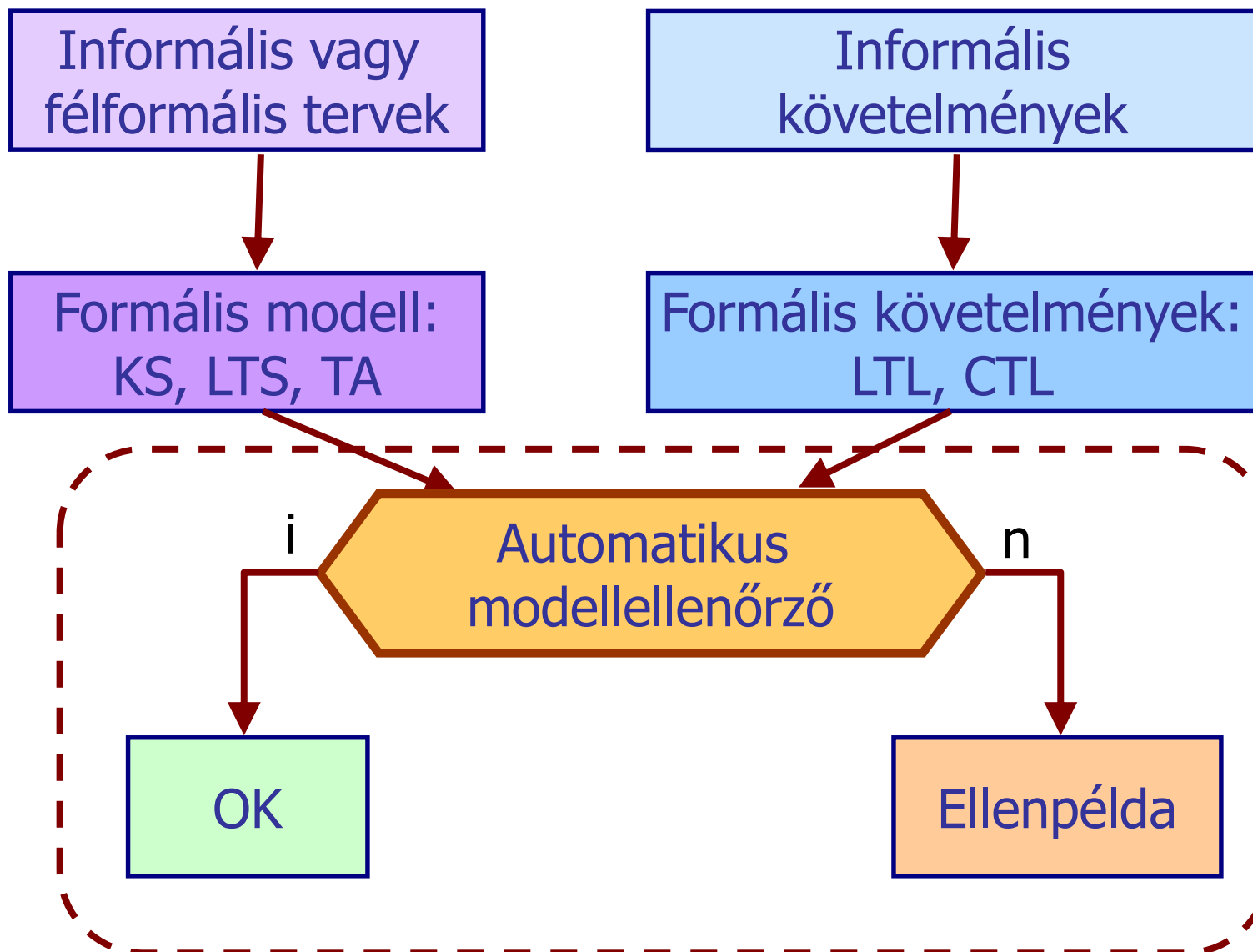


Modellellenőrző eszközök: A SPIN modellellenőrző (kedvcsináló)

