

Formalne Metode u oblikovanju sustava

FER

drugi ciklus predavanja, drugo predavanje
ver. 0.1.8
nadm.zadnje.rev.: 24. travnja 2009.

Ponavljjanje

- 1 Teorijska podloga: automati (*CFSM*, *FSM* i logika (*LTL*))
- 2 Modeliranje: procesi i *Promela* jezik
- 3 Protokoli i procesi u raspodjeljenim, konkurentnim i reaktivnim sustavima

Ponavljjanje

- 1 Teorijska podloga: automati (*CFSM*, *FSM* i logika (*LT**L*))
- 2 Modeliranje: procesi i *Promela* jezik
- 3 Protokoli i procesi u raspodjeljenim, konkurentnim i reaktivnim sustavima

Ponavljjanje

- 1 Teorijska podloga: automati (*CFSM*, *FSM* i logika (*LTL*))
- 2 Modeliranje: procesi i *Promela* jezik
- 3 Protokoli i procesi u raspodjeljenim, konkurentnim i reaktivnim sustavima

... slijede detalji o *Promela* jeziku

Promela model ili program se sastoji od:

- 1 deklaracije *tipova* podataka (eng. *type declaration*)
- 2 deklaracije *globalnih* varijabli (eng. *global variable declaration*)
- 3 deklaracije *komunikacijskih kanala* (eng. *channel declaration*)
- 4 deklaracije *procesa* (eng. *process declaration*)
- 5 deklaracije *početnog*, zajedničkog procesa (eng. *init process declaration*)

Promela i FSM

- 1 Poželjno je uvijek uočiti istoznačnost *FSM* i *Promela* procesa
- 2 Često se koristi kao sinonim *Promela* model ili *Promela* program

... slijede detalji o *Promela* jeziku

Promela model ili program se sastoji od:

- 1 deklaracije *tipova* podataka (eng. *type declaration*)
- 2 deklaracije *globalnih* varijabli (eng. *global variable declaration*)
- 3 deklaracije *komunikacijskih kanala* (eng. *channel declaration*)
- 4 deklaracije *procesa* (eng. *process declaration*)
- 5 deklaracije *početnog*, zajedničkog procesa (eng. *init process declaration*)

Promela i FSM

- 1 Poželjno je uvijek uočiti istoznačnost *FSM* i *Promela* procesa
- 2 Često se koristi kao sinonim *Promela* model ili *Promela* program

... slijede detalji o *Promela* jeziku

Promela model ili program se sastoji od:

- 1 deklaracije *tipova* podataka (eng. *type declaration*)
- 2 deklaracije *globalnih* varijabli (eng. *global variable declaration*)
- 3 deklaracije *komunikacijskih kanala* (eng. *channel declaration*)
- 4 deklaracije *procesa* (eng. *process declaration*)
- 5 deklaracije *početnog*, zajedničkog procesa (eng. *init process declaration*)

Promela i FSM

- 1 Poželjno je uvijek uočiti istoznačnost *FSM* i *Promela* procesa
- 2 Često se koristi kao sinonim *Promela* model ili *Promela* program

... slijede detalji o *Promela* jeziku

Promela model ili program se sastoji od:

- 1 deklaracije *tipova* podataka (eng. *type declaration*)
- 2 deklaracije *globalnih* varijabli (eng. *global variable declaration*)
- 3 deklaracije *komunikacijskih kanala* (eng. *channel declaration*)
- 4 deklaracije *procesa* (eng. *process declaration*)
- 5 deklaracije *početnog*, zajedničkog procesa (eng. *init process declaration*)

Promela i FSM

- 1 Poželjno je uvijek uočiti istoznačnost *FSM* i *Promela* procesa
- 2 Često se koristi kao sinonim *Promela* model ili *Promela* program

... slijede detalji o *Promela* jeziku

Promela model ili program se sastoji od:

- 1 deklaracije *tipova* podataka (eng. *type declaration*)
- 2 deklaracije *globalnih* varijabli (eng. *global variable declaration*)
- 3 deklaracije *komunikacijskih kanala* (eng. *channel declaration*)
- 4 deklaracije *procesa* (eng. *process declaration*)
- 5 deklaracije *početnog*, zajedničkog procesa (eng. *init process declaration*)

Promela i *FSM*

- 1 Poželjno je uvijek uočiti istoznačnost *FSM* i *Promela* procesa
- 2 Često se koristi kao sinonim *Promela* model ili *Promela* program

... slijede detalji o *Promela* jeziku

Promela model ili program se sastoji od:

- 1 deklaracije *tipova* podataka (eng. *type declaration*)
- 2 deklaracije *globalnih* varijabli (eng. *global variable declaration*)
- 3 deklaracije *komunikacijskih kanala* (eng. *channel declaration*)
- 4 deklaracije *procesa* (eng. *process declaration*)
- 5 deklaracije *početnog*, zajedničkog procesa (eng. *init process declaration*)

Promela i *FSM*

- 1 Poželjno je uvijek uočiti istoznačnost *FSM* i *Promela* procesa
- 2 Često se koristi kao sinonim *Promela* model ili *Promela* program

... slijede detalji o *Promela* jeziku

Promela model ili program se sastoji od:

- 1 deklaracije *tipova* podataka (eng. *type declaration*)
- 2 deklaracije *globalnih* varijabli (eng. *global variable declaration*)
- 3 deklaracije *komunikacijskih kanala* (eng. *channel declaration*)
- 4 deklaracije *procesa* (eng. *process declaration*)
- 5 deklaracije *početnog*, zajedničkog procesa (eng. *init process declaration*)

Promela i *FSM*

- 1 Poželjno je uvijek uočiti istoznačnost *FSM* i *Promela* procesa
- 2 Često se koristi kao sinonim *Promela* model ili *Promela* program

A što je u jeziku *Promela* izvršno ... ?

