Seminarski rad

**Provjera modela**

**Profesorica:** **Student:**

dr.sc. Nina Bijedić Mirza Kurtović, IB190147

Sadržaj

[1. Uvod 3](#_Toc31328377)

[2. Prelazni sistemi 4](#_Toc31328378)

[2.1 Izvršenja 5](#_Toc31328379)

[3. Programi zavisni o podacima 5](#_Toc31328380)

[4. Istovremeni sistemi 7](#_Toc31328381)

[4.1 Nezavisni sistemi 7](#_Toc31328382)

[4.2 Djeljene varijable 8](#_Toc31328383)

[4.3 Djeljene akcije 9](#_Toc31328384)

[5. NuSMV 10](#_Toc31328385)

[6. Kontroler raskrsnice 11](#_Toc31328386)

[6.1 Varijable 12](#_Toc31328387)

[6.2 Semafori 13](#_Toc31328388)

[6.3 Raskrsnica 15](#_Toc31328389)

[6.4 Testiranje 17](#_Toc31328390)

[7. Zaključak 19](#_Toc31328391)

[Reference 20](#_Toc31328392)

[Literatura 20](#_Toc31328393)

# Uvod

U hardverskom i softverskom inženjeringu formalne metode predstavljaju vrstu matematički utemeljenih tehnika za specifikaciju, razvoj i provjeru softverskih i hardverskih sistema. Upotreba formalnih metoda za dizajn softvera i hardvera motivirana je očekivanjem da će upotreba odgovarajućih matematičkih funkcija pridonijeti pouzdanosti i robusnosti samog dizajna.[1] Jedna od formalnih metoda koja se koristi za specifikaciju softvera, hardvera je provjera modela(engl. model checking).

Provjera modela(engl. model checking) predstavlja formalnu tehniku provjere(engl. verification) koja omogućava provjeru željenih ponašanja sistema. Ovo metoda nam omogućuje verifikaciju modela sistema pomoću sistematske provjere svih mogućih stanja samog modela.[2]

Provjera modela(engl. model checking) zahtijeva model sistema kojeg provjeravamo i željene osobine tog samog sistema, potom provjera dali dati model zadovoljava željene osobine. Provjera modela predstavlja automatizovanu tehniku provjere nepostojanja grešaka, a također se može smatrati kao tehnika uklanjanja grešaka. Provjera modela se najčešće koristi u područiju hardverske verifikacije i softverskog inženjeringa.[3]

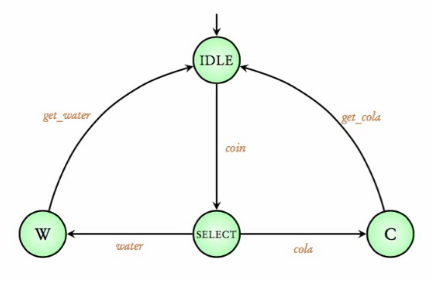
Tehnika verifikacija utemeljena na modelima temelji se na modelima koji matematički precizno i nedvosmisleno opisuju moguća ponašanja samog sistema. Prije bilo kojeg oblika verifikacije tačno modeliranje sistema često dovodi do otkrića nepotpunosti i nejasnoća u neformalnim specifikacijama sistema. Takvi se problem obično otkrivaju u mnogo kasnijoj fazi dizajna. Modeli sistema praćeni su algoritmima koji sistemski istražuju sva stanja modela sistema. To daje osnovu za čitav niz tehnika verifikacija u rasponu od provjere modela do simulacija ili testiranja u stvarnom okruženju.[4]

Prosto rečeno provjera modela je tehnika provjere svih mogućih stanja sistema. Isto kao što šahovski program provjerava moguće poteze tako i alat za provjeru modela(engl. model checker) provjerava sve moguće scenarije vezane za dati sistem. Ovako provjeravamo dali posmatrani sistem radi u sklopu nekih pravila.[5]

# Prelazni sistemi

Prelazni sistemi(engl. transition systems) se često koriste u poljima računarske nauke kako bi se opisalo ponašanje sistema. Prosto rečeno prelazni sistem predstavlja graf unutar kojeg čvorovi predstavljaju stanja(engl. state) unutar sistema a veze između tih čvorova predstavljaju prelaze(engl. transition). Stanje opisuje neke informacije o sistemu u određenom trenutku njegovog ponašanja.[6]

Na slici 1. je prikazana pojednostavljena verzija automata za prodaju. Pod stanje sistema podrazumjevamo čvorove(IDLE, SELECT, C, W), na primjer, SELECT podrazumjeva odabir, IDLE podrazumjeva čekanje korisnika(ubacivanje kovanice). Prelazi unutar ovog modela su coin, cola, water, get\_cola, get\_water.



Slika 1. Tranzicioni sistem automata za prodaju

U literaturi postoji više notacija prelaznih sistema. U ovom seminarskom će se koristiti prelazni sistemi sa nazivima akcija za prelaze i atomske prepozicije za stanja. Nazivi akcija će biti korišteni kako bi se opisali komunikacijski mehanizmi između samih procesa.[7]

Sintaksa prelaznom sistema(TS) je sljedeći:

* S predstavlja set stanja
* Act je set akcija
* −→ ⊆ S × Act × S je prelazna relacija
* I ⊆ S je set početnih stanja
* AP je set atomskih prepozicija
* L : S →2AP Predstavlja funkciju označavanja

Na slici 1. je prikazana pojednostavljena verzija automata za prodaju. U ovom modelu stanja prikazujemo kao S = {IDLE, SELECT, C, W}. Set početnih stanja(engl. initial states) sastoji se samo od jednog stanja a prikazujemo ga kao I = {IDLE}. Korisnička akcija insert\_coin predstavlja sam proces ubacivanja kovanice, dok akcije water i cola predstavljaju odabir vode i kole. Akcije get\_water i get\_cola predstavljaju dostavljanje datih proizvoda. Set akcija unutar sistema predstavljamo sa Act = {insert\_coin, get\_soda, get\_cola,water, cola}.

**Početno**(engl. initial) stanje kao što samo ime kaže podrazumjeva početak samog sistema. Unutar sistema možemo imati više početnih stanja ali minimalno jedno. U slučaju više početnih stanja jedno od njih određujemo nasumično.

Prosto rečeno **završno**(engl. terminal) stanje predstavlja prirodni fenomen i podrazumjeva završetak samog programa. Stanje s u prelaznom sistemu se naziva završno(engl. terminal) ako i samo ako Post(s) = ∅, to jeste ako stanje s ne vodi ni prema jednom drugom stanju.

## 2.1 Izvršenja

Do sada je ponašanje tranzicijskog sistema opisivano na intuitivnom nivou. Ovo će sada biti formalizirano korištenjem notacija izvršenja. Jedno izvršenje tranzicionog sistema pokazuje nam moguće ponašanje samog tranzicionog sistema.

Na osnovu slike 1. automat ima više mogućih izvršavanja, neka od tih izvršavanja su:

p1 = IDLE SELECT W IDLE SELECT W ...

p2 = IDLE SELECT W IDLE SELECT C ...

