UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Dragana P. Pejić

GENERISANJE DOKAZA NEZADOVOLJIVOSTI U SMT REŠAVAČIMA

master rad

Beograd, 2023.

Mentor:

dr Milan Banković, docent
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Filip Marić, redovni profesor
Univerzitet u Beogradu, Matematički fakultet

dr Sana Stojanović Đurđević, docent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Mojim roditeljima

Naslov master rada: Generisanje dokaza nezadovoljivosti u SMT rešavačima

Rezime: U ovom radu razmatraju se formati dokaza nezadovoljivosti za SAT i SMT rešavače sa posebnim osvrtom na format predstavljen u radu Hoenikea i Šindlerove na SMT radionici 2022 godine. U neophodnoj meri su predstavljeni teorijski osnovi, algoritmi za rešavanje SAT i SMT problema i najzastupljeniji formati dokaza. Dat je detaljan opis gorepomenutog formata i navedene su njegove aksiome. Deo ovog rada predstavlja i implementacija izvoza dokaza u ArgoSMT rešavaču. Detalji implementacije kao i evaluacija iste su dati u poslednjoj glavi rada.

Ključne reči: SAT problem, SAT rešavači, SMT problem, SMT rešavači, sertifikat, dokaz nezadovoljivosti

Sadržaj

Glava 1

Uvod

Problem ispitivanja zadovoljivosti formule F u datoj teoriji T zauzima jedno od centralnih mesta u matematičkoj logici kao i u teorijskom računarstvu. Ukoliko se radi o iskaznoj logici, ovaj problem se naziva SAT (engl. *Boolean satisfiability problem*) i prvi je problem za koji je dokazana NP kompletnost [5]. Za proveru zadovoljivosti iskaznih formula razvijeni su posebni softverski alati koji se nazivaju SAT rešavači. Tokom godina se radilo na njihovom usavršavanju što je za posledicu imalo ne samo poboljšanje njihove brzine i efikasnosti, već i kompleksnosti. Veća složenost rešavača dovela je do osnovane sumnje u rezultat njihovog rada te je postalo neophodno imati mogućnost njegove provere nekim jednostavnijim alatom. Savremeni SAT rešavači u slučaju zadovoljivosti formule koju ispituju mogu generisati model koji je lako proverljiv. Ukoliko je pak formula nezadovoljiva, može se generisati dokaz nezadovoljivosti u standardnom formatu za SAT rešavače koji se eksportuje u poseban izlazni fajl koji nazivamo *sertifikat*.

Medjutim, veliki broj problema je suviše komplikovan da bi bio uspešno predstavljen na jeziku iskazne logike pa samim tim i rešavan koristeći SAT rešavače. Za njihovo modelovanje, potrebno je koristiti izražajniji logički okvir, poput teorija prvog reda. Problem ispitivanja zadovoljivosti formule prvog reda nad unapred zadatom teorijom naziva se SMT problem (engl. *Satisfiability modulo theories*). Softverski alati koji implementiraju procedure odlučivanja za takve probleme se nazivaju SMT rešavači. Kao i kod SAT rešavača, rezultat rada SMT rešavača takođe mora biti proveriv nekim jednostavnijim alatom, s obzirom na kompleksnost njihove implementacije. Za razliku od SAT rešavača, još uvek ne postoji široko prihvaćen format dokaza za SMT rešavače, mada se na tome intenzivno radi. Jedan dobar korak u tom pravcu je rad Hoenikea i Šindlerove [10], predstavljen na SMT radionici 2022. godine.[1]

[1] rad je objavljen u zborniku radova sa radionice i dostupan je na adresi: https://ceur-ws.org/Vol-3185/

U ovom master radu je dat osvrt na evoluciju formata dokaza nezadovoljivosti za SAT rešavače, hronološki predstavljen put od najranijih rezolucijskih dokaza [11] do DRAT formata [16] koji je opšte prihvaćen kao standard.

Dalje, predstavljena su dosadašnja dostignuća na polju razvoja tehnika za produkciju dokaza nezadovoljivosti u SMT rešavačima i opisani su formati dokaza koji se koriste u najpopularnijim rešavačima. Centralni deo rada predstavlja detaljni prikaz formata dokaza dat u radu [10], a koji ćemo u nastavku ovog teksta zbog jednostavnosti nazivati *HS format* (po inicijalima autora). Izvoz dokaza u ovom formatu implementiran je u okviru SMT rešavača ArgoSMT [1], što je ujedno i glavni doprinos rada. Implementacija razvijena u okviru ovog rada je otvorenog koda i dostupna je javnosti.

Rad je organizovan po glavama na sledeći način. *Glava 2* donosi osnovne logičke definicije i pojmove neophodne za lakše razumevanje ostatka teksta. U *glavi 3* su predstavljena dva najvažnija algoritma za rešavanje SAT i SMT problema te je dat opis arhitekture SMT rešavača. *Glava 4* je posvećena generisanju dokaza u SAT rešavačima, navedeni su i ukratko objašnjeni najpoznatiji formati dokaza, dok su u *glavi 5* izloženi formati u kojima dva najpoznatija SMT rešavača - Z3 i CVC4 - generišu dokaze, a potom se prelazi na detaljni prikaz HS formata dokaza. *Glava 6* prikazuje detalje implementacije HS formata dokaza u okviru ArgoSMT rešavača. I za kraj, *glava 7* sumira postignute rezultate u ovom radu.

Glava 2

Osnovne definicije i pojmovi

2.1 Osnovni logički pojmovi

Neka je V prebrojiv skup iskaznih promenljivih. Iskazne formule nad V se grade na uobičajen način, polazeći od promenljivih iz V i logičkih konstanti $\top$ i $\bot$, koristeći logičke veznike $\neg$, $\wedge$, $\vee$, $\Rightarrow$ i $\Leftrightarrow$. Specijalno, literal je ili promenljiva v ili njena logička negacija $\neg v$. Suprotan literal literalu l označavamo sa $\bar{l}$. Ukoliko je literal l promenljiva v, tada je $\bar{l}$ negacija promenljive v. Ako je pak literal l negacija promenljive v, tada je njemu suprotan literal upravo promenljiva v.

Istinitosna vrednost literalima se pridružuje pomoću valuacija. Valuacija je funkcija koja skup svih literala preslikava u skup $\{0, 1\}$ (ili {netačno, tačno}, ...). Parcijalna valuacija je skup literala koji ne sadrži dva suprotna literala. Literal je tačan u datoj parcijalnoj valuaciji ako i samo ako pripada valuaciji, netačan ako njemu suprotan literal pripada valuaciji, a nedefinisan inače.

Za datu valuaciju v, istinitosna vrednost proizvoljne formule F se definiše induktivno na uobičajen način, podrazumevajući standardnu semantiku iskaznih veznika. Ukoliko je formula F tačna u valuaciji v, to označavamo sa $v \models F$. Kažemo još i da valuacija v zadovoljava formulu F, odnosno da je valuacija v model formule F.

Formula je *zadovoljiva* ako ima bar jedan model, a *nezadovoljiva* u suprotnom. Formula je tautologija (logički valjana) ako joj je svaka valuacija model, a formula je poreciva ako postoji valuacija koja joj nije model.

Formula je u disjunktivnoj normalnoj formi (DNF) ako je oblika $D_1 \vee \ldots \vee D_n$ pri čemu je svaki disjunkt D_i oblika $l_{i1} \wedge \ldots \wedge l_{im_i}$ gde su l_{ij} literali. Formula je u konjunktivnoj normalnoj formi (KNF) ako je oblika $K_1 \wedge \ldots \wedge K_n$ pri čemu je svaki konjunkt K_i oblika $l_{i1} \vee \ldots \vee l_{im_i}$ gde su l_{ij} literali. Disjunkcije literala se nazivaju *klauze*.

Signaturu (jezik) L čini skup funkcijskih simbola Σ, skup relacijskih simbola Π i funkcija $ar : (\Sigma \cup \Pi) \longrightarrow \mathbb{N}$ koja svakom simbolu dodeljuje arnost. Simboli konstanti se mogu shvatiti kao funkcijski simboli arnosti 0. Iskazne promenljive se mogu shvatiti kao relacijski simboli arnosti 0.

Termovi (jezika L) su izrazi izgrađeni primenom funkcijskih simbola na konstante i promenljive. Važi da je svaka promenljiva term, kao i da ako je c simbol konstante (jezika L), onda je c term, i shodno tome ako su $t_1, \ldots, t_k$ termovi, i f funkcijski simbol (jezika L) arnosti k, onda je i $f(t_1, \ldots, t_k)$ takođe term.

Atomičke formule se grade primenom relacijskih simbola na termove. Važi da su logičke konstante ($\top$ i $\bot$) atomičke formule kao i da je iskazno slovo (relacijski simbol arnosti 0) takođe atomička formula. Dodatno, ako je p relacijski simbol jezika L arnosti k, i ako su $t_1, \ldots, t_k$ termovi (jezika L), onda je $p(t_1, \ldots, t_k)$ atomička formula. Atomičke formule koje nisu logičke konstante, nazivamo atomima prvog reda (ili samo atomima).

Formule se grade od atomičkih formula primenom logičkih veznika ($\neg$, $\wedge$, $\vee$, $\Rightarrow$ i $\Leftrightarrow$) i kvantifikatora ($\forall$ i $\exists$) na uobičajan način.

Za dati jezik L, strukturu $\mathcal{D}$ čini:

- neprazan skup objekata (domen) D
- za svaki funkcijski simbol f arnosti k, njegova interpretacija $f_{\mathcal{D}} : D^k \longrightarrow D$ (tj. funkcija sa k argumenata nad D); specijalno, za svaki simbol konstante c, njegova interpretacija $c_{\mathcal{D}} \in D$ je fiksirani element domena D
- za svaki relacijski simbol p arnosti k, njegova interpretacija $p_{\mathcal{D}} \subseteq D^k$ (tj. relacija arnosti k nad D); specijalno, ako je p iskazno slovo, tada je njegova interpretacija $p_{\mathcal{D}} \in \{0, 1\}$ (ili $\{true, false\}$) fiksirana istinitosna vrednost

Valuacija prvog reda nad prebrojivim skupom promenljivih V je funkcija $v : V \longrightarrow D$ gde je D domen strukture $\mathcal{D}$. Ako je term t promenljiva x, onda je njegova vrednost u strukturi $\mathcal{D}$ i valuaciji v (u oznaci $\mathcal{D}_v(t)$) jednaka vrednosti promenljive x u valuaciji v, tj. $\mathcal{D}_v(t) = v(x)$. Ako je term t konstantni simbol c, onda je njegova vrednost interpretacija simbola c u strukturi $\mathcal{D}$, tj. $\mathcal{D}_v(t) = c_{\mathcal{D}}$. Ako je term t oblika $f(t_1, \ldots, t_k)$, onda je njegova vrednost jednaka rezultatu primene funkcije koja je interpretacija simbola f u strukturi $\mathcal{D}$ na vrednosti termova $t_1, \ldots, t_k$, tj. $\mathcal{D}_v(t) = f_{\mathcal{D}}(\mathcal{D}_v(t_1), \ldots, \mathcal{D}_v(t_k))$. Ako je atomička formula a iskazno slovo p, onda je njena istinitosna vrednost u strukturi $\mathcal{D}$ i valuaciji v (u oznaci $\mathcal{D}_v(p)$) jednaka istinitosnoj vrednosti iskaznog slova p u strukturi $\mathcal{D}$, tj. $\mathcal{D}_v(p) = p_{\mathcal{D}}$. Ako je atomička formula a oblika $p(t_1, \ldots, t_k)$, onda je njena istinitosna vrednost

tačno ako i samo ako k-torka $(\mathcal{D}_v(t_1), \ldots, \mathcal{D}_v(t_k))$ pripada relaciji $p_{\mathcal{D}}$ koji je interpretacija simbola p u strukturi $\mathcal{D}$.

Istinitosna vrednost složenih formula se definiše rekurzivno, imajući u vidu standardnu semantiku iskaznih veznika, kao i u iskaznoj logici. Posebno, formula $\forall x.F$ će biti tačna u strukturi $\mathcal{D}$ za valuaciju v ako je formula F tačna u strukturi $\mathcal{D}$ za svaku valuaciju v' koja se od valuacije v razlikuje eventualno samo po promenljivoj x. Slično, formula $\exists x.F$ će biti tačna u strukturi $\mathcal{D}$ i valuaciji v ako je formula F tačna u strukturi $\mathcal{D}$ i bar nekoj valuaciji v' koja se od valuacije v razlikuje eventualno samo po promenljivoj x.

Tačnost formule F u strukturi $\mathcal{D}$ pri valuaciji v označavamo sa $(\mathcal{D}, v) \models F$. Kažemo još i da valuacija v zadovoljava formulu F u strukturi $\mathcal{D}$, kao i da je par $(\mathcal{D}, v)$ model formule F.

Formula je valjana ako je tačna u svakoj valuaciji, pri svakoj interpretaciji. Valjanost formule F zapisujemo sa $\models F$. Formula F je zadovoljiva ako postoji struktura $\mathcal{D}$ i valuacija v takva da je $(\mathcal{D}, v) \models F$. Važi sledeće: formula F je valjana ako i samo ako je $\neg F$ nezadovoljiva.

Formula F je logička posledica skupa formula Γ (što označavamo sa $\Gamma \models F$) ako i samo ako za svaku interpretaciju $\mathcal{D}$ i valuaciju v važi da ako $(\mathcal{D}, v)$ zadovoljava svaku formulu iz Γ, onda zadovoljava i F (tj. ako je svaki model za skup Γ istovremeno i model za formulu F).

Teorija T nad signaturom L data je skupom struktura nad L koje nazivamo modelima teorije T.

Za formulu F kažemo da je valjana u teoriji T, odnosno T-valjana, u oznaci $\models_T F$ ako je tačna u svim njenim modelima. Formula je zadovoljiva u teoriji T (T-zadovoljiva) ako je tačna u bar jednom modelu teorije T. Formula F je logička posledica skupa formula Γ u teoriji T (što označavamo sa $\Gamma \models_T F$) ako i samo ako je F tačna u svim modelima teorije T u kojima su tačne sve formule iz Γ.

2.2 SAT problem i SAT rešavači

Problem ispitivanja zadovoljivosti iskazne formule naziva se SAT problem. Postoje naivni metodi za njegovo rešavanje, poput metoda istinitosnih tablica, ali su oni neefikasni zbog vremena koje je neophodno da se sprovedu. Problem se tehnički jednostavnije rešava ako je formula u nekom od specijalnih oblika kao što su DNF i KNF. Za DNF formule se zadovoljivost trivijalno ispituje, ali je postupak prevođenja formula u ovaj oblik netrivijalan. Sa druge strane, postoji efikasan postupak prevođenja formule u KNF

koji čuva zadovoljivost. Za ispitivanje zadovoljivosti KNF formule se koriste klauzalni algoritmi od kojh je najznačajniji CDCL algoritam [12] o kome će biti reči u poglavlju 3.1.

