Formalne metode u oblikovanju sustava

1. domaća zadaća - NuSMV

Zagreb, travanj 2020.

Sadržaj

Izjava

Zadaci priloženi uz ovu domaću zadaću pripadaju njihovim vlasnicima te služe isključivo u edukacijske svrhe. Bilo koja promjena originala je isključivo radi estetske i funkcionalne prirode i ne mijenja smisao informacije. Također, takve promjene ne primijenjuje nikakvu drukčiju licencu niti se smatraju intelektualnim vlasništvom uređivača.

Uz ovu datoteku obično dolaze i primjerci kôda, koji su zaštićeni licencom:

Copyright 2020 Miljenko Šuflaj

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Priprema

Potrebno je instalirati NuSMV prema uputama predmeta u repozitoriju, kao i primjere u direktoriju vježbe. Svi primjeri nalazit će se u datoteci `primjeri` u korijenu direktorija repozitorija vježbe. Direktorij NuSMV treba postaviti u korijen repozitorija - nakon toga će se izvršni program NuSMV nalaziti na `NuSMV/bin/NuSMV.exe`. Svi primjeri bit će pokrenuti iz direktorija `DZ1`, što znači da se naredbe odnose na scenarij kada smo pozicionirani u tom direktoriju.

1. DIO

Zadatak 1.1

Prouči primjer `mutex_1ex.smv`.

Odgovor

```
MODULE main
VAR
        turn : boolean;
        proc0 : process user(turn, FALSE);
        proc1 : process user(turn, TRUE);
ASSIGN
        init(turn) := FALSE;

MODULE user(turn, ind)
VAR
        state : {entering, critical, exiting, noncritical};
ASSIGN
        init(state) := {entering, noncritical};
        next(state) :=
                case
                        (state = entering) & (turn = ind) : critical;
                        state = critical : {critical, exiting};
                        state = exiting : noncritical;
                        state = noncritical : {entering, noncritical};
                        TRUE : state;
                esac;
        next(turn) :=
                case
                        (state = exiting) & (ind = FALSE) : TRUE;
                        (state = exiting) & (ind = TRUE) : FALSE;
                        TRUE : turn;
                esac;
```

Zadatak 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.

Odgovor

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

b) `AG !(proc0.state = critical & proc1.state = critical)`

Zadatak 1.3

Utvrdi pomoću sustava NuSMV je li ispunjeno navedeno obilježje. Objasni rezultat na temelju kôda primjera (ne na temelju ispisa traga).

Odgovor

Pokrenimo dvije naredbe:

- `NuSMV/bin/NuSMV DZ1/p1/task_02a.smv`
- `NuSMV/bin/NuSMV DZ1/p1/task_02b.smv`

Pokretanjem prve naredbe vidimo da specifikacija nije ispravna, dok pokretanjem druge utvrđujemo da je ona ispravna.

Ako analiziramo kôd, vidimo da kôd **prve** specifikacije želimo da postoji (barem jedan) put na kojem se događa da su istovremeno oba procesa u kritičnom odsječku. Međusobno isključivanje podrazumijeva da se ovo nikad ne događa, tj. da je u kritičnom odsječku uvijek **isključivo** jedan proces. Kod **druge** specifikacije želimo da ne postoji ni jedno stanje u kojem se može dogoditi da su oba procesa u kritičnom odsječku. Ovo ujedno i **zadovoljava** obilježje sigurnosti.

Zadatak 1.4

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

Specifikaciju napiši za oba procesa.

Odgovor

Specifikacija glasi:

```
CTLSPEC AG (proc0.state = entering -> AF (proc0.state = critical))
CTLSPEC AG (proc1.state = entering -> AF (proc1.state = critical))
```

Zadatak 1.5

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

Odgovor

Pokretanjem `NuSMV/bin/NuSMV DZ1/p1/task_04.smv` uviđamo da navedeno obilježje nije zadovoljeno. Ako pogledamo specifikaciju, konkretno liniju

```
state = critical : {critical, exiting};
```

Možemo vidjeti da proces teoretski vječno može zapeti u kritičnom odsječku. Naša specifikacija govori da za sve čvorove mora postojati neki put kako bi iz stanja `entering` prešli u stanje `critical`. Trace nam je naveo i slučajeve kad to neće biti istina. Ovo se može zaobići nekim konačnim brojem prijelaza u kritičnom odsječku nakon kojih će posljednje od njih preći u stanje `exiting`. Bez takvog ponašanja, praktički smo omogućili procesu da zaglavi.

