Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

SAT Rešavači

Seminarski rad iz Matematičke Logike

Radili:
Đorđe Stanković IN 13/2018
Mihajlo Živković IN 16/2018
Igor Jakovljević IN 24/2018

Profesor:
prof. Dr. Silvia Ghilezan

Jun 2020.

Sadržaj

Sažetak

U ovom seminarskom radu daćemo kratak pregled SAT rešavača. Objasnićemo SAT problem (Boolean Satisfiability Problem) i procedure za njegovo rešavanje na kojoj su zasnovani svi moderni SAT rešavači.

1. Predgovor

U poslednjih nekoliko godina susrećemo se sa ogromnim napretkom u oblasti performansi SAT rešavača. Bez obzira na njihovo eksponencijalno $O(2^n)$ vreme izvršavanja, SAT rešavači se sve više koriste u raznim oblastima rada, kao što su verifikacija softvera i hardvera, planiranje, veštačka inteligencija, pa čak i zahtevni algebarski problemi. Postojeća SAT takmičenja su dovela do stvaranja veoma naprednih implementacija postojećih SAT rešavača. Moderni SAT rešavači pružaju rešenje za izuzetno kompleksne probleme, koji mogu sadržati desetine hiljada varijabli i preko milion klauza. Danas se SAT rešavači smatraju kao jedni od najbitnijih elemenata EDA okruženja, koja predstavljaju kategorije softverskih alata za dizajniranje električnih sistema i integrisanih kola.

S obzirom na to da ima široke primene, posvećuje se velika pažnja algoritmima koji bi što efikasnije rešavali instance ovog problema u praksi. Kada kažemo rešavanje problema mislimo na ispitivanje SAT zadovoljivosti date iskazne formule.

1.1 SAT Problem

U logici i računarskim naukama, SAT (Boolean Satisfiability Problem) jeste problem određivanja postojanja interpretacije koja zadovoljava datu formulu. Drugim rečima, postavlja pitanje da li promenljive date formule mogu biti zamenjene vrednostima TAČNO ili NETAČNO tako da cela formula bude tačna. Ukoliko je ovo slučaj, formula je zadovoljiva. Sa druge strane, ukoliko ovakve zamene ne možemo napraviti, formula je nezadovoljiva. SAT je prvi problem za koji je dokazano da je NP-kompletan (tačnije 3-SAT problem, dok malo više ograničena verzija, 2-SAT problem, pripada klasi P). U NP klasu spadaju svi problemi odlučivanja koji mogu biti rešeni nedeterminističkim algoritmom za polinomijalno vreme. Ovo takođe znači da svi problemi koji su u klasi NP kompleksnosti ne mogu biti teži za rešiti od samog SAT.

1.2 Vrste SAT Rešavača

SAT rešavači se mogu svrstati u dve grupe - potpune i stohastičke. Potpuni SAT rešavači za datu iskaznu formulu sigurno nalaze zadovoljavajuću valuaciju ili utvrđuju da ona ne postoji. Sa druge strane, stohastički rešavači ne mogu dokazati nezadovoljivost formule, dok mogu pokazati njenu zadovoljivost. U određenim situacijama, stohastički rešavači mogu brže da pokažu zadovoljivost formule od potpunih, što je njihova prednost. Međutim, na određenim problemima, oni su neupotrebljivi.

Moderni potpuni SAT rešavači su najčešće zasnovani na DPLL proceduri, koja je nastala ranih šezdesetih godina. Više informacija o DPLL proceduri se nalazi u trećem poglavlju.

2. Pojmovi i Definicije

Iskazna logika se bavi istinitošću iskaza, koji su izgrađeni od iskaznih promenljivih pomoću iskaznih veznika. Iskazne promenljive predstavljaju elementarne iskaze. Način izgradnje složenijih iskaza je definisan sintaksom iskazne logike, dok istinitosnu vrednost ispravnih iskaza opisuje semantika iskazne logike. Kao što smo već spomenuli, osnovni problem iskazne logike (SAT) jeste problem ispitivanja da li je iskazna formula zadovoljiva, tj. da li postoji dodela vrednosti promenljivama za koju je iskazna formula tačna.