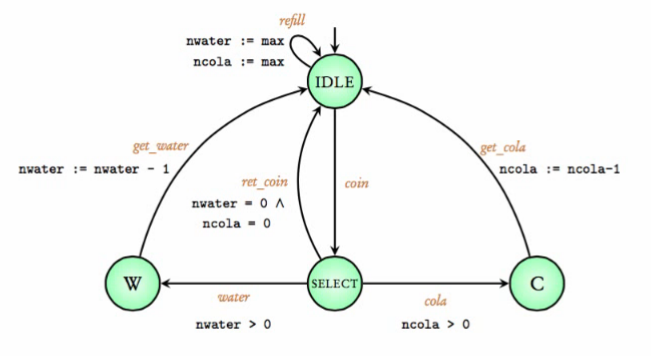
p3 = IDLE SELECT C IDLE SELECT C ...

p4 = IDLE SELECT W IDLE SELECT W ...

# Programi zavisni o podacima

Programi koji manipulišu sa podacima nazivaju se programi zavisni o podacima(engl. data-dependent programs). Programi zavisni o podacima sastoje se od varijabla bilo kojeg tipa, izjava sa uslovnim grananjem(npr. if, while petlje) i zadataka namjenjenim varijablama(npr. x:=x-1,y:=5).

Programski graf koji predstavlja automat za prodaju je prikazan na slici 2. Početno stanje sistema je IDLE. Kada korisnik ubaci kovanicu mašina prelazi u stanje SELECT. U stanju SELECT u zavisnosti od odabira korisnika mašina može preći u stanje W ili C.

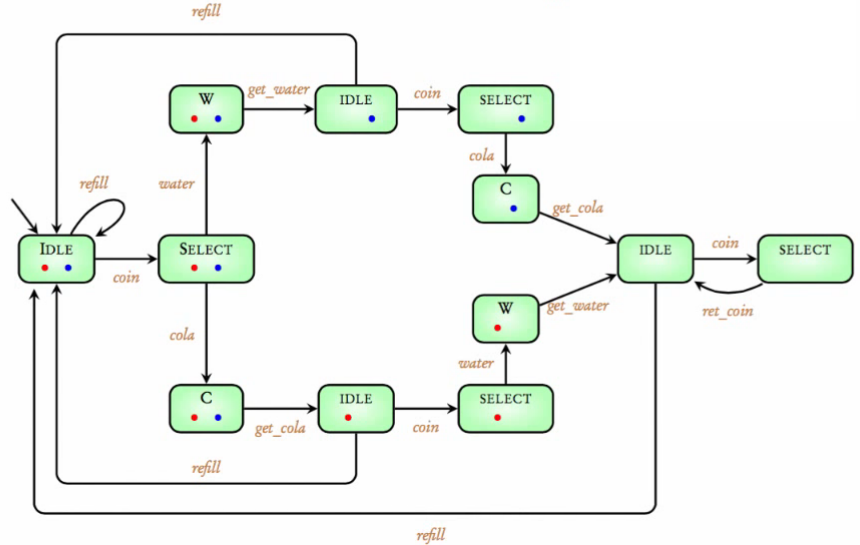


Slika 2. Programski graf automata za prodaju

Rad automata zavisi od nekoliko varijabla a to su: **nwater**, **ncola**, **max**. Varijabla max predstavlja maksimalan broj boca vode i kole u samom sistemu, a nwater i ncola predstavljaju trenutan broj istih. Inicijalno vrijednosti nwater i ncola varijabla je jednaka max. Dolazimo do pitanja kako će kod ovog automata mjenjati date varijable. Svaki put kada korisnik odabere vodu vrijednost od nwater se smanjiva za jedan(nwater:=nwater-1), isto tako kada korisnik odabere kolu vrijednost od ncola se smanjuje za jedan(ncola:=ncola-1). Kada je automat u stanju SELECT on može ići u stanje W ako i samo ako je nwater > 0 što u prevodu znači da korisnik može odabrati vodu ako je u mašini preostalu više boca vode, isto vrijedi i za kolu. U slučaju da su vrijednosti nwater i ncola jednaki nuli( nwater=0 & ncola=0) korisniku vraćamo kovanicu koju je ubacio. Akcija refil je dodano ubačena, prosto rečeno povremeno vrijednost varijabla nwater i ncola se postavlja na max(nwater=max, ncola=max). Ovakav sistem sa varijablama, uslovima, zadacima se naziva programski graf.

Tranzicioni sistem unutar svojih akcija može sadržavati samo nazive akcija a programski graf dodatno sadrži uslove(npr. x>0) i zadatke(npr. x:=x+1). Uz pomoć početnih stanja možemo kreirati tranzicioni sistem koji predstavlja rad programskog grafa sa tim početnim varijablama tj. svaki programski graf se može predstaviti kao tranzicioni sistem ako ima definisana početna stanja.

Na slici 3. je prikazan tranzicioni sistem koji koji je predstavljen sa max = 1. Broj plavih krugova unutar stanja predstavlja broj boca kole i jednak je ncola a broj crvenih krugova predstavlja broj boca vode a odgovara varijabli nwater. Početno stanje ovog automata je IDLE(lijeva strana slike), početno stanje varijabla nwater i ncola je jednako max varijabli tj. jedan. Kada korisnik ubaci kovanicu sistem prelazi u stanje W potom kada automat izbaci vodu prelazimo ponovo u IDLE stanje(gornji dio slike) ali korisnik više nema mogućnost odabira vode zato što je varijabla nwater jednaka nuli. Iz ovoga stanja korisnik jedino ima mogućnost odabira(SELECT) kole zato što je preostao samo plavi krug tj. preostala je samo jedna kola. Na kraju pošto su vrijednosti vode i kole jednake nuli sistem može samo ići u stanje IDLE gdje su obe vrijednosti nula. U slučaju da korisnik ubaci kovanicu a sistem je u ovom stanju tadu će se izvršiti akcija “ret\_coin” što znači da će se korisniku vratiti kovanica. Sistem također ima mogućnost “refill” što znači da se vrijednosti varijabla postavljaju na max.