Zadatak 1.6

U `mutex_1ex.smv` dodaj ograničenje pravednosti (engl. *fairness*): svaka instanca procesa obavlja se beskonačno mnogo puta. Napiši ovdje kako ono glasi.

Odgovor

U `mutex_1ex.smv` treba dodati liniju:

```
JUSTICE running
```

Zadatak 1.7

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

Odgovor

Pokretanjem `NuSMV/bin/NuSMV DZ1/p1/task_06.smv` vidimo da naša definicija još uvijek ne zadovoljava obilježje životnosti. Dodavanjem ograničenja pravednosti osigurali smo samo da naš sustav u svakom trenutku bira hoće li se proces izvršavati ili ne, što garancija da jedan od njih neće biti beskonačno u kritičnom odsječku.

Zadatak 1.8

U `mutex_1ex.smv` još dodaj ograničenje pravednosti: svaka instanca procesa ne može beskonačno dugo ostati u **kritičnom** odsječku. Napiši ovdje kako ono glasi. Provjeri sad svojstvo životnosti za `mutex_1ex.smv`.

Odgovor

U `mutex_1ex.smv` treba osim prethodnog ograničenja za `running` dodati:

```
JUSTICE !(state = critical)
```

Pokretanjem `NuSMV/bin/NuSMV DZ1/p1/task_08.smv` NuSMV dojavljuje da specifikacija još uvijek nije ispravna.

Zadatak 1.9

U `mutex_1ex.smv` dodaj još jedno ograničenje pravednosti: svaka instanca procesa ne može beskonačno dugo ostati u **nekritičnom** odsječku. Napiši ovdje kako ono glasi. Provjeri sad svojstvo životnosti za `mutex_1ex.smv`.

Odgovor

U `mutex_1ex.smv` treba još dodati:

```
JUSTICE !(state = noncritical)
```

Pokretanjem `NuSMV/bin/NuSMV DZ1/p1/task_09.smv` NuSMV dojavljuje da je sada specifikacija ispravna.

Zadatak 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.
```

Odgovor

```
CTLSPEC AG (proc0.state = exiting ->
            A[proc1.state = exiting U !(proc0.state = critical)])
```

Zadatak 1.11

Utvrdi je li zadovoljeno navedeno obilježje za `mutex_1ex.smv` (uz navedena ograničenja pravednosti). Koja obilježja protokola međusobnog isključivanja rješavaju ograničenja pravednosti prethodno navedena, a koji problem je još uvijek prisutan?

Odgovor

Pokretanjem `NuSMV/bin/NuSMV DZ1/p1/task_10.smv` utvrđujemo da je navedeno obilježje zadovoljeno.

Riješili smo probleme beskonačnog boravka u (ne)kritičnim odsječcima, međutim, recimo da smo pokrenuli **proces 0** i on je ušao u kritični odsječak te izašao iz njega. Ako sustav ponovo odluči pokrenuti **proces 0**, trebat će pričekati da se pokrene i završi **proces 1**. Drugim riječima, predodređen je redoslijed kojim procesi ulaze u kritičan odsječak.

2. DIO

Zadatak 2.1

Prouči primjer `mutex_2ex.smv`.

Odgovor

```
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;
```

Zadatak 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

Odgovor

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

b) `AG !(proc0.state = critical & proc1.state = critical)`

Zadatak 2.3

Utvrdi da li je ispunjeno zadano obilježje. Objasni rezultat na temelju kôda primjera (ne na temelju ispisa traga).

Odgovor

Zadano obilježje je iz sličnih razloga kao i u 1. dijelu zadovoljeno samo za b) inačicu: b) inačica garantira da se ni u jednom čvoru neće dogoditi da su oba procesa u kritičnom odsječku istovremeno.

Zadatak 2.4

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

Specifikaciju napiši za oba procesa.

Odgovor

Specifikacija glasi:

```
CTLSPEC AG (proc0.state = reading -> AF(proc0.state = critical))
CTLSPEC AG (proc1.state = reading -> AF(proc1.state = critical))
```

Zadatak 2.5

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

Odgovor

Nakon što pokrenemo `NuSMV/bin/NuSMV DZ1/p2/task_04.smv` vidimo da zadano obilježje nije zadovoljeno. Razlog tome je što proces može vječno ostati u stanju `reading`, kao i u prethodnom dijelu.

Zadatak 2.6

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

Odgovor

Specifikacija tog problema glasi:

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

Nakon provjere sustava s ovom specifikacijom NuSMV javlja da je ovaj uvjet zadovoljen.

Zadatak 2.7

Prouči primjer `mutex_3ex.smv`.

Odgovor

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

MODULE user(oflag)
VAR
        flag : boolean;
        state  : {start, testflag, testflag1, critical, noncritical};
ASSIGN
        init(state) := start;
        init(flag) := TRUE;
        next(state) :=
                case
                        state = start : testflag;
                        state = testflag & oflag : testflag1;
                        state = testflag & !oflag : critical;
                        state = testflag1 & oflag : testflag1;
                        state = testflag1 & !oflag : testflag;
                        state = critical : {critical, noncritical};
                        state = noncritical : {noncritical, start};
                        TRUE : state;
                esac;

        next(flag) :=
                case
                        state = start : TRUE;
                        state = testflag & oflag : FALSE;
                        state = testflag1 & !oflag : TRUE;
                        state = noncritical : FALSE;
                        TRUE : flag;
                esac;
```

Zadatak 2.8

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

Odgovor

Kao i prije, obilježje sigurnosti zadovoljeno je samo za b) inačicu:

```
CTLSPEC AG !(proc0.state = critical & proc1.state = critical)
```

Zadatak 2.9

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

Odgovor

Dodajmo specifikacije

```
CTLSPEC AG (proc0.state = testflag -> AF (proc0.state = critical))
CTLSPEC AG (proc1.state = testflag -> AF (proc1.state = critical))
```

Nakon izvršavanja `NuSMV/bin/NuSMV DZ1/p2/task_09.smv` NuSMV nam javlja da obilježje nije zadovoljeno.

Zadatak 2.10

Dodajte sad ograničenja pravednosti kao kôd zadataka 1.6, 1.8 i 1.9. Je li sad zadovoljeno obilježje životnosti?

Odgovor

Pokretanjem `NuSMV/bin/NuSMV DZ1/p2/task_10.smv` je lako vidljivo da nije zadovoljeno obilježje životnosti.

Zadatak 2.11

Koji je problem u ovoj implementaciji međusobnog isključivanja (bez obzira na uključena ograničenja pravednosti)? Gdje sustav može *zapeti*? Problem specificiraj u CTL notaciji i provjeri pomoću sustava NuSMV.

Odgovor

Problem je u sustavu testnih zastavica. Moguće je da oba procesa zapnu u iz `testflag1`, što znači da nikad neće doći u stanje `testflag`, preko kojeg bi neki proces mogao doći u kritični odsječak. Dovoljno je pokazati da se to implicira kad samo jedan od procesa bude u `testflag1`. Na primjer, u kôd iz `mutex_3ex.smv` možemo dodati liniju:

```
CTLSPEC EF (AG (proc0.state = testflag1) -> AG (proc1.state = testflag1))
```

Izvšimo li tada `NuSMV/bin/NuSMV DZ1/p2/task_11.smv`, vidimo da nam NuSMV kaže da je zadovoljena specifikacija, što znači da imamo problem.

Zadatak 2.12

Prouči primjer `mutex_4ex.smv`. Ovo je primjer uspješne implementacije međusobnog isključivanja. Zasniva se na rješenju kojeg je predložio T. Dekker a opisao E. W. Dijkstra.

