Modellezés UPPAAL-ban

Kockajáték mintafeladat és megoldása

dr. Bartha Tamás

A feladat

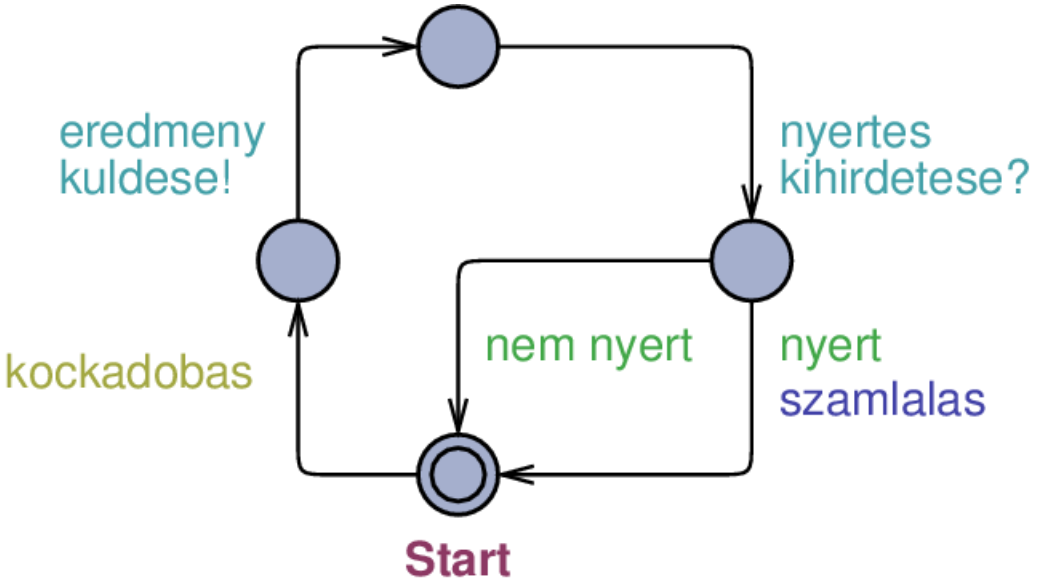
A feladat

- 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

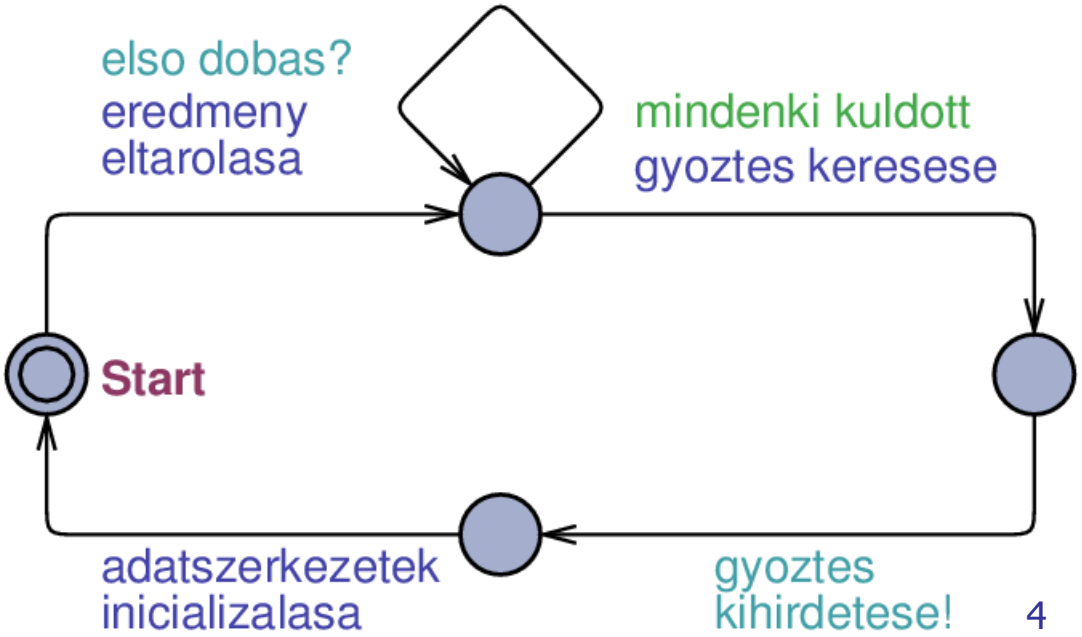
Szereplők és állapotok



Játékos

Bíró

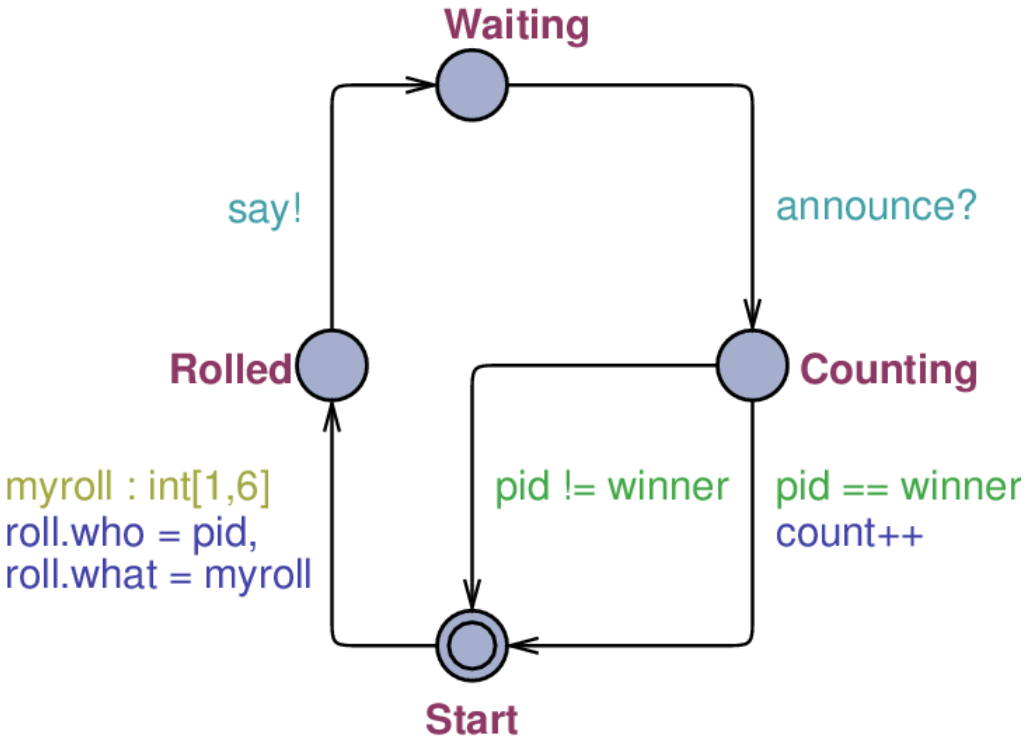
amig mindenki el nem kuldi:
kovetkezo dobas?
eredmeny eltarolasa