- ... upravo smo naveli samo **deklaracije** u jeziku *Promela*
- koje nalazimo i u npr. jeziku C (npr. `short int i=1, float a;` i sl.)
- gdje su izvršne naredbe npr. `i++; a=a*1+2; func10(a,i);`
- u jeziku *Promela* osnovna izvršna jedinica je **proces**

- ① Programski alat *Spin* opisuje ponašanje sustava kao skupa potencijalno interaktivnih, asinkronih, komunicirajućih dretvi, niti, tredova (eng. *threads*)
- ② deklaracija procesa (proctype konstrukt) opisuje ponašanje ali **izvršivost** (eng. *executability*) možemo postići jedino eksplicitnim pozivom procesa (preko `active` proctype ili sa `run` konstruktom)

A što je u jeziku *Promela* izvršno ... ?

- ... upravo smo naveli samo **deklaracije** u jeziku *Promela*
- koje nalazimo i u npr. jeziku C (npr. `short int i=1, float a;` i sl.)
- gdje su izvršne naredbe npr. `i++; a=a*1+2; func10(a,i);`
- u jeziku *Promela* osnovna izvršna jedinica je **proces**

- 1 Programski alat *Spin* opisuje ponašanje sustava kao skupa potencijalno interaktivnih, asinkronih, komunicirajućih dretvi, niti, tredova (eng. *threads*)
- 2 deklaracija procesa (proctype konstrukt) opisuje ponašanje ali **izvršivost** (eng. *executability*) možemo postići jedino eksplicitnim pozivom procesa (preko `active` proctype ili sa `run` konstruktom)

A što je u jeziku *Promela* izvršno ... ?

- ... upravo smo naveli samo **deklaracije** u jeziku *Promela*
- koje nalazimo i u npr. jeziku C (npr. `short int i=1, float a;` i sl.)
- gdje su izvršne naredbe npr. `i++; a=a*1+2; func10(a,i);`
- u jeziku *Promela* osnovna izvršna jedinica je **proces**

- 1 Programski alat *Spin* opisuje ponašanje sustava kao skupa potencijalno interaktivnih, asinkronih, komunicirajućih dretvi, niti, tredova (eng. *threads*)
- 2 deklaracija procesa (`proctype` konstrukt) opisuje ponašanje ali **izvršivost** (eng. *executability*) možemo postići jedino eksplicitnim pozivom procesa (preko `active proctype` ili sa `run` konstruktom)

A što je u jeziku *Promela* izvršno ... ?

- ... upravo smo naveli samo **deklaracije** u jeziku *Promela*
- koje nalazimo i u npr. jeziku C (npr. `short int i=1, float a;` i sl.)
- gdje su izvršne naredbe npr. `i++; a=a*1+2; func10(a,i);`
- u jeziku *Promela* osnovna izvršna jedinica je **proces**

- 1 Programski alat *Spin* opisuje ponašanje sustava kao skupa potencijalno interaktivnih, asinkronih, komunicirajućih dretvi, niti, tredova (eng. *threads*)
- 2 deklaracija procesa (`proctype` konstrukt) opisuje ponašanje ali **izvršivost** (eng. *executability*) možemo postići jedino eksplicitnim pozivom procesa (preko `active proctype` ili sa `run` konstruktom)

A što je u jeziku *Promela* izvršno ... ?

- ... upravo smo naveli samo **deklaracije** u jeziku *Promela*
- koje nalazimo i u npr. jeziku C (npr. `short int i=1, float a;` i sl.)
- gdje su izvršne naredbe npr. `i++; a=a*1+2; func10(a,i);`
- u jeziku *Promela* osnovna izvršna jedinica je **proces**

- 1 Programski alat *Spin* opisuje ponašanje sustava kao skupa potencijalno interaktivnih, asinkronih, komunicirajućih dretvi, niti, tredova (eng. *threads*)
- 2 deklaracija procesa (`proctype` konstrukt) opisuje ponašanje ali **izvršivost** (eng. *executability*) možemo postići jedino eksplicitnim pozivom procesa (preko `active proctype` ili `sa run` konstruktom)

Za vježbu:

Hello primjer i procesi

Modificirajte *Hello* primjer sa i bez *init* naredbe.

- 1) Što znači *active proctype* ?
- 2) Koliko ima ukupno procesa sa i bez *init* naredbe ?
- 3) Što znači *active proctype* bez *init* naredbe?
- 4) Pokrenite *./hello* i *./hello -d*. Što uočavate ?

Za vježbu:

Hello primjer i procesi

Modificirajte *Hello* primjer sa i bez *init* naredbe.

- 1) Što znači *active proctype* ?
- 2) Koliko ima ukupno procesa sa i bez *init* naredbe ?
- 3) Što znači *active proctype* bez *init* naredbe?
- 4) Pokrenite *./hello* i *./hello -d*. Što uočavate ?

Za vježbu:

Hello primjer i procesi

Modificirajte *Hello* primjer sa i bez *init* naredbe.

- 1) Što znači *active proctype* ?
- 2) Koliko ima ukupno procesa sa i bez *init* naredbe ?
- 3) Što znači *active proctype* bez *init* naredbe?
- 4) Pokrenite *./hello* i *./hello -d*. Što uočavate ?

Za vježbu:

Hello primjer i procesi

Modificirajte *Hello* primjer sa i bez *init* naredbe.

