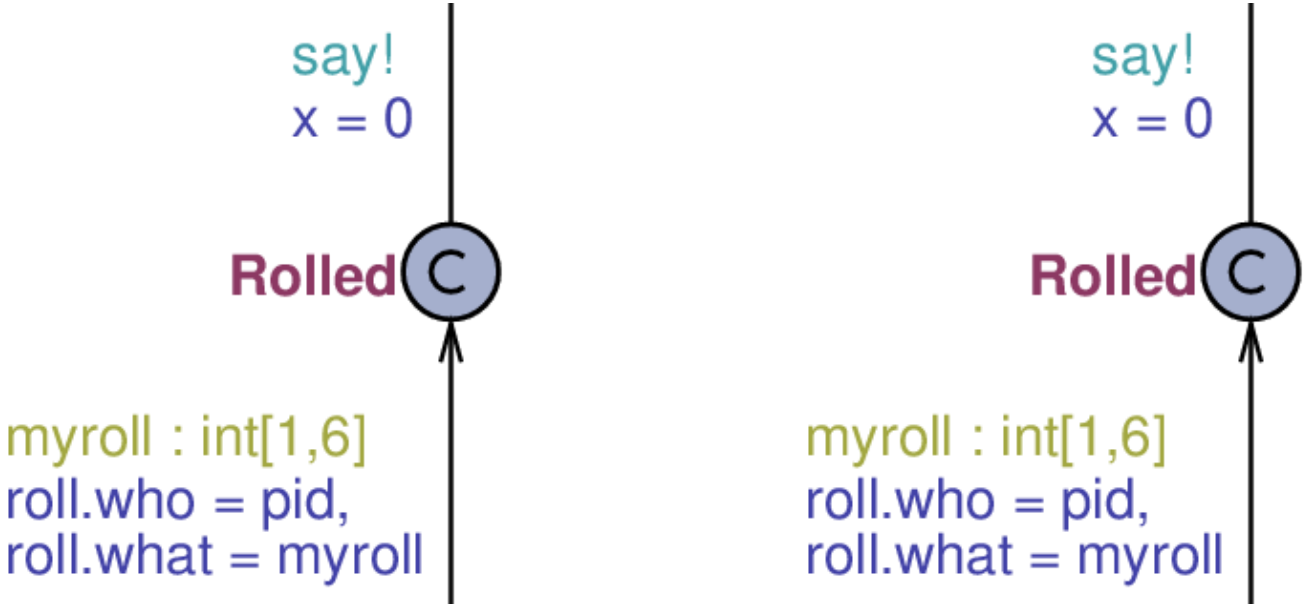


Player(0) felülírja majd a megosztott változót

Player(1) másét „küldi el”

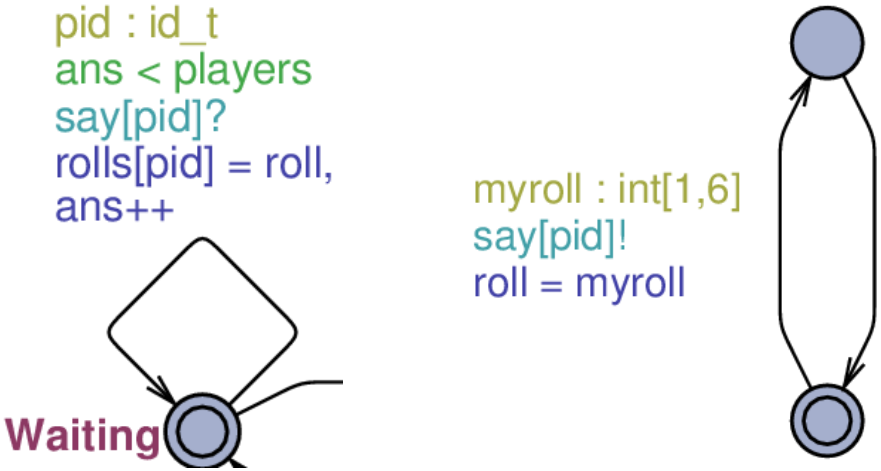
Nem kívánt konkurencia elkerülése (dice_roll_1.1)

- 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!



