Primene ne-KNF SAT rešavača

— Master teza —

Milan Todorović

10. oktobar 2011

mentor:
dr Predrag Janičić

članovi komisije:
dr Filip Marić
mr Mladen Nikolić
dr Predrag Janičić

Matematički fakultet
Univerzitet u Beogradu
2011

Sadržaj

1 Uvod **5**

2 Iskazna logika, SAT problem i KNF rešavači **7**
2.1 Sintaksa iskazne logike 7
2.2 Semantika iskazne logike 8
2.3 Cajtinova transformacija 11
2.4 SAT problem 12
2.5 KNF SAT rešavači 13
2.6 Primene SAT rešavača 15
2.7 Sistem URSA 17

3 Ne-KNF rešavači **19**
3.1 NoClause rešavač 19
3.2 NflSAT rešavač 24

4 Ugradnja ne-KNF SAT rešavača u sistem URSA **29**
4.1 Klasa NonCnfSolvers 29
4.2 NoClause i sistem URSA 30
4.3 NflSAT i sistem URSA 31

5 Eksperimentalni rezultati **35**
5.1 3-SAT problem 35
5.2 Problem n dama 38
5.3 Golombov lenjir 40

6 Zaključci i dalji rad **43**

Glava 1

Uvod

SAT problem je problem ispitivanja zadovoljivosti iskaznih formula. Veliki broj problema, kao što su to verifikacija hardvera i softvera, kombinatorni problemi, problemi planiranja i mnogi drugi, se mogu rešiti prevođenjem na SAT problem, koji se dalje rešava korišćenjem SAT rešavača. Suštinski, SAT rešavači predstavljaju opštu platformu za kombinatorno rezonovanje i pretragu koja kao jezik koristi iskaznu logiku. Bez obzira što prevođenje na SAT dovodi do primetnog povećanja veličine reprezentacije problema, često je prevođenje na SAT i rešavanje modernim SAT rešavačem efikasnije nego korišćenje specijalizovanog rešavača nad početnom reprezentacijom problema. Najveći broj modernih SAT rešavača se zasniva na *Davis-Putnam-Logemann-Loveland* (DPLL) proceduri, koja zahteva da logička formula bude u konjuktivnoj normalnoj formi (KNF). Međutim, značajan broj problema koji potiče iz prakse se prirodnije predstavlja formulom koja nije u KNF. Zbog toga, da bi se upotrebili SAT rešavači, ovi problemi se moraju pretvoriti u KNF. Jedan broj ključnih tehnika za povećanje efikasnosti DPLL SAT rešavača je zasnovan na iskorišćavanju jednostavne strukture KNF. Da bi veličina formule u KNF bila linearna u odnosu na prvobitnu formulu, koristi se *Cajtinovo* (engl. *Tseitin*) kodiranje koje uvodi nove promenljive. Međutim, pored povećanja broja promenljvih, prevođenje u KNF dovodi i do gubitka informacija o strukturi problema, koje bi se mogle iskoristiti za unapređenje efikasnosti rešavanja problema. Neka je, na primer, potrebno ispitati zadovoljivost formule $(a \Leftrightarrow b) \wedge \neg(a \Leftrightarrow c) \wedge (a \Leftrightarrow c)$. Na osnovu njenog početnog oblika se vidi da ona nije zadovoljiva jer promenljive a, b i c moraju imati iste vrednosti (što se vidi na osnovu $a \Leftrightarrow b$ i $a \Leftrightarrow c$) dok promenljive a i c moraju imati različite vrednosti (na osnovu $\neg(a \Leftrightarrow c)$) što je nemoguće ispuniti. Sa druge strane, prevođenjem formule u KNF dobija se formula $(\neg a \vee b) \wedge (b \vee \neg a) \wedge (\neg b \vee \neg c) \wedge (b \vee c) \wedge (\neg a \vee c) \wedge (a \vee \neg c)$. Na osnovu dobijene formule se ne može jednostavno zaključiti da je ona nezadovoljiva, jer je izgubljena struktura početne formule kod koje je to moguće. Ovakvi primeri su doprineli početku razvoja ne-KNF rešavača koji ne rade isključivo sa formulama u KNF, već direktno mogu da rade sa proizvoljnim iskaznim formulama. Ne-KNF SAT rešavači najčešće koriste kompaktne grafovske strukture kako bi

predstavili formulu. Ove strukture su često takve da se svako pojavljivanje neke podformule predstavlja samo jednim čvorom u grafu. Još jedna prednost ovih struktura je to što se u njima jednostavno predstavljaju n-arni logički veznici, pa ih ne-KNF SAT rešavači najčešće podržavaju. Ne-KNF rešavači najčešće koriste različite tehnike koje se koriste u KNF SAT rešavačima (na primer šema nadgledanih literala). Naravno, ove tehnike su prilagođene primeni na ne-KNF formulama.

Glava 2

Iskazna logika, SAT problem i KNF rešavači

U iskaznoj logici, svaka promenljiva predstavlja neki iskaz, koji se koristeći logičke veznike mogu kombinovati u složenije iskaze. Iskazna logika ima tri aspekta: sintaksu, semantiku i deduktivne sisteme. Glavni problem iskazne logike je ispitivanje da li je data iskazna formula tautologija, odnosno valjana, i da li je zadovoljiva. U nastavku će ukratko biti opisana sintaksa i semantika iskazne logike [7], kao i primene sistema za proveru zadovoljivosti u iskaznoj logici. U daljem tekstu neće biti razmatrani deduktivni sistemi za iskaznu logiku jer se SAT problem definiše u terminima semantike.

2.1 Sintaksa iskazne logike

U nastavku će iskazne formule biti definisane kao niske karaktera, odnosno biće korišćena konkretna sintaksa.

Definicija 1. *Azbuka iskazne logike* Σ *se sastoji od sledećih skupova:*

1. *prebrojivi skup iskaznih slova*
2. *skup logičkih veznika* $\{\neg, \wedge, \vee, \Rightarrow, \oplus, \Leftrightarrow\}$
3. *skup logičkih konstanti* $\{\bot, \top\}$
4. *skup pomoćnih simbola* $\{(,)\}$

Jezik iskazne logike, nad skupom iskaznih slova, je najmanji podskup svih reči azbuke Σ *tako da važi:*

- *logičke konstante i iskazna slova su izkazne formule*
- *ako su* A *i* B *iskazne formule, onda su to i* $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, $(A \oplus B)$, $(A \Rightarrow B)$ *i* $(A \Leftrightarrow B)$

Na primer, $((p \Rightarrow q) \vee (\neg r \Leftrightarrow (p \wedge \neg q)))$ je iskazna formula.

Logički veznici se nazivaju i bulovskim veznicima ili samo veznicima. Veznik $\neg$ se naziva *negacija*, $\wedge$ *konjunkcija*, $\vee$ *disjunkcija*, $\Rightarrow$ *implikacija* i $\Leftrightarrow$ *ekvivalencija*. Veznik $\neg$ je unarni veznik, dok su svu ostali binarni.

Iskazna slova se još nazivaju i *iskazne promenljive* ili *iskazne varijable*. Zajedno sa logičkim konstantama $\bot$ i $\top$, iskazna slova se nazivaju i *atomičke iskazne formule*. *Literal* je ili atomička iskazna formula, ili negacija atomičke iskazne formule.

Kako bi se pri tumačenju iskaznih formula izbegla višeznačnost, koriste se zagrade kao pomoćni simboli. Međutim, kako je njihov broj veliki, one se obično izostavljaju usvajajući konvenciju kojom se izbegava višeznačnost. Ova konvencija je zasnovana na prioritetu logičkih veznika, i to na sledeći način (veznici su poređani od onog sa najvećim prioritetom do onog sa najmanjim): $\neg \wedge \vee \oplus \Rightarrow \Leftrightarrow$.

Definicija 2. *Za svaku iskaznu formulu A postoji njen skup podformula. To je najmanji skup koji zadovoljava sledeće uslove:*

- *svaka iskazna formula je podformula sama sebi*
- *ako je formula A jednaka $\neg B$, onda je svaka podformula formule B ujedno i podformula formule A. Ako je formula A jednaka $B \wedge C$, $B \vee C$, $B \Rightarrow C$, $B \oplus C$ ili $B \Leftrightarrow C$, onda je svaka podformula formule B i svaka podformula formule C ujedno i podformula formule A.*

Na primer, skup podformula formule $p \wedge q \Rightarrow r$ jednak je $\{p, q, r, p \wedge q, p \wedge q \Rightarrow r\}$.

2.2 Semantika iskazne logike

Semantika iskazne logike govori o značenju formula. Preslikavanje v iz skupa promenljivih u bilo koji dvočlani skup, na primer $\{0, 1\}$, se naziva *valuacija*.

Definicija 3. *Svaka valuacija v određuje funkciju I_v koja se naziva interpretacija za valuaciju v. Interpretacija za datu valuaciju v slika skup iskaznih formula na skup $\{0, 1\}$ i to na sledeći način primitivnom rekurzijom:*

- $I_v(p) = v(p)$*, za svako iskazno slovo p iz skupa iskaznih slova*
- $I_v(\bot) = 0$*;*
- $I_v(\top) = 1$*;*
- $I_v(\neg A) = 1$ *ako je* $I_v(A) = 0$ *i* $I_v(\neg A) = 0$ *ako je* $I_v(A) = 1$
- $I_v(A \wedge B) = 1$ *ako je* $I_v(A) = 1$ *i* $I_v(B) = 1$*, inače je* $I_v(A \wedge B) = 0$
- $I_v(A \vee B) = 0$ *ako je* $I_v(A) = 0$ *i* $I_v(B) = 0$*, inače je* $I_v(A \vee B) = 1$

- $I_v(A \Rightarrow B) = 0$ *ako je* $I_v(A) = 1$ *i* $I_v(B) = 0$*, inače je* $I_v(A \Rightarrow B) = 1$
- $I_v(A \oplus B) = 1$ *ako je* $I_v(A) \neq I_v(B)$*, inače je* $I_v(A \oplus B) = 0$
- $I_v(A \Leftrightarrow B) = 1$ *ako je* $I_v(A) = I_v(B)$*, inače je* $I_v(A \Leftrightarrow B) = 0$

$I_v(A)$ se naziva vrednost iskazne formule A u interpretaciji I_v. Ako za neku valuaciju v važi da je $I_v(A) = 1$, onda je formula A *tačna* u interpretaciji I_v, odnosno u valuaciji v. U suprotnom se kaže da je *netačna*.

Definicija 4. *Za iskaznu formulu se kaže da je* zadovoljiva *ako postoji neka valuacija za koju je iskazna formula tačna. Formula je* valjana *(*tautologija*) ako je ona tačna za bilo koju valuaciju. S druge strane, ako za formulu ne postoji valuacija za koju je ona tačna, onda je formla* nezadovoljiva *(*kontradikcija*). Iskazna formula je* poreciva *ako postoji valuacija za koju je ona netačna.*

Definicija 5. *Za skup iskaznih formula kaže se da je zadovoljiv ako postoji valuacija za koju je svaka formula iz skupa tačna. Takva valuacija je* model *za navedeni skup formula. Skup iskaznih formula je nezadovoljiv, ako ne postoji valuacija koja zadovoljava svaku formulu iz tog skupa.*

Definicija 6. *Za datu iskaznu formulu A se kaže da je* logička posledica *skupa iskaznih formula Γ ako je svaki model za skup Γ ujedno i model formule A.*

Lako se pokazuju tvrđenja da je formula valjana ako i samo ako je logička posledica praznog skupa formula i da ako je skup formula kontradiktoran onda je svaka formula njegova logička posledica.

Definicija 7. *Dve formule A i B su* logički ekvivalentne*, u oznaci $A \equiv B$ ako je svaki model jedne formule ujedno i model druge, i obratno.*

Definicija 8. *Dve formule A i B su* ekvizadovoljive *ako važi da je formula A zadovoljiva ako i samo ako je formula B zadovoljiva.*

Definicija 9. Zamena *(*supstitucija*) svih pojavljivanja iskazne formule C u iskaznoj formuli A formulom D, u oznaci $A[C \mapsto D]$, se definiše rekurzivno na sledeći način:*

- *ako je* $A = C$ *onda je* $A[C \mapsto D] = D$*, inače*
- *ako je* $A = \top$ *onda je* $A[C \mapsto D] = \top$*, inače*
- *ako je* $A = \bot$ *onda je* $A[C \mapsto D] = \bot$*, inače*
- *ako je* $A = p$*, gde je p iskazno slovo, onda je* $A[C \mapsto D] = p$*, inače*
- *ako je* $A = \neg A_1$ *onda je* $A[C \mapsto D] = \neg(A_1[C \mapsto D])$*, inače*
- *ako je* $A = A_1 \wedge A_2$ *onda je* $A[C \mapsto D] = (A_1[C \mapsto D]) \wedge (A_2[C \mapsto D])$*, inače*

- *ako je* $A = A_1 \vee A_2$ *onda je* $A[C \mapsto D] = (A_1[C \mapsto D]) \vee (A_2[C \mapsto D])$, *inače*
- *ako je* $A = A_1 \oplus A_2$ *onda je* $A[C \mapsto D] = (A_1[C \mapsto D]) \oplus (A_2[C \mapsto D])$, *inače*
- *ako je* $A = A_1 \Rightarrow A_2$ *onda je* $A[C \mapsto D] = (A_1[C \mapsto D]) \Rightarrow (A_2[C \mapsto D])$, *inače*
- *ako je* $A = A_1 \Leftrightarrow A_2$ *onda je* $A[C \mapsto D] = (A_1[C \mapsto D]) \Leftrightarrow (A_2[C \mapsto D])$.

Definicija 10. Istinitosna funkcija *nad n argumenata je funkcija koja slika skup* $\{0,1\}^n$ *u* $\{0,1\}$.

