Formalne Metode u oblikovanju sustava

FER

drugi ciklus predavanja
ver. 0.1.7
nadn.zadnje.rev.: *17. travnja 2009.*

Ponavljanje

1. ciklus razvoja programa i formalna metode
2. verifikacija, provjera modela
3. komunicirajući automati (CFSM)

$I \models S$ ili $M \models \varphi$

a) provjera modela za programe

b) provjera modela za sklopovlje

Ponavljanje

1. ciklus razvoja programa i formalna metode
2. verifikacija, provjera modela
3. komunicirajući automati (CFSM)

$I \models S$ ili $M \models \varphi$

1. provjera modela za programe
2. provjera modela za sklopovlje

Ponavljanje

1. ciklus razvoja programa i formalna metode
2. verifikacija, provjera modela
3. komunicirajući automati (CFSM)

$I \models S$ ili $M \models \varphi$

1. provjera modela za programe
2. provjera modela za sklopovlje

Ponavljanje

1. ciklus razvoja programa i formalna metode
2. verifikacija, provjera modela
3. komunicirajući automati (CFSM)

$I \models S$ ili $M \models \varphi$

1. provjera modela za programe
2. provjera modela za sklopovlje

Ponavljanje

1. ciklus razvoja programa i formalna metode
2. verifikacija, provjera modela
3. komunicirajući automati (CFSM)

$I \models S$ ili $M \models \varphi$

1. provjera modela za programe
2. provjera modela za sklopovlje

Uvod

1. formalna verifikacija komunikacijskih protokola
2. ciljani sustav: komunikacijski protokoli u raspodijeljenim sustavima (*eng. distributed systems*)
3. modeliranje implementacije: Promela (**Pro**cces **me**ta **la**nguage)
4. sustav za verifikaciju: SPIN (**S**imple **P**rotocol **IN**terpreter)

U nastavku:

Cilj je analiza: modeliranje i verifikacija ponašanja **raspodjeljenih sustava** (**konkurentnih reaktivnih sustava**) pomoću `provjere modela` (*eng. model checking*)

Što je konkurentni reaktivni sustav? U kakvoj je vezi s protokolima i distribuiranim sustavima?

Uvod

1. formalna verifikacija komunikacijskih protokola
2. ciljani sustav: komunikacijski protokoli u raspodijeljenim sustavima (*eng. distributed systems*)
3. modeliranje implementacije: Promela (**Pro**cces **me**ta **la**nguage)
4. sustav za verifikaciju: SPIN (**S**imple **P**rotocol **IN**terpreter)

U nastavku:

Cilj je analiza: modeliranje i verifikacija ponašanja **raspodjeljenih sustava** (**konkurentnih reaktivnih sustava**) pomoću `provjere modela` (*eng. model checking*)
Što je konkurentni reaktivni sustav ? U kakvoj je vezi s protokolima i distribuiranim sustavima? . . .

Uvod

1. formalna verifikacija komunikacijskih protokola
2. ciljani sustav: komunikacijski protokoli u raspodijeljenim sustavima (*eng. distributed systems*)
3. modeliranje implementacije: Promela (**Pro**cces **me**ta **la**nguage)
4. sustav za verifikaciju: SPIN (**S**imple **P**rotocol **IN**terpreter)

U nastavku:

Cilj je analiza: modeliranje i verifikacija ponašanja **raspodjeljenih sustava** (**konkurentnih reaktivnih sustava**) pomoću `provjere modela` (*eng. model checking*)
Što je konkurentni reaktivni sustav ? U kakvoj je vezi s protokolima i distribuiranim sustavima? . . .

Uvod

1. formalna verifikacija komunikacijskih protokola
2. ciljani sustav: komunikacijski protokoli u raspodijeljenim sustavima (*eng. distributed systems*)
3. modeliranje implementacije: Promela (**Pro**cces **me**ta **la**nguage)
4. sustav za verifikaciju: SPIN (**S**imple **P**rotocol **IN**terpreter)

U nastavku:

Cilj je analiza: modeliranje i verifikacija ponašanja **raspodjeljenih sustava** (**konkurentnih reaktivnih sustava**) pomoću `provjere modela` (*eng. model checking*)
Što je konkurentni reaktivni sustav ? U kakvoj je vezi s protokolima i distribuiranim sustavima? . . .

Općenito o konkurentnim i reaktivnim sustavima

Sustav promatramo kao skup komunicirajućih procesa.
Procesi komuniciraju pomoću:

1. izmjene poruka (*eng. message pasing*)
2. preko repova–dijeljenje resursa (eng. *resource sharing*)

Primjer za vježbu:

- na ∗WIN platformi pokrenuti TASK MANAGER ili
  na ∗nux iz konzole npr. naredbu *ps -ax*
- Skicirajte međusobno povezane procese! Uočite procese karakteristične za operativni sustav odnosno pojedinu aplikaciju!

Općenito o konkurentnim i reaktivnim sustavima

Sustav promatramo kao skup komunicirajućih procesa.
Procesi komuniciraju pomoću:

1. izmjene poruka (*eng. message pasing*)
2. preko repova–dijeljenje resursa (eng. *resource sharing*)

Primjer za vježbu:

- na ∗WIN platformi pokrenuti TASK MANAGER ili
  na ∗nux iz konzole npr. naredbu *ps -ax*
- Skicirajte međusobno povezane procese! Uočite procese karakteristične za operativni sustav odnosno pojedinu aplikaciju!

Općenito o konkurentnim i reaktivnim sustavima

Sustav promatramo kao skup komunicirajućih procesa.
Procesi komuniciraju pomoću:

1. izmjene poruka (*eng. message pasing*)
2. preko repova–dijeljenje resursa (eng. *resource sharing*)

Primjer za vježbu:

- na ∗WIN platformi pokrenuti TASK MANAGER ili
  na ∗nux iz konzole npr. naredbu *ps -ax*
- Skicirajte međusobno povezane procese! Uočite procese karakteristične za operativni sustav odnosno pojedinu aplikaciju!

Općenito o konkurentnim i reaktivnim sustavima

Sustav promatramo kao skup komunicirajućih procesa.
Procesi komuniciraju pomoću:

1. izmjene poruka (*eng. message pasing*)
2. preko repova–dijeljenje resursa (eng. *resource sharing*)

Primjer za vježbu:

- na ∗WIN platformi pokrenuti TASK MANAGER ili
  na ∗nux iz konzole npr. naredbu *ps -ax*
- Skicirajte međusobno povezane procese! Uočite procese karakteristične za operativni sustav odnosno pojedinu aplikaciju!

Distribuirani sustav

Distribuirani sustav procesi su raspoređeni u prostoru.
(od više procesa na jednom procesoru preko *multi–core procesora* do mreže procesora)

Konkurentni sustav

Procesi se izvode sa različitim varijantama paralelnosti (istovremenosti)

Reaktivni sustav

Reaktivni sustav mora odmah reagirati na signale, poruke . . . okoline

Distribuirani sustav

Distribuirani sustav procesi su raspoređeni u prostoru.
(od više procesa na jednom procesoru preko *multi–core procesora* do mreže procesora)

Konkurentni sustav

Procesi se izvode sa različitim varijantama paralelnosti (istovremenosti)

Reaktivni sustav

Reaktivni sustav mora odmah reagirati na signale, poruke . . . okoline

Distribuirani sustav

Distribuirani sustav procesi su raspoređeni u prostoru.
(od više procesa na jednom procesoru preko *multi–core procesora* do mreže procesora)

Konkurentni sustav

Procesi se izvode sa različitim varijantama paralelnosti (istovremenosti)

Reaktivni sustav

Reaktivni sustav mora odmah reagirati na signale, poruke . . . okoline

1. komunukacijski protokol u *raspodjeljenom, konkuretnom reaktivnom sustavu* je podrška za izmjenu informacija/podataka među procesima.
2. kritičnost: protokol mora biti kvalitetan i robustan: analiza, modeliranje i verifikacija
3. kod distribuiranih aplikacija (web, klijent–server, dretve . . .) nestaje precizne granice između protokola i korisničke aplikacije

Kako naučiti verificirati protokol/raspodjeljenu aplikaciju ?

. . . ili kako primjeniti *provjeru modela* na zadani problem ?

