

IME I PREZIME: ~~XXXXXXXXXX~~

Ak. god. 2014/2015

JMBAG: ~~XXXXXXXXXX~~

## 1. domaća zadaća iz Formalnih metoda u oblikovanju sustava

### NuSMV

Najprije je potrebno instalirati sustav NuSMV prema uputama u datoteci "NuSMV\_upute.pdf"

#### 1. dio

1.1. Prouči primjer mutex\_3ex.smv.

1.2. Specificiraj i napiši u CTL notaciji obilježje sigurnosti (engl. *safety property*):  
«Dva procesa ne mogu biti istovremeno u kritičnom odsječku.»

Potrebno je napisati dva oblika obilježja:

- a) specifikacija da je moguće jedno nepoželjno ponašanje (engl. *refutation*)
- b) specifikacija da nema nepoželjnog ponašanja

Nepoželjno ponašanje je u ovom slučaju istovremeno nalaženje u kritičnom odsječku.

- a)  $\text{CTL}_{\text{SPEC}} \text{ EF } (\text{proc}0.\text{state} = \text{critical} \wedge \text{proc}1.\text{state} = \text{critical})$
- b)  $\text{CTL}_{\text{SPEC}} \text{ AG}! (\text{proc}0.\text{state} = \text{critical} \wedge \text{proc}1.\text{state} = \text{critical})$

1.3. Utvrdi pomoću sustava NuSMV je li ispunjeno navedeno obilježje. Objasni rezultat na temelju koda primjera.

Navedeno obilježje je ispunjeno pod b

1.4. Specificiraj i napiši u CTL notaciji obilježje (engl. *liveness property*):  
«Ako proces pokuša ući u kritični odsječak, konačno će i ući»

Specifikaciju napiši za oba procesa.

$\text{CTL}_{\text{PEC}} \text{ AF}(\text{proc}0.\text{state} = \text{entering} \rightarrow \text{AF}(\text{proc}0.\text{state} = \text{critical}))$  stanje  
 $\text{CTL}_{\text{PGC}} \text{ AG}(\text{proc}1.\text{state} = \text{entering} \rightarrow \text{AF}(\text{proc}1.\text{state} = \text{critical}))$  stanje

- 1.5. Utvrdi da li je zadovoljeno navedeno obilježje. Koji su sve problemi s ovom implementacijom?

Vjezdođeno navedeno obilježje jer je moguće  
da <sup>veliki</sup> proces beskonačno puta bude u kritičnom odsječku,  
dok drugi nikada neće ući

- 1.6. U mutex\_3ex.smv dodaj ograničenje pravednosti (engl. *fairness*): svaka instanca procesa obavlja se beskonačno mnogo puta.

JUSTICE running

- 1.7. Ponovno provjeri prethodno obilježje. Što smo postigli s ovim ograničenjem pravednosti?

Postigli smo to da procesor mora naišmjence upravljati  
procesima (ne učivo naišmjence, ali ne smije stalno upravljati ~~istim~~  
istim procesom). Ipak, nije mu definirano kako mora ujma  
upravljati (nije rečeno) pa obilježje još uvijek nije  
zadovoljeno

- 1.8. U mutex\_3ex.smv dalje dodaj ograničenje pravednosti: svaka instanca procesa ne može beskonačno dugo ostati u **kritičnom** odsječku. Provjeri sad svojstvo životnosti za mutex\_3ex.smv.

JUSTICE !(state = critical)

SUSMCE !(state = non critical)

R&D ! / i

- 1.9. U mutex\_3ex.smv dodaj još jedno ograničenje pravednosti: svaka instanca procesa ne može beskonačno dugo ostati u **nekritičnom** odsječku. Provjeri sad svojstvo životnosti za mutex\_3ex.smv.

SUSPENSE ! (state = noncritical)

ŽIVOTNOST  
SIGURNUĆE  
NEBLOKIRANJE  
NEDETERMI. RED.

- 1.10. Specificiraj i napiši u CTL notaciji:

«Ako proces proc0 uđe u kritični odsječak, proc0 neće ponovo ući u kritični odsječak sve dok proc1 nije prošao kroz svoj kritični odsječak.»

CTLSPEC  $AG(proc0.state = \text{existing} \rightarrow A[\neg(proc0.state = \text{critical}) \vee proc1.state = \text{existing}])$

- 1.11. Utvrdi je li zadovoljeno navedeno obilježje za mutex\_3ex.smv (uz ograničenja pravednosti). Koja obilježja protokola međusobnog isključivanja rješavaju ograničenja pravednosti prethodno navedena, a koji problem je još uvijek prisutan?

Zadovoljeno je, ali ako moramo nise putar za redom izvršiti isti proces, svali putar čemo morati očekati neki drugi proces (uvjet iemjenjivanja)

## 2. dio

2.1. Prouči primjer mutex\_4ex.smv

2.2. Specificiraj i napiši u CTL notaciji obilježje sigurnosti (engl. *safety property*):  
«Dva procesa ne mogu biti istovremeno u kritičnom odsječku.»

Potrebno je napisati dva oblika obilježja:

- a) specifikacija da je moguće jedno nepoželjno ponašanje (engl. *refutation*)
- b) specifikacija da nema nepoželjnog ponašanja

CTL SPEC  $\text{EF} (\text{proc0.state} = \text{critical} \wedge \text{proc1.state} = \text{critical})$   
CTL SPEC  $\text{AG} !(\text{proc0.state} = \text{critical} \wedge \text{proc1.state} = \text{critical})$

2.3. Utvrdi da li je ispunjeno zadano obilježje. Objasni rezultat na temelju koda primjera.

Ispunjeno je pod b

2.4. Specificiraj i napiši u CTL notaciji obilježje (engl. *liveness property*):  
«Ako proces pokuša ući u kritični odsječak, konačno će i ući»

Specifikaciju napiši za oba procesa.

CTL SPEC  $\text{AG} (\text{proc0.state} = \text{reading} \rightarrow \text{AF} (\text{proc0.state} = \text{critical}))$

~~Proc1~~  
CTL SPEC  $\text{AG} (\text{proc1.state} = \text{reading} \rightarrow \text{AF} (\text{proc1.state} = \text{critical}))$

- 2.5. Utvrdi je li zadovoljeno navedeno obilježje. Objasni koji je problem u ovoj implementaciji međusobnog isključivanja.

Nije, proces može zagnaviti a stanje reading.

- 2.6. Specificiraj i napiši u CTL notaciji taj problem i provjeri ga pomoću NuSMV sustava.

$\text{EF}(\text{AG}(\text{proc0.state} = \text{reading} \wedge \text{proc1.state} = \text{reading}))$

- 2.7. Prouči primjer mutex\_5ex.smv.

- 2.8. Je li zadovoljeno obilježje sigurnosti (2. dio, 2. pitanje)?

Zadovoljeno - je za b)

- 2.9. Je li zadovoljeno obilježje životnosti (2. dio, 4. pitanje)?

Nije  $\text{AG}(\text{proc0.state} = \text{testFlag} \rightarrow \text{AF}(\text{proc0.state} = \text{critical}))$

- 2.10. Koji je problem u ovoj implementaciji međusobnog isključivanja? Gdje sustav može «zapeti»? Problem specificiraj u CTL notaciji i provjeri pomoću sustava NuSMV.

Može „zapeti“ u test flag - anima

$\text{EF}(\text{AG}(\text{proc0.state} = \text{testFlag} \wedge \text{proc1.state} = \text{testFlag}))$

2.11. Prouči primjer mutex\_6ex.smv

Ovo je primjer uspješne implementacije međusobnog isključivanja. Zasniva se na rješenju kojeg je predložio T. Dekker a opisao E. W. Dijkstra.

2.12. Provjeri svojstva sigurnosti i životnosti. jesu li zadovoljena (uz dodavanje tri ograničenja pristranosti iz 1. dijela)?

Zadovoljeni su

2.13. Koje se ideje za kontrolu pristupa kritičnom odsječku iz prethodnih (neuspješnih) pokušaja nameću u ovom rješenju?

Trebalo bi zabraniti da proces digne ruku ako vidi da drugi proces ima ruku u zraku

2.14. Prouči primjer mutex\_7ex.smv

Ovaj je primjer implementacija Petersonovog algoritma, koji predstavlja pojednostavnjeno prethodnog (Dekkerovog) algoritma.

2.15. Je li zadovoljeno obilježje sigurnosti?

Zadovoljeno je

2.16. Specificiraj i napiši u CTL notaciji obilježje životnosti. Provjeri ga pomoći sustava NuSMV. Je li to obilježje zadovoljeno (uz dodavanje tri ograničenja pravednosti iz 1. dijela)?

