nedeljainformatike v7.0

Logika i vreme: kako naći greške u C programima

Andrej Ivašković

Matematička gimnazija

13. 04. 2022.

Agenda

1. Statička analiza i CBMC
2. Temporalna logika
3. Algoritmi za vršenje analize

Provera tačnosti programa

- Popularni načini provere da li program radi:
  - “Ne proveravam, radi 100%”
  - “Kompajlira se, dakle radi”
  - Testiranje (unit, integracijski, random, stress...)
- **Statička analiza**: ispitivanje tačnosti bez izvršavanja programa

```
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class TestSum {

   // test 5 + 2 = 7
   @Test
   public void testSum1() {
      assertEquals(sum(5, 2), 7);
   }

   // test 10 + (-2) = 8
   @Test
   public void testSum2() {
      assertEquals(sum(10, -2), 8);
   }
}
```

Statička analiza podrazumeva dosta toga!

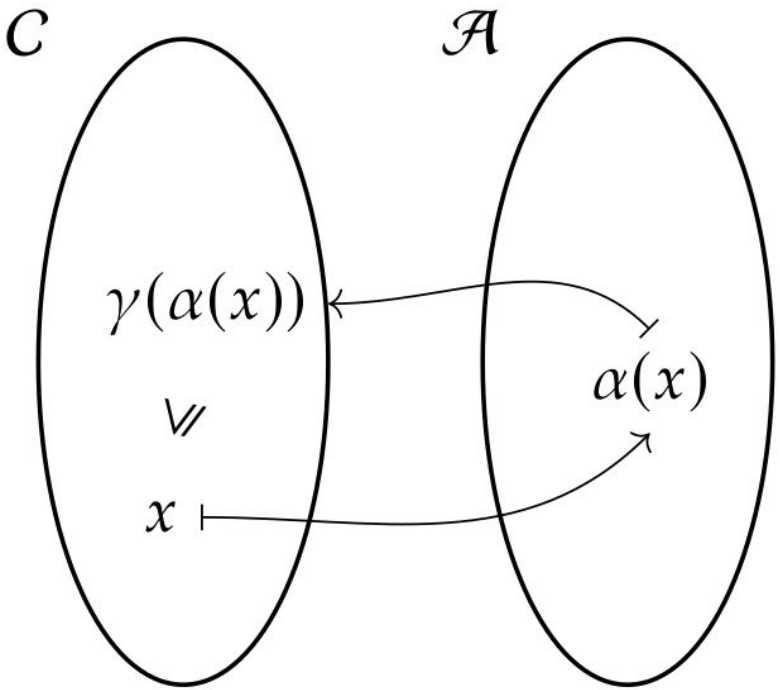
```c
#include <stdio.h>

int main() {
  int x;
  x = x + 1;
  return 0;
}
```

```
(base) ➜  ~ gcc stat.c -Wall
stat.c: In function ‘main’:
stat.c:5:5: warning: ‘x’ is used uninitialized [-Wuninitialized]
    5 |   x = x + 1;
      |   ~~^~~~~~~
```