1. Modeliranje: definirati model u `Promela` jeziku
2. Analiza–*provjera modela*: pozvati `SPIN` sa datotekom–modelom opisanim jezikom `Promela` i provjeririti valjanost uvjeta formulama *LTL* logike
3. **naučiti** teorijsku podlogu radi efikasnog i optimalnog korištenja `SPIN` programa

Kako naučiti verificirati protokol/raspodjeljenu aplikaciju ?

. . . ili kako primjeniti *provjeru modela* na zadani problem ?

1. Modeliranje: definirati model u `Promela` jeziku
2. Analiza–*provjera modela*: pozvati `SPIN` sa datotekom–modelom opisanim jezikom `Promela` i provjeririti valjanost uvjeta formulama *LTL* logike
3. **naučiti** teorijsku podlogu radi efikasnog i optimalnog korištenja `SPIN` programa

Za one koji hoće više . . .

Komunikacijski protokol $\longrightarrow$ aplikacija

Poželjno je svaku aplikaciju koju oblikujete–razvijate verificirati (*model-checking*).
Trenutačni trendovi u razvoju programa uključuju *skrivene formalne metode* u kojima "model-checking" postaje dio modela razvoja programa . . .

A sada formalno: teorijska podloga

Teorijska podloga sadržana je u slijedećoj formuli

$$M \models \varphi$$

sa slijedećim značenjem

(da li Vam je nešto slično poznato od ranije ?)

- M je model odnosno `Promela` program
- φ su LTL formule (**LTL** - **L**inear **T**emporal **L**ogic) kojima provjeravamo uvjete

Sve zajedno . . .

- `Promela` program (model) se sastoji od mreže komunicirajućih automata. Sintaksa je slična jeziku `C`, Dijkstrine "guarded" komande i `CSP` algebra čine teorijske temelje (detalji kasnije)
- `Spin` pronalazi sve moguće interakcije (Kripké struktura) kao sinkroni ili asinhroni produkt automata, efikasno ih kodira (*bit–state–hashing*)
- sastavni dio `Spina` je i konverzija *LTL* logičke formule u Büchi automat

Sve zajedno . . .

- `Promela` program (model) se sastoji od mreže komunicirajućih automata. Sintaksa je slična jeziku `C`, Dijkstrine "guarded" komande i `CSP` algebra čine teorijske temelje (detalji kasnije)
- `Spin` pronalazi sve moguće interakcije (Kripké struktura) kao sinkroni ili asinhroni produkt automata, efikasno ih kodira (*bit–state–hashing*)
- sastavni dio `Spina` je i konverzija *LTL* logičke formule u Büchi automat

Sve zajedno . . .

- `Promela` program (model) se sastoji od mreže komunicirajućih automata. Sintaksa je slična jeziku `C`, Dijkstrine "guarded" komande i `CSP` algebra čine teorijske temelje (detalji kasnije)
- `Spin` pronalazi sve moguće interakcije (Kripké struktura) kao sinkroni ili asinhroni produkt automata, efikasno ih kodira (*bit–state–hashing*)
- sastavni dio `Spina` je i konverzija *LTL* logičke formule u Büchi automat

U nastavku:

1. `SPIN/Promela` sustav sa uvodnim primjerom "Hello world" (tzv. snake preview ...)
2. konačni automat (FSM), sinkroni ili asinkroni produkt komunicirajućih automata (CFSM), *LTL* logika, pretvorba *LTL* u Büchi automat, Dijkstra "guarded" komande ili što sve *SPIN* uključuje ...

`SPIN` instalacija

1. Instalacija se svodi na kopiranje već pripremljenih izvršnih verzija sa `http://spinroot.com/spin/Man/README.html`.
2. Potreban je i *C* prevodioc (preporuka: *gcc*)
3. Za one koje hoće više:
   pogledati *SPIN newsletter* i *SPIN* simpozije

U nastavku:

1. `SPIN/Promela` sustav sa uvodnim primjerom "Hello world" (tzv. snake preview . . .)
2. konačni automat (FSM), sinkroni ili asinkroni produkt komunicirajućih automata (CFSM), *LTL* logika, pretvorba *LTL* u Büchi automat, Dijkstra "guarded" komande ili što sve *SPIN* uključuje . . .

`SPIN` instalacija

1. Instalacija se svodi na kopiranje već pripremljenih izvršnih verzija sa `http://spinroot.com/spin/Man/README.html`.
2. Potreban je i *C* prevodioc (preporuka: *gcc*)
3. Za one koje hoće više:
   pogledati *SPIN newsletter* i *SPIN* simpozije

U nastavku:

1. `SPIN/Promela` sustav sa uvodnim primjerom "Hello world" (tzv. snake preview . . .)
2. konačni automat (FSM), sinkroni ili asinkroni produkt komunicirajućih automata (CFSM), *LTL* logika, pretvorba *LTL* u Büchi automat, Dijkstra "guarded" komande ili što sve *SPIN* uključuje . . .

`SPIN` instalacija

1. Instalacija se svodi na kopiranje već pripremljenih izvršnih verzija sa `http://spinroot.com/spin/Man/README.html`.
2. Potreban je i *C* prevodioc (preporuka: *gcc*)
3. Za one koje hoće više:
   pogledati *SPIN newsletter* i *SPIN* simpozije

U nastavku:

1. `SPIN/Promela` sustav sa uvodnim primjerom "Hello world" (tzv. snake preview . . .)
2. konačni automat (FSM), sinkroni ili asinkroni produkt komunicirajućih automata (CFSM), *LTL* logika, pretvorba *LTL* u Büchi automat, Dijkstra "guarded" komande ili što sve *SPIN* uključuje . . .

`SPIN` instalacija

1. Instalacija se svodi na kopiranje već pripremljenih izvršnih verzija sa `http://spinroot.com/spin/Man/README.html`.
2. Potreban je i *C* prevodioc (preporuka: *gcc*)
3. Za one koje hoće više:
   pogledati *SPIN newsletter* i *SPIN* simpozije

U nastavku:

1. `SPIN/Promela` sustav sa uvodnim primjerom "Hello world" (tzv. snake preview . . .)
2. konačni automat (FSM), sinkroni ili asinkroni produkt komunicirajućih automata (CFSM), *LTL* logika, pretvorba *LTL* u Büchi automat, Dijkstra "guarded" komande ili što sve *SPIN* uključuje . . .

`SPIN` instalacija

1. Instalacija se svodi na kopiranje već pripremljenih izvršnih verzija sa `http://spinroot.com/spin/Man/README.html`.
2. Potreban je i *C* prevodioc (preporuka: *gcc*)
3. Za one koje hoće više:
   pogledati *SPIN newsletter* i *SPIN* simpozije

Primjer: "Hello world" ili kao opisujemo *CFSM*

```
/* A "Hello World" Promela model for SPIN. */
active proctype Hello() {
printf("Hello process, my pid is: %d\n", _pid);
}

init {
int lastpid;
printf("init process, my pid is: %d\n", _pid);
lastpid = run Hello();
printf("last pid was: %d\n", lastpid);
}
```

Primjer: "Hello world" ili kao opisujemo *CFSM*

```
/* A "Hello World" Promela model for SPIN. */
active proctype Hello() {
printf("Hello process, my pid is: %d\n", _pid);
}

init {
int lastpid;
printf("init process, my pid is: %d\n", _pid);
lastpid = run Hello();
printf("last pid was: %d\n", lastpid);
}
```

Napomena:

Promela ima uvijek barem jedan `init{}` proces

svaki *Promela* `proces` je jedan automat (FSM) iz CFSM (mreže komunicirajućih automata)

_pid "broj" procesa

predefinirane ili "unutarnje" varijable počinju s "_"

`run` pokreće druge procese

Promela `proces` $\neq$ ranije spomenutim procesima

slično sintaksi jezika *CC* ali oprez, **semantika** je drukčija

Napomena:

Promela ima uvijek barem jedan `init{}` proces
svaki *Promela* `proces` je jedan automat (FSM) iz CFSM (mreže komunicirajućih automata)
_pid "broj" procesa
predefinirane ili "unutarnje" varijable počinju s "_"
`run` pokreće druge procese
Promela `proces` $\neq$ ranije spomenutim procesima
slično sintaksi jezika CC ali oprez, **semantika** je drukčija