Odgovor

```
MODULE main
VAR
        turn  : boolean;
        proc0 : process user(proc1.flag, turn, FALSE);
        proc1 : process user(proc0.flag, turn, TRUE);
ASSIGN
        init(turn) := FALSE;

MODULE user(oflag, turn, ind)
VAR
        flag : boolean;
        state  : {start, testflag, testturn, testturn1, critical, noncritical};
ASSIGN
        init(state) := start;
        init(flag) := TRUE;
        next(state) :=
                case
                        state = start : testflag;
                        state = testflag & oflag : testturn ;
                        state = testflag & !oflag : critical;
                        (state = testturn) & (turn = ind)  : testflag;
                        (state = testturn) & (turn != ind)  : testturn1;
                        (state = testturn1) & (turn = ind)  : testflag;
                        (state = testturn1) & (turn != ind)  : testturn1;
                        state = critical : {critical, noncritical};
                        state = noncritical : {noncritical, start};
                esac;

        next(flag) :=
                case
                        state = start : TRUE;
                        (state = testturn) & (turn != ind) : FALSE;
                        (state = testturn1) & (turn = ind)  : TRUE;
                        state = noncritical : FALSE;
                        TRUE : flag;
                esac;
```

```
41          next(turn) :=
42                  case
43                          (state = noncritical) & (ind = FALSE) : TRUE;
44                          (state = noncritical) & (ind = TRUE) : FALSE;
45                          TRUE : turn;
46                  esac;
47
48  FAIRNESS running
49  FAIRNESS !(state=critical)
50  FAIRNESS !(state=noncritical)
```

Zadatak 2.13

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

Odgovor

Kao i inače, dodat ćemo par linija specifikacija:

```
CTLSPEC AG !(proc0.state = critical & proc1.state = critical)
CTLSPEC AG (proc0.state = testflag -> AF (proc0.state = critical))
CTLSPEC AG (proc1.state = testflag -> AF (proc1.state = critical))
```

Kad pokrenemo `NuSMV/bin/NuSMV DZ1/p2/task_13.smv`, vidimo da su sva 3 uvjeta zadovoljena, tj. ispunjena su svojstva sigurnosti i životnosti.

Zadatak 2.14

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

Odgovor

To su ideje da proces ne bi smio tražiti ulazak u kritični odsječak ako vidi da to drugi proces već radi.

Zadatak 2.15

Prouči primjer `mutex_5ex.smv`. Ovaj je primjer implementacija Petersonovog algoritma, koji predstavlja pojednostavnjenje prethodnog (Dekkerovog) algoritma.

Odgovor

```
MODULE main
VAR
        turn  : boolean;
        proc0 : process user(proc1.flag, turn, FALSE);
        proc1 : process user(proc0.flag, turn, TRUE);
ASSIGN
        init(turn) := {FALSE, TRUE};

MODULE user(oflag, turn, ind)
VAR
        flag : boolean;
        state  : {start, trying, critical, noncritical};
ASSIGN
        init(state) := start;
        init(flag) := TRUE;
        next(state) :=
                case
                        state = start : trying;
                        (state = trying) & (turn = ind) & (oflag) : trying;
                        (state = trying) & ( (turn != ind) | (!oflag) ) : critical;
                        state =  critical : {critical, noncritical};
                        state = noncritical : {noncritical, start};
                esac;

        next(flag) :=
                case
                        state = start : TRUE;
                        state = noncritical : FALSE;
                        TRUE : flag;
                esac;

        next(turn) :=
                case
                        (state = start) & (ind = FALSE) : FALSE;
                        (state = start) & (ind = TRUE) : TRUE;
                        TRUE : turn;
                esac;
FAIRNESS running
FAIRNESS !(state=critical)
FAIRNESS !(state=noncritical)
```

Zadatak 2.16

Je li zadovoljeno obilježje sigurnosti?

Odgovor

To ćemo provjeriti tako da umetnemo specifikaciju:

```
CTLSPEC AG !(proc0.state = critical & proc1.state = critical)
```

Kada pokrenemo `NuSMV/bin/NuSMV DZ1/p2/task_16.smv`, NuSMV nam kaže da je obilježje zadovoljeno.

Zadatak 2.17

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)?

Odgovor

Traženi izrazi su:

```
CTLSPEC AG (proc0.state = trying -> AF(proc0.state = critical))
CTLSPEC AG (proc1.state = trying -> AF(proc1.state = critical))
```

Nakon pokretanja `NuSMV/bin/NuSMV DZ1/p2/task_17.smv`, NuSMV nam kaže da je obilježje životnosti zadovoljeno.

3. DIO

Prouči potpoglavlja 3.1, 3.2, 3.5 i 3.7 iz NuSMV priručnika `NuSMV 2.6 User Manual`.
Nakon toga riješi sljedeće zadatke:

Zadatak 3.1

Pokreni interaktivno ljusku NuSMV-a. Učitaj model zadan datotekom `mutex_1ex_int.smv`.