CTL<sub>SPEC</sub>  $\text{AG } (\text{proc0.state} = \text{trying} \rightarrow \text{AF}(\text{proc0.state} = \text{critical}))$   
CTL<sub>SPEC</sub>  $\text{AG } (\text{proc1.state} = \text{trying} \rightarrow \text{AF}(\text{proc1.state} = \text{critical}))$

### 3. dio

Prouči potpoglavlja 3.1, 3.2, 3.5 i 3.7 iz NuSMV priručnika "NuSMV 2.5 User Manual". Nakon toga riješi sljedeće zadatke:

- 3.1. Pokreni interaktivno lјusku NuSMV-a. Učitaj model zadan datotekom mutex\_3ex\_int.smv.
- 3.2. Inicijaliziraj sustav za verifikaciju. Ukratko obrazloži što se sve događa prilikom pokretanja naredbe "go".

read-model - učitava NuSMV model  
flatten-hierarchy - zamjenjuje stvarnih parametara formalnim  
encode-variables - generira Booleove BDD varijable i ADD  
build-model - kompajlira hierarhiju u BDD (inicijalna stanja,  
invariante i prijelazi odnosi)

- 3.3. Simuliraj kretanje kroz 3 stanja (od proizvoljno odabranog početnoga). Navedi dvije naredbe koje se koriste da bi se to ostvarilo. Koju naredbu treba koristiti da bi se ispisao trag prolaska kroz ta stanja?

pick-state [-i] - odabire stanja iz skupine inicijalnih got. state (state label)  
simulate -k 3  
Da bi se ispisao trag: show-traces

- 3.4. Provjeri stroj s konačnim brojem stanja. Kakva je relacija prijelaza tog automata? Može li doći do potpunog zastoja?

check fsm, ima dočeknu relaciju prijelaza,  
ne može doći do potpunog zastoja

- 3.5. Koliko ukupno postoji stanja u modelu, a koliko postoji dosezljivih (engl. *reachable*) stanja? (napomena: *diameter* - promjer FSM-a je minimalan broj koraka potrebnih da bi se došlo do svih dosezljivih stanja)

print-reachable-states      16 - reachable  
                                  32 - nedosezljivih  
                                  check-spec

- 3.6. Provjeri prvu po redu CTL specifikaciju (redni broj 0). Je li ona istinita ili lažna? Koje obilježje protokola međusobnog isključivanja se njome provjerava? Je li to obilježje zadovoljeno?

Ova specifikacija je lažna. Provjerava se da li je moguće da u jednom trenutku oba procesa bude u K.O. - provjerava se sigurnost, obilježje je endoroftno

- 3.7. Provjeri drugu po redu CTL specifikaciju (redni broj 1). Je li ona istinita ili lažna? Koje obilježje protokola međusobnog isključivanja se njome provjerava? Je li to obilježje zadovoljeno?

Istinita je i provjerava se vrijeđnost, obilježje je zadovoljeno

- 3.8. Sada ukloni obilježja pravednosti iz datoteke mutex\_3ex\_int.smv, ponovi postupak učitavanja i pripreme za verifikaciju te provjeri drugu po redu CTL specifikaciju. Imat će kakve promjene u odnosu na prethodni zadatak?

Imat će promjene, 2. specifikacija nije zadovoljena i procesi se beskonačno vrte

- 3.9. Prouči naredbe za provjeru svojstava sustava za rad u stvarnom vremenu koje su zadane s ključnom riječi "COMPUTE" u datoteci mutex\_3ex\_int.smv. Koje je značenje svake od tih naredbi?

COMPUTE MIN (proc0.state = noncritical, proc0.state = exiting)  
-računa duljinu najkratčeg puta između stanja noncritical  
i exiting procesa 0 - ovde je vrijednost 3  
COMPUTE MAX (proc0.state = noncritical, proc0.state = critical)  
-najduže duljinu puta između noncritical i exiting,  
ovde je inf

- 3.10. Provjeri te naredbe u sustavu NuSMV (prva COMPUTE naredba ima redni broj 2 u modelu, a druga redni broj 3). Navedi rezultat izvođenja tih dviju naredbi. Uzima li naredba COMPUTE u obzir navedena ograničenja pravednosti?

Rezultati: COMPUTE MIN 3  
MAX Beskonačno

Ne uzima ograničenje pravednosti, bez obzira na  
ustnice, rezultat je isti

## 4. dio

4.1. Zadani kôd u NuSMV-u sadrži implicitni nedeterminizam uzrokovani varijablim *request*. Izmijeni zadani kôd tako da sadrži **isključivo** eksplisitni nedeterminizam.

```
MODULE main
VAR
    request: boolean (TRUE FALSE)
    flag: {red, blue};
ASSIGN
    init(flag) := red;
    next(flag) := case
        request:blue;
        TRUE: red;
    esac;
    next(request) := case
        TRUE : {TRUE ifi FALSE}
    esac;
```

4.2. Za zadani kôd u NuSMV-u nacrtaj odgovarajuću Kripke strukturu i odredi:

- skup svih mogućih stanja –  $S_A$
- skup svih dosezljivih stanja –  $S_R$  (uz pretpostavku da su sva početna stanja dosezljiva)

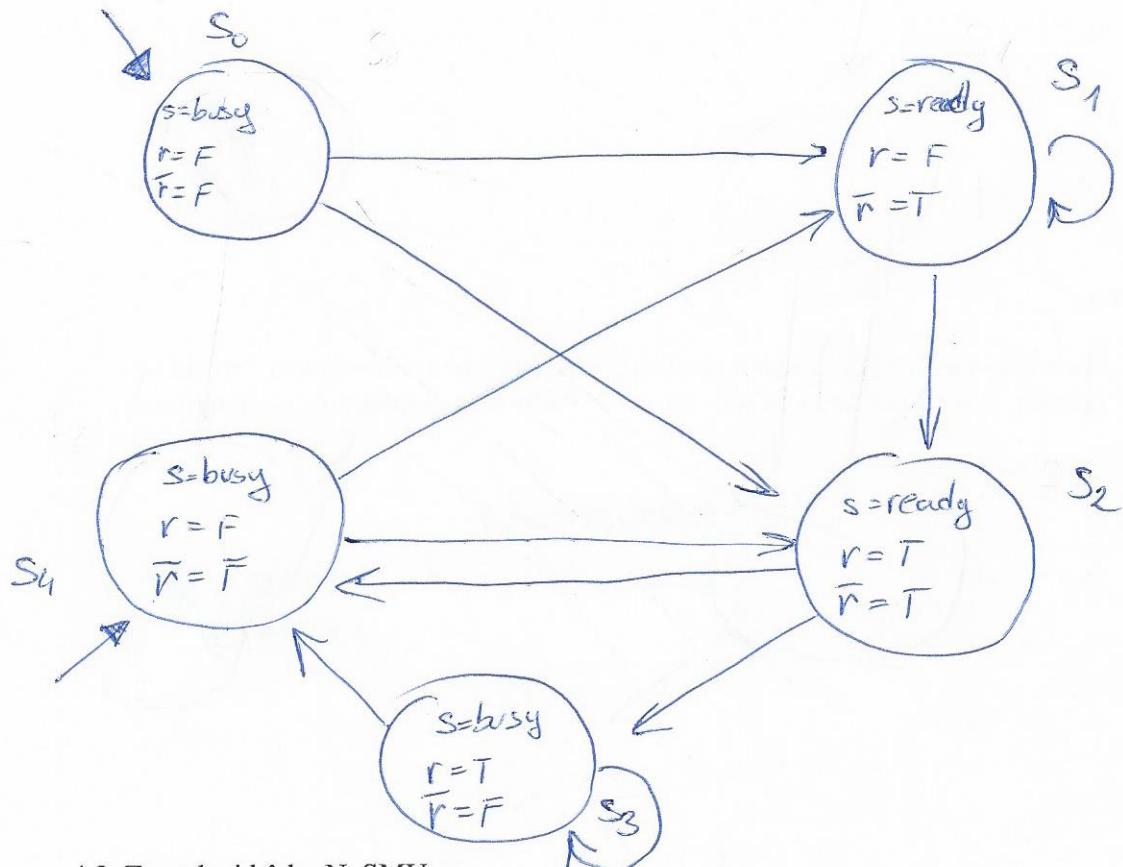
*eksplisitni nedeterminizam*

```
MODULE main
VAR
    request : boolean;
    status : {ready, busy};
    negReq : boolean;
ASSIGN
    init(status) := busy;
    init(request) := FALSE;
    next(status) := case
        request:busy;
        TRUE:ready;
    esac;
    next(negReq) := !request;
```