Svaka iskazna formula koja ima n iskaznih promenljivih generiše neku istinitosnu funkciju nad n argumenata, na osnovu interpretacije I_v za neku valuaciju v. Logički ekvivalentne iskazne formule, koje imaju isti broj iskaznih promenljivih, generišu identične iskazne funkcije. Koristeći logičke ekvivalencije $A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$ i $A \Rightarrow B \equiv \neg A \vee B$ u svakoj iskaznoj formuli se može eliminisati pojavljivanje veznika $\Leftrightarrow$ i $\Rightarrow$, pa se time pokazuje se da se svaka istinitosna funkcija može generisati nekom iskaznom formulom koja sadrži samo veznike $\neg$, $\vee$ i $\wedge$. Štaviše, koristeći ekvivalenciju $A \wedge B \equiv \neg(\neg A \vee \neg B)$ u iskaznoj formuli se može eliminisati i pojavljivanje veznika $\wedge$, pa se dobija logički ekvivalentna formula koja sadrži samo veznike $\{\vee, \neg\}$. Ovaj skup veznika je *potpun*, jer je svaka iskazna formula logički ekvivalentna nekoj iskaznoj formuli samo nad ova dva veznika. Kako je ovaj skup veznika potpun sledi da je svaka istinitosna funkcija generisana nekom iskaznom formulom koja sadrži samo ova dva veznika. Jednostavno se pokazuje da je skup $\{\wedge, \neg\}$ takođe potpun.

Definicija 11. *Za iskaznu formulu se kaže da je u* negacijskoj normalnoj formi *(skraćeno NNF) ako sadrži samo veznike* $\wedge$, $\vee$ *i* $\neg$, *gde se veznik* $\neg$ *mora nalaziti samo uz promenljivu.*

Definicija 12. *Iskazna formula je u* konjunktivnoj normalnoj formi *(skraćeno KNF) ako je oblika* $A_1 \wedge ... \wedge A_n$ *pri čemu je svaka od formula* A_i,$(1 \leq i \leq n)$ *disjunkcija literala, odnosno* klauza.

Definicija 13. *Iskazna formula je u* disjunktivnoj normalnoj formi *(skraćeno DNF) ako je oblika* $A_1 \vee ... \vee A_n$ *pri čemu je svaka od formula* A_i, $(1 \leq i \leq n)$ *konjunkcija literala.*

Ako je neka iskazna formula A logički ekvivalentna formuli B koja je u konjuktivnoj, disjunktivnoj ili negacijskoj normalnoj formi, onda se kaže da je formula B konjuktivna, disjunktivna ili normalna forma formule A. Za svaku iskaznu formulu postoji njena konjuktivna, disjunktivna i negacijska normalna forma, što je posledica toga da je sistem veznika $\{\wedge, \vee, \neg\}$ potpun. Korišćenjem određenih logičkih ekvivalencija, svaka iskazna formula, koja u sebi ne sadrži $\top$ i $\bot$, može biti transformisana u svoju konjuktivnu, disjunktivnu ili normalnu formu, bez uvođenja novih promenljivih. U slučaju da formula sadrži $\top$ ili $\bot$

ona se ne može prevesti u KNF i DNF bez uvođenja novih promenljivih. Jedna formula može imati više različitih normalnih formi istog tipa a takođe, jedna formula koja je u normalnoj formi može biti normalna forma za više iskaznih formula.

Algoritam za dobijanje konjuktivne normalne forme neke formule formule F, kod koga koraci 1-4 ujedno vrše i prevođenje formule u NNF, izgleda ovako:

1. Svaki veznik $\Leftrightarrow$ ukloniti pomoću logičke ekvivalencije $A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$.

2. Svaki veznik $\Rightarrow$ ukloniti pomoću logičke ekvivalencije $A \Rightarrow B \equiv \neg A \vee B$.

3. Dokle god je moguće, primenjivati logičke ekvivalencije $\neg(A \wedge B) \equiv \neg A \vee \neg B$ i $\neg(A \vee B) \equiv \neg A \wedge \neg B$ (De Morganova pravila).

4. Koristeći logičku ekvivalenciju $\neg\neg A \equiv A$ ukloniti višestruka pojavljivanja veznika $\neg$.

5. Dokle god je moguće, primenjivati logičke ekvivalencije $(A \vee (B \wedge C)) \equiv ((A \vee B) \wedge (A \vee C))$ i $((B \wedge C) \vee A) \equiv ((B \vee A) \wedge (C \vee A))$.

Disjunktivna normalna forma neke formule se dobija na sličan načina kao i konjunktivna. Jedina razlika u algoritmu je to što se u poslednjem koraku umesto navedenih logičkih ekvivalencija koriste $(A \wedge (B \vee C)) \equiv ((A \wedge B) \vee (A \wedge C))$ i $((B \vee C) \wedge A) \equiv ((B \wedge C) \vee (C \wedge A))$.

2.3 Cajtinova transformacija

KNF SAT rešavači zahtevaju da ulazna formula bude u obliku konjuktivne normalne forme. Kako prikazani algoritam transformacije ima potencijalno eksponencijalnu složenost, koristi se Cajtinovo kodiranje [16] za konstruisanje KNF date formule. Formula dobijena na ovaj način nije logički ekvivalentna, ali je ekvizadovoljiva polaznoj što je sasvim dovoljno za ispitivanje zadovoljivosti polazne formule. Cajtinovo kodiranje radi na principu dodavanja novih promenljivih koje odgovaraju podformulama, kao i klauze koje obezbeđuju vezu između podformula i promenljivih koje ih predstavljaju.

Cajtinova transformacija formule F koja je u NNF, se može opisati na sledeći način. Formula F se može predstaviti kao binarno stablo u kome unutrašnji čvorovi predstavljaju veznike $\wedge$ ili $\vee$, dok listovi stabla predstavljaju literale. Neka P označava novu promenljivu koja se trenutno uvodi i neka P_l i P_d označavaju promenljive koje predstavljaju levo i desno podstablo trenutnog čvora. Cajtinova transformacija se može izvesti postfiksnim obilaskom binarnog stabla formule i kreiranjem izlaznog skupa klauza, to na sledeći način:

- Ako je trenutni čvor list, koji predstavlja literal, ne radi se ništa.

- Ako je trenuti čvor unutrašnji čvor tipa $\wedge$, prvo se rekurzivno obrade njegova deca. Potom se pravi iskazna formula $P \Leftrightarrow P_l \wedge P_d$, čijim se prevođenjem u KNF dobijaju sledeće klauze koje se dodaju skupu klauza: $(\neg P \vee P_l)$,$(\neg P \vee P_d)$ i $(P \vee \neg P_l \vee \neg P_d)$.

- Ako je trenutni čvor unutrašnji čvor tipa $\vee$, prvo se rekurzivno obrade njegova deca. Potom se pravi iskazna formula $P \Leftrightarrow P_l \vee P_d$, čijim se prevođenjem u KNF dobijaju sledeće klauze koje se dodaju skupu klauza: $(P \vee \neg P_l)$,$(P \vee \neg P_d)$ i $(\neg P \vee P_l \vee P_d)$.

Na kraju se skupu klauza dodaje jedinična klauza koja se sastoji od promenljive koja predstavlja koren stabla, odnosno celu formulu.

Teorema 1. *Formula u KNF koja je ekvizadovoljiva polaznoj formuli jeste konjunkcija svih klauza iz generisanog skupa klauza.*

Transformacija formule F u NNF, sa početka algoritma, se može preskočiti. U tom slučaju bi se i ostali veznici mogli direktno obrađivati, čime bi se uvelo manje pomoćnih promenljivih.

Primer 1. *Neka je data formula* $(A \Rightarrow (C \wedge D)) \vee (B \Rightarrow (C \wedge E))$. *Prvo se uvodi nova promenljiva* F_1 *koja će da predstavlja podformulu* $C \wedge D$, *kao i nove klauze* $(\neg F_1 \vee C)$, $(\neg F_1 \vee D)$ *i* $(\neg C \vee \neg D \vee F_1)$. *Ove klauze služe da se obezbedi da se nova promenljiva* F_1 *ponaša isto kao podformula* $C \wedge D$. *Zatim se analogno uvodi promenljiva* F_2 *koja predstavlja podformulu* $C \wedge E$, *zajedno sa klauzama* $(\neg F_2 \vee C)$, $(\neg F_2 \vee E)$ *i* $(\neg C \vee \neg E \vee F_2)$. *Tako smo dobili formulu oblika* $(A \Rightarrow F_1) \vee (B \Rightarrow F_2)$. *Nakon toga se primenjuju logičke ekvivalencije* $A \Rightarrow F_1 \equiv \neg A \vee F_1$ *i* $A \Rightarrow F_2 \equiv \neg A \vee F_2$. *Potom se uvode još dve nove promenljive* F_3 *i* F_4, *zajedno sa klauzama* $(\neg F_3 \vee \neg A \vee F_1)$, $(A \vee F_3)$, $(\neg F_1 \vee F_3)$, $(\neg F_4 \vee \neg B \vee F_2)$, $(B \vee F_4)$ *i* $(\neg F_2 \vee F_4)$, *kako bi redom zamenile podformule* $(\neg A \vee F_1)$ *i* $(\neg B \vee F_2)$. *Na kraju se uvodi još jedna promenljiva* F_5 *koja predstavlja podformulu* $F_3 \vee F_4$, *zajedno sa klauzama* $(\neg F_5 \vee F_3 \vee F_4)$, $(\neg F_3 \vee F_5)$ *i* $(\neg F_4 \vee F_5)$. *Konačni rezultat je sledeća formula koja je u KNF:* $(\neg F_1 \vee C) \wedge (\neg F_1 \vee D) \wedge (\neg C \vee \neg D \vee F_1) \wedge (\neg F_2 \vee C) \wedge (\neg F_2 \vee E) \wedge (\neg C \vee \neg E \vee F_2) \wedge (\neg F_3 \vee \neg A \vee F_1) \wedge (A \vee F_3) \wedge (\neg F_1 \vee F_3) \wedge (\neg F_4 \vee \neg B \vee F_2) \wedge (B \vee F_4) \wedge (\neg F_2 \vee F_4) \wedge (\neg F_5 \vee F_3 \vee F_4) \wedge (\neg F_3 \vee F_5) \wedge (\neg F_4 \vee F_5)$.

2.4 SAT problem

Problem zadovoljivosti logičke formule, skraćeno SAT od engleskog *satisfiability*, je problem ispitivanja da li je formula zadovoljiva, tj. da li se promenljivima date formule mogu dodeliti vrednosti (tj. odrediti valuacija) tako da formula bude tačna. Ovaj problem je veoma težak i nije poznat nijedan algoritam koji bi ga efikasno rešio (u polinomijalnom vremenu). Štaviše, veruje se da takav algoritam ne postoji. SAT problem je prvi problem za koji je dokazano da je NP-kompletan, što su nezavisno jedan od drugog pokazali Kuk [2] i Levin[9].

2.5 KNF SAT rešavači

Postoje dve osnovne vrste SAT rešavača [14], kompletni i stohastički. Kompletni rešavači mogu da utvrde za bilo koju formulu da li je zadovoljiva ili ne, dok stohastički mogu da utvrde samo za zadovoljivu, često brže nego što bi to utvrdili kompletni rešavači, ali ne mogu da utvrde da je nezadovoljiva.

Jedan od načina rada nekih stohastičkih rešavača je gramziva lokalna pretraga. Uzima se slučajna valuacija i razmatraju se sve valuacije koje se od izabrane razlikuju samo u vrednosti od najviše jedne promenljive. Od razmatranih valuacija bira se ona koja zadovoljava najveći broj klauza, a postupak se ponavlja dok se ne nađe valuacija koja zadovoljava celu početnu formulu.

Većina kompletnih SAT rešavača se zasniva na DPLL proceduri. Ona kao ulaz očekuje formulu u KNF obliku, a kao izlaz vraća DA pod uslovom da je formula zadovoljiva a inače, vraća NE. Moderni SAT rešavači zasnovani na DPLL proceduri nazivaju se i CDCL rešavači (od *conflict-driven clause-learning*). Kako su konjunkcija i disjunkcija komutativne i asocijativne, poredak klauza C_i u ulaznoj formuli D je nebitan pa se ona može razmatrati kao skup klauza, od kojih se svaka može smatrati skupom literala. U proceduri se podrazumeva da je prazan skup klauza (*prazna formula*) zadovoljiv, a da je klauza koja ne sadrži nijedan literal (*prazna klauza*) nezadovoljiva, pa je samim tim i formula koja sadrži praznu klauzu nezadovoljiva.

Radi povećanja efikasnosti DPLL procedura, u SAT rešavačima, pretrpi složene izmene. Tako je jedna od bitnijih izmena to da se ne vrši transformacija nad formulom, već rešavač u koracima pravi valuaciju u kojoj bi formula trebala da bude tačna. U nekom trenutku DPLL procedure, nekoj promenljivoj se mora

```
funciton  DPLL (F :formula, v : valuacija):  (DA,NE)
begin
  if ako je F netačna u valuaciji v then return NE
  else if v je totalna then return DA
  else if postoji jedinična klauza (l ∨ l_1 ∨ ... ∨ l_k u F t.d.
     l, l̄ ∉ v, l̄_1, ..., l̄_k ∈ v)
     then return DPLL(F, v ∪ {l})
  else begin
     izaberi literal l t.d. l ∈ F, l, l̄ ∉ v
     if DPLL(v ∪ {l} )=DA then return DA
     else return DPLL(F, v ∪ {l̄})
  end
end
```

Slika 2.1: DPLL procedura koja kreira i menlja parcijalne valuacije.

dodeliti vrednost 0 ili 1. Odabir konkretne vrednosti se naziva *odluka*. Prilikom odluke, pored biranja promenljive čija će vrednost biti zadata, bira se i početni *polaritet*, odnosno određuje se koju će vrednost promenljiva prvu da dobije pri odluci.

