**1. doma**ć**a zada**ć**a iz Formalnih metoda u oblikovanju sustava 2014./2015.**

**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 kriti*č*nom odsje*č*ku.»*

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) EF (proc0.state=critical & proc1.state=critical)

b) !EF (proc0.state=critical & proc1.state=critical)

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

Obilježje napisano u obliku 1.2.b) je zadovoljeno – to i želimo.

Obilježje napisano u obliku 1.2.a) nije zadovoljeno – to znači da se niti u jednom trenutku ne mogu naći oba procesa u kritičnom odsječku – potvrdili smo obilježje sigurnosti kontradikcijom.

**1.4. Specificiraj i napiši u CTL notaciji obilježje (engl. *liveness property*):**

*«Ako proces pokuša u*ć*i u kriti*č*ni odsje*č*ak, kona*č*no* ć*e i u*ć*i»*

Specifikaciju napiši za oba procesa

AG(proc0.state=entering -> AF(proc0.state=critical))

AG(proc1.state=entering -> AF(proc1.state=critical))

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

Svojstvo nije zadovoljeno. Problem je u tome da proces može ući u kritični odsječak i u njemu ostati beskonačno dugo, a time neće vratiti turn drugom procesu.

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

Dodali smo JUSTICE running.

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

Ono što smo postigli je da se svaki proces treba izvršavati beskonačno često. Problem (ne)životnosti još nismo riješili.

**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.**

Dodali smo JUSTICE !(state=critical). Svojstvo životnosti i dalje nije zadovoljeno. Promatraju se samo oni putevi na kojima proces nije beskonačno u kritičnom odsječku. Međutim, to ne osigurava da će neki proces ući u kritičan odsječak, već samo da neće u njemu ostati beskonačno dugo.

**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.**

Dodavanjem JUSTICE !(state=noncritical) obje specifikacije postaju zadovoljene – važi svojstvo životnosti. Sada smo osigurali da nijedan proces ne može ostati beskonačno dugo niti u kritičnom niti u nekritičnom odsječku.

**1.10. Specificiraj i napiši u CTL notaciji:**

*«Ako proces proc0 u*đ*e u kriti*č*ni odsje*č*ak, proc0 ne*ć*e ponovo u*ć*i u kriti*č*ni odsje*č*ak sve dok proc1 nije prošao kroz svoj kriti*č*ni odsje*č*ak.»*

AG(proc0.state=critical -> A[!(proc0.state=critical) U proc0.state=critical])

AG(proc0.state=critical -> (proc0.state=critical |

A[!(proc0.state=critical) U proc1.state=critical]))

Prva formula u slučaju da ne želimo da „2 takta“ ostane u kritičnom, a druga ukoliko proc0 može ostati u kritičnom odsječku i više od „jednog takta“.

**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?**

Obje verzije svojstva su zadovoljene. Problem koji je još uvijek moguće prisutan je svojstvo nedeterminiranog redoslijeda pošto u glavnom modulu dajemo proc0 prednost (on prvi kreće).

**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 kriti*č*nom odsje*č*ku.»*

Potrebno je napisati dva oblika obilježja:

a) specifikacija da je moguće jedno nepoželjno ponašanje (engl. *refutation*)

b) specifikacija da nema nepoželjnog ponašanja

a) EF (proc0.state=critical & proc1.state=critical)

b) !EF (proc0.state=critical & proc1.state=critical)

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

Obilježje napisano u obliku 1.2.b) je zadovoljeno – to i želimo.

Obilježje napisano u obliku 1.2.a) nije zadovoljeno – to znači da se niti u jednom trenutku ne mogu naći oba procesa u kritičnom odsječku – potvrdili smo obilježje sigurnosti kontradikcijom.

**2.4. Specificiraj i napiši u CTL notaciji obilježje (engl. *liveness property*):**

*«Ako proces pokuša u*ć*i u kriti*č*ni odsje*č*ak, kona*č*no* ć*e i u*ć*i»*

Specifikaciju napiši za oba procesa.