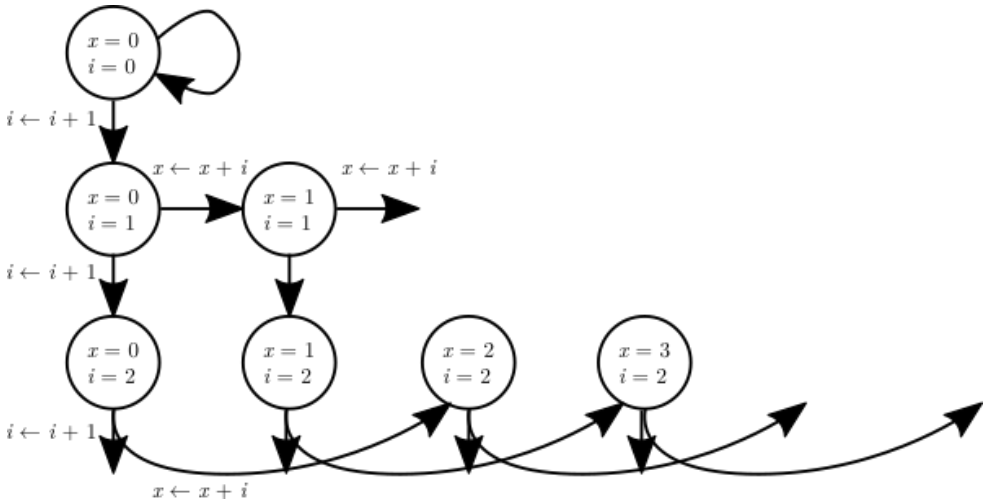
Neke od vrsta statičke analize

- Provera tipova
  - Tipovi podataka nam daju jako važne informacije!
  - Nekada tipovi nisu dati eksplicitno, ali mogu da se izvedu primenom raznih algoritama (npr. unifikacijom)
- Analize toka podataka
  - (provera da li su promenljive inicijalizovana, da li instrukcije mogu da se obrišu)
- **Provera modela**



Provera modela

- Glavna ideja: formulisati problem koristeći **stanja**, **puteve** i **oznake**.
  - Stanje odgovara stanju u programu (vrednosti svih promenljivih, lokacija u programu).
  - Prelazi između stanja odgovaraju pojedinačnim instrukcijama.
  - Putevi odgovaraju različitim putevima u programu.
- Za iskazivanje određenih svojstava se koristi odgovarajuća **temporalna logika**.



Digresija: logika u računarstvu

- Logika se koristi za potrebe opisivanja i dokazivanja važnih svojstava (kao i u matematici).
- Različiti nivoi složenosti logike imaju različite primene:
  - Iskazna logika
    - SAT: NP-kompletan problem.
    - Razijene i optimizovane SAT i SMT tehnologije.
  - Logika prvog reda (FOL)
    - Neodlučiv problem.
    - Odlični dokazivači teorema (npr. Z3).
  - Logika višeg reda (HOL)
    - Koristi se za asistente u dokazivanju teorema (npr. Isabelle).
  - Konstruktivne logike
    - Veza sa teorijom tipova, tema na Nedelji informatike v4.0.
  - Programske logike (Hoare, separation)
    - Takođe tema na Nedelji informatike v4.0!

Kako ovo funkcioniše u praksi?

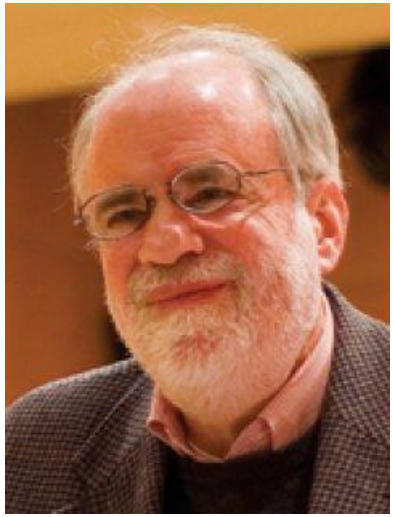
- **CBMC** (**C Bounded Model Checker**)
  - Deo **CProver** (http://www.cprover.org), projekat sa Oksforda.
  - Proverava važna svojstva u programima.
  - Program može da definiše dalja važna svojstva i proveri ih pomoću **assert**-ova.
  - Daje nam i **trag** koji vodi do greške!
- Sledi kratka **demonstracija**.

Agenda

1. Statička analiza i CBMC
2. Temporalna logika
3. Algoritmi za vršenje analize

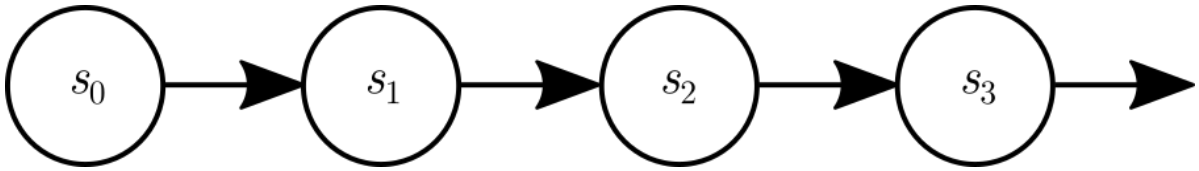
Kripke struktura

- **Kripke struktura** (ili sistem prelaza, model) je, za dati skup $\mathcal{A}$ atomskih iskaza, uređena četvorka
  $M = (S, I, T, L)$, definisana na sledeći način:
  - S je **skup stanja**;
  - $I \subseteq S$ je skup **početnih** (inicijalnih) **stanja**;
  - $T \subseteq S \times S$ je relacija prelaza, $(x, y) \in T$ znači da iz x može da se dođe do y u jednom koraku;
  - $L : S \to \mathcal{P}(\mathcal{A})$ je **funkcija interpretacije** (labeliranja) koja svakom stanju pridružuje skup atomskih iskaza.
- Sve dalje definicije se daju u kontekstu neke Kripke strukture/sistema/modela M.