- 1) Što znači `active proctype` ?
- 2) Koliko ima ukupno procesa sa i bez *init* naredbe ?
- 3) Što znači `active proctype` bez *init* naredbe?
- 4) Pokrenite `./hello` i `./hello -d`. Što uočavate ?

Značenje ";"

- u jeziku C: ";" **završava** naredbu
- (eng. *statement **terminator***)
- u jeziku *Promela* ";" **razdvaja, odvaja** naredbu
(eng. *statement **separator***)

Važno:

Umjesto ";" možemo koristiti i "->" kao separator naredbi u jeziku *Promela*

Značenje ";"

- u jeziku C: ";" **završava** naredbu
- (eng. *statement terminator*)
- u jeziku *Promela* ";" **razdvaja, odvaja** naredbu
(eng. *statement separator*)

Važno:

Umjesto ";" možemo koristiti i "->" kao separator naredbi u jeziku *Promela*

Specifičnosti jezika Promela

Blokirajuće ili izvršne naredbe

Sve promela naredbe su *izvršne* ili *blokirajuće*

Blokirajuće naredbe su implementacija Dijkstrinih *guarded* komandi: one blokiraju samo do trenutka kada je uvjet **G** zadovoljen, a *nakon* toga se izvode slijedeće naredbe

Primjer:

```
(turn == P) -> printf("Produce") ...
```

tek onda i samo onda kada je varijabla `turn` jednaka `P` ispisuje se "Produce"

sve dok ta jednakost ne vrijedi ili *guard* – propozicija ne postane istinita proces je (privremeno) blokirao

Specifičnosti jezika Promela

Blokirajuće ili izvršne naredbe

Sve promela naredbe su *izvršne* ili *blokirajuće*

Blokirajuće naredbe su implementacija Dijkstrinih *guarded* komandi: one blokiraju samo do trenutka kada je uvjet **G** zadovoljen, a *nakon* toga se izvode slijedeće naredbe

Primjer:

```
(turn == P) -> printf("Produce") ...
```

tek onda i samo onda kada je varijabla `turn` jednaka `P` ispisuje se "Produce"

sve dok ta jednakost ne vrijedi ili *guard* – propozicija ne postane istinita proces je (privremeno) blokiran

Specifičnosti jezika Promela

Blokirajuće ili izvršne naredbe

Sve promela naredbe su *izvršne* ili *blokirajuće*

Blokirajuće naredbe su implementacija Dijkstrinih *guarded* komandi: one blokiraju samo do trenutka kada je uvjet **G** zadovoljen, a *nakon* toga se izvode slijedeće naredbe

Primjer:

```
(turn == P) -> printf("Produce") ...
```

tek onda i samo onda kada je varijabla `turn` jednaka `P` ispisuje se "Produce"

sve dok ta jednakost ne vrijedi ili *guard* – propozicija ne postane istinita *proces* je (privremeno) blokiran

Specifičnosti jezika Promela

Blokirajuće ili izvršne naredbe

Sve promela naredbe su *izvršne* ili *blokirajuće*

Blokirajuće naredbe su implementacija Dijkstrinih *guarded* komandi: one blokiraju samo do trenutka kada je uvjet **G** zadovoljen, a *nakon* toga se izvode slijedeće naredbe

Primjer:

```
(turn == P) -> printf("Produce") ...
```

tek onda i samo onda kada je varijabla `turn` jednaka `P` ispisuje se "Produce"

sve dok ta jednakost ne vrijedi ili *guard* – propozicija ne postane istinita `proces` je (privremeno) blokiran

Tipovi podataka

- ⇒ varijable u jeziku *Promela* su lokalne ili globalne
- ⇒ Osnovni tipovi podataka:

bit	$0 \dots 1$	bit OK=1;
bool	false...true	bool flag = false;
byte	$0 \dots 255$	byte foo;
chan	$1 \dots 255$	chan AtoB;
mtype	$1 \dots 255$	mtype msg;
pid	$0 \dots 255$	pid p;
short	$-2^{15} \dots 2^{15} - 1$	short a = 137;
int	$-2^{31} \dots 2^{31} - 1$	int i = 13;
unsigned	$0 \dots 2^n - 1$	unsigned u:3;

Tipovi podataka

Nema ...

`real, float, pointer`

kao tipovi podataka ne postoje u jeziku *Promela*

modelira se **koordinacija** među procesima a ne izvode se numerički proračuni

Napomene o podacima:

- 1 inicijalne vrijednosti svih varijabli (lokalnih i globalnih) su jednake 0
- 2 sve varijable moraju biti deklarirane prije upotrebe
- 3 deklaracija se može nalaziti bilo gdje u programu

Tipovi podataka

Nema ...

`real, float, pointer`

kao tipovi podataka ne postoje u jeziku *Promela*

modelira se **koordinacija** među procesima a ne izvode se numerički proračuni

Napomene o podacima:

- 1 inicijalne vrijednosti svih varijabli (lokalnih i globalnih) su jednake 0
- 2 sve varijable moraju biti deklarirane prije upotrebe
- 3 deklaracija se može nalaziti bilo gdje u programu

Tipovi podataka

Polja

- U jeziku *Promela* moguće je definirati jednodimenzionalna polja
- Vrijednosti indeksa polja kreću od nule kao i kod jezika C

Primjer polja:

```
bit a[11];  
byte tr224[99];  
...
```

Tipovi podataka

Polja

- U jeziku *Promela* moguće je definirati jednodimenzionalna polja
- Vrijednosti indeksa polja kreću od nule kao i kod jezika C