Modellezés: A rendszer és egy játékos

Rendszer:

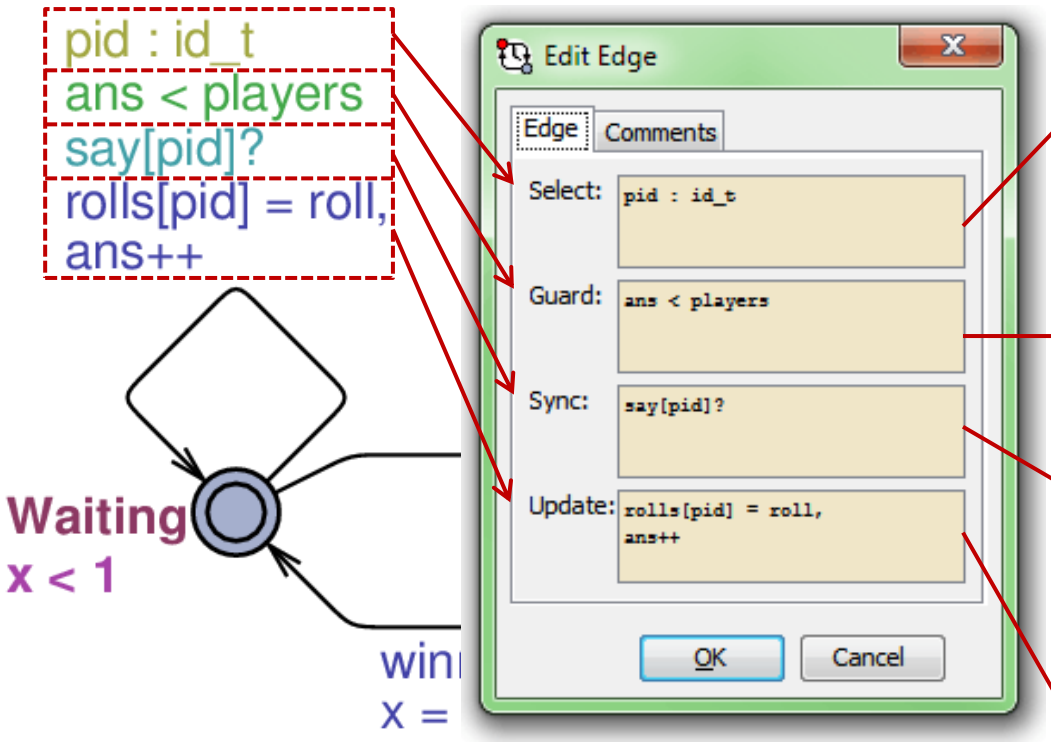
```
system Player, Referee;

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;
clock x;
```

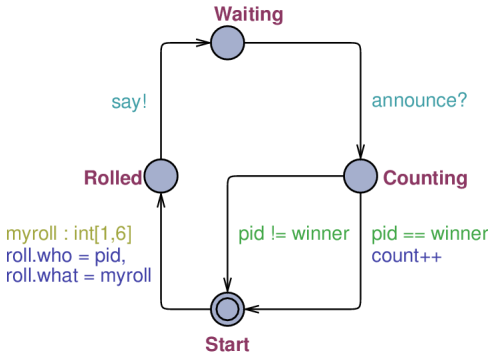
Kitérő: az élekhez rendelt kifejezések



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

- 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)