Saul Kripke
(izvor slike: Princeton)

Putevi



- **Put** π za sistem (S, I, T, L) je beskonačan niz s_0, s_1, s_2, s_3 ... (odnosno $\pi(0), \pi(1), \pi(2), \pi(3), ...$) za koji važi:
  - $s_i \in S$ za $i \in \mathbb{N}$
  - $s_0 \in I$
  - $(s_i, s_{i+1}) \in T$

- Za put π definišemo **sufiks** π_k: put $s_k, s_{k+1}, s_{k+2}, ...$
- Indeks se uglavnom interpretira *vremenski*.

Linearna temporalna logika (LTL)

- **LTL** je jezik kojim opisujemo svojstva puteva i Kripke struktura.
- Za LTL formulu φ koristimo tri bitne oznake valjanosti:
  - $\pi \vDash \varphi$: formula φ važi na putu π
  - $M \vDash \mathbf{A}\varphi$: $\pi \vDash \varphi$ na **svim** putevima π u modelu M
  - $M \vDash \mathbf{E}\varphi$: $\pi \vDash \varphi$ na **nekom** putu π u modelu M
- U okviru ovog predavanja pričamo samo o LTL, ali postoje i dalje zanimljive logike:
  - Computation Tree Logic (CTL)
  - PSL
  - CTL*

Linearna temporalna logika (LTL): sintaksa

- Formula φ u LTL može da ima sledeću strukturu:
  - p, gde je $p \in \mathcal{A}$
  - $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \rightarrow \varphi_2$, $\top$, $\bot$, $\neg\varphi$ (isto značenje kao i u iskaznoj logici)
  - $\mathbf{G}\varphi$ (*globalno*)
  - $\mathbf{F}\varphi$ (*finally*, odnosno "u nekom trenutku u budućnosti")
  - $\mathbf{X}\varphi$ (*neXt*, "naredni")
  - $\varphi_1 \mathbf{U} \varphi_2$ (*until*, "φ_1 sve dok ne φ_2")

Linearna temporalna logika (LTL): semantika

- Formula φ u LTL je valjana u odnosu na π, ($\pi \vDash \varphi$):
  - $\pi \vDash p$, ukoliko $p \in \pi(0)$
  - $\pi \vDash \varphi_1 \wedge \varphi_2$, ukoliko $\pi \vDash \varphi_1$ i $\pi \vDash \varphi_2$
    (slično za ostale veznike)
  - $\pi \vDash \mathbf{G}\varphi$, ukoliko $\pi_i \vDash \varphi$ za **sve** $i \in \mathbb{N}$
  - $\pi \vDash \mathbf{F}\varphi$, ukoliko $\pi_i \vDash \varphi$ za **neko** $i \in \mathbb{N}$
  - $\pi \vDash \mathbf{X}\varphi$, ukoliko $\pi_1 \vDash \varphi$
  - $\pi \vDash \varphi_1 \mathbf{U} \varphi_2$, ukoliko postoji $i \in \mathbb{N}$ takvo da:
    - $\pi_i \vDash \varphi_2$
    - za sve $j \in \mathbb{N}$, $j < i$, važi $\pi_j \vDash \varphi_1$
- **G** i **F** mogu da se izvedu!

Primeri formula

- **F**(`kraj`)
  - Nakon nekog vremena dolazi do kraja.
- **G**¬(`eksplozija`)
  - Nikada neće doći do eksplozije.
- **F**(`alarm` ∧ **X**(`robot_straza`))
  - U nekom trenutku u budućnosti će se uključiti alarm, nakon čega će se aktivirati stražari-roboti.
- **FG***p*
  - Od nekog trenutka u budućnost pa nadalje, *p* uvek važi.
- **GF***p*
  - *p* važi beskonačno mnogo puta.

