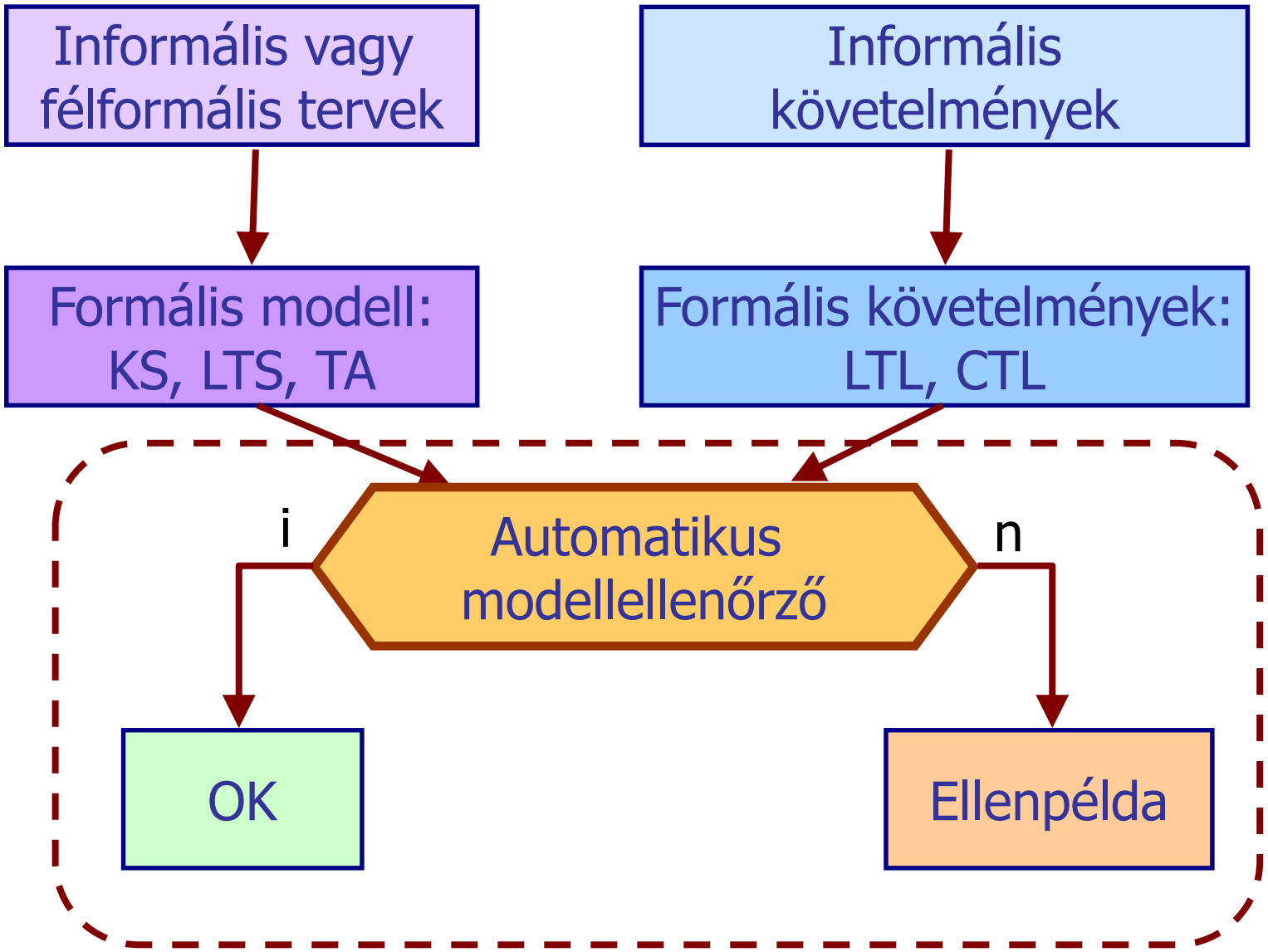
Modellellenőrző eszközök: A SPIN modellellenőrző (kedvcsináló)

dr. Majzik István
BME Méréstechnika és Információs Rendszerek Tanszék

Mire szolgálnak a modellellenőrzők?