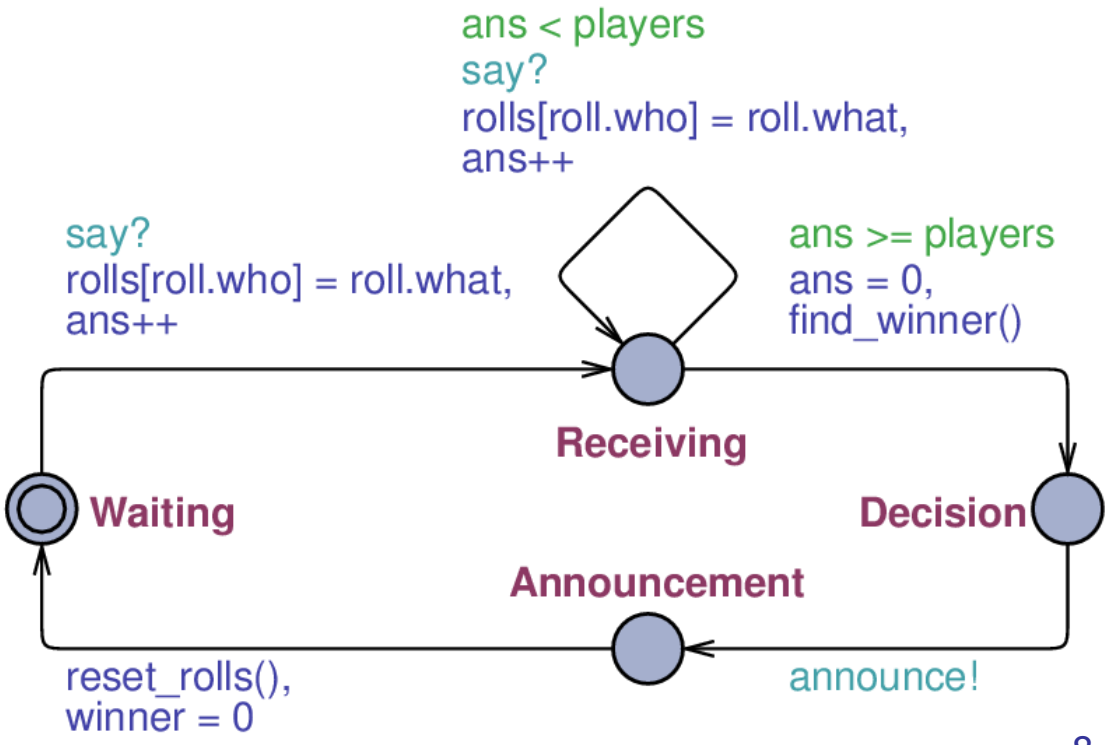
Modellezés: A bíró



Bíró:

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

clock x;

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!

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

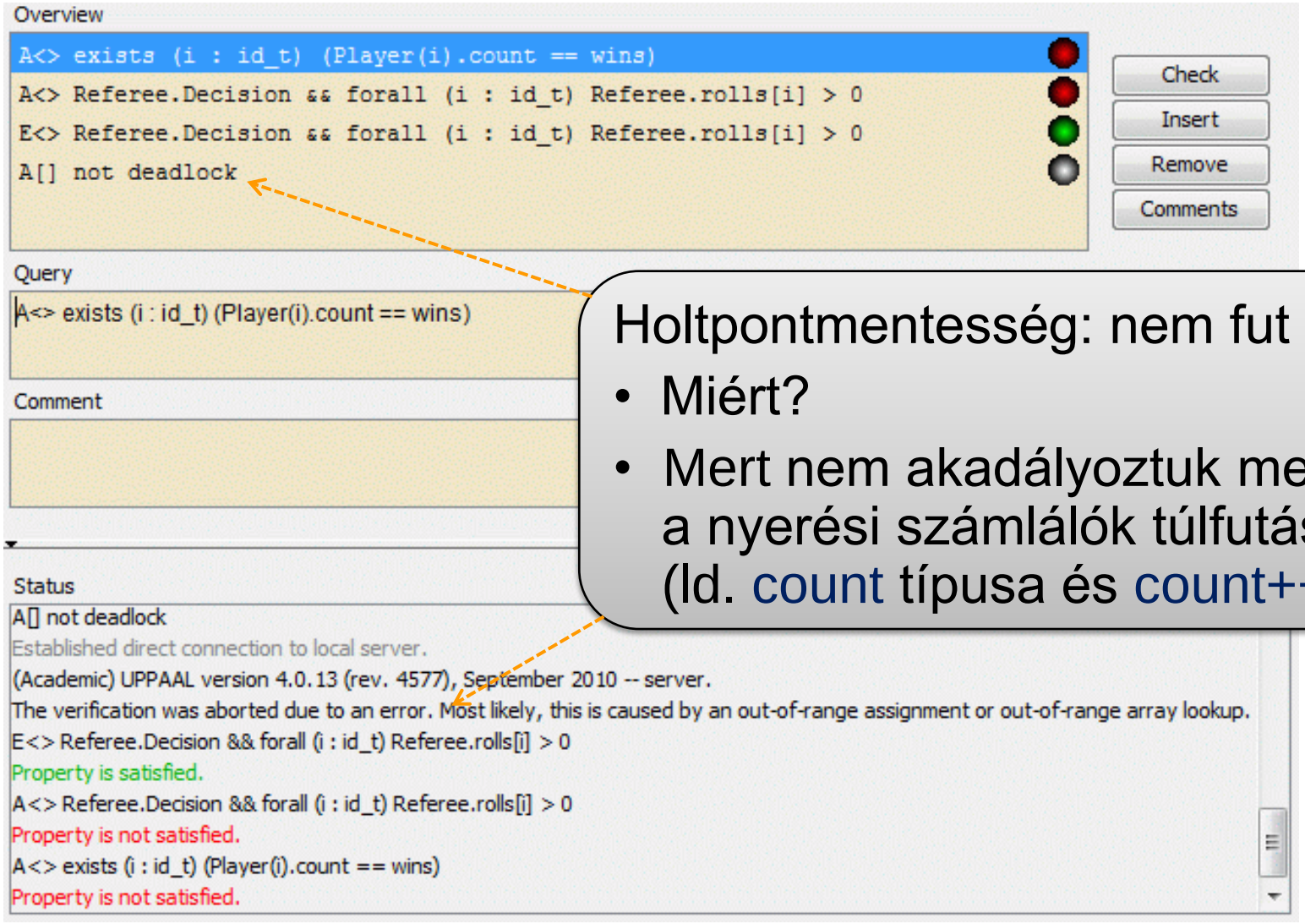
    A<> exists (i : id_t) (Player(i).count == wins)
- A bíró csak akkor hoz döntést, ha minden játékos dobott
  - Ez legalább egyszer megtörténik:

    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

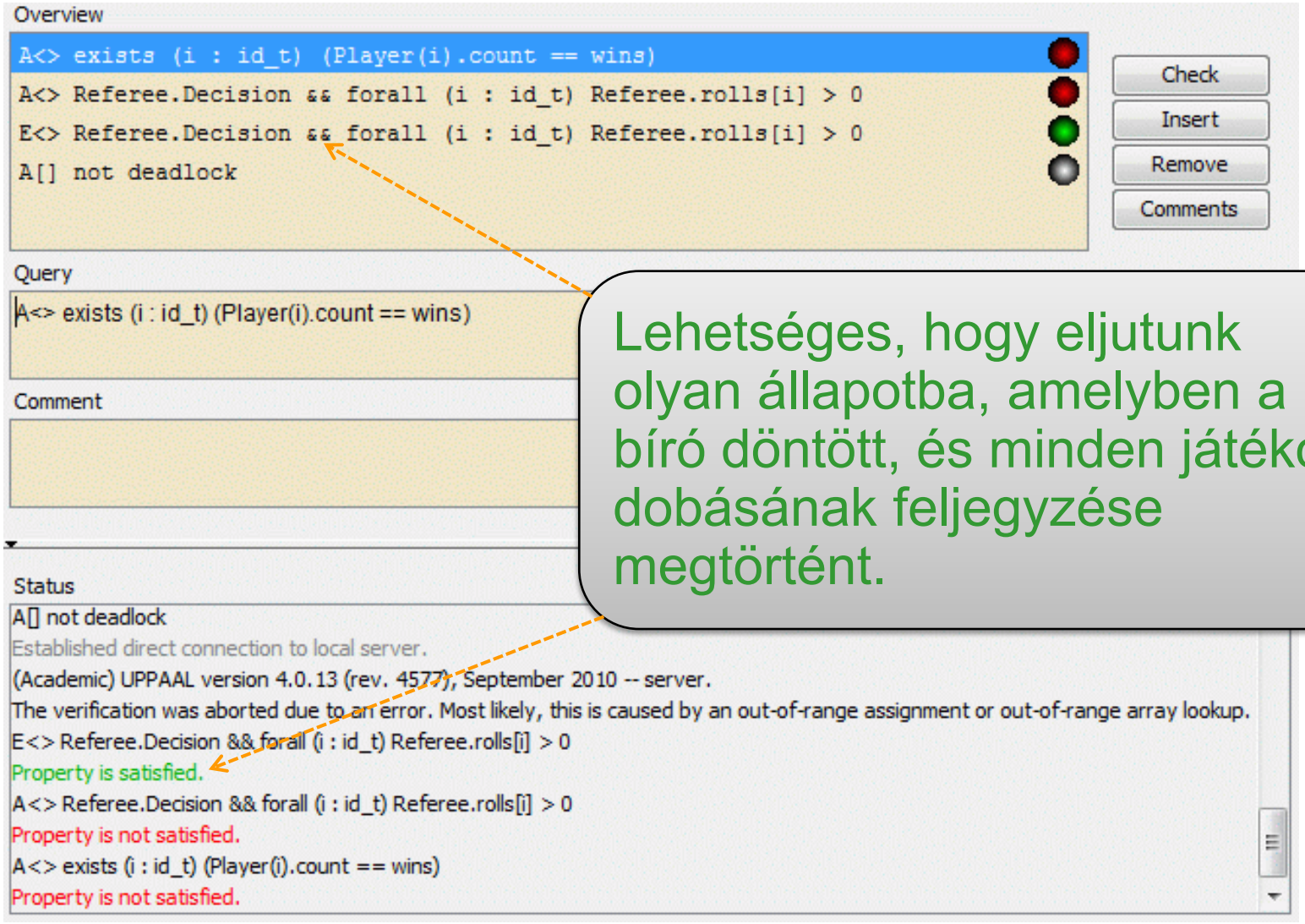
Ellenőrizzük a működést!



Holtpontmentesség: nem fut le.