Napomena:

Promela ima uvijek barem jedan `init{}` proces
svaki *Promela* `proces` je jedan automat (FSM) iz CFSM (mreže komunicirajućih automata)
_pid "broj" procesa
predefinirane ili "unutarnje" varijable počinju s "_"
`run` pokreće druge procese
Promela `proces` $\neq$ ranije spomenutim procesima
slično sintaksi jezika CC ali oprez, **semantika** je drukčija

Napomena:

Promela ima uvijek barem jedan `init{}` proces
svaki *Promela* `proces` je jedan automat (FSM) iz CFSM (mreže komunicirajućih automata)
_pid "broj" procesa
predefinirane ili "unutarnje" varijable počinju s "_"
`run` pokreće druge procese
Promela `proces` $\neq$ ranije spomenutim procesima
slično sintaksi jezika CC ali oprez, **semantika** je drukčija

Napomena:

Promela ima uvijek barem jedan `init{}` proces
svaki *Promela* `proces` je jedan automat (FSM) iz CFSM (mreže komunicirajućih automata)
_pid "broj" procesa
predefinirane ili "unutarnje" varijable počinju s "_"
`run` pokreće druge procese
Promela `proces` $\neq$ ranije spomenutim procesima
slično sintaksi jezika *C* ali oprez, **semantika** je drukčija

Napomena:

Promela ima uvijek barem jedan `init{}` proces
svaki *Promela* `proces` je jedan automat (FSM) iz CFSM (mreže komunicirajućih automata)
_pid "broj" procesa
predefinirane ili "unutarnje" varijable počinju s "_"
`run` pokreće druge procese
Promela `proces` $\neq$ ranije spomenutim procesima
slično sintaksi jezika CC ali oprez, **semantika** je drukčija

Napomena:

Promela ima uvijek barem jedan `init{}` proces
svaki *Promela* `proces` je jedan automat (FSM) iz CFSM (mreže komunicirajućih automata)
_pid "broj" procesa
predefinirane ili "unutarnje" varijable počinju s "_"
`run` pokreće druge procese
Promela `proces` $\neq$ ranije spomenutim procesima
slično sintaksi jezika *C*/*C* ali oprez, **semantika** je drukčija

SPIN: prema Kripke strukturi

```
spin -n2 hello.prm
init process, my pid is: 1
 last pid was: 2
 Hello process, my pid is: 0
 Hello process, my pid is: 2
 3 processes created
 running SPIN in
 random simulation mode
 random seed
```

Za vježbu:

Pokušajmo zajedno skicirati automate FSM za "Hello world" !

Što određuje broj `Promela` procesa ?

SPIN: prema Kripke strukturi

```
spin -n2 hello.prm
init process, my pid is: 1
 last pid was: 2
 Hello process, my pid is: 0
 Hello process, my pid is: 2
 3 processes created
 running SPIN in
 random simulation mode
 random seed
```

Za vježbu:

Pokušajmo zajedno skicirati automate FSM za "Hello world" !
Što određuje broj `Promela` procesa ?

`Hello world` je izveden u tzv. simulacijskom modu

slijede preporučeni koraci primjene *Promela/SPIN*...

ili

Kako iz CFSM dobiti Kripke strukturu ?

Kako provesti analizu primjenom *LTL* logike nad *Kripke* strukturom ?

Za one koji hoće više:

Kripke struktura i *dostupnost*

Kolika je *kompleksnost* ?

`Hello world` je izveden u tzv. simulacijskom modu
slijede preporučeni koraci primjene *Promela/SPIN*...
ili

Kako iz CFSM dobiti Kripke strukturu ?
Kako provesti analizu primjenom *LTL* logike nad *Kripke* strukturom ?

Za one koji hoće više:
Kripke struktura i *dostupnost*
Kolika je *kompleksnost* ?

`Hello world` je izveden u tzv. simulacijskom modu
slijede preporučeni koraci primjene *Promela/SPIN*...
ili

Kako iz CFSM dobiti Kripke strukturu ?
Kako provesti analizu primjenom *LTL* logike nad *Kripke* strukturom ?

Za one koji hoće više:
Kripke struktura i *dostupnost*
Kolika je *kompleksnost* ?

`Hello world` je izveden u tzv. simulacijskom modu
slijede preporučeni koraci primjene *Promela/SPIN*...
ili

Kako iz CFSM dobiti Kripke strukturu ?
Kako provesti analizu primjenom *LTL* logike nad *Kripke* strukturom ?

Za one koji hoće više:
Kripke struktura i *dostupnost*
Kolika je *kompleksnost* ?

`Hello world` je izveden u tzv. simulacijskom modu
slijede preporučeni koraci primjene *Promela/SPIN*...
ili

Kako iz CFSM dobiti Kripke strukturu ?
Kako provesti analizu primjenom *LTL* logike nad *Kripke* strukturom ?

Za one koji hoće više:
Kripke struktura i *dostupnost*
Kolika je *kompleksnost* ?

Roadmap ili postupak verifikacije

(1) definirati prototip ili model za verifikaciju
(*Promela program* $\equiv$ *CFSM*)

(2) *prekontrolirati sintaksu modela*: `spin -A`, `spin -c` ili `spin -p`
prevesti model/*Promela* program: `spin -a hello.prm`

(3) početi sa serijom *random* simulacija: `spin hello.prm` ili npr.
`spin -p -u10 hello.prm`

(4) kreirati verifikator: `spin -a hello.prm`
načiniti izvršnu verziju gcc ili cc -o hello pan.c
izvesti *hello* tj. verificirati

(5) po "tragu" (*eng.trail*) do grešaka: `spin -t -p hello.prm`

(6) redefinirati (ako treba) model tj. `hello.prm` i ponoviti sve korake
do željene kvalitete

Roadmap ili postupak verifikacije

(1) definirati prototip ili model za verifikaciju
(*Promela program* $\equiv$ *CFSM*)

(2) *prekontrolirati sintaksu modela*: `spin -A`, `spin -c` ili `spin -p`
prevesti model/*Promela* program: `spin -a hello.prm`

(3) početi sa serijom *random* simulacija: `spin hello.prm` ili npr.
`spin -p -u10 hello.prm`

(4) kreirati verifikator: `spin -a hello.prm`
načiniti izvršnu verziju gcc ili cc -o hello pan.c
izvesti *hello* tj. verificirati

(5) po "tragu" (*eng.trail*) do grešaka: `spin -t -p hello.prm`

(6) redefinirati (ako treba) model tj. `hello.prm` i ponoviti sve korake
do željene kvalitete

Roadmap ili postupak verifikacije

(1) definirati prototip ili model za verifikaciju
(*Promela program* $\equiv$ *CFSM*)

(2) *prekontrolirati sintaksu modela*: `spin -A`, `spin -c` ili `spin -p`
prevesti model/*Promela* program: `spin -a hello.prm`

(3) početi sa serijom *random* simulacija: `spin hello.prm` ili npr.
`spin -p -u10 hello.prm`

(4) kreirati verifikator: `spin -a hello.prm`
načiniti izvršnu verziju gcc ili cc -o hello pan.c
izvesti *hello* tj. verificirati

(5) po "tragu" (*eng.trail*) do grešaka: `spin -t -p hello.prm`

(6) redefinirati (ako treba) model tj. `hello.prm` i ponoviti sve korake
do željene kvalitete

Roadmap ili postupak verifikacije

(1) definirati prototip ili model za verifikaciju
(*Promela program* $\equiv$ *CFSM*)

(2) *prekontrolirati sintaksu modela*: `spin -A`, `spin -c` ili `spin -p`
prevesti model/*Promela* program: `spin -a hello.prm`

(3) početi sa serijom *random* simulacija: `spin hello.prm` ili npr.
`spin -p -u10 hello.prm`

(4) kreirati verifikator: `spin -a hello.prm`
načiniti izvršnu verziju gcc ili cc -o hello pan.c
izvesti *hello* tj. verificirati

(5) po "tragu" (*eng.trail*) do grešaka: `spin -t -p hello.prm`

(6) redefinirati (ako treba) model tj. `hello.prm` i ponoviti sve korake do željene kvalitete

