Verifikacija softvera

— Proveravanje modela —

Milena Vujošević Janičić

www.matf.bg.ac.rs/~milena

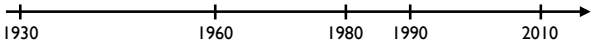
Matematički fakultet, Univerzitet u Beogradu

Pregled

Pregled

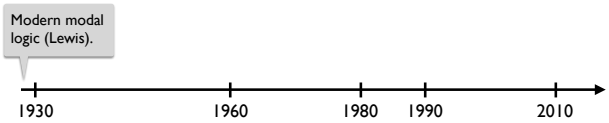
Kratka istorija

A brief history of model checking

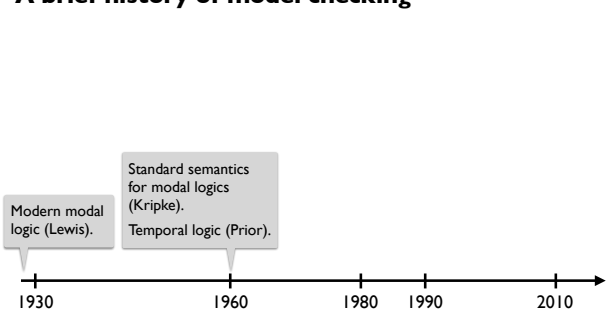


Kratka istorija

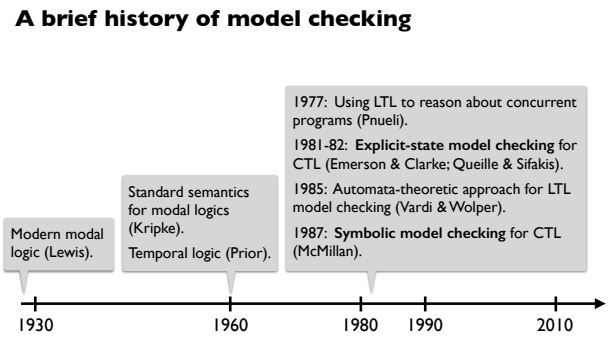
A brief history of model checking



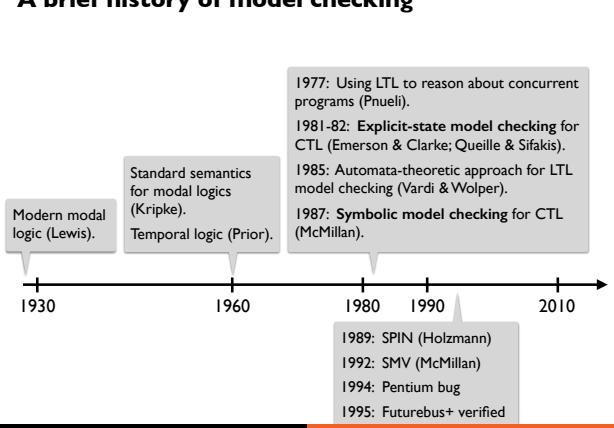
Kratka istorija



Kratka istorija

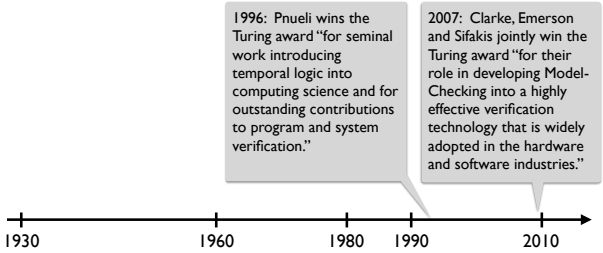


Kratka istorija



Kratka istorija

A brief history of model checking



O proveravanju modela

Poreklo ideje

- Amir Pnueli: The Temporal Logic of Programs FOCS 1977: 46–57
- E.M.Clarke, E.A.Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic (1982)
- J.P.Queille. J. Sifakis. Specification and verification of concurent systems in CESAR (1982)

O proveravanju modela

Tjuringova nagrada 1996.

A. Pnueli je dobitnik Tjuringove nagrade zbog uvođenja temporalne logike u računarstvo i za izuzetne doprinose verifikaciji programa i sistema.

Tjuringova nagrada 2007.

E.M.Clarke, E.A.Emerson i J. Sifakis su dobitnici Tjuringove nagrade zbog svojih doprinosa zasnivanju i razvoju proveravanja modela — veoma efikasne verifikacijske tehnike koja je široko prihvaćena u industriji hardvera i softvera.

O proveravanju modela

Proveravanje modela

- Tehnika verifikacije zasnovana na sistematskom ispitivanju svih mogućih putanja u izvršavanju nekog sistema
- Ispituje da li sistem ispunjava neko zadato svojstvo (invarijantu, sigurnosno svojstvo, i sl.)
- Zasniva se na preciznom formalnom opisu modela sistema, kao i svojstava koja se proveravaju.
- Oslanja se na matematičku logiku, teoriju formalnih jezika, teoriju grafova
- Primenjiva na hardverske sisteme, konkurentne sisteme i komunikacione protokole, kao i na softver

Verifikacija hardvera

Troškovi grešaka u hardveru

Proizvodnja novog hardvera je izuzetno skup proces. Popravljanje grešaka u hardveru nakon isporuke ima veoma visoke troškove. Greške u softveru se mogu ispravljati sa zakrpama i update-ovima, dok ispravljanje grešaka u hardveru zahteva ponovnu proizvodnju i ponovnu isporuku prozvoda.

Na primer, zamena Intelovog Pentium II procesora (problematična implementacija operacije deljenja u pokretnom zarezu) je uzrokovala gubitak od 475 miliona dolara (1994. godine).

Zbog toga, **dizajn čipa obično iznosi oko 27% ukupnog vremena, dok se ostatak posvećuje detekciji i prevenciji grešaka.** Ali, nekada ni to nije dovoljno...

Verifikacija hardvera

Procena štete: 17 milijardi dolara!

19. avgusta **2016**. godine pušten je u prodaju Samsung Galaxy Note 7. 2. septembra je telefon suspendovan (u Americi), a 15. septembra povučen iz prodaje jer je litijumska baterija povremeno mogla da razvije visoku temperaturu i da se zapali. Najpre je pokušano sa zamenama baterija, ali kako to nije pomoglo, 10. oktobra je obustavljena proizvodnja i aparat je povučen u potpunosti iz prodaje (u celom svetu). **Zanimljivost**: Korisnicima ovog uređaja na svim aerodromima je bio zabranjen ulazak u avion sa telefonom (slika levo). Samsung je na nekim aerodromima bio omogućio preuzimanje i zamenu telefona pre leta.

Verifikacija hardvera

Emulacija, simulacija i fabričko testiranje

- Emulacija koristi generički hardver (emulator) koji se konfiguriše tako da se ponaša kao kolo koje se razmatra i onda se tako konfigursan hardver intenzivno testira (tj proverava se da li se ponaša u skladu sa specifikacijom čipa).
- Kod simulacije, model kola koji se razmatra se konstruiše koristeći neki od jezika za opisivanje hardvera (npr Verilog ili VHDL) i onda se taj model testira.

Verifikacija hardvera

Emulacija, simulacija i fabričko testiranje

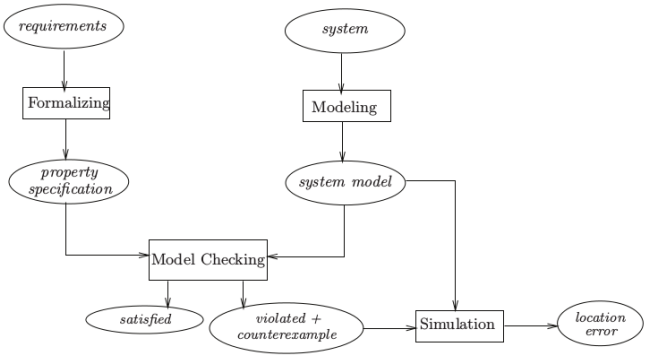
- Testiranje i kod emulacije i kod simulacije može da bude sa nekim određenim ulazima ili sa automatski generisanim ulazima, npr sa random generisanim ulazima.
- Testiranje, kao i kod softvera, ima preveliki broj mogućih ulaza da bi ponašanje hardvera moglo da se u potpunosti istraži na taj način.
- Fabričko testiranje odgovara traženju grešaka koje nastaju u procesu proizvodnje.

Verifikacija konkurentnih sistema

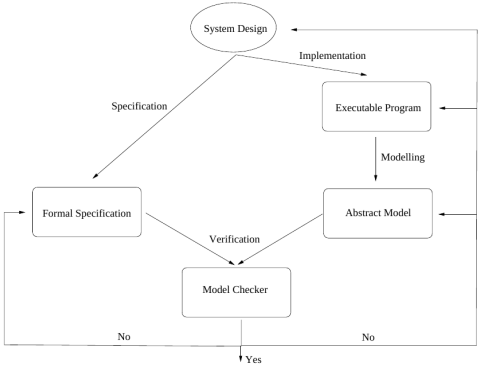
Konkurentnost

- Konkurentni sistemi su svuda
- Grešake u konkurentnim sistemima se mogu ispoljavati nedeterministički
- To znači da sa istim test primerom, sistem nekada može da se ponaša ispravno a nekada neispravno
- Razumevanje konkurentnih grešaka je često veoma teško
- Posebno su bitne greške mrtvih/živih petlji (deadlock/livelock) i izgladnjivanja (lockout)

Strukturni prikaz — sistemi uopšte



Strukturni prikaz — softver



Strukturni prikaz

Osnovni pojmovi

- Pravljenje modela (modelovanje): proces formulisanja apstraktnog modela sistema na osnovu konkretne implementacije
  - Apstraktni model: tranzicioni sistem (skup stanja i relacija prelaska nad tim skupom)
- Formalna specifikacija: precizan opis svojstava koje apstraktni model treba da zadovolji (na jeziku matematičke logike)
- Proveravač modela: alat koji automatski proverava da li apstraktni model zadovoljava formalnu specifikaciju
- U slučaju da ne zadovoljava, proveravač daje kontraprimer u vidu putanje u modelu koja narušava zadato svojstvo
- Kontraprimer se koristi za otkrivanje greške, korekciju modela i/ili specifikacije

Osnovni pojmovi

Modelovanje i model sistema

Model sistema se obično automatski generiše iz programskog jezika ili iz jezika za opisivanje hardvera. Specifikacija opisuje šta sistem treba/ne treba da radi dok model adresira ponašanje sistema. Proveravanje modela ispituje sva relevantna stanja sistema kako bi se proverilo da li ona zadovoljavaju željena svojstva.

