# Bojenje grafa primjenom logičke zadovoljivosti

**Nizić, Ivan**

SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

ZAVRŠNI RAD br. 1686

# BOJENJE GRAFA PRIMJENOM LOGIČKE ZADOVOLJIVOSTI

Ivan Nizić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

ZAVRŠNI RAD br. 1686

# BOJENJE GRAFA PRIMJENOM LOGIČKE ZADOVOLJIVOSTI

Ivan Nizić

Zagreb, lipanj 2024.

**SVEUČILIŠTE U ZAGREBU**
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

Zagreb, 4. ožujka 2024.

# ZAVRŠNI ZADATAK br. 1686

| | |
|---|---|
| Pristupnik: | **Ivan Nizić (0036532262)** |
| Studij: | Elektrotehnika i informacijska tehnologija i Računarstvo |
| Modul: | Računarstvo |
| Mentor: | izv. prof. dr. sc. Bruno Blašković |

Zadatak: **Bojenje grafa primjenom logičke zadovoljivosti**

Opis zadatka:

Programiranje s ograničenjima jedan je od postupaka koji se koristi pri rješavanju problema s bojenjem grafa. U redu je potrebno realizirati programsko pomagalo koje će rješavati problem bojenja grafa primjenom zadovoljivosti. U tu svrhu potrebno je koristiti programsko pomagalo yices. Za ilustraciju koristit će se slagalice Sudoku i Kakuro. Nadalje je potrebno razmotriti primjenu bojenja grafa u dubinskoj analizi procesa. Posebnu pažnju treba posvetiti konzistentnosti zadavanja problema kao i kompleksnosti izvođenja.

Rok za predaju rada: 14. lipnja 2024.

## Sadržaj

# Uvod

U ovom radu će se složeni i apstraktni problem bojanja grafa pojednostavniti i poistovjetiti s rješavanjem Sudoku igre, odnosno popunjavanjem Sudoku tablice. Rješenje temeljnog problema bojanja grafa biti će prikazano koristeći tri različita alata. Prvi alat koji će se koristiti je programiranje s ograničenjima (engl. Constraint Programming), zatim će rješavanje problema biti ilustrirano koristeći zadovoljivost (engl. Satisfiability) te će na posljetku biti prikazano rješavanje problema koristeći alat SMT. Prije rješavanja problema primjenom alata, svaki alat će biti posebno predstavljen te će se prikazati njihove međusobne razlike i sličnosti, budući svaki alat rješava različite probleme s različitom učinkovitošću. Problem bojanja grafa biti će prikazan koristeći Sudoku tablicu. Alat koji će omogućiti rješavanje Sudoku tablice programskim putem je Yices, alat SMT oblika. Koristeći Yices pomagalo biti će moguće riješiti zadanu Sudoku tablicu. Datoteka *.ys* će sadržavati Sudoku problem definiran u SMT obliku te će se generirati pomoću Perl skripte. Ta datoteka će na temelju početnih vrijednosti tablice i pravila Sudoku igre pokušati pronaći vrijednosti ostalih ćelija tablice, ako je to moguće. Osim u datoteci *.ys*, Sudoku problem biti će definiran i u datoteci C jezika. Alat Yices, točnije biblioteke koje on nudi za rad u višim programskim jezicima, omogućit će definiciju Sudoku problema koristeći jezik C.

# 1. Programiranje s ograničenjima i zadovoljivost

## 1.1. Programiranje s ograničenjima

Programiranje s ograničenjima (engl. **C**onstraint **P**rogramming - CP) je pristup programiranju kod kojeg središnju ulogu imaju ograničenja, što se vidi i iz samog imena. Rješenja problema dobivena ovim pristupom moraju zadovoljavati određeni skup ograničenja. Kako bi se došlo do rješenja problema koristeći CP potrebno je provesti sljedeće korake [1]:

1. Modelirati problem
2. Riješiti problem
3. Analizirati rješenje.

### 1.1.1. Modeliranje problema

Modeliranje problema obuhvaća definiciju skupa varijabli čije se vrijednosti traže u rješenju, zatim je potrebno definirati domene tih varijabli, naposljetku se definiraju ograničenja koja varijable moraju zadovoljiti. Varijable koje se nalaze unutar nekog ograničenja spadaju u njegov prostor. Ovisno o broju varijabli, unutar prostora ograničenja, ograničenja se dijele na unarne, jedna varijabla u prostoru, te *n*-arne, *n* varijabli u prostoru. [1]

### 1.1.2. Rješavanje problema

Rješavanje problema provodi se pomoću CP alata. Alat tijekom rješavanja problema stvara moguće svjetove, pridjeljujući vrijednosti iz domena svim varijablama. Svijet u kojem su sva ograničenja zadovoljena zove se model problema. Koristeći CP alat moguće je riješiti sljedeće probleme [1]:

- Ispitati postojanje modela zadanog problema
- Pronaći model
- Pronaći sve modele
- Prebrojati modele
- Pronaći najbolji model, optimizacija.

Ovisno o željenom rješenju i implementaciji CP alata, pri pronalasku rješenja alat može koristiti razne tehnike pretraživanja prostora mogućih svjetova kako bi došao do modela. Najzastupljenije tehnike su: propagacija ograničenja i povratno pretraživanje.

Propagacija ograničenja kao glavnu zadaću ima smanjiti prostor pretraživanja i time ubrzava pronalazak rješenja. Smanjenje prostora pretraživanja se postiže eliminacijom vrijednosti iz domena koje nisu u skladu s ograničenjima, a time ne mogu biti dio modela. Eliminacija završava kada se dođe do rješenja ili kada nema vrijednosti koje se mogu eliminirati. Ukoliko daljnja eliminacija nije moguća potrebno je provesti pretraživanje. [2]

Povratno pretraživanje (engl. backtracking) je sustavno pretraživanje svih mogućih svjetova u potrazi za modelom/ima. Pretraživanje se provodi na principu postupne gradnje djelomičnog rješenja te provjere ograničenja na tom rješenju. Ukoliko djelomično rješenje ne zadovoljava ograničenja, algoritam se vraća na mjesto prošle odluke te se odlučuje na različitu odluku. Budući prostor pretraživanja može biti jako velik, njega se može smanjiti propagacijom ograničenja. Kombinirajući te dvije tehnike se pri izboru vrijednosti varijable iz domena ostalih varijabli eliminiraju vrijednosti koje su u kontradikciji s ograničenjima. [2]