Tokom rešavanja, može se doći do toga da neka promenljiva mora da ima određenu vrednost (pravilo *unit propagation* DPLL procedure). Tada je reč o *implikaciji*. Ako se desi da je formula netačna u tekućoj valuaciji, reč je o *konfliktu*. Ako se ovo desi, znači da je u nekom trenutku doneta pogrešna odluka, pa se ona mora promeniti kako bi se našla zadovoljavajuća valuacija. Promena odluke zbog koje je došlo do konflikta se vrši *skokovima unazad*, tako što se rešavač vraća direktno na neki od prethodnih nivoa odluke, pod uslovom da važi da su samo odluke na tom i ranijim nivoima dovele do konflikta. Razlog do koga je došlo do konflikta se može predstaviti pomoću *naučenih klauza*, koje se dodaju početnom skupu klauza. Tokom rešavanja, broj naučenih klauza obično raste, a to dovodi do opterećenja rešavača, pa se ove klauze u nekom trenutku brišu. Postupak brisanja naučenih klauza naziva se *zaboravljanje*.

Rešavač može veliku količinu vremena da potroši pretraživajući deo prostora pretrage u kome se ne nalazi rešenje. Kako bi se ovo izbeglo, proces rešavanja s vremena na vreme počinje iz početka, naravno zadržavajući trenutno naučene klauze. Ovo se naziva *otpočinjanje iznova*. Za slučaj da je potrebno naći sve modele neke formule, mogu se koristiti *blokirajuće klauze* (eng. *blocking clause*). Kada se pronađe neki model formule, na osnovu njega se pravi nova klauza koja se povezuje konjunkcijom sa početnom formulom. Pravljenje klauze je jednostavno, iskazne promenljive čija je vrednost u modelu 1 u klauzu ulaze negirane, dok ostale ulaze bez negacije. Ta klauza sprečava da se pri ponovnom ispitivanju zadovoljivosti formule kao rešenje dobije već pronađeni model.

Primer 2. *Neka se ispituje zadovoljivost formule* $p \Rightarrow q$. *Jedna valuacija koja bi zadovoljavala ovu formulu bi bila* $v(p) = 0$ *i* $v(q) = 1$. *Na osnovu nje se dobija klauza* $(p \vee \neg q)$, *koja se povezuje sa početnom formulom konjunkcijom, pa se dobija formula* $(p \Rightarrow q) \wedge (p \vee \neg q)$. *Ovako se daljim ispitivanjem neće dobiti valuacija gde je* $v(p) = 0$ *i* $v(q) = 1$.

Šema nadgledanih literala je prvi put uvedena u KNF rešavaču zChaff[13], i od tada je postala standardna metoda većine KNF SAT rešavača. Njen cilj je poboljšanje efikasnosti propagiranja bulovskih ograničenja (eng. *Boolean constraint propagation*). Da bi ono bilo efikasno, potrebno je brzo obići klauze koje su upravo postale *implicirane* proširivanjem parcijalne valuacije. Klauza je implicirana ako i samo ako je vrednost svih njenih literala, osim jednog, 0. Intuitivno, ovo bi se moglo sprovesti obilaženjem svih klauza koje sadrže literal kome trenutna parcijalna valuacija dodeljuje vrednost 0 i prebrojavanjem koliko literala sa dodeljenom vrednošću 0 klauza ima. Međutim, ako klauza ima n literala, na ovaj način bi se ona obilazila svaki put kada joj se broj literala, čija je vrednost 0, poveća. Pomoću nadgledanja literala obezbeđuje se da se klauza obiđe samo kada joj se broj literala sa vrednošću 0 poveća sa $n-2$ na $n-1$. Svakoj klauzi se nadgledaju dva odabrana literala kojima nije dodeljena vrednost 0. Može se garantovati da ako nadgledani literali nemaju vrednost 0, klauza nema više od $n-2$ literala čija je vrednost 0 pa ona nije implicirana. Klauze se moraju posetiti samo u slučaju da je vrednost nekog od nadgledanih literala postala 0. Pri svakoj poseti, mora važiti jedan od sledeća dva uslova:

1. Klauza nije implicirana, odnosno postoje dva literala, od kojih je jedan nadgledan, čija vrednost nije 0. Tada se nenadgledani literal čija vrednost nije nula uzima kao zamena za nadgledani literal čija je vrednost postala 0, pa ostaje da važi da nadgledani literali nemaju vrednost 0.

2. Klauza je implicirana. Primećuje se da je implicirana promenljiva upravo nadgledani literal čija vrednost nije postala 0, zato što implicirana klauza ima samo jedan literal koji nema vrednost 0, a jedanom od dva literala je upravo dodeljena vrednost 0.

Invarijanta je da u bilo kom stanju kada klauza može da postane implicirana, oba nadgledana literala nemaju dodeljenu vrednost 0. Bitna karakteristika šeme nadgledanih literala je to što se u trenutku skoka unazad ne moraju menjati nadgledani literali.

Primer 3. *Neka je data klauza* $(\neg p_1 \vee p_2 \vee \neg p_3 \vee p_4 \vee p_5 \vee p_6)$ *i neka su* $\neg p_1$ *i* p_6 *njeni nadgledani literali. Neka je promenljivoj* p_1 *dodeljena vrednost* 1*. Tada literal* $\neg p_1$ *više ne može biti nadgledan jer mu je vrednost* 0*, pa se umesto njega nadgleda* p_2*. Neka su sada promenljivama* p_3 *i* p_4 *dodeljene vrednosti* 1 *i* 0*. U ovom slučaju se nadgledani literali ne menjaju. Neka je sada promenljivoj* p_6 *dodeljena vrednost* 0*. U tom slučaju jedini literal koji se može nadgledati umesto literala* p_6 *jeste* p_5*. Neka je, zatim, promenljivoj* p_2 *dodeljena vrednost* 0*. Pošto se ne može naći literal koji će se nadgledati umesto literala* p_2*, jer svi nenadgledani imaju vrednost* 0*, zaključuje se da je došlo do implikacije i da nadgledani literal* p_5 *mora imati vrednost* 1*. Neka je sada došlo do konflikta i neka se izvršilo vraćanje do trenutka kada je promenljiva* p_1 *dobila vrednost* 1*. Kako se nadgledani literali prilikom vraćanja ne menjaju, literali* p_1 *i* p_5 *ostaju i dalje nadgledani literali klauze.*

2.6 Primene SAT rešavača

Naglo povećanje efikasnosti SAT rešavača, koje je usledilo poslednjih godina, dovelo je do ideje za njihovu praktičnu primenu u različitim oblastima [11]. U nekim oblastima je njihova primena dovela do značajnog povećanja efikasnosti. Neke od tih oblasti su verifikacija hardvera i softvera, ispitivanje ispravnosti modela za sisteme sa konačnim stanjima, planiranje u veštačkoj inteligenciji i za izvođenje haplotipova u bioinformatici.

Primer 4. *Pri ispitivanju ispravnosti dizajna kola, neophodno je ispitivanje da li su neka dva kombinatorna kola ekvivalentna (na primer ispravno kolo i kolo sa greškom). Neka su* C_A *i* C_B *dva kola sa po* n *ulaza i* m *izlaza. Ova dva kola definišu sledeće funkcije:* $f_A : \{0,1\}^n \to \{0,1\}^m$ *i* $f_B : \{0,1\}^n \to \{0,1\}^m$*. Za* $x \in \{0,1\}^n$ *važi* $f_A = (F_{A,1}(x), ..., F_{A,m}(x))$ *i* $f_B = (F_{B,1}(x), ..., F_{B,m}(x))$*. Dva kola nisu ekvivalentna ako je iskazna formula* $\bigvee_{i=1}^{n}(f_{A,i}(x) \oplus f_{B,i}(x))$ *zadovoljiva, što se ispituje SAT rešavačem.*

Primer 5. *Planiranje u veštačkoj inteligenciji je jedno od prvih uspešnih praktičnih primena SAT rešavača. Deterministički tranzicioni sistem je uređena četvorka (S, I, T, G), gde je S skup stanja, $I \in S$ početno stanje, $T \subseteq S \times S$ je skup operatora koji opisuju promene stanja i $G \subseteq S$ skup ciljnih stanja. Skup promenljivih Y označava moguća stanja. Za dati broj stanja k, koji označava dužinu planiranja, potrebno je utvrditi da li se od početnog stanja može doći do nekog ciljnog u k koraka. Ovaj uslov se može predstaviti na sledeći način:*

$$\phi = I(Y_0) \wedge \bigwedge_{0 \leq i \leq k} T(Y_i, Y_{i+1}) \wedge G(Y_k)$$

$T(Y_1, Y_{i+1})$ ima vrednost 1 ako je moguć prelaz sa stanja Y_i na stanje Y_{i+1}. Razmatraju se svi mogući planovi, rastućih dužina, počev od dužine 0. Dužina se povećava dokle god je navedena iskazna formula netačna.

Primer 6. *Kvazigrupa [5] predstavlja uređeni par $(Q, \cdot)$, gde je Q skup, $a \cdot$ binarni operator nad Q tako da jednačine $a \cdot x = b$ i $y \cdot a = b$ imaju jedinstveno rešenje za svaki par elemenata a i b iz skupa Q. Kvazigrupe se mogu razmatrati i kao latinski kvadrat, odnosno $n \times n$ tabela u kojoj se svaki unos (boja) može javiti samo jednom u svakoj koloni i redu. Problem kompletiranja kvazigrupe se može prevesti u SAT problem koristeći n^3 logičkih promenljivih. Promenljiva x_{ijk} označava da je boja k dodeljena polju i, j tabele. Predstavljanje ovog problema kao SAT problem vrši se pomoću sledećih klauza:*

- *Za svako $i, j \in 1, ...n$ konstruiše se klauza $(x_{ij1} \vee x_{ij2} \vee ... \vee x_{ijn})$, koja obezbeđuje da se u svakom polju tabele mora dodeliti neka boja.*
- *Za svako $i, k \in 1, ...n$ konstruišu se klauze $(\neg x_{i1k} \vee \neg x_{i2k})$, $(\neg x_{i1k} \vee \neg x_{i3k})$,...,$(\neg x_{i1k} \vee \neg x_{ink})$, koje obezbeđuju da se u istom redu ne ponovi ista boja.*
- *Za svako $j, k \in 1, ...n$ konstruišu se klauze $(\neg x_{1jk} \vee \neg x_{2jk}), (\neg x_{1jk} \vee \neg x_{3jk}), ...(\neg x_{1jk} \vee \neg x_{njk})$, koje obezbeđuju da se u istoj koloni ne ponovi ista boja.*

Pored praktične primene, SAT rešavači su takođe uticali na značajan broj sličnih problema odlučivanja i optimizacije, koji se mogu smatrati proširenjem SAT problema. Mnogi od ovih problema koriste iste algoritamske tehnike koje se koriste kod SAT rešavača. Neke od ovih oblasti su pseudo-bulovska (PB) ograničenja, maksimalna zadovoljivost (*MaxSAT*) i kvantifikovane bulovske formule (eng. *Quantified-Boolean Formulas*).

Pseudo-bulovska ograničenja uopštavaju SAT problem tako što umesto klauza razmatraju linearne nejednakosti nad bulovskim promenljivim. Problem nalaženja optimalnog rešenja pri ovim uslovima je NP težak i za njihovo rešavanje se koriste tehnike koje su nadogradnja tehnika koje koriste SAT rešavači.

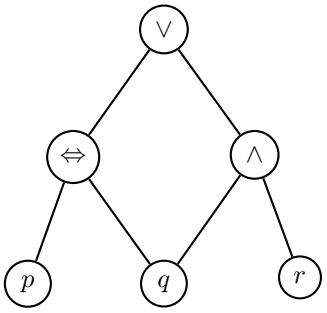
Problem maksimalne zadovoljivosti se definiše na sledeći način: za datu iskaznu formulu u KNF, pronaći onu valuaciju koja maksimizuje broj zadovoljenih klauza. Još neki oblici ovog problema su *parcijalni MaxSAT*, *težinski MaxSAT*

i *težinski parcijalni MaxSAT*. U parcijalnom MaxSAT problemu određene klauze moraju biti zadovoljene, dok ostale ne moraju biti zadovoljene. Kod težinskog MaxSAT problema, svaka klauza ima dodeljenu težinu, a cilj je maksimizovati sumu težina zadovoljenih klauza. U okviru težinskog parcijalnog MaxSAT problema postoje klauze koje moraju biti zadovoljene, dok se preostalim dodeljuju težine i cilj je maksimizovati sumu težina klauza koje budu zadovoljene. Većina efikasnih tehnika SAT rešavača se ne može direktno upotrebiti za rešavanje problema MaxSAT, ali se SAT rešavači mogu iterativno koristiti za rešavanje ovog problema.

Problem kvantifikovanih bulovskih formula predstavlja ispitivanje tačnosti formule $K_1x_1K_2x_2...K_nx_n\phi$ gde ϕ predstavlja iskaznu formulu koja nema slobodne bulovske promenljive, a $K_i \in \{\forall, \exists\}$. Noviji algoritmi za rešavanje ovog problema integrišu i nadograđuju veći deo efikasnih SAT tehnika.

