

# Formalne Metode u oblikovanju sustava

FER

drugi ciklus predavanja, treće predavanje  
ver. 0.1.8  
nadm.zadnje.rev.: 2. svibnja 2009.

# Ponavljjanje

- 1 *Promela* jezik
- 2 LTL formule
- 3 Instalacija i primjer (*Hello*)

## Napomena:

- zadanje predavanje u ovom ciklusu ima naglasak na praktičnoj upotrebi
- primjere pokušati samostalno rješavati sa računalom ...

# Spin i XSpin

## Spin

- *Spin* nema grafičko sučelje
- poziva se iz komandne linije:
- `spin --help` pokazuje sve dostupne opcije

## XSpin

- `Xspin` je grafičko sučelje za *Spin*
- `Xspin` je preprocesor i vizualizator za `spin`
- `Xspin` je napisan u TclTk skriptnom jeziku

## Korištenje alata *Spin*

- 1 programskom alatu *Spin* pristupamo preko jezika *Promela*.
- 2 *Spin* koristimo samostalno za analizu . . .
- 3 *Spin* koristimo kao dio nekog programskog alata
- 4 osnovna namjena: *Spin* služi za verifikaciju konkurentnih reaktivnih procesa
- 5 osim verifikacije *Spin* nalazi primjenu i u mnogim ostalima područjima razvoja programske potpore

## Kako uključiti Spin u ciklus razvoja programske potpore ?

- 1 kao analizator–verifikator: potrebno je razviti i analizirati modele
- 2 kao alat koje je već dio programskog alata ili eng. *model extractor* (npr. *JavaPathfinder* ...)
- 3 aplikacija / problem od interesa koristi algoritme unutar *Spin* alata. Tada **proširujemo** vlastiti programski alat ili aplikaciju (eng. *embedding*).
- 4 apstraktne strukture podataka su podržane preko *Promela* `typedef` konstrukta.

## Ograničenja *Spin Promela* alata:

- 1 konačni broj procesa
- 2 konačni broj stanja po procesu
- 3 varijable moraju biti ograničene
- 4 eksplozija stanja: (algoritmi za izračun grafova dostupnosti nisu polinomno kompletni!)
- 5 nemogućnost dinamičke deklaracije novih procesa

Spomenuta ograničenja odnose se na svaku analizu/verifikaciju provjerom modela  
(na svaki *model checking* program). . .

→ Programski alat *Spin* je industrijski relevantan proizvod (eng. *industrial strength tool*).

→ Do sada postoji mnogo značajnih primjena  
*Kako uspješno koristiti Spin programski alat ?*

- 1 **konačni** broj stanja, procesa i ograničenost varijabli: *modeliranje* je, načelno govoreći proces apstrakcije koji nije egzaktn. Pažljivim izborom apstrahiranja–modeliranja može se rješavati veliki broj problema
- 2 **explozija stanja**: *Spin* stanja kodira preko posebne tehnike (*bit–state–hashing*, koristi naprednu metodologiju za smanjenje broja stanja zbog pravilnosti u modelima, a i korisnik preko opcija može utjecati na kompleksnost analize)

Za one koji hoće više:

Na *Spin* stranici ([spinroot.comspin/whatispin.html](http://spinroot.comspin/whatispin.html)) proučite primjere industrijske primjene

Da li možete vlastite zadatke ili projekte opisati i analizirati

*Spin/Promela* modelima ?



## Spin u analizi modela

- Neka je problem za verifikaciju ili analizu opisan i iskazan u jeziku *Promela* ...
- problem se nalazi u datoteci `model.prm`...
- nakon uobičajenog uvodnog `hello.prm` modela ...
- slijede dodatne opcije koje se koriste prilikom verifikacije

### Primjer:

precrtajte sa ploče `model.prm` !  
na računalu sami pokrenite zadane opcije

## O radnom primjeru koji se rješava

Zadani primjer je poopćeni model konkurentnih procesa koje nalazimo u mnogim praktičnim primjenama:

- raspodjeljeni, konkurentni sustavi: komunikacija među procesima
- klijent server aplikacije
- *"mutex"* protokoli
- komunikacijski protokoli
- raspodjeljene *web* aplikacije
- komponente i oblikovni obrasci (eng. *"design patterns"*)

⇒ Pažnju usmjerite na **analizu** `modela.prm`

### Napomena:

→ **Modeliranje i apstrakcija** realnih problema je poseban problem i nije dobro istovremeno učiti modeliranje i korištenje

→ **Modeliranje i apstrakcija**  $\equiv$  definiranje *Promela modela* (tj. `modela.prm`)

## Spin opcije

```
spin -A model.prm
```

- To je prva opcija nakon što je definiran model.
- Provjerava se sintaksa i eventualni nekonzistentni konstrukti