Disclaimer

Svaka verifikacija koja koristi tehnike zasnovane na modelima je onoliko dobra koliko je dobar i model sistema!

Osnovni pojmovi

Formalna specifikacija

Potrebno je opisati ponašanje sistema matematički precizno i nedvosmisleno — dešava se da i pre same verifikacije, tj već u fazi definisanja formalne specifikacije, otkriju nekompletnosti, dvosmislenosti i nekonzistentnosti neformalne specifikacije sistema.

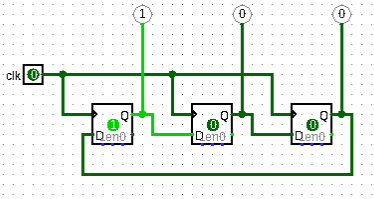
Na primer, formalizacija svih osobina podskupa ISDN (engl. *Integrated Services Digital Network*) korisničkog protokola (digitalna telefonska tehnologija nastala 70tih godina) je otkrila da je 55% originalnih neformalnih zahteva sistema bilo nekonzistentno!

Osnovni pojmovi

Proveravač modela

Proveravač modela automatski istražuje sva moguća stanja sistema koristeći neki algoritam za proveravanje modela ili grubom silom. Broj mogućih stanja koja su dostupna ovom tehnikom je značajno povećan prethodnih godina i od 10^8– 10^9 povećan je (korišćenjem pametnih algoritama i struktura podataka) na 10^{20} pa čak i u nekim primerima do 10^{476}!

Primer 1



Primer (hardver)

- Dat je trobitni ciklični registar sa početnom vrednošću 100
- Dokazati svojstvo: „U svakom trenutku je tačno jedan bit registra uključen"

Model

- Stanja: $xyz \in \{0,1\}^3$
- Relacija prelaska: $xyz \longrightarrow zxy$
- Početno stanje: 100
- Dostižna stanja: 100, 010, 001 (odavde svojstvo direktno sledi)

Primer 2

```
while(true)
{
  x = (x+1) % 3;
  if(x == 0)
   y = -y;
  z = z*y + 1;
}
```

Primer (softver)

- Dat je C-kod, sa početnim vrednostima: $x = 0, y = 1, z = 1$
- Dokazati svojstvo: „Kada god je $y < 0$, u nekom od narednih koraka biće $y > 0$"

Model

- Stanje su određena vrednostima promenljivih i trenutnom pozicijom u kodu
- Početno stanje je određeno početnim vrednostima promenljivih i ulaznom tačkom programa
- Relacija prelaska je određena semantikom naredbi programa
- Zadato svojstvo nije vezano za pojedinačno stanje, već za odnose među stanjima

Pregled

Tranzicioni sistem — Kripke struktura

Kripke struktura

Kripke struktura je vrsta tranzicionog sistema koji je originalno razvio Saul Kripke a koja se koristi u proveravanju modela za predstavljanje ponašanja sistema. Neformalno, to je graf čiji su čvorovi dostupna stanja sistema a grane predstavljaju prelaske između stanja sistema. Funkcija obeležavanja (labeliranja, mapiranja) je funkcija koja preslikava svaki čvor u skup osobina koje važe u odgovarajućem stanju. Originalno, Kripke struktura je uređena četvorka.

Kripke strukture

Kripke structures

A Kripke structure is a tuple $M = \langle S, S_0, R, L \rangle$

Kripke strukture

Kripke structures

A Kripke structure is a tuple $M = \langle S, S_0, R, L \rangle$

- S is a finite set of states.

Kripke strukture

Kripke structures

A Kripke structure is a tuple $M = \langle S, S_0, R, L \rangle$

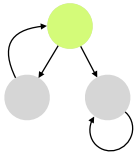
- S is a finite set of states.
- $S_0 \subseteq S$ is the set of initial states.

Kripke strukture

Kripke structures

A Kripke structure is a tuple $M = \langle S, S_0, R, L \rangle$

- S is a finite set of states.
- $S_0 \subseteq S$ is the set of initial states.
- $R \subseteq S \times S$ is the transition relation, which must be *total*.

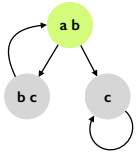


Kripke strukture

Kripke structures

A Kripke structure is a tuple $M = \langle S, S_0, R, L \rangle$

- S is a finite set of states.
- $S_0 \subseteq S$ is the set of initial states.
- $R \subseteq S \times S$ is the transition relation, which must be *total*.
- $L : S \to 2^{AP}$ is a function that *labels* each state with a set of *atomic propositions* true in that state.

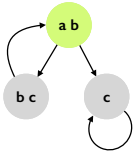


Kripke strukture

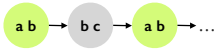
Kripke structures

A Kripke structure is a tuple $M = \langle S, S_0, R, L \rangle$

- S is a finite set of states.
- $S_0 \subseteq S$ is the set of initial states.
- $R \subseteq S \times S$ is the transition relation, which must be *total.*
- $L : S \to 2^{AP}$ is a function that *labels* each state with a set of *atomic propositions* true in that state.



A *path* in M **is an infinite sequence of states** $\pi = s_0 s_1 \ldots$ **such that for all** $i \geq 0$, $(s_i, s_{i+1}) \in R$**.**



Kripke strukture

Modeling systems with Kripke structures

```
// x==1, y==1
x := (x + y) % 2
```

- In a finite-state program, *system variables* V range over a *finite domain* D: V = {x, y} and D = {0, 1}.
- A *state* of the system is a *valuation* s : V → D.

Kripke strukture

Modeling systems with Kripke structures

```
// x==1, y==1
x := (x + y) % 2
```

$S = (x = 0 \vee x = 1) \wedge (y = 0 \vee y = 1)$
$S_0 = (x = 1) \wedge (y = 1)$
$R(x, y, x', y') = (x' = (x + y) \% 2) \wedge (y' = y)$

- In a finite-state program, *system variables* V range over a *finite domain* D: V = {x, y} and D = {0, 1}.
- A *state* of the system is a *valuation* s : V → D.
- Use FOL to describe the (initial) states and the transition relation.

Kripke strukture

Modeling systems with Kripke structures

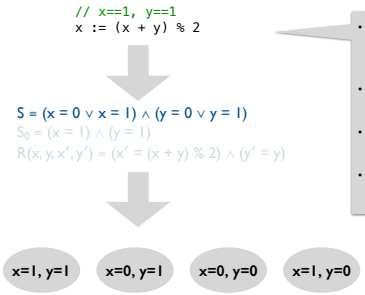
```
// x==1, y==1
x := (x + y) % 2
```

$S = (x = 0 \vee x = 1) \wedge (y = 0 \vee y = 1)$
$S_0 = (x = 1) \wedge (y = 1)$
$R(x, y, x', y') = (x' = (x + y) \% 2) \wedge (y' = y)$

- In a finite-state program, *system variables* V range over a *finite domain* D: V = {x, y} and D = {0, 1}.
- A *state* of the system is a *valuation* s : V → D.
- Use FOL to describe the (initial) states and the transition relation.
- Extract a Kripke structure from the FOL description.

Kripke strukture

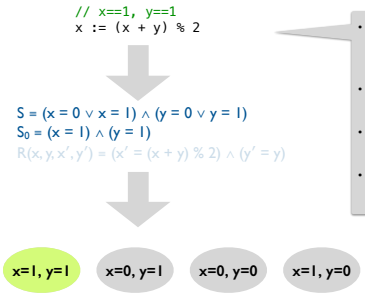
Modeling systems with Kripke structures



- In a finite-state program, *system variables* V range over a *finite domain* D: V = {x, y} and D = {0, 1}.
- A *state* of the system is a *valuation* s : V → D.
- Use FOL to describe the (initial) states and the transition relation.
- Extract a Kripke structure from the FOL description.

Kripke strukture

Modeling systems with Kripke structures



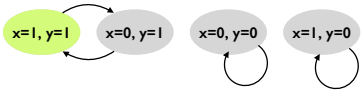
- In a finite-state program, *system variables* V range over a *finite domain* D: V = {x, y} and D = {0, 1}.
- A *state* of the system is a *valuation* s : V → D.
- Use FOL to describe the (initial) states and the transition relation.
- Extract a Kripke structure from the FOL description.

Kripke strukture

Modeling systems with Kripke structures

```
// x==1, y==1
x := (x + y) % 2
```

$S = (x = 0 \vee x = 1) \wedge (y = 0 \vee y = 1)$
$S_0 = (x = 1) \wedge (y = 1)$
$R(x, y, x', y') = (x' = (x + y) \% 2) \wedge (y' = y)$



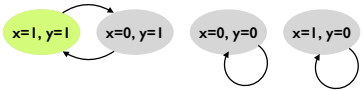
- In a finite-state program, *system variables* V range over a *finite domain* D: V = {x, y} and D = {0, 1}.
- A *state* of the system is a *valuation* s : V → D.
- Use FOL to describe the (initial) states and the transition relation.
- Extract a Kripke structure from the FOL description.