2.7 Sistem URSA

Veliki broj problema iz različitih oblasti se može svesti na SAT problem. URSA sistem [8], razvijen na Univerzitetu u Beogradu, služi za rešavanje različitih problema, na primer kombinatornih ili verifikacionih, svođenjem na SAT problem. Iako postoji nekoliko sličnih sistema, izabran je URSA zato što je otvorenog koda i ne mora da koristi samo jedan rešavač. Sistem je implementiran u programskom jeziku C++ i ima sopstveni jezik za opisivanje ulaznih problema, koji je kombinacija imperativnih i deklarativnih programskih paradigmi. Ulazni problem se prevodi u iskaznu formulu čijim rešavanjem se dobija rešenje početnog problema. Formula se rešava prosleđivanjem jednom od SAT rešavača koji su povezani sa sistemom URSA. Ako je formula zadovoljiva, njen model se prevodi u rešenje navedenog problema, u suprotnom problem nema rešenje. U ulaznoj datoteci, koja opisuje problem, može se zahtevati samo jedno ili sva rešenja problema. Formule se u sistemu URSA predstavljaju u obliku usmerenih acikličkih grafova. Kako bi se povećale prostorna i vremenska efikasnost, za čuvanje iskaznih formula se koristi tehnika deljenih izraza. Svaka podformula se čuva samo jednom, ali ona može biti deo više različitih formula. Tako se formule ne čuvaju pojedinačno, već u obliku jednog usmerenog grafa. SAT rešavači koji se koriste u sistemu URSA su KNF rešavači pa zahtevaju transformisanje formule u KNF, za šta se koristi Cajtinovo kodiranje.



Slika 2.2: Formula $(p \Leftrightarrow q) \vee (q \wedge r)$ predstavljena u sistemu URSA

Glava 3

Ne-KNF rešavači

Veliki broj problema se prirodno opisuje iskaznim formulama koje nisu u KNF. Njihovim pretvaranjem u KNF se gube strukturne informacije koje se mogu iskoristiti za poboljšanje efikasnosti SAT rešavača. Zbog toga se, tokom prethodnih godina, počelo raditi na razvijanju i ne-KNF SAT rešavača. U ovoj glavi biće opisana dva takva rešavača. Prvi rešavač je NoClause [15], koji primenjuje DPLL proceduru na formule koje nisu u KNF, dok je drugi rešavač NflSAT [6], koji radi sa formulama koje su u NNF. Opisana su samo dva navedena rešavača zbog toga što ne postoji veliki broj ne-KNF rešavača koji su javno dostupni.

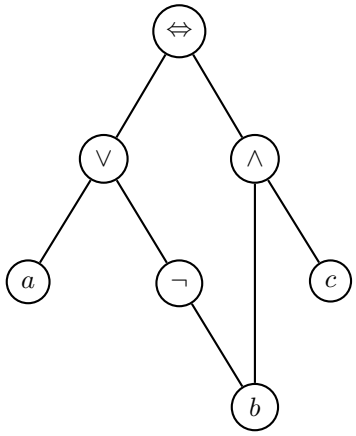
3.1 NoClause rešavač

Ne-KNF SAT rešavači mogu koristiti vrlo raznovrsne tehnike za rešavanje, koje se uopšte ne moraju zasnivati na DPLL proceduri. Ipak neki rešavači, kao NoClause [1] , primenjuju neku vrstu DPLL procedure prilagođenu formulama koje nisu u KNF.

Rešavač NoClause čuva formulu u obliku usmerenog acikličkog grafa (skraćeno DAG, od engleskog *Directed Acyclic Graph*) kome su sva pojavljivanja neke podformule predstavljena samo jednim čvorom u grafu (slika 3.1). Ovako predstavljene formule se često nazivaju bulovska kola. Cilj rešavača je da neprotivrečno označi čvorove grafa istinitosnim vrednostima tako da je čvor na najvišem nivou (koji predstavlja celu formulu) označen sa 1, ili da dokaže da takvo označavanje ne postoji. Označavanje je neprotivrečno, ako se poštuje logika čvorova grafa. Na primer, ako je neki čvor tipa $\wedge$ označen sa 1, onda svako od njegovo dete mora biti isto označeno sa 1.

Rešavač NoClause koristi *backtracking* proceduru, koja bira neoznačeni čvor (koji može biti bilo koji neoznačeni u grafu), označi ga sa 1 ili 0, i posledice tog označavanja propagira kroz graf. U slučaju kada nema mogućnosti za dalje

[1] `http://www.cs.toronto.edu/~fbacchus/sat.html#Noclause` Autori: C. Thiffault, F. Bacchus i T. Walsh



Slika 3.1: Formula $(a \vee \neg b) \Leftrightarrow (b \wedge c)$ predstavljena pomoću DAG-a

propagiranje, bira se sledeći čvor koji će se označiti. Ovaj postupak odgovara *split* pravilu DPLL procedure. Propagiranje se vrši kroz ceo graf, i ka višim i ka nižim nivoima, koristeći jednostavan skup pravila. Na primer, ako je čvor označen sa 0, onda se ta vrednost propagira svim njegovim roditeljima koji su tipa $\wedge$. Slično, ako je čvor označen sa 1, i ako je on tipa $\wedge$, onda se njegova vrednost propagira svoj deci. Takođe, ako je čvor tipa $\vee$ označen sa 0, njegova vrednost se propagira svoj njegovoj deci. Kontradikcija se detektuje u trenutku kada neki čvor treba da bude označen i sa 1 i sa 0. U tom slučaju dolazi do vraćanja unazad kako bi se isprobalo drugačije označavanje. Dodeljivanje istinitosne vrednosti čvoru tačno odgovara dodeljivanju istinitosne vrednosti promenljivoj koja predstavlja podformulu u Cajtinovom kodiranju. Takođe, propagiranje oznaka kroz graf odgovara pravilu *unit propagation* DPLL procedure.

Za svako označavanje do kog je došlo usled propagiranja, pamti se skup čvorova čije su oznake uzrokovale tu propagaciju. Tako, ako je došlo do kontradikcije u nekom čvoru, može se na osnovu zapamćenih skupova čvorova napraviti konfliktni skup, od koga dalje se može napraviti konfliktna klauza, nalik onoj kod KNF SAT rešavača. Na primer, ako se neki čvor tipa $\wedge$ označi sa 1 zato što su sva njegova deca označena sa 1, onda će se kao razlog označavanja pamtiti oznake 1 sve dece označenog čvora. Ako se taj isti $\wedge$-čvor kasnije označi sa 0 zbog propagiranja od nekog roditelja, postojao bi još jedan skup oznaka čvorova koji bi bio razlog za dodelu oznake 0. Ove oznake se onda kombinuju i dobija se konfliktni skup. Negacija oznaka u konfliktnom skupu daje konfliktnu klauzu koja je nalik onim konfliktnim klauzama koje se dobijaju u KNF SAT rešavačima. Ove klauze se čuvaju u bazi klauza i koriste se za propagiranje oznaka. Kako bi propagiranje bilo efikasnije, NoClause rešavač korist i varijantu šeme nadgledanih veznika koju primenjuje na bulovsko kolo.

Rešavač NoClause koristi nadgledanje uvek kada se propagiranje može učiniti efikasnijim. Tipičan primer za ovo je propagiranje kroz čvor tipa $\wedge$, čija su

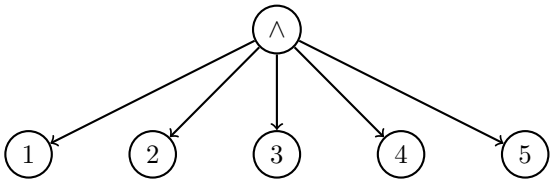
pravila sledeća:

1. Ako čvor $\wedge$ dobije oznaku 1, propagira se 1 svoj deci.
2. Ako neko dete dobije oznaku 0, propagira se 0 svim $\wedge$ roditeljima.
3. Ako sva deca postanu označena sa 1, propagira se 1 $\wedge$ roditelju.
4. Ako $\wedge$-čvor dobije oznaku 0 i sva deca, osim jednog, su označena kao 1, onda se vrednost 0 propagira neoznačenom detetu.

Nadgledanje ne pomaže za prva dva pravila, ali kod trećeg i četvrtog pravila poboljšanje je primetno. Svakom $\wedge$-čvoru dva deteta postaju *tačno nadgledana*. Kada je čvor tačno nadgledan njemu se ne dodeljuje oznaka 1, osim ako nema drugog izbora, ili je drugi nadgledani čvor (njegov parni) označen sa 0.

- Kada neki čvor dobije oznaku 1, razmatra se svaki njegov roditelj kome je on nadgledano dete. Svakom roditelju se prvo razmatra drugo nadgledano dete i ako ono ima oznaku 0 ništa se ne preduzima.
- Međutim, ako je on označen sa 1, onda su i sva ostala deca označena sa 1, pa se ta vrednost propagira i njihovom roditelju, što je aktiviranje trećeg pravila propagiranja za čvor $\wedge$.
- Ako ovo ne važi (drugi nadgledani čvor nema dodeljenu vrednost), onda se među ostalom decom traži ono koje ili nije označeno ili je označeno sa 0 kako bi postalo novi nadgledani čvor umesto nadgledanog čvora koji je upravo označen sa 1.
- Ako i to nije moguće, proverava se da li je $\wedge$-čvor označen sa 0. Ako jeste, onda se aktivira četvrto pravilo i propagira se 0 jedinom čvoru kome nije dodeljena vrednost. Ovo je moguće uraditi zbog toga što ako čvor $\wedge$ ima nadgledano dete označeno sa 1, jedino njegovo neoznačeno dete jeste baš drugi nadgledani čvor, zato što su i sva preostala deca označena sa 1.

Analogno se tretiraju i $\vee$-čvorovi, čiji deca mogu biti netačno nadgledani. Ovakva tehnika nadgledanja, koja se slično primenjuje i nad čvorovima tipa $\vee$, $\Leftrightarrow$ i $\oplus$, znatno utiče na povećanje efikasnosti rešavanja, pogotovo što NoClause podržava n-arne bulovske operatore.



Slika 3.2: Primer propagiranja

Primer 7. *Pomoću slike 3.2 biće ilustrovana opisana pravila propagiranja. Neka su čvorovi* 1 *i* 2 *trenutno nadgledani čvorovi, od kojih je čvor* 1 *označen sa* 1*, dok čvor* 2 *nema oznaku.*

Neka su sada čvorovi 1 *i* 2 *trenutno nadgledani čvorovi i neka ni jedan čvor nije označen. Ako čvor* 2 *na neki način dobije oznaku* 1*, on više ne može biti nadgledan čvor pa se mora naći zamena. Kako ni jedan od čvorova* 3*,* 4 *i* 5 *nema oznaku, bilo koji od njih može postati novi nadgledani čvor umersto čvora* 2*.*

Ako nadgledani čvor 2 *na neki način dobije oznaku* 1*, onda se ta oznaka može propagirati i roditeljskom čvoru. Ovo je izvodljivo zbog toga što je drugi nadgledani čvor, čvor* 1*, već označen sa* 1*, što znači da su i sva ostala deca sa istom oznakom, inače čvor* 1 *uopšte ne bi bio nadgledan, već bi to bio neki čvor koji ili nema oznaku, ili je označen sa* 0*. U ovom slučaju je došlo do trećeg slučaja propagiranja za čvor tipa* $\wedge$*.*

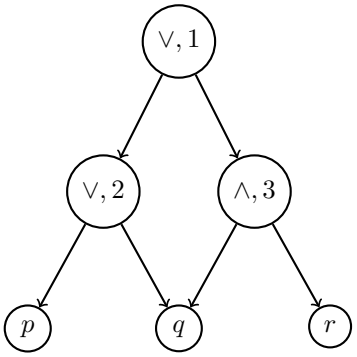
Ako roditeljski čvor na neki način dobije oznaku 0*, onda se ta oznaka može propagirati čvoru* 2*. Ovo je moguće uraditi zbog toga što nadgledani čvor* 1 *ima oznaku* 1*, pa su i svi ostali čvorovi (koji nisu nadgledani) označeni sa* 1*, što znači da je jedini neoznačeni čvor baš nadgledani čvor* 2*, pa se on mora označiti sa* 0*.*

Kako bi se izbeglo grananje pri pretrazi na promenljivim čija vrednost ne utiče na ukupnu vrednost formule, NoClause beleži promenljive čije vrednosti nisu bitne u odnosu na vrednost formule i pri tom dobija na efikasnonsti zbog korišćenja nadgledanja. Da li je vrednosti nekog čvora bitna u odnosu na drugi se vidi na primeru čvora koji je dete čvora tipa $\wedge$ označenog sa 0. Vrednost tog čvora ne utiče na vrednost svog $\wedge$ roditelja, ako je on označen sa 0 zbog, na primer, toga što je neko drugo njegovo dete označeno sa 0. Međutim, vrednost čvora može da utiče na vrednosti drugih roditelja, pa zato vrednost čvora postaje nebitna u odnosu na celu formulu samo ako za svakog njegovog roditelja važi ili da vrednost čvora nije bitna za vrednost tog roditelja ili da je sama vrednost roditelja postala nebitna u odnosu na celu formulu.