Roadmap ili postupak verifikacije

(1) definirati prototip ili model za verifikaciju
(*Promela program* $\equiv$ *CFSM*)

(2) *prekontrolirati sintaksu modela*: `spin -A`, `spin -c` ili `spin -p`
prevesti model/*Promela* program: `spin -a hello.prm`

(3) početi sa serijom *random* simulacija: `spin hello.prm` ili npr.
`spin -p -u10 hello.prm`

(4) kreirati verifikator: `spin -a hello.prm`
načiniti izvršnu verziju gcc ili cc -o hello pan.c
izvesti *hello* tj. verificirati

(5) po "tragu" (*eng.trail*) do grešaka: `spin -t -p hello.prm`

(6) redefinirati (ako treba) model tj. `hello.prm` i ponoviti sve korake do željene kvalitete

Roadmap ili postupak verifikacije

(1) definirati prototip ili model za verifikaciju
(*Promela program* $\equiv$ *CFSM*)

(2) *prekontrolirati sintaksu modela*: `spin -A`, `spin -c` ili `spin -p`
prevesti model/*Promela* program: `spin -a hello.prm`

(3) početi sa serijom *random* simulacija: `spin hello.prm` ili npr.
`spin -p -u10 hello.prm`

(4) kreirati verifikator: `spin -a hello.prm`
načiniti izvršnu verziju gcc ili cc -o hello pan.c
izvesti *hello* tj. verificirati

(5) po "tragu" (*eng.trail*) do grešaka: `spin -t -p hello.prm`

(6) redefinirati (ako treba) model tj. `hello.prm` i ponoviti sve korake
do željene kvalitete

Teorijska podloga `Promela modela-jezika`

Teorijska podloga `Promela modela-jezika` su:

(i) Dijkstrine "*guarded*" komande

(ii) *CSP* Hoare algebra (***C**ommunicating **S**equential **P**rocesses*) komunikacijska algebra kao posebna vrsta procesnih algebri

Teorijska podloga `Promela modela-jezika`

Teorijska podloga `Promela modela-jezika` su:

(i) Dijkstrine "*guarded*" komande

(ii) *CSP* Hoare algebra (***C****ommunicating* ***S****equential* ***P****rocesses*) komunikacijska algebra kao posebna vrsta procesnih algebri

Dijkstrine "*guarded*" komande

"*guarded*" komande su oblika $G \rightarrow S$, gdje je:

- G je propozicija, koju nazivamo guard
- S je izvršna naredba naredba koju "guard" može blokirati.

Semantika:

Semantika *Promele* i originalna *Dijkstrina* semantika nisu potpuno iste. (Vidjeti primjer za *Promelu*)

$\longrightarrow$ u trenutku kada G postane istinit . . .

$\longrightarrow$ . . . izvodi se naredba S,
ako G nije istinit nastupa "blokada"

$\longrightarrow$ nije greška u `Promela` jeziku kada "guard" privremeno blokira!
(npr. kada modeliramo čekanje prijema signala ili poruke!)

$\longrightarrow$ kod Dijkstre kod neistinite propozicije G *kontekst* odlučuje o daljnjoj akciji (nije od značaja za nas)

Dijkstrine "*guarded*" komande

"*guarded*" komande su oblika $G \rightarrow S$, gdje je:

- G je propozicija, koju nazivamo guard
- S je izvršna naredba naredba koju "guard" može blokirati.

Semantika:

Semantika *Promele* i originalna *Dijkstrina* semantika nisu potpuno iste. (Vidjeti primjer za *Promelu*)

$\longrightarrow$ u trenutku kada G postane istinit . . .

$\longrightarrow$ . . . izvodi se naredba S,
ako G nije istinit nastupa "blokada"

$\longrightarrow$ nije greška u `Promela` jeziku kada "guard" privremeno blokira!
(npr. kada modeliramo čekanje prijema signala ili poruke!)

$\longrightarrow$ kod Dijkstre kod neistinite propozicije G *kontekst* odlučuje o daljnjoj akciji (nije od značaja za nas)

Dijkstrine "*guarded*" komande

"*guarded*" komande su oblika $G \rightarrow S$, gdje je:

- G je propozicija, koju nazivamo guard
- S je izvršna naredba naredba koju "guard" može blokirati.

Semantika:

Semantika *Promele* i originalna *Dijkstrina* semantika nisu potpuno iste. (Vidjeti primjer za *Promelu*)

$\longrightarrow$ u trenutku kada G postane istinit . . .

$\longrightarrow$ . . . izvodi se naredba S,
ako G nije istinit nastupa "blokada"

$\longrightarrow$ nije greška u `Promela` jeziku kada "guard" privremeno blokira!
(npr. kada modeliramo čekanje prijema signala ili poruke!)

$\longrightarrow$ kod Dijkstre kod neistinite propozicije G *kontekst* odlučuje o daljnjoj akciji (nije od značaja za nas)

Dijkstrine "*guarded*" komande

"*guarded*" komande su oblika $G \rightarrow S$, gdje je:

- G je propozicija, koju nazivamo guard
- S je izvršna naredba naredba koju "guard" može blokirati.

Semantika:

Semantika *Promele* i originalna *Dijkstrina* semantika nisu potpuno iste. (Vidjeti primjer za *Promelu*)

$\longrightarrow$ u trenutku kada G postane istinit . . .

$\longrightarrow$ . . . izvodi se naredba S,
ako G nije istinit nastupa "blokada"

$\longrightarrow$ nije greška u `Promela` jeziku kada "guard" privremeno blokira!
(npr. kada modeliramo čekanje prijema signala ili poruke!)

$\longrightarrow$ kod Dijkstre kod neistinite propozicije G *kontekst* odlučuje o daljnjoj akciji (nije od značaja za nas)

Dijkstrine "*guarded*" komande

"*guarded*" komande su oblika $G \rightarrow S$, gdje je:

- G je propozicija, koju nazivamo guard
- S je izvršna naredba naredba koju "guard" može blokirati.

Semantika:

Semantika *Promele* i originalna *Dijkstrina* semantika nisu potpuno iste. (Vidjeti primjer za *Promelu*)

$\longrightarrow$ u trenutku kada G postane istinit . . .

$\longrightarrow$ . . . izvodi se naredba S,
ako G nije istinit nastupa "blokada"

$\longrightarrow$ nije greška u `Promela` jeziku kada "guard" privremeno blokira!
(npr. kada modeliramo čekanje prijema signala ili poruke!)

$\longrightarrow$ kod Dijkstre kod neistinite propozicije G *kontekst* odlučuje o daljnjoj akciji (nije od značaja za nas)

Dijkstrine "*guarded*" komande

"*guarded*" komande su oblika $G \rightarrow S$, gdje je:

- G je propozicija, koju nazivamo guard
- S je izvršna naredba naredba koju "guard" može blokirati.

Semantika:

Semantika *Promele* i originalna *Dijkstrina* semantika nisu potpuno iste. (Vidjeti primjer za *Promelu*)

$\longrightarrow$ u trenutku kada G postane istinit . . .

$\longrightarrow$ . . . izvodi se naredba S,
ako G nije istinit nastupa "blokada"

$\longrightarrow$ nije greška u `Promela` jeziku kada "guard" privremeno blokira!
(npr. kada modeliramo čekanje prijema signala ili poruke!)

$\longrightarrow$ kod Dijkstre kod neistinite propozicije G *kontekst* odlučuje o daljnjoj akciji (nije od značaja za nas)

Dijkstrine "*guarded*" komande–nastavak

Primjer (u `Promela` sintaksi):

```
.....
(A == msgOK) ⟶ G
naredbe iza ⟶ S
.....
```

naredbe S iza "*guarded*" komande G mogu biti izvedene samo ako varijabla A poprimi vrijednost *msgOK* .

Tko hoće više:

Dijkstra, Edsger W. "EWD472: Guarded commands, non-determinacy and formal. derivation of programs." (PDF)
http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD472.PDF

Dijkstrine "*guarded*" komande–nastavak

Primjer (u `Promela` sintaksi):

.....

`(A == msgOK)` $\longrightarrow$ *G*

`naredbe iza` $\longrightarrow$ *S*

.....

naredbe *S* iza "*guarded*" komande *G* mogu biti izvedene samo ako varijabla *A* poprimi vrijednost *msgOK* .