Kripke strukture

Modeling systems with Kripke structures

```
// x==1, y==1
x := (x + y) % 2
```

$S = (x = 0 \vee x = 1) \wedge (y = 0 \vee y = 1)$
$S_0 = (x = 1) \wedge (y = 1)$
$R(x, y, x', y') = (x' = (x + y) \% 2) \wedge (y' = y)$



- In a finite-state program, *system variables* V range over a *finite domain* D: V = {x, y} and D = {0, 1}.
- A *state* of the system is a *valuation* s : V → D.
- Use FOL to describe the (initial) states and the transition relation.
- Extract a Kripke structure from the FOL description.

State explosion: Kripke structure usually exponential in the size of the program.

Kripke strukture

A Kripke structure for a concurrent program

P_1

```
10 while (true) {
11   wait(turn == 0);
     // critical section
12   turn := 1;
13 }
```

P_2

```
20 while (true) {
21   wait(turn == 1);
     // critical section
22   turn := 0;
23 }
```

Two processes executing concurrently and asynchronously, using the shared variable `turn` to ensure ***mutual exclusion***:

They are never in the critical section at the same time.

Kripke strukture

A Kripke structure for a concurrent program

P_1

```
10 while (true) {
11   wait(turn == 0);
     // critical section
12   turn := 1;
13 }
```

P_2

```
20 while (true) {
21   wait(turn == 1);
     // critical section
22   turn := 0;
23 }
```

Two processes executing concurrently and asynchronously, using the shared variable `turn` to ensure ***mutual exclusion***:

They are never in the critical section at the same time.

State of the program described by the variable `turn` and the *program counters* for the two processes.

Kripke strukture

A Kripke structure for a concurrent program

P_1

```
10  while (true) {
11    wait(turn == 0);
      // critical section
12    turn := 1;
13  }
```

P_2

```
20  while (true) {
21    wait(turn == 1);
      // critical section
22    turn := 0;
23  }
```

Kripke strukture

A Kripke structure for a concurrent program

P_1

```
10 while (true) {
11   wait(turn == 0);
     // critical section
12   turn := 1;
13 }
```

P_2

```
20 while (true) {
21   wait(turn == 1);
     // critical section
22   turn := 0;
23 }
```

Kripke strukture

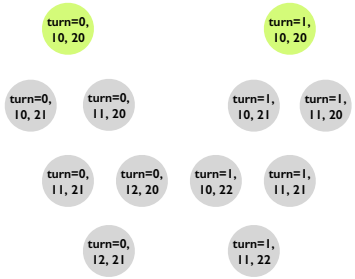
A Kripke structure for a concurrent program

P_1

```
10 while (true) {
11   wait(turn == 0);
     // critical section
12   turn := 1;
13 }
```

P_2

```
20 while (true) {
21   wait(turn == 1);
     // critical section
22   turn := 0;
23 }
```



Kripke strukture

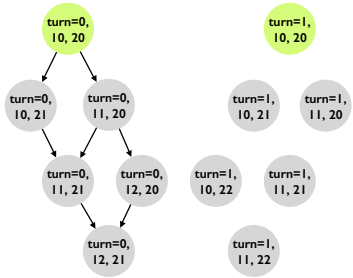
A Kripke structure for a concurrent program

P_1

```
10 while (true) {
11   wait(turn == 0);
     // critical section
12   turn := 1;
13 }
```

P_2

```
20 while (true) {
21   wait(turn == 1);
     // critical section
22   turn := 0;
23 }
```



Kripke strukture

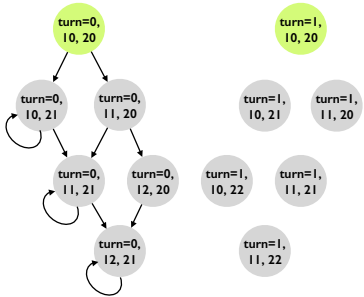
A Kripke structure for a concurrent program

P_1

```
10  while (true) {
11    wait(turn == 0);
      // critical section
12    turn := 1;
13  }
```

P_2

```
20  while (true) {
21    wait(turn == 1);
      // critical section
22    turn := 0;
23  }
```



Kripke strukture

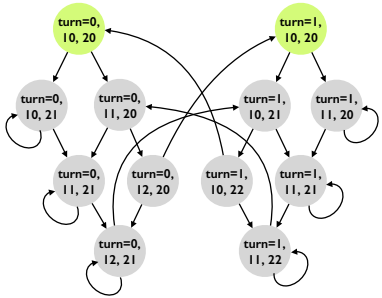
A Kripke structure for a concurrent program

P_1

```
10 while (true) {
11   wait(turn == 0);
     // critical section
12   turn := 1;
13 }
```

P_2

```
20 while (true) {
21   wait(turn == 1);
     // critical section
22   turn := 0;
23 }
```



Tranzicioni sistem — Kripke struktura

Tranzicioni sistem $(S, Act, \longrightarrow, I, \mathcal{V}, \lambda)$

Tranzicioni sistem je uređena šestorka $(S, Act, \longrightarrow, I, \mathcal{V}, \lambda)$, gde je:

- S skup stanja
- $\longrightarrow \subseteq S \times Act \times S$ relacija prelaska (u oznaci $s_i \xrightarrow{\alpha} s_j$, ukoliko akcija nije bitna, onda se oznaka α izostavlja)
- $I \subseteq S$ skup početnih stanja
- $\mathcal{V}$ skup iskaznih promenljivih (skup predikata)
- $\lambda : S \to \mathbb{P}\mathcal{V}$ funkcija obeležavanja (svakom stanju pridružuje skup predikata koji važe u tom stanju)

Tranzicioni sistem

Napomene

- Skup stanja može biti konačan ili beskonačan (tipično je konačan, ali veoma veliki)
- Bez ograničenja opštosti, može se pretpostaviti da je relacija $\longrightarrow$ *totalna*, tj. $(\forall s \in S)(\exists s' \in S)(s \longrightarrow s')$ (uslov koji se zahteva u definiciji Kripke struktura)
- Sistem može biti deterministički (po akcijama) ako važi

$$(\forall s \in S)(\forall \alpha \in Act)(\exists! s' \in S)(s \xrightarrow{\alpha} s')$$

  i nedeterministički
- Kod determinističkih sistema, podrazumeva se tačno jedno početno stanje

Tranzicioni sistem

Putanja i izvršavanje

Putanja u sistemu $T = (S, Act, \longrightarrow, I, \mathcal{V}, \lambda)$ je beskonačni niz koji alternira stanja i akcije

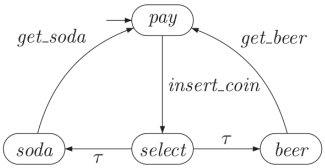
$$\sigma = s_0 \alpha_1 s_1 \alpha_2 s_2 \ldots$$

takav da važi $s_i \xrightarrow{\alpha_i} s_{i+1}$ za svako $i \geq 0$.
Pri tom ćemo koristiti oznake:

- $\sigma_i := s_i$ za i-to stanje putanje i
- $\sigma_{|i} := s_i \alpha_i s_{i+1} \alpha_{i+1} s_{i+2} \ldots$ za „rep" putanje počev od i-tog stanja.

Izvršavanje u sistemu T je bilo koja putanja σ takva da je $\sigma_0 \in I$.
Za stanje $s \in S$ kažemo da je **dostižno** ako postoji izvršavanje σ takvo da $s \in \sigma$.

Primer



Mašina za piće

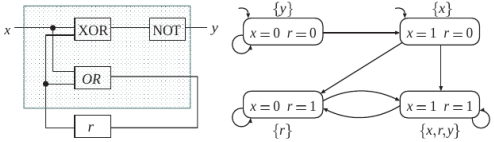
- $S = \{pay, select, soda, beer\}$
- $I = \{pay\}$
- $Act = \{insert\_coin, get\_soda, get\_beer, \tau\}$
- Primer prelaza $pay \xrightarrow{insert\_coin} select$
- $\mathcal{V}$ zavisi od osobine koju razmatramo. Na primer, "Mašina daje piće samo nakon što dobije novčić" $\mathcal{V} = \{paid, drink\}$
- $\lambda(pay) = \emptyset$, $\lambda(select) = \{paid\}$, $\lambda(soda) = \lambda(beer) = \{paid, drink\}$

Modelovanje hardvera

Modelovanje hardvera

- Stanja su određena stanjem registara u datom hardverskom sklopu i vrednostima bitova na ulazu
- Ulazi dolaze iz spoljašnjeg okruženja (druge komponente, interakcija sa korisnikom), i njihove vrednosti se ne mogu predvideti (*nedeterminizam*)
- Relacija prelaska opisuje ponašanje kola u svakom ciklusu časovnika
- Tipično imamo veliki broj stanja (eksponencijalan u odnosu na broj bitova registara i bitova ulaza)
- Skup $\mathcal{V}$ obično odgovara bitovima ulaza, bitovima registara, kao i bitovima izlaza
- $\lambda(s)$ je tada skup svih bitova koji imaju vrednost 1 u stanju s

Primer 3



Primer 3

- Jednobitni: x – ulaz, y – izlaz, r – registar
- Izlaz se računa po funkciji $\neg(x \oplus r)$, registar kao $x \vee r$
- Stanje: par bitova (x, r)
- Početna stanja: za $r = 0$
- Graf relacije prelaska je prikazan na desnoj slici
- Ulaz x uvodi nedeterminizam
- $\mathcal{V} = \{x, r, y\}$
- Skup $\lambda(s)$ je prikazan uz svako stanje u grafu