Formule za datu rečenicu

- Pretpostavimo da imamo atomske iskaze: `deadlock`, `halt`, `power_on` i `enabled`.
- “Program će se ili završiti ili doći u stanje zastoja (*deadlock*).”
  - **F**(`halt` ∨ `deadlock`)
- “Ako je mašina u stanju zastoja, nakon izvesnog vremena će se isključiti napajanje.“
  - **G**(`deadlock` → **F**¬(`power_on`))
- “Ako je napajanje uključeno, mašina može da dospe u stanje u kom nije omogućena i nakon toga je u stanju zastoja.”
  - Ne može da se opiše korišćenjem LTL.
  - Kjučno je “može da dospe u stanje”, što zahteva mogućnost da se iskaže “postojanje puta” .

Agenda

1. Statička analiza i CBMC
2. Temporalna logika
3. Algoritmi za vršenje analize

SAT solving

- **SAT:** Boolean satisfiability
  - Za datu iskaznu formulu odrediti postoji li vrednosti iskaznih slova za koju je formula tačna (**zadovoljiva**).
  - Na primer: $(\neg x \vee y) \wedge (x \rightarrow (x \wedge z))$, zadovoljeno je za $x = \bot$, $y = \top$, $z = \top$.
- SAT je **NP-kompletan** problem.
- **VAL:** Boolean validity
  - Odrediti da li je iskazna formula tačna za sve vrednosti iskaznih slova (tautologija, **valjana**).
  - Na primer: $(x \wedge (x \rightarrow y)) \rightarrow y$.
- Iskazna formula *F* je valjana *ako i samo ako* njena negacija $\neg F$ *nije* zadovoljiva.
  - Odnosno, zadovoljavajući primer je ujedno i kontraprimer negacije.
- VAL je **co-NP-kompletan** problem.

Ograničena provera modela

- Prethodna digresija o SAT je bitna jer želimo da predstavimo problem provere LTL modela kao ulaz za neki SMT solver.
  - Nekada je bilo popularno koristiti BDD strukturu (*binary decision diagrams*).
  - U praksi koristimo daleko bolje algoritme koji *ne zahtevaju* kanonski oblik.
- Prvi korak je da posmatramo **konačnu aproksimaciju** beskonačnih modela.
- Ograničena provera modela određuje tačnost modela uz gornju granicu dužine puta koji se razmatra.
- Postoji kontraprimer za φ dužine najviše $k \Leftrightarrow$ Iskazna formula F je zadovoljiva.

Kodiranje ograničene provere modela

- Neophodno je izvršiti prevod modela M i formule φ u formulu iskazne logike.
- Jednostavan primer: odrediti tačnost $M \vDash \mathbf{AG}p$ za atomski iskaz p.
  - Na primer: `assert(never_fail)` i slično.
  - Negacija: postoji stanje s do kog može da se dođe u najviše k koraka u kom je $p \notin L(s)$.
  - Indeksirajmo stanja u S sa $s_0, s_1, s_2, \ldots, s_n$, a stanja na putu $q_0, q_1, \ldots, q_k$.
  - Pomoćno iskazno slovo:

    $u_{i,j}$: $q_i = s_j$ (i-to stanje na putu je s_j)

  - Konačna formula je konjunkcija kodiranih raznih osobina puteva.

Kodiranje ograničene provere modela (formule)

Za dati model $M = (S, I, T, L)$:

- Na početku smo u jednom od inicijalnih stanja:

$$\bigvee_{i \in I} u_{0,i}$$

- Ni u jednom trenutku nismo u više od jednog stanja:

$$u_{i,j} \rightarrow \bigwedge_{\substack{0 \leq t \leq n \\ t \neq j}} \neg u_{i,t}, \qquad 0 \leq i \leq k,\ 0 \leq j \leq n$$

- Prelazi prate model i relaciju prelaza T:

$$u_{i,j} \rightarrow \bigvee_{(j,t) \in T} u_{i+1,t}, \qquad 0 \leq i \leq k-1,\ 0 \leq j \leq n$$

- U bar jednom od stanja na putu ne važi p (jedini deo koji zavisi od φ):

$$\bigvee_{0 \leq i \leq k} \left(\bigvee_{j \in \{s \mid s \in S \wedge p \notin L(s)\}} u_{i,j} \right)$$