U tekstu će se koristiti naredne definicije osnovnih pojmova iskazne logike.

Definicija 1 : Skup iskaznih formula Θ nad prebrojivim skupom iskaznih slova P je skup za koji važi:

- Iskazna slova iz skupa P i logičke konstante $\top$ i $\bot$ su iskazne formule;
- Ako su A i B iskazne formule, onda su i ($\neg$A), (A $\wedge$ B), (A $\vee$ B), (A $\Rightarrow$ B) i (A $\Leftrightarrow$ B) iskazne formule.

Iskazna slova nazivamo *iskaznim promenljivim*, a iskazna slova i logičke konstante nazivamo *atomičkim iskaznim formulama*. Atomičke iskazne formule i njihove negacije nazivamo *literalima*. Disjunkcije literala nazivamo *klauzama*.

Ako je *p* iskazno slovo i l literal nad tim iskaznim slovom, $\bar{l}$ označava literal $\neg p$ ukoliko je $l = p$, a literal p ukoliko je $l = \neg p$.

Definicija 2 : Funkcije koje preslikavaju skup iskaznih slova u skup {0, 1} se nazivaju valuacijama. Polazeći od valuacije $\mathcal{V}$ konstruiše se funkcija I_v koja preslikava skup iskaznih formula u skup {0, 1}. Nju nazivamo funkcijom istinitosne vrednosti za valuaciju $\mathcal{V}$ i definišemo je na sledeći način:

- $I_v(p) = v(p), \forall p \in P$
- $I_v(\top) = 1$
- $I_v(\bot) = 0$
- $I_v(\neg A) = 1$, ako je $I_v(A) = 0$
- $I_v(\neg A) = 0$, ako je $I_v(A) = 1$
- $I_v(A \wedge B) = 1$, ako je $I_v(A) = 1$ i $I_v(B) = 1$, inače $I_v(A \wedge B) = 0$
- $I_v(A \vee B) = 0$, ako je $I_v(A) = 0$ i $I_v(B) = 0$, inače $I_v(A \vee B) = 1$

Definicija 3 : Iskazna formula *F* je zadovoljiva ako postoji valuacija $\mathcal{V}$ takva da je $I_v(F) = 1$. Takva valuacija se naziva modelom formule *F*, što se zapisuje $v \models F$. U suprotnom, *F* je nezadovoljiva. Iskazna formula *F* je valjana (tautologija) ako važi $\forall v, I_v(F) = 1$. U suprotnom, *F* je poreciva. Formula *F* je logička posledica skupa formula Γ, što se zapisuje $\Gamma \models F$, ukoliko je svaka valuacija koja je model za sve formule iz skupa Γ, takođe model i za formulu *F*.

Definicija 4 : Iskazna formula je u konjunktivnoj normalnoj formi *(KNF)* ako je oblika

$$A_1 \wedge A_2 \wedge ... \wedge A_n,$$

pri čemu je svaka od formula $A_i \quad 1 \leq i \leq n$ klauza.

Definicija 5 : Supstitucija (ili zamena) formule C formulom D u formuli A je funkcija $A[C \mapsto D] : \Theta \times \Theta \times \Theta \mapsto \Theta$ definisana na sledeći način:

- Ako je $A = C$, onda je $A[C \mapsto D] = D$
- Ako $A \neq C$ i A je iskazno slovo, onda je $A[C \mapsto D] = A$
- Ako $A \neq C$ i važi $A = (\neg B)$ za neku iskaznu formulu B, onda je $A[C \mapsto D] = \neg(B[C \mapsto D])$
- Ako $A \neq C$ i važi $A = B_1 \wedge B_2$, odnosno $A = B_1 \vee B_2$, za neke iskazne formule B_1 i B_2, tada je $A[C \mapsto D]$ jednako $B_1[C \mapsto D]$ i $B_2[C \mapsto D]$, odnosno $B_1[C \mapsto D]$ ili $B_2[C \mapsto D]$.

