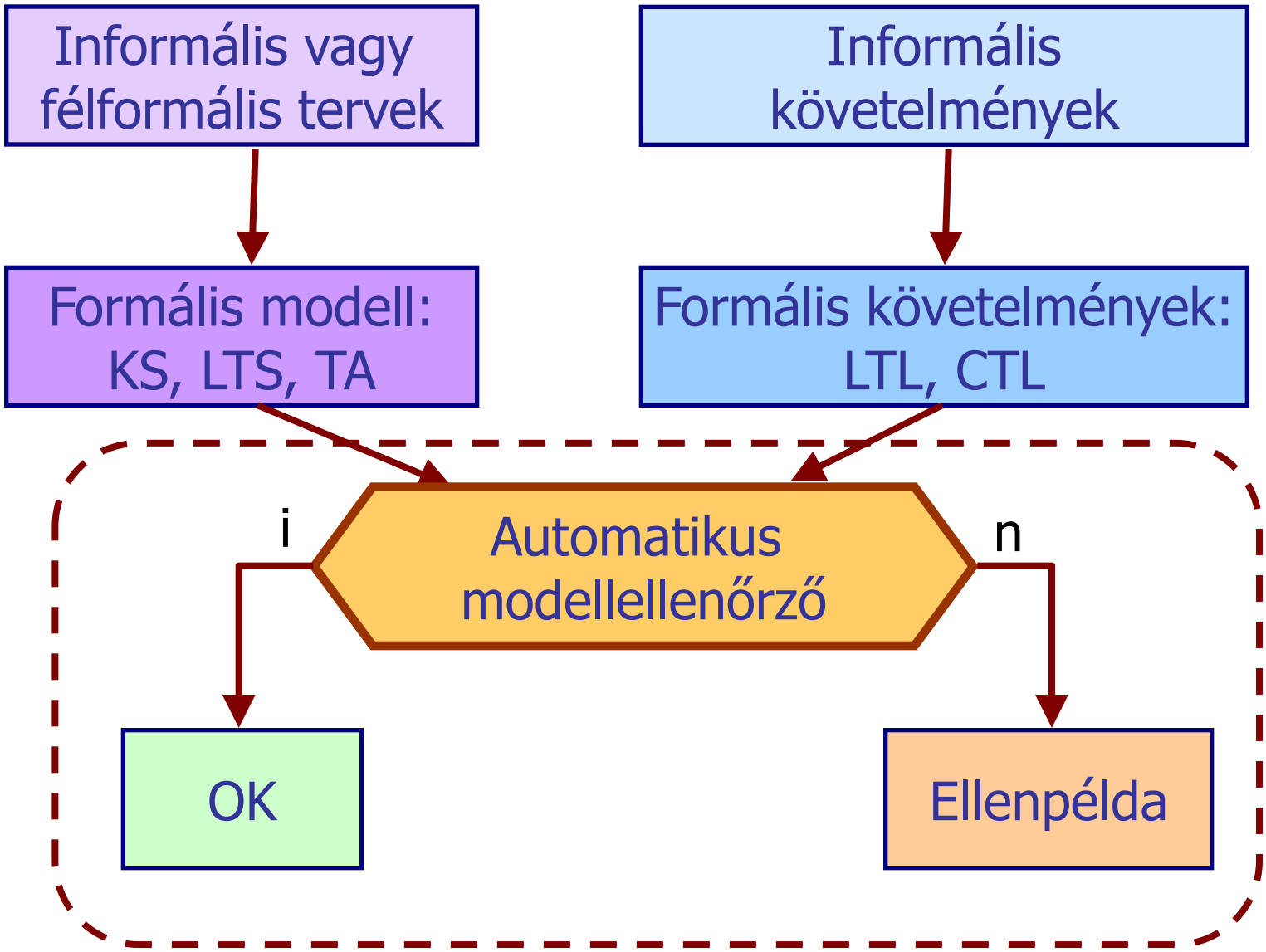
Modellellenőrző eszközök: A SPIN modellellenőrző

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 elemek

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

Adattípusok

- Alap adattípusok:
  - bool vagy bit (1 bit), byte (8 bit),
    short (16 bit, előjeles), int (32 bit, előjeles)
  - Felsorolás: mtype = {control, data, error}
- Csatornák
  - chan név = [csatornahossz] of {típusok} ← elem: n-es
  - Példa: chan c = [5] of {bit, int}
  - Pufferelt (aszinkron, FIFO), ha nem 0 a hossz
  - Nem pufferelt (szinkron), ha 0 hosszal van definiálva
- Strukturált típusok
  - Tömbök: int y[10]; chan c[3] = [6] of {bit, int, chan};
  - Struktúrák: typedef MSG {bit control[5]; int data}
  - Struktúrák használata: MSG m, m.control[3], m.data