A SPIN a klasszikus eszközök között

Eszköz	Modellek leírása	Tulajdonság leírása	Ajánlott használat
UPPAAL uppaal.org	Időzített automata, változókkal	Korlátozott CTL	Időfüggő viselkedés modellezése, szinkron kommunikáció
SPIN spinroot.com	Process Meta Language (Promela)	LTL, címkék, tulajdonság automata (never claim)	Aszinkron, üzenetekkel kommunikáló processzek protokolljai, algoritmusai
NuSMV nusmv.fbk.eu	Szinkron és aszinkron véges állapotú gépek (FSM)	CTL, LTL	Megosztott változókat használó komponensek algoritmusai, hardver rendszerek (szinkron)

A SPIN modell leíró nyelve: Promela

Promela: Process Meta Language

- Processzek: Konkurens végrehajtás egységei
  - Elosztott algoritmusok, protokollok elemei
  - Nemdeterminisztikus végrehajtás is megadható
- Csatornák: Processzek közötti interakciók
  - Aszinkron: FIFO üzenetcsatorna, adott hosszal
  - Szinkron (randevú, handshake)
- Változók: Adatmanipuláció modellezése
  - Lokális változók processzekben
  - Globális (megosztott) változók processzek között

Processz leírás: Ciklus és választás példa

```
proctype Euclid(int x, y) {
    do
    :: (x >  y)   ->   x = x - y
    :: (x <  y)   ->   y = y - x
    :: (x == y)  ->  goto done
    od;
done:
    printf("%i",x)
}
```

Eulideszi algoritmus két szám legnagyobb közös osztójának meghatározására

```
proctype random_count(int count) {
    do
    :: (count != 0) ->
          if
          :: count = count+1
          :: count = count-1
          fi
    :: (count == 0) -> break
    od
}
```

Nemdeterminisztikusan felfelé vagy lefelé számol, amíg 0-t el nem éri

Példa csatorna használatra

```
chan Product[2] = [5] of {byte};

proctype Producer(byte pid) {
        do
                :: Product[pid] ! 0
                :: Product[pid] ! 1
        od
}

proctype Consumer( ) {
        byte x;
        do
                :: Product[0] ? x;
                :: Product[1] ? x
        od
}

init { run Producer(0); run Producer(1); run Consumer( ) }
```

Két termelő és egy fogyasztó processz, 5 hosszúságú FIFO csatornákkal összekötve (csatornatömbbel, a csatornákban byte üzenetekkel)

Csatornák olvasása, írása

- Csatorna típusok:
  - Aszinkron (FIFO): nem 0 hosszal deklarálva; szinkron: 0 hosszal
- Szintaxis q csatorna esetén:
  - Írás: q! e1, e2, …, en ← egy elem: változók vagy konstansok
  - Olvasás: q? e1, e2, …, en ← egy elem: változók vagy konstansok
  - Vizsgálat: empty(q), nempty(q), full(q), nfull(q), len(q)
- Specifikus csatorna olvasások (és hasonlóan írások)

q? args	← csatorna elejéről olvas
q?? args	← bárhonnan a csatornából
q? <args>	← csak kimásol
q?? <args>	← bárhonnan, csak kimásol
q? [args]	← csak vizsgál
q?? [args]	← bárhonnan, csak vizsgál

- Lásd még: http://spinroot.com/spin/Man/promela.html

Ellenőrizendő tulajdonságok megadása

- Állítások: assert() feltétel, ami igaz kell legyen
  - Pl. assert(x!=y)
- Címkék utasításokon (ciklus, választás is)
  - Elfogadható végállapot: end prefix (pl. end, end1, end_a)
  - Végrehajtandó a haladáshoz: progress prefix
    (azaz progress nélküli végtelen végrehajtást hibaként jelez)
- LTL temporális logika
  - ltl property_name {…} alakban, pl. ltl p1 {p U q}
  - Operátorok: U, W van, F helyett <>, G helyett [], X nincs
- never állítás
  - Speciális processz, csak feltételekből áll
  - Ha illeszkedik a modell végrehajtására, akkor hibajelzés