## Spin opcije

```
spin -p model.prm
```

- pokreće simulaciju modela
- simulacija se vrši po slučajnom izboru
- opcija  $-nN$  sa npr.  $N = 12$  inicijalizira generator slučajnih brojeva
- ⇒ simulacijom se dobiva osnovni uvid o ponašanju modela

## Spin opcije

```
spin -p -c -u200 -j10 model.prm
```

- simulacija se zaustavlja nakon 200 koraka
- prvih 10 koraka u simulaciji se preskače

## Spin opcije

```
spin -l -g -r -s -c model.prm
```

- opcije pokazuju lokalne i globalne varijable kao i prijem odnosno predaju poruka
- simulacija pokazuje konzistentnost: ponašanje modela prema očekivanjima
- simulacija ukazuje na greške zbog nepoznavanja *Promela* semantike i sintakse

## Spin opcije

```
spin -a model.prm
```

```
spin -a -f 'ltl-formula' model.prm ili
```

```
spin -a -F LTL-file model.prm
```

- opcija `spin -a` generira u jeziku *C* analizator (`pan.[bchmt]`)
- `spin -a -f` i `spin -a -F` pridodaju *LTL* formule analizatoru
- *Spin* transformira *LTL* formule u Büchi automat
- Büchi automat je u *Promela* modelu kodiran sa `never { }` konstruktom
- moguće je i kreirati vlastite Büchi automate editiranjem `never { }` konstrukta



## Spin opcije

```
spin -a model.prm  
gcc -o pan pan.c  
./pan ili pan.exe
```

- generiranje analizatora kao `pan` ili `pan.exe` izvršnog programa
- spomenimo i dodatnu opciju `spin -m` kao intervenciju u *Promela* semantiku: predaja je uvijek izvršna, (nije blokirajuća) iako je rep pun

## Izlaz *pan* analizatora: rezultati

(Spin Version 5.1.7 -- 23 December 2008)

+ Partial Order Reduction

Full statespace search for:

never claim - (none specified)

assertion violations +

acceptance cycles - (not selected)

invalid end states +

State-vector 20 byte, depth reached 49, errors: 0

148 states, stored

129 states, matched

277 transitions (= stored+matched)

0 atomic steps

hash conflicts: 0 (resolved)

2.501 memory usage (Mbyte)

unreached in proctype mutex

line 26, state 23, "-end-"

(1 of 23 states)

pan: elapsed time 0 seconds

## Izlaz *pan* analizatora: rezultati

- Full statespace search for: pokazuje kao se provjerava model (npr. pogrešna završna stanja i pogrešne tvrdnje)
- State-vector: opisuje dubinu i veličinu grafa
- **errors: 0** – ukazuje na odsutnost grešaka. Pojava greške generira datoteku sa "tragom" (eng. *error trail*), tako da je moguće precizno utvrditi kojom sekvencom instrukcija dolazi do greške.
- unreached: pokazuje djelove modela koji su nedostupni

### Napomena:

Analizator pokaže koje su analize provedene

- Što zaista znače rezultati odnosno izlazne poruke analizatora ?

## Spin opcije

```
spin -t -p model.prm
```

→ ako analizator javi grešku opcija `-t` ispisuje sekvencu koja vodi prema greški

```
spin --help i pan --help
```

## Sve opcije

→ pokazuje sve opcije: preporuka je koristiti samo opcije koje za koje se detaljno razumije semantika

## Pan opcije

```
gcc -o pan pan.c
```

- generiranje analizatora (*pan*) svodi se na upotrebu programa prevodioca
- preporuča se upotreba *gcc* prevodioca

## Spin direktive prevodiocu

```
gcc -Dxxx -o pan pan.c
```

→ ponekad je potrebno prevoditi sa dodatnim opcijama:

- \* **-DNP** otkriva cikluse koje nemaju progresu (napretka)
- \* **-DBFS** generiranje stabla dostupnosti po širini (*breadth-first*) umjesto po dubini (*depth-first*)

## Spin direktive prevodiocu (primjeri)

```
gcc -DMEMLIM=512 -o pan pan.c
```

```
gcc -DHC4 -o pan pan.c
```

```
gcc -DBITSTATE -o pan pan.c
```

### Kako *Spin* maksimalno iskoristi postojeće ...

- problem svakog analizatora su memorijska ograničenja, opcije optimiziraju korištenje memorije
- potrebno je kroz niz iteracija pronaći optimalnu upotrebu s obzirom na model i dostupne resurse
- *Spin* kodira (preko  $10^{22}$ ) stanja primjenom BITSTATE-hash kodiranja



## Pan opcije

pan -w23

*Spin* pokušava sve izračunati unutar  $10^{18}$  stanja, ako nije dovoljno, opcija -w23 povećava na  $10^{23}$  stanja

## *Pan* opcije

```
pan -m100000
```

*Spin* može ograničiti i dubinu pretraživanja, u ovom slučaju na 100000.