Processzek

- Definíció („processz típus”):

  proctype procname (formal_parameters) {local_declarations; statements}

- Példányosítás
  - init nevű processz: alapértelmezetten induló processz
  - active [num] megadás a proctype előtt: automatikusan indul
  - run utasítás: processz indítása, pl. init { run A() }
  - Átadható processz paraméter: alaptípusú adat, csatorna
- Utasítások
  - Mellékhatás-mentes kifejezés is lehet
  - Egymást követő utasítások elválasztása ; vagy -> ekvivalens

```
byte state = 2;
proctype A( ) {
        (state == 1) -> state = 3
}
```

```
byte state = 2;
proctype A( ) {
        state == 1;
        state = 3
}
```

Utasítások végrehajtása

- Utasítás engedélyezett (végrehajtható) vagy blokkolt lehet
  - Blokkolt utasításon „megakad” a végrehajtás (amíg engedélyezett nem lesz)
  - Ha engedélyezett egy utasítás, akkor végrehajtható
- Üres utasítás: skip
  - Mindig engedélyezett
- Értékadás: pl. x=x-1
  - Mindig engedélyezett
- Kifejezés (feltétel)
  - Engedélyezett, ha kiértékeléskor nem 0 (azaz nem hamis)
  - Pl. (a == b) kifejezésen megakad a végrehajtás, ha a!=b
- Feltétlen ugrás: goto label egy label: címkéjű utasításra
  - Mindig engedélyezett
- Timeout: timeout
  - Engedélyezett, ha más utasítás nem

Választás utasítás (if ... fi)

- Szintaxis:

```
if
    :: utasítások
    ...
    :: utasítások
    :: else utasítások
fi
```

- Végrehajtás
  - Opciónak nevezett a :: -tal bevezetett utasítások sora
  - Egy opció engedélyezett, ha első utasítása engedélyezett
  - Az else akkor engedélyezett, ha más opció nem
  - Ha több opció engedélyezett, akkor véletlen választás történik
  - Választás engedélyezett, ha legalább egy opció engedélyezett

Ciklus utasítás (do ... od)

- Szintaxis:

```
do
    :: utasítások
    ...
    :: utasítások
    :: else utasítások
od
```

```
do
    :: (x >  y)   ->   x = x - y
    :: (x <  y)   ->   y = y - x
    :: (x == y)   ->   break
od
```

- Végrehajtás
  - A ciklus engedélyezett, ha legalább egy opciója engedélyezett (azaz annak első utasítása engedélyezett)
  - Ha több opció engedélyezett: véletlen választás
  - Az else akkor engedélyezett, ha más opció nem
  - Engedélyezett opció végrehajtása után a ciklus újrakezdődik
  - Kilépés: break vagy goto label

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)
}
```

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

Csatornák használatának alapesetei

- 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)
- Végrehajtás pufferelt csatorna esetén
  - Írás engedélyezett, ha nincs tele a csatorna
  - Íráskor a csatorna végére kerülnek az értékek
  - Olvasás engedélyezett, ha nem üres a csatorna és az olvasáskor megadott konstansok illeszkednek a csatorna elején található elem konstansaira
  - Olvasáskor a kivett elem v1, v2, … értékei lesznek az olvasásban megadott e1, e2, … változók értékei
- Végrehajtás nem pufferelt (szinkron) csatorna esetén
  - Olvasás és írás együtt engedélyezett, ha ezek szimultán végrehajthatóak, és a bennük lévő konstansok megegyeznek
  - Az írt értékek lesznek az olvasásban megadott változók értékei

Példa csatorna használatra

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

proctype Producer(byte pid) {
        do
                :: 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( ) }
```

Speciális kifejezések

- atomic kifejezés
  - Oszthatatlan egészként végrehajtható utasítások
    atomic { (state==1) -> state = state + 1 }
  - Nincs közben más processz konkurens végrehajtása
  - Belső blokkolás esetén az atomi végrehajtás elveszik
- d_step kifejezés
  - Hasonló az atomic kifejezéshez, plusz determinisztikus belső végrehajtás (véletlen választás esetén is)
  - Belőle kiugrani vagy címkével a belsejét elérni nem szabad
  - Belső blokkolás hibát okoz