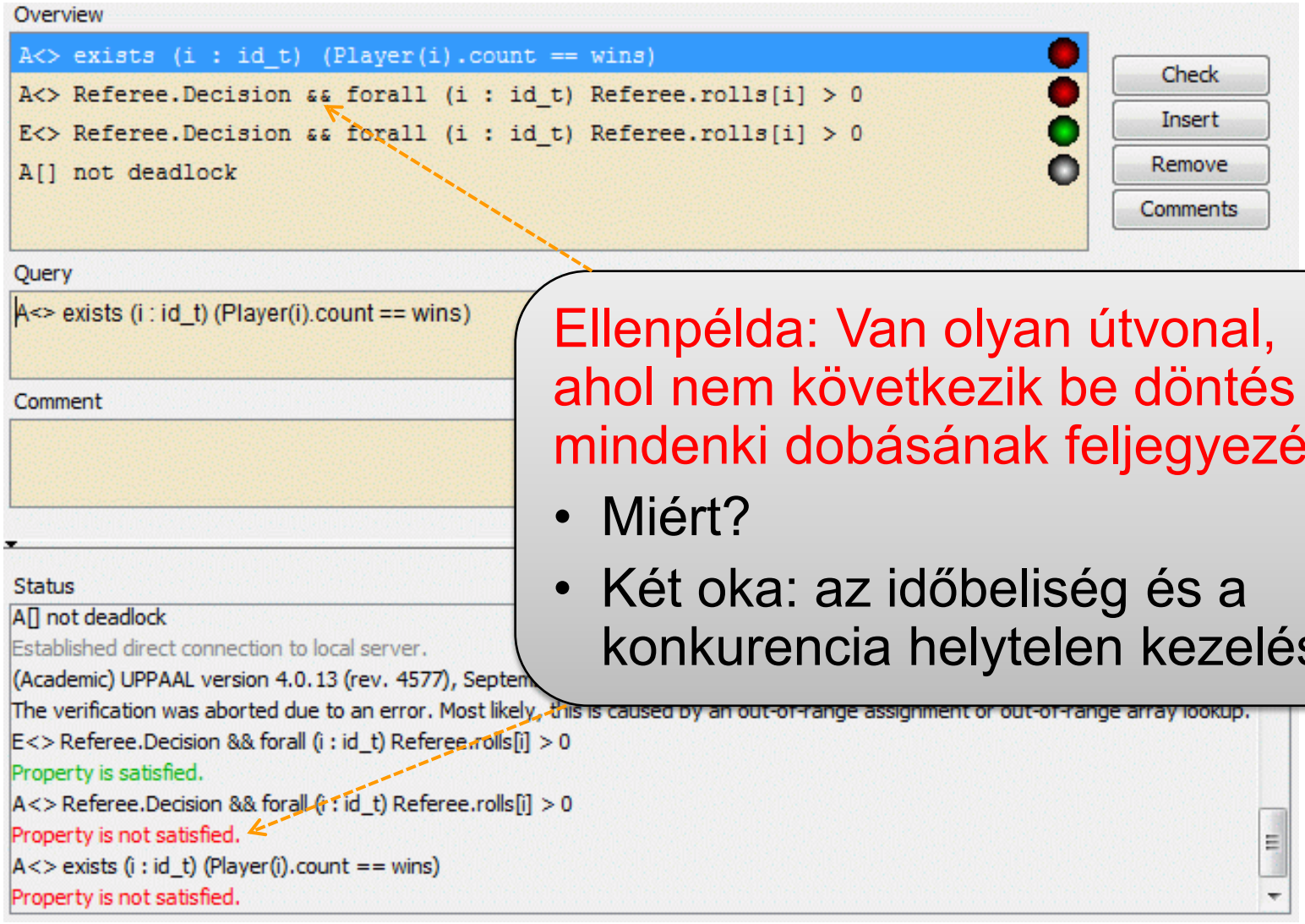
- Miért?
- Mert nem akadályoztuk meg a nyerési számlálók túlfutását (ld. count típusa és count++).

Ellenőrizzük a működést!



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

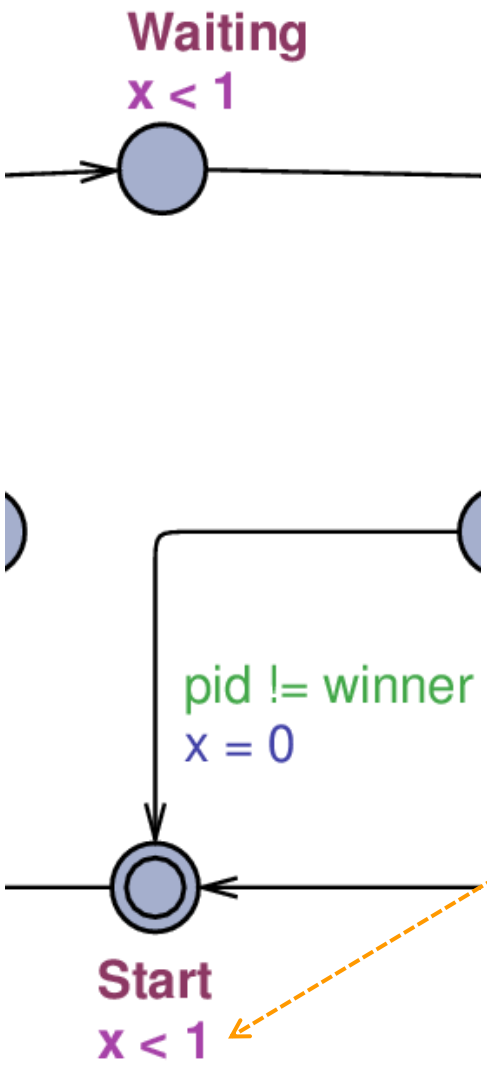
Ellenőrizzük a működést!



