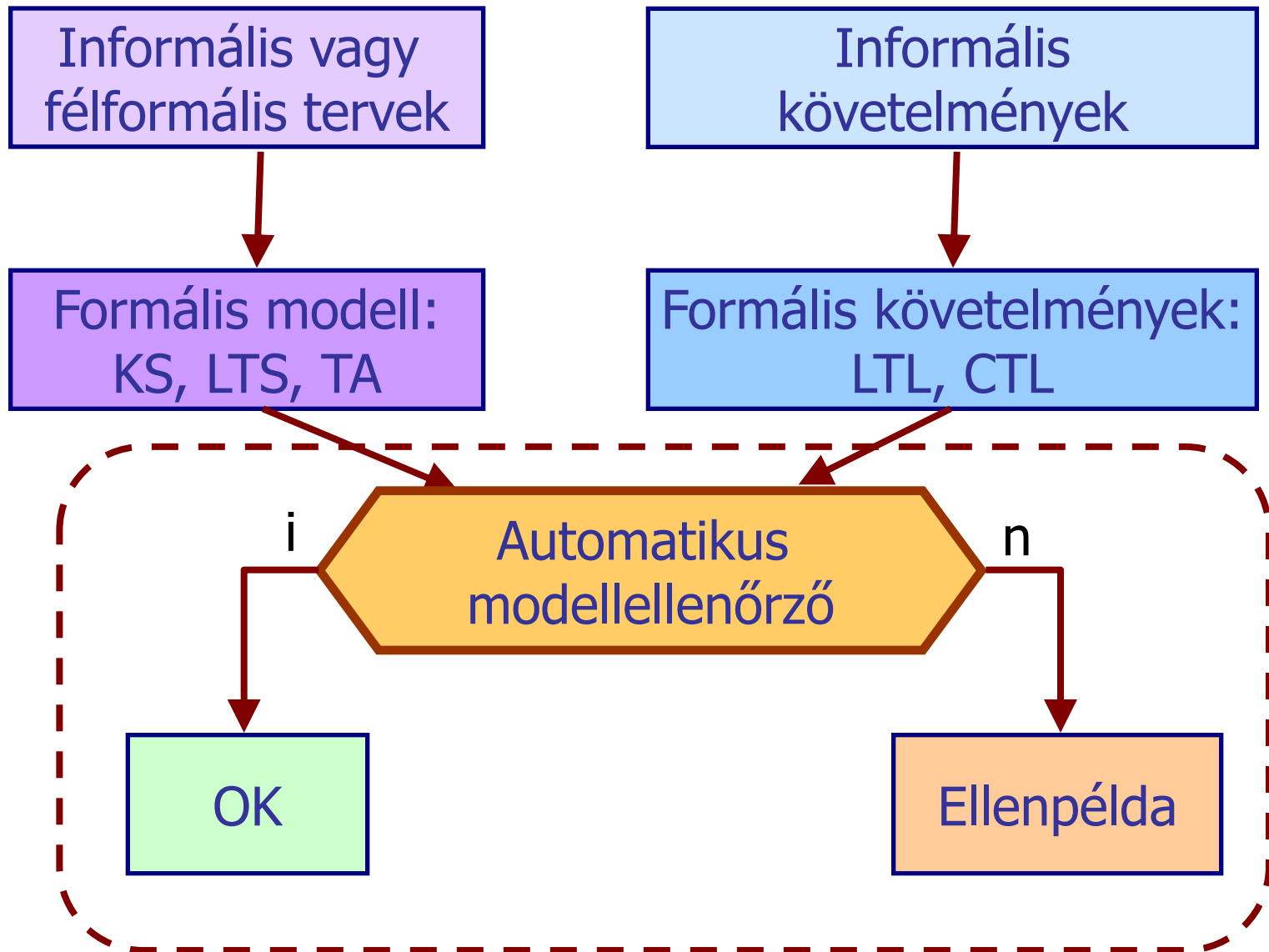


Modellellenőrző eszközök

dr. Majzik István

BME Méréstechnika és Információs Rendszerek Tanszék

Mire szolgálnak a modellellenőrők?