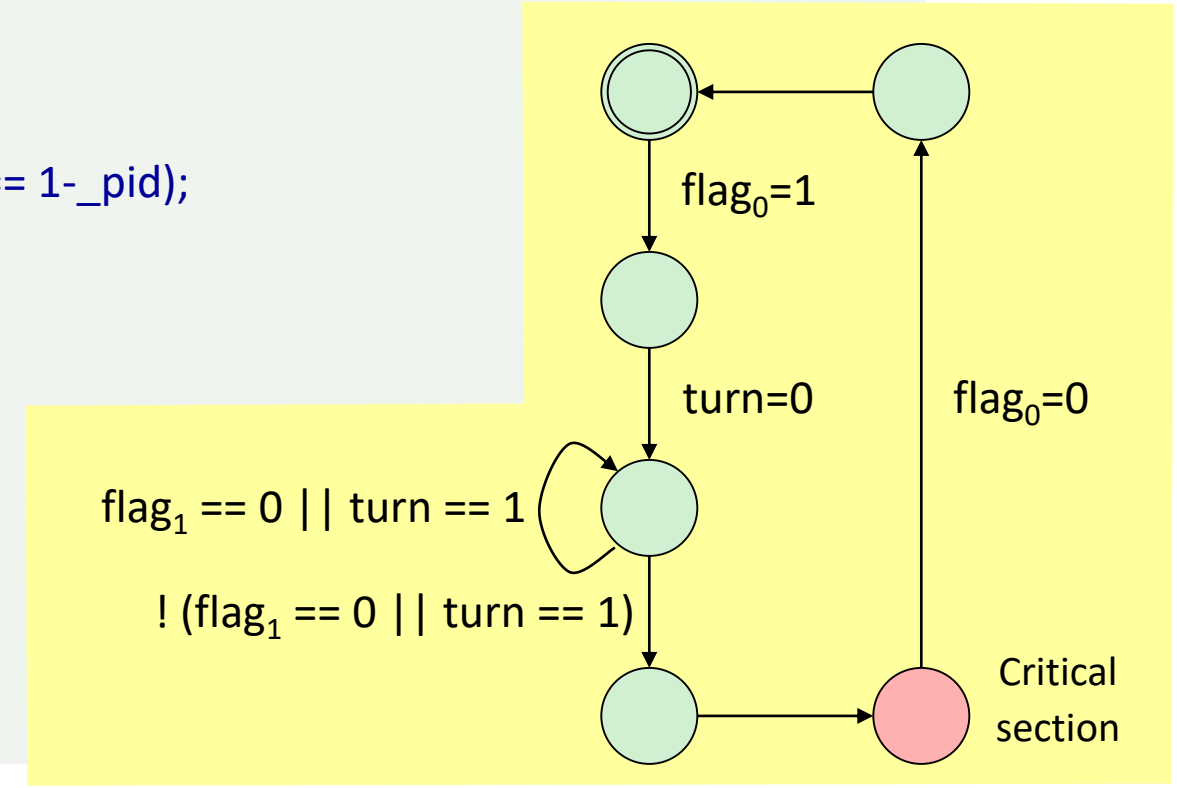
Peterson kölcsönös kizárás algoritmusa (assert)

```
bool turn, flag[2];              // the shared variables, booleans
byte ncrit;                      // number of processes in the critical section

active [2] proctype user()       // two processes with built-in identifier _pid
{
      assert(_pid == 0 || _pid == 1);
   again:
      flag[_pid] = 1;
      turn = _pid;
      (flag[1-_pid] == 0 || turn == 1-_pid);

      ncrit++;
      // critical section (CS)
      assert(ncrit == 1);
      ncrit--;

      flag[_pid] = 0;
      goto again
}
```