Slika 3. Tranzicioni sistem prikazan na osnovu programskog grafa

# Istovremeni sistemi

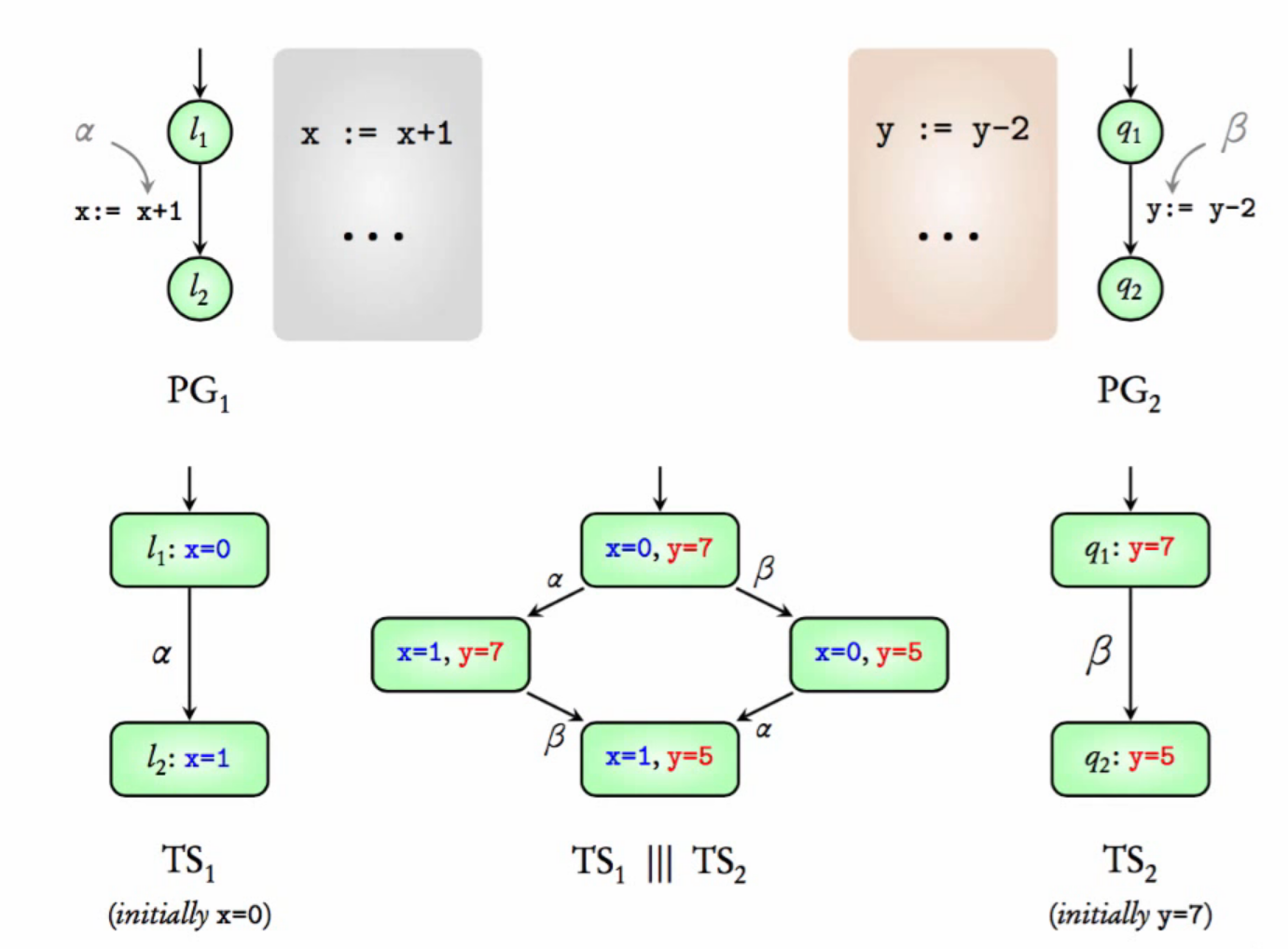
U prethodnim poglavljima predstavljeni su tranzicioni sistemi i sistemi ovisni o podacima( jednostavni sekvencijalni programi). U stvarnosti većina hardverskih i softverskih sistema nisu sukcesivne već paralelne prirode. Predpostavimo da imamo više tranzicionih sistema TS1,....,TSn. Zadatak je definisanje operatora III tako da dobijemo tranzicioni sistem TS = TS1 III ... III TSn koji specificira ponašanje paralelnog sistema tranzicijskih sistema sve od TS1 do TSn. Priroda operatora III zavisi od vrste komunikacije koja je podržana.[8]U nastavku će biti opisano nekoliko mehanizama za paralelne sisteme.

## 4.1 Nezavisni sistemi

Jedna od osnovnih paradigma paralelnog sistema jeste ispreplitanje(engl. interleaving). Ovaj sistem podrazumjeva model koji se sastoji od skupa(djelomično) nezavisnih komponenti. U suštini ovaj sistem podrazumjeva dostupnost samo jednog procesora na kojem su sve akcije međusobno povezane. Ovaj „jednoprocesorski pogled“ predstavlja koncept modeliranja i također se odvija ako se sami procesi odvijaju na više procesora. Pri tome ne postoji predpostavka o redosljedu izvršavanja različitih procesa. Ako imamo, na primjer, dva neprekidna procesa A i B koji su neovisni jedan o drugom tada imamo više mogućih sekvenci izvršavanja procesa A i B. Na primjer, kako bi isključili izvršenja gdje se samo proces A izvršava a proces B je u potpunosti ignoriše trebali bi uvesti odgovarajuća pravila koja će sprječiti ovakve vrste izvršenja. Ako nije drugačije navedeno prihvatamo sve moguće sekvence izvršavanja uključujući i ove nepravedne.

Nezavisni sistemi podložni su ideji da postoji neki planer koji međusobno povezuje korake istovremeno izvršavajućih procesa na osnovu prethodno nepoznate strategije. Ova vrsta reporezentacije u potpunosti zavisi od brzine procesa koji sudjeluju i na taj način modelira svaku moguću realizaciju od strane jednog procesorskog sistema ili nekoliko procesora s proizvoljnom brzinom.[9]

Na slici 4. je prikazan primjer isprepletenog sistema koji se sastoji od dva programa. Predpostavljamo da ova dva programa(PG1 i PG2) rade paralelno. Inicijalne vrijednosti su(x=0,y=7). Sada imamo dvije mogućnosti ili će se prvi program(PG1) izvršiti tj. varijabla x će se povećati za jedan(x:=x+1) a vrijednost varijable y će ostati ista. Druga mogućnost je da se drugi program izvrši tj. y će se smanjiti za dva (y:=y-2) a varijabla x će ostati ista. U drugom prelazu s obzirom da oba programa imaju po jedan prelaz α i β, ako je prethodno izvršena akcija α trenutno će se izvršiti β i obrnuto. Bez obzira kojim redosljednom izvršavamo akcije uvjek ćemo doći u stanje da je varijabla x jednaka jedan a varijabla y jednaka pet.