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

- Miért?
- Két oka: az időbeliség és a konkurencia helytelen kezelése!

Időzítés: Triviális ellenpélda kiküszöbölése

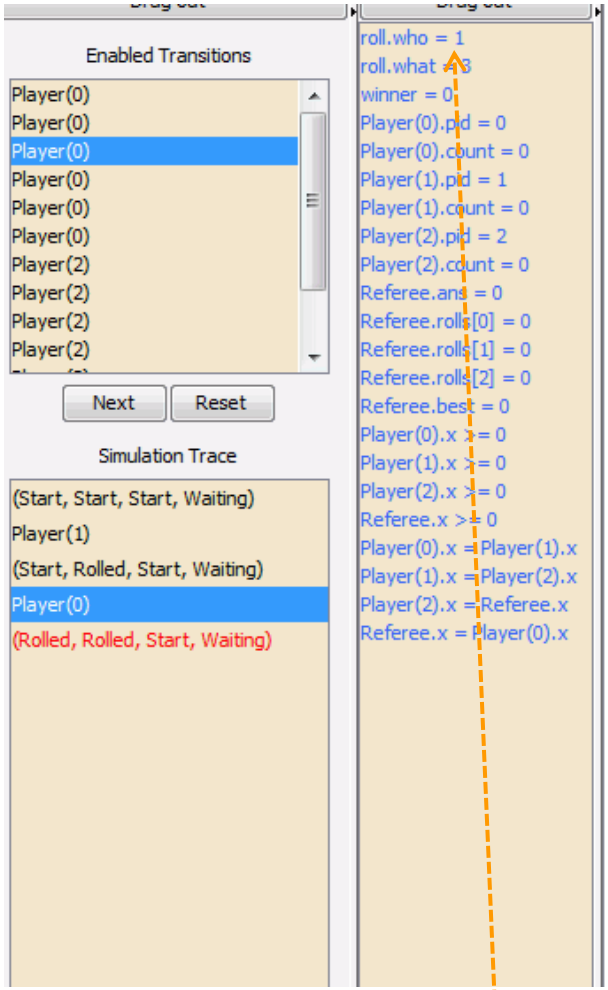


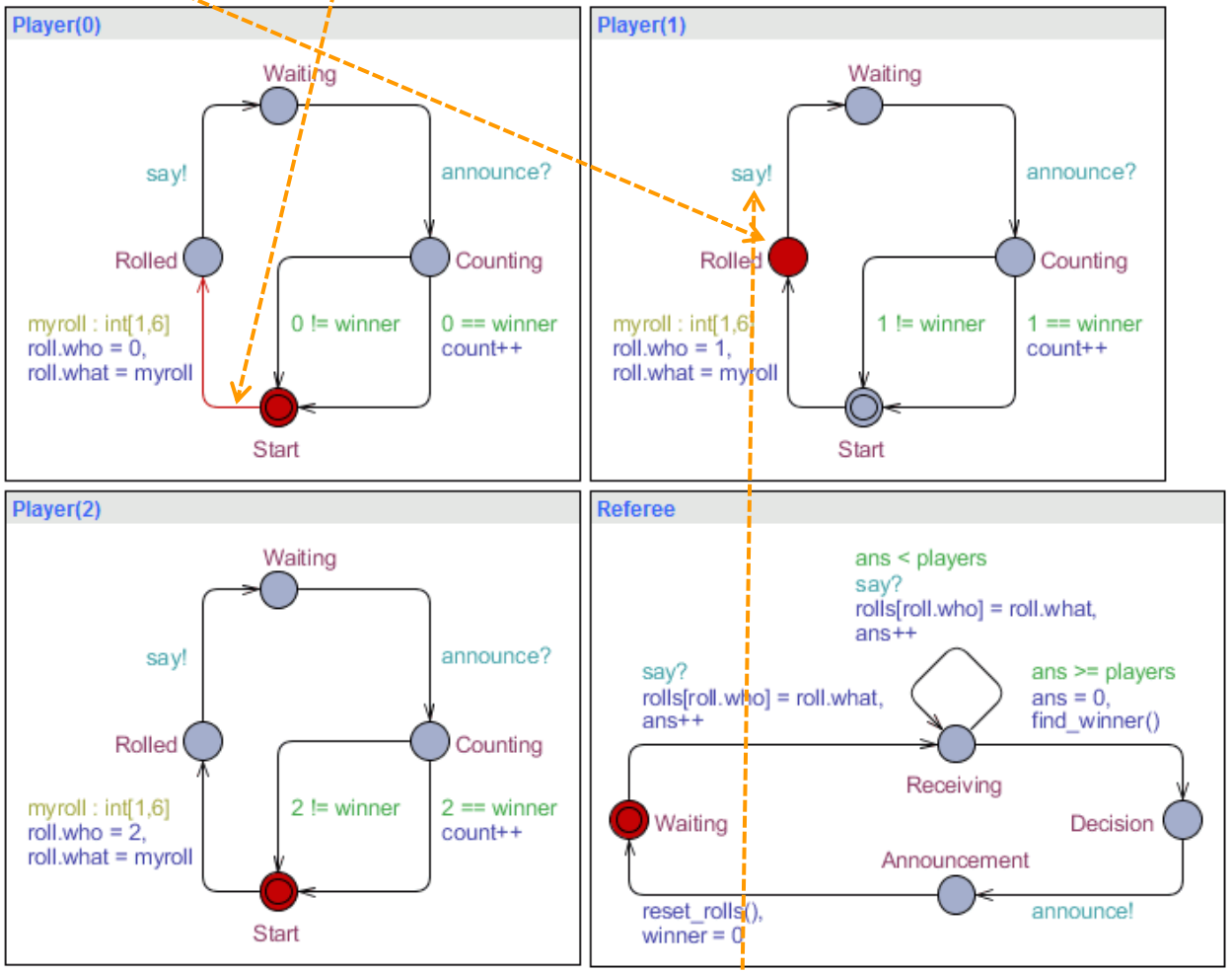
- Ha az összes lehetséges lefutást vizsgáljuk (pl. A<>), akkor az UPPAAL figyelembe veszi azt a lehetőséget is, hogy egy állapotot sosem hagyunk el
- Megoldás:
  - Óraváltozó bevezetése
    - Invariáns előírása
    - Legfeljebb 1 időegységig tartózkodhatunk az adott állapotban
    - Gondoskodjunk az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