Definicija 6 : Ako je A iskazna formula, a *l* literal, onda

- $A[[l \mapsto \top]]$ označava formulu $A[l \mapsto \top][\bar{l} \mapsto \bot]$ iz koje su uklonjene sve klauze koje sadrže konstantu $\top$ i sva pojavljivanja konstante $\bot$;
- $A[[l \mapsto \bot]]$ označava formulu $A[[\bar{l} \mapsto \top]]$.

3. SAT Rešavači

U ovoj glavi ćemo opisati dva najbitnija algoritma koja predstavljaju osnove za rad modernih potpunih SAT rešavača. Potpun SAT rešavač je onaj koji ili određuje zadovoljivost date formule, ili dokazuje da je ona nezadovoljiva.

3.1 DPLL Algoritam

Osnovni algoritam za proveru zadovoljivosti iskaznih formula je Dejvis-Patnam-Logman-Lovelandova procedura (DPLL). Iako je publiciran tek 1962. kao usavršena verzija Dejvis-Patnamovog algoritma (usavršeno po tome što troši manje memorije za rad), on i posle skoro 60 godina predstavlja osnovu za formiranje najefikasnijih kompletnih SAT rešavača, poput Chaff, GRASP i MiniSAT rešavača.

Formula čija se zadovoljivost ispituje mora biti u konjunktivno normalnoj formi (KNF). Formula se predstavlja kao skup klauzi, a klauza kao skup literala. Literal je čist (pure) ako se on pojavljuje u formuli, ali ne i njegov komplement (negacija) - odatle je funkcija pureLiteral(l, F) sa povratnom vrednošću TAČNO ako se samo literal l pojavljuje u F, a ne i $\bar{l}$, u suprotnom NETAČNO.

DPLL algoritam radi po principu backtracking procedure, odnosno rekurzivnog menjanja istinitosne vrednosti literala u celoj formuli, uproštavajući je, i proveravajući da li je ona zadovoljiva. Ako je uproštena formula zadovoljiva, onda je i početna, ako ne, rekurzivno se vraća na prethodnu dodeljenu istinitosnu vrednost i menja je - ako ni to ne uspe vraća se na menjanje prethodnog literala, i tako dalje dok ne ispita celo stablo mogućnosti. Ova tehnika je poznata pod nazivom “pravilo podele”, gde se početni problem deli na dva jednostavnija potproblema, gde se drugi neće izvršiti ako preko prvog algoritam dokaže zadovoljivost formule.

Ako se DPLL algoritam primeni nad negacijom formule F, onda će formula F biti tautologija ako je negacija formule F zadovoljiva, odnosno neće biti tautologija ako je negacija nezadovoljiva. Ovo takođe dokazuje i valjanost iste formule, jer je tautologija uži pojam.

Ovde je prikazan pseudokod rada DPLL algoritma na rekurzivan, neiterativan način. Povratna vrednost funkcije je Bulova vrednost TAČNO ili NETAČNO, što označava da li je početna KNF formula SAT zadovoljiva ili ne, odnosno da li je tautologija (valjana) ili ne drugom primenom algoritma, kada se u njega prosledi negacija početne formule.