Implementacije procedura odlučivanja za SAT problem nazivaju se SAT rešavači od kojih očekujemo da imaju sledeći tri osobine:

1. *saglasnost* - ako rešavač prijavi nezadovoljivost, polazna formula je zaista nezadovoljiva

2. *potpunost* - ako je polazna formula nezadovoljiva, algoritam će prijaviti nezadovoljivost

3. *zaustavljanje* - za svaku polaznu formulu, rešavač se zaustavlja nakon primene konačno mnogo koraka

U slučaju da je polazna formula zadovoljiva, SAT rešavač na svom izlazu može dati i model koji to potvrđuje. U slučaju nezadovoljivosti formule, SAT rešavač moze proizvesti dokaz nezadovoljivosti, o čemu će biti više reči u poglavlju 2.4.

2.3 SMT problem i SMT rešavači

SMT problem za teoriju T je problem ispitivanja T-zadovoljivosti date formule F. Odlučivost SMT problema zavisi od izbora teorije T. Za pojedine teorije, SMT problem je odlučiv samo za neke njene fragmente, odnosno samo za formule određenog oblika. Ispitivanje valjanosti formule F u teoriji se svodi na SMT problem za formulu $\neg F$ u toj teoriji.

Softverski alati koji implementiraju procedure za rešavanje SMT problema nazivaju se SMT rešavači. Oni su relativno nova tehnologija i datiraju sa početka 21. veka. Zasnovani su na SAT tehnologiji uz bazne procedure odlučivanja i instanciranje kvantifikatora. Iako od SMT rešavača očekujemo da imaju iste tri najvažnije osobine (saglasnost, potpunost, zaustavljanje), ovo nije uvek ispunjeno i zavisi od teorije. Za sve procedure važi svojstvo saglasnosti. Postoje teorije za koje odgovarajuće procedure nemaju svojstva potpunosti i zaustavljanja.

SMT rešavači se koriste za verifikaciju softvera i hardvera, za rešavanje problema zadovoljavanja ograničenja, probleme planiranja i sl.

Poput SAT rešavača, i SMT rešavači mogu generisati model u slučaju kada je polazna formula T-zadovoljiva dok u slučaju nezadovoljivih formula, može biti generisan dokaz nezadovoljivosti.

2.4 Dokaz, sertifikat i proveravači dokaza

S obzirom da su SAT i SMT rešavači složen softver kome ne možemo apsolutno verovati, postavlja se pitanje kako proveriti rezultat njihovog rada.

U slučaju kada prijave zadovoljivost, obezbeđuje se i generisanje modela koji predstavlja zadovoljavajuću valuaciju u slučaju SAT problema, odnosno interpretaciju slobodnih simbola u slučaju SMT problema. Ovaj model se jednostavno može proveriti nekim eksternim alatom. Sa druge strane, ukoliko rešavači prijave nezadovoljivost, tada se vrši produkcija dokaza nezadovoljivosti u nekom deduktivnom sistemu. Izgradnja ovog dokaza se obavlja u toku samog rešavanja problema. Po završetku rada, rešavač izvozi dokaz u nekom formatu u poseban fajl koji nazivamo *sertifikat*.

Dobijeni sertifikat se dovodi na ulaz posebnog softverskog alata koji služi da proveri da li je svaki korak dokaza izveden na ispravan način u datom deduktivnom sistemu. Ovaj softverski alat se naziva *proveravač* (engl. *proof checker*). Njegovi ulazni parametri pored sertifikata mogu biti i formula čija zadovoljivost se ispituje kao i aksiome odgovarajuće teorije (u slučaju SMT rešavača). Proveravač je po svojoj strukturi mnogo jednostavniji od rešavača i njegova korektnost se može ponekad čak i ručno verifikovati.

U glavama 4 i 5 biće predstavljeni različiti formati u kojima se dokaz nezadovoljivosti izvozi u sertifikat.

Glava 3

Algoritmi za rešavanje SAT i SMT problema

3.1 CDCL algoritam

CDCL algoritam [12] nastao je početkom 21. veka. To je iterativni algoritam koji se zasniva na inkrementalnoj izgradnji parcijalne valuacije dodavanjem jednog po jednog literala vodeći računa da nijedna od klauza formule u tekućoj valuaciji ne postane netačna. Ukoliko pak do toga dođe, ta situacija se naziva *konflikt*. Kako bismo odredili tačku u pretrazi na koju je potrebno da se vratimo kako bismo proces pretrage nastavili dalje, drugim putem, vrši se analiza konflikta.

Kažemo da je klauza c *jedinična* u odnosu na parcijalnu valuaciju v ako i samo ako sadrži literal l nedefinisan u v, dok su joj svi literali različiti od l netačni u v. Da bi jedinična klauza bila zadovoljena, neophodno je da njen nedefinisani literal postane tačan, pa ga možemo dodati u parcijalnu valuaciju. Ovaj postupak poznat je i kao *jedinična propagacija*.

Literali se u valuaciji dodaju ili kao *literali odlučivanja* (engl. *decision literal*) ili kao *izvedeni literali* (engl. *inferred literal*). Literali odlučivanja su rezultat naših odluka i njima započinju *nivoi odlučivanja*. Izvedeni literali predstavljaju rezultat rezonovanja (najčešće jedinične propagacije) i posledica su literala odlučivanja.

Nakon što dođe do konflikta, prilikom njegove analize određeni broj literala odlučivanja sa kraja valuacije će biti uklonjen zajedno sa svim izvedenim literalima koji su njihove logičke posledice. Ako se konflikt pojavi u situaciji kada valuacija ne sadrži literale odlučivanja, formula je nezadovoljiva. U slučaju kada je valuacija potpuna, a konflikt ne postoji, formula je zadovoljiva.

Razmatranje procedure opisane na nivou koda programa je suviše komplikovano pošto sadrži dosta implementacionih detalja koji zavise od odabira jezika. Stoga u nastavku navodimo opštu CDCL proceduru u vidu pseudokoda (Algoritam 1) koja može biti prilagođena specifičnostima odabranog programskog jezika za implementaciju.

Algoritam 1 CDCL algoritam

```
1:  CDCL(CNF formula F, assignment ν)
2:  dl = 0                                   ▷ inicijalizacija početnog nivoa odlučivanja
3:  while (true) do
4:      if (UnitPropagation(F, ν) == CONFLICT) then      ▷ provera da li jedinična
    propagacija dovodi do konflikta
5:          β = ConflictAnalysis(F, ν)                   ▷ analiza konflikta
6:          if (β < 0) then
7:              return UNSAT
8:          else
9:              Backtrack(F, ν, β)
10:             dl = β          ▷ usaglašavanje nivoa odlučivanja nakon povratnog skoka
11:         end if
12:     else if (not AllVariablesAssigned(F, ν)) then
13:         l = PickBranchingLiteral(F, ν)           ▷ odabir novog literala odlučivanja
14:         dl = dl + 1          ▷ povećanje nivoa odlučivanja nakon odluke o odabiru
15:         ν.push(l)
16:     else
17:         return SAT
18:     end if
19: end while
```

Sastavni deo CDCL algoritma su i pomoćne funkcije koje razmatramo u nastavku:

- **UnitPropagation** predstavlja primenu operacije propagacije jediničnih klauza. Neka je klauza c jedinična klauza sadržana u formuli F i neka je l literal sadržan u c koji jedini nije definisan u trenutnoj parcijalnoj valuaciji ν dok su svi ostali literali iz c netačni u ν. Tada literal l mora biti tačan u eventualnoj zadovoljavajućoj valuaciji. Propagacija jedinične klauze c predstavlja uvršćivanje literala l u trenutnu valuaciju sa vrednošću tačno i proveru da li definisanje ovog literala može dovesti da neka druga klauza postane jedinična (u kom slučaju se postupak ponavlja za tu klauzu) ili netačna (u kom slučaju procedura prijavljuje konflikt).

- **AllVariablesAssigned** proverava da li su svim literalima dodeljene vrednosti, tj. da li je valuacija potpuna. Ukoliko je to ispunjeno, algoritam završava sa radom uz prijavljivanje zadovoljivosti formule F.

- **PickBranchingLiteral** odabira nedefinisani literal l koji će biti dodan u trenutnu valuaciju v; time se započinje novi nivo odlučivanja valuacije v. SAT rešavači koriste različite heuristike za odabir novog pretpostavljenog literala, ali one u ovom radu neće biti razmatrane.

- **ConflictAnalysis**, kao što joj samo ime govori, obavlja analizu konflikta i određuje nivo povratnog skoka β. Rezulat njenog rada je i učenje novih klauza. Kako se na svakom nivou odlučivanja pridruži vrednost nekim od prethodno nedefinisanih literala, to nivoi odlučivanja definišu parcijalni poredak promenljivih. Krenuvši od netačne klauze koja je uzrok konflikta, procedura obilazi njene literale u obrnutom poretku od njihovog poretka u parcijalnoj valuaciji i identifikuje klauze koje su uzrokovale jedinične propagacije koje su te literale učinile netačnim. Ove klauze koristimo da pomoću njih modifikujemo trenutnu konfliktnu klauzu, primenom pravila rezolucije koje će biti objašnjeno u daljem tekstu. Ovaj proces se ponavlja sve dok ne dođemo u *tačku jednoznačne implikacije* (engl. *unique implication point* (UIP)), tj. u tačku u kojoj je samo jedan literal trenutne konfliktne klauze na tekućem nivou odlučivanja, dok su svi ostali na nižim nivoima. Najviši od tih nižih nivoa upravo predstavlja nivo povratnog skoka β.

  Preciznije, neka je d nivo odlučivanja na kome je došlo do konflikta, l_i poslednji odabrani literal odlučivanja i w_j klauza koja je izazvala konflikt. Neka $\odot$ predstavlja *operator rezolucije*. Ako za dve klauze w_j i w_k za koje postoji jedinstvena promenljiva x takva da jedna klauza sadrži literal x dok druga sadrži literal $\neg x$, tada $w_j \odot w_k$ sadrži sve literale prisutne u w_j i w_k sem x i $\neg x$. Procedura analize konflikta koja se koristi u modernim SAT rešavačima može biti definisana kao sekvenca primene pravila rezolucije na odgovarajućim klauzama koja na svakom koraku proizvodi novu privremenu klauzu. Neka je vrednost funkcije $\xi(w, v, d)$ jednaka literalu klauze w koji je poslednji poništen u valuaciji v na nivou d i neka je ukupan broj literala klauze w koji su u valuaciji v poništeni na nivou d dat funkcijom $\phi(w, v, d)$.

  Neka je $w_L^{d,i}$, gde je $i = 0, 1, \ldots$, privremena klauza dobijena nakon primene pravila rezolucije i puta. Koristeći prethodno definisane funkcije ξ i ϕ, $w_L^{d,i}$ možemo izraziti kao:

$$w_L^{d,i} = \begin{cases} w_j, & i = 0 \\ w_L^{d,i-1} \odot \alpha(l), & i \neq 0 \wedge \phi(w_L^{d,i-1}, v, d) > 1 \wedge \xi(w_L^{d,i-1}, v, d) = l \\ w_L^{d,i-1}, & i \neq 0 \wedge \phi(w_L^{d,i-1}, v, d) = 1 \end{cases}$$

Početna vrednost privremene klauze $w_L^{d,i}$ je klauza w_j koja je konflikt i izazvala. Iterativno se računa vrednost $w_L^{d,i}$ sve dok ne dođemo u tačku jednoznačne implikacije. Klauza $w_L^{d,i}$ tada predstavlja klauzu koju je potrebno dodati u skup klauza, tj. ovo je naučena klauza.

- **Backtrack** funkcija vrši povratni skok na nivo β koji je određen u fazi analize konflikta. Nakon povratka na nivo β, naučena klauza povratnog skoka će biti jedinična, pa će u sledećoj iteraciji `while` petlje izazvati jediničnu propagaciju odgovarajućeg literala i time pokrenuti novi ciklus jedinične propagacije.

Na kraju ovog poglavlja potrebno je naglasiti da je detaljan opis pojedinih delova CDCL algoritma namerno izostavljen. Cilj opisa algoritma je bio da čitaoca upozna sa njegovim osnovama koje će biti korisne za izlaganja u narednim poglavljima. Celovitiji opis algoritma može se naći u [12].

3.2 CDCL(T) algoritam

SMT rešavači su obično zasnovani na *lenjom pristupu* koji se karakteriše sledećim:

- najpre se odvija iskazna apstrakcija - (bazni) atomi prvog reda se zamenjuju iskaznim slovima, a teorija T se ostavlja po strani
- formula dobijena iskaznom apstrakcijom se predaje SAT rešavaču koji ispituje njenu zadovoljivost
- ukoliko SAT rešavač zaključi da je formula nezadovoljiva, tj. rezultat je UNSAT, tada je formula nezadovoljiva i u teoriji T
- ukoliko SAT rešavač prijavi SAT, zadovoljavajuća iskazna valuacija (model M) koja je dobijena određuje konjunkciju baznih literala
- posebna procedura odlučivanja koja utvrđuje zadovoljivost konjunkcije baznih literala nad teorijom T (koju nazivamo T-rešavač) proverava zadovoljivost modela M u teoriji T; ukoliko se pokaže da postoji model teorije T koji istovremeno zadovoljava sve literale iz M, tada je to ujedno i model početne formule; ukoliko to nije slučaj, tada T-rešavač negacijom konjunkcije literala M konstruiše klauzu (teorijsku lemu) koja se dodaje početnoj formuli, a čija svrha je isključenje M pri ponovnom pokretanju SAT rešavača.

- postupak se ponavlja sve dok SAT rešavač ili ne vrati UNSAT ili ne vrati model M za koji T-rešavač ustanovi da je zadovoljiv u teoriji T

Glavna prednost lenjog pristupa je njegova fleksibilnost pošto je jednostavno kombinovati bilo koji SAT rešavač sa bilo kojim T-rešavačem.

Arhitektura SMT rešavača koja se najčeće koristi, a zasnovana je na ovom pristupu je *CDCL(T)*. Ovi rešavači imaju modularnu strukturu, sastoje od SAT rešavača zasnovanog na CDCL algoritmu (Algoritam 1) i T-rešavača koji su jasno odvojeni i komuniciraju putem precizno definisanog interfejsa. SAT rešavač inkrementalno konstruiše zadovoljavajuće iskazne valuacije. T-rešavač ispituje zadovoljivost odgovarajuće konjunkcije literala prvog reda u teoriji T u toku konstrukcije zadovoljavajuće valuacije.

CDCL(T) arhitektura SMT rešavača pruža mogućnost dodatnih unapređenja lenjog pristupa koja mogu učiniti postupak rešavanja još efikasnijim. Dva najvažnija navodimo u nastavku.

Provera T-zadovoljivosti tokom izgradnje parcijalne valuacije. Zadovoljivost odgovarajuće konjunkcije literala prvog reda u teoriji T može biti proveravana u toku konstrukcije parcijalne valuacije.