## Pan opcije

$\text{pan} \neg a$  i  
 $\text{pan} \neg 1$

analiza za provjeru istinitosti *LTL* formula ( $\text{pan} \neg a$ )  
odnosno "petlji bez napretka" ( $\text{pan} \neg 1$ )

## Zaključak i nastavak ...

- (1) analizirajte gotove primjere modela koji slijede !
- (2) nakon toga pokušajte kreirati svoje *Promela* modele !
- (3) procijenite upotrebljivost programskog alata *Spin/Promela* na vašim primjerima !
- (4) planirajte daljnje učenje oko razvoja programske potpore ...
- (5) stalno pratite literaturu: verifikacija, modeliranje je područje koje se razvija i u kome slijedi još rezultata ...
- (6) pročitajte zadnje dvije folije: one ilustriraju sadašnju situaciju oko razvoja programske potpore ...
- (7) povežite gradivo ovog ciklusa sa ostalim ciklusima predavanja ili odgovorite na pitanje: **"How designer designs ?"**

## Za vježbu:

Slijede tri primjera protokol (*Bartlett*), "*Produce–Consumer*" i *Dekker* mutex protokol

- a) nacrtajte pripadne *FSM* !
- b) pokrenite simulacijski mod `spin -c -p ...`

## Za one koje žele više:

- generirajte verifikator (`pan`) !
- Kako bi pokazali odsutnost npr. "deadlocka" ?

# Bartlett protokol

```
#define MAX 4                      /* file ex.2 */
proctype A(chan in, out)
{
  byte mt; /* message data */
  bit  vr;

S1: mt = (mt+1)%MAX;
   out!mt,1;
   goto S2;

S2: in?vr;
   if
   :: (vr == 1) -> goto S1
   :: (vr == 0) -> goto S3
   :: printf("MSC: AERROR1\n") -> goto S5
   fi;

S3: out!mt,1;
   goto S2;

S4: in?vr;
   if
   :: goto S1
   :: printf("MSC: AERROR2\n"); goto S5
   fi;

S5: out!mt,0;
   goto S4
}
```

## Primjer - Bartlett protokol

```
proctype B(chan in, out)
{
  byte mr, lmr;
  bit ar;
  goto S2; /* initial state */
S1: assert(mr == (lmr+1)%MAX);
  lmr = mr;
  out!1;
  goto S2;
S2: in?mr,ar;
  if
  :: (ar == 1) -> goto S1
  :: (ar == 0) -> goto S3
  :: printf("MSC: ERROR1\n"); goto S5
  fi;
S3: out!1;
  goto S2;
S4: in?mr,ar;
  if
  :: goto S1
  :: printf("MSC: ERROR2\n"); goto S5
  fi;
S5: out!0;
  goto S4
}
}
```

## Primjer - Bartlett protokol

```
init {  
    chan a2b = [2] of { bit };  
    chan b2a = [2] of { byte, bit };  
    atomic {  
        run A(a2b, b2a);  
        run B(b2a, a2b)  
    }  
}
```



# Primjer - "Producer Consumer"

```
mtype = { P, C };

mtype turn = P;

active proctype producer()
{
do
:: (turn == P) ->
printf("Produce\n");
turn = C
od
}

active proctype consumer()
{
do
:: (turn == C) ->
printf("Consume\n");
turn = P
od
}
```

# Primjer - Dekker mutex

```
bit turn; bool flag[2]; byte cnt;
  active [2] proctype mutex() /* Dekker's 1965 algorithm */
  { pid i, j;
    i = _pid;
    j = 1 - _pid;
    again:
    flag[i] = true;

do
  :: flag[j] ->
  if
  :: turn == j ->
  flag[i] = false;
  !(turn == j);
  flag[i] = true
  :: else
  fi
  :: else ->
  break
  od;

  cnt++;
  assert(cnt == 1); /* critical section */
  cnt--;
  turn = j;
  flag[i] = false;
  goto again
}
```

## Šira literatura: (ponovljeno)

- (1.) Gerard J. Holzmann: The SPIN Model Checker–Primer and Reference Manual
- (2.) <http://spinroot.com/spin/Man/index.html>

## C.A.R. Hoare

CACM, 03/2009 Vol.52 No3.3 interview pp.41

*As far as the fundamental science is concerned, we still certainly do not know how to prove programs correct. We need a lot of steady progress in this area, which one can foresee, and a lot of breakthroughs where people suddenly find there is a simple way to do something that everybody hitherto has thought to be far too difficult.*

## Fortune Magazine: BrainstormTech.

IEEE Spectrum INT, September/2008 pp.05

**The Future of code** quote from user guide that comes with Your (. . . \*) computer

*This computer is not intended for use in the operation of nuclear facilities, aircraft navigation or communication systems, or air traffic control machines, or for any other uses where the failure of your computer system could lead to death, personal injury, or severe environmental damage.*

Pomaže li uvijek **Cntrl–Alt–Del** ili **Esc** ?