F - formula u KNF formatu čiju zadovoljivost ispitujemo;

l - literal;

C - klauza;

$\varnothing$ - prazan skup;

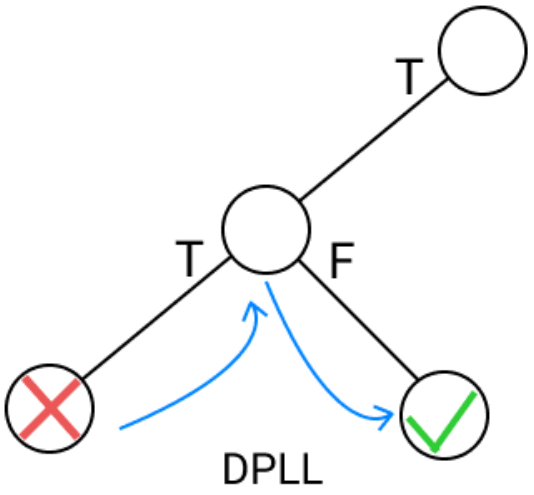
$F[[l \mapsto \top]]$ - novi parametar za rekurzivno pozvan DPLL algoritam, gde će se u trenutnoj formuli literal l zameniti sa istinitosnom vrednošću TAČNO;

$pureLiteral(l, F)$ - funkcija koja gleda da li se u trenutnoj formuli pojavljuje literal l, a ne njegov komplement (negacija) $\bar{l}$, takođe ima Bulovu povratnu vrednost.

```
function dpll (F: KNF formula) : bool
begin
    if F = ∅ then return true
    else if ∅ ∈ F then return false
    else if {l} ∈ F then return dpll(F[[l → T]])
    else if pureLiteral(l,F) then return dpll(F[[l → T]])
    else begin
        select l such that l ∈ C and C ∈ F
        if dpll(F[[l → T]]) = true then return true
        else return dpll(F[[l → ⊥]])
    end
end
```

Iz ovog pseudokoda se mogu zaključiti par pravila po kojem DPLL algoritam putuje kroz svoje stablo:

- Prazna formula je zadovoljiva;
- Prazna klauza označava da je formula nezadovoljiva;
- Literali sa vrednošću NETAČNO mogu da se izbrišu iz klauze, jer su u disjunkciji (ovo ne briše celu klauzu);
- Jedinična klauza (klauza sa jednim literalom) uvek ima vrednost TAČNO;
- Ako klauza sadrži literal vrednosti TAČNO, ili ako sadrži literal i njegov komplement (negaciju), izbriši tu klauzu iz konjunktivne forme;
- Ako svaki rekurzivni poziv DPLL-a vrati NETAČNO za svaku granu, onda je formula SAT nezadovoljiva.



Efikasnost DPLL algoritma dosta zavisi od izbora literala grananja, i u odnosu na to, vreme izvršavanja može biti konstantno ili eksponencijalno. Takvo grananje je heurističko u prirodi.

Ta efikasnost se tokom godina poboljšavala na par načina, kao na primer definisanjem specifičnih pravila za biranje literala grananja u funkciji, definisanje nove strukture podataka za rad sa DPLL algoritmom u programima, i pravljenje nove backtracking procedure, koja je ne-hronološka (backjumping, ne ide uvek jedan korak unazad) i koja koristi učenje klauza za brži rad u budućnosti, odnosno daljem rešavanju algoritma za veće klauze i formule.

Učenje klauzi je jako korisno jer će algoritam na bolji način uočiti kada će doći do konflikta ponovo sa izabranim istinitosnim vrednostima (konflikt je ćorsokak u rešavanju, odnosno kada DPLL vrati NETAČNO u jednoj grani i momenat kada se započinje backtracking). SAT rešavači koji koriste ovaj tip učenja se nazivaju **Conflict-Driven Clause Learning** rešavači, i oni su vrlo nova pojava u ovoj sferi nauke.

3.2 CDCL Algoritam

DPLL algoritam ima 3 nedostatka:

1. Pravi "naivne" odluke;
2. Kada naiđe na konflikt ne nauči ništa iz tog konflikta osim činjenice da je parcijalna valuacija dovela do njega;
3. Backtracking može da se radi samo jedan nivo unazad, što može da dovede do ponovnog istraživanja prostora pretrage koji je osuđen na propast.