Primer 8. *Neka je formula* $(p \vee q) \vee (q \wedge r)$ *predstavljena grafovski kao na slici 3.3. Neka je čvoru* p *označen sa* 1*. Tada se ta vrednost može propagirati na njegov roditeljski čvor (*2*), a na sličan način i na koreni čvor* 1*, koji ujedno predstavlja celu formulu, pa je i cela formula zadovoljena. U tom slučaju oznake čvorova* q*,* r *i* 3 *ne utiču na vrednost cele formule.*

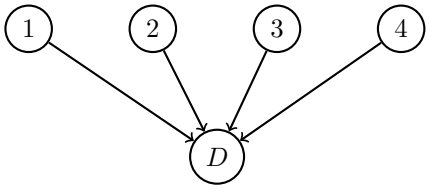
Neka su čvorovima p *i* r *dodeljene oznake* 0*. Tada je oznaka čvora* q *ne utiče na oznaku čvora* 3*, koja je* 0 *zbog deteta* r*, ali utiče na to da li je formula zadovoljiva ili ne.*

Da bi se efikasno propagirala oznaka **nebitno** kroz ceo graf, za svaki čvor se nadgleda jedan roditelj. Nadgledani roditelj ne sme da ima vrednost **nebitno**, niti sme da se desi da deca za koju se nadgleda budu nebitna u odnosu na njegovu vrednost. Kad god se čvoru dodeli istinitosna vrednost koja njegovu decu čini nebitnom u odnosu na njegovu vrednost, ili kada on bude označen sa



Slika 3.3: Formula $(p \vee q) \vee (q \wedge r)$ predstavljena pomoću DAG-a

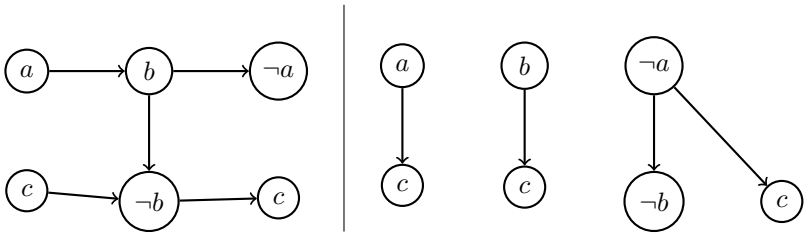
nebitno pomoću propagiranja, za svako dete se traži novi roditelj koji će se nadgledati. Ako se takav roditelj ne nađe, onda se vrednost **nebitno** prosleđuje deci. Korišćenje nadgledanja omogućava da se izračunavanje vrši samo kada se nadgledani roditelj menja, promena vrednosti drugih roditelja ne zahteva nikakvo računanje.



Slika 3.4: Primer propagiranja oznake **nebitno**

Primer 9. *Pomoću slike 3.4 biće prikazan primer propagiranja oznake* ***nebitno****. Neka je čvor* 1 *nadgledani roditelj čvora D. Neka je oznaka čvora D ili nebitna u odnosu na oznake čvorova* 2, 3 *i* 4*, ili su ti čvorovi već označeni sa* ***nebitno****. Neka je čvor* 1 *dobio oznaku* ***nebitno****. Kako ne postoji ni jedan roditeljski čvor koji bi se mogao nadgledati, zbog toga što oznaka čvora D ne utiče na oznake nenadgledanih roditelja, to se oznaka* ***nebitno*** *može propagirati čvoru D.*

NoClause rešavač koristi tehniku koja se naziva redukcija konfliktnih klauza (eng. *Conflict Clause Reduction*), koja se zasniva na korišćenju informacija dobijenih iz strukture formule. Kada rešavač nauči klauzu na osnovu konflikta, ona sadrži oznake nekih čvorova. NoClause na osnovu strukture formule ispituje da li su neke od njih "lokalno" redundantne. Oznaka a čini oznaku a' redundantnom ako jedno od pravila propagiranja generiše a' krenuvši od a. Na primer, neka je a čvor tipa $\wedge$ i neka je a' njegovo dete. Ako čvorovi a i a' imaju vrednost 0, onda a čini redundantnim a'. Tada, ako se u nekoj naučenoj klauzi pojave čvorovi a i a' sa navedenim vrednostima, onda se zbog redundantnosti iz klauze može izbaciti a'.



Slika 3.5: *Hpgraph* (levo) i *vpgraph* formule $((a \vee b) \wedge c) \vee (\neg a \wedge (\neg b \vee c))$

3.2 NflSAT rešavač

Rešavač NflSAT [2] transformiše ulaznu formulu u negacijsku normalnu formu (skraćeno NNF), koju dalje čuva u određenim grafovskim strukturama. Ispitivanje zadovoljivosti NflSAT vrši primenom modifikovane DPLL procedure na navedene grafovske strukture.

Pretvaranje formule u negacijsku normalnu formu, NflSAT rešavač vrši u dve etape. U prvoj etapi veznici $\Leftrightarrow$, $\oplus$, $\Rightarrow$ se menjaju odgovarajućom kombinacijom $\wedge$, $\vee$ i $\neg$ veznika. Da bi se izbeglo naglo povećanje veličine formule dozvoljava se deljenje podformula. Druga etapa uklanja deljenja uvođenjem novih promenljivih, nakon čega se vrši uvlačenje negacija pomoću De Morganovih pravila. Ovo prevođenje u NNF se ze vrši u linearnom vremenu.

NflSAT čuva NNF formule u vidu dva posebna grafa, koji se zovu *hpgraph* i *vpgraph*. Oni se na osnovu ulazne formule, za koju će se dalje smatrati da je u NNF, dobijaju rekurzivno u linearnom vremenu. Neka je ϕ ulazna formula.

Hpgraph formule ϕ, u oznaci $G_h(\phi)$, se definiše kao uređena petorka (V, R, L, E, Lit), gde su V skup čvorova koji odgovara svim pojavljivanjima literala u NNF formule, $R \subseteq V$ skup korenih čvorova, $L \subseteq V$ skup listova i $E \subseteq V \times V$ skup grana. $Lit(n)$ označava literal koji je predstavljen čvorom $n \in V$. Ulazni stepen korenih čvorova i izlazni stepen listova je 0. Za *hpgraph* važi da svaka putanja od korena do lista predstavlja jednu klauzu koja se sastoji od literala koji se javljaju na toj putanji. Skup svih klauza (putanja od korena do lista) koje se javljaju u *hpgraph*-u čini KNF formule ϕ.

Vpgraph formule ϕ, u oznaci $G_v(\phi)$, se definiše kao uređena petorka $(V^{'}, R^{'}, L^{'}, E^{'}, Lit^{'})$ na sličan način kao i *hpgraph*. Razlika je u tome što kod *vpgraph-a* važi da svaka putanja od korena do lista predstavlja konjunkciju literala koji se nalaze na tom putu. Skup svih tih konjunkcija čini disjunktivnu normalnu formu (DNF) ulazne formule ϕ.

Na osnovu date formule ϕ, koja je u NNF, *vpgraph* se konstruiše rekurzivno u linearnom vremenu na sledeći način:

1. Ako je $\phi = l$, gde je l literal, onda se pravi graf koji sadrži samo jedan čvor sa oznakom i, gde je i slobodan identifikator. Literal predstavljen čvorom

[2] `http://www.cs.cmu.edu/~hjain/nflsat-web/` Autori: Himanshu Jain i Edmund M. Clarke

i je baš l. Povratna vrednost je graf $G_v(\phi) = (\{i\}, \{i\}, \{i\}, \emptyset, Lit)$, gde je $Lit(i) = l$.

2. Ako je $\phi = \phi_1 \vee \phi_2$, onda se *vpgraph* formule ϕ dobija unijom *vpgraph*-ova formula ϕ_1 i ϕ_2. Ako su $G_v(\phi_1) = (V_1, R_1, L_1, E_1, Lit_1)$ i $G_v(\phi_2) = (V_2, R_2, L_2, E_2, Lit_2)$, onda je graf $G_v(\phi)$ jednak uniji ova dva grafa, koja se dobija kao $G_v(\phi) = (V_1 \cup V_2, R_1 \cup R_2, L_1 \cup L_2, E_1 \cup E_2, Lit_1 \cup Lit_2)$.

3. Ako je $\phi = \phi_1 \wedge \phi_2$, onda se *vpgraph* formule ϕ dobija konkatenacijom *vpgraph*-a formule ϕ_1 i *vpgraph*-a formule ϕ_2. Ako su $G_v(\phi_1) = (V_1, R_1, L_1, E_1, Lit_1)$ i $G_v(\phi_2) = (V_2, R_2, L_2, E_2, Lit_2)$, onda $G_v(\phi)$ sadrži sve čvorove i grane grafova $G_v(\phi_1)$ i $G_v(\phi_2)$. Pored toga, u grafu $G_v(\phi)$ postoje i grane koje počinju u listovima grafa $G_v(\phi_1)$ i završavaju se u korenim čvorovima grafa $G_v(\phi_2)$, pa je $G_v(\phi) = (V_1 \cup V_2, R_1, L_2, E_1 \cup E_2 \cup (L_1 \times R_2), Lit_1 \cup Lit_2)$

Konstrukcija *hpgraph*-a je slična prethodno opisanoj proceduri za dobijanje *vpgraph*-a. Razlika je u tome što se u slučaju kada je formula $\phi = \phi_1 \wedge \phi_2$ njen *hpgraph* se dobija kao unija *hpgraph*-ova za formule ϕ_1 i ϕ_2, dok se u slučaju kada je $\phi = \phi_1 \vee \phi_2$ njen *hpgraph* dobija konkatenacijom *hpgraph*-ova za formule ϕ_1 i ϕ_2.

Pomoću *hpgraph*-a ulazne formule, mogu se detektovati konflikti i implikacije u SAT rešavaču i to na sledeći način. Neka je data parcijalna valuacija formule ϕ. Ta valuacija negira ϕ (došlo je do konflikta) ako i samo ako postoji neki put od korena do lista grafa $G_h(\phi)$, tako da ta valuacija negira svaki čvor na tom putu. Slično, neka postoji put od korena do lista, u grafu $G_h(\phi)$, kome čvor a pripada i neka je data valuacija koja negira svaki čvor tog puta, osim čvora a, za koji važi da $Lit(a)$ nema dodeljenu vrednost. Tada je literal $Lit(a)$ impliciran literal.

Za efikasnu detekciju konflikata i implikacija, NflSAT koristi uopštenje šeme dvostrukog nadgledanja literala koja se primenjuje na *hpraph*-u. Neka je data parcijalna valuacija v fomule ϕ, koja je u NNF. Za dati *hpgraph* $G_h(\phi)$ skup $C \subseteq V$ je *presek koren-list* (eng. *root leaf cut*) ako i samo ako se uklanjanjem svih čvorova skupa C iz grafa $G_h(\phi)$ prekidaju svi putevi od korenih čvorova ka listovima. Presek koren-list je *prihvatljiv*, za neku parcijalnu valuaciju, ako i samo ako ne postoji literal, predstavljen čvorom iz datog preseka, čija je vrednost 0 za datu valuaciju. Presek koren-list je *zadovoljen*, za neku parcijalnu valuaciju, ako i samo ako je svaki literal, predstavljen čvorovima koji pripadaju preseku, tačan u datoj valuaciji. Za neki presek koren-list se kaže da je *minimalan* ako i samo ako ne postoji pravi podskup tog preseka koji je takođe presek koren-list.

Svaka putanja P od korena do lista u *hpgraph*-u predstavlja jednu klauzu, pa se šema dvostrukog nadgledanja literala svodi na nadgledanje dva čvora a_1 i a_2, koji pripadaju putu P. Ovo odgovara nadgledanju literala $Lit(a_1)$ i $Lit(a_2)$ klauze predstavljene putanjom P. Međutim, kako u *hpgraph*-u obično ima eksponencijalno mnogo putanja od korena do lista, ovakvo nadgledanje bi bilo skupo. Zbog toga se nadgledanje vrši pomoću preseka koren-list i to koristeći osobinu da ovakav presek sadrži bar po jedan čvor svake putanje od korena do

lista, odnosno po jedan literal svake klauze predstavljene tim putanjama. Tako se šema dvostrukog nadgledanja literala svodi na nadgledanje dva preseka koren-list *hpgraph*-a.

Za dati *hpgraph* čuvaju se dva preseka koren-list, C_1 i C_2, koji se nadgledaju. Na početku, to mogu biti bilo koja dva disjunktna preseka (nemaju zajednički čvor). Tokom reăvanja jedan od preseka uvek mora biti prihvatljiv, a kad god je to moguće, preseci su i međusobno disjunktni. Slučajevi koji se mogu javiti su sledeći :

1. Oba preseka C_1 i C_2 su prihvatljivi i međusobno disjunktni. Tada ne postoji konflikt niti implikacija, zato što svaka klauza u *hpgraph*-u sadrži dva literala koja nisu netačna.

   Neka jedan od preseka, na primer C_2 nije više prihvatljiv.

2. Neka je sada presek C_1 zadovoljen. Ni u ovom slučaju nema konflikta niti implikacije, zato što je svaka klauza predstavljena *hpgraph*-om zadovoljena, zbog zadovoljenosti preseka C_1, pa se ne menja presek C_2.

   Ako se nije desio nijedan od prethodna dva slučaja, onda se traži zamena za presek C_2 i to takva da je prihvatljiva i da je disjunktna u odnosu na presek C_1. Ako ona ne postoji, onda se desio neki od sledeća dva slučaja.

3. **Konflikt:** Ne postoji prihvatljivi presek u *hpgraph*-u. Tada trenutna valuacija negira formulu.

4. **Implikacija:** Postoji prihvatljivi presek C_3 koji bi zamenio C_2, ali nije disjunktan sa presekom C_1. Tada, svaki literal $Lit(a)$, $a \in C_1 \cap C_3$, jeste implicirani literal, uz pretpostavku da $Lit(a)$ već nema vrednost tačno. Ako važi da je $C_1 \neq C_3$, onda se presek C_2 menja presekom C_3.

Da bi se efikasno pronalazili i ažurirali preseci koren-list, NflSAT koristi *vpgraph*, zbog osobine da sve putanje od korena do lista u *vpgraph*-u odgovaraju svim minimalnim presecima koren-list u *hpgraph*-u. Za preseke koji se nadgledaju uzimaju se dva minimalna preseka koren-list, koji se nalaze pretraživanjem odgovarajućeg *vpgraph*-a pretragom u dubinu.