8 stanja

req	negreq	status
0	1	r
0	0	b
1	1	r
1	0	b
1	0	r
0	1	b
0	0	r
0	1	b

9



4.3. Za zadani kôd u NuSMV-u:

- 1) opiši na koji se način izvršavaju moduli sustava;
- 2) izmijeni i napiši kompletan kôd tako da se moduli izvršavaju asinkrono;
- 3) napiši ograničenje pravednosti: varijabla value instance modula invertora mora biti jednaka TRUE beskonačno često.

```

MODULE main
VAR
    invertorA : invert;
    invertorB : invert;

MODULE invert
VAR
    value : boolean;
ASSIGN
    init(value):= FALSE;
    next(value):= !value;
  
```

- 1) Moduli se izvršavaju paralelno u sinkronom načinu rada, a odvojeno u asinkronom
- 2) VAR invertor A: process invert;  
invertor B: process invert;
- 3) SOURCE invertorA.value = TRUE

IME I PREZIME: \_\_\_\_\_  
JMBAG: \_\_\_\_\_

Ak. god. 2014/2015

## 2. domaća zadaća iz Formalnih metoda u oblikovanju sustava

### Java PathFinder

Najprije je potrebno instalirati sustav Java PathFinder prema uputama u datoteci "Java\_PathFinder\_instalacija.pdf"

Svrha 1. dijela 2. domaće zadaće je upoznavanje s projektom jpf-core i pokretanje provjere modela jednostavnih primjera programa. U 2. dijelu domaće zadaće uvode se dodatni razredi slušači koji nadograđuju osnovnu funkcionalnost projekta jpf-core. 3. dio domaće zadaće pokriva izvođenje provjere modela nad primjerima iz projekta jpf-aprop, koji je dodatni projekt koji se može koristiti za provjeru različitih anotacija u programu. Konačno, u 4. dijelu domaće zadaće studenti će napraviti svoj projekt od početka, uključiti korištenje projekata jpf-core i jpf-aprop te provjeravati model zadanog programa uz stalne izmjene.

#### 1. dio - Provjera modela za jednostavne primjere u projektu jpf-core

Proučite strukturu projekta jpf-core. Primjeri nad kojim će se raditi provjera modela nalaze se u paketu src/examples. Osim ako nije drugačije zadano, programi se mogu pokrenuti desnim klikom na odgovarajuću \*.jpf datoteku i odabirom opcije "Verify..." Alternativno, ako niste uspjeli podesiti plugin "Verify..." da radi (vidjeti instalacijske upute), moguće je pokrenuti program unoseći puni put u komandnoj liniji koji je obično ovakvog oblika:

```
java -jar C:\Users\Korisnik\projects\jpf\jpf-core\build\RunJPF.jar  
+shell.port=4242 C:\Users\Korisnik\Documents\NetBeansProjects\jpf-  
core\src\examples\HelloWorld.jpf
```

**1.1.** Otvorite konfiguracijsku datoteku projekta jpf-core koja se naziva jpf.properties i koja se nalazi u korijenskom direktoriju projekta. Do datoteke se može doći putem kartice "Files" koja se nalazi pokraj kartice "Projects" u NetBeansu ili putem Windows Explorera.

Navedite koja se defaultna strategija koristi za pretragu prostora stanja. Koja se standardna svojstva provjeravaju prilikom pretrage? Osim naziva strategije i svojstava, navedite i puna kvalificirajuća imena razreda u projektu jpf-core koji za to služe.

54. linija : DFSearch (search.class = gov.nasa.jpf.search.DFSearch)  
78. linija : NotDeadlockedProperty, NoUncaughtExceptionProperty.  
(gov.nasa.jpf.vm.NotDeadlockedProperty )  
(gov.nasa.jpf.vm.NoUncaughtExceptionProperty )

1.2. Proučite najjednostavniji primjer aplikacije HelloWorld.java. Što se ispisuje pokretanjem provjere modela tog programa u dijelu nakon search started? Koje su pogreške dojavljene?

Zapisuje se točno vrijeme kada je započela pretraga te izlaz iz sustava ("I won't say it!").  
Nema pogresaka.

1.3. Proučite primjer ograničenog međuspremnika BoundedBuffer.java. Navedite koji su sudionici u ovom primjeru.

Producer i Consumer  $\Rightarrow$  static class Producer extends Thread  
static class Consumer extends Thread

? 1.4. Kojim mehanizmom u Javi su ostvareni ti sudionici? Koje metode koriste koji sudionici?

Koriste metodu run()

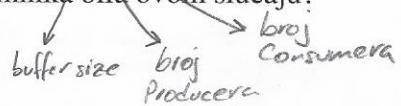
1.5. Pokrenite aplikaciju ograničenog međuspremnika. Koje svojstvo je narušeno izvođenjem ovog programa koristeći argumente navedene u konfiguracijskoj datoteci (2,4,1)? Što se dogodilo s pojedinim sudionicima? Kolika je bila veličina međuspremnika bila ovom slučaju?

Narušeno je svojstvo NotDeadlock Property.

Svi sudionici su zagnavili u jednom stanju

(status: WAITING, priority:5, isDaemon: false, lockCount:1, suspendCount:0)

Veličina spremnika je bila 2



1.6. Pokrenite istu aplikaciju, samo s argumentima (4,1,1). Kakva je sad situacija?

(Napomena: NetBeans će vas možda gnjaviti da ne može spremiti izmjene u datoteci \*.jpf jer da je datoteka otvorena vjerojatno samo za čitanje. Obično spremanje promjena ipak uspješno prođe nakon što prođe neko vrijeme (manje od minute), no u slučaju da ne prođe, možete napraviti Save As... i pohraniti datoteku pod drugim imenom)

Nema pogresaka

1.7. Opišite ukratko što rade i nad čime se pokreću Javine metode wait() i notify().

Pozivom `wait()` prestaje se biti vlasnikom monitora i čeka se dok neka druga dretva ne pozove `notify()`

Pozivom `notify()` odabire se sljedjuća jedna dretva koja je čekala i koja će dalje raditi za izvođenje kritičnog odsječka kod trenutka dretva `varisi` s njim.

Metode se pokreću nad trenutnim dretvom

1.8. Obrazložite ukratko zašto dolazi do narušavanja svojstva u ovom primjeru.

Sve dretve čekaju na objekt `Bounded Buffer` @ 163

1.9. Proučite primjer Rand. java i pridruženu konfiguracijsku datoteku Rand. jpf. Što specificira konfiguracijska naredba `cg.enumerate_random = true` i zašto je ona bitna za ovaj problem?

Konfiguracijska naredba `cg.enumerate_random = true` specificira da se primjer Randjava testira s više mogućih kombinacija elemenata redom, a ne samo jednim.  
Kada se testirajuje izvodi samo s jednom od mogućih kombinacija, moguće je da nikad ne nultima ni dijeliće s nulom.

1.10. Pokrenite aplikaciju za verifikaciju. Koje svojstvo je ovdje narušeno? Što je programer ovog ili ovome sličnog koda previdio? Kako se mogao unaprijed osigurati da se cijeli sustav ne sruši? Koje su konkretne vrijednosti varijabli `a` i `b` srušile program?

No Uncaught Exceptions Property, division by zero

$a=0 \text{ i } b=2$ , a mogu ga srušiti i  $a=1 \text{ i } b=1$  te  $a=2 \text{ i } b=0$

1.11. Sad izbrišite specifikaciju `cg.enumerate_random = true` iz konfiguracijske datoteke te pokrenite aplikaciju. Što se sad dogodilo? Objasnite.

Nema pogreske, provjeren je samo 1 slučaj:  $a=0 \text{ i } b=1$

## 2. dio - Provjera modela u projektu **jpf-core** korištenjem dodatnih slušača

**2.1.** Proučite primjer Racer.java i konfiguracijsku datoteku Racer.jpf. Napravite kopiju konfiguracijske datoteke koju ćete nazvati Racer\_2.jpf i u kojoj ćete izbrisati liniju `listener=gov.nasa.jpf.listener.PreciseRaceDetector`. Pokrenite aplikaciju za verifikaciju putem Racer.jpf i potom putem Racer\_2.jpf. Koje svojstvo je narušeno u slučaju Racer\_2.jpf, a što piše pod error 1 u slučaju Racer.jpf?

Racer\_2.jpf - Division by zero

Racer.jpf - race for field Racer@15b.d

**2.2.** Opišite zašto može doći do problema prilikom izvođenja ovog primjera. Može li instanca dretve t pristupiti liniji koda `int c = 420 / racer.d;`?

Ne može jer se evog nje i dogodila utika izmedu dretvi.  
Liniji se može pristupiti samo za d=0, prije utike

**2.3.** Otvorite kod razreda gov.nasa.jpf.listener.PreciseRaceDetector. Ukratko pojasnite (na temelju komentara razreda) koja je ideja kod implementacije detektora utrke za resursom. Također navedite koji Adapter nasljeđuje ovaj slušač i koje metode nadjačava.