Sa **CDCL** algoritmom su poboljšana sva 3 aspekta, a to je postignuto implementacijom sledećih pravila:

Učenje

Mehanizam učenja (eng. *clause learning*) omogućava SAT rešavačima da tokom procesa rešavanja prošire skup klauzi, koje sačinjavaju formulu, (redundantnim) klauzama koje su posledice formule F.

Učenje se opisuje narednim pravilom:

$$\frac{F \models c, \quad c \subseteq L}{F := F@c}$$

Uslov $F \models c$ zahteva da je klauza *c* logička posledica formule *F* i on obezbeđuje da je nova formula logički ekvivalentna sa polaznom formulom *F*. Uslov $c \subseteq L$ zahteva da su svi literali u klauzi koja se uči sadžani u skupu literala *L* i time obezbeđuje da se pravilom učenja ne uvode literali koji se ne nalaze u skupu *L*. Ovaj uslov je takođe bitan kako bi se obezbedilo zaustavljanje procedure. Pravilo ne specifikuje način na koji se klauza *c* određuje. Najčešće je slučaj da se uče klauze koje su određene kao rezultat analize konflikta.

Zaboravljanje

Tokom rada rešavača sa učenjem broj klauza tekuće formule raste i formula se uvećava. Ukoliko formula postane jako velika , ispitivanje da li je ona u skladu sa tekućom parcijalnom valuacijom i pronalaženje jediničnih klauza postaje jako sporo. Zato je potrebno s vremena na vreme uklanjati neke (redundantne) klauze iz *F*. Neophodne ja da važi da je formula nakon uklanjanja klauza ekvivalentna polaznoj formuli. Rešavači obično ovaj uslov obezbeđuju tako što se uklanjaju isključivo naučene klauze. Klauze početne formule se obično ne uklanjaju. Zaboravljanje klauza se može opisati sledećim pravilom:

$$\frac{F \backslash c \models c}{F := F \backslash c}$$

Uslov $F \backslash c \models c$ obezbeđuje da je klauza koja se uklanja zaista redundantna.

Backjumping

Povratni skokovi predstavljaju vid naprednijeg , tzv. nehronološkog povratka u pretrazi i oni dozvoljavaju rešavačim da ponište nekoliko pretpostavki odjednom, sve do poslednjeg pretpostavljenog literala koji je zaista učestvovao u konfliktu. Ovim se izbegavaju nepotrebne redundantnosti u procesu rešavanja.

Povratni skokovi su obično vođeni klauzom koja se naziva ***klauzom povratnog skoka*** (eng. *backjump clause*) i označavati sa *C*. Ova klauza je semantička posledica formule *F* i poželjno je odgovara promenljivama koje su dovele do konflikta. Kada se konstruiše klauza povratnog skoka, literali koji se nalaze pri vrhu tekuće označene valuacije *M* se uklanjaju, sve dok klauza povratnog skoka ne postane jedinična klauza u odnosu na tekuću valuaciju. Od tog trenutka, njen jedinični litreal se postavlja na vrh valuacije *M*, čime se konflikt razrešava i nakon toga proces pretrage se nastavlja uobičajenim tokom.

Povratni skokovi se realizuju korišćenjem sledećeg pravila:

$$\frac{C = l \vee l_1 \vee ... \vee l_k, F \models C, M = M' | dM'', \bar{l_1}, ..., \bar{l_k} \in M', \bar{l} \in M''}{M := M' l}$$

Pravilo je moguće primeniti kada je klauza povratnog skoka C takva da postoji pretpostavljeni litreal d u M takav da se iznad njega u M nalazi tačno jedan literal $\bar{l}$ suprotan literalu l klauze C , dok se svi ostali literali suprotni literalima klauze C nalaze u M ispod njega. Pravilo poništava literal d i sve literale iznad njega i u valuaciju dodaje literal l. Broj pretpostavljenih literala ispred literala d naziva se ***nivo povratnog skoka*** (eng. *backjump level*). Uslov da je d pretpostavljeni literal je bitan samo kako bi se obezbedilo zaustavljanje.