Primjer polja:

```
bit a[11];  
byte tr224[99];  
...
```

Tipovi podataka

Polja

- U jeziku *Promela* moguće je definirati jednodimenzionalna polja
- Vrijednosti indeksa polja kreću od nule kao i kod jezika C

Primjer polja:

```
bit a[11];  
byte tr224[99];  
...
```


Tipovi podataka

U jeziku *Promela* korisnik može definirati vlastite tipove podataka (sintaksa slijedi jezik C):

typedef primjer:

```
typedef adtStruct {  
    short foo29;  
    byte vxcount = 12;  
}
```

```
adtStruct serverstatus;  
serverstatus.vxcount = 159;
```

Napomena:

Definiranje *vlastitih podatkovnih struktura* u jeziku *Promela* znatno proširuje dosege upotrebe programskog alata *Spin* (u velikoj mjeri podržani su apstraktni tipovi podataka)

Osim toga u jeziku *Promela* moguće je ubaciti i dijelove pisane u jeziku *C* (eng. *C embedded code*)

Napomena:

Definiranje *vlastitih podatkovnih struktura* u jeziku *Promela* znatno proširuje dosege upotrebe programskog alata *Spin* (u velikoj mjeri podržani su apstraktni tipovi podataka)

Osim toga u jeziku *Promela* moguće je ubaciti i dijelove pisane u jeziku *C* (eng. *C embedded code*)

O komunikaciji. . .

. . . preko kanala

Promela procesi komuniciraju preko kanala (chan)

Kanale je potrebno deklarirati

Kanali su **globalnog** karaktera

Problem: odrediti kapacitet kanala u općem slučaju

. . . i preko globalnih varijabli

Promela procesi komuniciraju i preko globalnih varijabli

Problem: prava dostupnosti globalnim varijablama ("mutual exclusion")

O komunikaciji. . .

. . . preko kanala

Promela procesi komuniciraju preko kanala (chan)

Kanale je potrebno deklarirati

Kanali su **globalnog** karaktera

Problem: odrediti kapacitet kanala u općem slučaju

. . . i preko globalnih varijabli

Promela procesi komuniciraju i preko globalnih varijabli

Problem: prava dostupnosti globalnim varijablama ("mutual exclusion")

Kanali s porukama

Primjer:

Precrtajte u bilježnicu primjer UML sekvencnog (ili MSC dijagrama)

Promela procesi komuniciraju slanjem/prijemom poruka kroz kanale:

```
chan AtoB = [10] of {int, short, bit}
```

- **chan** deklaracija komunikacijskog kanala
- **AtoB** ime kanala
- **[10]** kapacitet kanala: maksimalni broj poruka kapacitet **[0]** znači sinkronu izmjenu poruka
- **{int, short, bit}** struktura poruke koja se šalje kroz kanal

Kanali s porukama

Primjer:

Precrtajte u bilježnicu primjer UML sekvencnog (ili MSC dijagrama)

Promela procesi komuniciraju slanjem/prijemom poruka kroz kanale:

```
chan AtoB = [10] of {int, short, bit}
```

- **chan** deklaracija komunikacijskog kanala
- **AtoB** ime kanala
- **[10]** kapacitet kanala: maksimalni broj poruka kapacitet **[0]** znači sinkronu izmjenu poruka
- **{int, short, bit}** struktura poruke koja se šalje kroz kanal

Kanali s porukama

Primjer:

Precrtajte u bilježnicu primjer UML sekvencnog (ili MSC dijagrama)

Promela procesi komuniciraju slanjem/prijemom poruka kroz kanale:

```
chan AtoB = [10] of {int, short, bit}
```

- **chan** deklaracija komunikacijskog kanala
- **AtoB** ime kanala
- **[10]** kapacitet kanala: maksimalni broj poruka kapacitet **[0]** znači sinkronu izmjenu poruka
- **{int, short, bit}** struktura poruke koja se šalje kroz kanal

Kanali s porukama

Primjer:

Precrtajte u bilježnicu primjer UML sekvencnog (ili MSC dijagrama)

Promela procesi komuniciraju slanjem/prijemom poruka kroz kanale:

```
chan AtoB = [10] of {int, short, bit}
```

- `chan` deklaracija komunikacijskog kanala
- `AtoB` ime kanala
- `[10]` kapacitet kanala: maksimalni broj poruka kapacitet `[0]` znači sinkronu izmjenu poruka
- `{int, short, bit}` struktura poruke koja se šalje kroz kanal

mtype deklaracija

deklaracija tipa poruke omogućuje pojednostavljeno rukovanje porukama:

osim standardnih tipova `mtype` je ugrađeni tip koji se tipično koristi unutar kanala:

```
chan toServer = [2] of { mtype, data, adress0 }
```

Primjer:

- `mtype = ack, req, setFGL;`
- `mtype m;` – neinicijalizirana poruka ima vrijednost 0
- `mtype mblockA = wsdp;` – inicijalizirana poruka, ima vrijednost različitu od 0

Dozvoljeno je do 255 različitih poruka.

mtype deklaracija

deklaracija tipa poruke omogućuje pojednostavljeno rukovanje porukama:

osim standardnih tipova `mtype` je ugrađeni tip koji se tipično koristi unutar kanala:

```
chan toServer = [2] of { mtype, data, adress0 }
```

Primjer:

- **mtype** = ack, req, setFGL;
- **mtype** m; – neinicijalizirana poruka ima vrijednost 0
- **mtype** mblockA = wsdp; – inicijalizirana poruka, ima vrijednost različitu od 0

Dozvoljeno je do 255 različitih poruka.