Potpostavlja da sva dva dretva može odabrati istu koja će uveljaviti  
utika te prolazi kroz sve moguće izvore dretvi i provjerava da li  
više od jedne dretve istovremeno koriste iste resurse

**2.4.** Proučite kod primjera NumericValueChecker.java i konfiguracijsku datoteku NumericValueChecker.jpf. Zatim pokrenite aplikaciju. Koju grešku je dojavio JPF?

local variable sumVariable out of range: 12345,00

**2.5.** Primijetite na koji način je specificirano u konfiguracijskoj datoteci na koju varijablu i na koji način se odnosi provjeravanje raspona numeričkih vrijednosti. Pogledajte sad kod odgovarajućeg slušača koji implementira provjeravanje raspona vrijednosti. Koje su dvije mogućnosti rada tog slušača (što se može provjeriti)? Navedite i sintaksu tih provjera.

Mož provjeriti da li je varijable izvra dopuštenog raspona  
Odnosno ta li je manja od minimum ili veća od maksimuma  
range. vars = <variable\_name>  
range. <variable\_name>. var = <class\_name> <method\_name>: <var\_name>  
range. <variable\_name>. min = minimum  
range. <variable\_name>. max = maximum

**2.6.** Proučite kod primjera TestExample.java i konfiguracijsku datoteku TestExample-coverage.jpf. Korištenjem slušača CoverageAnalyzer omogućena je analiza pokrivanja koda. Pokrenite aplikaciju i promotrite resultantnu tablicu koju je ispisao CoverageAnalyzer. Koji razred je bio bolje pokriven s ispitnim primjerima u metodi main? Koje sučelje je morao implementirati ovaj slušač kako bi izmjenio izgled ispisa izvještaja?

Razred T1 9/11 linija  
.listener. CoverageAnalyzer ??

**2.7.** S obzirom na rezultate ispisa i dani kod, koja bi to bila metoda <init>() koja piše u tablici?

Konstruktor (Linije 19-40)

**2.8.** Dodajte u konfiguracijsku datoteku TestExample-coverage.jpf pod razrede koje treba uključiti za provjeru dodatno i sam razred TestExample, uz postojeće razrede T1 i T2. Spremite datoteku i pokrenite aplikaciju za verifikaciju. Proučite rezultat. Iz samog koda, očito je da će se proći kroz sve linije metode main. Koje naredbe (linije koda) analizator pokrivanja preskače kad izyeštava da je prošao kroz samo 3/8 linija koda metode main (osma linija koda uključuje implicitni return;)? Kojom specifikacijom bi isključili ispisivanje provjeravanja pokrivenosti grana u izlaznoj tablici?

Predaje sve linije koda assert (služe za testiranje programa  
coverage. show\_branches = false

### **3. dio - Provjera modela u projektu jpf-aprop**

Instalirajte projekt (paket) jpf-aprop koji služi za provjeru specifičnih svojstava programa pisanih u Javi koja su zadana u kodu u obliku anotacija (oznaka @). Za instalaciju odaberite redom "Team" -> "Remote" -> "Clone other..." i upišite pod Repository URL:

<http://babelfish.arc.nasa.gov/hg/jpf/jpf-aprop>

Po završetku kloniranja, nakon što se projekt jpf-aprop našao u otvorenim projektima, dodajte u datoteku site.properties, koju ste ranije stvorili u postupku instalacije, sljedeće retke na kraj datoteke i pohranite promjene.

```
# annotation properties extension
jpf-aprop = ${jpf.home}/jpf-aprop
extensions+=, ${jpf-aprop}
```

Prvo što možete zamijetiti sa skinutim paketom je to da postoje uskličnici uz dosta razreda koda paketa jpf-aprop. Prvo što trebate napraviti je da otvorite skriptu build.xml od projekta jpf-aprop i da potražite redak property name="src\_level" koji postavite na vrijednost 8 (umjesto dosadašnjih 6). Zatim desnim klikom na projekt jpf-aprop pa redom "Properties" -> "Java sources" -> "Source Level" postavite na JDK 1.8. Sada pokrenite skriptu build.xml (desni klik pa "Run Target" -> "build").

Skripta bi trebala vratiti 62 pogreške i neuspjelu izgradnju. Razlog tome je taj što se jpf-aprop paket nije mijenjao u zadnjih nekoliko godina u odnosu na jpf-core koji je više-manje stalno evoluirao. Ipak, razlika nije toliko drastična da se ne bi mogla uglavnom popraviti pažljivim izmjenama u kodu.

Promotrite pogreške koje su vam dojavljene prilikom izgradnje. Većina pogrešaka vezana je uz import razreda InvokeInstruction, ReturnInstruction, FieldInstruction i InstanceFieldInstruction i to iz paketa gov.nasa.jpf.jvm.bytecode.

Kako bi to riješili, u svim razredima za koje je dojavljen taj problem potrebno je ručno napraviti izmjene (najlakše uz pomoć CTRL+H, pri čemu treba zamijeniti sva pojavljivanja (Replace All) razreda InvokeInstruction s JVMInvokeInstruction, ReturnInstruction s JVMReturnInstruction, FieldInstruction s JVMFieldInstruction te InstanceFieldInstruction s JVMInstanceFieldInstruction iz istoga paketa (uvjerite se da ti razredi zaista postoje u jpf-coreu). Možete povremeno pokrenuti nanovo build.xml da vidite kako se broj prijavljenih pogrešaka smanjuje kako radite ispravke.

Na kraju bi trebale ostati tri pogreške, dvije vezane uz razred ArrayInstruction i jedna vezana uz razred JVMInvokeInstruction. Prve dvije ćete ispraviti tako što ćete preimenovati razred ArrayInstruction u ARRAYLENGTH is istog paketa. Treću pogrešku (poziv metode hasAttrRefArgument) vjerojatno nećete moći ispraviti. Potrebno je zakomentirati (uz pomoć /\* ... \*/) dio metode executeInstruction u razredu koji prijavljuje tu pogrešku i to od linije koja počinje s "if(call.hasAttrRefArgument ..." pa do uključivo pretpretzadnje vitičaste zagrade. Sad bi build.xml trebao uspješno proći izgradnju projekta i svi preostali uskličnici (pogreške) bi trebali nestati.

3.1. Proučite datoteku jpf.properties projekta jpf-aprop. Koja se standardna svojstva provjeravaju prilikom pretrage kad se koristi projekt jpf-aprop? Gdje je to uopće definirano? Navedite put do direktorija s razredima koji se kao primjeri provjeravaju uz pomoć jpf-aprop. Koje svojstvo to pokazuje u datoteci jpf.properties?

Standardna svojstva označena su anotacijom @ te su odgovarajući razredi uključeni u knjižnicu jpf-aprop-annotations.jar u kojem se nalazi popis razreda s anotacijama. Ova nisu eksplicitno navedena u properties datoteci.

PUT DO PRIMJERATA src/examples

svojstvo jpf-aprop.sourcepath

3.2. Proučite jednostavni primjer ConstViolation.java i pridruženu konfiguracijsku datoteku ConstViolation.jpf te odgovarajućeg slušača. Opišite što se događa u kodu razreda ConstViolation.java. Koja metoda je označena s @Const anotacijom i što to točno znači?

Varijabla se postavlja nakon poziva metode dontDoThis()

Metode označene s @Const anotacijom ne smiju mijenjati vrijednosti polja svoje instance ili klase

3.3. Pokrenite tu aplikaciju za verifikaciju. Koja vrsta pogreške je dojavljena?

AssertionError

3.4. Proučite primjer ContractViolation.java i pridruženu konfiguracijsku datoteku ContractViolation.jpf. Navedite koje sve slušače koristi ovaj program (puna kvalificirajuća imena).

aprop.Listener.NonNullChecker

aprop.Listener.LockChecker

aprop.Listener.ConstChecker

aprop.Listener.NonSharedChecker

aprop.Listener.ContractVerifier@9bc

3.5. Pronadite u strukturi projekta odgovarajućeg slušača u kojem se provjerava svojstvo nonshared.throw\_on\_cycle. U kodu pronadite i napišite koju bi iznimku bacio JPF ako bi dretva bila uhvaćena u ciklusu nad objektom koji nije predviđen za višedretvenost.