Primetimo da se klasični povratak (eng. *backtracking*) može smatrati specijalnim slučajem povratnog skoka. U tom slučaju se klauza povratnog skoka gradi od literala koji su suprotni svim pretpostavljenim literalima u valuaciji M. Ta klauza postaje jedinična klauza tačno u trenutku kada se poslednji pretpostavljeni literal ukloni iz M i na osnovu nje se može zaključiti da njemu suprotni mora da bude deo valuacije. Ovako napravljena klauza povratnog skoka , naravno ne oslikava na pravi način konflikt koji se dogodio.

Analiza konflikata

Da bi klauze povratnog skoka istinski oslikavale konflikt koji se razrešava i time dovele do najvećeg mogućeg odsecanja prostora pretrage, one se konstruišu u procesu koji je poznat pod imenom ***analiza konflikata*** (eng. *conflict analysis*).

Proces analize konflikata se nekad opisuje korišćenjem pristupa zasnovanog na grafovima i klauze povratnog skoka se dobijaju obilaskom tzv. grafa zaključivanja (eng. *implication graph*). Proces analize konflikata se nekad opisuje i kao proces rezolucije koji kreće od konfliktne klauze i nastavlja se sa klauzama koje su uzroci propagacije konfliktnih literala.

Postoji nekoliko strategija za analizu konflikata. Njihove varijacije uključuju razne načine izbora klauze koja vodi povratni skok. Ipak, najveći broj strategija za analizu konflikata se zasniva na sledećoj tehnici:

1. Proces analize konflikta započinje sa konfliktnom klauzom (tj. sa klauzom formule F koja je postala netačna u M). Klauza analize konflikata C se u tom trenutku inicijalizuje na konfliktnu klauzu.
2. Svaki literal sadržan u tekućoj klauzi analize konflikata C je netačan u tekućoj valuaciji M i predstavlja ili pretpostavku procedure pretrage ili rezultat neke propagacije. Za svaki propagiran literal l moguće je pronaći klauzu c koja je bila uzrok propagacije literala c. Ove klauze se zovu

razlozi propagacije (eng. *reason clauses*). Propagirani literali iz klauze C se onda zamenjuju (govorićemo *objašnjavaju*) ostalim literalima iz klauza koji predstavljaju razloge njihove propagacije. Ovim se nastavlja proces analize konflikata.

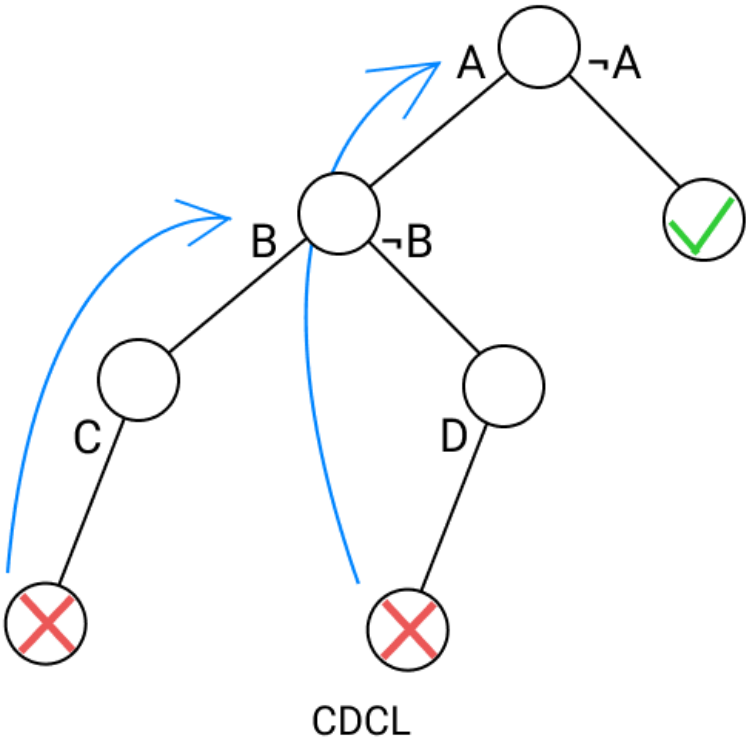
Analiza konflikata se može opisati sa naredna dva pravila:

$$\frac{l_1 \vee ... \vee l_k \in F, \bar{l_1}, ..., \bar{l_k} \in M,}{C := l_1 \vee ... \vee l_k}$$

$$\frac{\bar{l} \in C, \quad l \vee l_1 \vee ... \vee l_k \in F, \quad \bar{l_1}, ..., \bar{l_k} \prec^M l}{C := (C \backslash \bar{l}) \vee l_1 \vee ... \vee l_k}$$

Pravilo konflikta odgovara prvom koraku, a pravilo objašnjavanja odgovara drugom gore opisane procedure. Faza analize konflikta prestaje primenom povratnog skoka. Uslov $\bar{l_1}, ..., \bar{l_k} \prec^M l$ označava da literali $\bar{l_1}, ..., \bar{l_k}$ prethode literalu l u valuaciji M.

Opisana procedura se ponavlja sve dok se ne ispuni neki od uslova za završetak analize kada se klauza analize konflikata proglašava klauzom povratnog skoka. Uslovi završetka moraju biti definisani na način koji obezbeđuje da klauza C zaista zadovoljava uslove potrebne za povratni skok.



Iako je DPLL algoritam publiciran polovinom prošlog veka, CDCL ga je koristio kao inspiraciju i objavljen je 1997. što ga čini vrlo mladim algoritmom u ovoj oblasti. Ovaj algoritam se primenjuje u različitim SAT rešavačima, kao što su MiniSAT, ZChaffSAT, Z3, ManySAT, itd. CDCL algoritam je ojačao SAT rešavače toliko da se oni danas koriste u oblastima veštačke inteligencije, bioinformatike, proveravanje modela softvera i hardvera, kriptografije, itd.

3.3 Stohastični SAT Rešavači

Za razliku od potpunih SAT rešavača, koji ili sigurno nalaze zadovoljivu valuaciju ili utvrđuju da ona ne postoji (zasnovani na DPLL i CDCL algoritmima), stohastični ne mogu dokazati nezadovoljivost formule, već samo njenu zadovoljivost. Sa druge strane, njihova prednost je u tome što utvrđuju zadovoljivost formule dosta brže od potpunih.

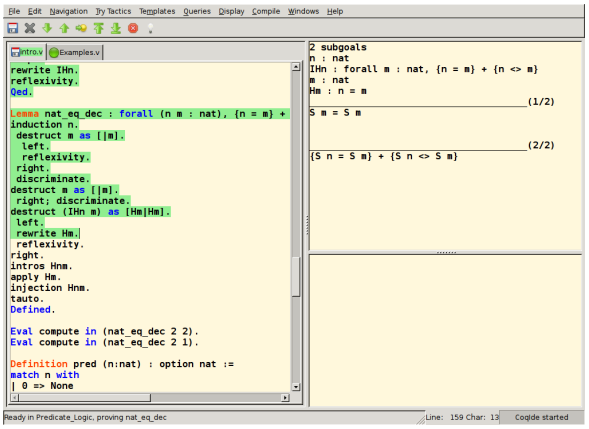
Primeri SAT rešavača koji spadaju u klasu stohastičnih jesu **WalkSAT** i **GSAT**, i oni rade po principu nasumičnog dodeljivanja vrednosti svakom literalu, i nakon određivanja zadovoljivosti takve formule odrede kojim literalima menjaju istinitosnu vrednost, tako da minimizuju broj nezadovoljivih klauzi. Ako slučajno dodeljivanje vrednosti literalima dovede do zadovoljivosti svih klauza, rešavač završava sa radom. WalkSAT i GSAT se razlikuju jedino po metodi izbora literala čiju vrednost menjaju. WalkSAT prvo bira klauzu koja je trenutno nezadovoljiva i menja nasumično izabran literal unutar te klauze. Klauza se bira na takav način da će u sledećoj iteraciji najmanji broj klauzi koje su bile zadovoljive, postati nezadovoljive. GSAT pravi promenu literala samo kada vidi da će ta promena napraviti manji broj nezadovoljivih klauzi u sledećoj iteraciji, sa malom verovatnoćom da će izabrati literal nasumično. Kada se bira optimalna klauza, WalkSAT radi manje kalkulacija od GSAT-a, zato što razmatra manje mogućnosti. MaxWalkSAT je verzija WalkSAT-a koja je dizajnirana da rešava MAX-SAT problem.