Konkurens működés - mi a baj?

2. játékos kockával dobott

1. játékos kockával dob

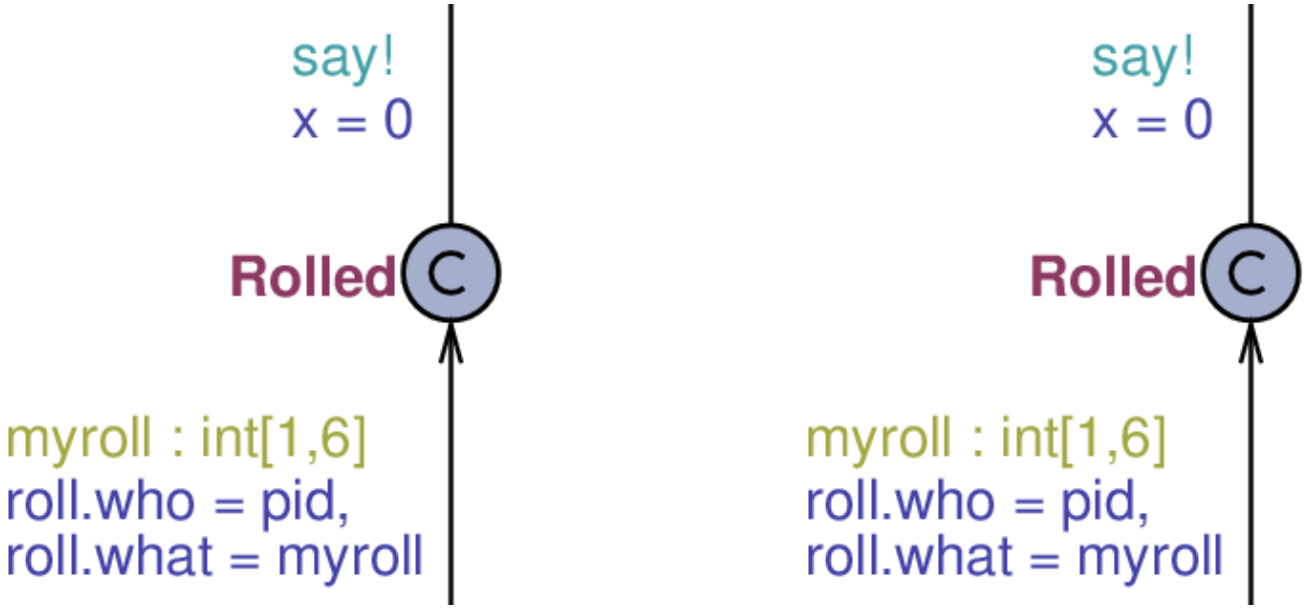




1. játékos felülírja a megosztott változót

2. játékos hibásan „küldi el”