NonSharedChecker.java

if (throwOnCycle) {  
 setNextPC (ti.createAndThrowException ("java.lang.AssertionError"))  
}

**3.6.** Što znače i čemu služe anotacije @Requires, @Invariant i @Ensures u kodu programa ContractViolation.java?

One prosljedjuju preduvjete, nepromjenjeno stavlje i postavljaju slušać u contractVerifier

**3.7.** Pokrenite aplikaciju za verifikaciju. Koji dio ugovora je narušen? Napišite pogrešku koja je dojavljena.

java.lang.AssertionError: postcondition violated: "(result >= 0) AND (result > 0)", values  
{result = 0}

**3.8.** Pažljivo proučite dojavljenu pogrešku. Metoda kojeg točno razreda je izazvala narušavanje ugovora?

Razreda ContractViolationBase jer je njegov uvjet bio da result  
mora biti strogo veći od 0

**3.9.** Promijenite ugovore dviju metoda tako da obje ispituju uvjet ( $\text{Result} \geq 0$ ). Spremite izmijenjenu datoteku ContractViolation.java. Pokrenite ponovno skriptu build.xml. Kad se izmijenjeni primjer preveo, ponovno ga pokrenite. Kakva je sad situacija?

Nema pogreške

#### 4. dio - Provjera modela vlastitog projekta

**4.1.** U NetBeansu napravite novi projekt ("File" -> "New Project" -> "Java Application") koji ćete nazvati JavaFV. Napravite ga bez razreda JavaFV s metodom main. Zatim desnim klikom na Source Packages napravite novi paket pod nazivom fv, a onda desnim klikom na paket fv napravite novi razred pod imenom Verifikacija.java i statičkom metodom main (unutar razreda napišite public static void main(String[] args) {}). Isto tako, izbrišite redak package fv; iz napravljene datoteke Verifikacija.java.

Napravite unutar istoga paketa novu datoteku (desni klik na fv, pa "New" -> "Other..." -> "Other" -> "Empty File" i nazovite ga Verifikacija.jpf. U tu datoteku dodajte zasad samo jedan redak kojim ćete omogućiti pokretanje razreda Verifikacija.java i spremite ju. Kako izgleda taj redak?

target = Verifikacija

U korijenskom direktoriju projekta JavaFV zatim napravite datoteku jpf.properties jednostavnog sadržaja:

```
JavaFV = ${config_path}

JavaFV.classpath=\
${JavaFV}/build/classes

JavaFV.sourcepath=\
${JavaFV}/src/fv
```

Na kartici "Projects" kliknite desnim klikom na vaš projekt JavaFV i odaberite "Clean and build".

Nakon što se projekt izgradio, provjerite da se međukod Verifikacija.class nalazi pod direktorijem build/classes. Probajte pokrenuti verifikaciju koja bi trebala proći bez pogrešaka.

Objasnite zašto je redak JavaFV.classpath=\\\${JavaFV}/build/classes

nužno navesti u datoteci jpf.properties?

Kako bi JPF znao gdje se nalaze ciljane (targeted) klase

**4.2.** Sada izmjenite sadržaj datoteke Verifikacija.java tako da sadrži kod koji se nalazi u istoimenoj datoteci koja se nalazi u repozitoriju kolegija FMUOS (pod 2. DZ). Također, izmjenit ćete sadržaj datoteke Verifikacija.jpf tako da sadrži specifikacije prema istoimenoj datoteci koja se nalazi u repozitoriju kolegija. Nakon kopiranja koda i specifikacija spremite datoteku, no nećete moći prevesti datoteku Verifikacija.java budući da sadrži importe koji su nepoznati samom projektu JavaFV.

**4.3.** Potrebno je uključiti izgrađene knjižnice (.jar) od jpf-core i jpf-aprop u projekt JavaFV kako bi se kod razreda Verifikacija.java mogao prevesti. To se radi tako da odaberete JavaFV pa desni klik, a zatim "Properties" -> "Libraries" -> "Add JAR/Folder". Pronadite u direktoriju jpf-core -> build -> jpf.jar, jpf-classes.jar i jpf-annotations.jar te ih dodajte. Ostali JAR-ovi nisu bitni. Od projekta jpf-aprop potrebno je dodati samo JAR jpf-aprop-annotations.jar. Odaberite "Ok". Pogreške bi sada trebale nestati. Sad prevedite Verifikacija.java (desni klik -> "Compile File").

**4.4.** Pokrenite aplikaciju za verifikaciju. Koja pogreška vam je dojavljena? Objasnite zašto je došlo do te pogreške s obzirom na konfiguracijsku datoteku i zadani kod.

java.lang.AssertionError : null assignment to @NotNull instance field  
Verifikacija.id  
Verifikacija.java , linija 10: @NotNull private String id;  
linija 23: Verifikacija v = new Verifikacija ("a");  
24 : v.setId(null); ← uzrokovalo pogresku

**4.5.** U konfiguracijsku datoteku dodajte ovaj redak na kraj:

```
search.multiple_errors = true
```

Ovime se prolazi svim putevima izvođenja kroz program i dojavljuje se za svaki put izvođenja prva pogreška na koju se naletilo. Pokrenite aplikaciju za verifikaciju. Koja je razlika između prethodnog ispisa pogrešaka i sadašnjega?

*Osigurali smo više struka u pisivanju grešaka. Ne ćemo učiniti  
ispisivanja prve greške.*

**4.6.** Uklonite redak `search.multiple_errors = true` te zakomentirajte redak u kodu koji smatrate odgovornim za dojavu ove vrste pogreške. Prevedite kod i pokrenite ponovno aplikaciju za verifikaciju. Koja se sada pogreška pojavila, na kojem retku koda i zašto je prijavljena?

*java.lang.AssertionError : write of const instance field outside constructor  
linija: 31 jer je goće DNA bilo mijenjano a na liniji 11  
je postavljena anotacija @Const private String DNA;*

**4.7.** Zakomentirajte redak u kodu koji smatrate odgovornim za dojavu ove vrste pogreške. Prevedite kod i pokrenite ponovno aplikaciju za verifikaciju. Koja je sada pogreška dojavljena? Objasnite zašto se ova pogreška dogodila.

*java.lang.AssertionError  
assert boolean-expression: string-expression == false - Assertion Error  
Nije prošlo jer je u Verifikacija.jpf postavljena velocity\_low=20  
i velocity\_high=100 (100=100), a uvjet je shoga manje*

**4.8.** Izmijenite naredbu (`assert`) u kodu tako da više ne dolazi do ove vrste pogreške. Prevedite kod i provjerite da se pogreška zaista više ne događa. Koji ste broj trebali navesti kao uvjet u naredbi (`assert`) da ne dođe do pogreške?

*101 ili <=100*

**4.9.** Provjerite metode razreda `gov.nasa.jpf.vm.Verify`. Postoje li metode `getDouble()` i `getInt()` bez argumenata? Objasnite. Objasnite na primjeru što su to generatori izbora i koja je namjena navođenja heuristika pri korištenju generatora izbora. Koja se heuristika koristila u primjeru `Verifikacija.java`?

*Moguće im je poslati prazan argument, što rezultira vraćanjem nasmješne vrijednosti*

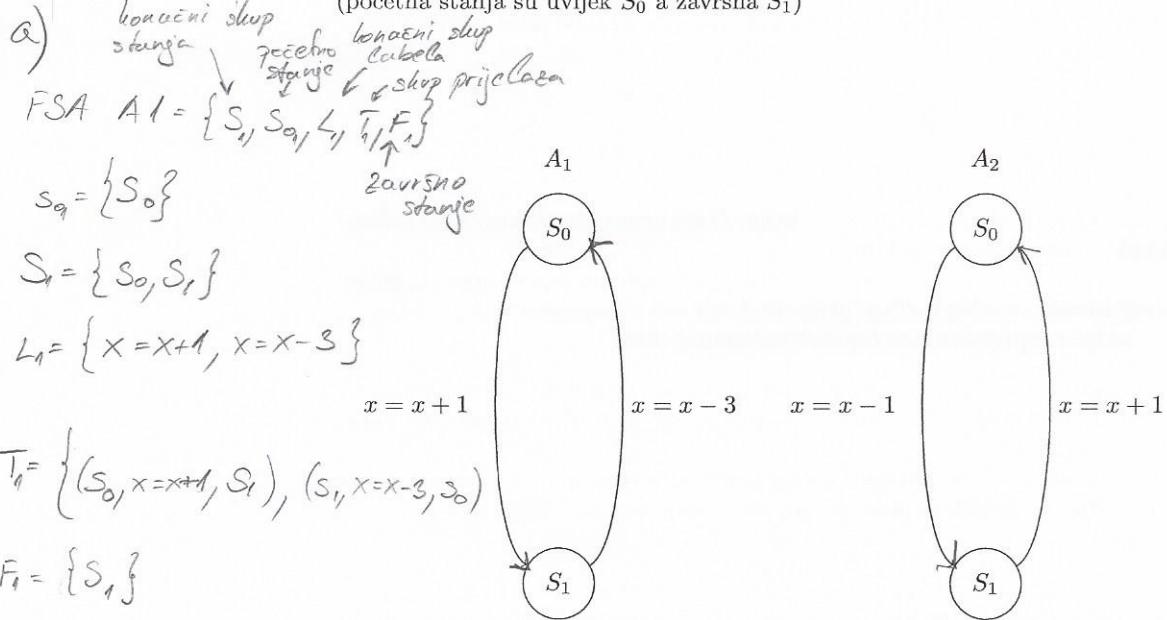
*ChoiceGenerator je mehanizam JPF-a da sustavno istražuje prostor stanja kako bi došao do rješenja. Mechanizam odabira odgovarajućeg generatora izbora i parametara. Heuristike odvojenje je od samog programa - navodi se u konfiguraciji, što je dobro, kao svojstvo odgovarajućeg imena zajedno s ostalim svojstvima.*

Formalne metode u oblikovanju sustava  
 Upute za 3. domaću zadaću  
 (8. 9 i 10. predavanje)

Kolokvij se piše u ponedjeljak 1. lipnja 2015 godine u 18:00 sati prema rasporedu koji će biti naknadno objavljen.