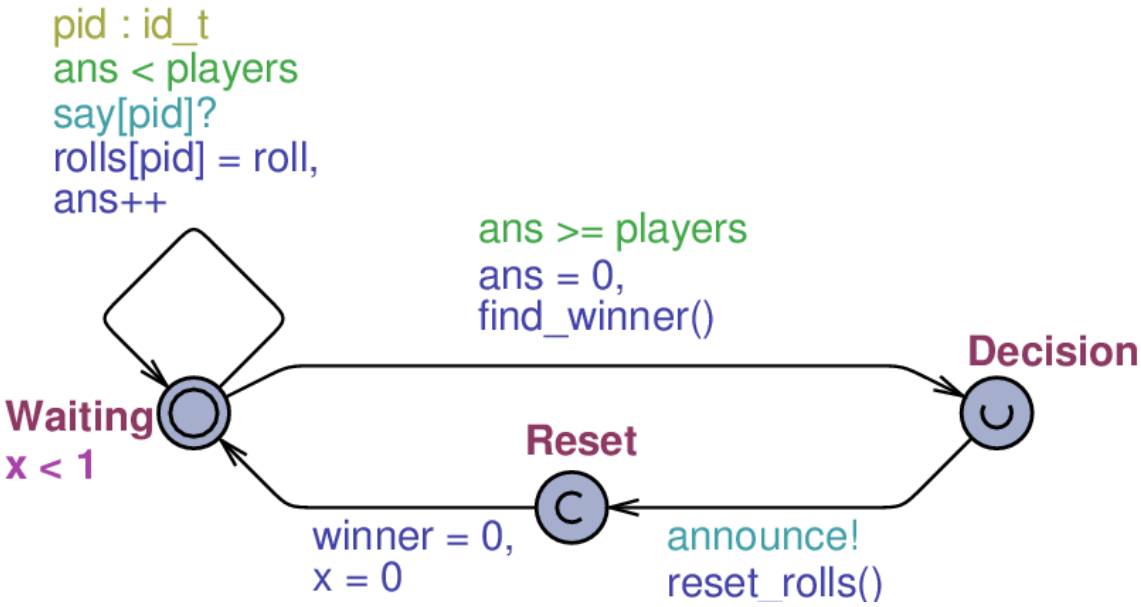
- Iterátorok alkalmazása
  for (i = 0; i < players; i++)
  helyett rövidebben:

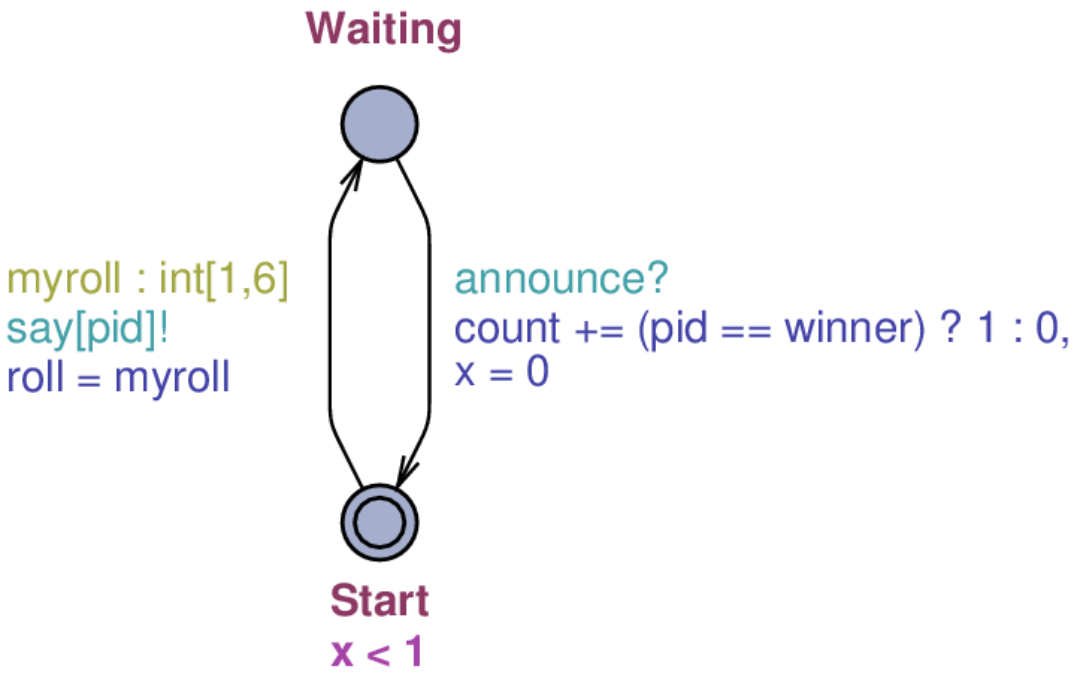
```
void reset_rolls() {
  for (i : id_t) {rolls[i] = 0;}
}

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 élen a küldő 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<> q tulajdonság használata esetén ki kell zárnunk a triviális ellenpéldát (adott állapotban marad végtelen ideig)
  - Óraváltozók használatával (vezérlési hely invariáns)
  - Urgent állapotok beállításával
  - A p --› q „leads to” része az A<>, lásd A[] (p imply A<> 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 holtpontot sem tudja ellenőrizni)
  - Ezek a modellezési elemek lehetőleg kerülendők