Prijem i predaja poruka

Sintaksa

Sintaksa simbola predaje i prijema je preuzeta iz *CSP* algebre:

- simbol "!" se koristi za predaju
- simbol "?" se koristi za prijem
- prijem/predaja znače stavljanje/uzimanje poruke u kanal koji opisuje komunikaciju između dva procesa
- naredba sa prijemom/predajom je izvršna ako kanal nije prazan/pun
- ponašanje kanala slično je ponašanju repa (eng. *queue*)

Prijem i predaja

Prijem

ch?*const*₁ ili *var*₁ ...*const*_n ili *var*_n

*const*_i i *var*_i moraju odgovarati poljima u poruci

Predaja

ch! *expr*₁ ... *expr*_n

*expr*_i mora po tipu odgovarati poljima u poruci

Prijem i predaja

Prijem

ch?*const*₁ ili *var*₁ ...*const*_n ili *var*_n

*const*_i i *var*_i moraju odgovarati poljima u poruci

Predaja

ch!*expr*₁ ...*expr*_n

*expr*_i mora po tipu odgovarati poljima u poruci

Primjeri

```
mtype = req;    chan chn = [N] of mtype, bit;  
bit nmsg;  
chn?req,nmsg;  
chn!ack,1;
```

Za vježbu:

Da li je moguće komunikaciju riješiti
bez korištenja `chan` te prijema/ predaje ?
Obrazložite mogućnosti !

Prijelazi u *FSM* i *Promela* naredbe

Svaki *Promela* proces predstavlja *FSM*.

Prijelaze (*T* prema *FSM* definiciji) možemo definirati:

- uvijek izvršne
npr.: (`printf`, `assert`, "assertions" kao `x++`, `y=x-3`)
- izvršne kada su istiniti uvjeti ("*guard*")
npr.: (`x == 2`), (`N < 4`)
- izvršne kada kanal nije pun (predaja `send`)
- izvršne kada kanal nije prazan (prijem `receive`)

Prijelazi u *FSM* i *Promela* naredbe

Svaki *Promela* proces predstavlja *FSM*.

Prijelaze (*T* prema *FSM* definiciji) možemo definirati:

- uvijek izvršne
npr.: (`printf`, `assert`, "assertions" kao `x++`, `y=x-3`)
- izvršne kada su istiniti uvjeti ("*guard*")
npr.: (`x == 2`), (`N < 4`)
- izvršne kada kanal nije pun (predaja `send`)
- izvršne kada kanal nije prazan (prijem `receive`)

Prijelazi u *FSM* i *Promela* naredbe

Svaki *Promela* proces predstavlja *FSM*.

Prijelaze (*T* prema *FSM* definiciji) možemo definirati:

- uvijek izvršne
npr.: (`printf`, `assert`, "assertions" kao `x++`, `y=x-3`)
- izvršne kada su istiniti uvjeti ("*guard*")
npr.: (`x == 2`), (`N < 4`)
- izvršne kada kanal nije pun (predaja `send`)
- izvršne kada kanal nije prazan (prijem `receive`)

Prijelazi u *FSM* i *Promela* naredbe

Svaki *Promela* proces predstavlja *FSM*.

Prijelaze (*T* prema *FSM* definiciji) možemo definirati:

- uvijek izvršne
npr.: (`printf`, `assert`, "assertions" kao `x++`, `y=x-3`)
- izvršne kada su istiniti uvjeti ("*guard*")
npr.: (`x == 2`), (`N < 4`)
- izvršne kada kanal nije pun (predaja `send`)
- izvršne kada kanal nije prazan (prijem `receive`)

Sinkronost vs. asinkronost

- ⇒ U svojoj suštini **svi** procesi su *asinkroni*
- ⇒ *Sinkronost* uvodimo zbog potrebe modeliranja: često je potrebno analizirati samo bitno, zato se *asinkrone* pojave apstrahiraju
- ⇒ kod analize zahtjeva (eng. requirements analysis) često ne promatramo asinkrone popratne pojave
- ⇒ *Spin* preko `chan Chan = [0] of msg1, msg2 ...` podržava *sinkroni* način rada

Sinkronost vs. asinkronost

- ⇒ U svojoj suštini **svi** procesi su *asinkroni*
- ⇒ *Sinkronost* uvodimo zbog potrebe modeliranja: često je potrebno analizirati samo bitno, zato se *asinkrone* pojave apstrahiraju
- ⇒ kod analize zahtjeva (eng. requirements analysis) često ne promatramo asinkrone popratne pojave
- ⇒ *Spin* preko `chan Chan = [0] of msg1, msg2 ...` podržava *sinkroni* način rada

Sinkronost vs. asinkronost

- ⇒ U svojoj suštini **svi** procesi su *asinkroni*
- ⇒ *Sinkronost* uvodimo zbog potrebe modeliranja: često je potrebno analizirati samo bitno, zato se *asinkrone* pojave apstrahiraju
- ⇒ kod analize zahtjeva (eng. requirements analysis) često ne promatramo asinkrone popratne pojave
- ⇒ *Spin* preko `chan Chan = [0] of msg1, msg2 ...` podržava *sinkroni* način rada

Sinkronost vs. asinkronost

- ⇒ U svojoj suštini **svi** procesi su *asinkroni*
- ⇒ *Sinkronost* uvodimo zbog potrebe modeliranja: često je potrebno analizirati samo bitno, zato se *asinkrone* pojave apstrahiraju
- ⇒ kod analize zahtjeva (eng. requirements analysis) često ne promatramo asinkrone popratne pojave
- ⇒ *Spin* preko `chan Chan = [0] of msg1, msg2 ...` podržava *sinkroni* način rada