Ime i prezime: \_\_\_\_\_ Potpis: \_\_\_\_\_

Zadana su dva konačna diskretna automata  $A_1$  i  $A_2$  prema slici:  
 (početna stanja su uvijek  $S_0$  a završna  $S_1$ )



Slika 1: FSA  $A_1$  i  $A_2$

U okviru domaće zadaće potrebno je:

- a) Detaljno opisati strukturu automata  $A_1$  i  $A_2$  prema definiciji FSA  $A = (S, s_0, L, T, F)$  (odrediti elemente svakog od skupova  $S, s_0, L, \dots$ )
- b) Odrediti asinkroni produkt automata  $A_1$  i  $A_2$  i nacrtati ga.
- c) Odrediti ekspandirani asinkroni produkt za prvih 5–10 po volji odabralih članova i nacrtati ga.
- d) Pomoću ekspandiranog produkta odrediti istinitost LTL formule  $\Diamond \Box p$  ako je  $p \equiv x \leq 0$ . Obrazložiti rješenje, posebice komentirati mogućnost rješavanja bez primjene programskih alata.
- e) Pomoću ekspandiranog produkta odrediti istinitost LTL formule  $\Diamond p$  ako je  $p \equiv x < 0$ . Obrazložiti rješenje, posebice komentirati mogućnost rješavanja bez primjene programskih alata.
- f) Nacrtati moguću realizaciju Büchi automata za LTL formulu:  $\Diamond p \longrightarrow \Diamond q$ .

- 1) Instalirajte programski alat Spin (<http://spinroot.com/spin/Bin/index.html>). Instalacije se svodi na kopiranje izvršnog programa. Za one koji hoće više, sve instrukcije jezika Promela možete pronaći na <http://spinroot.com/spin/Man/promela.html> kao i službene upute (“manual”) na <http://spinroot.com/spin/Man/Manual.html>.
- 2) Editirajte automate  $A_1$  i  $A_2$  kao promela procese A i B (vidjeti predložak na zadnjoj strani). Napomena: Promela file nazvati prezime.prm (npr. `blaskovic.prm`). Polazeći od zadanog Promela modela koji se sastoji od dva procesa A i B analizirati će se LTL formula  $\Diamond p$  gdje je  $p \equiv (x <= 0)$ .
- 3) Pokrenite simulaciju: `spin -u20 -p -c -g prezime.prm`. Prepišite prvih 12 članova. Pismeno obrazložite istovjetnosti i razlike između ekspandiranog asinkronog produkta iz domaće zadaće i rezultata simulacije.
- 4) Generirajte analizator: `spin -a prezime.prm`.
- 5) Prevedite u izvršni oblik npr.: `gcc -o pan pan.c`.
- 6) Pozovite analizator: `pan -a ili ./pan -e`. Pismeno obrazložite da li je uvjet  $p$  zadovoljen ?

- 7) Pokrenite "error trail" opciju (pronalaženje protuprimjera) sa spin -t -p -c -g prezime.prm.  
Da li postoji sekvenca u kojoj varijabla  $x$  na kraju poprima vrijednost  $x \leq 0$ ?  
Koliko koraka (eng. "steps") sadrži ?
- 8) Prepišite instrukcije za Büchi automat koje generira Spin spin -f ' $\neg \square q$ '.  
Nacrtajte pripadni Büchi automat.
- 9) Prepišite instrukcije za Büchi automat koje generira Spin spin -f ' $\neg \square \square q$ '.  
Nacrtajte pripadni Büchi automat.
- 10) Na isti način koristeći Spin nacrtajte automat iz Vaše domaće zadaće (pitanje f)).
- 11) Generirajte analizator sa spin -a -o3 prezime.prm, prevedite te pozovite analizator sa pan -d.  
Precrtajte tako dobivene FSA. Objasnite razlike kao i istovjetnosti prema automatima iz domaće zadaće ? Usporedite stanja prema slici na stranici 1 i stanja dobivena s opcijom pan -d. U čemu je razlika ?

Odgovorite na sljedeća pitanja:

Ponovite postupak za  $\phi \square p$  (modificirati "never claim" spin -f ' $\neg \square \square p$ ' na kraju prezime.prm datoteke).

- p1) Da li postoji sekvenca u kojoj varijabla  $x$  na kraju poprima vrijednost  $x \leq 0$ ?
- p2) Koliko koraka (eng. "steps") sadrži ?
- p3) Da li je moguće problem riješiti bez LTL formule samo pomoću assert naredbi ? Obrazložite odgovor.
- p4) Da li je moguće problem riješiti bez LTL formule samo pomoću simulacije ?  
Obrazložite odgovor.

Zadan je Promela model komunikacijskog protokola koji opisuje dio moguće realizacije protokola za preuzimanje datoteka (eng. "download"). Ako je  $N = 8$ , pomoću programskog alata Spin odredite:

- a) stanja iz kojih nema napretka (eng. "deadlock"),
- b) dolazi li protokol u završno stanje ?

Ako je potrebno odredite protuprimjere.

\*Obrazložite da li je potrebno uvoditi dodatne instrukcije pomoću *LTL* formule koje će omogućiti provjeru postojanja stanja iz kojih nema napretka (eng. "deadlock") ? (nije obavezno pitanje)

---

```

1
2
3 mtype = { ini, ack, dreq, data, shutup, quiet, dead };
4
5 #define N
6
7 chan M = [N] of { mtype };
8 chan W = [N] of { mtype };
9
10 active proctype Mproc()
11 {
12     W!ini;                      /* connection      */
13     M?ack;                      /* handshake       */
14
15     timeout ->                  /* wait           */
16     if                         /* two options:   */
17     :: W!shutup                /* start shutdown */
18     :: W!dreq;                  /* or request data */
19         M?data ->             /* receive data   */
20         do
21             :: W!data            /* send data      */
22             :: W!shutup;        /* or shutdown    */
23             break
24         od
25     fi;
26
27     M?shutup;                  /* shutdown handshake */
28     W!quiet;
29     M?dead
30 }
31
32 active proctype Wproc()
```

```
33  {
34      W?ini;          /* wait for ini      */
35      M!ack;          /* acknowledge      */
36
37      do             /* 3 options:        */
38          :: W?dreq ->    /* data requested   */
39              M!data          /* send data        */
40          :: W?data ->    /* receive data     */
41      #if 1
42          M!data
43      #else
44          skip           /* no response      */
45      #endif
46      :: W?shutup ->
47          M!shutup;       /* start shutdown   */
48          break
49      od;
50
51      W?quiet;
52      M!dead
53  }
```

---

Predložak *Promela* programa za laboratorijsku vježbu ( $N = 50$ ):

---

```

1 #define N 50
2
3 #define p (X<=0)
4
5 int x = N;
6
7 active proctype A()
8 {
9 do
10      X=X+1; X=X-3
11 od;
12 }
13
14 active proctype B()
15 {
16 do
17      X=X-1; X=X+1
18 od;
19 }
20
21 never { /* LTL formula */
22 TO_init:
23
24 do
25    :: atomic {((p))} -> assert (!((p)))
26
27    od; :: () -> goto TO_init
28 accept-all:
29
30 skip
31
32 }
33

```

---

\*

Rješenja za ovaj zadatak uz detaljna obrazloženja možete priložiti uz kolokvij (nije obavezno).

(FWGC -Farmer, Wolf, Goat and Cabagge problem poznat iz AI)

Farmer mora prevesti čamcem preko rijeke vuka (wolf), kozu (goat) i zelje (cabbage). U čamcu su samo dva mjesta: za farmera koji uvijek mora biti u čamcu te za zelje ili jednu od životinja. Pri tome nikada ne smiju zajedno ostati bez prisustva farmera vuk i koza odnosno koza i zelje.