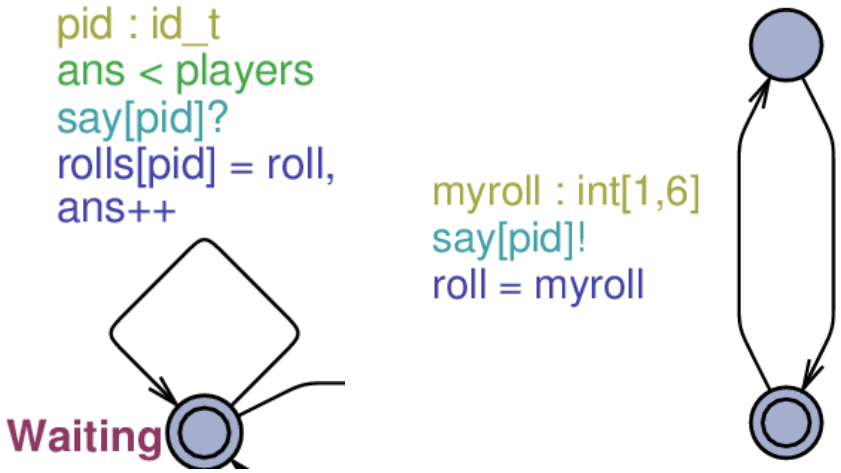
Nem kívánt konkurencia kiküszöbölése



- 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

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

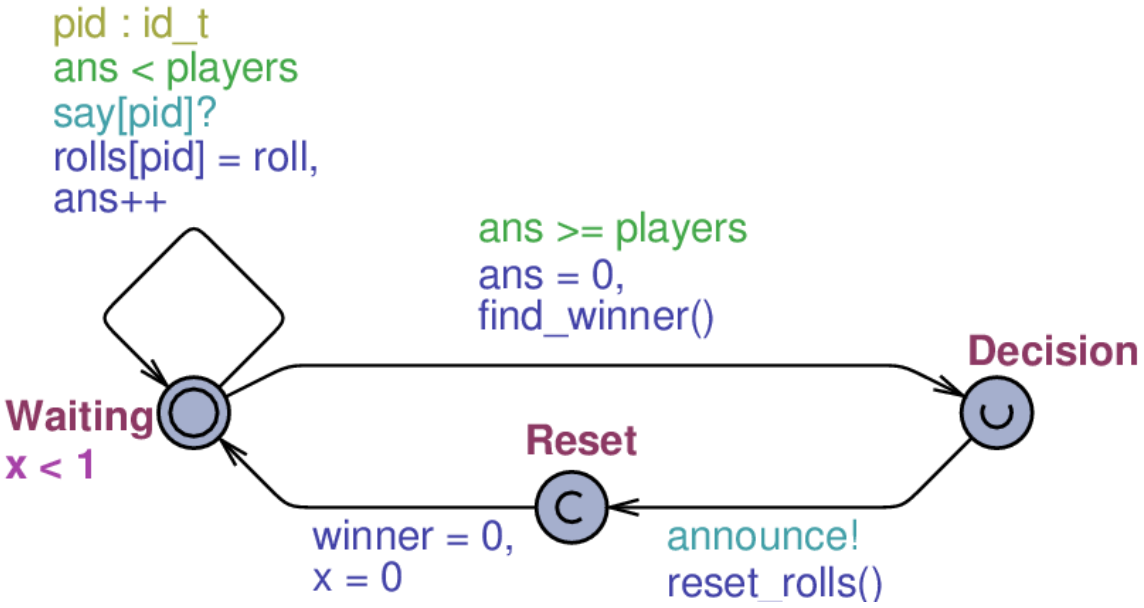


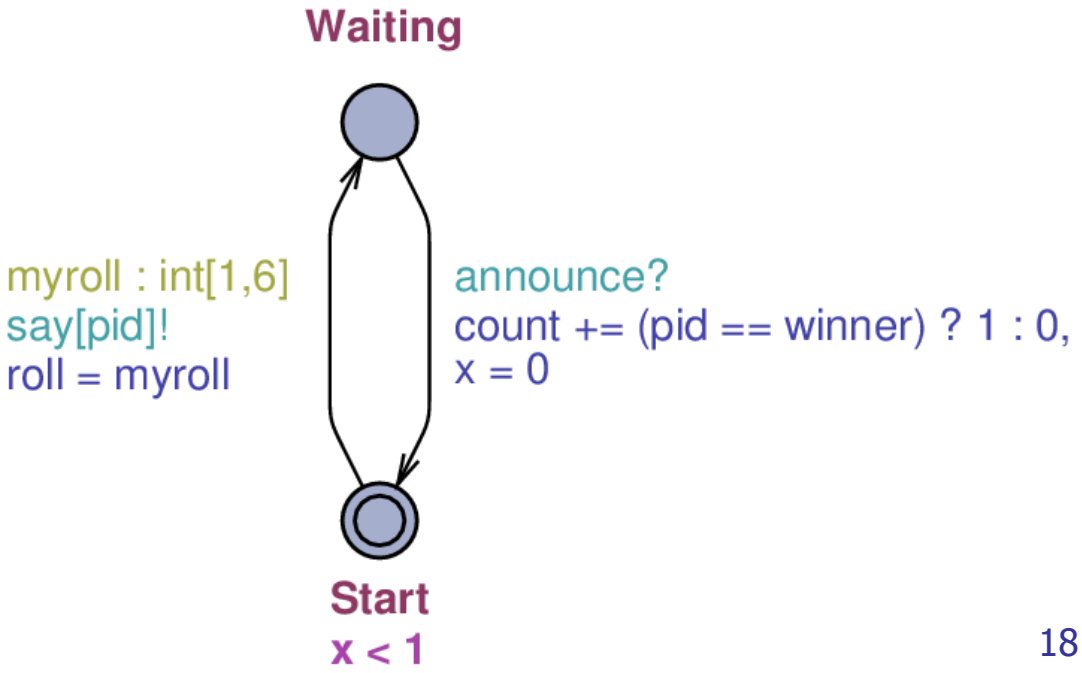
- Iterátorok alkalmazása

```
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 szinkronizáló él által beállított globális változót (azt előbb kell beállítanunk, pl, az előző élen)
  - Nem „védhetünk meg” Guard-dal egy Sync-ben levő változót
- 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 „eventually” A<> q tulajdonság használata esetén ki kell zárnunk a triviális ellenpéldát
  - Óraváltozók használatával
  - „Urgent” állapot beállításával
  - Emlékezzünk a „leads to” p --> q szemantikájára: A[] (p imply A<> q)
- Ne felejtsük el az óraváltozókat megfelelően inicializálni!
- A csatorna-, vagy automataszintű prioritások használata
  - Az UPPAAL modellellenőrzője nem támogatja (pl. a holtpontot sem tudja ellenőrizni)
  - Ezek a modellezési elemek lehetőleg kerülendők