Tko hoće više:

Dijkstra, Edsger W. "EWD472: Guarded commands, non-determinacy and formal. derivation of programs." (PDF)
`http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD472.PDF`

Dijkstrine "*guarded*" komande–nastavak

Primjer (u `Promela` sintaksi):

```
......
(A == msgOK) ⟶ G
naredbe iza ⟶ S
......
```

naredbe S iza "*guarded*" komande G mogu biti izvedene samo ako varijabla A poprimi vrijednost *msgOK* .

Tko hoće više:

Dijkstra, Edsger W. "EWD472: Guarded commands, non-determinacy and formal. derivation of programs." (PDF)
http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD472.PDF

Dijkstrine "*guarded*" komande–nastavak

Primjer (u `Promela` sintaksi):

```
.....
(A == msgOK) ⟶ G
naredbe iza ⟶ S
.....
```

naredbe S iza "*guarded*" komande G mogu biti izvedene samo ako varijabla A poprimi vrijednost *msgOK* .

Tko hoće više:

Dijkstra, Edsger W. "EWD472: Guarded commands, non-determinacy and formal. derivation of programs." (PDF)
http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD472.PDF

Dijkstrine "*guarded*" komande–nastavak

Primjer (u `Promela` sintaksi):

```
.....
(A == msgOK) ⟶ G
naredbe iza ⟶ S
.....
```

naredbe S iza "*guarded*" komande G mogu biti izvedene samo ako varijabla A poprimi vrijednost *msgOK* .

Tko hoće više:

Dijkstra, Edsger W. "EWD472: Guarded commands, non-determinacy and formal. derivation of programs." (PDF)
http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD472.PDF

Dijkstrine "*guarded*" komande–nastavak

Primjer (u `Promela` sintaksi):

```
.....
(A == msgOK) ⟶ G
naredbe iza ⟶ S
.....
```

naredbe S iza "*guarded*" komande G mogu biti izvedene samo ako varijabla A poprimi vrijednost *msgOK* .

Tko hoće više:

Dijkstra, Edsger W. "EWD472: Guarded commands, non-determinacy and formal. derivation of programs." (PDF)
http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD472.PDF

Dijkstrine "*guarded*" komande–nastavak

Primjer (u `Promela` sintaksi):

```
.....
(A == msgOK) ⟶ G
naredbe iza ⟶ S
.....
```

naredbe S iza "*guarded*" komande G mogu biti izvedene samo ako varijabla A poprimi vrijednost *msgOK* .

Tko hoće više:

Dijkstra, Edsger W. "EWD472: Guarded commands, non-determinacy and formal. derivation of programs." (PDF)
http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD472.PDF

Dijkstrine "*guarded*" komande–nastavak

Primjer (u `Promela` sintaksi):

```
......
(A == msgOK) ⟶ G
naredbe iza ⟶ S
......
```

naredbe S iza "*guarded*" komande G mogu biti izvedene samo ako varijabla A poprimi vrijednost *msgOK* .

Tko hoće više:

Dijkstra, Edsger W. "EWD472: Guarded commands, non-determinacy and formal. derivation of programs." (PDF)
http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD472.PDF

Dijkstrine "*guarded*" komande–nastavak

Primjer (u `Promela` sintaksi):

```
......
(A == msgOK) ⟶ G
naredbe iza ⟶ S
......
```

naredbe S iza "*guarded*" komande G mogu biti izvedene samo ako varijabla A poprimi vrijednost *msgOK* .

Tko hoće više:

Dijkstra, Edsger W. "EWD472: Guarded commands, non-determinacy and formal. derivation of programs." (PDF)
http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD472.PDF

Dijkstrine "*guarded*" komande–nastavak

Primjer (u `Promela` sintaksi):

```
.....
(A == msgOK) ⟶ G
naredbe iza ⟶ S
.....
```

naredbe S iza "*guarded*" komande G mogu biti izvedene samo ako varijabla A poprimi vrijednost *msgOK*.

Tko hoće više:

Dijkstra, Edsger W. "EWD472: Guarded commands, non-determinacy and formal. derivation of programs." (PDF)
http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD472.PDF

Dijkstrine "*guarded*" komande–nastavak

Primjer (u `Promela` sintaksi):

```
.....
(A == msgOK) ⟶ G
naredbe iza ⟶ S
.....
```

naredbe S iza "*guarded*" komande G mogu biti izvedene samo ako varijabla A poprimi vrijednost *msgOK*.

Tko hoće više:

Dijkstra, Edsger W. "EWD472: Guarded commands, non-determinacy and formal. derivation of programs." (PDF)
`http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD472.PDF`

FSM–konačni automat

Definicija FSM

Konačni automat A_i je $5-torka\ (S, s_0, L, T, F)$, gdje je:

- **S** konačni skup *stanja* (S) (eng. *states*),
- s_0 *inicijalno* (početno) stanje, $s_0 \in S$,
- **L** konačni skup *labela*,
- **T** skup *prijelaza*, $T \subseteq (S \times L \times S)$,
- **F** skup konačnih (finalnih-završnih) stanja, $F \subseteq S$.

Za vježbu:

Nacrtajte FSM (automat) prema primjeru na ploči i označite svaku od sastavnica u (S, s_0, L, T, F).
Koja je veza prema UML ili SDL (MSC) i *Promela* procesu?
Kako ga možete programski realizirati ?

FSM–konačni automat

Definicija FSM

Konačni automat A_i je $5-torka\ (S, s_0, L, T, F)$, gdje je:

- → **S** konačni skup *stanja* (S) (eng. *states*),
- → **s_0** *inicijalno* (početno) stanje, $s_0 \in S$,
- → **L** konačni skup *labela*,
- → **T** skup *prijelaza*, $T \subseteq (S \times L \times S)$,
- → **F** skup konačnih (finalnih-završnih) stanja, $F \subseteq S$.

Za vježbu:

Nacrtajte FSM (automat) prema primjeru na ploči i označite svaku od sastavnica u (S, s_0, L, T, F).
Koja je veza prema UML ili SDL (MSC) i *Promela* procesu?
Kako ga možete programski realizirati ?

FSM–konačni automat

Definicija FSM

Konačni automat A_i je $5-torka\ (S, s_0, L, T, F)$, gdje je:

- → **S** konačni skup *stanja* (S) (eng. *states*),
- → **s_0** *inicijalno* (početno) stanje, $s_0 \in S$,
- → **L** konačni skup *labela*,
- → **T** skup *prijelaza*, $T \subseteq (S \times L \times S)$,
- → **F** skup konačnih (finalnih-završnih) stanja, $F \subseteq S$.

Za vježbu:

Nacrtajte FSM (automat) prema primjeru na ploči i označite svaku od sastavnica u (S, s_0, L, T, F).
Koja je veza prema UML ili SDL (MSC) i *Promela* procesu?
Kako ga možete programski realizirati ?

FSM–konačni automat

Definicija FSM

Konačni automat A_i je $5-torka\ (S, s_0, L, T, F)$, gdje je:

- → **S** konačni skup *stanja* (S) (eng. *states*),
- → **s_0** *inicijalno* (početno) stanje, $s_0 \in S$,
- → **L** konačni skup *labela*,
- → **T** skup *prijelaza*, $T \subseteq (S \times L \times S)$,
- → **F** skup konačnih (finalnih-završnih) stanja, $F \subseteq S$.

Za vježbu:

Nacrtajte FSM (automat) prema primjeru na ploči i označite svaku od sastavnica u (S, s_0, L, T, F).
Koja je veza prema UML ili SDL (MSC) i *Promela* procesu?
Kako ga možete programski realizirati ?

FSM–konačni automat

Definicija FSM

Konačni automat A_i je $5-torka\ (S, s_0, L, T, F)$, gdje je:

- → **S** konačni skup *stanja* (S) (eng. *states*),
- → **s_0** *inicijalno* (početno) stanje, $s_0 \in S$,
- → **L** konačni skup *labela*,
- → **T** skup *prijelaza*, $T \subseteq (S \times L \times S)$,
- → **F** skup `konačnih (finalnih-završnih) stanja`, $F \subseteq S$.