Redoslijed odabira varijabli koje se promatraju može biti nasumičan ili vođen nekim principom. Princip odabira varijable je odabir one varijable koja je najograničenija (engl. **M**ost **R**emaining **V**alues), tj. potrebno je odabrati onu varijablu koja ima najmanje mogućih vrijednosti, što smanjuje faktor grananja. Slično kao i kod odabira varijable, odabir vrijednosti koja se pridružuje varijabli, može biti nasumičan ili vođen principom. Odabire se vrijednost koja će eliminirati najmanje vrijednosti iz domena ostalih varijabli (engl. **L**east **C**onstraining **V**alue), što pruža veću fleksibilnost pri odabiru vrijednosti za druge varijable. Iako se izrečeno može činiti kontradiktorno, u stvarnosti ubrzava dolazak do rješenja. Kada dođe red na varijable s velikim domenama, do tada će propagacija ograničenja te domene smanjiti, a LCV povećava šansu za rani pronalazak rješenja. [3]

### 1.1.3. Analiza rješenja

Nakon što je CP alat pronašao rješenje problema, ako ono postoji, po potrebi to rješenje se može preoblikovati u prihvatljiviji oblik, prilagođen problemu koji se rješavao. Također se analizira i optimalnost rješenja ukoliko je ona bitna. [2]

### 1.1.4. Primjena CP-a

CP se može koristiti pri rješavanju raznih problema iz kombinatorike i logike te optimizacije mreža, također je koristan u bojanju grafova, više o tome u nastavku. Najpoznatiji primjeri uporabe CP-a su: raspored 8 kraljica na šahovskoj ploči, bojanje karte, Sudoku, Kakuro te ostale logičke zagonetke. U poglavljima 2.2.3. i 3.4. se nalaze konkretni primjeri uporabe CP, pri bojanju grafa i rješavanju Sudoku tablice.

## 1.2. Zadovoljivost

Zadovoljivost (engl. Satisfability- SAT) je jedan od fundamentalnih problema kojima se bave teorija računarstva i matematička logika. [1]

### 1.2.1. SAT

Za rješavanje problema koristeći SAT, potrebno je problem definirati kao formulu propozicijske logike u konjunktivnom normalnom obliku (1). Kao i kod modeliranja problema kod CP-a potrebno je definirati varijable i njihove domene. Za razliku od CP-a, varijable kod SAT-a su varijable Booleove logike, što znači da su njihove domene sačinjene od dvije mogućnosti: istina i laž. Također se kao i kod CP-a definiraju ograničenja koja moraju biti zadovoljena u rješenju. Oblikujući ograničenja u disjunkcije literala te potom spajajući ih u konjunkciju, dolazi se do formule zapisane u konjunktivnom normalnom obliku, koja je onda spremna za rješavanje SAT alatom. Dodatna sličnost s CP-om je mogućnost rješavanja različitih tipova problema: pronalazak modela ili provjeravanje postojanja modela za dani problem. Nakon što je definirana formula kojom je opisan problem, formula se prosljeđuje na rješavanje SAT alatu. Cilj alata je pronaći raspodjelu istinitosnih vrijednosti varijabli tako da formula bude istinita, a samim time i ograničenja koja proizlaze iz problema. Konkretni primjeri primjene SAT alata na rješavanje problema slijede u nastavku. [4]

$$\bigwedge_{i=1}^{n}(x_i \vee x_{i+1} \vee x_{i+2}) \tag{1}$$

### 1.2.2. Primjena SAT-a

SAT je moguće primijeniti u mnogim područjima stvarnog svijeta. U računarstvu se SAT alati primjenjuju za provjeru ispravnosti softverskih i hardverskih dijelova sustava te optimizaciju istih. SAT se koristi i pri rješavanju problema kombinatorike i logike, poput problema najkraćeg puta. Svakodnevni problemi poput izrade rasporeda se također mogu riješiti primjenom SAT alata. Prilikom rješavanja problema SAT se može koristiti različitim algoritmima, najpoznatiji su:

- WalkSAT
- DPLL (**D**avis-**P**utnam-**L**ogemann-**L**oveland)
- CDCL (**C**onflict-**D**riven **C**lause **L**earning).

*WalkSAT* obavlja pretraživanje prostora vođeno heuristikom. DPLL je algoritam koji koristi povratno pretraživanje te širenje i brisanje literala, slično propagaciji ograničenja. CDCL je naprednija verzija DPLL-a te uključuje učenje iz konfliktnih stanja i koristi heuristiku pri pretraživanju. U poglavljima 2.2.4. i 3.5. se nalaze konkretni primjeri uporabe CP, pri bojanju grafa i rješavanju Sudoku tablice.

## 1.3. SMT

Budući neke probleme nije moguće riješiti koristeći samo SAT i propozicijsku logiku te je zbog toga SAT proširen na SMT (**S**atisfiability **M**odulo **T**heories). SMT, osim Booleovih varijabli, uključuje i druge matematičke i logičke teorije, poput aritmetike, teoriju skupova, nizova i drugih. Glavni zadatak SMT-a je odlučivanje o zadovoljivosti formule prvog reda, uzimajući u obzir teorije potrebne za rješavanje problema. Za to ima dva specijalizirana dijela: SMT, koji rješava Booleove probleme, dok teoretski dio rješava probleme drugih teorija. Budući SMT uzima u obzir i druge teorije osim Booleove, u formuli čija zadovoljivost se provjerava, osim Booleovih varijabli i operatora mogu se naći i druge varijable i operatori, npr. aritmetički (2). [5]

$$(x + 2f(x) \geq 12\ ) \wedge (x \neq 5) \tag{2}$$

### 1.3.1. Lijeni pristup SMT-u

Kod lijenog pristupa, SMT alatu se šalje formula za koju se treba pronaći zadovoljivo rješenje. Pri traženju rješenja teorija informacija koristi "lijeni" pristup, tj. ne navodi pretraživanje, nego pretražuje sve moguće svjetove dok ne pronađe model. Lijeni pristup ima i svoje prednosti, a to su: velika fleksibilnost u problemima koji se mogu rješavati, dobra raspodjela posla na dijelove te se SAT uz 40-ak linija koda pretvara u lijeni SMT. [5]

Lijeni pristup je također moguće optimizirati kako bi mu se povećala učinkovitost pri traženju rješenja. Prvo se umjesto provjere zadovoljivosti potpunih svjetova, odvijaju provjere zadovoljivosti djelomičnih svjetova. Ti se svjetovi mogu odbaciti ukoliko ne zadovoljavaju formulu, čime je izbjegnuto nekoliko koraka pridjeljivanja vrijednosti varijablama, a time je skraćeno ukupno vrijeme pretrage. Ukoliko se pokaže da raspored vrijednosti $M$ ne zadovoljava formulu, umjesto da se samo (3) doda u formulu, potrebno je pronaći (4), koji također ne zadovoljava formulu te dodati (5) u formulu. Kada se pokaže da raspodjela ne zadovoljava formulu, potrebno je ustvrditi gdje dolazi do konflikta i vratiti se na to mjesto, donekle slično pretraživanju unatrag.

$$\neg M \tag{3}$$

$$M_0 \subseteq M \tag{4}$$

$$\neg M_0 \tag{5}$$

### 1.3.2. DPLL(T)

Moguće je popraviti i drugi nedostatak lijenog pristupa, tj. pretraživanje bez navođenja. Pretraživanje je moguće navoditi koristeći dio zadužen za rješavanje teorijskih dijelova formule. Iako je to moguće ostvariti jednostavnim alatima, za učinkovito pretraživanje potrebni su specijalizirani alati. Taj pristup se naziva DPLL(T). Algoritam DPLL(T) po mnogome sliči na CDCL SAT algoritam. Osnovne operacije koje algoritam obavlja su:

- Širenje prostora pretraživanja, SAT dio
- Provjera zadovoljivosti, teoretski dio
- Širenje teorije, teoretski dio.