Modelovanje softvera

Program

Svaki program P se sastoji iz:

- konačnog skupa promenljivih $Var(P) = x_1, \ldots, x_n$, pri čemu promenljiva x_i uzima vrednosti iz skupa $Dom(x_i)$
- konačnog skupa lokacija $Loc(P) = L_0, L_1, \ldots, L_m$, pri čemu je L_0 početna lokacija (tj. ulazna tačka programa)
- skupa uslova $Cond(P)$ nad $Var(P)$ (npr. $x_1 < x_2$, $x_1 > 0$ i sl.)
- skupa preduslova $Pred(P) \subset Cond(P)$ (moraju biti zadovoljeni na početku)
- skupa akcija $Act(P)$ nad $Var(P)$ (npr. $x_1 \leftarrow x_1 + 1$, $x_2 \leftarrow x_1 \cdot x_1$)
- relacije toka programa $\rightarrow \subseteq Loc(P) \times (Cond(P) \cup Act(P)) \times Loc(P)$ (npr. $L_0 \xrightarrow{x_1 \leftarrow x_1 + 1} L_1$ znači da se na lokaciji L_0 izvršava akcija $x_1 \leftarrow x_1 + 1$ kojom se menja vrednost promenljive x_1 i prelazi se na lokaciju L_1, dok $L_1 \xrightarrow{x_1 < x_2} L_2$ znači da se na lokaciji L_1 ispituje uslov $x_1 < x_2$ i ako je ispunjen, prelazi se na lokaciju L_2).

Primer 4

```
    while(true)
    {
L0:   x = (x+1) % 3;
L1:   if(x == 0)
L2:    y = -y;
L3:   z = z*y + 1;
    }
```

Formalni opis programa P iz Primera 2

- $Var(P) = \{x, y, z\}$, $Dom(x) = Dom(y) = Dom(z) = Int$
- $Loc(P) = \{L_0, L_1, L_2, L_3\}$
- $Pred(P) = \{x = 0, y = 1, z = 1\}$
- $L_0 \xrightarrow{x \leftarrow (x+1)\%3} L_1$
- $L_1 \xrightarrow{x=0} L_2$
- $L_1 \xrightarrow{x \neq 0} L_3$
- $L_2 \xrightarrow{y \leftarrow -y} L_3$
- $L_3 \xrightarrow{z \leftarrow z \cdot y + 1} L_0$

Modelovanje softvera

Izvršavanje programa

Izvršavanje programa P se može formalno predstaviti sledećim tranzicionim sistemom:

- skup stanja je $S = Loc(P) \times Val(P)$ gde je $Val(P) = Dom(x_1) \times Dom(x_2) \ldots \times Dom(x_n)$ skup valuacija promenljivih $x_1, \ldots, x_n$ iz $Var(P)$
- skup početnih stanja $I = \{L_0\} \times PredVal(P)$ gde je $PredVal(P)$ skup svih valuacija iz $Val(P)$ koje zadovoljavaju sve preduslove iz $Pred(P)$
- relacija prelaska je definisana na sledeći način:
  - ako važi $L_i \xrightarrow{a} L_j$, gde je $a \in Act(P)$, tada za svaku valuaciju $\eta \in Val(P)$ postoji prelaz $(L_i, \eta) \longrightarrow (L_j, \eta[a])$, pri čemu je $\eta[a]$ valuacija koja nastaje primenom akcije a na valuaciju η
  - ako važi $L_i \xrightarrow{c} L_j$, gde je $c \in Cond(P)$, tada za svaku valuaciju $\eta \in Val(P)$ postoji prelaz $(L_i, \eta) \longrightarrow (L_j, \eta)$ akko valuacija η zadovoljava uslov c
- skup predikata $\mathcal{V}$ je bilo koji konačan podskup skupa $Cond(P)$
- obeležavanje $\lambda((L_i, \eta))$ sadrži uslove iz $\mathcal{V}$ koji su zadovoljeni u valuaciji η

Primer 5

```
    while(true)
    {
L0:   x = (x+1) % 3;
L1:   if(x == 0)
L2:    y = -y;
L3:   z = z*y + 1;
    }
```

Tranzicioni sistem prethodnog programa

- Stanja: $(L_i, [v_x, v_y, v_z])$, gde je L_i lokacija u programu, a $[v_x, v_y, v_z]$ valuacija promenljivih x, y, z
- Početno stanje: $(L_0, [0, 1, 1])$
- Relacija prelaska:

$$
\begin{array}{lcl}
(L_0, [v_x, v_y, v_z]) & \longrightarrow & (L_1, [(v_x + 1)\%3, v_y, v_z]) \\
(L_1, [0, v_y, v_z]) & \longrightarrow & (L_2, [0, v_y, v_z]) \\
(L_1, [v_x, v_y, v_z]) & \longrightarrow & (L_3, [v_x, v_y, v_z]) \text{ (za } v_x \neq 0) \\
(L_2, [v_x, v_y, v_z]) & \longrightarrow & (L_3, [v_x, -v_y, v_z]) \\
(L3, [v_x, v_y, v_z]) & \longrightarrow & (L_0, [v_x, v_y, v_z \cdot v_y + 1])
\end{array}
$$

(Primetimo da je ovo deterministički tranzicioni sistem)

- Skup predikata je npr. $\mathcal{V} = \{y > 0, |z| < 3\}$
- Na primer, važi $\lambda((L_2, [1, -1, -2]) = \{|z| < 3\}$

Modelovanje softvera

Napomene

- Pri modelovanju softvera, broj stanja je veoma veliki (npr. samo jedna `int` promenljiva u C-u ima preko 4 milijarde mogućih stanja)
- Postoje različiti načini da se izborimo sa tim problemom (npr. apstrakcija predikata)
- Softver se takođe može modelovati nedeterminističkim tranzicionim sistemom (tipično za konkurentne aplikacije, operativne sisteme i sl.)

Pregled

Svojstva stanja

Svojstvo stanja (engl. state property)

- Svojstva koja izražavaju uslov koji se odnosi na pojedinačno stanje nazivaju se *svojstva stanja*
- Mogu se izraziti iskaznim formulama nad predikatima tranzicionog sistema
- Primer: $x > 0 \wedge ((z + y) \bmod 2 = 0)$
- Nisu preterano zanimljiva sama za sebe (jer ne uzimaju u obzir relaciju prelaska) već kao gradivni element pri formulisanju složenih svojstava

Složena svojstva

Složena svojstva

Neka interesantna svojstva koja ćemo razmatrati su

- invarijante,
- sigurnosna svojstva,
- svojstva živosti i
- svojstva pravednosti.

Složena svojstva

Invarijanta

- *Invarijanta* je svojstvo koje treba da važi u svim dostižnim stanjima tranzicionog sistema
- Na primer, „u svim dostižnim stanjima je $x \neq 0$“

Sigurnosno svojstvo (engl. safety property)

- *Sigurnosno svojstvo* je svojstvo koje izražava da se neka negativna pojava nikada neće desiti

Složena svojstva

Sigurnosno svojstvo (engl. safety property)

- Svaka invarijanta je i sigurnosno svojstvo (npr. „u svim dostižnim stanjima je $x \neq 0$" znači da „nikada se neće desiti da je $x = 0$")
- Postoje sigurnosna svojstva koja nisu invarijante (npr. „kada x postane 1, tada x mora ostati 1 dokle god je $y = 0$")
- Sigurnosno svojstvo je narušeno u tranzicionom sistemu ako postoji *loš prefiks*, tj. izvršavanje σ takvo da za neki njegov konačni prefiks ne važi dato svojstvo (npr. prethodno svojstvo je narušeno ako imamo prefiks $\sigma_0\sigma_1 \ldots \sigma_i\sigma_{i+1}$ izvršavanja σ takav da u stanju σ_i važi $x = 1, y = 0$, a u stanju σ_{i+1} važi $x = 0, y = 0$)
- To znači da je putanja greške konačna

Složena svojstva

Svojstvo živosti (engl. liveness property)

- *Svojstvo živosti* je svojstvo koje izražava da će se neka pozitivna pojava sigurno desiti u budućnosti
- Primer: „kada x postane 1, sigurno će u nekom narednom trenutku i y postati 1"
- Ovakva svojstva koristimo za izražavanje progresa u sistemu (npr. odsustvo uzajamnog blokiranja)
- Ovakvo svojstvo ne može biti narušeno lošim prefiksom (za razliku od sigurnosnog svojstva), jer ako nešto ne važi do nekog trenutka, možda će važiti u budućnosti.
- To znači da je putanja greške beskonačna

Složena svojstva

Svojstvo pravednosti (engl. fairness property)

- *Svojstvo pravednosti* je svojstvo koje izražava da će se neka pojava dešavati beskonačno puta tokom izvršavanja
- Primer: „ako x postaje 1 beskonačno puta tokom izvršavanja, tada će i y postajati 1 beskonačno puta tokom izvršavanja“
- Ovakva svojstva koristimo da izrazimo da u sistemu nema „izgladnjivanja” (npr. da će svaki proces na nekom operativnom sistemu kontinuirano dobijati priliku da se izvršava)

Temporalne logike

Temporalna logika

Temporalna logike su logike koje omogućavaju predstavljanje tvrđenja i rasuđivanje o tvrđenjima koja su kvalifikovana vremenskim odrednicama. Na primer, u temporalnoj logici možemo da izrazimo tvrđenja kao što su „Ja sam **uvek** gladan", „Ja ću **u nekom trenutku** biti gladan" i „Ja ću biti gladan **sve dok** ne pojedem nešto."