Odgovor

Treba izvršiti niz naredbi:

- `../NuSMV/bin/NuSMV -int`
- `read_model -i primjeri/mutex_1ex_int.smv`

Zadatak 3.2

Inicijaliziraj sustav za verifikaciju. Ukratko obrazloži što se sve događa prilikom pokretanja naredbe `go`.

Odgovor

Naredba `go` odgovara sekvenci naredbi:

1. `read_model` - učitava model s puta definiranog u env varijabli `input-file`
2. `flatten_hierarchy` - instancira module i procese zamjenom stvarnih parametara sa formalnim
3. `encode_variables` - generira varijable binarnih dijagrama odluka (engl. *BDD variables*) i algebarskih dijagrama odluka (engl. *ADD variables*)
4. `build_flat_model` - prevodi spljoštenu hijerarhiju u skup početnih stanja, invarijanta i prijelaznih relacija
5. `build_model` - prevodi spljošteni model u binarne dijagrame odluka

Ako je netka od ovih naredbi već izvršena, preskočit će se njeno izvođenje.

Zadatak 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?

Odgovor

Prvo je potrebno odabrati proizvoljno stanje. To ćemo moći učiniti upisivanjem `pick_state -r`.

Zatim, ako želimo proći kroz n stanja, trebamo upisati `simulate -k n`, gdje ćemo n zamijeniti s brojem prijelaza. U našem slučaju je to **3**, pa naredba glasi `simulate -k 3`.

Naposljetku, da bi ispisali trag, potrebno je upisati `show_traces`.

Zadatak 3.4

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

Odgovor

Provjeriti KA možemo s naredbom `check_fsm`. Kad to upišemo, za datoteku `mutex_1ex_int.smv` nam se prikaže:

```
############################################################
The transition relation is total: No deadlock state exists
############################################################
```

To u prijevodu znači da je relacija prijelaza potpuna i da ne može doći do potpunog zastoja.

Zadatak 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)

Odgovor

Naredbom `print_reachable_states` doznajemo da je broj dosezljivih stanja **16**, a da ukupno postoji **32** stanja.

Zadatak 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?

Odgovor

Provjera prve po redu CTL specifikacije moći će se izvršiti upisivanjem naredbe `check_ctlspec -n 0`. NuSMV nam potom za danu datoteku vraća infomaciju da je tvrdnja lažna. Provjeravala se specifikacija

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

čijom se neistinitošću potvrdilo da vrijedi obilježje sigurnosti.

Zadatak 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?

Odgovor

Slično kao i prije, možemo provjeriti drugu po redu specifikaciju onosom naredbe `check_ctlspec -n 1`. NuSMV nam je vratio rezultat da je specifikacija istinita. Provjeravala se specifikacija

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

kojom se provjerava vrijedi li obilježje životnosti za **proces 0**. Obilježje je zadovoljeno samo djelomično, jer trebamo provjeriti i **proces 1**.

Zadatak 3.8

Sada ukloni obilježja pravednosti iz datoteke `mutex_1ex_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?

Odgovor

Kako ne bi izmijenjivali sadržaj direktorija `primjeri`, uređeni kôd smjestili smo u datoteku `DZ1/p3/task_08.smv`. Nakon pokretanja `check_ctlspec -n 1` druga po redu CTL specifikacija pada, što znači da obilježje životnosti više nije zadovoljeno.

Zadatak 3.9

Prouči naredbe za provjeru svojstava sustava za rad u stvarnom vremenu koje su zadane s ključnom riječi `COMPUTE` u datoteci `mutex_1ex_int.smv`. Koje je značenje svake od tih naredbi?

Odgovor

Linija `COMPUTE MIN[proc0.state = noncritical, proc0.state = exiting]` računa najkraću udaljenost puta od stanja `noncritical` do `exiting`. Uz malo višu razinu apstrakcije, ovo računa koliko je najmanje koraka potrebno za jedan ciklus procesa.

Linija `COMPUTE MAX[proc0.state = noncritical, proc0.state = critical]` računa najdulju udaljenost puta od stanja `noncritical` do `exiting`. Uz malo višu razinu apstrakcije, ovo računa koliko je najviše koraka potrebno za jedan ciklus procesa.

Zadatak 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?