Prilikom izvođenja tih operacija algoritam prvo izvodi jeftinije operacija te se neke operacije mogu preskočiti ukoliko su preskupe. SAT dio, osim širenja, provodi i analizu konflikata, tj. analizu mjesta na kojima formula nije zadovoljena. Teoretski dio je, osim za gore navedeno, zadužen i za provođenje pretraživanja unatrag. [5]

### 1.3.3. Yices

Yices je jedan od programskih pomagala koji se služe SMT-om pri rješavanju problema. Glavna zadaća Yices alata je odrediti zadovoljivost zadane formule, pri čemu na raspolaganju ima teorije aritmetike, nizova, vektora, skalara i n-torki. Yices 2 ima podršku i za linearnu i nelinearnu aritmetiku. Programi izvođeni u alatu Yices, pisani su u SMT-LIB notaciji, kod Yices 2 alata moguće je uporaba posebnog jezika alata. Yices 2 također nudi biblioteke koje omogućavaju korištenje Yices alata u višim programskim jezicima, poput Python-a i C-a.

U ovom radu Yices je korišten kao pomagalo koje pronalazi rješenja zadane Sudoku tablice na temelju početnih vrijednosti i definiranih ograničenja, tj. pravila Sudoku igre, primjer u poglavlju 3.5. Osim standardne primjene Yices alata, u radu su korištene i biblioteke koje Yices 2 nudi za pisanje programa u jeziku C, koji također rješava zadanu Sudoku tablicu.

## 1.4. Usporedba CP, SAT i SMT

Svaki od ovih pristupa rješavanju problema ima svoje prednosti i nedostatke. Učinkovitost pojedinog pristupa također ovisi o vrsti problema te vrsti rješenja koje se traži.

Prednosti CP-a su [5]:

- Mijenjanjem i dodavanjem ograničenja se model lako može prilagoditi novom problemu
- Prirodno izražavanje i modeliranje problema
- Korištenje specijaliziranih algoritama za mnoga ograničenja poboljšava učinkovitost rješavanja pojedinih problema.

Nedostatci CP-a su [5]:

- Pristranost umjetnim, ne realističnim problemima
- Nema učenja, povratno pretraživanje umjesto skakanja na mjesto konflikta
- Potreba za ručnim namještanjem heuristika za svaki problem.

Prednosti SAT-a su [5]:

- Velika uspješnost pri rješavanju problema iz stvarnog života
- Jednostavan i automatiziran izbor varijabli pri pronalasku modela
- Rješavanje problema se svodi na tumačenje ograničenja
- Učenje iz pronađenih stanja koja ne zadovoljavaju ograničenja.

Nedostatci SAT-a su [5]:

- Nemogućnost primjene na probleme koji nisu primjeri Booleove logike
- Teško rješavanje optimizacijskih problema, veći naglasak na zadovoljivosti
- Jezik niske razine te su mu potrebno modeliranje i prevođenje u jezike više razine.

Prednosti SMT-a su [5]:

- Mogućnost rješavanja problema koji sadrže elemente različitih teorija
- SAT rješavač proširen rješavačima raznih teorija matematike i logike omogućuje rješavanje složenih problema.

Nedostatci SMT-a su:

- Zbog mogućnosti kombiniranja različitih teorija, sintaksa izražavanja formula je složenija.

Općenito gledajući CP obuhvaća probleme zadovoljivosti, tj. pronalaska rješenja i probleme optimizacije, tj. najboljeg rješenja na temelju zadanih ograničenja. CP najviše uspjeha ima prilikom rješavanja problema s ograničenim brojem varijabli i ograničenim domenama. SAT se uglavnom bavi problemima zadovoljivosti, tj. pronalaska rješenja ako ono postoji. SMT je proširenje SAT-a, koje osim Booleove logike na raspolaganju ima i mnoge teorije matematike i logike, zbog toga se može primijeniti na veći broj složenih problema, u odnosu na SAT. [6]

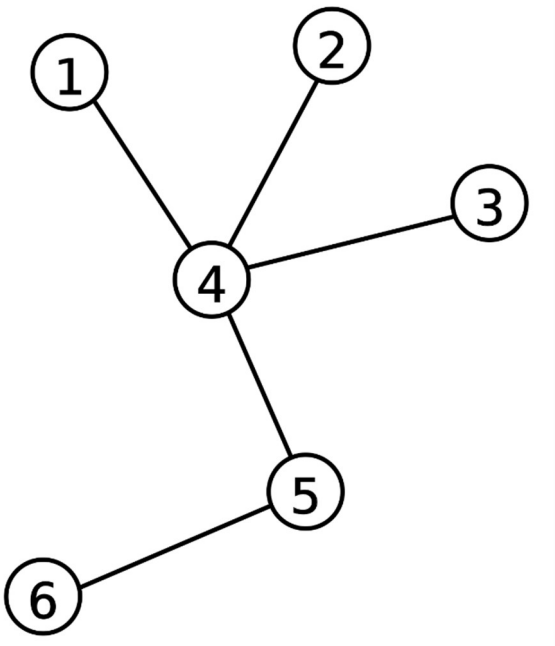
# 2. Grafovi

## 2.1. Definicija grafa

Graf je predstavljen skupom vrhova (engl. **V**erticles) i skupom bridova (engl. **E**dges ), svaki brid je skup dva međusobno povezana vrha. Tu definiciju grafa moguće je zapisati i u matematičkom obliku (1). Za dva vrha vrijedi da su susjedni ako su povezani bridom. Broj bridova koji su incidentni s nekim vrhom određuje stupanj tog vrha (2). [7]

$$G = \{V, E\} \tag{1}$$

$$\deg(v) = \begin{cases} 0, izolirani\ vrh \\ 1, krajnji\ vrh \\ k, k \in \mathrm{N} \end{cases} \tag{2}$$




Slika 2.1 Graf [8]