Primer 10. *Neka su dati* hpgraph *i* vpgraph *formule* $((a \vee b) \wedge c) \vee (\neg a \wedge (\neg b \vee c))$ *(slika 3.5) i neka su posmatrani preseci* $C_1 = \{a, c\}$ *i* $C_2 = \{\neg a, \neg b\}$. *Neka je presek* C_2 *postao neprihvatljiv, odnosno neka za trenutnu valuaciju* v *važi da je, na primer,* $v(b) = 1$. *Tada se traži novi prihvatljivi presek koji će se nadgledati umesto preseka* C_2. *Postoje dva takva preseka* $C_3 = \{b, c\}$ *i* $C_4 = \{\neg a, c\}$, *ali oni nisu disjunktni sa presekom* C_1, *već imaju zajednički literal* c. *Neka je presek* C_2 *zamenjen presekom* C_3. *Kako je zajednički literal preseka* C_1 *i* C_3 *literal* c, *dolazi do implikacije pa se njemu mora dodeliti vrednost* 1. *Tada nadgledani presek* C_3 *postaje zadovoljen, pa je ujedno i cela formula zadovoljiva.*

Ovakvo nadgledanje, samo pomoću dva preseka, može biti neefikasno kada *hpgraph* ima milione čvorova, zato što minimalni koren-list presek može biti

veliki. Kako u praksi *hpgraph* ima veliki broj komponenti povezanosti, koje su male po broju čvorova u odnosu na sam graf, zbog efikasnosti se nadgledaju po dva koren-list preseka za svaku komponentu povezanosti. NflSAT na osnovu konflikta uči nove klauze, koje posebno čuva, tako da se pri nadgledanju, pored *hpgraph*-a, u obzir uzimaju i te klauze.

Glava 4

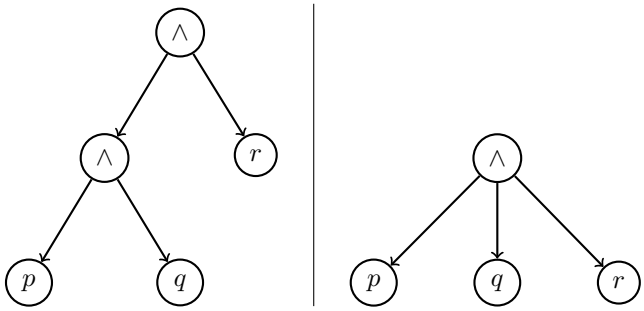
Ugradnja ne-KNF SAT rešavača u sistem URSA

U daljem tekstu je opisano na koji način je ostvarena veza sistema URSA sa rešavačima NoClause i NflSAT. Oba rešavača su predstavljena apstraktnom klasom `NonCnfSolvers`, a samo povezivanje rešavača sa sistemom URSA je napisano u jeziku C++.

4.1 Klasa NonCnfSolvers

Kako sistem URSA za rešavanje koristi SAT rešavače, pogodno je da se oni predstave na jedinstven način. Ipak, postoji dovoljno razlika između ne-KNF i KNF rešavača koje otežavaju da se svi rešavači predstave jedinstveno. Na primer, KNF rešavači kao ulaz očekuju formulu u KNF, pa im se može prosleđivati niz klauza. Kod ne-KNF rešavača ovo nije slučaj jer oni kao ulaz očekuju bilo koju formulu koja se zatim interno predstavlja na način koji se može razlikovati od rešavača do rešavača. Zbog toga su svi ne-KNF rešavači predstavljeni apstraktnom klasom `NonCnfSolvers`, koja sadrži sledeće metode:

- `void addFormula(Formula *f)`
  - prosleđuje pokazivač na formulu rešavaču
- `void InitSolver()`
  - vrši podešavanja početnih parametara rešavača
- `bool solve()`
  - ispituje zadovoljivost formule
- `unsigned int varNum()`
  - vraća proj promenljivih koje se javljaju u formuli



Slika 4.1: Formula $(p \wedge q) \wedge r$ predstavljena u sistemu URSA(levo) i rešavaču NoClause (desno)

- `bool isTrueVar(unsigned int var)`
  - vraća vrednost određene promenljvie

Tokom konstrukcije KNF neke formule, što se dešava pre poziva KNF SAT rešavača, mogu se pobrojati sve promenljive koje se u formuli javljaju. Ovo se u slučaju ne-KNF SAT rešavača, zbog efikasnosti, radi tokom konstrukcije interne reprezentacije formule pa je neophodno da postoji metod koji će da vrati broj promenljivih koje sadrži formula.

4.2 NoClause i sistem URSA

Nakon što rešavač dobije pokazivač na formulu čiju zadovoljivost treba da ispita, formula se mora predstaviti na onaj način na koji rešavač čuva formulu. NoClause, kao što je već rečeno, čuva formulu u obliku usmerenog acikličkog grafa. Ovaj način čuvanja je sličan načinu koji koristi sistem URSA, uz jednu bitnu razliku. Naime, čvorovi grafa u sistemu URSA koji predstavljaju logičke operatore $\wedge$ i $\vee$ su binarni, dok su kod rešavača NoClause oni n-arni.

Ipak, ove dve strukture su dovoljno slične pa je moguće napraviti reprezentaciju formule u NoClause rešavaču samo jednim prolazom kroz usmereni graf kojim je predstavljena formula u sistemu URSA, i to koristeći pretragu u dubinu. U slučaju da se pretragom naiđe na dva ili više uzastopna čvora koji predstavljaju isti logički operator tipa $\wedge$ ili $\vee$ oni se u NoClause-u predstavljaju jednim čvorom, a deca svih tih čvorova (ne računajući čvorove koji se spajaju u jedan) postaju deca tog jednog koji ih predstavlja.

Kao što je ranije pomenuto, sistem URSA može vratiti ili jedno ili sva rešenja ulaznog problema. Da bi vratio sva rešenja, SAT rešavač mora biti u mogućnosti da vrati sve modele formule. NoClause, sam po sebi, ispituje samo da li je formula zadovoljiva i nema mogućnost vraćanja svih mogućih modela. Ovaj problem je prevaziđen korišćenjem blokirajućih klauza. Radi efikasnosti, kada se izgradi graf koji predstavlja formulu u rešavaču, čuva se i njegov duplikat, jer se tokom rada rešavača početni graf transformiše u kompaktniju reprezentaciju

radi uštede memorije, koja se kasnije ne može menjati. Svaki put kada se nađe model formule, duplikat se koristi kao prethodni graf, kome se doda blokirajuća klauza. Dodavanje je jednostavno u slučaju da je koreni čvor tipa $\wedge$ klauza se direktno nadovezuje na koren. U suprotnom, pravi se novi koren koji je tipa $\wedge$, na koji se vezuju prethodni graf i blokirajuća klauza. Pravljenje novog korena se može desiti samo jednom i to nakon nalaženja prvog modela. Kada se napravi novi graf sa dodatom blokirajućom klauzom, čuva se njegova kopija umesto duplikata prethodnog grafa.

Po rečima autora rešavača NoClause, rad rešavača sa formulama koje sadrže veznike tipa $\Leftrightarrow$ i $\oplus$ nije dovoljno testiran zbog nedostatka test primera sa tim veznicima, pa nije sigurno da će rešavač dobro raditi sa takvim formulama. Zbog toga su ograničili da ulaz ne sadrži veznike $\Leftrightarrow$ i $\oplus$. Radi testiranja, omogućeno je da rešavač radi sa navedenim veznicima, ali već nakon prvih test primera je utvrđeno da zaista za neke formule koje sadrže navedene veznike, rešavač ne radi ispravno. Ovaj problem je prevaziđen na dva različita načina.

Prvi način je jednostavan. Pri izgradnji grafa koji predstavlja formulu, veznici $\Leftrightarrow$ i $\oplus$ se menjaju odgovarajućim kombinacijama veznika $\wedge$, $\vee$ i $\neg$, koristeći logičke ekvivalencije $p \Leftrightarrow q \equiv (\neg p \vee q) \wedge (p \vee \neg q)$ i $p \oplus q \equiv (\neg p \wedge q) \vee (p \wedge \neg q)$. Ovo ipak dovodi do eksponencijalnog rasta grafa kojim se u NoClause predstavlja formula, što znatno utiče na efikasnost rešavača.

Kako bi se izbegao eksponencijalni rast, drugi način koristi Cajtinovo kodiranje za uklanjanje veznika $\wedge$ i $\oplus$ ali po ceni uvođenja novih promenljvih. Tako se, na primer, podformula $P \oplus Q$ uklanja korišćenjem dve nove promenljive i šest klauza. Najpre se iskoristi ekvivalencija koja je korišćena u prvom pristupu, a zatim se svaka konjunkcija menja sa tri klauze i sa jednom novom promenljivom. Nove klauze se dodaju na isti način na koji se dodaju blokirajuće klauze.

Ipak, i jedan i drugi način znatno usporavaju rad rešavača, zbog toga što se u oba slučaja narušava struktura ulazne formule, a tehnike rešavanja koje NoClause koristi su upravo strukturno zasnovane. U krajnjoj implementaciji je ostavljena mogućnost biranja da li će i na koji će se način menjati veznici $\Leftrightarrow$ i $\oplus$.

4.3 NflSAT i sistem URSA

Za razliku od rešavača NoClause, koji ima metode za pravljenje grafovske reprezentacije formule, rešavač NflSAT nema javne metode za pristup i izgradnju samih struktura. Međutim, kod njega postoje metode koje rade sa različitim vrstama ulaznih datoteka, pa se one mogu koristiti za pravljenje struktura *hpgraph* i *vpgraph* na osnovu grafa iz sistema URSA. Skup metoda koji je korišćen za spajanje NflSAT-a sa sistemom URSA jeste onaj koji se koristi za obradu ne KNF DIMACS formata.

Ne-KNF DIMACS format, ili EDIMACS, je format kojim se opisuje logička formula (bulovsko kolo) koja može sadržati veznike bilo kog tipa. Svaka linija u jednoj EDIMACS datoteci ima sledeći oblik:

```
VEZNIK BRPARAMETARA PARAMETAR1 ... PARAMETARk UI0 ... UIn 0
```

- `VEZNIK` - ovo je pozitivan ceo broj,različit od nule, koji predstavlja vrstu veznika. Moguće vrednosti će biti kasnije navedene.
- `BRPARAMETARA` - označava broj parametara koji sledi posle njega i ne sme biti nula. Vrednost -1 označava da posle njega nema nijednog parametra.
- `PARAMETARi` - ceo broj, čije tumačenje zavisi od vrste veznika.
- `UIi` - može biti bilo koji ceo broj, različit od nule, i označava ulaz/izlaz trenutnog veznika. Njegovo tumačenje zavisi od vrste veznika. Ako je negativan broj, znači da je taj ulaz/izlaz negiran. Najveći `UIi` broj označava izlaz cele formule.
- `0` - svaki red se završava ovim karakterom.

Kao što je rečeno, svaki veznik je predstavljen pozitivnim celim brojem koji je različit od nule. Postoji određen broj veznika čiji je broj već definisan, ali postoji prostor za definisanje novih. U datoj tabeli su navedeni veznici koji su bitni za vezu između NflSAT-a i sistema URSA. Vrednost njihovog `BRPARAMETARA` je uvek -1, pa se neće navoditi u tabeli.

`VEZNIK`	Tip	`UI`	opis
3	$\neg$	`UI0`- izlaz, `UI1`- ulaz	unarna negacija `UI0`= $\neg$ `UI1`
4	$\wedge$	`UI0`- izlaz, `UI1`... `n` - ulazi	n-arna konjunkcija, `UI0`= `UI1`$\wedge$ ... $\wedge$ `UIn`
6	$\vee$	`UI0`- izlaz, `UI1`... `n` - ulazi	n-arna disjunkcija, `UI0`= `UI1`$\vee$ ... $\vee$ `UIn`
8	$\oplus$	`UI0`- izlaz, `UI1`... `n` - ulazi	n-arna ekskluzivna disjunkcija, `UI0`= `UI1`$\oplus$ ... $\oplus$ `UIn`
11	$\Leftrightarrow$	`UI0`- izlaz, `UI1`... `n` - ulazi	n-arna ekvivalencija, `UI0`= `UI1`$\Leftrightarrow$ ... $\Leftrightarrow$ `UIn`

Primer 11. *Zapis formule* $(p \Leftrightarrow q) \vee (r \wedge \neg s)$ *u EDIMACS formatu:*

```
p noncnf 7
11 5 1 2
4 6 3 -4
6 7 5 6
```

Skup funkcija rešavača NflSAT koji se koristi za čitanje ulaza u EDIMACS formatu je iskorišćen za pravljenje veze sa sistemom URSA. Najpre se jednom pretragom u dubinu, od grafa koji u sistemu URSA predstavlja formulu, pravi vektor pokazivača na čvorove grafa. U slučaju čvorova koji predstavljaju promenljive, pamti se i njihov identifikator. Zatim se, prolazeći kroz ovaj vektor unazad, određuju identifikatori preostalih čvorova koji će se koristiti kao ulazi, odnosno izlazi (`UIi`). Najveću vrednost identifikatora će imati baš koren grafa,

što i odgovara EDIMACS reprezentaciji. Nakon toga se na osnovu istog vekora i stabla prave elementi EDIMACS formata i unose u rešavač. Svaki čvor grafa, koji nije promenljiva, je predstavljen jednim redom u EDIMACS formatu. Na osnovu tipa čvora i navedene tabele se određuje prvi broj, nakon koga se uvek navodi -1. Potom se navodi identifikator čvora koji predstavlja izlaz (`UIO`), pa zatim niz identifikatora koji predstavljaju ulaze. Ovi identifikatori su zapravo identifikatori dodeljeni svoj deci trenutnog čvora, pa ih je na osnovu grafa koji predstavlja formulu jednostavno naći. Kada se završi unos, rešavač pravi *hpgraph* i *vpgraph*.