Za vježbu:

Nacrtajte FSM (automat) prema primjeru na ploči i označite svaku od sastavnica u (S, s_0, L, T, F).
Koja je veza prema UML ili SDL (MSC) i *Promela* procesu?
Kako ga možete programski realizirati ?

LTL logika i Büchi automat

- logičku formula kojom *verificiramo* (provjeravamo) zadana svojstva *SPIN* pretvara u posebnu vrstu automata – Büchi automat
- Büchi automat je posebna vrsta FSM koja prihvaća beskonačne sekvence labela L. Büchi automat interpretiramo nad Kripke strukturom
- kažemo: Büchi automat ima konačan broj stanja i svojstvo ω prihvaćanja:

  $\alpha_0, \alpha_1, \ldots, \alpha_i, \alpha_{i+1}, \ldots \alpha_n$ gdje $n \longrightarrow \infty$ i $\alpha_i \in L$
- u praktičnoj realizaciji Büchi automat je pridodan *CFSM* mreži automata
- oprez s *LTL* formulama: preporuča se uputreba do maksimalno tri temporalna *LTL* operatora zbog memorijskih ograničenja
- *SPIN* pridodaje u *Promela* model dodatne instrukcije za *LTL* formulu (`never claim` no o tome više kasnije . . .)

LTL logika i Büchi automat

- logičku formula kojom *verificiramo* (provjeravamo) zadana svojstva *SPIN* pretvara u posebnu vrstu automata – Büchi automat
- Büchi automat je posebna vrsta FSM koja prihvaća beskonačne sekvence labela *L*. Büchi automat interpretiramo nad Kripke strukturom
- kažemo: Büchi automat ima konačan broj stanja i svojstvo ω prihvaćanja:

  $\alpha_0, \alpha_1, \ldots, \alpha_i, \alpha_{i+1}, \ldots \alpha_n$ gdje $n \longrightarrow \infty$ i $\alpha_i \in L$
- u praktičnoj realizaciji Büchi automat je pridodan *CFSM* mreži automata
- oprez s *LTL* formulama: preporuča se uputreba do maksimalno tri temporalna *LTL* operatora zbog memorijskih ograničenja
- *SPIN* pridodaje u *Promela* model dodatne instrukcije za *LTL* formulu (`never claim` no o tome više kasnije . . .)

LTL logika i Büchi automat

- logičku formula kojom *verificiramo* (provjeravamo) zadana svojstva *SPIN* pretvara u posebnu vrstu automata – Büchi automat
- Büchi automat je posebna vrsta FSM koja prihvaća beskonačne sekvence labela L. Büchi automat interpretiramo nad Kripke strukturom
- kažemo: Büchi automat ima konačan broj stanja i svojstvo ω prihvaćanja:

  $\alpha_0, \alpha_1, \ldots, \alpha_i, \alpha_{i+1}, \ldots \alpha_n$ gdje $n \longrightarrow \infty$ i $\alpha_i \in L$
- u praktičnoj realizaciji Büchi automat je pridodan *CFSM* mreži automata
- oprez s *LTL* formulama: preporuča se uputreba do maksimalno tri temporalna *LTL* operatora zbog memorijskih ograničenja
- *SPIN* pridodaje u *Promela* model dodatne instrukcije za *LTL* formulu (`never claim` no o tome više kasnije . . .)

LTL logika i Büchi automat

- logičku formula kojom *verificiramo* (provjeravamo) zadana svojstva *SPIN* pretvara u posebnu vrstu automata – Büchi automat
- Büchi automat je posebna vrsta FSM koja prihvaća beskonačne sekvence labela L. Büchi automat interpretiramo nad Kripke strukturom
- kažemo: Büchi automat ima konačan broj stanja i svojstvo ω prihvaćanja:

  $\alpha_0, \alpha_1, \ldots, \alpha_i, \alpha_{i+1}, \ldots \alpha_n$ gdje $n \longrightarrow \infty$ i $\alpha_i \in L$
- u praktičnoj realizaciji Büchi automat je pridodan *CFSM* mreži automata
- oprez s *LTL* formulama: preporuča se uputreba do maksimalno tri temporalna *LTL* operatora zbog memorijskih ograničenja
- *SPIN* pridodaje u *Promela* model dodatne instrukcije za *LTL* formulu (`never claim` no o tome više kasnije . . .)

LTL logika i Büchi automat

- logičku formula kojom *verificiramo* (provjeravamo) zadana svojstva *SPIN* pretvara u posebnu vrstu automata – Büchi automat
- Büchi automat je posebna vrsta FSM koja prihvaća beskonačne sekvence labela L. Büchi automat interpretiramo nad Kripke strukturom
- kažemo: Büchi automat ima konačan broj stanja i svojstvo ω prihvaćanja:

  $\alpha_0, \alpha_1, \ldots, \alpha_i, \alpha_{i+1}, \ldots \alpha_n$ gdje $n \longrightarrow \infty$ i $\alpha_i \in L$
- u praktičnoj realizaciji Büchi automat je pridodan *CFSM* mreži automata
- oprez s *LTL* formulama: preporuča se uputreba do maksimalno tri temporalna *LTL* operatora zbog memorijskih ograničenja
- *SPIN* pridodaje u *Promela* model dodatne instrukcije za *LTL* formulu (`never claim` no o tome više kasnije . . .)

LTL logika i Büchi automat

- logičku formula kojom *verificiramo* (provjeravamo) zadana svojstva *SPIN* pretvara u posebnu vrstu automata – Büchi automat
- Büchi automat je posebna vrsta FSM koja prihvaća beskonačne sekvence labela L. Büchi automat interpretiramo nad Kripke strukturom
- kažemo: Büchi automat ima konačan broj stanja i svojstvo ω prihvaćanja:

  $\alpha_0, \alpha_1, \ldots, \alpha_i, \alpha_{i+1}, \ldots \alpha_n$ gdje $n \longrightarrow \infty$ i $\alpha_i \in L$
- u praktičnoj realizaciji Büchi automat je pridodan *CFSM* mreži automata
- oprez s *LTL* formulama: preporuča se uputreba do maksimalno tri temporalna *LTL* operatora zbog memorijskih ograničenja
- *SPIN* pridodaje u *Promela* model dodatne instrukcije za *LTL* formulu (`never claim` no o tome više kasnije . . .)

LTL logika

LTL – **L**inearna **T**emporalna **L**ogika

Sintaksa:

LTL sadrži propozicijske varijable $p_1, p_2, ...$, uobičajene logičke konektore $\neg$, $\vee$,$\wedge$,$\rightarrow$ i slijedeće temporalne modalne operatore:

* (`next`) (X, N, $\bigcirc$)
* (`always,globally`) (G, $\Box$) npr. $\Box$p
* (`eventually,finally`) (F, $\Diamond$) npr. $\Diamond$p
* (`until`) (U, $\mathcal{U}$) npr. pUq

LTL logika

LTL – **L**inearna **T**emporalna **L**ogika

Sintaksa:

LTL sadrži propozicijske varijable $p_1, p_2, ...$, uobičajene logičke konektore $\neg$, $\vee$,$\wedge$,$\rightarrow$ i slijedeće temporalne modalne operatore:

* (next) (X, N, $\bigcirc$)
* (always,globally) (G, $\Box$) npr. $\Box$p
* (eventually,finally) (F, $\Diamond$) npr. $\Diamond$p
* (until) (U, $\mathcal{U}$) npr. pUq

LTL logika

LTL – **L**inearna **T**emporalna **L**ogika

Sintaksa:

LTL sadrži propozicijske varijable $p_1, p_2, ...$, uobičajene logičke konektore $\neg$, $\vee$,$\wedge$,$\rightarrow$ i slijedeće temporalne modalne operatore:

* (next) (X, N, $\bigcirc$)
* (always,globally) (G, $\Box$) npr. $\Box$p
* (eventually,finally) (F, $\Diamond$) npr. $\Diamond$p
* (until) (U, $\mathcal{U}$) npr. pUq

LTL logika

LTL – **L**inearna **T**emporalna **L**ogika

Sintaksa:

LTL sadrži propozicijske varijable $p_1, p_2, ...$, uobičajene logičke konektore $\neg$, $\vee$,$\wedge$,$\rightarrow$ i slijedeće temporalne modalne operatore:

* (next) (X, N, $\bigcirc$)
* (always, globally) (G, $\Box$) npr. $\Box$p
* (eventually, finally) (F, $\Diamond$) npr. $\Diamond$p
* (until) (U, $\mathcal{U}$) npr. pUq

LTL logika – nastavak

Semantiku donosimo za operatore koji se koriste u *SPIN*-u.