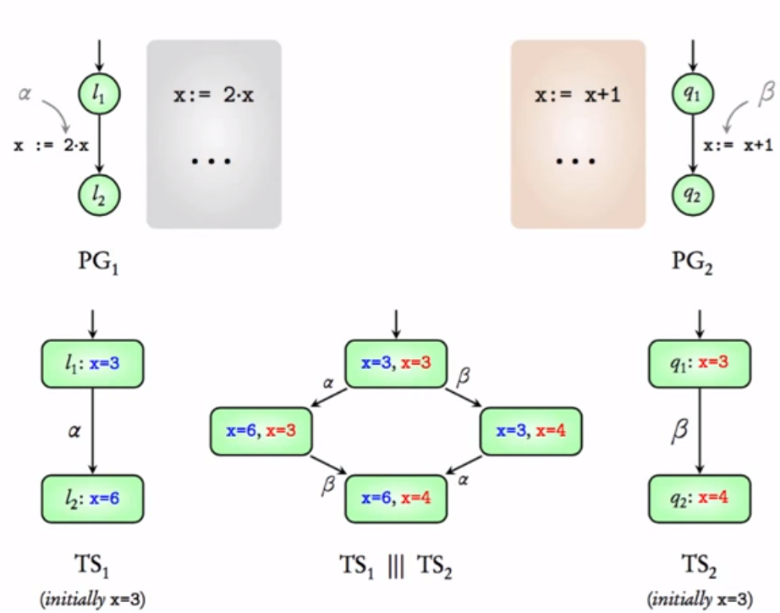


Slika 4. Programski graf nezavisnih sistema

## 4.2 Djeljene varijable

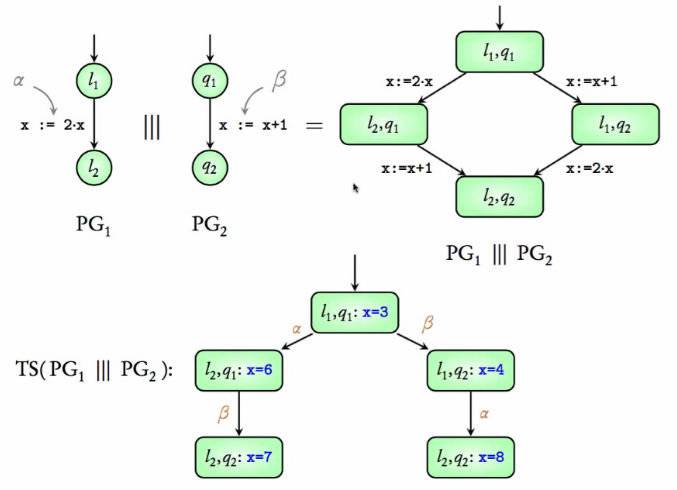
Operator III možemo koristiti kako bi modelirali asihroni istovremeni sistem kod kojeg su podprocesi potpuno neovisni jedan od drugog npr. bez bilo kojeg oblika djeljenih varijabla. Sam operator III je previše jednostavan za većinu paralelnih sistema koji imaju jednu ili više komponenti koje međusobno komuniciraju.[10]

Na slici 5. su prikazana dva programa(PG1,PG2) koji paralelno rade i djele varijablu x. Problem unutar ovog primjera je taj što akcije α i β pristupaju djeljenoj varijabli x i stoga se takmiče. Prosto rečeno operator III kod tranzicionih sistema ne uzima u obzir moguće konflikte. Kada se izvrše oba programa dobijemo da je x jednako 6 i 4 što naravno nije moguće, zbog ovoga operator III kojeg smo definisali za nezavisne sisteme ne radi kada imamo varijable koje se djele između više sistema. Kako bi rješili ovaj problem paralelnih programa sa djeljenim varijablama operator ispreplitanja bi trebali definisati na nivou programskih grafova a ne direktno na nivou tranzicionih sistema.



Slika 5. Programski graf nezavisnih sistema sa djeljenom varijablom

Sada model posmatramo kao jedan tranzicioni sistem koji se sastoji od dva programska grafa a označavamo ga kao TS(PG1 III PG2). Prethodno smo model posmatrali kao dva odvojena tranziciona sistema TS(PG1) III TS(PG2). Dato rješenje je prikazano na slici 6. U zavisnosti od redosljeda izvršavanja procesa možemo imati dva različita rješenja, varijabla x može imati vrijednost sedam ili osam.



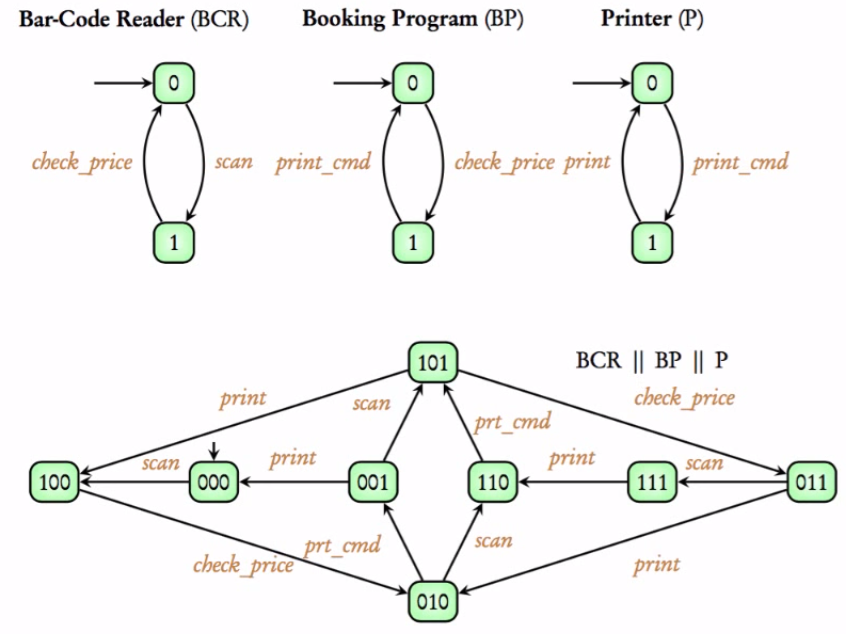
Slika 6. Tranzicioni sistem programskih grafova sa djeljenom varijablom

## 4.3 Djeljene akcije

Jedan od mehanizama za paralelne procese je ***handshake.*** Pojam *handshake* znači da istovremeni procesi koji međusobno žele komunicirati to moraju raditi sinhrono. Procesi mogu međusobno komunicirati samo ako oba istovremeno sudjeluju u toj interakciji tj. „rukuju se“. Informacije koje se razmjenjuju u toku rukovanja mogu biti različitih tipova tj. podaci mogu varirati od prostog broja, niza podataka ili složenih struktura.

Postoje dva neka pravila vezana za handshake. Na osnovu prvog pravila komponenta može izvršiti radnje koje ne podliježu rukovanju(ta komponenta nije djeljena sa nekim drugim sistemom) na potpuno nezavisan način tj. ponaša se kao nezavisna komponenta opisana u poglavlju 4.1. Drugo pravilo glasi da procesi Tsi i Tsj (i=j) moraju izvršiti svaku akciju rukovanju u Acti ∩ Actj zajedno što u prevodu znači ako dva ili više sistema imaju istu akciju ta akcija se izvršava istovremeno za sve sisteme. [11]

Na slici 7. su prikazana tri sistema koji međusobno djele neke od akcija. U suštini kod nezavisnih sistema kada jedan program dođe do djeljene akcije svaki program koji ima tu datu akciju mjenja stanje. Inicijalno stanje je 000 kada se izvrši akcija scan BCR prelazi u stanje 1 a nakon toga ima jedino mogućnost izvršenja akcije check\_price. Kada se izvrši akcija check\_price svaki program koji ima datu akciju mora promjeniti svoje stanje tako da će vrijednost programa BP biti jednaka 0 a trenutno stanje svih sistema je 010. U slučaju da neka akcija nije djeljena tj. izvršava se unutar samo jednog sistema tada samo taj trenutni program mjenja stanje, na primjer, akcija scan je povezana samo za BCR i kada se izvrši data akcija stanje iz 000 prelazi u 010.



Slika 7. Tranzicioni sistemi sa djeljenim akcijama

# NuSMV

Kao što je u prethodnim poglavljima objašnjeno provjera modela je tehnika provjere svih mogućih stanja sistema. Isto kao što šahovski program provjerava moguće poteze tako i alat za provjeru modela(engl. model checker) provjerava sve moguće scenarije vezane za dati sistem. Ovako provjeravamo dali posmatrani sistem radi u sklopu nekih pravila. Kada kreiramo programski graf i dodamo odgovarajuća pravila koja taj graf treba da poštuje tada trebamo na neki način provjerit sva moguća stanja tog odgovarajućeg grafa. Pomoću alata za provjeru modela provjeravamo sva moguća stanja samog sistema, jedan od tih alata je NuSMV.