Ovo svojstvo omogućava ubrzanje rada rešavača pošto se nekonzistentnosti sa teorijom T mogu otkriti odmah nakon što nastanu. Ukoliko je provera nakon svakog dodavanja novog literala u valuaciju preskupa, može se definisati period nakon koga se vrši provera (npr. nakon svakog dodavanja k literala u parcijalnu valuaciju T-rešavač utvrđuje da li je valuacija konzistentna sa teorijom).

U trenutku pisanja ovog rada, većina implementacija SMT rešavača u svom sklopu sadrže inkrementalni T-rešavač. Ovakva upotreba T-rešavača postavlja novi zahtev za njegovu implementaciju. Da bi inkrementalni pristup bio efikasan u praksi, potrebno je da rešavač bude brži u obradi jednog dodatnog literala l nego u ponovnoj obradi prethodne valuacije i dodatnog literala l zajedno.

Teorijska propagacija. U pristupu opisanom do sada T-rešavač generiše dodatne informacije koje mogu biti od značaja za rešavanje problema tek nakon što je kreirana parcijalna valuacija koja je nezadovoljavajuća u teoriji T. Slobodno možemo reći da je T-rešavač tu da validira pronadjenu valuaciju *aposteriori*, umesto da potpomogne njeno pronalaženje *apriori*. Kako bi se prevazišlo ovo ograničenje, T-rešavač može pronaći literale koji su logička posledica trenutne parcijalne valuacije M u teoriji T. Ovakvi literali se mogu dodati u valuaciju. Opisani proces se naziva *teorijska propagacija*.

U nastavku navodimo pseudokod CDCL(T) procedure (Algoritam 2). U pseudokodu ispod, T-rešavač je označen sa T-solver, a komunikacija između SMT rešavača i T-

rešavača je prikazana kroz pozive metoda interfejsa od strane SMT rešavača koji su zapisani sa $T\_solver \longrightarrow imeMetode(spisak\ parametera)$.

Algoritam 2 CDCL(T) procedura

```
1:  CDCL_T(CNF formula F, assignment ν)
2:  dl = 0
3:  while (true) do
4:      if (UnitPropagation(F, ν) == CONFLICT
5:          || T_solver ⟶ theoryPropagate(ν) == CONFLICT) then
6:          β = ConflictAnalysis(F, ν)
7:          if (β < 0) then
8:              return UNSAT
9:          else
10:             Backtrack(F, ν, β)
11:             dl = β
12:             T_solver ⟶ backtrack(dl)
13:         end if
14:     else if (not AllVariablesAssigned(F, ν)) then
15:         l = PickBranchingLiteral(F, ν)
16:         dl = dl + 1
17:         T_solver ⟶ newLevel(dl)
18:         ν.push(l)
19:         T_solver ⟶ assert(l)
20:     else
21:         return SAT
22:     end if
23: end while
```

Razmotrimo pomoćne funkcije koje su se javile u pseudokodu.

- **UnitPropagation**, **AllVariablesAssigned**, **PickBranchingLiteral**, **ConflictAnalysis** i **Backtrack** predstavljaju iste funkcije koje su objašnjene u pseudokodu CDCL procedure (Algoritam 1). Funkcija **ConflictAnalysis** je proširena pozivima $T\_solver \longrightarrow explainLiteral(l)$ i $T\_solver \longrightarrow explainConflict()$. Za svaki literal l u konfliktnoj klauzi koji je propagiran teorijskom propagacijom, poziva se $explainLiteral(l)$. Ova funkcija vraća klauzu koja predstavlja objašnjenje i koja se rezolvira sa tekućom konfliktom klauzom. Ukoliko je konflikt prijavljen u teoriji, tada dolazi do poziva $explainConflict()$ kako bi se dobila inicijalna konfliktna klauza (teorijska lema).

- metod **theoryPropagate** se poziva nakon dodavanja novih literala u valuaciju i T-rešavaču prosleđuje trenutnu valuaciju na osnovu koje T-rešavač generiše sve ulazne literale koji su njena T-posledica i njih prosleđuje SAT rešavaču.

- poziv metoda **newLevel** u T-rešavaču uspostavlja novi nivo odlučivanja, dok **assert** dodaje literal na stek T-rešavača nakon što je on dodat na stek SAT rešavača. Metoda **assert** se takođe poziva i unutar funkcije **UnitPropagation**, nakon svakog dodavanja literala u parcijalnu valuaciju jediničnom propagacijom, kako bi teorijski rešavač bio upoznat sa stanjem parcijalne valuacije.

- ukoliko postoji konflikt u teoriji, dobijen kao rezultat metoda **theoryPropagate**, neophodno je pozvati metod **explainConflict** koji će vratiti klauzu koja predstavlja objašnjenje konflikta i koja će biti naučena.

Dokaz korektnosti CDCL(T) procedure, detalji nekih od navedenih metoda kao i primeri primene CDCL(T) procedure nad konkretnim teorijama mogu se naći u [13].

Glava 4

Generisanje dokaza u SAT rešavačima

Veliki broj modernih SAT rešavača je zasnovan na CDCL algoritmu. Jedan od najvažnijih delova ovog algoritma predstavlja analiza konflikata koja za posledicu može imati dodavanje novih klauza u postojeći skup, kao što je diskutovano u poglavlju 3.1.

Dodatno, moderne CDCL SAT rešavače karakteriše i upotreba različitih tehnika prilikom pretprocesiranja formule, ali i tokom izvršavanja samog algoritma, što za posledicu može imati ne samo dodavanje novih klauza u tekući skup već i brisanje nekih postojećih.

Stoga je neophodno da dokaz koji CDCL rešavač generiše bude u formatu koji je pogodan za proveru da li je dodavanje novih klauza tokom rada rešavača uticalo na zadovoljivost početnog skupa klauza.

U nastavku poglavlja dat je osvrt na postojeće formate dokaza.

4.1 Rezolucijski dokazi

Prvi pokušaji produkcije dokaza iz CDCL rešavača su bili zasnovani na metodu rezolucije [11].

Pokazano je u [4] da novodobijene klauze koje su nastale prilikom analize konflikata mogu biti predstavljene kao posledica uzastopne primene pravila rezolucije nad odabranim klauzama iz ulaznog skupa. Kako je rezolucija relativno jednostavna operacija, to postoje jednostavniji algoritmi koji mogu biti korišćeni za njenu proveru [11, 6].

Tokom svog rada, SAT rešavač generiše rezolucijski dokaz koji alat za proveru uzima kao ulazni parametar zajedno sa početnom KNF formulom. Ovaj alat, znatno jednostavniji od rešavača, verifikuje da postoji niz uzastopne primene pravila rezolucije nad ulaznim ili već izvedenim klauzama čiji je rezultat prazna klauza.

Ukoliko je ishod rada rešavača UNSAT, a proveravač ne može da potvrdi ispravnost izvođenja prazne klauze, to je znak da postoji greška u implementaciji SAT rešavača.

Međutim, dokazi zasnovani na ovom metodu mogu biti izuzetno veliki i nepraktični za obradu. Takođe potrebno je izmeniti SAT rešavače kako bi podržali generisanje rezolucijskih dokaza što može biti komplikovano i zahtevati značajno dodatno vreme za njihov razvoj pošto pored pažljivog odabira klauza na koje se primenjuje rezolucija treba voditi računa i o redosledu izvođenja operacije. Odabrati pravi redosled često nije jednostavno te modifikovanje rešavača da generišu rezolucijske dokaze nije jednostavan zadatak čak ni za njihove programere.

4.2 RUP format dokaza

Klauzalni dokazi, poznati i kao RUP (engl. *Reverse Unit Propagation*) dokazi, su zasnovani na ideji dokaza Goldberga i Novikova [7]. Oni su pokazali da svaka klauza povratnog skoka L koja proistekne iz analize konflikta može biti proverena koristeći algoritam jedinične propagacije, poznat i kao BCP (*Boolean constraint propagation*) algoritam [3]. Ako rezultat $BCP(F \cup \bar{L})$, gde je F skup klauza koje trenutno postoje u formuli, sadrži praznu klauzu $\emptyset$, tada L sledi iz skupa F. Za svaku klauzu L za koju važi ova osobina kažemo da ima svojstvo asimetrične tautologije (AT). Ovo svojstvo se još naziva i RUP svojstvo.

Pseudokod algoritma za proveru RUP dokaza dat je u algoritmu 3. Ulazni parametri su KNF formula F i lista klauza povratnog skoka Q u redosledu njihovog učenja.

Klauzalni dokazi su predstavljeni u obliku niza klauza $(L_1, \ldots, L_m)$ gde važi da je $L_m = \emptyset$. Za datu KNF formulu F, dokaz se sastoji od klauza koje imaju RUP svojstvo u odnosu na klauze koje sačinjavaju formulu F. Dokaz se izgrađuje iterativno na sledeći način. Neka je $F_0 = F$, tada je $F_i = F_{i-1} \cup L_i$ gde je L_i klauza povratnog skoka u i-tom koraku za koju važi da poseduje RUP svojstvo u odnosu na F_{i-1}.

Zbog toga što se RUP dokaz sastoji samo iz klauza povratnog skoka, on je značajno kraći od rezolucijskog dokaza. Takođe, može biti izražen u obliku KNF-a (u DIMACS formatu). Potrebne su minimalne promene samih SAT rešavača kako bi se obezbedilo generisanje klauzalnih dokaza.

Algoritam 3 RUP provera; BCP algoritam

```
1: RUP(CNF formula F, queue Q of clauses)
2: while Q is not empty do
3:     L = Q.pop()
4:     F' = BCP(F ∪ L̄)
5:     if ∅ ∉ F' then
6:         return "checking failed"
7:     end if
8:     F = BCP(F ∪ L)
9:     if ∅ ∈ F then
10:        return "unsatisfiable"
11:    end if
12: end while
13: return "all clauses validated"
14:
15: BCP(CNF formula F)
16: while ∃x ∈ F do
17:    for C ∈ F with x̄ ∈ C do
18:        C = C\{x̄}
19:    end for
20:    for C ∈ F with x ∈ C do
21:        F = F\{C}
22:    end for
23: end while
24: return F
```

4.3 RAT format dokaza

Neka je KNF formula F i $C = (l_1 \vee l_2 \vee \ldots \vee l_n) \in F$ proizvoljna klauza. Kažemo da C ima svojstvo RAT (engl. *Resolution Asymmetric Tautology*) ako važi:

1. C ima svojstvo AT, ili
2. postoji literal $l \in C$ takav da za svaku klauzu $C' \in F$ takvu da $\bar{l} \in C'$, rezolventa dobijena rezolucijom nad klauzama C i C' ima svojstvo AT

Potrebno je polinomijalno vreme za proveru da li skup klauza ima RAT svojstvo. Dodavanje klauza koje imaju RAT svojstvo čuva zadovoljivost, odnosno nezadovoljivost početnog skupa klauza. Može se pokazati da sve promene skupa koje se dešavaju tokom pretprocesiranja, obrade i rešavanja problema prilikom rada modernih SAT rešavača mogu biti predstavljene preko dodavanja i uklanjanja klauza koje imaju RAT svojstvo [9].

RAT dokazi predstavljaju uopštenje RUP dokaza pošto pored klauza dobijenih analizom konflikata, oni mogu opravdati i klauze dobijene različitim vrstama preprocesiranja koje moderni SAT rešavači uključuju.

U nastavku je naveden pseudokod algoritma za proveru ovih dokaza. Ulazni parametri su isti kao i za RUP algoritam provere. $F_{\bar{l}}$ predstavlja skup klauza koje sadrže literal $\bar{l}$.

Algoritam 4 RAT provera

```
1:  RAT(CNF formula F, queue Q of clauses)
2:  while Q is not empty do
3:      L = Q.pop()
4:      if ∅ ∉ BCP(F ∪ L̄) then
5:          let l be the first literal in L
6:          for each C′ ∈ F_l̄ do
7:              R = Resolution(C′, L)
8:              if ∅ ∉ BCP(F ∪ R̄) then
9:                  return "checking failed"
10:             end if
11:         end for
12:     end if
13:     F = BCP(F ∪ L)
14:     if ∅ ∈ F then
15:         return "unsatisfiable"
16:     end if
17: end while
18: return "all clauses validated"
```

4.4 DRUP i DRAT formati dokaza

Algoritam 3 koji je predstavljen u poglavlju 4.2 je zahtevan za dokaze koji se sastoje od velikog broja klauza pošto vreme njegovog izvršavanja može biti nekoliko puta duže od vremena rada SAT rešavača. Tokom pretrage, SAT rešavači uklanjaju neke od prethodno naučenih klauza time smanjujući prostor pretrage, dok predloženi RUP algoritam to nije u stanju da verifikuje. Štaviše, on dopušta samo dodavanje novih klauza.

Kako bi se prevazišao taj problem, predloženo je da se generisanje dokaza proširi sa zapisivanjem informacija o uklonjenim klauzama. Novi format dokaza je nazvan DRUP, što je skraćenica od *Deletion RUP* [8]. Ostavljena je mogućnost dodavanja klauza (baš kao i kod RUP dokaza), ali je omogućeno i njihovo brisanje. Primer sertifikata eksportovanog u DRUP formatu je naveden ispod.

```
    −2  0
d   −2  3  0
    −1  0
d   −1  2  0
     3  0
     0
```

Ovaj format sa DIMACS formatom ima sličnosti u tome što su literali predstavljeni brojevima u redosledu u kome se pojavljuju u klauzama, dok je njihova negacija predstavljena znakom minus ispred broja literala. Obrisane klauze imaju prefiks *d*. Alat koji se koristi za proveru ovih dokaza je *DRUP-trim* koji ima dva ulazna parametra - KNF formulu predstavljenu u DIMACS formatu i sertifikat koji je SAT rešavač proizveo u DRUP formatu.

DRAT je sinonim za *Deletion RAT* i predstavlja format dokaza zasnovan na RAT dokazu uz dodatak zapisa informacija o uklonjenim klauzama. Sertifikat u DRAT formatu je sintakstno identičan DRUP formatu. Alat za proveru ovih dokaza je *DRAT-trim* [16] koji za svaku liniju dokaza proverava da li važi RAT svojstvo. Njegovi ulazni podaci su isti kao i za DRUP-trim. I pored toga što nije formalno verifikovan, u trenutku pisanja ovog rada, DRAT-trim je jedan od najčešće korišćenih alata za proveru dokaza izvezenih iz SAT rešavača.

Glava 5

Generisanje dokaza u SMT rešavačima

Osnovni motiv za generisanje dokaza u SMT rešavačima ogledao se u želji da se korisnicima omogući nezavisna provera rezultata njihovog rada umesto da se oslanjaju na korektnost implementacije samog rešavača koja zavisi od koda koji se stalno menja i unapređuje. Dodatni motiv je predstavljala nada da generisanje dokaza može imati ulogu u lakšem otkrivaju grešaka u samim rešavačima.

Istorijski gledano, najvažniji doprinos implementaciji generisanja dokaza u SMT rešavačima je efikasna integracija SAT rešavača u SMT rešavač.