Struktura procesa

Strukturu *Promela* procesa definiramo preko strukture *FSM* sa slijedecim konstruktima:

- **";"** , **"goto"** i *labele*
- nedeteriministička selekcija (*Promela if*)
- nedeteriministička iteracija (*Promela do* petlja)
- *promela* "unless": { } unless { }
- atomske (nedjeljive) sekvence (atomic { } id_step { })

Struktura procesa

Strukturu *Promela* procesa definiramo preko strukture *FSM* sa slijedecim konstruktima:

- `";"` , `"goto"` i *labele*
- nedeteriministička selekcija (*Promela* `if`)
- nedeteriministička iteracija (*Promela* `do` petlja)
- *promela* `"unless": { } unless { }`
- atomske (nedjeljive) sekvence (`atomic { } id_step { }`)

Struktura procesa

Strukturu *Promela* procesa definiramo preko strukture *FSM* sa slijedecim konstruktima:

- `;"` , `"goto"` i *labele*
- nedeteriministička selekcija (*Promela* `if`)
- nedeteriministička iteracija (*Promela* `do` petlja)
- *promela* `"unless": { } unless { }`
- atomske (nedjeljive) sekvence (`atomic { } id_step { }`)

Struktura procesa

Strukturu *Promela* procesa definiramo preko strukture *FSM* sa slijedecim konstruktima:

- `";"` , `"goto"` i *labele*
- nedeterministička selekcija (*Promela* `if`)
- nedeterministička iteracija (*Promela* `do` petlja)
- *promela* `"unless": { } unless { }`
- atomske (nedjeljive) sekvence (`atomic { } id_step { }`)

Struktura procesa

Strukturu *Promela* procesa definiramo preko strukture *FSM* sa slijedecim konstruktima:

- `;` , `goto` i *labele*
- nedeteriministička selekcija (*Promela* `if`)
- nedeteriministička iteracija (*Promela* `do` petlja)
- *promela* `"unless": { } unless { }`
- atomske (nedjeljive) sekvence (`atomic { } id_step { }`)

nedeterministička selekcija (*if*)

if

*guard*₁ \longrightarrow *stmtnt*_{1,1}; *stmtnt*_{1,2}; *stmtnt*_{1,3};

*guard*₁ \longrightarrow *stmtnt*_{2,1}; *stmtnt*_{2,2}; *stmtnt*_{2,3};

...

*guard*₁ \longrightarrow *stmtnt*_{*n*,1}; *stmtnt*_{*n*,2}; *stmtnt*_{*n*,3};

fi

\rightarrow ako je barem jedan "guard" izvršan, *if* je izvršan

\rightarrow ako je više od jedan "guard" izvršan, izvodi se "guard" po slučajnom odabiru

\rightarrow ako niti jedan "guard" nije izvršan, *if* *blokira*

Za vježbu:

Precrtajte dio pripadnog automata (*FSM*) za *Promela if* naredbu !

nedeterministička selekcija (*if*)

if

*guard*₁ \longrightarrow *stmtnt*_{1,1}; *stmtnt*_{1,2}; *stmtnt*_{1,3};

*guard*₁ \longrightarrow *stmtnt*_{2,1}; *stmtnt*_{2,2}; *stmtnt*_{2,3};

...

*guard*₁ \longrightarrow *stmtnt*_{*n*,1}; *stmtnt*_{*n*,2}; *stmtnt*_{*n*,3};

fi

- ako je barem jedan "guard" izvršan, *if* je izvršan
- ako je više od jedan "guard" izvršan, izvodi se "guard" po slučajnom odabiru
- ako niti jedan "guard" nije izvršan, *if* *blokira*

Za vježbu:

Precrtajte dio pripadnog automata (*FSM*) za *Promela if* naredbu !

nedeterministička selekcija (*if*)

if

*guard*₁ \longrightarrow *stmtnt*_{1,1}; *stmtnt*_{1,2}; *stmtnt*_{1,3};

*guard*₁ \longrightarrow *stmtnt*_{2,1}; *stmtnt*_{2,2}; *stmtnt*_{2,3};

...

*guard*₁ \longrightarrow *stmtnt*_{*n*,1}; *stmtnt*_{*n*,2}; *stmtnt*_{*n*,3};

fi

- ako je barem jedan "guard" izvršan, *if* je izvršan
- ako je više od jedan "guard" izvršan, izvodi se "guard" po slučajnom odabiru
- ako niti jedan "guard" nije izvršan, *if* *blokira*

Za vježbu:

Precrtajte dio pripadnog automata (*FSM*) za *Promela* *if* naredbu !

nedeteriministička iteracija (do)

do

$guard_1 \longrightarrow stmt_{1,1}; stmt_{1,2}; stmt_{1,3};$

$guard_1 \longrightarrow stmt_{2,1}; stmt_{2,2}; stmt_{2,3};$

...

$guard_1 \longrightarrow stmt_{n,1}; stmt_{n,2}; stmt_{n,3};$

od

→ do u *Promeli* je if u beskonačnoj petlji ...

→ ... iz koje se izlazi sa break ili goto naredbom

Za vježbu:

Precrtajte dio pripadnog automata (FSM) za *Promela* do naredbu !

Koja je semantika *Pomela* goto naredbe (iz većine jezika izbačene) ?

nedeterministička iteracija (do)

do

$guard_1 \longrightarrow stmt_{1,1}; stmt_{1,2}; stmt_{1,3};$

$guard_1 \longrightarrow stmt_{2,1}; stmt_{2,2}; stmt_{2,3};$

...