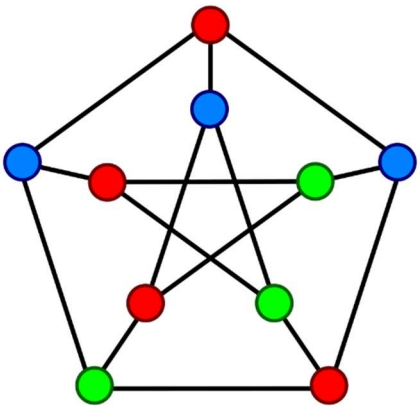
## 2.2. Bojanje grafa

### 2.2.1. Kratka povijest

Problem bojanja grafa pojavljuje se već u 19. stoljeću, kada se počinju crtati prve karte u Europi. Tijekom crtanja tih karata javlja se problem bojanja država. Države je trebalo obojati različitim bojama ukoliko dijele granicu. [7]

### 2.2.2. Bojanje vrhova

Bojanje grafa $G$ je pridruživanje nekog skupa boja skupu vrhova od $G$, tako da je svakom vrhu pridružena jedna boja, uz poštivanje pravila da su različite boje pridružene dvama susjednim vrhovima. Najmanji broj boja potrebnih za bojanje svih vrhova grafa, poštujući sva navedena pravila, zove se kromatski broj grafa te se označava s $\chi(G)$. Glavni problem kod bojanja grafova je pronalazak kromatskog broja zadanog grafa. [7]



Slika 2.2 Obojani graf [9]

### 2.2.3. Bojanje grafova i CP

CP pristup problemu bojanja grafa se pokazao kao učinkovito rješenje. Najviše uspjeha CP postiže pri provjeri postojanja k-bojanja za dani graf. Pri definiciji problema bojanja grafa korištenjem CP logike, potrebno je odrediti varijable, domenu i skup ograničenja problema. Varijable su vrhovi grafa koje je potrebno obojati, domena su boje kojim se vrhovi mogu obojati, a ograničenje je da se dva susjedna vrha moraju obojati različitom bojom. [10]

### 2.2.4. Bojanje grafova i SAT

Problem bojanja grafa može se riješiti primjenom SAT alata. Kako bi se to postiglo, problem bojanja grafa mora se svesti na formulu propozicijske logike, točnije u konjunktivnu normalnu formulu. Prvo se za svaki vrh $v$ grafa i svaku boju $i$ iz konačnog skupa boja definira Booleova varijabla $p_{v,i}$, koja je istinita kada je vrh $v$ obojen bojom $i$, a lažna inače. Potom se definiraju ograničenja: svaki vrh $v$ je obojan barem jednom bojom (3), također je potrebno osigurati da je svaki vrh obojan samo jednom bojom (4), naposljetku susjedni čvorovi moraju biti obojani različitim bojama (5). Dobivene formule se potom predaju SAT alatu, koji provjerava je li graf $k$-obojiv. [11]

$$\bigvee_{1 \le i \le k} p_{v,i} \qquad (\forall v \in V) \tag{3}$$

$$\bigwedge_{1 \le i \le j \le k} \neg p_{v,i} \vee \neg p_{v,j} \qquad (\forall v \in V) \tag{4}$$

$$\bigwedge_{1 \le i \le k} \neg p_{v,i} \vee \neg p_{u,i} \qquad (\forall (v,u) \in V) \tag{5}$$

### 2.2.5. Bojanje grafova u stvarnom svijetu

Problem bojanja grafa danas ima širok prostor primjene u raznim dijelovima života. Osim već spomenute primjene u kartografiji, pri bojanju susjednih regija, bojanje grafa može se primijeniti i pri rješavanju Sudoku igre, o tome više riječ u nastavku rada. Računalna znanost se također oslanja na problem bojanja grafa pri analizi procesa i algoritama te optimizaciji istih. Komunikacijske mreže su na najprimitivnijoj razini grafovi, vrhovi predstavljaju uređaje koji se koriste za komunikaciju, a bridovi vezu između uređaja. Nadalje, problem bojanja grafa se osim u složenijim sustavima može primijeniti i na trivijalne probleme poput sastavljanja rasporeda za ispite i dvorane, rasporede polijetanja i slijetanja na pistu u zračnoj luci i sl.

# 3. Sudoku

Sudoku je popularna igra s brojevima koju je danas moguće pronaći u gotovo svim tiskanim novinama i časopisima. Za one najstrastvenije igrače čak i postoje natjecanja u rješavanju Sudoku.

## 3.1. Pravila igre

Igra Sudoku može se pronaći u mnogim varijantama (leptir, cvijet, par-nepar i mnoge druge), u ovom radu razmatra se klasična varijanta s tablicama veličine $N\,x\,N$ uz (1). Glavni cilj igre je popuniti tablicu brojevima od 1 do $N$, uz pravilo da se svaki od tih brojeva smije samo jednom pojaviti u svakom retku, stupcu i posebno označenom kvadratu veličine $n\ x\ n$, uz (2).

$$N = \{4, 9, 16\} \tag{1}$$

$$n = \sqrt{N} \tag{2}$$

Kako igra ne bi bila dosadna i lagana, početna Sudoku tablica dolazi djelomično popunjena. Zbog te djelomične popunjenosti Sudoku može biti manje ili više zahtjevan. Dobar Sudoku ima samo jedno rješenje, što se postiže pažljivim odabirom početnih vrijednosti. Drugi važni preduvjet jedinstvenosti rješenja je i broj potrebnih početnih vrijednosti. Taj broj još uvijek nije konačno definiran te se trenutno smatra da je 17 najmanji broj početnih vrijednosti za koje se može pronaći jedinstveno rješenje Sudoku tablice. Jedinstvenost rješenja je također bitna budući je broj mogućih Sudoku tablica veličine 9*x*9 jednak 6,670,903,752,021,072,936,960.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | | 7 | | | | |
| 6 | | | 1 | 9 | 5 | | | |
| | 9 | 8 | | | | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | 2 | 8 | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |

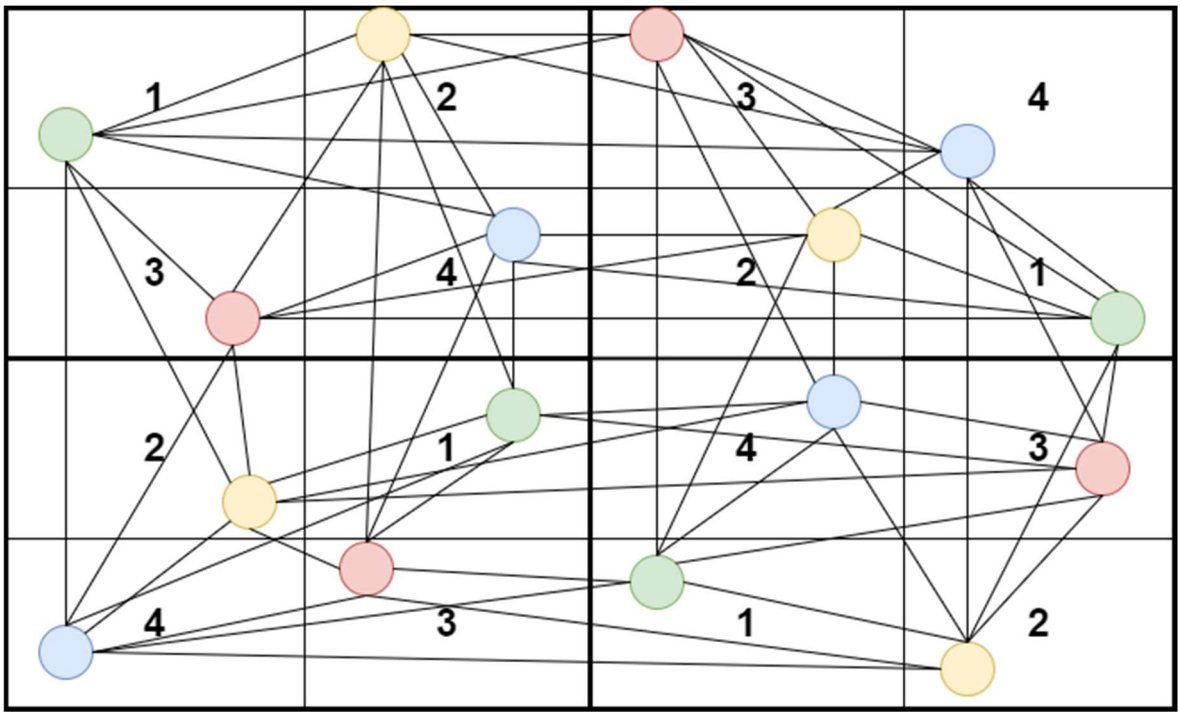
Slika 3.1 Početna Sudoku tablica [12]

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | 4 | 6 | 7 | 8 | 9 | 1 | 2 |
| 6 | 7 | 2 | 1 | 9 | 5 | 3 | 4 | 8 |
| 1 | 9 | 8 | 3 | 4 | 2 | 5 | 6 | 7 |
| 8 | 5 | 9 | 7 | 6 | 1 | 4 | 2 | 3 |
| 4 | 2 | 6 | 8 | 5 | 3 | 7 | 9 | 1 |
| 7 | 1 | 3 | 9 | 2 | 4 | 8 | 5 | 6 |
| 9 | 6 | 1 | 5 | 3 | 7 | 2 | 8 | 4 |
| 2 | 8 | 7 | 4 | 1 | 9 | 6 | 3 | 5 |
| 3 | 4 | 5 | 2 | 8 | 6 | 1 | 7 | 9 |

Slika 3.2 Riješena Sudoku tablica [12]

## 3.2. Sudoku i bojanje grafa

Sudoku tablica se može prikazati u obliku grafa, pri čemu jedno polje Sudoku tablice odgovara jednom vrhu grafa. Vrhovi, tj. polja, su susjedni ako predstavljaju polja unutar istog retka, stupa ili označenog kvadrata. Za bojanje takvog grafa na raspolaganju je N boja, koje su predstavljene znamenkama od 1 do N. Radi jednostavnosti u prikazu će se koristiti tablica veličine 4 *x* 4 (Slika 3.1).




Slika 4.3.3 Sudoku u obliku grafa

## 3.3. Sudoku i CP

Sudoku problem može se riješiti koristeći programiranje s ograničenjima. Popust ostalih problema rješivih s tim pristupom, rješavanje i ovog problema zahtjeva provođenje standardnih koraka CP-a za dobivanje rješenja. Modeliranjem problema definiraju se varijable, *NxN* varijable koja predstavljaju *NxN* čelija tablice *NxN*. Domene varijabli su prirodni brojevi od 1 do N. Na varijable se stavlja 3*N *alldiferent* ograničenja, koja osiguravaju da su vrijednosti u N redova, N stupaca i N kvadrata različite. Unutar ograničenja se također postavljaju i početne vrijednosti. Nakon modeliranja problema model se prosljeđuje CP alatu na rješavanje, izbor tehnike pronalaska rješenja u ovom slučaju ovisi o implementaciji alata. Nakon što alat pronađe rješenje, ako ono postoji, dobiveno rješenje

se preoblikuje u oblik koji zadovoljava Sudoku tablicu, kako bi ga bilo jednostavnije čitati. [13]

## 3.4. Sudoku i SAT

Kako bi Sudoku bila rješiva koristeći SAT alate, prvo ju je potrebno pretvoriti u formulu propozicijske logike, koja je istinita onda i samo onda ako Sudoku ima rješenje. Tako oblikovana formula je onda predana SAT alatu, koji pronalazi rješenje, ako ono postoji, koje se potom preoblikuje u Sudoku.

### 3.4.1. Pretvorba u SAT

Za ilustraciju pretvorbe koristiti će se alat Isabelle, koji se služi logikom višeg reda. Skup od 9 ćelija tablice predstavljen je s $x_1,\ldots,x_9$ te je valjan onda i samo onda kada sadrži svaku znamenku od 1 do 9, jednadžba (3).

$$valid(x_{1,\ldots,}x_9) \equiv \bigwedge_{d=1}^{9} \bigvee_{i=1}^{9} x_i = d \tag{3}$$

Nadalje jednadžbu (3) je potrebno prilagoditi potrebama Sudoku problema, a to znači da se (3) mora primijeniti na svaki stupac, redak i kvadrat, nakon čega dolazimo do sljedećeg (4):

$$\begin{aligned} valid\left(\{x_{ij}\}_{i,j\in\{1,\ldots,9\}}\right) \equiv\ & \wedge_{i=1}^{9} valid(x_{i1},\ldots,x_{i9})\ \wedge \\ & \wedge_{j=1}^{9} valid(x_{1j},\ldots,x_{9j}) \wedge \\ \wedge_{i,j=\{1,4,7\}} valid(x_{ij}, & x_{i(j+1)}, x_{i(j+2)}, \ldots, x_{(i+2)j}, x_{(i+2)(j+1)}, x_{(i+2)(j+2)}) \end{aligned} \tag{4}$$

Potom Sudoku definiramo sa $9^3 = 729$ Booleovih varijabli, svaka ćelija po 9 varijabli. Svaka od tih varijabli $p_{ij}^d$ je istinita ako je jednadžba (5) istinita. Formulama (6) je osigurano da ćelija $x_{ij}$ ima samo jednu vrijednost u danom trenutku.

$$x_{ij} = d \tag{5}$$

$$\bigwedge_{1\le d\le d^{"}\le 9} \neg p_{ij}^{d} \vee \neg p_{ij}^{d^{"}} \tag{6}$$