Odgovor

Provjeru ovih naredbi možemo napraviti s `check_compute`. Pokrenemo li ovu naredbu za datoteku `primjeri/mutex_1ex_int.smv`, dobit ćemo izlaz:

```
the result of MIN[proc0.state=noncritical, proc0.state=exiting] is 3
the result of MAX[proc0.state=noncritical, proc0.state=critical] is infinity
```

Pokrenemo li istu naredbu za `p3/task_08.smv` (jer tu nema ograničenja pravednosti), dobit ćemo izlaz:

```
the result of MIN[proc0.state=noncritical, proc0.state=exiting] is 3
the result of MAX[proc0.state=noncritical, proc0.state=critical] is infinity
```

Vidimo da su rezultati isti, pa stoga zaključujemo da ograničenja pravednosti ne utječu na naredbu `COMPUTE`. Napomena - izlazima su maknuti neki razmaci i `--` na početku kako bi stali na stranicu.

4. DIO

Zadatak 4.1

Prouči primjer `ferryman.smv`.

Odgovor

```
1  -- Players: ferryman, wolf, goat, cabbage.
2  -- Problem:
3  --         All start on one (right) side of the river.
4  --         All should end up on the other (left) side of the river.
5  --        The ferryman can only take one object at a time.
6  --        If left alone without the ferryman:
7  --                The wolf eats the goat, or
8  --                 The goat eats the cabbage.
9
10 MODULE main
11 VAR
12 -- the ferryman and the three items
13   cabbage  : {right,left};
14   goat     : {right,left};
15   wolf     : {right,left};
16   ferryman : {right,left};
17   carry_cabbage: boolean;
18   carry_goat : boolean;
19   carry_wolf : boolean;
20   no_carry : boolean;
21
22 -- possible moves
23   move    : {c, g, w, e};
24
25
26 -- initially everything is on the right bank
27 ASSIGN
28   init(cabbage)  := right;
29   init(goat)     := right;
30   init(wolf)     := right;
31   init(ferryman) := right;
32
33   carry_cabbage := move=c;
34   carry_goat := move=g;
35   carry_wolf := move=w;
36   no_carry := move=e;
37
38
39
40
```

```
41  TRANS
42    carry_cabbage ->
43                  cabbage=ferryman & -- cabbage is carried by the ferryman
44                  next(cabbage)!=cabbage &
45                  next(ferryman)!=ferryman &
46                  next(goat)=goat &
47                  next(wolf)=wolf
48
49  TRANS
50    carry_goat ->
51                  goat=ferryman &
52                  next(goat)!=goat &
53                  next(ferryman)!=ferryman &
54                  next(cabbage)=cabbage &
55                  next(wolf)=wolf
56
57  TRANS
58    carry_wolf ->
59                  wolf=ferryman &
60                  next(wolf)!=wolf &
61                  next(ferryman)!=ferryman &
62                  next(goat)=goat &
63                  next(cabbage)=cabbage
64
65  TRANS
66    no_carry ->
67          next(ferryman)!=ferryman &
68          next(goat)=goat &
69          next(cabbage)=cabbage &
70          next(wolf)=wolf
71
72
73  -- goat and wolf must not be left unattended !
74  -- goat and cabbage must not be left unattended !
75  DEFINE
76    safe_state := (goat = wolf | goat = cabbage) -> goat = ferryman;
77    goal := cabbage = left & goat = left & wolf = left;
```

Zadatak 4.2

Specificiraj i napiši u CTL notaciji obilježje:
`Ne postoji siguran put kojim se dolazi do cilja problema`. Pritom se u specifikaciji trebaju koristiti već definirane makro-instrukcije programa.

Odgovor

Izjava se može preoblikovati u:
`Nemoguće je stići do cilja dok se ne zađe na nesiguran put`. Tada je to lagano opisati specifikacijom:

```
CTLSPEC A [!goal U !safe_state]
```

Zadatak 4.3

Provjeri zadano svojstvo. Je li ono zadovoljeno? Što nam u ovom slučaju daje ispis traga programa? Opiši redoslijed izvođenja kojim se uspješno dolazi do cilja problema.