4. Ispravnost SAT Rešavača

Moderni SAT rešavači predstavljaju veoma složene softverske sisteme. Njihove implementacije zasnovane na CDCL sistemu pretrage uključuju kompleksne algoritme. Zbog ovoga je izuzetno teško odrediti da li sadrže greške ili ne, bez njihove formalne analize. Nove procedure se najčešće opisuju konkretnim implementacijama i njihovim tehničkim detaljima, dok su dokazi ispravnosti ovih procedura najčešće dati u veoma neformalnom obliku. S obzirom na njihovu primenu u aplikacijama poput verifikacije softvera i hardvera u kojima je ispravnost od kritičnog značaja, dokazivanje ispravnosti ovih procedura nosi ogroman značaj u rešavanju SAT problema.

Originalno, dokazi ispravnosti SAT rešavača rađeni na papiru sadržali su nedorečenosti i nepreciznosti. Pored toga, nisu sadržali dokaze ispravnosti algoritama i složenih struktura podataka koje se koriste u samim implementacijama. Napredak u oblasti specijalizovanog softvera za proveru formalnih dokaza (interaktivni dokazivači) doveo je do preciznijeg dokazivanja ispravnosti modernih SAT rešavača.

Internacionalna akademska konferencija za interaktivne dokazivače i asistente u dokazivanju (ITP Conference) je mesto za okupljanje zajednica koje koriste sisteme baziranih na logici višeg reda, kao ACL2, Coq, Isabelle, itd. ITP je deo IJCAR skupa konvencija, zajedno sa CADE (konferencija za automatizovanu dedukciju) i TABLEAUX. Na ovim događajima se takođe debatuje i o teorijskim osnovama nauke, implementacija njenih aspekata, i aplikacijama za programsku verifikaciju i bezbednost. Ovo samo pokazuje koliko je ova zajednica velika i široko rasprostranjena.



Interfejs Coq rešavača

Literatura

- Mladen S. Nikolić - **Usmeravanje Pretrage u Automatskom Dokazivanju Teorema**, Matematički Fakultet - http://nardus.mpn.gov.rs/bitstream/handle/123456789/2846/Disertacija.pdf?sequence=1&isAllowed=y
- Filip Marić - **Formalizacija, Implementacija i Primene SAT Rešavača**, Matematički Fakultet - http://poincare.matf.bg.ac.rs/~filip/phd/doktorat.pdf#section.4.4
- Lawrence C. Paulson - **Logic and Proof**, University of Cambridge - https://www.cl.cam.ac.uk/teaching/1213/LogicProof/logic-notes.pdf
- Predrag Janičić - **Matematička Logika u Računarstvu**, Matematički Fakultet - http://poincare.matf.bg.ac.rs/~janicic/books/mlr.pdf
- Carla P. Gomes, Henry Kautz, Ashish Sabharwal, Bart Selman - **Satisfiability Solvers**, Cornell Computer Science - https://www.cs.cornell.edu/gomes/pdf/2008_gomes_knowledge_satisfiability.pdf
- Add Gritman, Anthony Ha, Tony Quach, Derek Wenger - **Conflict Driven Clause Learning**, University of Washington - https://cse442-17f.github.io/Conflict-Driven-Clause-Learning/