Temporalne logike

Osnovni operatori

U temporalnim logikama srećemo različit izbor dodatnih operatora, i samim tim one opisuju različite modele vremena. Dakle, ne postoji jedinstvena temporalna logika, već čitav niz različitih logičkih sistema koji, svaki na svoj način, opisuju fenomene koji su povezani sa vremenom. Najčešći operatori koji se dodaju su operatori G (**G**lobaly, always), F (**F**uture, eventually) i X (ne**X**t).

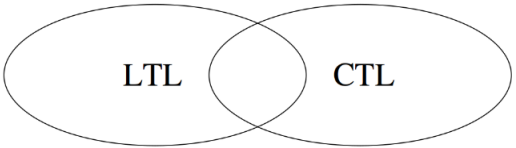
Izražavanje složenih svojstava

Složena svojstva koja se koriste u verifikaciji softvera mogu se formalno izraziti u terminima temporalnih logika. Posebno važne temporalne logike za proveravanje modela su LTL, CTL i CTL*.

Temporalne logike

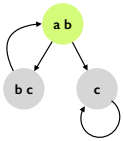
Odnosi

CTL* je obuhvata logike LTL i CTL. Postoje tvrđenja koja se mogu iskazati u LTL logici a ne mogu u CTL, postoje tvrđenja koja se mogu iskazati u CTL logici a ne mogu u LTL, postoje tvrđenja koja se mogu iskazati u obe logike. Praktična važna razlika je u algoritmima odlučivanja — najveća složenost je za CTL*, najefikasniji algoritam odlučivanja je za CTL, a najintuitivnije za razumevanje je modelovanje u LTL.

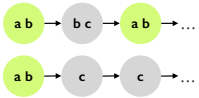


LTL i CTL

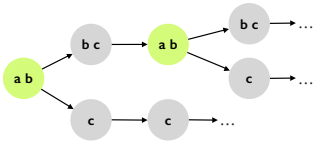
Expressing properties in temporal logics



Linear time: properties of computation paths



Branching time: properties of computation trees



Linearna temporalna logika

Linearna temporalna logika (LTL) – sintaksa

Neka je dat tranzicioni sistem $T = (S, Act, \longrightarrow, I, \mathcal{V}, \lambda)$. LTL formula nad T je:

- $\top$ i $\bot$ (logičke konstante)
- $p \in \mathcal{V}$ (atomičke formule)
- $\neg\phi$, gde je ϕ LTL formula
- $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, $\phi_1 \Rightarrow \phi_2$, gde su ϕ_1 i ϕ_2 LTL formule
- $\phi_1 U \phi_2$ i $\phi_1 W \phi_2$, gde su ϕ_1 i ϕ_2 LTL formule
- $X\phi$, $F\phi$ i $G\phi$, gde je ϕ LTL formula

Linearna temporalna logika

Linearna temporalna logika (LTL) – semantika

Neka je sada data proizvoljna putanja σ nad T. Formula ϕ je tačna na σ (u oznaci $\sigma \vDash \phi$) ako:

- $\sigma \vDash p$ ako $p \in \lambda(\sigma_0)$ (tačno u početnom stanju putanje)
- $\sigma \vDash \neg\phi_1$ ako $\sigma \nvDash \phi_1$
- $\sigma \vDash \phi_1 \wedge \phi_2$ ako $\sigma \vDash \phi_1$ i $\sigma \vDash \phi_2$ (slično za ostale iskazne veznike)
- $\sigma \vDash X\phi_1$ ako $\sigma_{|1} \vDash \phi_1$
- $\sigma \vDash \phi_1 U \phi_2$ ako $(\exists j)(\sigma_{|j} \vDash \phi_2 \wedge (\forall i < j)(\sigma_{|i} \vDash \phi_1))$
- $F\phi$ je ekvivalentno sa $\top U \phi$
- $G\phi$ je ekvivalentno sa $\neg F \neg \phi$
- $\phi_1 W \phi_2$ je ekvivalentno sa $(\phi_1 U \phi_2) \vee G\phi_1$

NAPOMENA: Primetimo da se semantika operatora U, W, F, G i X definiše rekurzivno u odnosu na odgovarajuće repove putanje σ (rep putanje je takođe putanja).

Linearna temporalna logika

Linearna temporalna logika (LTL) – semantika

Intuitivno:

- $X\phi$ znači da ϕ važi u sledećem stanju putanje σ — ne**X**t
- $\phi_1 U \phi_2$ znači da ϕ_1 važi duž putanje σ dokle god ϕ_2 ne postane tačno („ϕ_1 dok se ne ispuni ϕ_2") — **U**ntil
- $\phi_1 W \phi_2$ je slično, s tim što ϕ_2 nikada ne mora da postane tačno, u kom slučaju će ϕ_1 važiti zauvek, duž cele putanje („ϕ_1 dok se eventualno ne ispuni ϕ_2") — **W**eak until
- $G\phi$ znači da će ϕ važiti duž cele putanje („zauvek ϕ") — **G**lobally
- $F\phi$ znači da će ϕ sigurno postati tačno u nekom od narednih stanja putanje („pre ili kasnije ϕ") — **F**uture state

Linearna temporalna logika

Linearna temporalna logika (LTL) – semantika

Neke česte kombinacije operatora:

- $XG\phi$: „važi ϕ počev od sledećeg stanja zauvek"
- $GF\phi$: „ϕ važi u beskonačno mnogo stanja duž putanje" (za svako stanje važi da postoji stanje u budućnosti na kojem važi ϕ)
- $FG\phi$: „od nekog trenutka u budućnosti, zauvek će važiti važiti ϕ "
- $G(\phi_1 \Rightarrow F\phi_2)$: „kad god važi ϕ_1 u nekom stanju, tada mora da važi ϕ_2 u nekom od narednih stanja"
- $G(\phi \Rightarrow X\neg\phi)$: „kad god važi ϕ u nekom stanju, tada ne važi u sledećem"

Linearna temporalna logika

Linearna temporalna logika (LTL) – semantika

- Formula ϕ je zadovoljiva u tranzicionom sistemu T ako postoji izvršavanje σ (tj. putanja koja kreće iz nekog početnog stanja), takvo da je $\sigma \vDash \phi$
- Formula ϕ je valjana u tranzicionom sistemu T ako za svako izvršavanje σ važi da je $\sigma \vDash \phi$

Svojstva na LTL jeziku

Svojstva na LTL jeziku

- LTL omogućava formulisanje većine svojstava koja se tipično javljaju u verifikaciji zasnovanoj na proveravanju modela
- Svojstva se formulišu kao LTL formule tako da sistem koji se verifikuje zadovoljava dato svojstvo akko je odgovarajuća formula valjana u tranzicionom sistemu T koji je model početnog sistema

Svojstva na LTL jeziku

Invarijante

- Invarijante se izražavaju formulama oblika $G\phi$, gde je ϕ iskazna formula
- Primer: $G(x = 0 \vee y = 0)$: „u svim dostižnim stanjima je x ili y jednako nula“

Sigurnosna svojstva

- Takođe imaju G kao vodeći veznik, ali podformula može sadržati i druge ne-iskazne veznike
- $G(x = 1 \Rightarrow (x = 1\ W\ y \neq 0))$: „kad x postane 1, ostaće 1 dokle god je y jednako 0”

Svojstva na LTL jeziku

Svojstva živosti

$G(x = 1 \Rightarrow F(y = 1))$: „kad god x postane 1, sigurno će i y u nekom narednom trenutku postati 1"

Svojstva pravednosti

$G(F(x = 1)) \Rightarrow G(F(y = 1))$: „ako x postaje 1 beskonačno često, tada i y mora postati 1 beskonačno često

Svojstva na LTL jeziku

Primer svojstva koje se ne može izraziti na LTL jeziku...

... je svako svojstvo koje uključuje vremensko rezonovanje o svim putanjama. Na primer,

- „Iz svakog dostižnog stanja se može stići u neko početno stanje".
- „Počevši od nekog trenutka, sve putanje će imati svojstvo ϕ."

CTL* i CTL

Logika stabla izračunavanja (Computational tree logic)

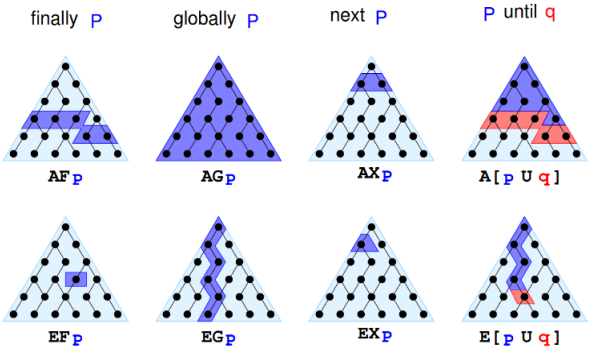
Uvodimo operatore nad putanjama: A i E

A ϕ : formula ϕ važi na svim putanjama počevši od trenutnog stanja.

E ϕ : postoji najmanje jedna putanja počevši od trenutnog stanja tako da na njoj važi formula ϕ.

U CTL*, temporalni opeartori mogu da se mešaju proizvoljnim redosledom. U CTL-u, ovi operatori se ne mogu koristiti u proizvoljnom poretku, već isključivo u kombinaciji sa operatorima nad stanjima (X, G, F, U, W), tj uvek moraju da budu u grupama od dva: jedan operator nad putanjom za kojim sledi operator nad stanjem.