Prvi pokušaji ove integracije su bili neuspešni. Deo SMT rešavača koji se bavio rezonovanjem u teoriji predstavljao je crnu kutiju u odnosu na ostatak sistema i bez pružanja dodatnih informacija, proglašavao je čitave delove problema nezadovoljivim. Bez dodatnih informacija (poput poznavanja klauze koja objašnjava teorijski konflikt i dovodi do nezadovoljivosti), SAT rešavač nije mogao u potpunosti da iskoristi tehnike nehronološkog vraćanja unazad i učenja novih klauza te često nije bio u mogućnosti da završi sa radom i nađe rešenje, čak ni za jednostavne probleme za koje je rešenje bilo unapred poznato.

U razgovoru naučnika sa univerziteta Stenford i Kormaka Flanagana, došlo se na ideju da se infrastruktura korišćena za generisanje dokaza koristi i za računanje konfliktnih klauza (više o ovome se može pročitati u [2]). Ova ideja je bila ključ uspešne integracije SAT rešavača u SMT rešavače i poslužila za razvoj CDCL(T) arhitekture na kojoj je zasnovana velika većina modernih SMT rešavača.

S obzirom da je, u slučaju CDCL(T) arhitekture, jezgro SMT rešavača CDCL-zasnovan SAT rešavač, srž dokaza koji generiše SMT rešavač je rezolucijski dokaz dobijen SAT rešavačem. Specijalno, proces analize konflikta i učenja novih klauza je posebno važan za SMT dokaze. SAT rešavač može da generiše rezolucijski dokaz koji odgovara ovom pro-

cesu.

U kontekstu SAT-a, konflikt se uvek javlja kao posledica toga što je neka od klauza postala netačna u tekućoj parcijalnoj valuaciji. Drugačija vrsta konflikta se može javiti kod SMT-a. Ilustrujmo je na primeru. Neka je SAT rešavač postavio valuaciju iskazne promenljive $p_{a=b}$ (koja predstavlja apstrakciju atoma $a = b$) na tačno, samo da bi u nekom od narednih koraka konstatovao da je $p_{f(a)=f(b)}$ netačno; ove konstatacije su deo valuacije, ali su nekonzistentne sa teorijom nad kojom se radi (u odnosu na teoriju EUF [10] u ovom slučaju). T-rešavač koji je sastavni deo SMT rešavača proverava da li je svaka nova dodela konzistentna sa teorijom. U ovom slučaju, prijavio bi konflikt SAT rešavaču i generisao klauzu $\neg p_{a=b} \vee p_{f(a)=f(b)}$ koja se dodaje u trenutni skup. Ovako generisana klauza se naziva *lema teorije* i može biti korištena u nastavku kao i svaka druga klauza.

Ukoliko teorijski rešavač može da proizvede i dokaze lema, onda se dokaz nezadovoljivosti SMT formule može dobiti kombinacijom rezolucijskog dokaza koji generiše SAT rešavač i dokaza lema teorije koje generiše teorijski rešavač. Međutim, neophodno je obezbediti prostor za sakupljanje i skladištenje svih ovih dokaza. Iako je ovo problem tehničke prirode, može dovesti do značajnog usporenja rešavača.

Dodatne poteškoće pri generisanju dokaza u SMT rešavačima predstavljaju i:

- skolemizacija
- pretprocesiranje

Postupak skolemizacije nije trivijalan. Eliminacija egzistencijalnih kvantifikatora iz formule se odvija kroz uvođenje novih simbola neinterpretiranih konstanti ili funkcijskih simbola.

SMT rešavači obično uključuju i modul za pretprocesiranje čiji je zadatak da uprosti polaznu formulu i svede je na KNF pre započinjanja samog procesa ispitivanja da li je problem zadovoljiv. Pored transformacije polazne formule u KNF, ovaj modul može dodatno uključivati različite transformacije, od jednostavnih prezapisivanja (npr. $x = y$ biće zapisano kao $y = x$) pa do kompleksnih globalnih uprošćavanja koja menjaju strukturu formule. Na kraju pretprocesiranja, formula obično biva pretvorena u KNF. Klauze prisutne u KNF formuli će biti deo rezolucijskog dokaza koji se generiše stoga je neophodno generisati i dokaze koji opravdavaju njihovo kreiranje tokom procesa pretprocesiranja.

Zbog svega navednog, bitno je usaglasiti format dokaza kako bi rezultat rada svih SMT rešavača mogao biti proverljiv od strane istog proveravača. Do sada nije došlo do pronalaska formata koji bi zadovoljio pojedinosti svih najvažnijih SMT rešavača, a istovremeno bio praktičan, čitljiv i kompaktan.

U nastavku poglavlja, biće ukratko opisani formati dokaza koje koriste dva najpoznatija SMT rešavača Z3 i CVC4, nakon čega će biti predstavljen HS format dokaza predložen u [10] koji je u fokusu ovog rada.

5.1 Format dokaza u Z3 rešavaču

Z3 je jedan od najpoznatijih SMT rešavača današnjice. Razvio ga je Majkrosoft Risirč [1]. Javno je dostupan i besplatan za nekomercijalnu upotrebu. Najčešće se koristi kao podrška drugim alatima, pre svega alatima za analizu i ispitivanje ispravnosti softvera. Podržava teoriju neinterpretiranih funkcija, teoriju linearne aritmetike, teoriju nelinearne aritmetike, teoriju bitvektora i teoriju nizova.

U Z3 rešavaču, objekti od kojih su izgrađeni dokazi su predstavljeni termovima. Motiv za generisanje dokaza na osnovu termova je modularna struktura samog rešavača, tj. bilo je bitno da se omogući efikasna komunikacija između CDCL zasnovanog SAT rešavača i T-rešavača koji zajedno učestvuju u procesu otkrivanja i analize konflikta.

Svaki rešavač na osnovu izabranih hipoteza iz datog skupa izvodi zaključak primenom određenih pravila. Osnovna pravila koja se primenjuju su:

- *hypothesis*, uvodi nove pretpostavke
- *lemma*, eliminiše pretpostavke
- *unit resolution*, propagacija pretpostavki

Z3 rešavač je zasnovan na CDCL(T) arhitekturi koju karakteriše stanje $M||F$ koje se održava tokom rada. M je parcijalna valuacija, predstavljena kao lista literala prvog reda nad atomima sadržanim u formuli F. Dodatno, podrazumeva se da je formula F u konjunktivnoj normalnoj formi. Tokom pretrage dolazi do dodavanja atoma u M na na osnovu jedinične i teorijske propagacije sve dok se svim atomima prvog reda koji se javljaju u formuli F ne dodeli vrednost, a da pritom nije otkriven konflikt sa nekom od klauza ili sa teorijom.

Metod za generisanje dokaza u rešavačima CDCL(T) arhitekture pogodan je za kreiranje rezolucijskih dokaza pošto se pamte klauze koje su izazvale jedinične propagacije. Pored toga, teorijski rešavači su u stanju da generišu objašnjenja teorijskih propagacija. Ovo omogućava da se tokom analize konflikta generiše rezolucijsko izvođenje naučene klauze povratnog skoka.

[1] Z3 je dostupan na lokaciji: https://github.com/Z3Prover/z3

Pristup odabran u Z3 rešavaču isključuje zapisivanje i umesto toga izgrađuje objekte dokaza tokom rešavanja konflikta. Svakoj klauzi koja tada nastane se pridružuje dokaz. Klauzama koje su nastale od ulaznih klauza tokom pretprocesiranja pridružen je dokaz generisan u toj fazi rada rešavača.

U CDCL(T) arhitekturi rešavača, procedure odlučivanja za teoriju T identifikuju skup literala sadržanih u parcijalnoj valuaciji koji nisu konzistentni sa teorijom T. Drugim rečima, disjunkcija negiranih literala parcijalne valuacije je T-valjana. Ovo za posledicu ima da termovi dokaza koji su generisani u T-rešavačima mogu biti iskazani jedinstvenim termom koji se naziva lema teorije. Neke leme sadrže i dodatne informacije koje pomažu proveravačima dokaza da rekonstruišu dokaz u celosti.

Modul za pretprocesiranje Z3 primenjuje standardna pravila pojednostavljivanja sadržana u podržanim teorijama. Treba primetiti da pravila prezapisivanja u Z3 rešavaču nisu predstavljena aksiomama te se i za njih generiše dokaz. Zbog toga je i njih potrebno proveriti proveravačem.

Posto su dokazi u Z3 sačinjeni od termova, mogu biti eksportovani u SMT-LIB formatu.

5.2 Format dokaza u CVC4 rešavaču

CVC (engl. *Cooperating Validity Checker*) rešavač [14] prvi je SMT rešavač koji je podržavao generisanje dokaza. Njegovi tvorci su Aron Stamp i Klark Baret sa univerziteta Stenford. CVC je zamišljen kao zamena za SVC (engl. *Stanford Validity Checker*) rešavač i predviđen je kao platforma za istraživanje ideja i algoritama vezanih za SMT problem na univerzitetu Stenford.

Format dokaza koji koristi CVC rešavač je zasnovan na LF-u (engl. *Edinburgh Logical Framework*). Dokaz može biti proveren pomoću Flea proveravača koji je razvijan paralelno sa CVC rešavačem.

Svaki deduktivni korak koji se napravi u rešavaču imao je ekvivalentan korak u produkciji dokaza. Na početku je CVC rešavač imao interni modul iskazne logike koji je čuvao dokaze. Dokazi nisu bili dostupni kada je rad SMT rešavača bio sveden na korišćenje internog SAT rešavača, tj. kada je CVC bio upotrebljavan za klasične SAT probleme. U verzijama CVC Lite i CVC3 ovo je promenjeno. Kreirana je specijalna klasa nazvana *Theorem* za koju je previđeno da čuva sve izvedene klauze i njihove dokaze. Objekti ove klase su mogli biti kreirani od strane posebnih funkcija za produkciju dokaza. Ovo

je omogućilo da sav kôd čijem radu se bezuslovno veruje, tj. kôd koji predstavlja pravila dokazivanja, bude sačuvan u nekoliko fajlova.

Pokazalo se da je ovaj pristup, iako elegantan, izuzetno neefikasan. Naime, mnogi generisani deduktivni koraci nisu igrali ulogu u krajnjem dokazu, ali su svejedno bili sačuvani. Ono što je možda čak bilo i važnije je da se generisanje dokaza nije moglo isključiti.

U verziji rešavača koja je usledila nazvanoj CVC4 promenjen je način generisanja dokaza i on se sada obavlja daleko efikasnije. Prva promena koju je donela nova verzija je uvođenje markera koji označava da li je generisanje dokaza uključeno. Ovaj marker se podešava pre početka rada rešavača i kompilatoru sugeriše da li modul za dokazivanje treba uključiti. Ova izmena je uvedena kako bi se moglo isključiti generisanje dokaza ukoliko ono nije potrebno.

Ukoliko pak korisnik naknadno odluči da ipak želi dokaz, njega je moguće dobiti retroaktivno. Naime, SAT rešavači beleže minimalni skup koraka na osnovu kojih može biti rekonstruisan rezolucijski dokaz. Kada se rešavač nalazi u ovom režimu rada, moduli teorije ne pamte nijedan korak, to se dokazi teorijskih lema koje se koriste u rezolucijskom dokazu naknadno generišu u modulima teorije tako što se procedure odlučivanja koje su sadržane u njima ponovo izvrše.

Kada je marker postavljen na vrednost koja uključuje generisanje dokaza, procedure odlučivanja sadržane u modulima teorije beleže korake dedukcije u posebne objekte koji na kraju procesa mogu biti spojeni u jedinstven dokaz.

Rešavač CVC4 doneo je još jednu promenu. Format dokaza je zamenjen. Umesto LF formata koji su koristile prethodne verzije, CVC4 koristi LFSC. Više detalja sintakse ovog formata kao i njegove pojedinosti mogu se naći u [15]. Za potrebe ovog poglavlja dovoljno je reći da se dokazi u mnogim teorijama mogu predstaviti u ovom formatu što ga čini pogodnim za primenu u SMT rešavačima gde se neretko dokaz dobije kombinovanjem dokaza koji su generisani od strane različitih modula teorije.

5.3 HS format dokaza

Jezik SMT-LIB

Pre nego što predstavimo novi format dokaza, važno je reći da je sintaksa koju koristimo za zapis termova preuzeta iz SMT-LIB standarda [3]. On je proistekao iz istoimene inicijative koja je za cilj imala standardizaciju logičkog okvira, ulaznog i izlaznog jezi-

ka za SMT rešavače, kao i skupa relevantnih teorija. Osim navedenog, ova inicijativa je doprinela formiranju korpusa instanci standardizovanih teorija za potrebe lakšeg razvoja, evaluacije i poređenja SMT rešavača. Opšta sintaksa SMT-LIB izraza je veoma jednostavna i podseća na izraze u jeziku LISP. Zbog ovoga se ona može lako proširivati. Ovo omogućava da se formati dokaza lako mogu uklopiti u ovakvu sintaksu iako to nije eksplicitno definisano SMT-LIB standardom. Upravo ova ideja je iskorišćena za format HS dokaza u radu [10], te zato u ovom odeljku dajemo osnovni pregled SMT-LIB sintakse.

SMT-LIB je zasnovan na višesortnoj logici prvog reda, gde svaki funkcijski simbol iz signature ima zadat *rang* oblika $\tau_1 \times \tau_2 \times \ldots \times \tau_n \longrightarrow \tau$ (intuitivno, $\tau_1, \ldots, \tau_n$ su sorte argumenata, dok je τ sorta povratne vrednosti). Specijalno, ako je rang simbola oblika τ (tj. nema argumenata), tada je u pitanju simbol konstante i on sam predstavlja term sorte τ. Takodje, ako su $t_1, \ldots, t_n$ termovi sorti $\tau_1, \ldots, \tau_n$ redom, a f je simbol ranga $\tau_1 \times \tau_2 \times \ldots \times \tau_n \longrightarrow \tau$, tada je $(f t_1 \ldots t_n)$ term sorte τ. Termovi sorte Bool nazivaju se formule.

Iskazni veznici se u SMT-LIB standardu definišu kao funkcijski simboli `not`, `and`, `or`, $\Longrightarrow$, i sl., sa rangovima Bool $\longrightarrow$ Bool (za not) , odnosno Bool $\times$ Bool $\longrightarrow$ Bool, u slučaju binarnih veznika. Takođe, postoji simbol jednakosti = koji je definisan kao polimorfni funkcijski simbol ranga $\tau \times \tau \longrightarrow$ Bool za svaku sortu τ. Simboli true i false su konstante sorte Bool.

SMT-LIB standard, pored baznih termova i formula, podržava i kvantifikatore. Kvantifikovane formule su oblika

`(forall ((`x_1 `:` τ_1`) (`x_2 `:` τ_2`) ... (`x_n `:` τ_n`)) `t`)` ili

`(exists ((`x_1 `:` τ_1`) (`x_2 `:` τ_2`) ... (`x_n `:` τ_n`)) `t`)`

gde su $x_1, \ldots, x_n$ promenljive sorti $\tau_1, \ldots, \tau_n$ redom, a t je formula koja može sadržati ove promenljive (svaka promenljiva sorte τ je ujedno i term sorte τ).