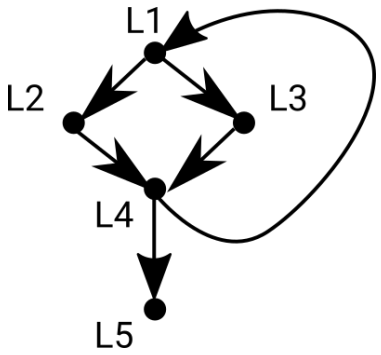
Kodiranje ograničene provere modela (uopšte)

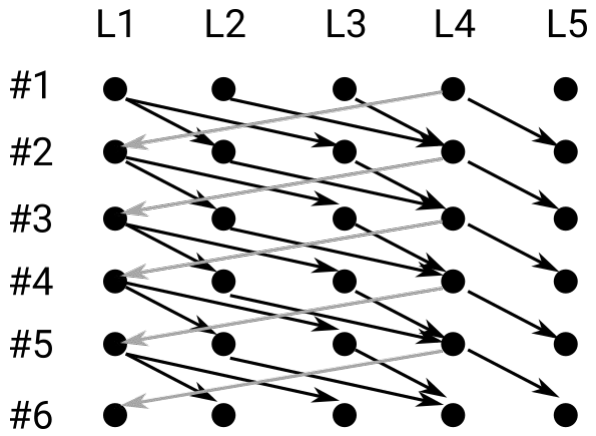
- Poslednja formula u prethodnom kodiranju je način prevoda negacije **G***p*.
- Možemo da definišemo ovakav prevod za ostale vrste formula ...
  - ... ali to nećemo ovde pokazati, probajte sami!
  - $M \vDash$ **AG***p* je dovoljno za veliki broj analiza.
- U praksi se radi nešto malo drugačije ...
  - Biere, Armin & Cimatti, Alessandro & Clarke, Edmund & Zhu, Yunshan. (1999). **Symbolic Model Checking without BDDs**. Proc. of Fifth Int. Conf. on Tools for Construction and Analysis of Systems, TACAS '99. 4144. 10.1007/3-540-49059-0_14.

Formiranje ograničenog modela

- Program se pretvara u graf.
- Koja Kripke struktura odgovara programu?
  - Neophodno je uraditi **loop unrolling** (“odmotavanje ciklusa”) i definisati maksimalan broj koraka.
  - **Stanje** je par (PC, #korak), gde je PC *program counter* (broj instrukcije).
  - Samo jedno **inicijalno stanje**.
  - **Tranziciona relacija** može da se definiše jednostavno i *zamalo* je deterministička.
  - **Atomski iskazi**: vrednosti promenljivih itd.

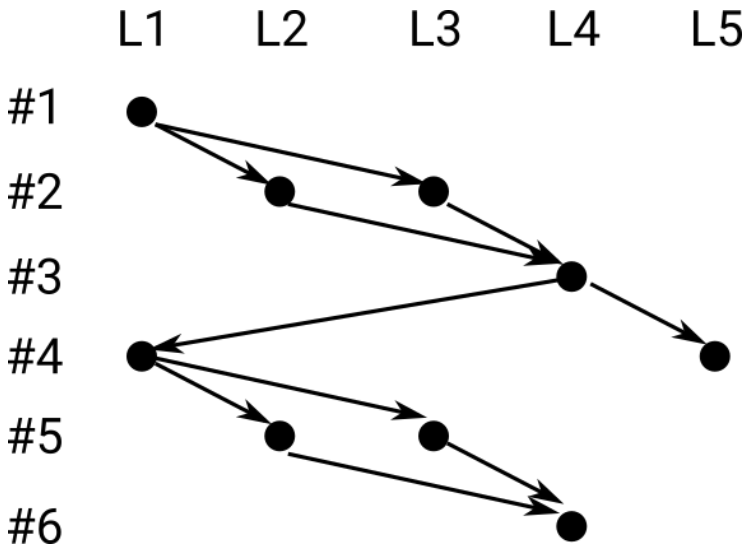
```
L1: IF b1 THEN GOTO L2 ELSE GOTO L3
L2: c1; GOTO L4
L3: c2; GOTO L4
L4: IF b2 THEN GOTO L1 THEN GOTO L5
L5: ...
```