Kako bi se povećala učinkovitost SAT alata, jednadžba (3) preoblikovati će se u (7). Dok (3) prevedena u SAT daje 9 formula, (7) prevedena u SAT daje 324 formule, ali duljine 2

literala. Time dolazimo do brojke od 11745 formula, ukupno gledajući: 81 formula duljine 9 literala, kojim se definiraju varijable, 81*36 formula za jedinstvenost varijabli te 27*324 formule za različitost vrijednosti u redovima, stupcima i kvadratima. Budući su vrijednosti u nekim ćelijama (barem 17 ćelija) već poznate, broj potrebnih formula i Booleovih varijabli je znatno manji od 729 i 11745. Nakon potrebnih pretvorbi, formule i varijable se predaju SAT alatu koji potom pronalazi rješenje Sudoku problema, ako ono postoji. [4]

$$valid(x_1, \dots, x_9) \Leftrightarrow \bigwedge_{1 \le i \le j \le 9} x_i \neq x_j \Leftrightarrow \bigwedge_{1 \le i \le j \le 9} \bigwedge_{d=1}^{9} x_i \neq d \vee x_j \neq d \quad (7)$$

## 3.5. Sudoku i SMT

Slično kao i u slučaju sa SAT alatom, Sudoku problem je potrebno pretvoriti u logičke izraze koji se mogu čitati SMT alatom. U ovom slučaju upotrjebljeno je pomagalo Yices. Na početku potrebno je definirati varijable. Varijabla N predstavlja veličinu tablice (u primjeru tablica veličine *9x9*), a S predstavlja sumu retka, stupca ili kvadrata, s T je označena minimalna vrijednost varijabli.

```
(define N::int 9)
(define S::int (* (/ (+ N 1) 2) N))
(define T::int 1)
(define x1_1::int)(define x1_2::int)
…… do x9_9
```

Potom se definiraju poznate varijable.

```
(assert (= x1_1 3))
…… za ostale vrijednosti
```

Nakon toga potrebno je postaviti ograničenja na varijable. U ovom slučaju varijable mogu poprimiti vrijednosti od 1 do 9 (od *T* do *N*).

```
(assert (<= x1_1 N)) (assert (<= T x1_1))
(assert (<= x1_2 N)) (assert (<= T x1_2))
…… do x9_9
```

Svaki red, stupac i označeni kvadrat mora sadržavati različite vrijednosti, budući je to osnovno pravilo Sudoku.

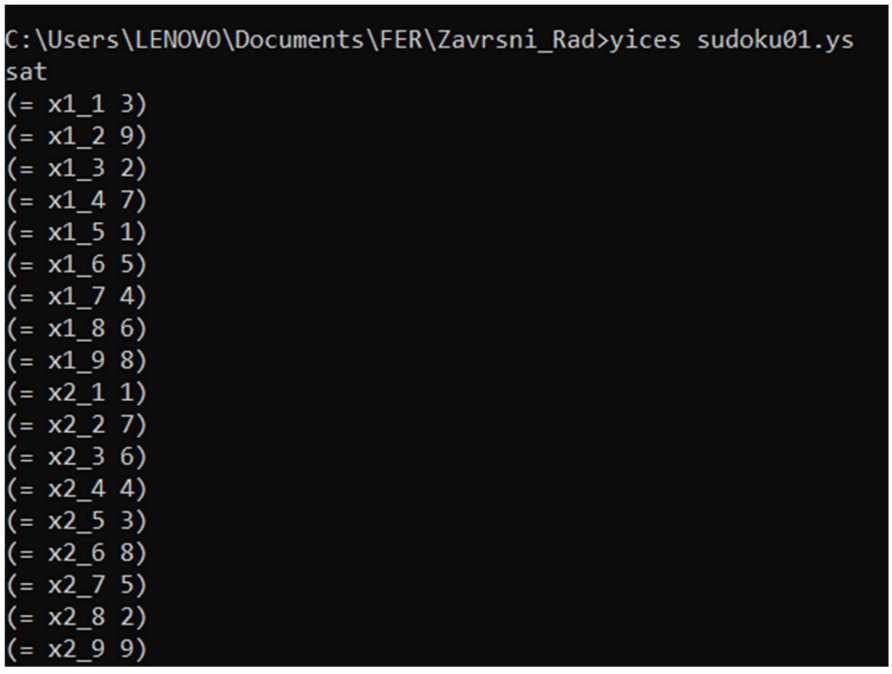
```
(assert (distinct x1_1 x1_2 x1_3 x1_4 x1_5 x1_6 x1_7 x1_8 x1_9))
…… za sve retke
(assert (distinct x1_1 x2_1 x3_1 x4_1 x5_1 x6_1 x7_1 x8_1 x9_1))
…… za sve stupce
(assert (distinct x1_1 x1_2 x1_3 x2_1 x2_2 x2_3 x3_1 x3_2 x3_3))
…… za sve kvadrate
```

Osim uobičajenih pravila Sudoku igre, moguće je definirati i dodatna pravila poput sume retka, stupca i kvadrata. Dodavanje tog pravila može smanjiti vrijeme potrebno za pronalazak rješenja.

```
(assert (= (+ x1_1 x1_2 x1_3 x1_4 x1_5 x1_6 x1_7 x1_8 x1_9) S))
…… za ostale retke, stupce i kvadrate
```

Na posljetku potrebno je pokrenuti alat i zatražiti ispis dobivenih vrijednosti, ako je moguće pronaći rješenje. Ako rješenje postoji alat ispisuje "sat" popraćen dobivenim vrijednostima, a ako rješenje ne postoji alat ispisuje "*unsat*".

```
(check)
(show-model)
```




Slika 3.4 Poziv i ispis datoteke

# Zaključak

U radu je prikazan problem bojanja grafa te su predstavljeni neki od načina kako se taj problem može riješiti. Iako je problem bojanja grafa moguće riješiti svim alatima prikazanim u radu, u programskom dijelu se za to koristi SMT alat, tj. Yices pomagalo koje je primjer SMT rješavala. Problem bojanja grafa se uspoređuje s rješavanjem Sudoku tablice, popunjavanje tablice različitim brojevima poput bojanja vrhova različitim bojama, što se pokazalo kao vjerodostojna usporedba. Budući je za učinkovito rješavanje Sudoku problema, osim Booleove logike, potrebna i teorija aritmetike, kao najbolji alat za to se iskazao SMT, preciznije pomagalo Yices. U programskom dijelu rada je ostvaren rješavač Sudoku problema, koristeći Yices pomagalo. Program koji se pokreće unutar pomagala Yices se dinamički generira koristeći Perl skriptu, koja definira ograničenja na temelju ulaznih podataka Sudoku tablice. Biblioteke Yices pomagala su omogućile i pisanje programa u jeziku C, koji također rješava Sudoku tablicu, pri tome koristeći SMT. Detaljan opis programa pisanog Yices sintaksom se nalazi u poglavlju 3.5 *Sudoku i SMT*, dok se detaljan opis programa pisanog u jeziku C nalazi u privitku rada.