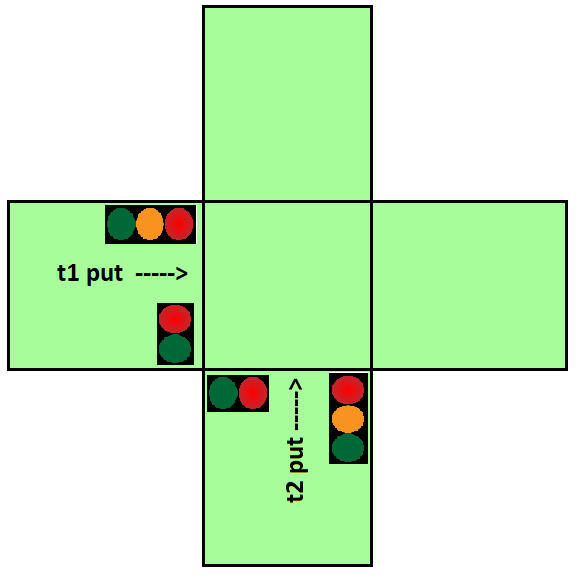
NuSMV je reimplementacija i proširenje SMV-a, prva provjera modela zasnovana na BDD-ovima. NuSMV dizajniran je kao otvorena arhitektura za provjeru modela koja se pouzdano može koristiti za verifikaciju industriskih dizajna, kao jezgra za prilagođene alate za provjeru i kao osnova za tehnike formalne verifikacije.[12]

# Kontroler raskrsnice

Raskrsnica predstavlja tačku u kojoj se spajaju dvije ili više cesta. Na slici 8. je prikazana raskrsnica u kojoj se spajaju dva puta(t1 i t2). U modelu su prikazana dva semafora(jedan semafor se sastoji od djela za automobile i djela za pješake), za svaku ulicu po jedan. Vrijednosti semafora su jednake na obe strane t1 ulice tako da nema potrebe za modeliranje oba semafora, isto vrijedi za ulicu t2.

Prilikom kreiranja kontrolera vezanog za ovu raskrsnicu u obzir treba uzet nekoliko pravila. Prva grupa pravila vezana je za samu sigurnost svih učesnika saobraćaja. Prvo sa strane sigurnosti **nikada** ne smije istovremeno biti zeleno svjetlo pješacima i automobilima na putu t1 ili t2. Drugo pravilo je također vezano za prvo a to je da **nikada** ne smije pješacima na t1 i t2 putu biti aktivno zeleno svjetlo. Isto kao i drugo pravilo **nikada** automobilima na putu t1 i t2 ne smije biti zeleno svjetlo.

Drugo vrsta pravila vezana je za živnost. Jedan semafor ne može biti aktivan više ciklusa, saobraćaj mora ići naizmjenično, na primjer, na putu t1 semafor ne smije biti aktivan više ciklusa a t2 da bude aktivan više ciklusa, isto vrijedi i za t2. Na slici 8. je prikazan model date raskrsnice.



Slika 8. Model raskrsnice

## 6.1 Varijable

Kako bi ova raskrsnica radila po planu treba da vodimo računa o nekoliko varijabla. U isječku koda 1. je prikazan dio koda u kojem deklarišemo i inicijaliziramo sve varijable potrebne za rad samog kontrolera. Varijable t1\_color i t2\_color predstavljaju dio semafora namjenjenog automobilima a mogu imati vrijednosti red, yellow, green. Varijable t1\_pedestrian\_color i t2\_pedestrian\_color predstavljaju dio semafora namjenjenog pješacima. Varijabla curr\_light predstavlja semafor koji je „trenutno“ zelene boje(objašnjeno u poglavlju 6.3).

Prosto rečeno čitav program se posmatra kao *while* petlja, program se ponavlja a svako ponavlja se naziva *tick*, a *tick* možemo posmatrati kao neku vremensku jedinicu, na primjer, minuta ili sekunda. Kako bi unutar raskrsnice imali neko vrijeme vezano za trajanje svjetla dodane su varijable: time\_left\_green, time\_left\_yellow, time\_left\_red(Unutar NuSMV jezika ne možemo definisati neku varijablu samo tipa *integer,* potrebno je dodati interval u kojem se ta varijabla smije kretati, na primjer, 10..20). Varijabla time\_left\_green podrazumjeva vrijeme koliko će trenutno svjetlo ostati zeleno isto to vrijedi i za varijablu time\_left\_yellow i time\_left\_red.

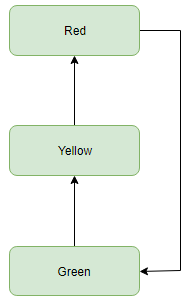
Kada se sistem prvi put pokrene varijable dobiju vrijednosti koje su definisane unutar ***ASSIGN***dijela. Semafori(namjenjeni automobilima) imaju početnu vrijednost crveno i zeleno isto tako semafori namjenjeni pješacima imaju obrnutu vrijednost. Trenutno aktivno svijetlo(curr\_light) je ono koje ima zelenu vrijednosti(t1). Preostalo vrijeme za sva svijetla se postavlja na maksimalnu vrijednost.

|  |
| --- |
| **Isječak koda: varijable** |
| 1 MODULE main  2     VAR  3         t1\_color: {red, yellow, green};  4         t1\_pedestrian\_color:{red,green};  5         t2\_color: {red, yellow, green};  6         t2\_pedestrian\_color:{red,green};  7  8         curr\_light: {t1, t2};  9         time\_left\_green: 0..50;  10        time\_left\_yellow: 0..5;  11        time\_left\_red: 0..2;  12 ASSIGN  13    init(t1\_color) := green;  14    init(t2\_color) := red;  15    init(t1\_pedestrian\_color):=red;  16    init(t2\_pedestrian\_color):=green;  17    init(curr\_light) := t1;  18    init(time\_left\_green) := 50;  19    init(time\_left\_yellow) := 5;  20    init(time\_left\_red) := 2; |

Isječak 1. Varijable kontrolera

## 6.2 Semafori

Unutar raskrsnice imamo dvije vrste semafora. Prva vrsta semafora je namjenjena automobilima i sastoji se od tri boje: crvena, žuta i zelena. Kada posmatramo pojedinačno semafor on naizmjenično mjenja boje u nekom određenom intervalu a interval promjena ne mora biti isti. Na slici 9. je prikazan model semafora, sastoji se od tri stanja a svako od tih stanja predstavlja jednu boju na semaforu. Semafor direktno prelazi iz crvenog u zeleno stanje ali kada se vraća u crveno prvo prolazi kroz žuto stanje.