További lehetőségek

- Lásd: http://spinroot.com/spin/Man/promela.html
- Specifikus csatorna olvasások (és hasonlóan írások)

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

- Speciális konstrukciók
  - for(...), do ... od unless(...)
  - select
  - enabled
  - eval()
  - ... és még sok más

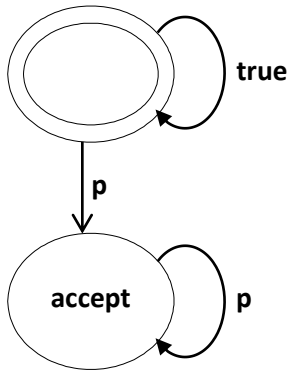
Ellenőrizendő tulajdonságok megadása

- 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 keres)
- never állítás
  - Speciális processz, csak feltételekből áll
  - Ha illeszkedik a modell végrehajtására, akkor hibajelzés
- LTL temporális logika
  - ltl property_name {…} alakban, pl. ltl p1 {p U q}
  - Operátorok: U, W van, F helyett <>, G helyett [], X nincs
  - Külön leképezhető never állításra

Példa never állításra

- Példa: Ne álljon fenn, hogy a jövőben p feltétel folyamatosan igazzá válik (azaz ne legyen F G p)

```
never {       /* <>[]p */
      do
      :: true   /* after an arbitrarily long prefix */
      :: p -> break  /* p becomes true */
      od;
accept:
      do
      :: p  /* and remains true forever after */
      od
}
```



- Speciális címke: accept prefix
  - Ha a never állításban az accept végtelen sokszor elérhető, akkor az hiba (illeszkedik a never állítás)