# Literatura


[1] Mckworth, A. *Solving Constraint Satisfaction Problems (CSPs) using Search*, (28. siječnja 2013.), str. 6-37.

[2] Lazebnik S., Veloso M. , *Constraint Satisfaction Problems*, General class of problems , Chapter 4, Massachusetts Institute of Technology, (2012.)

[3] Hahmann T., *Constraint Satisfaction Problems (Backtracking Search)*, CSC384 Introduction to Artificial Intelligence, University of Toronto, (2011.)

[4] Weber, T. *A SAT-based Sudoku Solver*, Institut für Informatik, Techniche Universität München, (2010.)

[5] Nieuwenhuis R., Oliveras A., *Introduction to SMT and Solving CSP'S with SMT*, Seminar on Constraint Programming, University of Bergen, (31. ožujka 2011.)

[6] Sébastien Bardin, Nikolaj Bjørner, and Cristian Cadar. Bringing CP, SAT and SMT together: Next Challenges in Constraint Solving (Dagstuhl Seminar 19062). In Dagstuhl Reports, Volume 9, Issue 2, pp. 27-47, Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2019)

[7] Kovačević D., Nakić A., Krnić M., Pavčević M.O., *Diskretna matematika 1*, Skripta, Sveučilište u Zagrebu Fakultet elektrotehnike i računarstva, (2020. travanj), str. 63-163.

[8] https://upload.wikimedia.org/wikipedia/commons/thumb/2/24/Tree_graph.svg/180px-Tree_graph.svg.png, posjećeno (11. lipnja 2024.)

[9] https://e.math.cornell.edu/people/mann/classes/brown/Petersen.png, posjećeno (11. lipnja 2024.)

[10] Gualandi S., Malucelli F., *Exact Solution of Graph Coloring Problem via Constraint Programming and Column Generation*, Dipartimento di Elettronica ed Informazione, Politecnio di Milano, Milano, (2010.)

[11] Schulz K. W., *Reducing Graph Coloring to SAT,* University of Texas at Austin, Austin (2011.)

[12] https://en.wikipedia.org/wiki/Sudoku, posjećeno (11. lipnja 2024.)

[13] Reeson C. G., *Using constraint processing to model, solve and support interactive solving of Sudoku puzzles*, Završni rad, The College of Arts and Science at the University of Nebraska, Lincoln, (2017. svibanj)

# Sažetak

Bojanje grafa primjenom logičke zadovoljivosti

Rad daje uvide u različite alate programiranja s ograničenjima i zadovoljivosti. Alati koji su predstavljeni uključuju CP, SAT, SMT te kao primjer SMT-a pomagalo Yices. Pomagalo Yices je ja također korišteno kod programskog dijela rada. Definiran je i glavni problem kojim se rad bavi, a to je bojanje vrhova grafa. Prikazano je i kako se taj problem rješava primjenom pojedinog alata programiranja s ograničenjima i zadovoljivosti. Igra Sudoku je korištena za ilustraciju problema bojanja grafa. Bojanje vrhova grafa različitim bojama se uspoređuje s popunjavanjem Sudoku tablice različitim brojevima. Programski dio rada u tu svrhu rješava zadanu Sudoku tablicu koristeći SMT pristup, odnosno Yices pomagalo.

**Ključne riječi**: Programiranje s ograničenjima; Zadovoljivost; Yices; Bojanje grafa; Sudoku

# Summary

Graph Colouring with SMT solver

This paper provides an insight into constraint programming and satisfiability solvers. Those solvers include CP, SAT, SMT (Satisfiability Modulo Theories) and Yices solver, as an example of a SMT solver. Yices solver is also used in the practical part of the paper. The main problem presented in the paper is graph coloring. There are examples of how that problem is solved using each of the solvers above. The Sudoku game is used to explain the main problem of graph coloring. The problem of deciding which color to use on which point is compared to deciding which number to put in which cell of the Sudoku table. The practical part of the paper for that reason solves given Sudoku table by using the SMT solver Yices.

**Keywords**: Constraint programming; Satisfiability; Yices; Graph coloring; Sudoku

# Skraćenice

| | | |
|---|---|---|
| CP | *Constraint Programming* | programiranje s ograničenjima |
| BT | *Backtracking* | povratno pretraživanje |
| MVC | *Most Remaining Values* | tehnika odabira varijabli kod CP |
| LVC | *Least Constraining Value* | tehnika odabira vrijednosti kod CP |
| SAT | *Satisfability* | zadovoljivost |
| DPLL | *Davis-Putnam-Logemann-Loveland* | algoritam za rješavanje SAT |
| CDCL | *Conflict-Driven Clause Learning* | algoritam za rješavanje SAT |
| SMT | *Satisfiability Modulo Theories* | |
| SMT-LIB | *Satisfiability Modulo Theories Library* | standardizacija SMT-a |

# Privitak

### Programski kod

Kod programa pisanog u C-u koji rješava Sudoku

Biblioteke potrebne za prevođenje i izvršavanje programa te definicija datoteke iz koje se čitaju ulazni podatci:

```
#include<stdio.h>
#include<stdbool.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include "../PROJEKT_R/yices-2.6.1/include/yices.h"
#define MAX_LINE_LENGTH 1024
#define F "in.txt"
```

Metoda za ispis rezultata:

```
static void print_term(term_t term) {
    char *s;

    s = yices_term_to_string(term, 1024, 128, 1);
    if (s == NULL) {
        // An error occurred
        s = yices_error_string();
        fprintf(stderr, "Error in print_term: %s\n", s);
        yices_free_string(s);
        exit(1);
    }
    printf("%s\n", s);
    yices_free_string(s);
}
```

Metoda koja pronalazi rješenje Sudoku tablice:

```
static void sud(int N){
    double S = ((N + 1.0) / 2.0) * N;   //define row/column Sum

    context_t *ctx = yices_new_context(NULL);   //create context
    type_t int_type = yices_int_type();
```

Inicijalizacija varijabli i imenovanje oblika: *xi_j*

```
    term_t x[N][N];     //define variables
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            x[i][j] = yices_new_uninterpreted_term(int_type);
            char name[5];
            snprintf(name, sizeof(name), "x%d_%d", i, j);
            yices_set_term_name(x[i][j], name);
        }
    }
```