Klasszikus eszközök

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 modellellenőrző és a Promela nyelv alapjai

A SPIN modell leíró nyelve

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**
 - 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 x[10]; chan c[3] = [6] of {bit, int, chan};`
 - Strukturák: `typedef MSG {bit control[5]; int data}`
 - Strukturák 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 (hamis)
 - Pl. $(a == b)$ kifejezésen megakad a végrehajtás, ha $a \neq b$
- Feltétlen ugrás: goto label egy label: címkéjű utasításra
 - Mindig engedélyezett
- Timeout: timeout
 - Engedélyezett, ha más utasítás nem

Választás

- 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

- Szintaxis:

do

:: utasítások

...

:: utasítások

:: else utasítások

od

```
do
```

```
:: count = count + 1;
```

```
:: count = count - 1;
```

```
:: (count == 0) -> break
```

```
od
```

- Végrehajtás

- A ciklus engedélyezett, ha legalább egy opciója engedélyezett (azaz első utasítása engedélyezett)
- Ha több opció engedélyezett: véletlen választás
- Az **else** akkor engedélyezett, ha más opció nem
- Engedélyezett opció végrehajtása után a ciklus újratezdődik
- 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:  
  skip  
}
```

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

Csatornák használatának alapesetei

- Szintaxis q csatorna esetén:
 - Írás: $q! e_1, e_2, \dots, e_n$ <- egy elem, változók vagy konstansok
 - Olvasás: $q? e_1, e_2, \dots, e_n$ <- egy elem, változók vagy konstansok
 - Vizsgálat: $\text{empty}(q)$, $\text{nempty}(q)$, $\text{full}(q)$, $\text{nfull}(q)$, $\text{len}(q)$
- Végrehajtás pufferelt csatorna esetén
 - Írás nem engedélyezett, ha tele van a csatorna, egyébként a csatorna végére kerülnek az értékek
 - Olvasás engedélyezett, ha a csatorna nem üres, és az olvasáskor megadott konstansok illeszkednek a csatorna elején található elem konstansaira
 - Olvasáskor a kivett elem v_1, v_2, \dots értékei lesznek az olvasásban megadott e_1, e_2, \dots 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 írások
 - `q?` `args`
 - `q??` `args` (bárhonnan a csatornából)
 - `q?` `<args>` (csak kimásol)
 - `q??` `<args>` (bárhonnan, csak kimásol)
 - `q?` `[args]` (vizsgál)
 - `q??` `[args]` (bárhonnan, vizsgál)
- Speciális konstrukciók
 - `for(...), do ... od unless(...)`
 - `select`
 - `enabled`
 - `eval()`
 - ... és még sok más

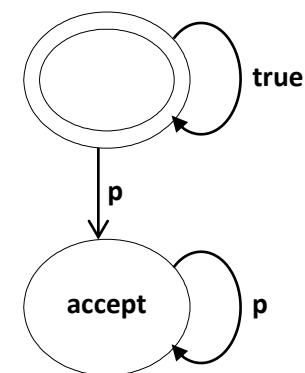
Ellenőrizendő tulajdonságok megadása

- **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 végrehajtás kereshető)
- **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`, `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)

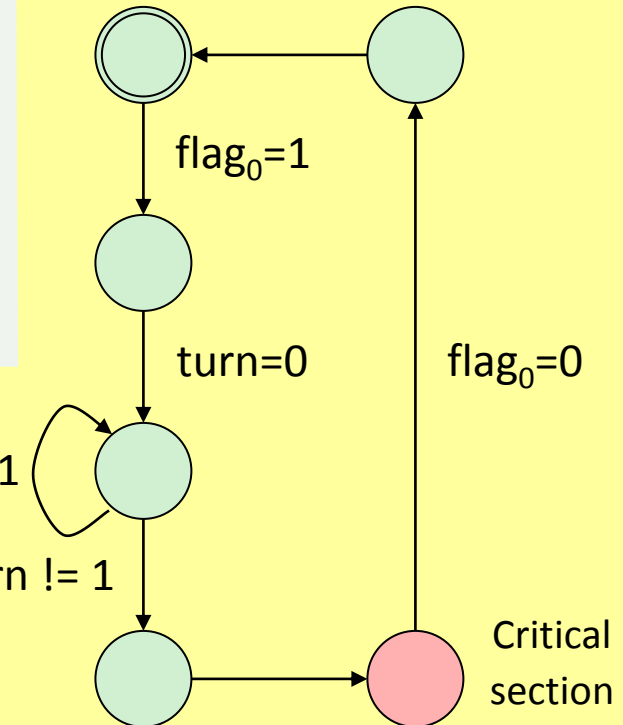
Peterson kölcsönös kizárás algoritmus (assert)