Odgovor

Pokretanjem provjere, NuSMV vraća trag:

```
-- specification A [ !goal U !safe_state ]   is false
```

Svojstvo nije zadovoljeno. Ispis traga nam govori o putu koji je kontradikcija ovog svojstva: to je ujedno i put za koji se postiže uvjet `goal` bez da se ostvari uvjet `!safe_state`. Kontraprimjer koji je izbacio NuSMV glasi:

1. skeledžija, vuk, ovca i kupus započinju na desnom koritu
2. skeledžija se penje na skelu s ovcom
3. skeledžija vozi ovcu na lijevo korito
4. skeledžija se iskrcava s ovcom na lijevo korito
5. skeledžija se penje na skelu sam
6. skeledžija se vozi na desno korito
7. skeledžija se iskrcava na desno korito
8. skeledžija se penje na skelu s vukom
9. skeledžija vozi vuka na lijevo korito
10. skeledžija se iskrcava s vukom na lijevo korito
11. skeledžija se penje na skelu s ovcom

12. skeledžija vozi ovcu na lijevo korito
13. skeledžija se iskrcava s ovcom na desno korito
14. skeledžija se penje na skelu s kupusom
15. skeledžija vozi kupus na lijevo korito
16. skeledžija se iskrcava s kupusom na lijevo korito
17. skeledžija se penje na skelu sam
18. skeledžija se vozi na desno korito
19. skeledžija se iskrcava na desno korito
20. skeledžija se penje na skelu s ovcom
21. skeledžija vozi ovcu na lijevo korito
22. skeledžija se iskrcava s ovcom na lijevo korito

Zadatak 4.4

Zadani kôd u NuSMV-u sadrži implicitni nedeterminizam uzrokovan varijablom `request`. Izmijeni zadani kôd tako da sadrži isključivo eksplicitni nedeterminizam (napomena: ne mijenjati tipove varijabli!).

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

Odgovor

Potrebni dodaci su:

```
init(request) := FALSE;

next(request) :=
case
    TRUE : {TRUE, FALSE};
esac;
```

Ovo se također nalazi u `p4/task_04.smv`, te je komentirano.

Zadatak 4.5

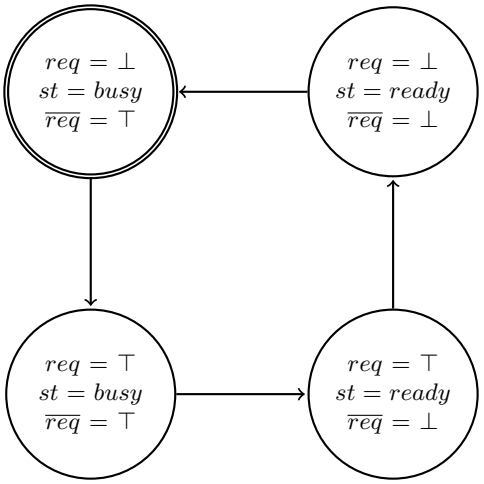
Za zadani kôd u NuSMV-u nacrtaj odgovarajuću Kripke strukturu i odredi:

a) skup svih mogućih stanja -– S_A

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

```
1  MODULE main
2  VAR
3      request : boolean;
4      status : {ready, busy};
5      negReq : boolean;
6
7  ASSIGN
8      init(request) := FALSE;
9      init(status) := busy;
10     init(negReq) := TRUE;
11
12     next(request) :=
13     case
14         (status = ready) : FALSE;
15         TRUE: TRUE;
16     esac;
17
18     next(status) :=
19     case
20         request : ready;
21         TRUE : busy;
22     esac;
23
24     next(negReq) := !request;
```

Odgovor



Napomena: na dijagramu je početno stanje prikazano čvorom s dvostrukim rubom. Varijable `request`, `status` i `negReq` su redom prikazane kao **req**, **st** i $\overline{\textbf{req}}$.

request	status	negReq
$\bot$	ready	$\bot$
$\bot$	ready	$\top$
$\bot$	busy	$\bot$
$\bot$	busy	$\top$
$\top$	ready	$\bot$
$\top$	ready	$\top$
$\top$	busy	$\bot$
$\top$	busy	$\top$

a) skup svih mogućih stanja -– S_A

request	status	negReq
$\bot$	busy	$\top$
$\top$	busy	$\top$
$\top$	ready	$\bot$
$\bot$	ready	$\bot$

b) skup svih dosezljivih stanja -– S_R

Napomena: zasivljeni retci označavaju početno stanje.