Još jedna od važnih SMT-LIB konstrukcija je i `define-fun` pomoću koje se uvodi novi funkcijski simbol kao zamena za navedenu formulu. Oblik ove konstrukcije je

`(define-fun` f `((`x_1 `:` τ_1`) (`x_2 `:` τ_2`) ... (`x_n `:` τ_n`)) `τ t`)`

gde su $x_1, \ldots, x_n$ promenljive sorti $\tau_1, \ldots, \tau_n$ redom, t je term sorte τ koja može sadržati ove promenljive, dok je f novi funkcijski simbol koji se uvodi kao zamena za formulu t.

Zarad kompaktnijeg zapisa termova koji uključuju višestruko ponavljanje pojedinih podtermova, SMT-LIB standard podržava i let konstrukcije oblika

`(let ((`x_1 t_1`) (`x_2 t_2`) ... (`x_n t_n`)) `t`)`

gde su $x_1, \ldots, x_n$ promenljive koje se vezuju redom za termove $t_1, \ldots, t_n$ (sorte promenljivih su određene sortama odgovarajućih termova). Intuitivno, ova konstrukcija predstavlja

term koji se dobija istovremenom zamenom promenljivih $x_1, \ldots, x_n$ termovima $t_1, \ldots, t_n$ redom u termu t.

Format HS dokaza

Predloženi format dokaza je rezolucijski dokaz koji dokazuje nezadovoljivost polazne formule tako što izvodi praznu klauzu. U kontekstu HS dokaza, sve formule se posmatraju kao atomičke, tj. definišemo *uopšteni literal* koji može biti ili proizvoljna formula ili njena negacija (pri čemu koristimo meta-simbol $\neg$ da označimo negativni uopšteni literal, umesto objektnog simbola `not` koji se koristi za negaciju u SMT-LIB sintaksi). Takođe, pod klauzom u nastavku podrazumevamo *uopštenu klauzu*, koja predstavlja skup uopštenih literala koji se interpretira kao disjunkcija tih literala. U listovima stabla dokaza nalaze se ulazne formule i aksiome. Jedino pravilo rezonovanja koje se primenjuje je pravilo rezolucije definisano na sledeći način:

$$\frac{\{p\} \cup C_1 \quad \{\neg p\} \cup C_2}{C_1 \cup C_2}$$

gde je p formula, a C_1 i C_2 su klauze.

U kontekstu HS dokaza, ugrađeni funkcijski simboli `and, or,` $\Longrightarrow$, i sl., kao ni simboli interpretirani u teorijama nemaju predefinisano znacenje. Njima značenje daju aksiome koje čine jezgro teorije u kojoj se problem rešava.

Svaka aksioma (tačnije, aksiomatska shema) predstavlja parametrizovanu klauzu. Aksiome navodimo u obliku (*naziv_aksiome* $par_1 \ldots par_n$) : { $l_1\ l_2 \ldots l_m$}, gde su $par_1, \ldots, par_n$ parametri, a $l_1, \ldots, l_m$ literali klauze koji mogu sadržati te parametre. Na primer, zapis `(not-` p`):` {$\neg$`(not` p`),` $\neg p$} definiše aksiomu koja se zove `not-`, ima jedan parametar - formulu p, a data je klauzom {$\neg$`(not` p`),` $\neg p$}.

Termovi dokaza (koje ćemo označavati sa $prf, prf_1, prf_2, \ldots$) se definišu na sledeći način:

- svaka aksioma je term dokaza (npr. `(not+ p)`). Ovaj term dokaza dokazuje odgovarajuću klauzu iz definicije aksiome.
- za svaku formulu p koja predstavlja ulazno tvrđenje (tj. zadato na ulazu sa `(assert` p`)`) izraz `(assume` p`)` predstavlja term dokaza. Ovaj term dokaza dokazuje klauzu $\{p\}$ (tj. uvodi pretpostavku sa ulaza).
- ako su prf_1 i prf_2 termovi dokaza, a p formula, tada je i `(res` p prf_1 prf_2`)` takođe term dokaza. Ovaj term dokaza dokazuje klauzu koja nastaje rezolucijom klauza koje dokazuju dokazi prf_1 i prf_2 po uopštenom atomu p.

Dokaz nezadovoljivosti skupa formula $p_1, \ldots, p_n$ zadatih na ulazu je bilo koji term dokaza koji dokazuje praznu klauzu, a koji može sadržati termove oblika `(assume p)` isključivo za $p \in \{p_1, \ldots, p_n\}$.

Poddokazi i podtermovi. Tokom konstrukcije dokaza neretko se dešava da je potrebno koristiti isti deo dokaza nekoliko puta. Ovakav deo dokaza nazivamo *poddokaz*. Kako bismo zapis dokaza učinili kompaktnijim, poddokaze koji se ponavljaju možemo vezati za promenljivu koristeći sintaksu kao u sledećem primeru:

```
(let-proof ((C (res (not q) (assume (not q)) (not- q)))) prf)
```

Sada se promenljiva C može koristiti u nastavku dokaza prf, na bilo kom mestu gde se očekuje da stoji dokaz klauze $\{\neg q\}$. Primetimo da je upotrebljena sintaksa slična *let*-sintaksi za SMT-LIB termove.

Nadalje, format dokaza podržava i upotrebu `let` konstrukcije sa sintaksom identičnom kao u SMT-LIB standardu da veže promenljive za termove (dakle, obične SMT-LIB termove, ne termove dokaza) koji se koriste veći broj puta u dokazu. Ovo je važno kako bi se sprečio eksponencijalni rast veličine dokaza s obzirom da podtermovi imaju tendenciju da se pojavljuju veliki broj puta, pogotovo u dokazima koji dokazuju korektnost KNF transformacije velike ulazne formule.

Napomenimo da je za vezivanje termova dokaza i običnih termova neophodno koristiti dve različite ključne reči (`let-proof` i `let` respektivno). Ukoliko bi se u oba slučaja koristila ista ključna reč, tada bi simboli poput `res, and+,` itd. morali postati predefisani. Upotrebom dve različite ključne reči, parseru proveravača je jasno iz konteksta da li se radi o termu dokaza ili običnom termu.

Definisanje funkcija. Prilikom generisanja dokaza, neretko je poželjno imati pomoćne funkcije koje definiše rešavač i koje se ne pojavljuju kao deo ulaznog problema. U predloženom formatu dokaza, funkcije se definišu kao:

$$\texttt{((define-fun f ((}x_1\ \tau_1\texttt{) ...(}x_n\ \tau_n\texttt{)) }t\texttt{) }prf\texttt{)}$$

gde je f simbol ranga $\tau_1 \times \tau_2 \times \ldots \times \tau_n \longrightarrow \tau$, t term sorte τ koji sadrži promenljive $x_1, \ldots, x_n$ koje su sorte $\tau_1, \ldots, \tau_n$ redom. Navedeni term dokaza dokazuje istu klauzu kao i poddokaz prf. Poddokaz prf može koristiti funkcijski simbol f na isti način kao i sve ostale funkcijske simbole.

Leme.[2] Kako bi bilo olakšano izvođenje dokaza, ponekad je korisno navesti činjenicu bez njenog dokazivanja i koristiti je kao lemu. Ovo se često primenjuje ukoliko dokaz leme zahteva uvođenje novih aksioma (u cilju proširenja bazne teorije nad kojom se radi). Lemu definišemo na sledeći način:

$$\texttt{(oracle} \pm p_1 \ \texttt{...}\pm p_n \ \textit{Attributes}\texttt{)}$$

pri čemu ± označava + (literal) ili − (negaciju literala). *Attributes* je korisnički definisana lista parova (ključ, vrednost) u formi (: *key value*) gde je : *key* identifikator, a *value* proizvoljan izraz koji može biti i izostavljen.

Lema uvodi klauzu $\{(\neg)p_1, \ldots, (\neg)p_n\}$ gde je literal p_i negiran ako i samo ako mu prethodi znak − u definiciji leme.

Aksiome

Aksiome se u kontekstu HS dokaza koriste za davanje značenja funkcijskim simbolima date teorije, uključujući i logičke veznike. U nastavku sledi njihov spisak.

Aksiome logičkih veznika

U ovom odeljku navodimo aksiome za logičke veznike.

$$\texttt{true+} : \{\texttt{true}\}$$
$$\texttt{false-} : \{\neg\texttt{false}\}$$
$$\texttt{(not+} \ p_0\texttt{)} : \{\texttt{(not} \ p_0\texttt{)}, p_0\}$$
$$\texttt{(not-} \ p_0\texttt{)} : \{\neg\texttt{(not} \ p_0\texttt{)}, \neg p_0\}$$
$$\texttt{(and+} \ p_0 \ldots p_n\texttt{)} : \{\texttt{(and} \ p_0 \ldots p_n\texttt{)}, \neg p_0, ..., \neg p_n\}$$
$$\texttt{(and-} \ i \ p_0 \ldots p_n\texttt{)} : \{\neg\texttt{(and} \ p_0 \ldots p_n\texttt{)}, p_i\}$$
$$\texttt{(or+} \ i \ p_0 \ldots p_n\texttt{)} : \{\texttt{(or} \ p_0 \ldots p_n\texttt{)}, \neg p_i\}$$
$$\texttt{(or-} \ p_0 \ldots p_n\texttt{)} : \{\neg\texttt{(or} \ p_0 \ldots p_n\texttt{)}, p_0 \ldots p_n\}$$
$$\texttt{(=>+} \ i \ p_0 \ldots p_n\texttt{)} : \{\texttt{(=>}p_0 \ldots p_n\texttt{)}, (\neg)p_i\}, \quad p_i \text{ se negira kada je } i = n$$
$$\texttt{(=>-} \ p_0 \ldots p_n\texttt{)} : \{\neg\texttt{(=>}p_0 \ldots p_n\texttt{)}, \neg p_0, \ldots, p_n\}$$
$$\texttt{(=+1} \ p_0 \ p_1\texttt{)} : \{\texttt{(=} \ p_0 \ p_1\texttt{)}, p_0, p_1\}$$
$$\texttt{(=-1} \ p_0 \ p_1\texttt{)} : \{\neg\texttt{(=} \ p_0 \ p_1\texttt{)}, p_0, \neg p_1\}$$

[2] autori rada [10] ovde koriste naziv *oracle*, ali doslovni prevod „proročanstvo" nema mnogo smisla u kontekstu rada

(=+2 p_0 p_1) : {(= p_0 p_1), ¬p_0, ¬p_1}

(=-2 p_0 p_1) : {¬(= p_0 p_1), ¬p_0, p_1}

(xor+ (l_1) (l_2) (l_3)) : {(xor l_1), (xor l_2), ¬(xor l_3)}

(xor- (l_1) (l_2) (l_3)) : {¬(xor l_1), ¬(xor l_2), ¬(xor l_3)}

Za svaki logički operator, navedena je aksioma uvođenja (npr. and+) kao i aksioma eliminacije (npr. and-) koje se mogu iskoristiti da uvedu ili eliminišu odgovarajući operator pravilom rezolucije. Aksiome logičkih veznika and-, or+ i =>+ imaju još jedan dodatni parameter, broj i čija je vrednost prirodni broj iz skupa $[0, n]$, npr. (and- 0 p_0 p_1) predstavlja dokaz klauze {¬ (and p_0 p_1), p_0}.

Jednakost dva terma sorte Bool u SMT-LIB sintaksi odgovara logičkoj ekvivalenciji. Shodno tome, definišemo pravila uvođenja =+1 i =+2 kao i pravila eliminacije =-1 i =-2. Ove aksiome mogu biti primenjivane samo nad termovima koji su tipa Bool pri čemu proveravač dokaza mora proveriti da li je ispunjen uslov za primenu ovih aksioma.

U pravilima xor+ i xor-, l_1, l_2, l_3 predstavljaju liste termova u kojima je ukupan broj pojavljivanja svakog terma paran broj. Ukoliko je l_i lista koju čini samo jedan term, tada (xor l_i) predstavlja term l_i.

Aksiome nad kvantifikatorima

Aksiome nad kvantifikatorima su sledeće:

(forall+ ((x_1 τ_1) ... (x_n τ_n)) p) :
{(forall ((x_1 τ_1) ... (x_n τ_n)) p),
¬(let ((x_1 (choose (x_1 τ_1) (forall ((x_2 τ_2) ... (x_n τ_n)) (not p)))))
(let ((x_2 (choose (x_2 τ_2) (forall ((x_3 τ_3) ... (x_n τ_n)) (not p)))))
⋮
(let ((x_n (choose (x_n τ_n) (not p)))) p)...))}

(forall- ((x_1 t_1) ... (x_n t_n)) p) :
{¬(forall ((x_1 τ_1) ... (x_n τ_n)) p), (let ((x_1 t_1) ... (x_n t_n)) p)}

(exists+ ((x1 t1) ... (xn tn)) p) :

{(exists ((x1 τ1) ... (xn τn)) p), ¬(let ((x1 t1) ... (xn tn)) p)}

(exists- ((x1 τ1) ... (xn τn)) p) :

{¬(exists ((x1 τ1) ... (xn τn)) p),

(let ((x1 (choose (x1 τ1) (exists ((x2 τ2) ... (xn τn)) p))))

(let ((x2 (choose (x2 τ2) (exists ((x3 τ3) ... (xn τn)) p))))

⋮

(let ((xn (choose (xn τn) p))) p)...))}

Kvantifikatori su definisani koristeći operator choose koji predstavlja Hilbertov ε-operator. Naime (choose (x τ) p) predstavlja bilo koji term sorte τ za koji važi literal p (u kome figuriše x). Ukoliko takav term ne postoji, tada je povratna vrednost operatora choose proizvoljan term sorte τ. Uzimajući ovo u obzir, (exists ((x τ)) p) je ekvivalentno sa p[x -> (choose (x τ) p)], tj. u termu p zamenjujemo x sa odgovarajućim termom sorte τ za koji p važi (ako takvog terma nema, onda će ovo biti netačno, kao i exists).

U aksiomama forall- i exists+ sorte τ_i se implicitno određuju na osnovu tipa izraza t_i.

Aksiome EUF teorije

Aksiome jednakosti su sledeće:

(refl t) : {(= t t)}

(symm t0 t1) : {(= t0 t1), ¬(= t1 t0)}

(trans t0 ... tn) : {(= t0 tn), ¬(= t0 t1), ..., ¬(= tn−1 tn)}

(=+ t0 ... tn) : {(= t0 ... tn), ¬(= t0 t1), ..., ¬(= tn−1 tn)}

(=- i j t0 ... tn) : {¬(= t0 ... tn), (= ti tj)}

(distinct+ t0 ... tn) : {(distinct t0 ... tn), (= t0 t1), ..., (= t0 tn), ..., (= tn−1 tn)}

(distinct- i j t0 ... tn) : {¬(distinct t0 ... tn), ¬(= ti tj)} gde je $i \neq j$

(cong (f t0 ... tn) (f t0' ... tn')) :

{(= (f t0 ... tn) (f t0' ... tn')), ¬(= t0 t0'), ..., ¬(= tn tn')}

Aksiome =- i distinct- imaju još dva dodatna parametra, brojeve i i j čija je vrednost prirodni broj iz skupa $[0, n]$.