Formiranje ograničenog modela, optimizacija

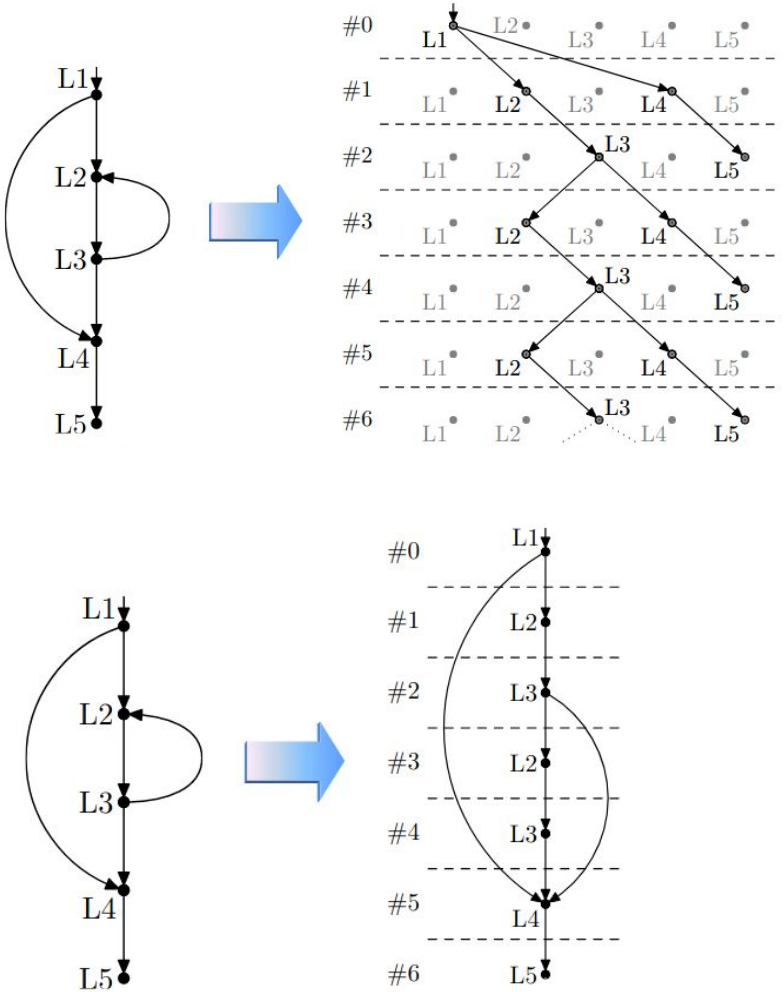
- Do nekih stanja nije moguće stići polazeći od inicijalnog stanja.
  - Ne generišemo delove formula koji pominju ovakva stanja.
- Formula je i dalje nezgodna zbog ciklusa, možemo li vršiti dalju optimizaciju?



Formiranje ograničenog modela, optimizacija 2

- Ako znamo maksimalan broj iteracija ciklusa, svakom trenutku pridružujemo jednu lokaciju.
- Broj koraka *k* mora da bude veći, ali ovo je lakše za SAT solvere.

(izvor slika: CBMC dokumentacija)



Za kraj: SMT

- SAT solver pokušava da nađe tragove kroz program koji ne zadovoljavaju tražena svojstva.
- Nisu svi tragovi validni!
- **SMT solver** (Satisfiability Modulo Theory)
  - Utvrđuje se da li je put moguć.
  - Na osnovu redosleda izvršavanja instrukcija zaključujemo kako se promenljive menjaju u svakom koraku, i samim tim detektujemo nemoguće situacije.
  - Neophodno je da teorija bude **odlučiva** (rešavanje nekih sistema jednačina).

Šta dalje istražiti?

- Razne druge logike:
  - CTL: orijentisano ka stanjima umesto putevima, sadrži kvantifikatore.
  - CTL*: kombinacija LTL i CTL.
- Drugi načini provere modela: NuSMV.
- Alternativne strategije traženja kontraprimera u konačnom modelu:
  - optimizacije u slučaju ciklusa nam dozvoljavaju da koristimo manje *k*.

Pregled predavanja

- Provera modela je praktičan način za otkrivanje bagova u programu.
- Provera modela se vrši korišćenjem raznih temporalnih logika.
- U praksi se koristi konačan model primenom raznih heuristika.
- Rešavanje SAT problema je osnova ovih konačnih metoda.

Hvala na pažnji!

Pitanja?