Peterson kölcsönös kizárás algoritmusa (LTL)

```
bool turn, flag[2];
bool critical[2];       // Registering processes in CS

active [2] proctype user()
{
   assert(_pid == 0 || __pid == 1);
 again:
   flag[_pid] = 1;
   turn = _pid;
   (flag[1 - _pid] == 0 || turn == 1 - _pid);

   critical[_pid] = 1;
   // critical section (CS)
   critical[_pid] = 0;

   flag[_pid] = 0;
   goto again;
}
```

LTL expressions:

```
[] !(critical[0] && critical[1])

[] <> (critical[0])
[] <> (critical[1])

[] (critical[0] ->
  (critical[0] U
    (!critical[0] &&
      ((!critical[0] &&
        !critical[1]) U critical[1]))))

[] (critical[1] ->
  (critical[1] U
    (!critical[1] &&
      ((!critical[1] &&
        !critical[0]) U critical[0]))))
```

Példa: Alternáló bit protokoll üzenetvesztéssel

Alternating Bit Protocol

- A küldő minden üzenethez egy ellenőrző bitet csatol
- A vevő nyugtáz minden üzenetet azzal, hogy visszaküldi a kapott ellenőrző bitet
  - De csak akkor dolgozza fel az üzenetet, ha a kapott ellenőrző bit az előzőleg feldolgozott üzenet ellenőrző bitjének negáltja
  - Ha nem így van, akkor is (újra)küldi a nyugtát
- Ha a küldő azt látja, hogy a nyugtában visszakapta az előzőleg küldött ellenőrző bitet, akkor a következő üzenetküldéshez negálja azt
  - Ha nem azt az ellenőrző bitet kapta vissza, akkor eldobja a nyugtát és (a bit negálása nélkül) újraküldi az üzenetet
  - Ha nem jön a nyugta, újraküldi az üzenetet

Példa: Alternáló bit protokoll üzenetvesztéssel

```
mtype {MSG, ACK};
chan StoR = [2] of {mtype, bit};
chan RtoS = [2] of {mtype, bit};

active proctype Receiver()
{
   bit recvbit;
   do
   :: StoR ? MSG, recvbit ->
      RtoS ! ACK, recvbit
   :: StoR ? MSG, recvbit ->
      skip        /* ACK lost */
   od
}
```

- A vevő nyugtáz minden üzenetet azzal, hogy visszaküldi a kapott ellenőrző bitet
  - Nincs modellezve: csak akkor dolgozza fel az üzenetet, ha a kapott ellenőrző bit az előzőleg feldolgozott üzenet ellenőrző bitjének negáltja
- A nyugta elveszhet

Példa: Alternáló bit protokoll üzenetvesztéssel

- A küldő minden üzenethez egy ellenőrző bitet csatol
- Üzenet elveszhet
- Ha megjött a nyugta
  - Ha küldő a nyugtában visszakapta a küldött ellenőrző bitet, akkor a következő küldésben negálja
  - Ha nem a küldött ellenőrző bitet kapta vissza, akkor eldobja a nyugtát és (a bit negálása nélkül) újraküldi az üzenetet
- Ha nem jön a nyugta, újraküldi az üzenetet

```
active proctype Sender()
{
   bit sendbit, ackbit;
   do
   :: if   /* Sending */
      :: StoR ! MSG, sendbit
      :: skip          /* MSG lost */
      fi
      if   /* Waiting for ack */
      ::  RtoS ? ACK, ackbit;
         if
         :: ackbit == sendbit ->
            sendbit = 1-sendbit
         :: else -> skip
         fi
      :: timeout -> skip
      fi
   od
}
```