Nakon ove obrade, rešavač je spreman za rad. Kao što je slučaj sa rešavačem NoClause, ni rešavač NflSAT nema mogućnost vraćanja svih modela formule, pa je to i u ovom slučaju implementirano koristeći blokirajuće klauze. Nakon što rešavač nađe neko rešenje, koje se posebno pamti, moraju se ponovo izgraditi grafovi *hpgraph* i *vpgraph* jer se tokom rada rešavača oni menjaju. Oni se izgrađuju na isti način na koji su se izgradili prvi put (pomoću funkcija za rad sa EDIMACS formatom) ali se još uz početne unose dodaju oni koji predstavljaju blokirajuće klauze. Jedna blokirajuća klauza je predstavljena jednim unosom koji predstavlja $\vee$ čvor, nakon koga sledi niz identifikatora promenljivih, čije su vrednosti negativne u slučaju da promenljiva u klauzi treba biti negirana.

Glava 5

Eksperimentalni rezultati

U ovoj glavi su prikazani rezultati eksperimenata u kojima se ispituje efikasnost rešavača NoClause i NflSAT. Njihov učinak se poredi sa učinkom KNF rešavača Clasp[1] i ArgoSAT[2]. Za eksperimente su korišćeni sledeći problemi: 3-SAT problem, problem n dama kao i problem nalaženje dužine optimalnog Golombovog lenjira.

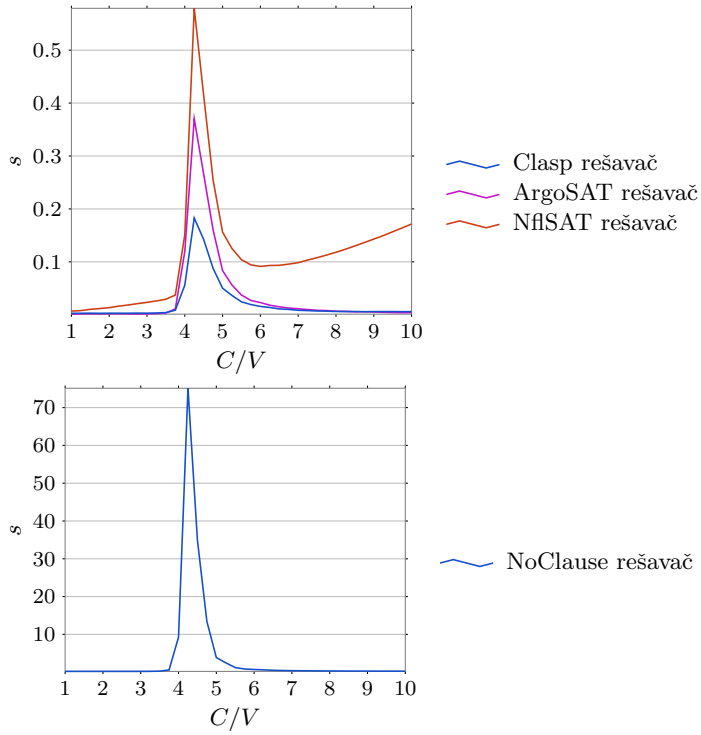
5.1 3-SAT problem

3-SAT problem se sastoji u ispitivanju zadovoljivosti logičke formule koja je data u KNF i za koju važi da je svaka njena klauza dužine 3. Težina rešavanja ovih problema zavisi od odnosa broja klauza C i broj promenljivih V u formuli [12]. Formule za koje je najteže ispitati da li su zadovoljive ili ne su one kod kojih ima približno $4,25$ puta više klauza nego promenljivih. Eksperiment je sproveden nad slučajno generisanim 3-SAT formulama sa brojem promenljivih 200, 250 i 300. Generisano je po 100 formula za svaki odnos broja klauza i broja promenljivih u rasponu od 1 do 10 sa korakom $0,25$. Cilj eksperimenta je provera da li su ispitivani ne-KNF rešavači sporiji na KNF primerima od KNF rešavača (što je očekivano i opravdava dalje eksperimente nad ne-KNF instancama).

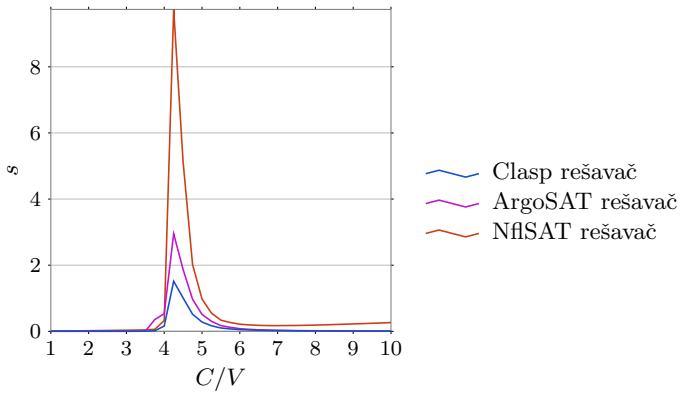
Bitna napomena u vezi sa ovim eksperimentom jeste da su KNF rešavači u prednosti u odnosu na ne-KNF rešavače, zbog toga što ne-KNF rešavači koriste strukturne informacije formule kojih nema u KNF. Na slici 5.1 je prikazano potrebno vreme za ispitivanje zadovoljivosti formula od 200 promenljivih u zavisnosti od odnosa broja klauza i broja promenljivih. Kao što je i očekivano, vreme rešavanja raste oko odnosa $4,25$, ali bez obzira na to, rešavači Clasp, ArgoSAT i NflSAT rešavaju formule od 200 promenljivih za manje od pola sekunde. Kod rešavača NoClause situacija je drugačija. Formule čiji je odnos

[1] `http://www.cs.uni-postdam.de/clasp/` Autori: Martin Gebser, Benjamin Kaufmann, André Neumann i Torsten Schaub.

[2] `http:://argo.matf.bg.ac.rs/?content=research` Autor: Filip Marić



Slika 5.1: Grafik vremena rešavanja u zavisnosti od odnosa broja klauza i promenljivih za 3-SAT formule veličine 200 promenljivih rešavača Clasp, ArgoSAT i NflSAT (gore) i rešavača NoClause (dole).



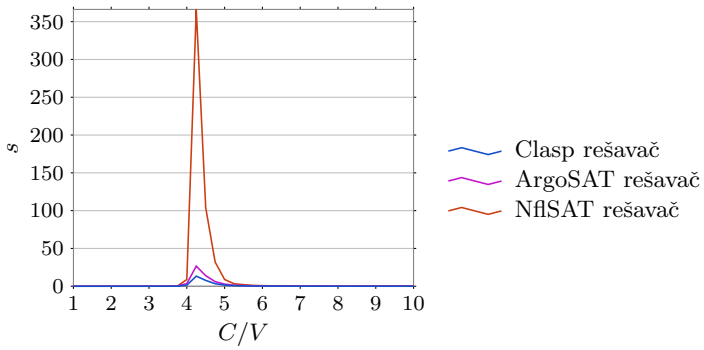
Slika 5.2: Grafik vremena rešavanja u zavisnosti od odnosa broja klauza i promenljivih za 3-SAT formule veličine 250 promenljivih rešavača Clasp, ArgoSAT i NflSAT.

broja klauza i broja promenljvih oko $4,25$ bitno utiču na rad rešavača, pa je u slučaju formula sa 200 promenljivih NoClause rešavaču bilo potrebno u proseku oko 70 sekundi da ispita njihovu zadovoljivost. Na primerima formula sa 250 promenljivih, rešavaču NoClause je trebalo i preko dva sata da ispita njihovu zadovoljivost, pa nije bilo svrhe ispitivati njegov rad na formulama sa većim brojem promenljivih.

Kada su u pitanju formule sa 250 promenljivih, ponašanje rešavača Clasp, ArgoSAT i NflSAT se ne menja bitno. Ipak, primećuje se da je u proseku potrebno nešto više vremena za ispitivanje zadovoljivosti formula čiji je odnos broja klauza i broja promenljivih oko $4,25$. Takođe se vidi da je NflSAT rešavaču potrebno više vremena za ispitivanje zadovoljivosti ovih formula, nego što je potrebno rešavačima Clasp i ArgoSAT.

U slučaju kada formule imaju 300 promenljivih, razlike u vremenu rešavanja kod KNF i ne-KNF rešavača postaju veće. Rešavačima Clasp i ArgoSAT je potrebno u prosečno 20 do 25 sekundi da ispitaju zadovoljivost formula čiji je odnos broja klauza i broja promenljivih oko $4,25$, što je primetan skok u odnosu na formule sa 250 promenljivih. Međutim, povećanje broja promenljivih mnogo značajnije utiče na učinak NflSAT rešavača, jer je njemu za ispitivanje zadovoljivosti tih istih formule prosečno potrebno oko 350 sekundi.

Dobijeni rezultati ovog eksperimenta pokazuju da su ispitivani ne-KNF rešavači sporiji na 3-SAT primerima od KNF rešavača. Ovakav ishod je očekivan, zbog toga što iz formula u KNF, ne-KNF rešavači ne mogu izvući strukturne informacije koje koriste za efikasnije rešavanje instanci.



Slika 5.3: Grafik vremena rešavanja u zavisnosti od odnosa broja klauza i promenljivih za 3-SAT formule veličine 300 promenljivih rešavača Clasp, ArgoSAT i NflSAT.

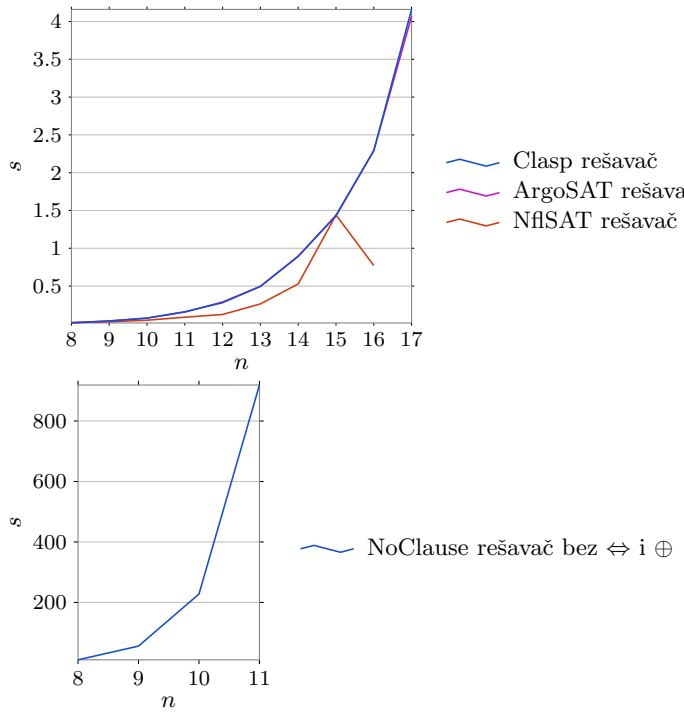
5.2 Problem n dama

Problem n dama je definisan na sledeći način. Cilj je rasporediti n šahovskih dama na šahovskoj tabli dimenzije $n \times n$ ali tako da se dame međusobno ne napadaju (u skladu sa standardnim šahovskim pravilima). Kako svaki red na tabli može imati samo jednu damu, problem predstavlja određivanje kolone, za svaki red, na koje se može smestiti dama. Eksperiment je rađen na problemima dimenzija od $n = 8$ do $n = 17$. Prvi deo eksperimenta se sastojao u nalaženju samo jednog rešenja problema, dok su se u drugom delu tražila sva rešenja. Za ukupno vreme rešavanja kod ne-KNF rešavače (Clasp i ArgoSAT) uzeto je vreme potrebno da se početna formula pretvori u KNF kao i vreme potrebno za ispitivanje zadovoljivosti KNF. Cilj eksperimenta je da uporedi efikasnost ne-KNF i KNF rešavača nad zadovoljivim ne-KNF instancama.

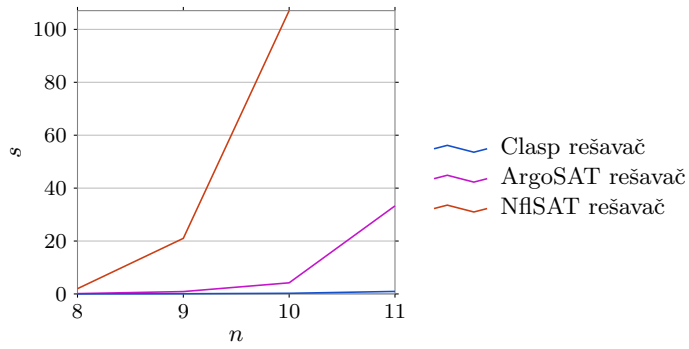
Na slici 5.4 se vidi da rešavač NflSAT uspešno (brže nego KNF rešavači Clasp i ArgoSAT) rešava probleme koji su veličine do $n = 16$. Međutim, već za problem veličine $n = 17$ vreme rešavanja NflSAT rešavača se drastično poveća (preko sat vremena), dok rešavači Clasp i ArgoSAT rešavaju isti problem za nekoliko sekundi. Rešavač NoClause nije uspeo ispravno da reši nijedan od ovih problema kada je koristio veznike $\Leftrightarrow$ i $\oplus$. Kada je NoClause pokrenut da reši iste probleme bez korišćenja navedenih veznika, on ih je rešio ali mnogo sporije u odnosu na ostale rešavače, zbog toga što je eliminisanjem veznika $\Leftrightarrow$ i $\oplus$ narušena početna struktura problema, na koju se ovaj rešavač oslanja.

Slika 5.5 prikazuje vreme potrebno za pronalaženje svih rešenja problema n dama. Za rešavač NflSAT se primećuje da već probleme manjih dimenzija rešava znatno duže nego što ih rešavaju rešavači Clasp i ArgoSAT. Ipak, ovakav rezultat je i očekivan, zbog toga što NflSAT rešavač originalno nije dizajniran da nalazi sva rešenja problema. NoClause rešavač nije testiran za nalaženje svih rešenja problema n dama, jer je bio i neefikasan za nalaženje prvog rešenja.