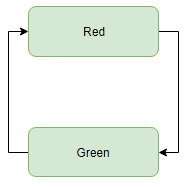
Slika 9. Tranzicioni sistem jednog semafora(automobili)

U isječku koda 2. prikazan je dio koda namjenjen za promjenu boje na semaforu. Sve varijable su detaljno objašnjene u prethodnom poglavlju. Radi se o obliku *switch* petlje a imamo četri moguća rješenja. U slučaju da je boja posmatranog semafora crvena, trenutno aktivan semafor je t2(Semafor se smatra posljednjim aktivnim ako je posljednji bio zelene boje) i vrijeme preostalo za crveno svijetlo je nula tada se boja semafora mjenja u zelenu. Ako je boja posmatranog semafora žuta i posljednji aktivni semafor na raskrsnici je trenutni i preostalo vrijeme žutog svjetla je jednako nuli onda će se boja promjeniti u crvenu. Ako je svjetlo posmatranog semafora zelene boje i vrijeme preostalo je jednako nuli onda će se boja trenutno semafora promjeniti u žutu. Na kraju u slučaju da se ni jedan od datih uslova nije ispunjen onda boja semafora ostaje ista.

|  |
| --- |
| **Isječak koda: promjena boje na semaforu(automobili)** |
| 1    next(t1\_color) := case  2        (t1\_color = red) & (curr\_light = t2) & (time\_left\_red = 0): green;  3        (t1\_color = yellow) & (curr\_light = t1) & (time\_left\_yellow = 0): red;  4        (t1\_color = green) & (time\_left\_green = 0): yellow;  5        TRUE: t1\_color;  6    esac; |

Isječak 2. Promjena boje na semaforu(automobili)

Druga vrsta semafora je namjenjena pješacima a sastoji se od dvije boje: crvena i zelena. Vrijednost ovog semafora(boja) zavisi samo od glavnog semafora ne sastoji se od nikakvih vremenskih tajmera. U slučaju da na datom djelu raskrsnice automobili imaju zeleno ili žuto svjetlo pješaci će imati crveno, a ako automobili imaju crveno svjetlo pješaci će imati zeleno. Na slici 10. je prikazan dio semafora namjenjen pješacima, crvena i zelena boja se mjenjaju naizmjenično.



Slika 10. Tranzicioni sistem jednog semafora(pješaci)

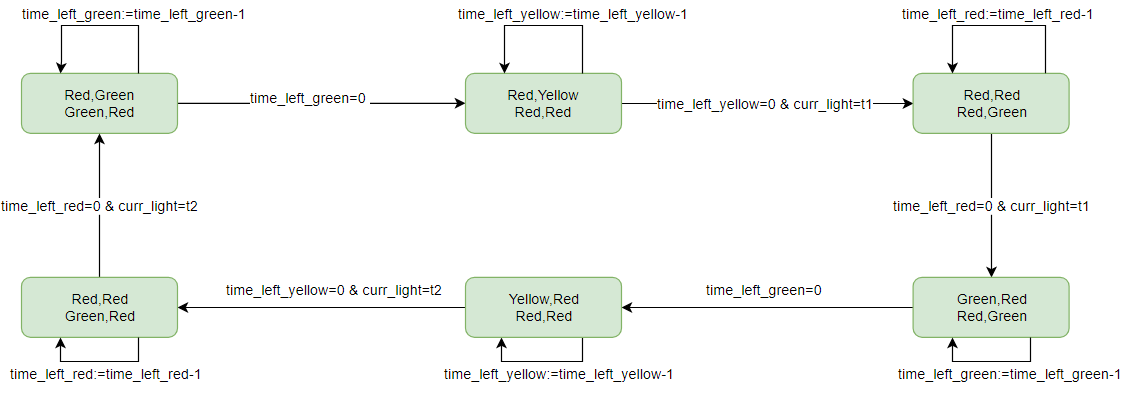
U isječku koda 3. se nalazi dio koda najmjenjen za promjenu boje semafora namjenjenog pješacima. Imamo tri moguća rezultata. U slučaju da je boja semafora(namjenjenog automobilima) crvena i preostalo vrijeme veće od jedan tada će boja semafora(namjenjenog pješacima) biti zelena. Ako je boja semafora(namjenjenog automobilima) zelena ili žuta ili je vrijeme preostalo za crveno svjetlo jednako jedan tada će boja biti crvena. Ako se ne ispuni ni jedan od ova dva uslova boja semafora ostaje ista.

|  |
| --- |
| **Isječak koda: promjena boje na semaforu(pješaci)** |
| 1     next(t1\_pedestrian\_color):=case  2     (t1\_color=red) & (time\_left\_red > 1):green;  3     (t1\_color=green) | (t1\_color=yellow) | (time\_left\_red = 1):red;  4     TRUE:t1\_pedestrian\_color;  5     esac;  6  7  8     next(t2\_pedestrian\_color) :=case  9     (t2\_color=red) & (time\_left\_red > 1):green;  10    (t2\_color=green) | (t2\_color=yellow) | (time\_left\_red = 1):red;  11    TRUE:t1\_pedestrian\_color;  12    esac; |

Isječak 3. Promjena boje na semaforu(pješaci)

## 6.3 Raskrsnica

Na slici 11. se nalazi programski graf raskrsnice. Unutar grafa je prikazano šest mogućih stanja. Kada je jedan semafor(t1 ili t2) „aktivan“ drugi semafor ima crvenu boju. U gornjem dijelu stanja(zeleni praugaonici) su prikazani semafori t1 i t2 namjenjeni automobilima, a u donjem dijelu su semafori t1 i t2 namjenjeni pješacima. Unutar svakog stanja, nezavisno od boje svjetla, svaki ciklus vrijednost time\_left parametara se smanjuje za jedan. Kada vrijednost time\_left parametra bude jednaka nula program prelazi u sljedeće stanje. U zavisnosti od semafora namjenjenih automobilima vrijednost semafora namjenjenih pješacima se mjenja.



Slika 11. Programski graf raskrsnice

Unutar isječka koda 4. se nalazi dio koda koji mjenja varijable u svakoj iteraciji. Od 1 do 12 linije nalazi se dio za mjenjanje boje samih semafora t1 i t2 i detaljno je objašnjen u poglavlju 6.2.

Kako je prethodno objašnjeno varijabla curr\_light mjenja vrijednost sa t1 na t2 i obrnuto.Kada vrijednost parametra time\_left\_red bude jednaka nuli tada se smatra da je jedan ciklus završen tj. Jedan semafor je prošo kroz sve boje( zelena, žuta, crvena). Kada se završi ciklus vrijednost curr\_light se mjenja sa t1 na t2 I obrnuto.

Od 18 do 27 linije koda unutar isječka 4. nalazi se dio za mjenjanje boje semafora(namjenjenih pješacima). Vrijednost varijabla t1\_pedestrian\_color i t2\_pedestrian\_color zavisi od glavnih semafora. Sam proces promjene pješačkih semafora je detaljnije objašnjen u poglavlju 6.2.