$guard_1 \longrightarrow stmt_{n,1}; stmt_{n,2}; stmt_{n,3};$

od

→ do u *Promeli* je if u beskonačnoj petlji ...

→ ... iz koje se izlazi sa break ili goto naredbom

Za vježbu:

Precrtajte dio pripadnog automata (FSM) za *Promela* do naredbu !

Koja je semantika *Pomela* goto naredbe (iz većine jezika izbačene) ?

nedeterministička iteracija (do)

do

$guard_1 \longrightarrow stmt_{1,1}; stmt_{1,2}; stmt_{1,3};$

$guard_1 \longrightarrow stmt_{2,1}; stmt_{2,2}; stmt_{2,3};$

...

$guard_1 \longrightarrow stmt_{n,1}; stmt_{n,2}; stmt_{n,3};$

od

→ do u *Promeli* je *if* u beskonačnoj petlji ...

→ ... iz koje se izlazi sa *break* ili *goto* naredbom

Za vježbu:

Precrtajte dio pripadnog automata (FSM) za *Promela* *do* naredbu !

Koja je semantika *Pomela* *goto* naredbe (iz većine jezika izbačene) ?

Atomske (nedjeljive sekvence)

```
atomic { }
```

`atomic { }` sekvencu ili blok naredbi *Spin* u simulaciji/verifikaciji promatra kao da su nedjeljive

```
d_step { }
```

`d_step { }` je rigorozniji oblik `atomic { }` direktive.

unutar `d_step { }` nisu dozvoljeni `goto`, nedeterminizam i naredbe koje mogu "blokirati".

Za vježbu:

Koja je glavna namjena `atomic { }` i `d_step { }` direktiva ?
Kako utječe na memorijske i vemenske resuse *Spin* alata ?

Atomske (nedjeljive sekvence)

```
atomic { }
```

`atomic { }` sekvencu ili blok naredbi *Spin* u simulaciji/verifikaciji promatra kao da su nedjeljive

```
d_step { }
```

`d_step { }` je rigorozniji oblik `atomic { }` direktive.

unutar `d_step { }` nisu dozvoljeni `goto`, nedeterminizam i naredbe koje mogu "blokirati".

Za vježbu:

Koja je glavna namjena `atomic { }` i `d_step { }` direktiva ?
Kako utječe na memorijske i vemenske resuse *Spin* alata ?

Atomske (nedjeljive sekvence)

```
atomic { }
```

`atomic { }` sekvencu ili blok naredbi *Spin* u simulaciji/verifikaciji promatra kao da su nedjeljive

```
d_step { }
```

`d_step { }` je rigorozniji oblik `atomic { }` direktive.

unutar `d_step { }` nisu dozvoljeni `goto`, nedeterminizam i naredbe koje mogu "blokirati".

Za vježbu:

Koja je glavna namjena `atomic { }` i `d_step { }` direktiva ?
Kako utječe na memorijske i vemenske resuse *Spin* alata ?

Primjer: asinkroni produkt

```
#define N 4
#define p (x < N)

int x = N;

active proctype A1()
{
do
:: x%2 -> x = 3*x+1
od
}
```

Primjer: asinkroni produkt

```
active proctype A2()  
{  
do  
  :: !(x%2) -> x = x/2  
od  
}
```

*LT*L ili never-claim u *Promeli*:

```
never {      /* <>[]p */
T0_init:
    if
    :: p -> goto accept_S4
    :: true -> goto T0_init
    fi;
accept_S4:
    if
    :: p -> goto accept_S4
    fi;
}
```


Analiza A_1 i A_2

Za vježbu:

- a) što je predikat p u *LTL* formuli
- b) da li je potreban `init { }` dio programa
- c) generirati `never { }` "claim" sa spin -f <>[]p
- d) nacrtati Büchi automat B
- e) nacrtati A_1 i A_2 direktno iz koda u *Promeli*
- f) nacrtati A_1 i A_2 preko `pan -d` naredbe
- g) usporediti i provjeriti dobivene grafove automata A_1 i A_2
- h) provesti analizu: preko simulacije i verifikacije

Analiza A_1 i A_2

Za vježbu:

- a) što je predikat p u *LTL* formuli
- b) da li je potreban `init { }` dio programa
- c) generirati `never { }` "claim" sa spin -f <>[]p
- d) nacrtati Büchi automat B
- e) nacrtati A_1 i A_2 direktno iz koda u *Promeli*
- f) nacrtati A_1 i A_2 preko `pan -d` naredbe
- g) usporediti i provjeriti dobivene grafove automata A_1 i A_2
- h) provesti analizu: preko simulacije i verifikacije

Analiza A_1 i A_2

Za vježbu:

- a) što je predikat p u *LTL* formuli
- b) da li je potreban `init { }` dio programa
- c) generirati `never { }` "claim" sa spin -f <>[]p
- d) nacrtati Büchi automat B
- e) nacrtati A_1 i A_2 direktno iz koda u *Promeli*
- f) nacrtati A_1 i A_2 preko `pan -d` naredbe
- g) usporediti i provjeriti dobivene grafove automata A_1 i A_2
- h) provesti analizu: preko simulacije i verifikacije

Analiza A_1 i A_2

Za vježbu:

- a) što je predikat p u *LTL* formuli
- b) da li je potreban `init { }` dio programa
- c) generirati `never { }` "claim" sa spin -f <>[]p
- d) nacrtati Büchi automat B
- e) nacrtati A_1 i A_2 direktno iz koda u *Promeli*
- f) nacrtati A_1 i A_2 preko `pan -d` naredbe
- g) usporediti i provjeriti dobivene grafove automata A_1 i A_2
- h) provesti analizu: preko simulacije i verifikacije

Analiza A_1 i A_2

Za vježbu:

- a) što je predikat p u *LTL* formuli
- b) da li je potreban `init { }` dio programa
- c) generirati `never { }` "claim" sa spin -f <>[]p
- d) nacrtati Büchi automat B
- e) nacrtati A_1 i A_2 direktno iz koda u *Promeli*
- f) nacrtati A_1 i A_2 preko `pan -d` naredbe
- g) usporediti i provjeriti dobivene grafove automata A_1 i A_2
- h) provesti analizu: preko simulacije i verifikacije

Za one koji hoće više:

Službeni SPIN tutori

- najbolji način za pretvaranje *Promele*/*SPIN* u još moćan i pouzdan programski alat za svakodnevnu upotrebu je samostalno modeliranje
- poželjno je proučiti što više (rješених) primjera
- pri tome se može pomoći *tutorima* i člancima dostupnim na [www](http://www.fer.hr) stranicama

Zaključak:

- (1) modeli sa **konačnim** brojem stanja (ali sa ω -prihvatljivosti)
- (2) **asinkronost**: nema unaprijed definiranog mehanizma za sinkronizaciju kao ni sistemskog "sata" (clock)
- (3) **nedeterministička** upravljačka struktura: prijelazi u *FSM* su nedeterministički
- (4) **izvršivost** preko blokirajućih naredbi ("*guards*")
- (5) mogućnost dodavanja koda u jeziku *C* i vlastite strukture podataka
- (6) proširenje osnovne namjene: osim analize konkurentnih reaktivnih programa *Spin* se primjenjuje i u testiranju, planiranju, ... kao sastavni dio raznih programskih alata ...

Zaključak:

- (1) modeli sa **konačnim** brojem stanja (ali sa ω -prihvatljivosti)
- (2) **asinkronost**: nema unaprijed definiranog mehanizma za sinkronizaciju kao ni sistemskog "sata" (clock)
- (3) **nedeterministička** upravljačka struktura: prijelazi u *FSM* su nedeterministički
- (4) **izvršivost** preko blokirajućih naredbi ("*guards*")
- (5) mogućnost dodavanja koda u jeziku *C* i vlastite strukture podataka
- (6) proširenje osnovne namjene: osim analize konkurentnih reaktivnih programa *Spin* se primjenjuje i u testiranju, planiranju, ... kao sastavni dio raznih programskih alata ...

Zaključak:

- (1) modeli sa **konačnim** brojem stanja (ali sa ω -prihvatljivosti)
- (2) **asinkronost**: nema unaprijed definiranog mehanizma za sinkronizaciju kao ni sistemskog "sata" (clock)
- (3) **nedeterministička** upravljačka struktura: prijelazi u *FSM* su nedeterministički
- (4) **izvršivost** preko blokirajućih naredbi ("*guards*")
- (5) mogućnost **dodavanja** koda u jeziku *C* i **vlastite strukture** podataka
- (6) **proširenje osnovne namjene**: osim analize konkurentnih reaktivnih programa *Spin* se primjenjuje i u testiranju, planiranju, ... kao sastavni dio raznih programskih alata ...

Zaključak:

- (1) modeli sa **konačnim** brojem stanja (ali sa ω -prihvatljivosti)
- (2) **asinkronost**: nema unaprijed definiranog mehanizma za sinkronizaciju kao ni sistemskog "sata" (clock)
- (3) **nedeterministička** upravljačka struktura: prijelazi u *FSM* su nedeterministički
- (4) **izvršivost** preko blokirajućih naredbi ("*guards*")
- (5) mogućnost **dodavanja** koda u jeziku *C* i **vlastite strukture** podataka
- (6) **proširenje osnovne namjene**: osim analize konkurentnih reaktivnih programa *Spin* se primjenjuje i u testiranju, planiranju, ... kao sastavni dio raznih programskih alata ...

Zaključak:

- (1) modeli sa **konačnim** brojem stanja (ali sa ω -prihvatljivosti)
- (2) **asinkronost**: nema unaprijed definiranog mehanizma za sinkronizaciju kao ni sistemskog "sata" (clock)
- (3) **nedeterministička** upravljačka struktura: prijelazi u *FSM* su nedeterministički
- (4) **izvršivost** preko blokirajućih naredbi ("*guards*")
- (5) mogućnost **dodavanja** koda u jeziku *C* i **vlastite strukture** podataka
- (6) **proširenje osnovne namjene**: osim analize konkurentnih reaktivnih programa *Spin* se primjenjuje i u testiranju, planiranju, ... kao sastavni dio raznih programskih alata ...

Zaključak:

- (1) modeli sa **konačnim** brojem stanja (ali sa ω -prihvatljivosti)
- (2) **asinkronost**: nema unaprijed definiranog mehanizma za sinkronizaciju kao ni sistemskog "sata" (clock)
- (3) **nedeterministička** upravljačka struktura: prijelazi u *FSM* su nedeterministički
- (4) **izvršivost** preko blokirajućih naredbi ("*guards*")
- (5) mogućnost **dodavanja** koda u jeziku *C* i **vlastite strukture** podataka
- (6) **proširenje osnovne namjene**: osim analize konkurentnih reaktivnih programa *Spin* se primjenjuje i u testiranju, planiranju, ... kao sastavni dio raznih programskih alata ...

Šira literatura:

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