Preostale aksiome

$$(\texttt{ite1}\ p_0\ t_1\ t_2) : \{(\texttt{=}\ (\texttt{ite}\ p_0\ t_1\ t_2)\ t_1),\ \neg p_0\}$$

$$(\texttt{ite2}\ p_0\ t_1\ t_2) : \{(\texttt{=}\ (\texttt{ite}\ p_0\ t_1\ t_2)\ t_2),\ p_0\}$$

$$(\texttt{del!}\ t\ldots) : \{(\texttt{=}\ (\texttt{!}\ t\ldots)\ t)\}$$

$$(\texttt{expand}\ (\texttt{f}\ t_0 \ldots t_n)) : \{(\texttt{=}\ (f\ t_0 \ldots t_n)\ (\texttt{let}\ ((x_0\ t_0)\texttt{...}(x_n\ t_n))\ t))\}$$

gde je f definisana sa $(\texttt{define-fun f}\ ((x_0\ \tau_0)\ \ldots\ (x_n\ \tau_n))\ t)$

Aksiome `ite1` i `ite2` mogu biti primenjivane kako na termove koji su tipa Bool, tako i na termove koji su drugih tipova. Aksioma del! obezbeđuje da dodavanje anotacija SMT-LIB termu ne menja njegovo značenje dok aksioma `expand` omogućava dokaz korektnosti ekspanzije definicija funkcijskih simbola iz ulaznog SMT-LIB fajla (pomoću define-fun sintakse).

Aksiome linearne aritmetike

Kako bi se olakšalo, ali i opravdalo, rezonovanje u linearnoj aritmetici, potrebne su aksiome koje u sebi sadrže dodatne uslove koji koriste sabiranje i/ili množenje racionalnih brojeva i polinoma. Prikazaćemo ih u ovom odeljku.

Koristimo kanonsku SMT-LIB formu za numeričke konstante. Brojeve tipa Real zapisujemo u decimalnom obliku. Razlomci kod kojih su brojilac i imenilac celobrojne vrednosti pišu se kao npr. `(/ 1 3)`, dok se razlomci kod kojih su realne vrednosti prisutne u deljenju pišu u obliku `(/ (- 2.5) 3.8)`.

Polinomi se u SMT-LIB standardu zapisuju na prirodan način, kao zbirovi monoma, pri čemu su monomi proizvodi oblika $(\texttt{*}\ c\ t_1 \ldots t_n)$, gde je c numerička konstanta (koja može biti izostavljena u zapisu kada je $c = 1$ i $n > 0$), a $t_1 \ldots t_n$ su termovi koji nisu numeričke konstante i ne sadrže operatore + i $*$. Simbol $*$ takođe može biti izostavljen iz zapisa u slučaju kada za njim sledi samo jedan argument, pa tako monom $(\texttt{*}\ 1\ t)$ može biti zapisan kao $(\texttt{*}\ t)$, a potom kao t. Nekoliko primera polinoma su naredni izrazi: $(\texttt{+}\ x\ 1)$, $(\texttt{+}\ (\texttt{*}\ 2\ x\ x)\ x\ 1)$, $(\texttt{+}\ (\texttt{*}\ x\ x\ x)\ (\texttt{*}\ 3\ x)\ 5)$. Specijalno, konstantni polinomi se zapisuju odgovarajućom numeričkom ili racionalnom konstantom.

Polinom može biti tipa Real ili Int. Svi termovi od kojih je izgrađen polinom moraju biti istog tipa. U slučaju kada je polinom tipa Int, tada sve konstante koje sadrži takođe moraju bili celobrojne dok kod tipa Real takvo ograničenje ne postoji, tj. tip konstante je proizvoljan. Tip polinoma može biti promenjen iz Int u Real tako što se na svaki term primeni funkcija *to_real* i celobrojni koeficijenti pretvore u racionalne koeficijente dodavanjem .0 na njihov kraj.

Nakon što smo se upoznali sa osnovnim pojmovima i njihovim zapisom, navedimo aksiome linearne aritmetike nakon čega će najvažnije od njih biti objašnjene.

(poly+ (+ $g_1 \ldots g_n$) g) : {(= (+ $g_1 \ldots g_n$) g)} gde je $g = g_1 + \cdots + g_n$

(poly* (+ $g_1 \ldots g_n$) g) : {(= (* $g_1 \ldots g_n$) g)} gde je $g = g_1 * \cdots * g_n$

(farkas c_1 (<=? g_1 g_1') ... c_n (<=? g_n g_n')) :

{¬(<=? g_1 g_1'), ..., ¬(<=? g_n g_n')}

gde su c_i pozitivni celi brojevi,

suma $c_1(g_1 - g_1') + \ldots + c_n(g_n - g_n')$ je nenegativna konstanta,

<=? predstavlja <, <= ili =,

ako je prethodna suma jednaka nuli, tada bar jedan od literala mora biti <

(trichotomy t_1 t_2) : {(< t_1 t_2), (= t_1 t_2), (< t_2 t_1)}

(total t_1 t_2) : {(<= t_1 t_2), (< t_2 t_1)}

(total-int t_1 c) : {(<= t_1 c), (<= $c + 1$ t_1)}

gde je t_1 tipa Int, c je celobrojna konstanta zapisana u kanonskoj formi,

i $c + 1$ je ista celobrojna konstanta uvećana za jedan

(to_real g) : {(= (to_real g) g')}

gde je g' rezultat promene tipa polinoma g iz Int u Real

(>def t_1 t_2) : {(= (> t_1 t_2) (< t_2 t_1))}

(>=def t_1 t_2) : {(= (>= t_1 t_2) (<= t_2 t_1))}

(/def t_1 $t_2 \ldots t_n$) : {(= t_1 (* $t_2 \ldots t_n$ (/ t_1 $t_2 \ldots t_n$))), (= t_2 0),...,(= t_n 0)}

(-def t_1) : {(= (- t_1) (* (- 1) t_1))}

(-def t_1 $t_2 \ldots t_n$) : {(= (- t_1 $t_2 \ldots t_n$) (+ t_1 (* (- 1) t_2))...(* (- 1) t_1))}

(to_int-low t_1) : {(<= (to_real (to_int t_1)) t_1)}

(to_int-high t_1) : {(< t_1 (+ (to_real (to_int t_1)) 1.0))}

(abs-def t_1) : {(= (abs t_1) (ite (< t_1 0) (- t_1) t_1))}

(div-low t_1 t_2) : {(<= (* t_2 (div t_1 t_2)) t_1), (= t_2 0)}

(div-high t_1 t_2) : {(< t_1 (+ (* t_2 (div t_1 t_2)) (abs t_2))), (= t_2 0)}

(mod-def t_1 t_2) : {(= (mod t_1 t_2) (- t_1 (* t_2 (div t_1 t_2)))), (= t_2 0)}

(divisible-def c t_1) : {(= ((_ divisible c) t_1) (= t_1 (* c (div t_1 c))))}

Prve dve aksiome `poly+` i `poly-` mogu se koristiti da se pojednostave aritmetički termovi ulazne formule i imaju poduslov koji zahteva sabiranje, odnosno množenje, polinoma. Proces dobijenja ovog polinoma mora biti uključen u dokaz jer njegov SMT-LIB zapis nije jedinstven, npr. aksiome `(poly+ (+ x y x) (+ (* 2 x) y))` i `(poly+ (+ x y x) (+ y (* 2 x)))` su obe ispravne, ali dokazuju različite klauze. Rešavač može izabrati zapis i proveravač treba da dokaže da je g zaista ispravan zapis polinoma koji je zbir ili proizvod $g_1, \ldots, g_n$.

Najvažniji deo aritmetičkog dokaza je aksioma `farkas` koja dokazuje nezadovoljivost konjunkcije nejednakosti tako što proverava da li je njihova linearna kombinacija jednaka 0. Ukoliko je to slučaj, došlo je do kontradikcije, tj. data konjunkcija je nezadovoljiva. Opravdanje za ovakvu aksiomu nalazimo u Farkasovoj lemi, koju navodimo u nastavku.

Lema. *Skup nejednakosti $Ax \leq b$ je nezadovoljiv ako i samo ako postoji vektor $y = (c_1, \ldots, c_n)$ takav da je $y \geq 0$, $A^T y = 0$ i $b^T y < 0$.*

Dozvoljeno je da literali koji učestvuju u `farkas` aksiomi budu celobrojni i realni. Prilikom računanja njihovog zbira, celobrojni literali se implicitno pretvaraju u realne.

Aksiome `farkas`, `trichotomy`, `total` i `total-int` zajedno sa aksiomama logičkih veznika i aksiomama EUF teorije formiraju potpun skup aksioma u smislu da može biti dokazana svaka klauza koja je logička posledica datih pretpostavki u teoriji aritmetike.

Da bi se dokazale klauze koje sadrže negirane literale koristi se aksioma `farkas`, dok se preostale tri pomenute aksiome koriste da se negirani literali prevedu u pozitivne.

Ukoliko ulazni parameter nije polinom, pravila `poly+` i `poly-` u kombinaciji sa aksiomom `cong` mogu biti upotrebljena da se on pretvori u polinom.

Primeri HS formata dokaza

Nakon detaljnog predstavljanja HS formata dokaza kroz jezik koji se koristi za njegov zapis kao i aksioma koje se mogu koristiti prilikom generisanja dokaza, u ovoj sekciji navodimo nekoliko jednostavnih primera nezadovoljivih formula i njihovih dokaza.

Svaki primer se sastoji iz nekoliko delova: zapisa problema u SMT-LIB formatu, dokaza nezadovoljivosti koji SMT rešavač generiše kao i grafičkog prikaza dokaza kao stabla rezolucije. Primeri prate redosled kojim su navedene aksiome u prethodnim podsekcijama pa tako prvi primer predstavlja problem iskazne logike, drugi je primer EUF teorije dok treći primer ilustruje jednostavnu formulu linearne aritmetike.

Primer dokaza nezadovoljivosti za iskaznu logiku

Kako bi se ispitala nezadovoljivost skupa formula $\{q1, \neg q2, \neg q1 \wedge q2\}$ upotrebom SMT rešavača, neophodno je problem zapisati na jeziku SMT-LIB. Ukoliko želimo da bude generisan dokaz nezadovoljivosti, neophodno je navesti navesti opciju `(set-option :produce-proofs true)` na početku i `(get-proof)` na samom kraju kako bi dokaz bio ispisan. Takođe, potrebno je definisati i logiku u kojoj se radi. Oznaka koja se koristi kada se ispituje zadovoljivost iskaznih formula kao i formula EUF teorije je `QF_UF`. Uz sve navedeno, neophodno je definisati i iskazna slova p i q u skladu sa SMT-LIB standardom. Jedan od načina na koji može biti formiran ulazni fajl na SMT-LIB jeziku za SMT rešavače predstavljen je ispod.

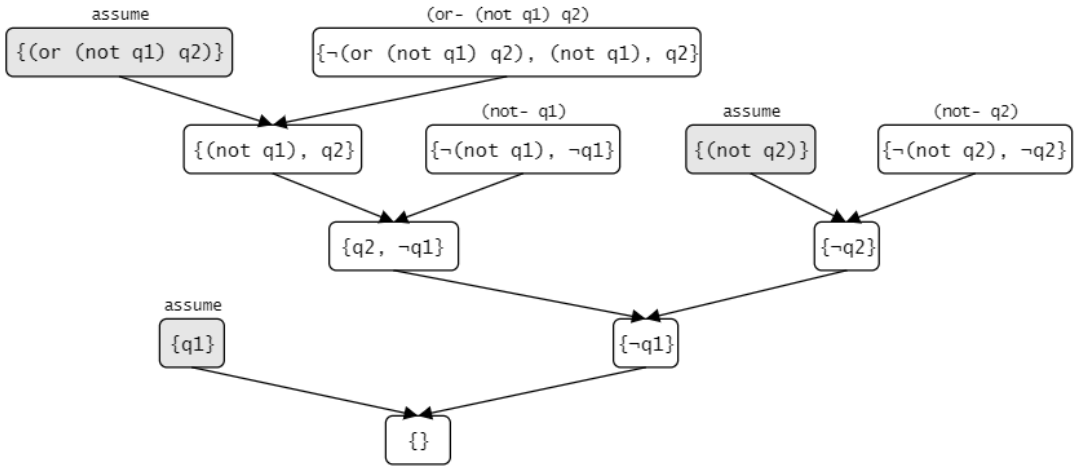
```
(set-option :produce-proofs true)
(set-logic QF_UF)
(declare-fun q1 () Bool)
(declare-fun q2 () Bool)
(assert q1)
(assert (not q2))
(assert (or (not q1) q2))
(check-sat)
(get-proof)
(exit)
```

Formatiran ulaz se predaje SMT rešavaču čiji rezultat rada je informacija o zadovoljivosti skupa formula praćena dokazom u slučaju da je skup nezadovoljiv.

```
unsat
(res q1 (assume q1)
    (res q2 (res (not q1) (res (or (not q1) q2)
    (assume (or (not q1) q2)) (or- (not q1) q2)) (not- q1))
    (res (not q2) (assume (not q2)) (not- q2))))
```

Ovako dobijen dokaz može potom biti predat na proveru nezavisno razvijenom proveravaču. Kao što možemo da primetimo, dokaz nije lak za praćenje. Stoga je na slici 5.1 predstavljen grafički prikaz dokaza. Vidimo da je dokaz drvoidnog oblika gde listovi predstavljaju skupove dobijene primenom aksioma na neku od formula polaznog proble-

ma dok unutrašnji čvorovi predstavljaju rezultat primene pravila rezolucije. Iznad svakog lista zapisana je aksioma čijom primenom je dobijen skup formula koji se u njemu nalazi. Uzastopnom primenom pravila rezolucije na kraju se stiže do prazne klauze, odnosno do toga da je polazni skup formula nezadovoljiv.