AG(proc0.state=reading-> AF(proc0.state=critical))

AG(proc1.state=reading-> AF(proc1.state=critical))

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

Svojstvo nije zadovoljeno. Problem je u tome da oba procesa mogu postaviti flag u TRUE i zapeti u stanju reading.

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

EF(AG(proc0.state=reading & proc1.state=reading))

Specifikacija je zadovoljena.

**2.7. Prouči primjer mutex\_5ex.smv.**

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

EF (proc0.state=critical & proc1.state=critical) == FALSE

!EF (proc0.state=critical & proc1.state=critical) == TRUE

Svojstvo sigurnosti je zadovoljeno.

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

AG(proc0.state=testflag -> AF(proc0.state=critical)) == FALSE

AG(proc1.state=testflag -> AF(proc1.state=critical)) == FALSE

Svojstvo životnosti nije zadovoljeno.

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

Problem je da procesi mogu zapeti u stanju testflag1 zbog nedeterminiranosti stanja critical. Jedan proces može držati svoju zastavicu u TRUE, zbog čega će drugi zapeti u stanju testflag1.

EF(EG(proc0.state=testflag1 & proc1.state=critical)) == TRUE

**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)?**

Sigurnost:

EF (proc0.state=critical & proc1.state=critical) == FALSE

!EF (proc0.state=critical & proc1.state=critical) == TRUE

Svojstvo sigurnosti je zadovoljeno.

Životnost:

AG(proc0.state=testflag -> AF(proc0.state=critical)) == TRUE

AG(proc1.state=testflag -> AF(proc1.state=critical)) == TRUE

Svojstvo životnosti je zadovoljeno.

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

Ideja testne zastavice, dodatnog stanja za test te korištenje varijable turn.

**2.14. Prouči primjer mutex\_7ex.smv**

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

**2.15. Je li zadovoljeno obilježje sigurnosti?**

EF (proc0.state=critical & proc1.state=critical) == FALSE

!EF (proc0.state=critical & proc1.state=critical) == TRUE

Svojstvo sigurnosti je zadovoljeno.

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

AG(proc0.state=trying -> AF(proc0.state=critical)) == TRUE

AG(proc1.state=trying -> AF(proc1.state=critical)) == TRUE

Svojstvo životnosti je zadovoljeno.

**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 ljusku NuSMV-a. Učitaj model zadan datotekom mutex\_3ex\_int.smv.**

> NuSMV -int

NuSMV > read\_model -i mutex\_3ex\_int.smv (Reads a NuSMV file into NuSMV)

**3.2. Inicijaliziraj sustav za verifikaciju. Ukratko obrazloži što se sve događa prilikom pokretanja naredbe "go".**

Naredba:

NuSMV > go (Initializes the system for the verification)

je istovjetna slijedu naredbi:

NuSMV > read\_model -i mutex\_3ex\_int.smv (Reads a NuSMV file into NuSMV)

NuSMV > flatten\_hierarchy (Flattens the hierarchy of modules)

NuSMV > encode\_variables (Builds the BDD variables necessary to compile

the model into a BDD.)

NuSMV > build\_model (Compiles the flattened hierarchy into a BDD)

[NuSMV > show\_vars] (Shows model’s symbolic variables and their values)

**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?**

NuSMV > pick\_state -i (Picks a state from the set of initial states)

\* odabrano je stanje 0) turn = FALSE, proc0.state = proc1.state = entering

NuSMV > simulate -v -k 3 (Performs a simulation from the current selected state)

\* -v za opširan ispis, -p za ispis promjena

NuSMV > show\_traces [-v]

**3.4. Provjeri stroj s konačnim brojem stanja. Kakva je relacija prijelaza tog automata?**

**Može li doći do potpunog zastoja?**

NuSMV > check\_fsm (Checks the transition relation for totality)

Relacija je potpuna (binarna relacija *R* nad setom *X* je potpuna ako za svaki *a* i b iz X vrijedi *aRb* ili *bRa* (ili oboje, ne radi se o xor)). Za potpune relacije se ponekada kaže da imaju svojstvo usporedivosti.