Semantika

* $\Box\phi$: φ je istinit na *cijelom* putu (u Kripke strukturi)
* $\Diamond\phi$: φ je *na kraju, u konačnici* istinit (istinit je negdje na putu)
* $\psi\mathcal{U}\phi$: φ je istinit u trenutnoj i budućim pozicijama, a ψ mora biti istinit do te pozicije

Za vježbu:

Skicirati dijagram za svaki od operatora !

LTL logika – nastavak

Semantiku donosimo za operatore koji se koriste u *SPIN*-u.

Semantika

* $\Box\phi$: φ je istinit na *cijelom* putu (u Kripke strukturi)
* $\Diamond\phi$: φ je *na kraju, u konačnici* istinit (istinit je negdje na putu)
* $\psi\mathcal{U}\phi$: φ je istinit u trenutnoj i budućim pozicijama, a ψ mora biti istinit do te pozicije

Za vježbu:

Skicirati dijagram za svaki od operatora !

LTL logika – nastavak

Semantiku donosimo za operatore koji se koriste u *SPIN*-u.

Semantika

* $\Box\phi$: φ je istinit na *cijelom* putu (u Kripke strukturi)
* $\Diamond\phi$: φ je *na kraju, u konačnici* istinit (istinit je negdje na putu)
* $\psi\mathcal{U}\phi$: φ je istinit u trenutnoj i budućim pozicijama, a ψ mora biti istinit do te pozicije

Za vježbu:

Skicirati dijagram za svaki od operatora !

Neka je zadan *FSM* automat A=(S, s_0, L, T, F).
Uvode se tri posebna tipa labela L:

1. *A.A* je skup stanja označenih kao stanja prihvaćanja (*eng. accept-state labels*)
2. *A.E* je skup stanja označenih kao konačna stanja (*eng. end-state labels*)
3. *A.P* je skup stanja označenih kao stanja napredovanja (*eng. progress-state labels*)

Zašto posebne labele *A.A*, *A.P* i *A.E*

Posebne labele su dio *Promele*.
Na osnovu njihovog položaja *SPIN* može utvrditi neizvedene prijelaze, nemogućnost završetka . . .
. . . mogu se usporediti sa *BREAK–points* i *WATCH–points* tijekom "debuggiranja" programa

Neka je zadan *FSM* automat A=(S, s_0, L, T, F).
Uvode se tri posebna tipa labela L:

1. *A.A* je skup stanja označenih kao stanja prihvaćanja (*eng. accept-state labels*)
2. *A.E* je skup stanja označenih kao konačna stanja (*eng. end-state labels*)
3. *A.P* je skup stanja označenih kao stanja napredovanja (*eng. progress-state labels*)

Zašto posebne labele *A.A*, *A.P* i *A.E*

Posebne labele su dio *Promele*.
Na osnovu njihovog položaja *SPIN* može utvrditi neizvedene prijelaze, nemogućnost završetka . . .
. . . mogu se usporediti sa *BREAK–points* i *WATCH–points* tijekom "debuggiranja" programa

Neka je zadan *FSM* automat A=(S, s_0, L, T, F).
Uvode se tri posebna tipa labela L:

1. *A.A* je skup stanja označenih kao stanja prihvaćanja (*eng. accept-state labels*)
2. *A.E* je skup stanja označenih kao konačna stanja (*eng. end-state labels*)
3. *A.P* je skup stanja označenih kao stanja napredovanja (*eng. progress-state labels*)

Zašto posebne labele *A.A*, *A.P* i *A.E*

Posebne labele su dio *Promele*.
Na osnovu njihovog položaja *SPIN* može utvrditi neizvedene prijelaze, nemogućnost završetka . . .
. . . mogu se usporediti sa *BREAK–points* i *WATCH–points* tijekom "debuggiranja" programa

Asinkroni produkt

Asinkroni produkt konačnog skupa automata $A_1, A_2, \ldots A_n$ je također novi *FSM* (S, s_0, L, T, F):

- $A.S$ je kartezijev produkt $A_1.S \times \ldots \times A_n.S$,
- $A.s_0$ je n-torka $(A_1.s_0, \ldots, A_n.s_0)$,
- $A.L$ je unija skupova $A_1.L \cup \ldots \cup A_n.L$,
- $A.T$ je skup *n-torki*, $((x_1, \ldots, x_n), |, (y_1, \ldots, y_n))$ takvi da $\exists i, 1 \leq i \leq n, (x_i, |, y_i) \in A_i.T$ i $\forall j, 1 \leq j \leq n, i \neq j \longrightarrow (x_i \equiv y_j)$
- $A.F$ je podskup $A.S$ takav da $\forall (A_1.s, \ldots, A_n.s) \in A.F$, $\exists i, A_i.s \in A_j.F$

Asinkroni produkt

Asinkroni produkt konačnog skupa automata $A_1, A_2, \ldots A_n$ je također novi *FSM* (S, s_0, L, T, F):

- $A.S$ je kartezijev produkt $A_1.S \times \ldots \times A_n.S$,
- $A.s_0$ je n-torka $(A_1.s_0, \ldots, A_n.s_0)$,
- $A.L$ je unija skupova $A_1.L \cup \ldots \cup A_n.L$,
- $A.T$ je skup *n-torki*, $((x_1, \ldots, x_n), |, (y_1, \ldots, y_n))$ takvi da $\exists i, 1 \leq i \leq n$, $(x_i, |, y_i) \in A_i.T$ i $\forall j, 1 \leq j \leq n, i \neq j \longrightarrow (x_i \equiv y_j)$
- $A.F$ je podskup $A.S$ takav da $\forall (A_1.s, \ldots, A_n.s) \in A.F$, $\exists i, A_i.s \in A_j.F$

Asinkroni produkt

Asinkroni produkt konačnog skupa automata $A_1, A_2, \ldots A_n$ je također novi *FSM* (S, s_0, L, T, F):

- $A.S$ je kartezijev produkt $A_1.S \times \ldots \times A_n.S$,
- $A.s_0$ je n-torka $(A_1.s_0, \ldots, A_n.s_0)$,
- $A.L$ je unija skupova $A_1.L \cup \ldots \cup A_n.L$,
- $A.T$ je skup *n-torki*, $((x_1, \ldots, x_n), |, (y_1, \ldots, y_n))$ takvi da $\exists i, 1 \leq i \leq n$, $(x_i, |, y_i) \in A_i.T$ i $\forall j, 1 \leq j \leq n, i \neq j \longrightarrow (x_i \equiv y_j)$
- $A.F$ je podskup $A.S$ takav da $\forall (A_1.s, \ldots, A_n.s) \in A.F$, $\exists i, A_i.s \in A_j.F$

Asinkroni produkt

Asinkroni produkt konačnog skupa automata $A_1, A_2, \ldots A_n$ je također novi *FSM* (S, s_0, L, T, F):

- $A.S$ je kartezijev produkt $A_1.S \times \ldots \times A_n.S$,
- $A.s_0$ je n-torka $(A_1.s_0, \ldots, A_n.s_0)$,
- $A.L$ je unija skupova $A_1.L \cup \ldots \cup A_n.L$,
- $A.T$ je skup *n-torki*, $((x_1, \ldots, x_n), |, (y_1, \ldots, y_n))$ takvi da $\exists i, 1 \leq i \leq n, (x_i, |, y_i) \in A_i.T$ i $\forall j, 1 \leq j \leq n, i \neq j \longrightarrow (x_i \equiv y_j)$
- $A.F$ je podskup $A.S$ takav da $\forall (A_1.s, \ldots, A_n.s) \in A.F$, $\exists i, A_i.s \in A_j.F$

Asinkroni produkt

Asinkroni produkt konačnog skupa automata $A_1, A_2, \ldots A_n$ je također novi *FSM* (S, s_0, L, T, F):

- $A.S$ je kartezijev produkt $A_1.S \times \ldots \times A_n.S$,
- $A.s_0$ je n-torka $(A_1.s_0, \ldots, A_n.s_0)$,
- $A.L$ je unija skupova $A_1.L \cup \ldots \cup A_n.L$,
- $A.T$ je skup *n-torki*, $((x_1, \ldots, x_n), |, (y_1, \ldots, y_n))$ takvi da $\exists i, 1 \leq i \leq n$, $(x_i, |, y_i) \in A_i.T$ i $\forall j, 1 \leq j \leq n, i \neq j \longrightarrow (x_i \equiv y_j)$
- $A.F$ je podskup $A.S$ takav da $\forall (A_1.s, \ldots, A_n.s) \in A.F$, $\exists i, A_i.s \in A_j.F$

Za vježbu:

1. Precrtati FSM sa ploče i odrediti asinkroni produkt.
2. U kakvoj su vezi asinkroni produkt i graf dostupnosti ?
3. U kakvoj su vezi graf dostupnosti i Kripke struktura ?
4. U kakvoj su vezi asinkroni produkt Kripke struktura ?