Slika 5.1: Grafički prikaz dokaza nezadovoljivosti skupa iskaznih formula

Primer dokaza nezadovoljivosti za formule EUF teorije

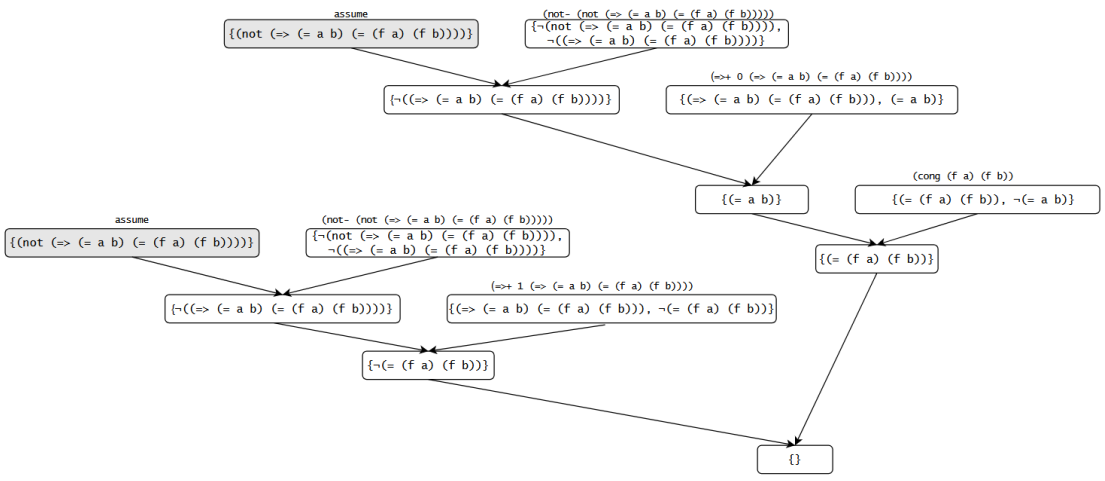
Primer na kome ilustrujemo generisanje dokaza za formule EUF teorije je formula $a = b \Rightarrow f(a) = f(b)$ za čiju negaciju se zna da je nezadovoljiva u EUF teoriji. U skladu sa SMT-LIB standardom definišemo najpre sortu, a potom funkciju f i konstante a i b. Ostali elementi poput definisanja logike, opcija za generisanje i ispis dokaza, su identični kao u prethodnom primeru. Ulazni podaci za SMT rešavač su sledeći:

```
(set-option :produce-proofs true)
(set-logic QF_UF)
(declare-sort S 0)
(declare-fun f (S) S)
(declare-fun a () S)
(declare-fun b () S)
(assert (not (=> (= a b) (= (f a) (f b)))))
(check-sat)
(get-proof)
(exit)
```

Kao rezultat, SMT rešavač ispisuje da je formula nezadovoljiva. Ovo je praćeno ispisom dokaza koji se nalazi ispod.

```
unsat
(res (= (f a) (f b))
    (res (= a b) (res (=> (= a b) (= (f a) (f b)))
    (=>+ 0 (=> (= a b) (= (f a) (f b))))
    (res (not (=> (= a b) (= (f a) (f b))))
    (assume (not (=> (= a b) (= (f a) (f b)))))
    (not- (not (=> (= a b) (= (f a) (f b)))))))
    (cong (f a) (f b)))
    (res (=> (= a b) (= (f a) (f b)))
    (=>+ 1 (=> (= a b) (= (f a) (f b))))
    (res (not (=> (= a b) (= (f a) (f b))))
    (assume (not (=> (= a b) (= (f a) (f b)))))
    (not- (not (=> (= a b) (= (f a) (f b)))))))))
```

U odnosu na prethodni primer, dokaz je vidno duži, kompleksniji i teži za razumevanje. Međutim, grafički prikaz dokaza na slici 5.2 umnogome olakšava njegovu interpretaciju. Primetimo da dokaz u sebi sadrži poddokaz koji se ponavlja dva puta. Radi se o podstablima čiji je koren čvor koji u sebi sadrži `{¬((=> (= a b) (= (f a) (f b))))}`.



Slika 5.2: Grafički prikaz dokaza nezadovoljivosti EUF formule

Primer dokaza nezadovoljivosti za formule linearne aritmetike

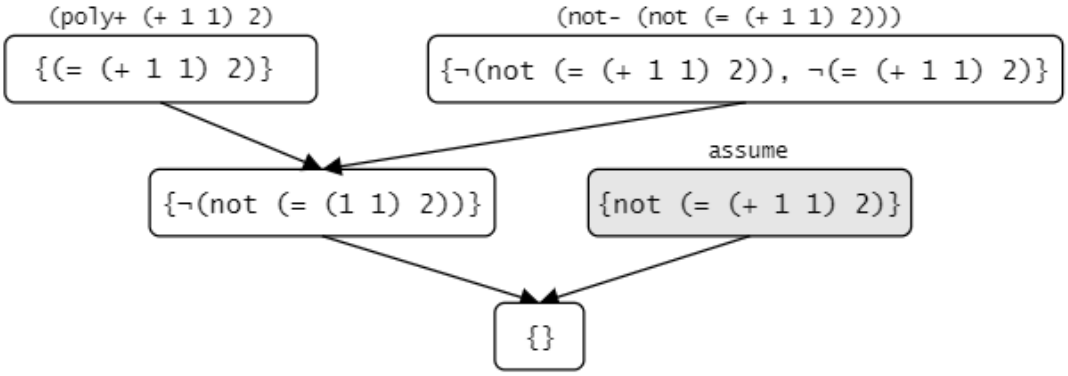
HS format dokaza za formule linearne aritmetike ilustrujemo na jednostavnom primeru formule `(not (= (+ 1 1) 2)`. Oznaka koja se koristi kako bi rešavači prepoznali da se radi o formuli linearne aritmetike je `QF_LIA`. SMT rešavaču se formula prosleđuje formatirana na sledeći način:

```
(set-option :produce-proofs true)
(set-logic QF_LIA)
(assert (not (= (+ 1 1) 2)))
(check-sat)
(get-proof)
(exit)
```

Kao rezultat rada SMT rešavača, dobija se:

```
unsat
(let ((eq (= (+ 1 1) 2)))
(res (not eq)
    (assume (not eq))
    (res eq (poly+ (+ 1 1) 2) (not- (not eq)))))
```

Primetimo da se u dokazu koristi konstrukcija `let` koja je objašnjena u poglavlju 5.3. Prilikom provere dokaza, svako pojavljivanje imena `eq` menja se sa `(= (+ 1 1) 2)` što se vidi i na grafičkom prikazu dokaza ispod.



Slika 5.3: Grafički prikaz dokaza nezadovoljivosti formule linearne aritmetike

Glava 6

Implementacija i evaluacija

Ova glava je podeljena na dva poglavlja – Implementacija i Evaluacija. U prvom od njih je prikazana implementacija generisanja i izvoza dokaza u HS formatu za probleme koji su definisani kao iskazne formule i formule EUF teorije u okviru SMT rešavača ArgoSMT, dok su u drugom poglavlju predstavljeni rezultati evaluacije veličine dokaza, te vremena rada rešavača sa i bez generisanja dokaza.

6.1 Implementacija

Kao osnova za implementaciju generisanja dokaza upotrebljen je ArgoSMT rešavač [1]. Modifikacije koje su bile neophodne kako bi se obezbedio izvoz dokaza obuhvataju dodavanje novih komponenti i manje izmene nekih od postojećih. Ove izmene ne utiču na ispravnost rezultata rada rešavača, tj. izmene se ne tiču samih implementiranih algoritama već se ogledaju u proveri da li je uključena opcija generisanja dokaza i koracima koje to povlači. Modifikovana verzija ArgoSMT rešavača proširena tako da omogući izvoz dokaza u HS formatu je javno dostupna za preuzimanje na GitHub-u.[1]

Nove komponente predstavljaju fajlovi `data_structures/proofs.hpp` i `data_structures/proofs.cpp`. U njima se nalaze klase koje implementiraju generisanje dokaz. Svaka klasa je izvedena iz osnovne klase `proof_node` koja se nalazi u fajlu `src/expression_library/smtlib_api.hpp` i u kojoj su deklarisane apstraktne funkcije:

- `print_proof` koja ispisuje dokaz u odgovarajućem formatu u dati izlazni tok

[1] izmenjena verzija se nalazi na adresi: https://github.com/mr211071/argosmt-with-proofs

- `check_proof` koja proverava da li je dokaz ispravno generisan; ova funkcija se koristi za internu proveru dokaza u svrhu pronalaženja i ispravljanja grešaka tokom razvoja rešavača dok se prava provera obavlja korišćenjem nezavisno razvijenog proveravača[2]

Za svaku aksiomu predloženu HS formatom dokaza izvedena je po jedna klasa koja u sebi sadrži parametre u skladu sa definicijom aksiome. Imena klasa oslikavaju nazive aksioma, pa tako klasa koja predstavlja aksiomu `(not+ p)` nosi naziv `axiom_not_intro` dok ime `axiom_not_elim` nosi klasa koja odgovara aksiomi `(not- p)`. S obzirom da su aksiome tvrđenja za koja se veruje da su tačna, tako je vrednost koju vraća funkcija `check_proof` u ovim klasama uvek Bool vrednost `true`. Pored ovih klasa dokaza, postoje još dve – `assume_proof` i `resolution_proof`. Prva od njih predstavlja uvođenje ulaznih pretpostavki u dokaz. Kao što je rečeno ranije u opisu HS formata dokaza, svakoj formuli p koja je na ulazu data sa `(assert p)` odgovara term dokaza `(assume p)` koji dokazuje klauzu $\{p\}$. Stoga je potrebna klasa koji odgovara ovoj karakteristici formata kako bi bilo omogućeno korišćenje ulaznih podataka u samom dokazu nezadovoljivosti. Vrednost koju vraća njena funkcija `check_proof` je takođe `true`. Kako je HS format dokaza u osnovi rezolucijski dokaz, to je potrebna struktura koja ga predstavlja. Upravo je to klasa `resolution_proof`. Kada je klauza dobijena rezolucijom dve klauze, njen dokaz treba sadržati informaciju o literalu po kome je izvršena rezolucija, kao i dokaze klauza na koje je primenjeno pravilo rezolucije. Stoga su atributi klase `resolution_proof` literal po kome je izvršena rezolucija te pokazivači na dokaze klauza (u nastavku teksta poddokaz) umesto samih klauza. Funkcija `check_proof` svodi se na rekurzivnu proveru korektnosti poddokaza i proveru da li klauze koje su izvedene pomoću tih poddokaza mogu biti rezolvirane po datom literalu. Zbog strukture klase `resolution_proof`, funkcija `print_proof` će u sebi imati pozive funkcija za štampanje poddokaza pa će tako konačni ispis dokaza imati format

(res <ispis_literala> <ispis_prvog_poddokaza> <ispis_drugog_poddokaza>).

Kako bi se omogućilo efikasno ulančavanje dokaza tako da jedan dokaz u sebi može sadržavati pokazivače na druge, kreiran je tip `proof` kao `shared_ptr<proof_node>`.

SMT-LIB standard predviđa generisanje dokaza, te se opcija koja to omogućava u rešavaču zadaje dodavanjem `(set-option :produce-proofs true)` u zaglavlje ulaznog fajla. Kako bi generisani dokaz bio ispisan, neophodno je u samom ulaznom fajlu dodati i komandu `(get-proof)` nakon komande `(check-sat)`. Ukoliko je dodata opci-

[2]proveravač je dostupan na adresi:
https://ultimate.informatik.uni-freiburg.de/smtinterpol/online/proof.html

ja za generisanje dokaza, u samom rešavaču se promenljiva `_produce_proofs` tipa Bool postavlja na `true` i dostupna je iz svih komponenti rešavača, što je neophodno za korektno formiranje dokaza. Ukoliko rešavač prijavi `unsat` kao rezultat svog rada, to znači da je prilikom analize konflikta na nultom nivou kreirana prazna klauza. Dokaz pridružen ovoj klauzi je upravo dokaz nezadovoljivosti.

Dokaz se gradi tokom rada rešavača prilikom svakog kreiranja nove klauze te je tako bila neophodna izmena komponenti rešavača u kojima se menja skup klauza čija zadovoljivost se ispituje, u prvom redu klasa `src/solver/formula_transformer.cpp` i `src/solver/solver.cpp`. Klasa `formula_fransformer` sadrži funkcije koje vrše transformaciju ulaznih formula u ekvizadovoljive KNF formule primenjujući Cajtinovu transformaciju na sledeći način:

- za svaku podformulu q polazne formule uvodi se nova konstanta x_i
- posebno, uvodi se nova konstanta koja označava celu polaznu formulu
- polazna formula se transformiše tako da se svako pojavljivanje neke od podformula zameni novouvedenom konstantom za tu podformulu
- kreira se nova formula koja predstavlja konjunkciju svih ekvivalencija oblika $x_i \Leftrightarrow q$ i konstante koja je uvedena kao ime cele formule
- ovako nastala formula se pretvara u KNF prateći logičke zakonitosti

Ilustrujmo Cajtinovu transformaciju na primeru formule $((p \vee q) \wedge r) \Rightarrow (\neg s)$. Podformule i novouvedene promenljive su: $x_1 \Leftrightarrow \neg s$, $x_2 \Leftrightarrow p \vee q$ i $x_3 \Leftrightarrow x_2 \wedge r$. Uvodimo novi simbol x_4 za polaznu formulu i u njoj zamenjujemo podformule novim promenljivima. Dobija se $x_4 \Leftrightarrow (x_3 \Rightarrow x_1)$. Konjunkcija ovih formula daje:

$$x_4 \wedge (x_1 \Leftrightarrow \neg s) \wedge (x_2 \Leftrightarrow p \vee q) \wedge (x_3 \Leftrightarrow x_2 \wedge r) \wedge (x_4 \Leftrightarrow (x_3 \Rightarrow x_1))$$

Novodobijena formula se potom prezapisuje u KNF prateći logičke zakonitosti. Podformula $x_2 \Leftrightarrow p \vee q$ se u sledećih nekoliko koraka pretvara u KNF:

$$\begin{aligned} x_2 \Leftrightarrow p \vee q &\equiv (x_2 \Rightarrow (p \vee q)) \wedge ((p \wedge q) \Rightarrow x_2) \\ &\equiv (\neg x_2 \vee p \vee q) \wedge (\neg(p \vee q) \vee x_2) \\ &\equiv (\neg x_2 \vee p \vee q) \wedge ((\neg p \wedge \neg q) \vee x_2) \\ &\equiv (\neg x_2 \vee p \vee q) \wedge (\neg p \vee x_2) \wedge (\neg q \vee x_2) \end{aligned}$$

Na sličan način se i ostale podformule mogu prevesti u KNF. Nakon što se to učini za sve podformule, novodobijena formula je u KNF-u i ekvizadovoljiva je sa polaznom formulom. Kao što možemo da primetimo, prilikom Cajtinove transformacije, dolazi do kreiranja novih klauza koje je neophodno dokazati. Stoga se u funkcijama `top_level_cnf_transformation` i `cnf_transformation` čija implementacija se nalazi u fajlu `src/solver/formula_transformer.cpp` nakon svakog kreiranja nove klauze proverava tačnost promenljive `_produce_proofs` i kreira odgovarajući dokaz.