Definicija početnih varijabli, koje se čitaju iz datoteke definirane na početku programa u varijablu "*F*" :

```
    //Constraints: defined variables
    term_t formula = yices_arith_eq_atom(yices_int32(1),
yices_int32(1));

    FILE *file = fopen(F, "r");
    if(file == NULL){
        fprintf(stderr, "Error opening file!");
        return;
    }
    char line [MAX_LINE_LENGTH];
    int j = 0;

    while(fgets(line, sizeof(line), file)){
```

```
            line[2 * N - 1] = ' ';
            //printf("%s\n", line);
            int k = 0;
            for(int i = 0; i < N * 2 - 1; i++){
                if(line[i] == ' '){
                    continue;
                }
                if(line[i] != '0'){
                    char res[N];
                    sprintf(res, "x%d_%d", j, k);
                    int a = line[i] - '0';
                    formula = yices_and2(formula,
yices_arith_eq_atom(yices_get_term_by_name(res), yices_int32(a)));
                    yices_assert_formula(ctx,
yices_arith_eq_atom(yices_get_term_by_name(res), yices_int32(a)));
                }
                k++;
            }
            j++;
        }
```

Postavljanje ograničenja na varijable: $1 \leq xi\_j \leq N$:

```
        //set range for xij >= 1 && <= N
        for(int i = 0; i < N; i++){
            for(int j = 0; j < N; j++){
                formula = yices_and3(formula,
yices_arith_gt0_atom(x[i][j]), yices_arith_leq_atom(x[i][j],
yices_int32(N)));
                yices_assert_formula(ctx,
yices_arith_gt0_atom(x[i][j]));
                yices_assert_formula(ctx, yices_arith_leq_atom(N,
x[i][j]));
            }
```

```
        }
```

Definiranje ograničenja na sumu retka:

```
        //row sum == S
        for(int i = 0; i < N; i++){
            term_t row[N];
            for(int j = 0; j < N; j++){
                row[j] = x[i][j];
            }
            formula = yices_and2(formula,
    yices_arith_eq_atom(yices_sum(N, row), yices_int32(S)));
            yices_assert_formula(ctx, yices_arith_eq_atom(yices_sum(N,
    row),yices_int32(S)));
        }
```

Definiranje ograničenja na sumu stupca:

```
        //column sum == S
        for(int i = 0; i < N; i++){
            term_t column[N];
            for(int j = 0; j < N; j++){
                column[j] = x[j][i];
            }
            formula = yices_and2(formula,
    yices_arith_eq_atom(yices_sum(N, column), yices_int32(S)));
            yices_assert_formula(ctx,  yices_arith_eq_atom(yices_sum(N,
    column), yices_int32(S)));
        }
```

Postavljanje ograničenja DISTINCT (različite vrijednosti) na retke:

```
        //distinct values in rows
        for(int i = 0; i < N; i++){
            term_t row[N];
            for(int j = 0; j < N; j++){
                row[j] = x[i][j];
```

```
            }
            formula = yices_and2(formula, yices_distinct(N, row));
            yices_assert_formula(ctx, yices_distinct(N, row));
        }
```

Postavljanje ograničenja DISTINCT (različite vrijednosti) na stupce:

```
        //distinct values in columns
        for(int i = 0; i < N; i++){
            term_t column[N];
            for(int j = 0; j < N; j++){
                column[j] = x[j][i];
            }
            formula = yices_and2(formula, yices_distinct(N, column));
            yices_assert_formula(ctx, yices_distinct(N, column));
        }
```

Postavljanje ograničenja DISTINCT (različite vrijednosti) na kvadrate:

```
        //distinct values in squares
        for(int a = 0; a < sqrt(N); a++){       //selects column section
            for(int i = 0; i < sqrt(N); i++){   //selects row section
                term_t square[N];
                int index = 0;

                for(int j =  i * sqrt(N); j < i * sqrt(N) + sqrt(N);
    j++){
                    for(int k = a * sqrt(N); k < a * sqrt(N) + sqrt(N);
    k++){
                        square[index++] = x[(int) (j)][(int) (k)];
                    }
                }
                formula = yices_and2(formula, yices_distinct(N,
    square));
                yices_assert_formula(ctx, yices_distinct(N, square));
```

```
            }
        }
```

Provjera valjanosti formule:

```
        term_t code = yices_assert_formula(ctx, formula);
        if (code < 0) {
            fprintf(stderr, "Assert failed: code = %d, error = %d\n",
                    code, yices_error_code());
            yices_print_error(stderr);
        }
```

Provjera statusa formule te ispis statusa valjanosti formule i rješenja, ako postoji:

```
        switch (yices_check_context(ctx, NULL)){
        case STATUS_SAT:
            printf("The formula is satisfiable\n");

            model_t* model = yices_get_model(ctx, true);
            if (model == NULL) {
                fprintf(stderr, "Error in get_model\n");
                yices_print_error(stderr);
            } else {
                printf("Model:\n");
                code = yices_pp_model(stdout, model, 80, N * N, 0);
                /*
                int32_t v;
                for(int i = 0; i < N; i++){
                    for(int j = 0; j < N; j++){
                        code = yices_get_int32_value(model, x[i][j],
&v);
                        if(code < 0){
                            printf("Error in get_int32_value for
'x%d%d'\n", i+1, j+1);
                            yices_print_error(stderr);
```

```
                    }else{
                        printf("|%d", v);
                    }
                }
                printf("|\n");
                printf("---------\n");   //N*3-
            }
            */
            yices_free_model(model);
        }
        break;
    case STATUS_UNSAT:
        printf("The formula is not satisfiable\n");
        model_t* m = yices_get_model(ctx, true);
        if (m == NULL) {
            fprintf(stderr, "Error in get_model\n");
            yices_print_error(stderr);
        } else {
            printf("Model:\n");
            //code = yices_pp_model(stdout, model, 80, N * N, 0);
            int32_t v;
            for(int i = 0; i < N; i++){
                for(int j = 0; j < N; j++){
                    code = yices_get_int32_value(m, x[i][j], &v);
                    if(code < 0){
                        printf("Error in get_int32_value for
'x%d%d'\n", i+1, j+1);
                        yices_print_error(stderr);
                    }else{
                        printf("|%d", v);
                    }
```

```
                }
                printf("|\n");
                printf("---------\n");   //N*3-
            }
            yices_free_model(m);
        }
        break;
    case STATUS_UNKNOWN:
        printf("The status is unknown\n");
        break;
    case STATUS_IDLE:
    case STATUS_SEARCHING:
    case STATUS_INTERRUPTED:
    case STATUS_ERROR:
        fprintf(stderr, "Error in check_context\n");
        yices_print_error(stderr);
        break;
    }
    yices_free_context(ctx);
}
```

Funkcija main:

```
int main(int argc, char *argv[]){
    yices_init();
    int n = atoi(argv[1]);
    sud(n);
    yices_exit();
    return 0;
}
```