CTL operatori



Pregled

Provera invarijanti

Invarijanta je svojstvo koje treba da važi u svim dostižnim stanjima

- Jednostavan način da se ovo proveri je obilazak grafa tranzicionog sistema (polazeći iz početnih stanja)
- Obilazak u dubinu: omogućava jednostavno pronalaženje kontraprimera (kada naiđemo na stanje u kome ne važi invarijanta, na steku se nalaze svi njegovi prethodnici na toj putanji)
- Obilazak u širinu: omogućava pronalaženje najkraćeg puta do nekog stanja koje narušava invarijantu (za rekonstrukciju celog puta neophodno je čuvati dodatne informacije tokom pretrage)
- Problem: u slučaju prevelikog broja stanja graf postaje preveliki

Bihijevi automati

Bihijevi automati

- Julias Richard Büchi (1924-1984): švajcarski matematičar
- Koriste se za verifikaciju opštih LTL svojstava (ne samo invarijanti)
- Zasnivaju se na uopštenju teorije formalnih jezika
- Izvršavanja u tranzicionom sistemu se posmatraju kao beskonačne reči nad azbukom stanja

ω-regularni jezici

ω-jezici

Neka je data konačna azbuka Σ. ω-reč nad Σ je bilo koji beskonačni niz $w = a_0 a_1 a_2 \ldots$ slova azbuke Σ. Skup svih ω-reči označavamo sa Σ^ω. ω-jezik nad Σ je bilo koji podskup skup Σ^ω.

ω-regularni jezici

ω-jezik je ω-regularan ako se dobija konačnom primenom sledećih pravila:

- Ako je L neprazan regularan jezik koji ne sadrži ϵ, tada je $L^\omega = \{w_1 w_2 \ldots \mid w_i \in L\}$ ω-regularan
- Ako je L_1 regularan jezik, a L_2 ω-regularan, tada je i $L_1 L_2$ ω-regularan
- Ako su L_1 i L_2 ω-regularni, tada je i $L_1 \cup L_2$ ω-regularan

Primer ω-regularnog jezika

$(0|1)^*(10)^\omega \mid 01^\omega$: skup svih beskonačnih reči koje imaju „rep” oblika $1010\ldots$ ili reč $0111\ldots$

Bihijevi automati

Bihijevi automati

Bihijev automat je uređena petorka $\mathcal{A} = (Q, \Sigma, Q_0, \delta, F)$, gde je:

- Q konačan skup stanja
- Σ konačna azbuka
- $Q_0 \subseteq Q$ skup početnih stanja
- $F \subseteq Q$ skup završnih stanja
- $\delta : Q \times \Sigma \to \mathbb{P}Q$ funkcija prelaska

Izračunavanje u Bihijevom automatu je bilo koje beskonačno izvođenje oblika $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} q_3 \ldots$, pri čemu je $q_0 \in Q_0$ i $q_{i+1} \in \delta(q_i, a_{i+1})$ za svako $i = 0, 1, 2, \ldots$. Etiketa tog izračunavanja je ω-reč $a_1 a_2 a_3 \ldots$. Za izračunavanje kažemo da je uspešno ako sadrži beskonačno mnogo završnih stanja. ω-jezik $L(\mathcal{A})$ definisan automatom $\mathcal{A}$ je skup svih ω-reči koje su etikete uspešnih izračunavanja automata $\mathcal{A}$.

Bihijevi automati

Teorema

ω-jezik L je ω-regularan akko je prepoznat nekim (u opštem slučaju nedeterminističkim) Bihijevim automatom

LTL i Bihijevi automati

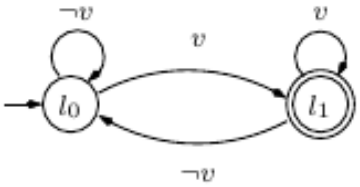
LTL i Bihijevi automati

- Za svaki tranzicioni sistem $T = (S, \longrightarrow, I, \mathcal{V}, \lambda)$, svaka njegova putanja $\sigma = s_0 s_1 s_2 \ldots$ je jedna ω-reč nad azbukom stanja S
- Proizvoljna LTL formula definiše skup putanja koje je zadovoljavaju, tj. definiše jedan ω-jezik nad azbukom S
- Može se pokazati da je svaki tako definisan jezik ω-regularan, pa se može prepoznati Bihijevim automatom

LTL i Bihijevi automati

Primer

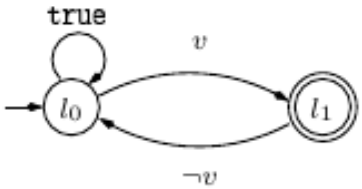
Bihijev automat za formulu GFv („beskonačno puta važi v"), gde je v neko svojstvo stanja (u prikazu automata su stanja tranzicionog sistema apstrahovana azbukom $\{v, \neg v\}$, jer nas samo to svojstvo zanima)



LTL i Bihijevi automati

Primer

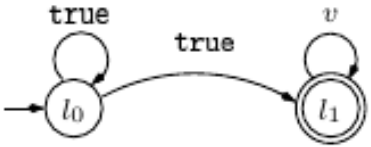
Bihijev automat za formulu $GF(v \wedge X\neg v)$ („beskonačno puta imamo da u nekom stanju važi v, pri čemu odmah u sledećem stanju važi $\neg v$")



LTL i Bihijevi automati

Primer

Bihijev automat za formulu FGv („počev od nekog stanja v zauvek važi"). Za ovo svojstvo ne postoji deterministički Bihijev automat



LTL i Bihijevi automati

Postupak provere LTL svojstva ϕ

- Konstruišemo Bihijev automat $\mathcal{A}$ za formulu $\neg\phi$
- Napravimo presek dobijenog automata sa tranzicionim sistemom T (tj. razmatramo samo one ω-reči koje predstavljaju izvršavanja u sistemu T)
- Ako dobijeni presek definiše prazan jezik, svojstvo ϕ je valjano u T
- Ispitivanje da li je jezik koji definiše neki Bihijev automat prazan se svodi na traženje dostižnih ciklusa u grafu, što je linearne složenosti u odnosu na veličinu grafa

LTL i Bihijevi automati

Problemi

- Za proizvoljnu LTL formulu ϕ veličina dobijenog automata $\mathcal{A}$ je eksponencijalna u odnosu na veličinu formule ϕ
- Veličina preseka automata $\mathcal{A}$ sa tranzicionim sistemom T je proporcionalna proizvodu $|\mathcal{A}| \cdot |T|$. Iako su formule (i njima odgovarajući automati) često relativno male, tranzicioni sistemi po pravilu nisu

Kombinatorna eksplozija

Kombinatorna eksplozija

- Na primeru Bihijevih automata videli smo da je glavni problem u prevelikom broju stanja tranzicionih sistema koji se verifikuju
- U slučaju jako velikog broja stanja, često nije moguće ni predstaviti odgovarajući graf u memoriji
- Ovaj problem se rešava na više načina:
  - Apstrakcijom: stanja se apstrahuju skupovima predikata koji su u tim stanjima zadovoljeni
  - Simboličkom proverom modela: stanja i relacija prelaska se ne predstavljaju eksplicitno, već pomoću iskaznih formula i BDD dijagrama
  - Ograničenom proverom modela: tražimo kontraprimer u prvih k koraka, za dato k

Apstrakcija predikata

Apstrakcija predikata

Neka je dat tranzicioni sistem $T = (S, \longrightarrow, I, \mathcal{V}, \lambda)$. *Apstrakcija predikata* podrazumeva sistem T' kod koga je:

- skup stanja $S' = \{s' \mid s' = \lambda(s)\} \subseteq \mathbb{P}\mathcal{V}$ je familija podskupova skupa $\mathcal{V}$ (stanje $s' = \lambda(s)$ nazivamo *apstrakcijom stanja* s)
- skup početnih stanja I' odgovara apstrakcijama stanja iz I
- relacija prelaska: prelaz $s'_1 \longrightarrow' s'_2$ postoji u T' akko postoji prelaz $s_1 \longrightarrow s_2$ u T, gde su s'_1 i s'_2 apstrakcije stanja s_1 i s_2 sistema T
- skup predikata $\mathcal{V}'$ se poklapa sa $\mathcal{V}$
- funkcija obeležavanja je funkcija identiteta: $\lambda'(s') = s'$

Apstrakcija predikata

Napomena

Ako neko svojstvo važi u apstraktnom sistemu, važi i u polaznom. Obrnuto ne mora da važi, tj. ako neko svojstvo ne važi u apstraktnom modelu, možda je to zato što je model suviše apstraktan (u tom slučaju ga moramo popraviti)

Primer 6

```
    while(true)
    {
L0:   x = (x+1) % 3;
L1:   if(x == 0)
L2:    y = -y;
L3:   z = z*y + 1;
    }
```

Tranzicioni sistem iz Primera 5

- Stanja: $(L_i, [v_x, v_y, v_z])$, gde je L_i lokacija u programu, a $[v_x, v_y, v_z]$ valuacija promenljivih x, y, z
- Ako su x, y, z promenljive tipa `int`, imamo preko $4000000000^3 \cdot 4$ stanja
- Neka je skup predikata $\mathcal{V} = \{y > 0, |z| < 3\}$
- Apstrakcijom predikata dobijamo skup stanja oblika (L_i, uv) gde je u istinitonosna vrednost predikata $y > 0$, a v predikata $|z| < 3$ (samo $4 \cdot 4 = 16$ stanja)
- Početno stanje: $(L_0, 11)$
- Relacija prelaska:

$$
\begin{aligned}
(L_0, uv) &\longrightarrow (L_1, uv) \\
(L_1, uv) &\longrightarrow (L_2, uv) \\
(L_1, uv) &\longrightarrow (L_3, uv) \\
(L_2, uv) &\longrightarrow (L_3, \overline{u}v) \\
(L_3, uv) &\longrightarrow (L_0, uv) \\
(L_3, uv) &\longrightarrow (L_0, u\overline{v})
\end{aligned}
$$