Za vježbu:

1. Precrtati FSM sa ploče i odrediti asinkroni produkt.
2. U kakvoj su vezi asinkroni produkt i graf dostupnosti ?
3. U kakvoj su vezi graf dostupnosti i Kripke struktura ?
4. U kakvoj su vezi asinkroni produkt Kripke struktura ?

Asinkroni produkt i konkurentnost

- Asinkroni produkt opisuje ponašanje sustava opisanog kao *CFSM*
- **Važno:** dozvoljen je samo jedan prijelaz po automatu
- na taj način *asinkroni produkt* opisuje "*interleaving*" semantiku konkurentnih procesa

Asinkroni produkt i konkurentnost

* Asinkroni produkt opisuje ponašanje sustava opisanog kao *CFSM*
* **Važno:** dozvoljen je samo jedan prijelaz po automatu
* na taj način *asinkroni produkt* opisuje "*interleaving*" semantiku konkurentnih procesa

Asinkroni produkt i konkurentnost

* Asinkroni produkt opisuje ponašanje sustava opisanog kao *CFSM*
* **Važno:** dozvoljen je samo jedan prijelaz po automatu
* na taj način *asinkroni produkt* opisuje "*interleaving*" semantiku konkurentnih procesa

Sinkroni produkt i *LTL* formule

Neka je sustav *Sys* opisan *Promela* modelom koji se sastoji od Büchi automata B i asinkronog produkta automata A_i iz *CFSM*:

$$Sys = B \oplus \prod_{i=1}^{n} A_i$$

- Operator $\oplus$ predstavlja sinkroni produkt.

Sinkroni produkt

Sinkroni produkt je automat A=(S, s_0, L, T, F):

- $A.S$ je kartezijev produkt $P'.S \times B.S$, gdje P' ima pridodane *nil* (prazne) prijelaze u svakom stanju u P koje nema napretka (progresa)
- $A.s_0$ je $(P.s_0, B.s_0)$,
- $A.L$ je $P'.L \times B.L$,
- $A.T$ je skup parova (t_1, t_2) takvi da $t_1 \in P'.T$ i $t_2 \in B.T$,
- $A.F$ je skup parova $(s_1, s_2) \in A.S$ gdje $s_1 \in P.F \vee s_2 \in B.F$

Sinkroni produkt

Sinkroni produkt je automat A=(S, s_0, L, T, F):

- $A.S$ je kartezijev produkt $P'.S \times B.S$, gdje P' ima pridodane *nil* (prazne) prijelaze u svakom stanju u P koje nema napretka (progresa)
- $A.s_0$ je $(P.s_0, B.s_0)$,
- $A.L$ je $P'.L \times B.L$,
- $A.T$ je skup parova (t_1, t_2) takvi da $t_1 \in P'.T$ i $t_2 \in B.T$,
- $A.F$ je skup parova $(s_1, s_2) \in A.S$ gdje $s_1 \in P.F \vee s_2 \in B.F$

Sinkroni produkt

Sinkroni produkt je automat A=(S, s_0, L, T, F):

- $A.S$ je kartezijev produkt $P'.S \times B.S$, gdje P' ima pridodane *nil* (prazne) prijelaze u svakom stanju u P koje nema napretka (progresa)
- $A.s_0$ je $(P.s_0, B.s_0)$,
- $A.L$ je $P'.L \times B.L$,
- $A.T$ je skup parova (t_1, t_2) takvi da $t_1 \in P'.T$ i $t_2 \in B.T$,
- $A.F$ je skup parova $(s_1, s_2) \in A.S$ gdje $s_1 \in P.F \vee s_2 \in B.F$

Sinkroni produkt

Sinkroni produkt je automat A=(S, s_0, L, T, F):

- $A.S$ je kartezijev produkt $P'.S \times B.S$, gdje P' ima pridodane *nil* (prazne) prijelaze u svakom stanju u P koje nema napretka (progresa)
- $A.s_0$ je $(P.s_0, B.s_0)$,
- $A.L$ je $P'.L \times B.L$,
- $A.T$ je skup parova (t_1, t_2) takvi da $t_1 \in P'.T$ i $t_2 \in B.T$,
- $A.F$ je skup parova $(s_1, s_2) \in A.S$ gdje $s_1 \in P.F \vee s_2 \in B.F$

Sinkroni produkt

Sinkroni produkt je automat A=(S, s_0, L, T, F):

- $A.S$ je kartezijev produkt $P'.S \times B.S$, gdje P' ima pridodane *nil* (prazne) prijelaze u svakom stanju u P koje nema napretka (progresa)
- $A.s_0$ je $(P.s_0, B.s_0)$,
- $A.L$ je $P'.L \times B.L$,
- $A.T$ je skup parova (t_1, t_2) takvi da $t_1 \in P'.T$ i $t_2 \in B.T$,
- $A.F$ je skup parova $(s_1, s_2) \in A.S$ gdje $s_1 \in P.F \vee s_2 \in B.F$

Procesne algebre: CSP i SPIN

Spomenimo najvažnije poveznice između *CSP* algebri (**C**ommunicating **S**equential **P**rocesses) i *SPIN*/Promele

$\longrightarrow$ $\mathcal{A} = \alpha_1, \alpha_2, \ldots, \alpha_n \equiv A.L$
(alfabet $\mathcal{A}$ je predstavljen labelama u *FSM*)

$\longrightarrow$ operatori *CSP* algebre (deterministički i nedeterministički izbor, kompozicija CSP procesa)
$\Longrightarrow$ realizirani kroz sinkronu i asinkronu kompoziciju automata

Procesne algebre: CSP i SPIN

Spomenimo najvažnije poveznice između *CSP* algebri (**C**ommunicating **S**equential **P**rocesses) i *SPIN*/Promele

$\longrightarrow$ $\mathcal{A} = \alpha_1, \alpha_2, \ldots, \alpha_n \equiv A.L$
(alfabet $\mathcal{A}$ je predstavljen labelama u *FSM*)

$\longrightarrow$ operatori *CSP* algebre (deterministički i nedeterministički izbor, kompozicija CSP procesa)
$\Longrightarrow$ realizirani kroz sinkronu i asinkronu kompoziciju automata

. . . i na kraju teme

- kvaliteta *SPIN* programskog alata spada u klasu "*industrial strength*" odnosno koristi se kao stabilan, pouzdan i upotrebljiv "model–checker" sustav pogodan za industriju i istraživanje
- pravilna upotreba ovisi o poznavanju svih mogućnosti i opcija
- mnogi alati na ključnim mjestima koriste *SPIN*: (*JavaPathFinder, Bandera* . . .)
- . . . slijedi praktična primjena

Šira literatura:

(1.) Gerard J. Holzmann: *The SPIN Model Checker–Primer and Reference Manual*, Addison-Wesley., 2004
(pokriva u tančine sve aspekte *SPIN* Promele – od teorijske podloge do praktične upotrebe)

(2.) izvor "on–line" dokumentacije je:
`http://spinroot.com/spin/Man/index.html`

(3.) `http://spinroot.com/spin/whatispin.html`
je službena stranica autora Spina koja sadržava mnogo članaka kao i sve do sada održane simpozije