Ne postoji stanje 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)**

NuSMV > print\_reachable\_states [-v]

Dohvatljivih stanja ima 16 (24) od ukupno 32 (25) moguća stanja.

Promjer FSM-a je 7.

**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?**

Radi se o specifikaciji EF(proc0.state = critical & proc1.state = critical) koja provjerava postoji li put na kojemu se u nekom stanju oba procesora nalaze u kritičnom odsječku. Time se provjerava obilježje sigurnosti.

NuSMV > check\_ctlspec -n 0

Specifikacija nije zadovoljena. Obilježje sigurnosti je zadovoljeno.

**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?**

Radi se o specifikaciji AG(proc0.state = entering -> AF proc0.state = critical) koja provjerava da li proces nakon što postavi zahtjev za ulaz u kritični odsječak u konačnom vremenu i ulazi u njega. Time se provjerava obilježje životnosti.

NuSMV > check\_ctlspec -n 1

Specifikacije je zadovoljena. Obilježje životnosti 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. Ima li kakve promjene u odnosu na prethodni zadatak?**

Specifikacija više nije zadovoljena, što znači da obilježje životnosti više ne vrijedi. Ono što je osiguravalo životnost su bila obilježja pravednosti koja smo dodavali u 1.6, 1.8 i 1.9.

**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?**

**+ 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?**

COMPUTE MIN[proc0.state = noncritical, proc0.state = exiting]

\* MIN[start, final] – vraća dužinu najkraćeg puta od stanja start do stanja final.

NuSMV > check\_compute -n 2

Vraća se rezultat 3, što znači da u FSM-u iz stanja noncritical možemo doći do stanja exiting u najmanje 3 prijelaza.

COMPUTE MAX[proc0.state = noncritical, proc0.state = critical]

\* MAX[start, final] – vraća dužinu najdužeg pita od stanja start do stanja final.

NuSMV > check\_compute -n 3

Vraća rezultat ∞, što znači da se u nekim od stanjima između noncritical i exiting možemo „zavrtiti“ u beskonačnim petljama (konkretno, u noncritical ili u critical).

Navedeni rezultati su rezultati dobiveni uz uklonjena ograničenja pravednosti. Rezultati su isti i uz ograničenja pravednosti, što znači da se ograničenja pravednosti uzimaju u obzir prilikom provjere CTL specifikacije, ali ne tokom provjere svojstava za rad u stvarnom vremenu.

**4. dio**

**4.1. Zadani kôd u NuSMV-u sadrži implicitni nedeterminizam uzrokovan varijablom**

***request.* Izmijeni zadani kôd tako da sadrži *isključivo* eksplicitni nedeterminizam.**

MODULE main

VAR

request: boolean

flag: {red, blue};

ASSIGN

init(flag) := red;

next(flag) := case

request:blue;

TRUE: red;

esac;

Izvor implicitnog nedeterminizma je varijabla request jer je neinicijalizirana. Eksplicitni nedeterminizam se postiže nedeterminističkom dodjelom vrijednosti varijabli request – pridjeljuje se vrijednost iz skupa {true, false}. Prilagođeni kod je sljedeći:

MODULE main

VAR

request: {TRUE, FALSE}

flag: {red, blue};

ASSIGN

init(flag) := red;

next(flag) := case

request:blue;

TRUE: red;

esac;

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

MODULE main

VAR

request : boolean;

status : {ready, busy};

negReq : boolean;

ASSIGN

init(status) := busy;

init(request) := FALSE;

next(status) := case

request:busy;

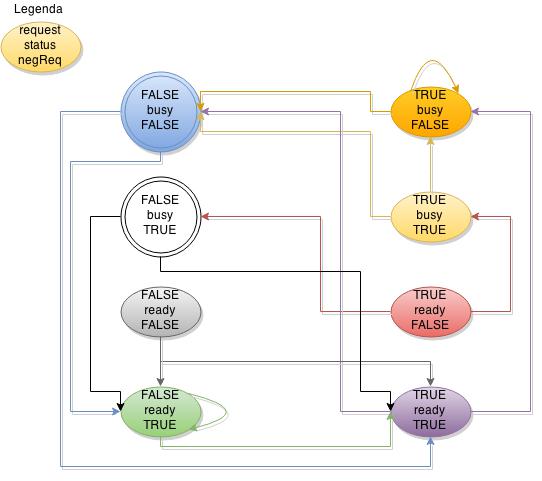
TRUE:ready;

esac;

next(negReq) := !request;

a) skup svih mogućih stanja – SA

b) skup svih dosežljivih stanja – SR (uz pretpostavku da su sva početna stanja dosežljiva)