(Primetimo da je ovo nedeterministički tranzicioni sistem)

Apstrakcija predikata

Apstrakcija predikata

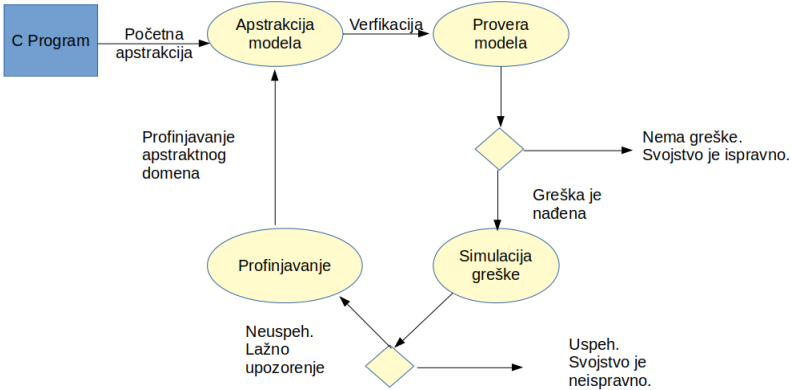
- U prethodnom primeru dobili smo značajno manji tranzicioni sistem koji apstrahuje nebitne detalje (razmatra samo svojstva koja nas zanimaju i šta se sa njima dešava tokom izvršavanja)
- Dobijeni apstraktni sistem može biti nedeterministički
- U prethodnom primeru, na lokaciji L_1 sledeća lokacija može biti L_2 ili L_3 i to ne zavisi od vrednosti predikata $y > 0$ i $|z| < 3$

Apstrakcija predikata — profinjavanje

Apstrakcija predikata — *CEGAR*

- Ovakav sistem često može biti previše apstraktan i neprecizan (veoma mali broj svojstava se može dokazati u takvom modelu)
- Uobičajeno je da se dobijeni kontraprimeri u apstraktnom modelu iskoriste za dobijanje preciznije apstrakcije
- Ovaj postupak se naziva CEGAR (engl. *CounterExample-Guided Abstraction Refinement*)
- Preciznije apstrakcije razmatraju veći broj predikata i samim tim imaju više stanja, ali se u njima može dokazati više svojstava

CEGAR — prikaz procesa



Binarni dijagrami odlučivanja

Binarni dijagrami odlučivanja

- Binarni dijagrami odlučivanja (engl. *binary decision diagram* (BDD)) predstavljaju kompaktnu strukturu podataka kojom se opisuje semantika iskaznih formula
- Ekvivalentne iskazne formule imaju identične BDD-ove
- Logičke operacije konjunkcije, disjunkcije i negacije se jednostavno izvode nad BDD-ovima

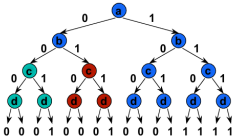
Binarni dijagrami odlučivanja

Napomena

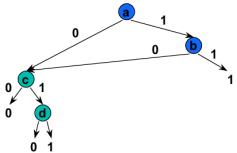
- Svaka klasa ekvivalentnih formula (samim tim i svaki BDD) jednoznačno odgovara skupu valuacija koje te formule zadovoljavaju
- Ovo znači da se BDD-ovi mogu koristiti i za predstavljanje skupova (elemente skupa treba kodirati binarnim brojevima koje posmatramo kao valuacije) — dakle, stanja u tranzicionom sistemu možemo predstavljati sa BDD-om
- Kako su i relacije skupovi, sledi da se BDD-ovima mogu predstavljati i relacije — dakle, i relacija prelaska se može predstaviti BDD-om

Primer 7

Stablo odlučivanja za formulu $(a \wedge b) \vee (c \wedge d)$:



Redukovani BDD za istu formulu:



Binarni dijagrami odlučivanja

Postupak formiranja BDD-a

- Jedan način je da se najpre napravi potpuno binarno stablo odlučivanja, a da se zatim izvrši redukcija tako što se eliminiše dupliranje identički jednakih podstabala
- Efikasniji način je da se vrši dekompozicija formule, uz rekurzivno kreiranje BDD-a za podformule koji se zatim kombinuju primenom logičkih operacija
- U svakom slučaju, formiranje BDD-a u najgorem slučaju zahteva eksponencijalno vreme u odnosu na veličinu formule

Primena BDD-ova u proveravanju modela

Primena BDD-ova u proveravanju modela

- Setimo se da se svako stanje tranzicionog sistema može apstrahovati skupom predikata koji su tačni u tom stanju
- Ovaj skup predikata koji su tačni možemo da razumemo i kao valuaciju nad svim predikatima
- Otuda svim stanjima kojima odgovara isti skup tačnih predikata odgovara jedinstveni BDD
- Slično, relacija prelaska $s \longrightarrow s'$ se može predstaviti BDD-om nad predikatima iz $\mathcal{V} \cup \mathcal{V}'$, gde je $\mathcal{V}' = \{p'_1, \ldots, p'_n\}$ skup kopija predikata $p_1, \ldots, p_n$ iz $\mathcal{V}$ (intuitivno, predikati $p_1, \ldots, p_n$ su iz stanja s, a $p'_1, \ldots, p'_n$ su iz stanja s')
- Ovi BDD-ovi se dalje koriste za proveravanje svojstava, bez eksplicitnog čuvanja grafa tranzicionog sistema u memoriji

Simboličko proveravanje invarijanti

Simboličko proveravanje invarijanti

- Neka je $I(s)$ formula (ili BDD) koja opisuje početne uslove, a $T(s, s')$ formula koja opisuje relaciju prelaska tranzicionog sistema
- Stanja koja su dostupna iz početnih stanja u jednom koraku se mogu predstaviti sledećom formulom: $(\exists s')(I(s') \wedge T(s', s))$
- Primenom odgovarajućih operacija nad polaznim BDD-ovima dobijamo novi BDD $B_1(s)$ koji opisuje stanja dostižna u jednom koraku
- Dijagram $D_1(s) = B_1(s) \vee I(s)$ predstavlja uniju početnih stanja i stanja dobijenih u jednom koraku

Simboličko proveravanje invarijanti

Simboličko proveravanje invarijanti

- Sličnim postupkom nad $D_1(s)$ i $T(s, s')$ dobijamo dijagram $D_2(s)$ koji opisuje stanja dostižna u najviše dva koraka, itd. (važi $D_1(s) \subseteq D_2(s) \subseteq D_3(s) \subseteq \ldots$)
- Kako je ukupan broj stanja konačan, opisani postupak ima fiksnu tačku (tj. za neko k važi $D_{k+1}(s) = D_k(s)$).
- Dijagram $D(s) = D_k(s)$ predstavlja sva dostižna stanja
- Ako je invarijanta data iskaznom formulom $\phi = \phi(s)$, tada formula $D(s) \wedge \neg\phi(s)$ treba da bude nezadovoljiva (kako bi bilo ispunjeno svojstvo da je u pitanju invarijanta)

Ograničena provera modela

Ograničena provera modela

- Ideja je da kontraprimer tražimo samo u prvih k koraka
- Vrednost k se može povećavati do neke razumne granice, zavisno od resursa kojima raspolažemo
- Metoda je korisna kod jako velikih modela, jer se ograničava samo na jedan deo grafa, bez njegove eksplicitne konstrukcije
- Može pronaći mnoge česte tipove grešaka, ali može i propustiti neke greške
- Zasniva se na korišćenju SAT i SMT rešavača

Ograničena provera modela

Ograničena semantika LTL operatora

- Semantika LTL formula je definisana na (beskonačnim) putanjama σ tranzicionog sistema T
- Ukoliko se ograničimo na prefikse tih putanja dužine k, tada to menja semantiku LTL operatora
- Formulu ćemo smatrati tačnom ako nema kontraprimera dužine najviše k

Ograničena provera modela

Ograničena semantika LTL operatora

- Formalno, za rep $\sigma_{|i}$ putanje σ definišemo:
  - $\sigma_{|i} \vDash X\phi$ ako $\sigma_{|i+1} \vDash \phi$ za $i < k$, odnosno uvek ako je $i \geq k$ (jer ne možemo da pretpostavimo da nije, ako je to izvan domašaja prefiksa dužine k)
  - $\sigma_{|i} \vDash F\phi$ za svako σ i svako i (jer je kontraprimer za $F\phi$ putanja na kojoj ϕ ne važi nikad, a to ne možemo da tvrdimo na osnovu prefiksa dužine k)
  - $\sigma_{|i} \vDash G\phi$ ako je $\bigwedge_{j=i,\ldots,k}(\sigma_{|j} \vDash \phi)$ za $i < k$; za $i \geq k$ je uvek tačna (jer nemamo kontraprimer dužine manje od k)
  - Slično za $\phi_1 U \phi_2$ i $\phi_i W \phi_2$

Ograničena provera modela

SAT kodiranje tranzicionog sistema

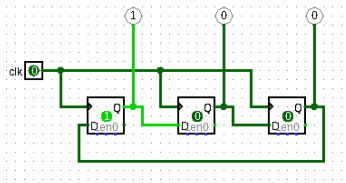
- Neka je $\mathcal{V} = \{p_1, \ldots, p_n\}$ skup predikata tranzicionog sistema T
- Pretpostavimo ponovo da je u T izvršena apstrakcija datih predikata, tj. da su stanja poistovećena sa skupovima predikata koji su tačni u tim stanjima
- Za dato k napravimo $k+1$ kopija skupa $\mathcal{V}$ (iskazne promenljive $p_1^i, \ldots, p_n^i$ odgovaraju predikatima u stanju s_i, za $i = 0, \ldots, k$).
- Neka je $I(s)$ iskazna formula nad promenljivama koje odgovaraju stanju s koja definiše uslove za početna stanja (tj. $I(s)$ znači „stanje s je početno stanje")