|  |
| --- |
| **Isječak koda: Kontroler** |
| --  ...  1    next(t1\_color) := case  2        (t1\_color = red) & (curr\_light = t2) & (time\_left\_red = 0): green;  3        (t1\_color = yellow) & (curr\_light = t1) & (time\_left\_yellow = 0): red;  4        (t1\_color = green) & (time\_left\_green = 0): yellow;  5        TRUE: t1\_color;  6    esac;  7    next(t2\_color) := case  8        (t2\_color = red) & (curr\_light = t1) & (time\_left\_red = 0): green;  9        (t2\_color = yellow) & (curr\_light = t2) & (time\_left\_yellow = 0): red;  10        (t2\_color = green) & (time\_left\_green = 0): yellow;  11        TRUE: t2\_color;  12    esac;  13    next(curr\_light) := case  14        (curr\_light = t1) & (time\_left\_red = 0): t2;  15        (curr\_light = t2) & (time\_left\_red = 0): t1;  16        TRUE: curr\_light;  17    esac;  18    next(t1\_pedestrian\_color):=case  19    (t1\_color=red) & (time\_left\_red > 1):green;  20    (t1\_color=green) | (t1\_color=yellow) | (time\_left\_red = 1):red;  21    TRUE:t1\_pedestrian\_color;  22    esac;  23    next(t2\_pedestrian\_color) :=case  24    (t2\_color=red) & (time\_left\_red > 1):green;  25    (t2\_color=green) | (t2\_color=yellow) | (time\_left\_red = 1):red;  26    TRUE:t1\_pedestrian\_color;  27    esac;  28    next(time\_left\_green) := case  29        (time\_left\_green > 0):time\_left\_green - 1;  30        (time\_left\_red = 0): 50;  31        TRUE: time\_left\_green;  32    esac;  33    next(time\_left\_yellow) := case  34        (time\_left\_yellow > 0) & (time\_left\_green = 0): time\_left\_yellow - 1;  35        (time\_left\_red = 0): 5;  37        TRUE: time\_left\_yellow;  38    esac;  39    next(time\_left\_red) := case  40        (time\_left\_red > 0) & (time\_left\_yellow = 0): time\_left\_red - 1;  41        (time\_left\_red = 0): 2;  42        TRUE: time\_left\_red;  43    esac; |

Isječak 4. Kontroler semafora

## 6.4 Testiranje

Tranzicioni sistem zadovoljava sve zahtijeve samo kada sva njegova izvršenja zadovoljavaju zahtijeve. U prethodnim poglavljima bila su specificirana neka pravila kako bi sama raskrsnica bila sigurna.

Kako bi prvo mogli testirati to dali ova raskrsnica zadovoljava određena pravila prvo moramo naći sva moguća stanja ovog kontrolera. Unutar NuSMV-a postoji jednostavna komanda **print\_reachable\_states** koja nam pokazuje broj mogućih stanja, u ovom slučaju broj mogućih stanja iznosi 117, a kako bi dobili listu svih mogućih stanja na datu komadu dodajemo dodatak -v i dobijamo listu svih mogućih stanja kao što je prikazano na slici 12.[13]

Simuliranje rada kontrolera se izvodi pomoću komande **simulate**, ako želimo omogućiti korisniku da interaktivno bira stanja korak po korak dodajemo nastavak **-i** a na kraju biramo broj koraka pomoću dodatka **-k steps**, na primjer, **simulate -i -k 10**. Kako bi mogli simulirati rad kontrolera prethodno je potrebno odabrati početno stanje pomoću komande **pick\_state -i**.[14]

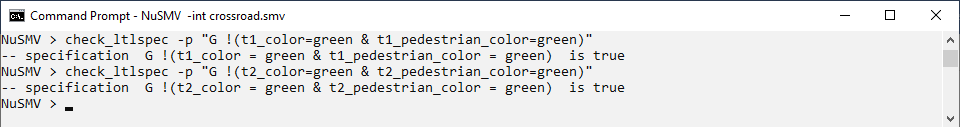


Slika 12. Pregled svih mogućih stanja modela unutar NuSMV-a

Unutar NuSMV-a postoje komanda za provjeru **check\_ltlspec**.Ova komanda ima dva nastavka, prvi nastavak **G** provjerava ispunjenost zadanih uslova na globalnom nivou. Jedno izvršenje zadovoljava uslov samo ako sva njegova stanja zadovoljavaju taj dati uslov. Tranzicioni sistem zadovoljava **G(izraz)** ako i samo ako sva njegova izvršenja zadovoljavaju **G(izraz)**.

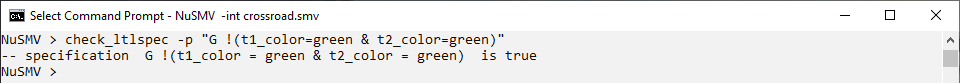
Komanda **F** ima značenje budućnost(engl. future) i provjera ispunjenost uslova. Neko izvršenje zadovoljava **F(izraz)** ako se izraz ispunjava u nekom od stanja. Tranzicioni sistem zadovoljava **F(izraz)** ako i samo ako sva njegova izvršenja zadovoljavaju **F(izraz).**[15]

Kako bi testirali dali ova raskrsnica zadovoljava navedena pravila koristit ćemo **G** nastavak. Prvo pravilo sa strane sigurnosti bilo je da na nekoj ulici t1 ili t2 pješaci i automobili nemaju istovremeno zeleno svjetlo. Ako je t1\_color = green i t1\_pedestrian\_color = green onda naš kod ne zadovoljava dato pravilo. Pomoću komande check\_ltlspec provjeravamo dali postoji stanje u kojem su istovremeno semafori za pješake i automobile imaju zeleno svjetlo. Kada unesemo potrebnu komandu dobijemo rezultat true što znači da ne postoji stanje unutar kojeg je dati uslov tačan. Istom komandom je provjereno stanje za t2 ulicu i dobijen je također rezultat true.



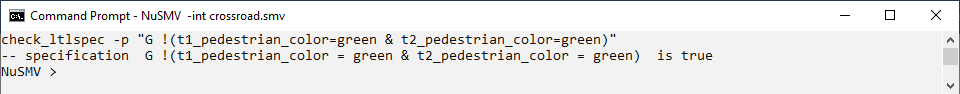
Slika 13. Testiranje sigurnosnog prafila(pješaci i automobili)

Drugo pravilo je vezano za sigurnost automobila a znači to da u isto vrijeme ne smije biti zeleno svjetlo na ulicima t1 i t2. Ako je t1\_color = green i t2\_color = green onda naš kod ne zadovoljava pravilo sa aspekta sigurnosti. Na slici 14. je prikazano izvršenje iste komande kao u prethodnom pravilu(check\_ltlspec). Izvršenjem komande dobijemo rezultat true što znači da ne postoji niti jedno stanje u kojem je dati uslov tačan.



Slika 14. Testiranje sigurnosnog pravila(automobili)

Treće pravilo je vezano za pješake a znači da ne smije na ulici t1 i t2 pješacima istovremeno biti zeleno svjetlo. Ako je t1\_pedestrian\_color = green i t2\_pedestrian\_color = green onda naš kod ne zadovoljava pravilo sa aspekta sigurnosti. Radi se o sličnom pravilu/uslovu kao dva prethodna. Na osnovu slike 15. zaključujemo da je uslov ispunjen tj. dobijamo rezultat true a to znači da ne postoji niti jedno stanje gdje su t1\_pedestrian\_color = green i t2\_pedestrian\_color = green.



Slika 15. Testiranje sigurnosnog pravila(pješaci)

Druga vrsta pravila vezana je za živnost tj. jedan semafor ne može biti aktivan više ciklusa, saobraćaj mora ići naizmjenično. Varijabla curr\_light ima vrijednosti t1 ili t2, svaki put kada se završi ciklus vrijednost se mjenja sa t1 na t2 i obrnuto, sa ovim osiguravamo da se t1 ili t2 ne mogu ponoviti dva ili više puta zaredom.