Rezultati dobijeni ovim eksperimentom pokazuju da je rešavač NoClause



Slika 5.4: Grafik zavisnosti vremena rešavanja problema n dama u odnosu na veličinu problema. (Prvo rešenje)



Slika 5.5: Grafik zavisnosti vremena rešavanja problema n dama u odnosu na veličinu problema. (Sva rešenja)

znatno sporiji u odnosu na preostale ispitivane rešavače. Ovakav njegov učinak je očekivana posledica toga što on veznike $\Leftrightarrow$ i $\oplus$ ne koristi. Rešavač NflSAT se pokazao bolje, jer je za jedan broj problema nalazio prvo rešenje brže od rešavača Clasp i ArgoSAT, što ukazuje da informacije iz strukture ne-KNF formule mogu

```
nDim = 8;
bHorizontal = true;
for(ni=0; ni<nDim; ni++)
  bHorizontal &= ((n[ni] & n[ni]-1)==0) & (n[ni]!=0);
nVertical = 0;
for(ni=0; ni<nDim; ni++)
  nVertical |= n[ni];
bVertical = (nVertical+1 == 0);
bDiagonal = true;
for(nAi=0; nAi<nDim-1; nAi++)
  for(nAj=0; nAj<nDim; nAj++)
    for(nBi=nAi+1; nBi<nDim; nBi++)
      for(nBj=0; nBj<nDim; nBj++)
        if (nBi-nAi==nBj-nAj | nBi-nAi==nAj-nBj)
          bDiagonal &= (((n[nBj]<<(nBi-nAi)) & n[nAj])==0);

assert_all(bHorizontal & bVertical & bDiagonal);
```

Slika 5.6: Problem n dama predstavljen URSA kodom

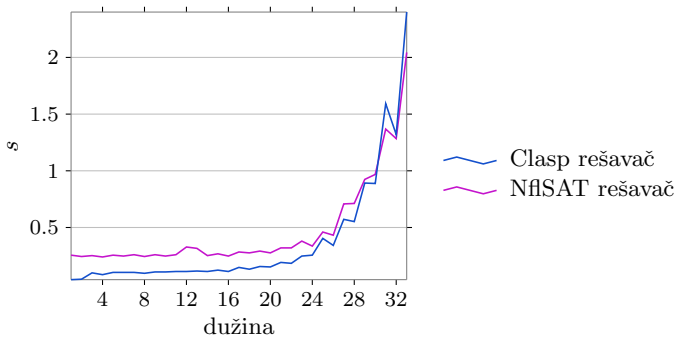
doprineti bržem radu rešavača.

5.3 Golombov lenjir

Golombov lenjir [3] je skup oznaka koje su izabrane sa celobrojnih pozicija na zamišljenom lenjiru, ali tako da ne postoje dva para oznaka kod kojih je međusobna udaljenost oznaka jednaka. Razlika najveće i najmanje oznake na lenjiru predstavlja njegovu *dužinu*, dok broj oznaka predstavlja njegov *red*. Golombovi lenjiri za koje važi da imaju najmanju moguću dužinu za svoj red se nazivaju *optimalnim*. Problem nalaženje optimalnog Golombovog lenjira se smatra NP-teškim, iako to nije dokazano. Najbolji poznati algoritam za nalaženje optimalnog Golombovog lenjira zadatog reda se zasniva na ispitivanju da li je za trenutnu dužinu i zadati red lenjir Golombov, počev od najmanje dužine lenjira.

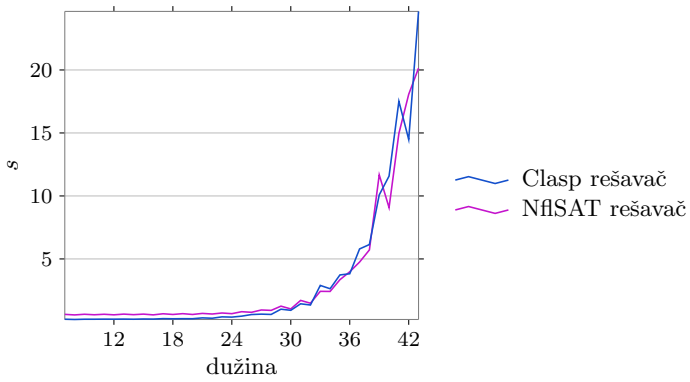
Eksperiment je vršen nad problemima nalaženja dužine optimalnih Golombovih lenjira čiji su redovi 8, 9 i 10. Poznato je da optimalani Golombovi lenjiri navedenih redova, imaju redom dužine 34, 44 i 55. Za sve dužine manje od optimalne, za dati red, beležena su vremena potrebna da rešavači utvrde da lenjir nije Golombov (tim tvrđenjima odgovaraju nezadovoljive iskazne formule). Svrha ovog eksperimenta jeste poređenje efikasnosti ne-KNF i KNF rešavača nad nezadovoljivim ne-KNF formulama.

Na grafiku (slika 5.7) je prikazana zavisnost vremena ispitivanja da li je lenjir Golombov u odnosu na njegovu dužinu. Red lenjira je fiksiran i iznosi 8. Rešavači Clasp i NflSAT su za približno vreme uspeli da pokažu da lenjiri



Slika 5.7: Vreme ispitivanja da li postoji Golombov lenjir reda 8 za datu dužinu.

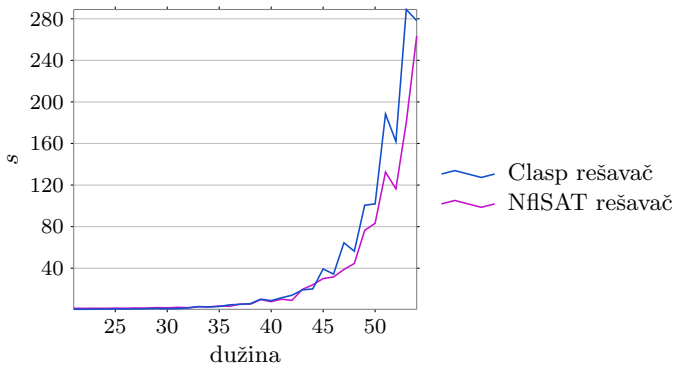
sa navedenim dužinama nisu Golombovi. Vreme ispitivanja počinje da raste za dužine preko 25, ali ne prelazi 3 sekunde. Rešavač NoClause nije uspeo da reši ovaj problem ispravno koristeći veznike $\Leftrightarrow$ i $\oplus$. U slučaju kada je pokrenut da radi bez tih veznika, vreme izvršavanja je bilo preveliko, pa nije bilo svrhe beležiti njegov učinak.



Slika 5.8: Vreme ispitivanja da li postoji Golombov lenjir reda 9 za datu dužinu.

Grafik na slici 5.8 prikazuje zavisnost vremena ispitivanja da li je lenjir Golombov u odnosu na dužinu lenjira reda 9. Kao i kod lenjira reda 8, učinak rešavača Clasp i NflSAT je približno isti. Međutim, primetno je da za veće dužine lenjira rešavač NflSAT brže pokazuje da oni ne mogu biti Golombovi nego što to čini rešavač Clasp. Slika 5.9 prikazuje sličan grafik kao i slika 5.8, samo što je u pitanju lenjir reda 10. Na njemu se još jasnije vidi da je rešavač NflSAT za veće dužine dao bolje rezultate u odnosu na rešavač Clasp.

Dobijeni rezultati jasno pokazuju da postoje ne-KNF instance na kojima ne-KNF rešavači mogu biti uspešniji nego KNF rešavači. Kako će se ne-KNF rešavači u budućnosti tek razvijati i unapređivati, verovatno je da će instance koje oni budu efikasno rešavali biti brojnije i raznovrsnije.



Slika 5.9: Vreme ispitivanja da li postoji Golombov lenjir reda 10 za datu dužinu.

```
nM=8;
minimize(nL,1,100);
nA[0]=0;
nA[nM-1]=nL;
bDomain=true;
for(ni=0;ni+1<nM;ni++)
  bDomain = bDomain & (nA[ni] < nA[ni+1]);
bDistinctDiff=true;
for(ni=0;ni+1<nM;ni++)
  for(nj=ni+1;nj<nM;nj++)
    for(nk=0;nk+1<nM;nk++)
      for(nl=nk+1;nl<nM;nl++)
        if(ni!=nk)
           bDistinctDiff = bDistinctDiff &
                           (nA[ni]-nA[nj] != nA[nk]-nA[nl]);
assert(bDomain & bDistinctDiff);
```

Slika 5.10: Problem nalaženja Golombovog lenjira reda 8 sa najmanjom dužinom predstavljen URSA kodom.

Glava 6

Zaključci i dalji rad

U ovom radu su predstavljena dva ne-KNF rešavača, NoClause i NflSAT, koji su integrisani sa sistemom URSA. Pored toga, analizirana je efikasnost ovih rešavača pri rešavanju različitih problema sa ciljem da se vidi da li se oni mogu praktično primeniti.

Rezultati su pokazali da ne-KNF rešavači značajno zaostaju za KNF rešavačima kada se rešavaju KNF formule, što je i bilo očekivano. Ipak, rešavač NflSAT, koji se pokazao kao uspešniji ne-KNF rešavač od dva razmatrana, je uspeo da se za određen broj ne-KNF instanci nosi sa uspešnim savremenim KNF rešavačima. Ove instance su dobijene od kombinatornih zadovoljivih i nezadovoljivih problema, te ovi rezultati, zajedno sa činjenicom da će se ne-KNF rešavači u budućnosti unapređivati, ukazuju na to da će ne-KNF rešavači moći uspešno da zamene KNF rešavače na širem spektru problema. Dodatno, ne-KNF rešavači još nisu dostigli zrelost koju imaju KNF rešavači, tako da postoje i greške u implementacijama, ali u vremenu koje dolazi, ovi rešavači će verovatno zauzeti značajno mesto u rešavanju problema iskazne zadovoljivosti ali i u rešavanju raznovrsnih problema koji se mogu svesti na problem iskazne zadovoljivosti.

Dalji rad, pored integrisanja novih ne-KNF rešavača u sistem URSA, bi mogao biti i razvoj sistema za automatsko biranje ne-KNF rešavača za zadatu ulaznu instancu. Birao bi se rešavač za koji se očekuje da će najbrže rešiti zadatu instancu. Način biranja bi se mogao zasnovati na analiziranju vremena rada raspoloživih ne-KNF rešavača na različitim instancama. Takođe je moguć i razvoj novog ne-KNF rešavača.

Bibliografija

[1] Bacchus Fahiem, Walsh Toby - A non-CNF DIMACS style, dostupno na http://www.satcompetition.org/2005/edimacs.pdf

[2] Cook Stephen A. - The Complexity of Theorem-Proving Procedures, Proceedings of the Third Annual ACM Symposium on Theory of Computing, pp. 151-158 (1971)

[3] Dimitrimanolakis Apostolos - Analysis of the Golomb Ruler and the Sidon Set Problems, and Determination of Large, Near-Optimal Golomb Rulers, Diploma Thesis, Technical University of Crete (2002) http://www.cs.utoronto.ca/~apostol/golomb/main.pdf

[4] Gomes Carla P., Kautz Henry, Sabharwal Ashish, Selman Bart - Satisfiability Solvers, Handbook of Knowledge Representation, Elsevier B.V., pp. 89-122

[5] Gomes Carla P., Shmoys David - Completing Quasigroups or Latin Squares: A Structured Graph Coloring Problem, in Proc. Computational Symposium on Graph Coloring and Generalizations (2002)

[6] Jain Himanshu, Clarke Edmund M. - Efficient SAT Solving for Non-Clausal Formulas Using DPLL, Graphs, and Watched Cuts. Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE, pp. 563-568

[7] Janičić Predrag - Matematička logika u računarstvu, Matematički fakultet Beograd, četvrto, elektronsko izdanje (2008) http://www.matf.bg.ac.rs/~janicic/mlr.zip

[8] Janičić Predrag - Uniform Reduction to SAT (2010) http://arxiv.org/abs/1012.1255

[9] Levin Leonid A. - Universal Sequential Search Problems, Problems of Information Transmission, 1973, Vol.9, Iss 3, pp. 256-266 (1973)

[10] Marić Filip, Janičić Predrag - Formalization of abstract state transition systems for SAT,In Logical Methods in Computer Science Vol. 7 (3:19) 2011, pp. 1-37

[11] Marques-Silva J. - Practical Applicationsof Boolean Satisfiability, International Workshop on Discrete Event Systems, WODES 2008, pp. 74-80

[12] Mitchell David, Selman Bart, Levesque Hector - Hard and Easy Distributions of SAT Problems. In Proceedings of the Tenth National Conference on Artificial Intelligence, pp. 459-465 (1992)

[13] Moskewicz Matthew W, Madigan Conor F, Zhao Ying, Zhang Lintao, Malik Sharad - Chaff: Engineering an Efficient SAT Solver, Annual ACM IEEE design automation conference, pp. 530-535 (2001)

[14] Nikolić Mladen - Metodologija izbora pogodnih vrednosti parametara SAT rešavača, Magistarska teza, Matematički Fakultet, Univerzitet u Beogradu, 2008.

[15] Thiffault Christian, Bacchus Fahiem, Walsh Toby - Solving Non-clausal Formulas with DPLL Search, Principles and Practice of Constraint Programming - CP 2004, Springer Berlin / Heidelberg pp. 663-678

[16] Tseitin G. - On the complexity of proofs in propositional logics. In Automation of Reasoning: Classical Papers in Comp. Logic 1967-1970, vol.2, 1983