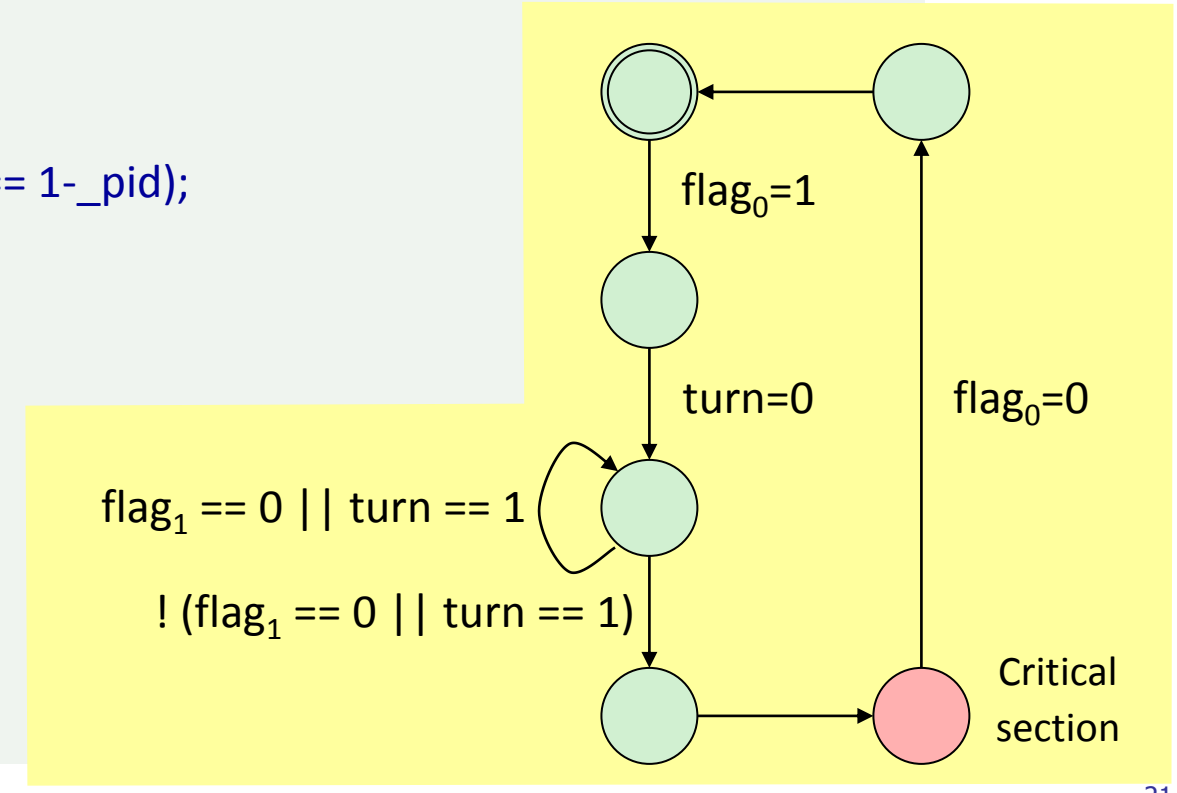
Peterson kölcsönös kizárás algoritmusa (assert)

```
bool turn, flag[2];                 // the shared variables, booleans
byte ncrit;                         // nr of processes in 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 jön a következő üzenet, akkor újraküldi 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 */
   :: timeout ->
      RtoS ! ACK, recvbit
   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
- Ha nem jön a következő üzenet a küldőtől, akkor újraküldi a nyugtát

Példa: Alternáló bit protokoll üzenetvesztéssel

- A küldő minden üzenethez egy ellenőrző bitet csatol
- Üzenet elveszhet
- Ha megjött a nyugta
  - Ha 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

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

Példa: Alternáló bit protokoll üzenetvesztéssel

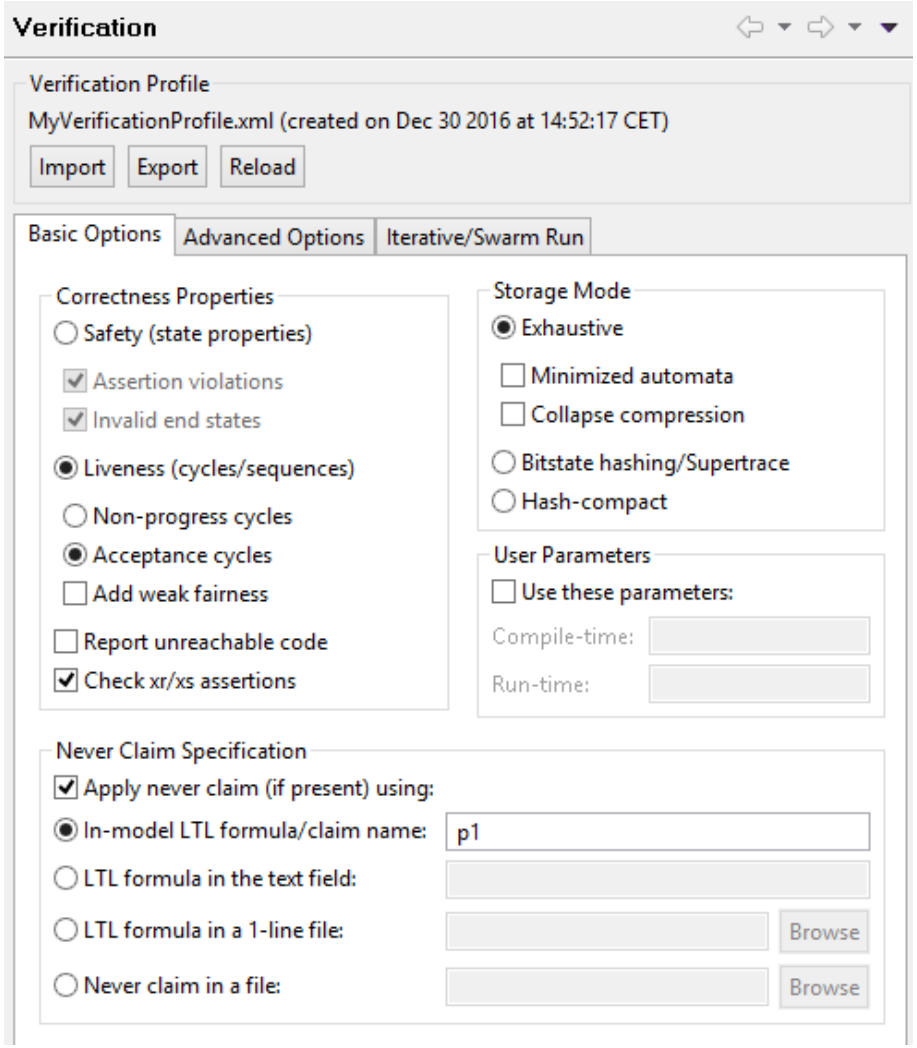
```
mtype {MSG, ACK};
chan RtoS = [2] of {mtype, bit};
chan StoR = [2] of {mtype, bit};

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

```
active proctype Sender()
{
   bit sendbit, ackbit;
   do
   :: if
      :: StoR ! MSG, sendbit
      :: skip          /* MSG lost */
      fi
      if
      ::  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