```
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++;
    assert(ncrit == 1);
    ncrit--;

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



Peterson kölcsönös kizárás algoritmus (LTL)

```
bool turn, flag[2];
bool critical[2];

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 */
    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])))
```

A SPIN modellellenőrző

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

The screenshot shows the 'Verification' dialog box in SpinRCP. It features a 'Verification Profile' section with the name 'MyVerificationProfile.xml' and creation details. Below this are 'Import', 'Export', and 'Reload' buttons. The main configuration area is divided into three tabs: 'Basic Options', 'Advanced Options', and 'Iterative/Swarm Run'. The 'Basic Options' tab is active and contains several sections: 'Correctness Properties' with radio buttons for 'Safety (state properties)', 'Liveness (cycles/sequences)', and 'Acceptance cycles', and checkboxes for 'Assertion violations', 'Invalid end states', 'Add weak fairness', 'Report unreachable code', and 'Check xr/xs assertions'; 'Storage Mode' with radio buttons for 'Exhaustive', 'Bitstate hashing/Supertrace', and 'Hash-compact', and checkboxes for 'Minimized automata' and 'Collapse compression'; 'User Parameters' with a checkbox for 'Use these parameters' and input fields for 'Compile-time' and 'Run-time'; and 'Never Claim Specification' with a checked checkbox for 'Apply never claim (if present) using:' and radio buttons for 'In-model LTL formula/claim name:' (with a text field containing 'p1'), 'LTL formula in the text field:', 'LTL formula in a 1-line file:', and 'Never claim in a file:' (each with a 'Browse' button).

SpinRCP teljes nézet

The screenshot displays the SpinRCP software interface, version 3.1.0, running on 30 December 2016. The interface is divided into several panes:

- Model Navigator:** Shows a tree view of project files, including examples, exercises, and various .pmi and .pdf files.
- Code Editor:** Displays the source code for `leader.pmi`. The code defines a process with a queue and a selection mechanism for a winner among five processes.
- MSC Viewer:** Shows a sequence diagram with five vertical lifelines labeled `node2` through `node5`. Messages labeled `winner:5` are sent from the lifelines to a common horizontal line.
- Console:** Shows the output of a random simulation, including process creation, termination, and the final state of the simulation.
- Simulation / Replay:** Contains controls for running, stopping, and stepping through the simulation. It also shows simulation view options and data values.
- Spin Trail to MSC:** Provides options for converting a spin trail file to an MSC file, including renaming channels and creating virtual processes.
- Simulation Data Values and Queues:** Displays the current state of the simulation, including variable values and queue contents.

The console output shows the following sequence of events:

```

197: proc 6 (nnode:1) leader.pmi:91 (state 44)
MSC: LOST
198: proc 5 (nnode:1) leader.pmi:81 (state 32)
199: proc 5 (nnode:1) leader.pmi:87 (state 38)
200: proc 5 (nnode:1) leader.pmi:88 (state 39)
201: proc 5 (nnode:1) leader.pmi:91 (state 45)
202: proc 5 (nnode:1) leader.pmi:91 (state 44)
202: proc 5 (nnode:1) terminates
202: proc 4 (nnode:1) terminates
202: proc 3 (nnode:1) terminates
202: proc 2 (nnode:1) terminates
202: proc 1 (nnode:1) terminates
202: proc 0 (init:1) terminates
6 processes created
Random simulation trail written to leader.pmi.xnd
    
```

The simulation data values and queues are as follows:

Variable values	Queue contents values
nnode(2):Active = 0	queue 1 (nnode(1):inp)
nnode(2):neighbourR = 1	queue 2 (nnode(2):inp)
nnode(2):nr = 5	queue 3 (nnode(3):inp)
nnode(3):Active = 0	queue 4 (nnode(4):inp)
nnode(3):neighbourR = 3	queue 5 (nnode(5):inp)
nnode(3):nr = 5	
nnode(4):Active = 0	
nnode(4):neighbourR = 4	
nnode(4):nr = 5	
nnode(5):know_winner =	
nnode(5):maximum = 5	
nnode(5):neighbourR = 5	
nnode(5):nr = 5	
nr_leaders = 1	