Ograničena provera modela

SAT kodiranje tranzicionog sistema

- Neka je $T(s, s')$ iskazna formula koja opisuje relaciju prelaska, tj. vezu između vrednosti predikata u tekućem i u narednom stanju (tj. $T(s, s')$ znači „postoji prelaz iz s u s'")
- Sada se izvršavanje sistema T u prvih k koraka može opisati iskaznom formulom $I(s_0) \wedge T(s_0, s_1) \wedge \ldots \wedge T(s_{k-1}, s_k)$

Primer 8



Nastavak primera 1

- Neka su p, q, r predikati koji odgovaraju bitovima registra sa leva u desno i neka je $k = 3$
- $I(s_i) \equiv (p^i \wedge \neg q^i \wedge \neg r^i)$ (početno stanje je 100)
- $T(s_i, s_j) \equiv (p^j \Leftrightarrow r^i) \wedge (q^j \Leftrightarrow p^i) \wedge (r^j \Leftrightarrow q^i)$ (s_i prethodi s_j ako s_j nastaje cikličnim pomeranjem s_i u desno)
- Rad kola u prva 3 koraka se opisuje formulom: $I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge T(s_2, s_3)$

Ograničena provera modela

SAT kodiranje LTL formula

- LTL formule se svode na SAT na sličan način, imajući u vidu njihovu ograničenu semantiku
- $\sigma_{|i} \vDash X(\phi(p_1, \ldots, p_n))$ kodiramo sa $\phi(p_1^{i+1}, \ldots, p_n^{i+1})$ za $i < k$, ili sa $\top$ za $i \geq k$
- $\sigma_{|i} \vDash F(\phi(p_1, \ldots, p_n))$ kodiramo sa $\top$
- $\sigma_{|i} \vDash G(\phi(p_1, \ldots, p_n))$ kodiramo sa $\bigwedge_{j=i,\ldots,k} \phi(p_1^j, \ldots, p_n^j)$, ili sa $\top$ za $i \geq k$
- Na primer, $\sigma \vDash G(p \Rightarrow Xq)$ se može predstaviti formulom $\bigwedge_{i=0,\ldots,k}(p^i \Rightarrow q^{i+1})$

Ograničena provera modela

SAT kodiranje problema ograničene provere modela

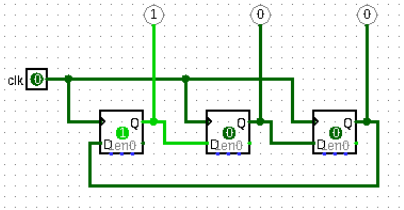
- Neka je svojstvo dato LTL formulom ϕ i neka je $\sigma \vDash \phi$ za prvih k koraka predstavljeno iskaznom formulom F
- Problem ispitivanja da li je svojstvo ϕ valjano u tranzicionom sistemu T (do na k prvih koraka) svodi se na ispitivanje valjanosti formule $I(s_0) \wedge T(s_0, s_1) \wedge \ldots \wedge T(s_{k-1}, s_k) \Rightarrow F$
- Ovo je ekvivalentno sa proverom zadovoljivosti formule $I(s_0) \wedge T(s_0, s_1) \wedge \ldots \wedge T(s_{k-1}, s_k) \wedge \neg F$

Ograničena provera modela

SAT kodiranje problema ograničene provere modela

- Ukoliko je formula nezadovoljiva, tada nema kontraprimera dužine k (ovo ne znači da nema dužih kontraprimera)
- Ukoliko je zadovoljiva, zadovoljavajuća valuacija nam daje niz stanja, tj. prefiks izvršavanja koji ne zadovoljava svojstvo ϕ

Primer 9



Nastavak prethodnog primera

- Svojstvo: „u svakom trenutku je tačno jedan bit uključen”
- Za svako $i = 0, 1, 2, 3$ važi: $F_i \equiv (p^i \vee q^i \vee r^i) \wedge (\neg p^i \vee \neg q^i) \wedge (\neg p^i \vee \neg r^i) \wedge (\neg q^i \vee \neg r^i)$
- Kodiranje svojstva: $F \equiv F_0 \wedge F_1 \wedge F_2 \wedge F_3$
- Formula čija se zadovoljivost ispituje: $I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge T(s_2, s_3) \wedge \neg F$

Ograničena provera modela

Ograničena provera modela u verifikaciji softvera

- SAT kodiranje je obično dovoljno u verifikaciji hardvera, jer se sve izražava u funkciji bitova u sistemu
- U slučaju softvera, predikati obično izražavaju složena svojstva poput $x < y$, $x \ mod \ y = 3$ i sl.
- Slično, relacija prelaska se opisuje akcijama oblika $x \leftarrow y + z$ i sl.
- Kodirati ove uslove i akcije na jeziku iskazne logike je veoma teško
- Predikati se mogu apstrahovati iskaznim slovima, međutim, time se gubi semantička veza među njima (npr. iz $x < y$ i $y < z$ sledi $x < z$, ali to na iskaznom nivou nije moguće zaključiti)
- Rešenje: korišćenje SMT rešavača

Ograničena provera modela

SMT rešavači

- Predstavljaju nadogradnju SAT rešavača tako da omogućavaju rezonovanje u nekoj odlučivoj teoriji prvog reda
- Najpre se ispituje zadovoljivost formule u iskaznom smislu (zanemarujući značenje predikata u teoriji), a zatim se za dobijeni iskazni model proverava zadovoljivost u teoriji
- Podržava aritmetičke teorije koje su pogodne za izražavanje akcija i uslova u standardnim programima
- Podržava teoriju neinterpretiranih funkcija koja omogućava apstrakciju
- Podržava teorije nizova i rekurzivnih struktura podataka (listi, stabala)
- Podržava teoriju bitvektora za precizno izražavanje semantike tipova u programskim jezicima

Ograničena provera modela

Postupak SMT kodiranja

- Programske promenljive predstavljamo promenljivama u odgovarajućoj teoriji (npr. `int` promenljive možemo predstaviti promenljivama sorte `Int`)
- Lokaciju u programu takođe možemo predstaviti celobrojnom promenljivom
- Sada za stanja $s_0, \ldots, s_k$ imamo $k+1$ kopiju odgovarajućih promenljivih (npr. programska promenljiva x u i-tom stanju biće predstavljena promenljivom x_i u SMT formuli)

Ograničena provera modela

Postupak SMT kodiranja

- Početni uslovi se izražavaju kao formula nad promenljivama u stanju s_0
- Relacija prelaska ima oblik višestruke $if - else - if$ konstrukcije koja razmatra različite lokacije i primenjuje odgovarajuće akcije na tim lokacijama (tj. povezuje promenljive iz tekućeg stanja sa promenljivama iz narednog stanja)
- Svojstva se izražavaju u terminima SMT promenljivih, na sličan način kao i u SAT slučaju

Primer 10

Primer SMT kodiranja

- Neka na lokaciji p imamo naredbu `x = x - 1`
- Neka je l_i promenljiva koja sadrži redni broj programske lokacije u i-tom stanju, a x_i promenljiva koja sadrži vrednost promenljive x u i-tom stanju
- Relacija prelaska će biti izražena konstrukcijom $\mathit{if}\ l_i = p\ \mathit{then}\ x_{i+1} = x_i - 1 \wedge \bigwedge_{y \neq x}(y_{i+1} = y_i) \wedge l_{i+1} = p + 1\ \mathit{else}\ R$, gde je R ostatak $\mathit{if} - \mathit{else} - \mathit{if}$ konstrukcije koji razmatra ostale lokacije
- Ovo znači: „na lokaciji p umanjujemo x za jedan, dok ostale promenljive (označene sa y) ostaju nepromenjene. Prelazimo na sledeću lokaciju"

Primer 10

Napomene

- Primetimo da na ovaj način izbegavamo potpunu apstrakciju predikata: stanja sistema jesu određena vrednostima atomičkih predikata koji se pojavljuju u formuli, ali se njihova semantika ne gubi, jer imamo SMT teoriju koja rezonuje o njima
- Korišćenje aritmetičke teorije je samo aproksimacija stvarne semantike programa, jer su `Int` promenljive u SMT-u nad beskonačnim domenom $\mathbb{Z}$, dok je u računaru domen konačan
- Teorija bitvektora podrazumeva semantiku koja je identična semantici tipova u programskim jezicima
- Korišćenje aproksimacije omogućava da se lakše dokažu neka tvrđenja

Pregled

Zaključak

Zaključak

- Provera modela predstavlja metod verifikacije koji se zasniva na sistematskom ispitivanju putanja u tranzicionom sistemu u potrazi za kontraprimerom koji narušava zadato svojstvo
- Omogućava pronalaženje velikog broja grešaka koje obično testiranje često ne pronalazi
- Zasniva se na matematičkoj logici, algoritmici, teoriji formalnih jezika, teoriji grafova, ...
- Glavni problem: ogroman broj stanja u realnim sistemima
- Rešenja: apstrakcija, simbolička provera modela, ograničena provera modela
- Aktuelni pravac istraživanja: primena SAT i SMT rešavača

Literatura

Zahvalnica

U izradi ovih materijala učestvovao je doc. dr Milan Banković

Literatura

- Biere A., Heule M., van Maaren H. Handbook of satisfiability. 2009.
- Baier C., Katoen J. P., Larsen K. G. Principles of model checking. 2008.
- Merz, S. An introduction to model checking. 2008.