Vaš je zadatak definirati Promela model sa rasporedom (sekvencom ili slijedom akcija) kojim će farmer prevesti vuka, kozu i zelje preko rijeke.

Kod rješavanja kao varijablu za trenutni položaj možete koristiti polje bitova  $Pl[i]$ , gdje su  $i$  farmer, vuk, zelje odnosno koza, ako su na polaznoj strani,  $Pl[i]$  je neistinit, a na odredišnoj strani je  $Pl[i]$  je istinit.

Kod rješavanja možete po slobodnom izboru koristiti model sa ili bez protupri-mjera.



1. b)

$$A_1 \times A_2 = A$$

Product

$$A \cdot S = A_1 \cdot S \times \dots \times A_n \cdot S$$

$$A \cdot S = \{(S_0, S_0), (S_0, S_1), (S_1, S_0), (S_1, S_1)\}$$

$$A \cdot S_0 = A_1 \cdot S_0 \times \dots \times A_n \cdot S_0$$

$$A \cdot S_0 = \{(S_0, S_0)\}$$

$$A \cdot L = A_1 \cdot L \times \dots \times A_n \cdot L$$

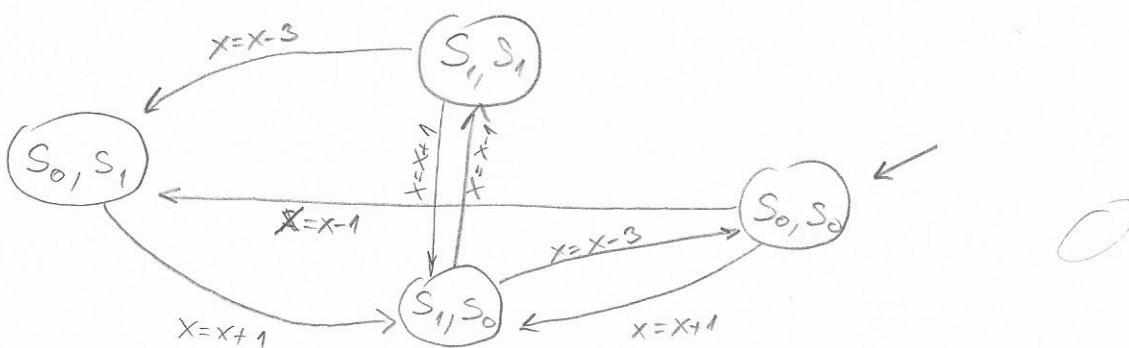
$$A \cdot L = \{(x=x+1, x=x-1), (x=x+1, x=x+1), \\ (x=x-3, x=x-1), (x=x+3, x=x+1)\}$$

$$A \cdot F = A_1 \cdot F \times \dots \times A_n \cdot F$$

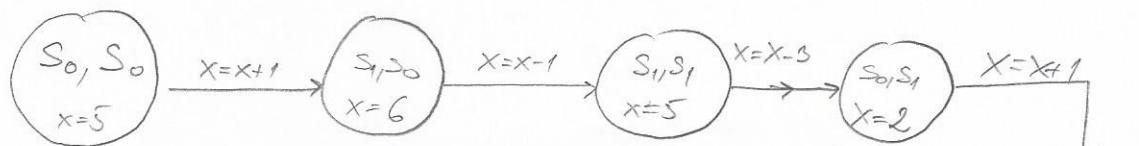
$$A \cdot T = \{[(S_0, S_0), x=x+1, (S_1, S_0)], [(S_0, S_0), x=x-1, (S_0, S_1)],$$

$$[(S_0, S_1), x=x+1, (S_1, S_0)], [(S_1, S_0), x=x-3, (S_0, S_0)], [(S_1, S_0), x=x-1, (S_1, S_1)],$$

$$[(S_1, S_1), x=x-3, (S_0, S_1)], [(S_1, S_1), x=x+1, (S_1, S_0)]\}$$



c)  $x=3$

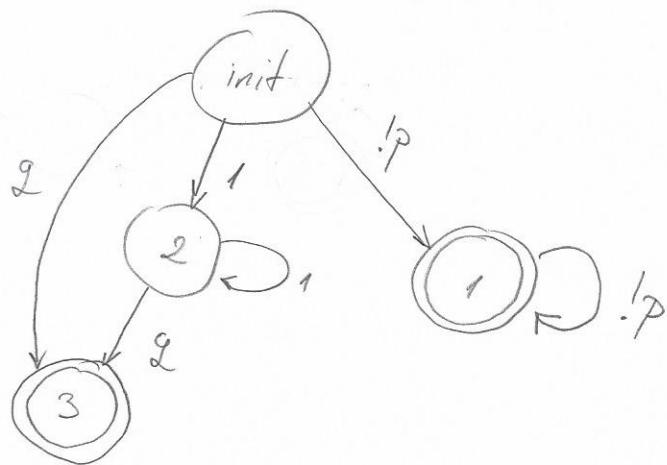


d)  $\Diamond \Box p$ ,  $p \equiv x \leq 0$  ne vrijedi;

$(S_1, S_0)$  i  $(S_0, S_1)$  mogu doći u beskonačnoj ... posljednji prije nego dođe do slučaja da je  $x \leq 0$

e)  $\Diamond P$ ,  $P \equiv x < 0$  istinit je (iz primjera  $\rightarrow$  smo mogli nastaviti putem  $x=x-1$ , tada bi  $x$  postao -1)

f)  $\Diamond P \rightarrow \Diamond Q$



(2.) 3)

1: proc O(A) goes to state 3 [x = (x+1)]  
 $x = 51$

2: - " - . prm: 10 (state 2) [x = (x-3)]  
 $x = 48$

3: - " - . prm: 12 (state 4) [. (goto)]

4: - " - . prm: 9 (state 3) [x = (x+1)]  
 $x = 49$

5: - " - . prm: 10 (state 2) [x = (x-3)]  
 $x = 46$

6: - " - . prm: 12 (state 4) [. (goto)]

7: - " - . prm: 9 (state 3) [x = (x+1)]  
 $x = 47$

8: - " - . prm: 10 (state 2) [x = (x-3)]  
 $x = 44$

9: - " - . prm: 12 (state 4) [. (goto)]

10: - " - . prm: 9 (state 3) [x = (x+1)]  
 $x = 45$

11: - " - . prm: 10 (state 2) [x = (x-3)]  
 $x = 42$

12: - " - . prm: 12 (state 4) [. (goto)]

① spin -p -c -g -l -u72 model.prml

- l ispisuje sve lokalne varijable
- u72 zavrsjava simulaciju nakon 72 koraka
- P ispisuje sve stanja
- c ispisuje -s i -r a stupcima
- g ispisuje globalne varijable
- t error trail

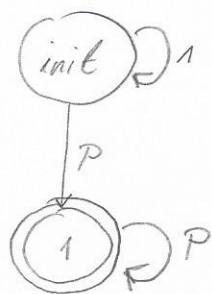
2) Dolazi u savršeno stanje,  $a=1$

3)  $\square \diamond (x > 5)$  mora postojati stanje u kojem je  $x > 5$

$\diamond \square (x > 5)$  postoji stanje u kojem vrijedi  $x > 5$  te nabor ujega uvijek vrijedi  $x > 5$

nije ishinito (nigdje nije  $x > 5$ )

4)  $\square \square p$



never {

T0\_init:

if

:: (1) → goto T0\_init

:: (p) → goto accept-S2

fi;

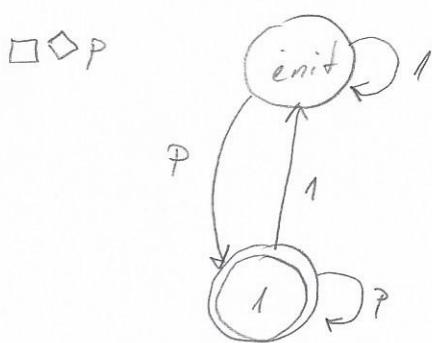
accept-S2:

if:

:: (p) → goto accept-S2

fi;

}



never {

T0-init:

if

:: (p) → goto accept-S1

:: (1) → goto T0-init

fi;

accept-S1

if

:: (p) → goto accept-S1

:: (1) → goto T0-init

fi;

}

5)

intype = {req, done, alarm, ack};

# define N

chan U = [N] of {intype};

chan S = [N] of {intype};

active procedure User()

{

RDY:

if

:: (!req)  $\rightarrow$  goto WAIT

:: (?alarm)  $\rightarrow$  goto RCQS

fi;

WAIT:

if

:: (?done)  $\rightarrow$  goto RDY

**Ime i prezime:** \_\_\_\_\_

**JMBAG:** \_\_\_\_\_

**Provjera znanja iz 1. domaće zadaće**

1. (1 bod) Opišite namjenu ulaznog jezika u sustavu NuSMV. Koja struktura se koristi za opis relacije prijelaza? Od kojih se bitnih dijelova sastoje NuSMV programi?