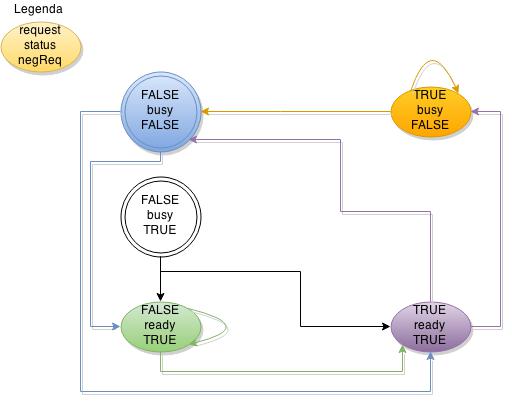


Nadam se da je dobro nacrtano (ispravke su dobrodošle). Ovako sam ja to shvatio iz koda:

* ako je prvi redak u stanju (request) TRUE, onda se prelazi u stanja u kojima je drugi redak (status) busy,
  + inače prelazi u stanja u kojima je drugi redak (status) ready
* uz gornji uvjet mora još kod prijelaza vrijediti sljedeće:
  + treći redak u novom stanju (negReg) mijenja vrijednost u suprotnu od vrijednosti request u trenutnom stanju

Zatim još treba maknuti nedohvatljiva stanja:

* maknemo odmah sivo i crveno stanje, nema strelica u njih,
* nakon micanja crvenog stanja nema strelica u žuto, pa mičemo i njega.



SA = {(FALSE, busy, FALSE), (FALSE, busy, TRUE), (FALSE, ready, FALSE), (FALSE, ready, TRUE),

(TRUE, busy, FALSE), (TRUE, busy, TRUE), (TRUE, ready, FALSE), (TRUE, ready, TRUE)}

SR = {(FALSE, busy, FALSE), (FALSE, busy, TRUE), (FALSE, ready, TRUE), (TRUE, busy, FALSE),

(TRUE, ready, TRUE)}

**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) NuSMV podržava opis sustava razbijanjem na više modula. Modul se instancira tako da se deklarira varijabla čiji je tip ime modula. Pristup varijablama unutar modula obavlja se pomoću točke „.“.

Postoje dva načina izvršavanja:

* sinkroni – postoji globalni sad, i svakim taktom globalnog sata, svaki od modula (njegov ASSIGN blok) se izvršava paralelno,
* asinkroni način – pojedini dijelovi sustava (moduli) odvijaju se različitim brzinama, pri čemu se njihova izvođenja proizvoljno isprepliću:
  + u svakom taktu, proizvoljno se odabire jedan od modula i izvršava se (unutarnja varijabla running za taj modul se postavlja na TRUE),
  + ukoliko se želi da se neki modul izvršava u *interleaved* načinu izvođenja potrebno je postaviti ključnu riječ process prilikom instanciranja modula,
  + korisno za opis komunikacijskih protokola, asinkronih sklopova i ostalih sustava koji nisu sinkronizirani pomoću nekog globalnog sata te za modeliranje višedretvenosti.

2)

MODULE main

VAR

invertorA : process invert;

invertorB : process invert;

MODULE invert

VAR

value : boolean;

ASSIGN

init(value):= FALSE;

next(value):= !value;

3)

MODULE main

VAR

invertorA : process invert;

invertorB : process invert;

JUSTICE invertorA.value

CTLSPEC EF(EG(invertorA.value))

MODULE invert

VAR

value : boolean;

ASSIGN

init(value):= FALSE;

next(value):= !value;