Példa: Alternáló bit protokoll üzenetvesztéssel

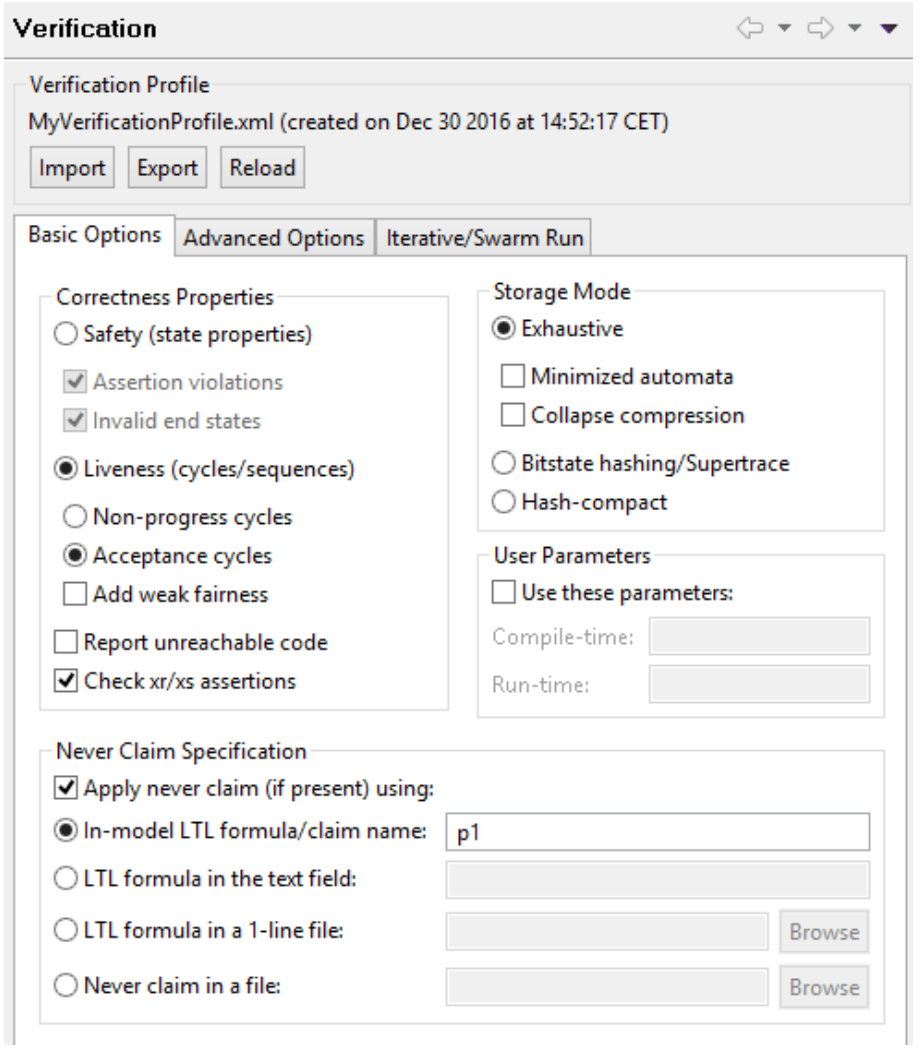
```
mtype {MSG, ACK};
chan RtoS = [2] of {mtype, bit};
chan StoR = [2] of {mtype, bit};

active proctype Receiver()
{
   bit recvbit;
   do
   :: StoR ? MSG, recvbit ->
      RtoS ! ACK, recvbit
   :: StoR ? MSG, recvbit ->
      skip        /* ACK lost */
   od
}
```

```
active proctype Sender()
{
   bit sendbit, ackbit;
   do
   :: if   /* Sending */
      :: StoR ! MSG, sendbit
      :: skip          /* MSG lost */
      fi
      if   /* Waiting for ack */
      ::  RtoS ? ACK, ackbit;
         if
         :: ackbit == sendbit ->
            sendbit = 1-sendbit
         :: else -> skip
         fi
      :: timeout -> skip
      fi
   od
}
```

A SPIN modellellenőrző

- Parancssori verzió
  - Sokféle kapcsoló
- Eclipse RCP keret: SpinRCP
  - Modell szerkesztő
  - Szintaxis ellenőrző
  - Automata nézet
  - Szimuláció (MSC jellegű megjelenítéssel)
  - Verifikáció különféle paraméterezéssel



SpinRCP teljes nézet