2. (1 bod) Prilikom dodjeljivanja vrijednosti (blok ASSIGN) u jeziku NuSMV, što se sve može naći na **lijevoj** strani izraza? Navedite primjere.

3. (1 bod) Objasnite i na nekom primjeru pokažite na koja dva načina se u NuSMV-u može modelirati nedeterminizam.
4. (1 bod) Objasnite sinkroni i asinkroni način rada modula u sustavu NuSMV. Kako se označava modul koji se treba izvršavati asinkrono?
5. (2 boda) Napišite CTL specifikacije u jeziku NuSMV na temelju primjera iz domaće zadaće koje odgovaraju rečenicama prirodnoga jezika te odgovorite na pitanje.
- A. *Dva procesa ne mogu biti istovremeno u kritičnom odsječku.* (napisati specifikaciju da nema nepoželjnog ponašanja)
  - B. *Ako proces proc0 uđe u kritični odsječak, proc0 neće ponovo ući u kritični odsječak sve dok proc1 nije izašao iz svog kritičnog odsječka.*
- Koje svojstvo se provjerava specifikacijom A? Koji je problem kod protokola *mutex* prisutan ako se zahtjeva da specifikacija B bude istinita?
- Napomena: pretpostavite da su označke stanja: izlazak iz kritičnog odsječka: "e"; pokušaja ulaska u kritični odsječak: "t"; ulazak/boravak u kritičnom odsječku: "c".

6. (2 boda) Zadan je kod primjera mutex\_4ex.smv:

```
MODULE main
VAR
    proc0 : process user(proc1.flag);
    proc1 : process user(proc0.flag);

MODULE user(oflag)
VAR
    flag : boolean;
    state : {setflag, reading, critical, exiting, noncritical};
ASSIGN
    init(state) := {setflag};
    init(flag) := FALSE;
    next(state) :=
        case
            state = setflag : reading;
            state = reading & !oflag : critical;
            state = critical : {critical, exiting};
            state = exiting : noncritical;
            state = noncritical : {noncritical, setflag};
            TRUE : state;
        esac;
    next(flag) :=
        case
            state = setflag : TRUE;
            state = exiting : FALSE;
            TRUE : flag;
        esac;
```

Za zadani primjer:

- A. (0.5 boda) Specificirajte svojstvo životnosti CTL formulom i to za oba procesa.
- B. (1 bod) Jasno objasnite riječima koji je problem sa životnosti u ovom primjeru.
- C. (0.5 boda) Nakon što ste objasnili problem, specificirajte CTL formulu kojim bi utvrdili da taj problem postoji.

7. (1 bod) Pretpostavite da imate 10 instanci istog modula pod nazivom proc (instance proc0-proc9), koji se izvode asinkrono. Navedite ograničenje **pravednosti**: instanca modula proc1 mora se izvoditi beskonačno često.

8. (1 bod) Objasnite značenje naredbe "COMPUTE MIN [proc0.state = noncritical, proc0.state = exiting]" u jeziku NuSMV. Koji je rezultat izvođenja te naredbe u slučaju da se u implementaciju protokola *mutex* iz domaće zadaće doda ograničenja pravednosti?

**Ime i prezime:** \_\_\_\_\_

**JMBAG:** \_\_\_\_\_

**Provjera znanja iz 2. domaće zadaće**

1. (1 bod) Osnovni projekt Java PathFindera, `jpf-core`, otkriva tzv. nefunkcijska svojstva. Kakva su to svojstva s obzirom na ispoljavanje pogrešaka programa? Navedite dva primjera nefunkcijskih svojstava koje otkriva `jpf-core` po *defaultu*.

2. (1 bod) Navedite što sve Java PathFinder pamti u svakom stanju svojeg vlastitog virtualnog stroja, kao alat za provjeru modela s eksplicitnim pamćenjem stanja izvođenja programa.

7. (1 bod) Koji mehanizam koristi JPF za smanjenje prostora stanja prilikom poziva metode:

```
double vel = Verify.getDouble ("velocity");
```

pri čemu je velocity definiran u konfiguracijskoj datoteci kao:

```
velocity.class =
gov.nasa.jpf.vm.choice.DoubleThresholdGenerator
velocity.threshold = 50.0
velocity.low = 10.0
velocity.high = 80.0
```

Što se tim mehanizmom gubi? Koliko točno double vrijednosti DoubleThresholdGenerator provjerava?

8. (1 bod) Koja je razlika pri verifikaciji programa ako se Java PathFinderu specificira svojstvo cg.enumerate\_random=true za program koji sadrži linije koda:

```
Random random = new Random(42);
int b = random.nextInt(3);
```

ili ako mu se dotično svojstvo ne specificira? Objasnite.

Kolokvij iz 3.DZ iz *Formalnih metoda u oblikovanju sustava*

Inačica: 2ip3D8

IME I PREZIME: \_\_\_\_\_ JMBAG: \_\_\_\_\_ Profil: \_\_\_\_\_ 1.6.2015.

- 1) (2 boda) Opisite što rade sljedeće naredbe:

a) chan ch2 = [0] of { byte, bit, int, bit }

---

b) run

---

c) spin -p -c -g -l -u33 model.prml

---

- 2) (2 boda) Odredite: dolazi li proces *FSA* u završno stanje ili ostaje blokiran? Koju vrijednost na kraju poprima varijabla *a*? Obrazložite odgovor! Opisite kako se može osigurati da *Promela* model uvijek dođe do naredbe u 10. redu (do regularnog završetka) i da pri tome uvijek **assert(a==2)** bude istinito?

```

1 active [2] proctype FSA() {
2     byte a=2;
3     do
4         :: (a==2) -> a++;
5         :: (a==3) -> a++;
6         :: else -> goto end_FSA;
7         od;
8     end_FSA:
9         printf("a=%d\n",a); assert(a==2);
10    }

```

---

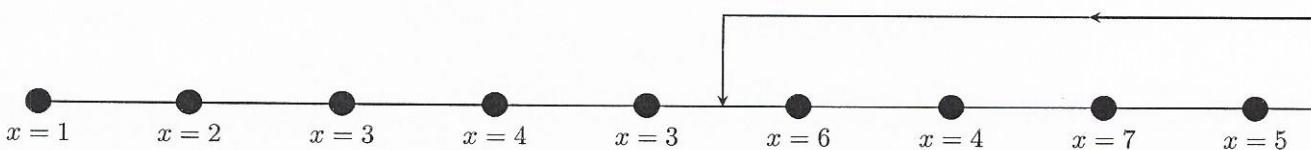


---



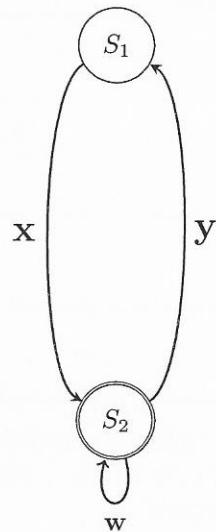
---

- 3) (1 bod) Opisite što provjerava temporalna formula  $\square\Diamond(x > 5)$  te odredite istinitost temporalne formule.  
*Napomena:* promatrati samo sekvencu  $\sigma$  (dio ekspandiranog produkta) prema slici!

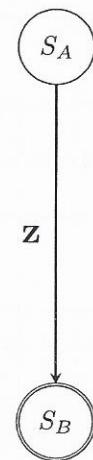


- 4) (1 bod) Skicirajte moguću realizaciju Büchi automata za sljedeću *LTL* formulu:  
 $\Diamond!\Diamond p$

5) (4 boda) Zadana su dva FSA  $A_1$  i  $A_2$  prema slici:



automat  $A_1$



automat  $A_2$

a) (2 boda) Formalno zapišite i nacrtajte asinkroni produkt  $C_{FSA} = A_1 \times A_2 = (C.S, C.s_0, C.L, C.T, C.F)$ .

b) (1 bod) Nastavite zapis ekspandiranog asinkronog produkta za sljedećih šest članova (zapisom globalnih stanja):

$S_1S_A$  , \_\_\_\_\_ , \_\_\_\_\_ , \_\_\_\_\_ , \_\_\_\_\_ , \_\_\_\_\_ , \_\_\_\_\_

c) (1 bod) Obrazložite postoji li sekvenca kojom se  $C_{FSA}$  vraća u početno stanje  $S_1S_A$ !

---

---

---