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
}
```

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

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

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 algoritmus (assert)

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

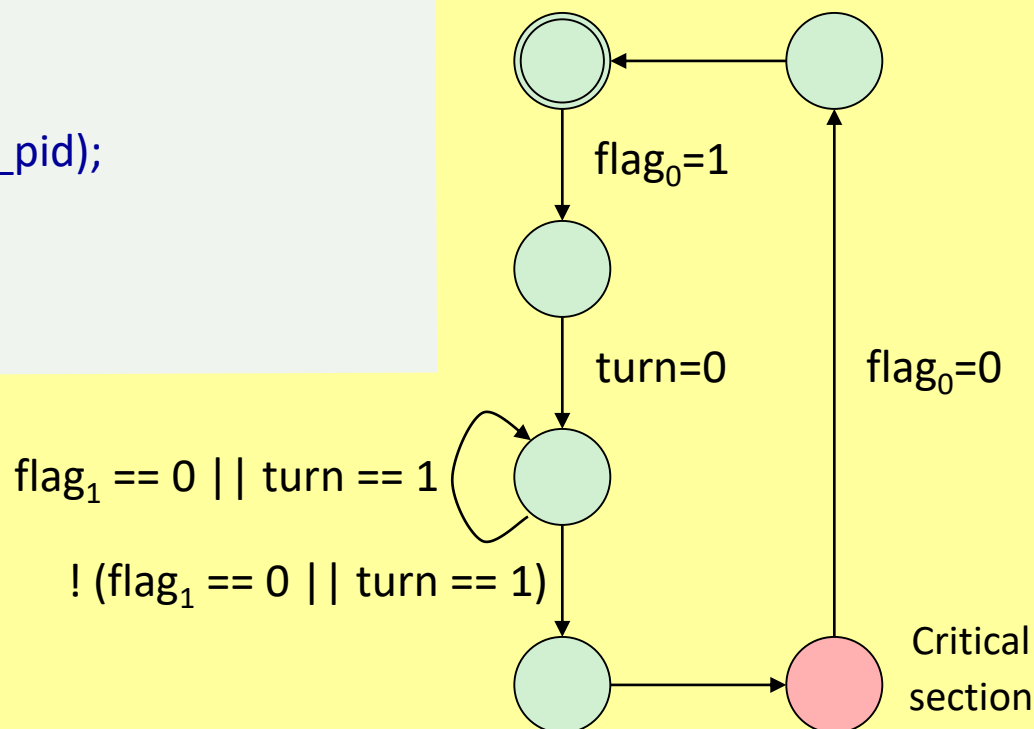
```
active [2] proctype user()    // two processes with built-in identifier _pid
{
```

```
    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 algoritmus (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:

$[] \neg(\text{critical}[0] \ \&\& \ \text{critical}[1])$

$[] \langle \rangle (\text{critical}[0])$

$[] \langle \rangle (\text{critical}[1])$

$[] (\text{critical}[0] \rightarrow$
 $(\text{critical}[0] \cup$
 $(\neg \text{critical}[0] \ \&\&$
 $((\neg \text{critical}[0] \ \&\&$
 $\neg \text{critical}[1]) \cup \text{critical}[1])))$

$[] (\text{critical}[1] \rightarrow$
 $(\text{critical}[1] \cup$
 $(\neg \text{critical}[1] \ \&\&$
 $((\neg \text{critical}[1] \ \&\&$
 $\neg \text{critical}[0]) \cup \text{critical}[0])))$

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 rcvbit;  
    do  
        :: StoR ? MSG, rcvbit ->  
            RtoS ! ACK, rcvbit  
        :: StoR ? MSG, rcvbit ->  
            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 rcvbit;  
    do  
        :: StoR ? MSG, rcvbit ->  
            RtoS ! ACK, rcvbit  
        :: StoR ? MSG, rcvbit ->  
            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

The screenshot shows the 'Verification' dialog box in the SpinRCP Eclipse RCP environment. The dialog has a title bar with navigation icons. Below the title bar, it displays the 'Verification Profile' as 'MyVerificationProfile.xml (created on Dec 30 2016 at 14:52:17 CET)'. There are three buttons: 'Import', 'Export', and 'Reload'. The dialog is divided into three tabs: 'Basic Options', 'Advanced Options', and 'Iterative/Swarm Run'. The 'Basic Options' tab is active. It contains several sections: 'Correctness Properties' with radio buttons for 'Safety (state properties)', 'Liveness (cycles/sequences)', and 'Non-progress cycles', and checkboxes for 'Assertion violations', 'Invalid end states', 'Acceptance cycles', 'Add weak fairness', 'Report unreachable code', and 'Check xr/xs assertions'. 'Storage Mode' has radio buttons for 'Exhaustive', 'Minimized automata', 'Collapse compression', 'Bitstate hashing/Supertrace', and 'Hash-compact'. 'User Parameters' has a checkbox for 'Use these parameters:' and input fields for 'Compile-time:' and 'Run-time:'. 'Never Claim Specification' has a checked checkbox for 'Apply never claim (if present) using:' and radio buttons for 'In-model LTL formula/claim name:', 'LTL formula in the text field:', 'LTL formula in a 1-line file:', and 'Never claim in a file:', each with an associated input field and a 'Browse' button.

Verification

Verification Profile
MyVerificationProfile.xml (created on Dec 30 2016 at 14:52:17 CET)

Import Export Reload

Basic Options Advanced Options Iterative/Swarm Run

Correctness Properties

☐ Safety (state properties)

☒ Assertion violations

☒ Invalid end states

☒ Liveness (cycles/sequences)

☐ Non-progress cycles

☒ Acceptance cycles

☐ Add weak fairness

☐ Report unreachable code

☒ Check xr/xs assertions

Storage Mode

☒ Exhaustive

☐ Minimized automata

☐ Collapse compression

☐ Bitstate hashing/Supertrace

☐ Hash-compact

User Parameters

☐ Use these parameters:

Compile-time:

Run-time:

Never Claim Specification

☒ Apply never claim (if present) using:

☒ In-model LTL formula/claim name:

☐ LTL formula in the text field:

☐ LTL formula in a 1-line file: Browse

☐ Never claim in a file: Browse

SpinRCP teljes nézet