# Zaključak

U hardverskom i softverskom inženjeringu formalne metode predstavljaju vrstu matematički utemeljenih tehnika za specifikaciju, razvoj i provjeru softverskih i hardverskih sistema.

Provjera modela(engl. model checking) predstavlja formalnu tehniku provjere(engl. verification) koja omogućava provjeru željenih ponašanja sistema. Ovo metoda nam omogućuje verifikaciju modela sistema pomoću sistematske provjere svih mogućih stanja samog modela.

Osnova provjere modela je sami model tj. prelazni sistemi(engl. transition systems). Prelazni sistem se često koriste u poljima računarske nauke kako bi se opisalo ponašanje sistema. Prosto rečeno prelazni sistem predstavlja graf unutar kojeg čvorovi predstavljaju stanja(engl. state) unutar sistema a veze između tih čvorova predstavljaju prelaze(engl. transition). Stanje opisuje neke informacije o sistemu u određenom trenutku njegovog ponašanja. Zavisno od tranzicionog sistema on može imati jedno ili više mogućih izvršenja a u većini slučajeva sistem ima više mogućih izvršenja. Jedno izvršenje tranzicionog sistema pokazuje nam moguće ponašanje samog tranzicionog sistema.

Tranzicioni sistem je sam po sebi ograničen sa prostim prelazima između stanja kako bi omogućili kreiranje kompleksnijih sistema potrebno je omogućiti upravljanje varijabla unutar samog tranzicionog sistema. Programi koji manipulišu sa podacima nazivaju se programi zavisni o podacima(engl. data-dependent programs). Programi zavisni o podacima sastoje se od varijabla bilo kojeg tipa, izjava sa uslovnim grananjem(npr. if, while petlje) i zadataka namjenjenim varijablama(npr. x:=x-1,y:=5).

U stvarnosti većina hardverskih i softverskih sistema nisu sukcesivne već paralelne prirode radi toga je potrebno kreiranje načina komunikacije između paralelnih procesa. Prvi i najjednostavniji tip paralelnih sistema su neovisni sistemi a podrazumjevaju postojanje više procesa koji paralelno rade ali ne djele nikakve varijable, ovaj sistem se još naziva jednoprocesorski zato što sami procesi djele „procesor“. Drugi tip paralelnih sistema su sistemi sa djeljenim varijabla, kako bi sistem funkcionisao oba procesa posmatramo kao jedan programski graf a ne odvojeno. Treći tip paralelnih sistema su sistemi sa djeljenim akcija. Jedan od mehanizama za paralelne procese je *handshake****.*** Pojam *handshake* znači da istovremeni procesi koji međusobno žele komunicirati to moraju raditi sinhrono. Procesi mogu međusobno komunicirati samo ako oba istovremeno sudjeluju u toj interakciji tj. „rukuju se“. Informacije koje se razmjenjuju u toku rukovanja mogu biti različitih tipova tj. podaci mogu varirati od prostog broja, niza podataka ili složenih struktura.

Alat za provjeru modela(engl. model checker) provjerava sve moguće scenarije vezane za dati sistem. Ovako provjeravamo dali posmatrani sistem radi u sklopu nekih pravila. Kada kreiramo programski graf i dodamo odgovarajuća pravila koja taj graf treba da poštuje tada trebamo na neki način provjerit sva moguća stanja tog odgovarajućeg grafa. Pomoću alata za provjeru modela provjeravamo sva moguća stanja samog sistema, jedan od tih alata je NuSMV I korišten je za provjeru kontrolera raskrsnice koji je objašnjen u prethodnim poglavljima.

**Isječci**

[Isječak 1. Varijable kontrolera 12](#_Toc31328447)

[Isječak 2. Promjena boje na semaforu(automobili) 13](#_Toc31328448)

[Isječak 3. Promjena boje na semaforu(pješaci) 14](#_Toc31328449)

[Isječak 4. Kontroler semafora 16](#_Toc31328450)

**Slike**

[Slika 1. Tranzicioni sistem automata za prodaju 4](#_Toc31328455)

[Slika 2. Programski graf automata za prodaju 5](#_Toc31328456)

[Slika 3. Tranzicioni sistem prikazan na osnovu programskog grafa 6](#_Toc31328457)

[Slika 4. Programski graf nezavisnih sistema 8](#_Toc31328458)

[Slika 5. Programski graf nezavisnih sistema sa djeljenom varijablom 8](#_Toc31328459)

[Slika 6. Tranzicioni sistem programskih grafova sa djeljenom varijablom 9](#_Toc31328460)

[Slika 7. Tranzicioni sistemi sa djeljenim akcijama 10](#_Toc31328461)

[Slika 8. Model raskrsnice 11](#_Toc31328462)

[Slika 9. Tranzicioni sistem jednog semafora(automobili) 13](#_Toc31328463)

[Slika 10. Tranzicioni sistem jednog semafora(pješaci) 14](#_Toc31328464)

[Slika 11. Programski graf raskrsnice 15](#_Toc31328465)

[Slika 12. Pregled svih mogućih stanja modela unutar NuSMV-a 17](#_Toc31328466)

[Slika 13. Testiranje sigurnosnog prafila(pješaci i automobili) 18](#_Toc31328467)

[Slika 14. Testiranje sigurnosnog pravila(automobili) 18](#_Toc31328468)

[Slika 15. Testiranje sigurnosnog pravila(pješaci) 18](#_Toc31328469)

# Reference

1. R.W. Butler. What is formal method?. 06. Oktobar 2001.
2. Christel Baier, Joost-Pieter Katoen. Principles of model checking. 25. April 2008. str 14.
3. Christel Baier, Joost-Pieter Katoen. Principles of model checking. 25. April 2008. str 16.
4. Christel Baier, Joost-Pieter Katoen. Principles of model checking. 25. April 2008. str 26.
5. Christel Baier, Joost-Pieter Katoen. Principles of model checking. 25. April 2008. str 27.
6. Christel Baier, Joost-Pieter Katoen. Principles of model checking. 25. April 2008. str 19.
7. Christel Baier, Joost-Pieter Katoen. Principles of model checking. 25. April 2008. str 20.
8. Christel Baier, Joost-Pieter Katoen. Principles of model checking. 25. April 2008. str 35-36.
9. Christel Baier, Joost-Pieter Katoen. Principles of model checking. 25. April 2008. str 36-39.
10. Christel Baier, Joost-Pieter Katoen. Principles of model checking. 25. April 2008. str 39-47.
11. Christel Baier, Joost-Pieter Katoen. Principles of model checking. 25. April 2008. str 47-53.
12. Online [<http://nusmv.fbk.eu/>]
13. Online[<http://nusmv.fbk.eu/NuSMV/userman/v21/nusmv_4.html>]
14. Online[<http://nusmv.fbk.eu/NuSMV/userman/v11/html/nusmv_31.html>]
15. Online[<http://nusmv.fbk.eu/NuSMV/progman/v2/html/check_ltlspecCmd.html>]

# Literatura

1. Christel Baier, Joost-Pieter Katoen. Principles of model checking. 25. April 2008.
2. NuSMV 2.6 User Manual Online[<http://nusmv.fbk.eu/NuSMV/userman/v26/nusmv.pdf>]
3. Izvor koda Online[<https://github.com/mirzakurtovic1/crossroad_controler>]