Kako što smo videli u primerima predstavljenim u poglavlju 5.3, može se ispitivati zadovoljivost skupa formula, a ne samo jedne. Formule su zadate na ulazu preko konstrukcije `assert`. Potrebno je svaku formulu iz tog skupa Cajtinovom transformacijom prevesti u KNF. Funkcija `top_level_cnf_transformation` svaku formulu iz ulaznog skupa prosleđuje funkciji `cnf_transformation` koja vrši transformaciju. Nakon izvršene transformacije, u funkciji `top_level_cnf_transformation` se kreira nova klauza sa novim imenom ulazne formule. Dokaz ove klauze je upravo term dokaza `assume`. Ovo je implementirano na sledeći način:

```
expression name;
cnf_transformation(expr, clauses, name);
clause * name_unit = new clause();
name_unit->push_back(name);
if(_produce_proofs){
    proof assume =
     std::shared_ptr<proof_node>(new proof_assume(expand_names(name)));
    name_unit->set_proof(assume);
}
clauses.push_back(name_unit);
```

`expr` predstavlja izraz uveden sa `assert` koji transformišemo u KNF, `name` preko reference dobija vrednost novouvedene konstante kojom se zamenjuje `expr`. Kako u dokazu klauze koju čini novo ime izraza ne možemo koristiti to ime, pozivamo funkciju `expand_names(name)` koji vraća izraz koji je imenovan imenom `name`, pri čemu se sva imena u tom izrazu rekurzivno razvijaju.

U funkciji `cnf_transformation` se uvode nova imena za podformule i vrši transformacija u KNF. Za izgradnju dokaza klauza koje nastaju tokom ovog procesa se koriste aksiome za uvođenje i eliminaciju iskaznih veznika. Ako pogledamo primer prezapisivanja formule $x_2 \Leftrightarrow p \vee q$ koji je naveden iznad, vidimo da se formula koja je nastala na kraju sastoji od tri klauze koje mogu biti zapisane kao skupovi $\{\neg x_2, p, q\}$, $\{x_2, \neg p\}$ i $\{x_2, \neg q\}$. Ukoliko uporedimo ove skupove sa skupovima koji se dobijaju prilikom pri-

mene aksioma za uvođenje i eliminaciju disjunkcije, vidimo da prvom skupu odgovara primena aksiome `(or- p q)`, drugom `(or+ 0 p q)` dok trećem odgovara `(or+ 1 p q)`. Analogno se razmatraju i ostale kombinacije logičkih veznika. Način na koji je izvršena implementacija za disjunkciju je sledeći:

```
unsigned i = 0;
for(expression_vector::const_iterator it = op_names.begin(),
    it_end = op_names.end(); it != it_end; ++it)
{
    clause * short_clause = new clause();
    short_clause->push_back(name);
    short_clause->push_back(negate_name(*it));
    if(_produce_proofs){
        proof or_intro =
         std::shared_ptr<proof_node>(new axiom_or_intro(expand_names(expr), i));
        short_clause->set_proof(or_intro);
        i++;
    }
    clauses.push_back(short_clause);
    long_clause->push_back(*it);
}
long_clause->push_back(not_name);
if(_produce_proofs){
   proof or_elim =
    std::shared_ptr<proof_node>(new axiom_or_elim(expand_names(expr)));
   long_clause->set_proof(or_elim);
}
clauses.push_back(long_clause);
```

Kao i u prethodnom delu koda, i ovde `expr` predstavlja izraz koji se transformiše u KNF. `op_names` je vektor Cajtinovih imena kojima su zamenjeni njegovi podizrazi.

U fajlovima `src/solver/solver.hpp` i `src/solver/solver.cpp` je definisana klasa `solver`. Njena funkcija `solve` predstavlja implementaciju CDCL(T) algoritma. SAT rešavač koji se koristi za propagaciju jediničnih klauza implementiran je u klasi `clause_theory_solver` dok je T-rešavač za EUF teoriju implementiran u sklopu klase `euf_theory_solver`. Kako su oba rešavača potomci klasa koje su izvedene iz bazne klase `theory_solver`, to oba implementiraju funkciju `check_and_propagate` koja je zadužena za jediničnu propagaciju kod SAT rešavača, odnosno za teorijsku propagaciju kod kod T-rešavača. U funkciji `solve` se uzastopno pozivaju funkcije `check_and_propagate`

za oba rešavača sve dok ne bude prijavljen konflikt. Kada dodje do konflikta, pristupa se njegovom objašnjenju. Negacije literala klauze koja je izazvala konflikt smeštene su u vektoru `_explained`. Objašnjava se literal po literal počevši od prvog neobjašnjenog literala. Utvrđuje se da li je on potekao iz konflikta izazvanog prilikom jedinične ili teorijske propagacije i u skladu sa tim poziva funkcija `explain_literal` odgovarajućeg rešavača. Svakom literalu je pridružena klauza koja predstavlja njegovo objašnjenje. Ovim klauzama je pridružen dokaz.

Funkcija `explain_literal` implementirana u klasi `clause_theory_solver` formira objašnjenje konfliktnog literala na osnovu klauza koja

Prilikom objašnjenja konflikta dolazi do kreiranja novih klauza. Kako klauza nastaje tokom proceca objašnjavanja pojedinačnih literala, tako se njen dokaz postepeno gradi. U izgradnji dokaza ne učestvuju literali, već cela konfliktna klauza. Ona se može dobiti iz vektora `_explained` negacijom njegovih neobjašnjenih elemenata pre samog početka objašnjavanja i potom koristiti za izgradnju dokaza. U funkciji `explain_literal` dolazi do objašnjenja literala tako što se pronalazi koja je njegova vrednost u trenutnoj parcijalnoj valuaciji, potom klauza koja je dovela do dodavanja literala u valuaciju, tj. njegov razlog. Potom se konfliktni literal rezolvira sa klauzom razloga i nastaje nova privremena klauza. Njen dokaz je rezolucijski dokaz po literalu koji se objašnjava i u kome učestvuju dokaz konfliktne klauze i dokaz klauze razloga. Pri objašnjenju sledećeg literala, ovaj dokaz će biti iskorišten za objašnjenje nove privremene klauze. Postupak se ponavlja sve dok svi literali nisu objašnjeni. Potom se proverava na kom nivou je došlo do konflikta. Ukoliko se radi o nultnom nivou, SMT rešavač završava sa radom i prijavljuje da je početni skup formula nezadovoljiv. Dokaz prazne klauze je upravo dokaz koji je generisan prilikom objašnjavanja konflikta. U protivnom, dolazi do poziva funkcije `backjump` koja na osnovu literala koji se nalaze u vektoru `_conflicting` formira novu klauzu, pridružuje joj prethodno izgrađeni dokaz i dodaje je u skup `_learnt_clauses`. Potom iz parcijalne valuacije uklanja sve vrednosti koji su definisane počevši od nivoa povratka pa nadalje kako bi prilikom sledećeg poziva funkcije `check_and_propagate` za oba rešavača bio ispravno formiran sledeći nivo odlučivanja. U funkciji `backjump` takođe dolazi do pražnjenja vektora `_explained` i `_conflicting` kako bi bila omogućena analiza iz početka bez bojazni da bi konflikti iz prethodnih iteracija imali uticaj na novi pokušaj rešavanja. Sa postupkom se nastavlja sve dok ili ne dođe do konflikta na nultom nivou ili ne bude pronađen model u slučaju zadovoljivosti skupa formula.

Kao što je ranije rečeno, obezbeđeno je kreiranje dokaza za novonaučene klauze i ukratko je opisan način na koji se to čini.

Kreiranje dokaza je implementirano u funkcijama `top_level_cnf_transformation` i `cnf_transformation`, te u klasama `solver`, `clause_theory_solver` i `euf_theory_solver` na svim mestima na kojima dolazi do kreiranja novih klauza.

6.2 Evaluacija

Za evaluaciju implementacije generisanja dokaza korišćen je računar sa 8GM RAM-a i AMD Ryzen 7 4800H procesorom sa 8 jezgara (16 niti) i frekvencijom 2.9GHz (4.2GHz Max Boost). Za iskazne formule ne postoji standardni skup instanci za evaluaciju, pa je za potrebe ilustracije generisanja nezadovoljivosti iskaznih formula kreirano nekoliko tautologija različite složenosti čija je negacija predata rešavaču na proveru. Pre nego što navedemo korištene tautologije, napomenimo da je nakon svakog primera navedeno ime koje će biti korišteno u nastavku teksta za svaku od instanci. Za evaluaciju su upotrebljene sledeće tautologije:

- $p \vee \neg p \quad (ex_1)$
- $\neg(p \wedge q) \Leftrightarrow (\neg p \vee \neg q) \quad (ex_2)$
- $(p \Rightarrow q) \Leftrightarrow (\neg q \Rightarrow \neg p) \quad (ex_3)$
- $((p \vee q) \wedge (p \Rightarrow r) \wedge (q \Rightarrow r)) \Rightarrow r \quad (ex_4)$
- $((p \wedge q) \Rightarrow r) \Leftrightarrow (p \Rightarrow (q \Rightarrow r)) \quad (ex_5)$

U tabeli 6.1 po vrstama je prikazano vreme rada rešavača u slučaju kada nije uključeno generisanje dokaza kao i u slučaju kada je generisanje dokaza uključeno (oznake t_1 i t_2 redom u tabeli), potom veličina ulaznog *.smt2* fajla kao i veličina dokaza eksportovanog u *.smt2* fajl za primere koji su navedeni kao imena kolona.

	ex_1	ex_2	ex_3	ex_4	ex_5
t_1	1.34e-04	1.48e-04	1.89e-04	1.61e-04	1.94e-04
t_2	2.17e-04	2.33e-04	2.88e-04	3.92e-04	3.54e-04
veličina ulaznog fajla	172B	226B	219B	249B	246B
veličina dokaza	268B	2.37KB	1.68KB	713B	1.97KB

Tabela 6.1: Rezultati evaluacije za iskazne formule

Pre početka evaluacije, laički se moglo pretpostaviti da će doći do usporenja rada rešavača kada je omogućeno generisanje dokaza s obzirom da formiranje, čuvanje i ispis dokaza takođe zahteva izvesno vreme. Direktno poređenje vremena izvršavanja na našem testnom skupu, otkriva da se faktor usporenja kreće između $1,52$ i $2,43$. Najmanje usporenje je zabeleženo za primer ex_3, dok je najveće bilo za primer ex_4.

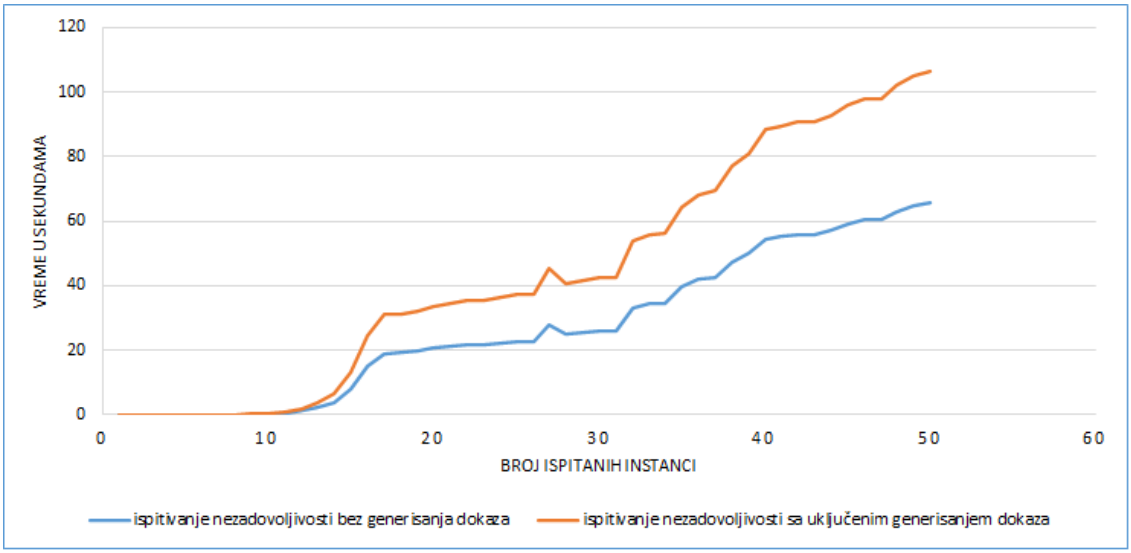
Kao što se i moglo očekivati, dokazi za instance veće veličine su čak i nekoliko puta veći od ulaznog fajla. Za primer ex_1 koji je i najjednostvaniji uvećanje je dvostruko dok je za ex_2 veće od desetostrukog. Zanimljivo je primetiti da za ulazne fajlove slične veličine, (ex_4 i ex_5), veličina dokaza značajno varira. Uzrok ovom nesrazmeru veličina dokaza možemp potražiti u razlici u složenosti formula čiju nezadovoljivost dokazuju. Ukoliko pogledamo strukturu formule ex_4, možemo da primetimo da u njoj ne postoji operator $\Leftrightarrow$ za razliku od formule ex_5. Prisustvo ovog operatora doprinosi značajnom uvećanju skupa klauza prilikom transformacije formule u KNF.

Međutim, bitno je napomenuti da veličina dokaza ne zavisi samo od veličine ulazne formule, već i od veličine konflikte klauze, odnosno od broja literala koje je neophodno objasniti. Svaki literal dodaje jedan nivo dokaza. Kako je dokaz drvoidne strukture, što je grafički prikazano kroz primere u 5.3, to slobodno možemo reći da njegova visina odgovara broju literala u konfliktnoj klauzi. Imajući ovo u vidu, sasvim je opravdano da dokazi nastali na osnovu manju konfliktne klauze budu manji. Ali ovde nije kraj. Naime, ukoliko se konflikt javi na nivou većem od nultog, dokaz koji se generiše u sebi sadrži dokaz klauze koja biva dodana u početni skup klauza prilikom povratnom skoka. Klauze koje su inicijalno deo skupa za proveru imaju dokaze koje slede iz aksioma HS dokaza i koje se generišu prilikom Cajtinove transformacije. Za razliku od ovih dokaza, dokaz naučene klauze je izgradjen prilikom objašnjavanja konfliktnih literala. Ovaj dokaz ne samo da zahteva dodatno vreme za generisanje već može i doprinosti izgradnji dokaza nezadovoljivosti samog problema, što doprinosi veličini istog. Uzevši i ovo razmatranje u obzir, nameće se zaključak da veličina dokaza zavisi od nekoliko faktora - složenosti formule, veličine konfliktne klauze, kao i veličine dokaza naučene klauze ukoliko do konflikta ne dodje na nultom nivou. Sve ovo treba imati u vidu i prilikom razmatranja evaluacije formula EUF teorije.

Evaluiranje generisanja dokaza nad instancama formula EUF teorije vršeno je nad javno dostupnim korpusima kojima se može pristupiti na adresi
`https://clc-gitlab.cs.uiowa.edu:2443/SMT-LIB-benchmarks/QF_UF`.

Imajući u vidu memorijska ograničenja implicitno određena hardverskim ograničenjima samog sistema kao i ograničenja samog rešavača, od velikog broja dostupnih instanci,

nasumično je odabrano 50 instanci. Na grafikonu prikazanom na slici 6.1 je predstavljeno vreme neophodno za rešavanje zadatog broja instanci bez i sa generisanjem dokaza nezadovoljivosti. Ukupne vrednosti vremena su date u tabeli 6.2.



Slika 6.1: Grafički prikaz rezultata evaluacije za formule EUF teorije

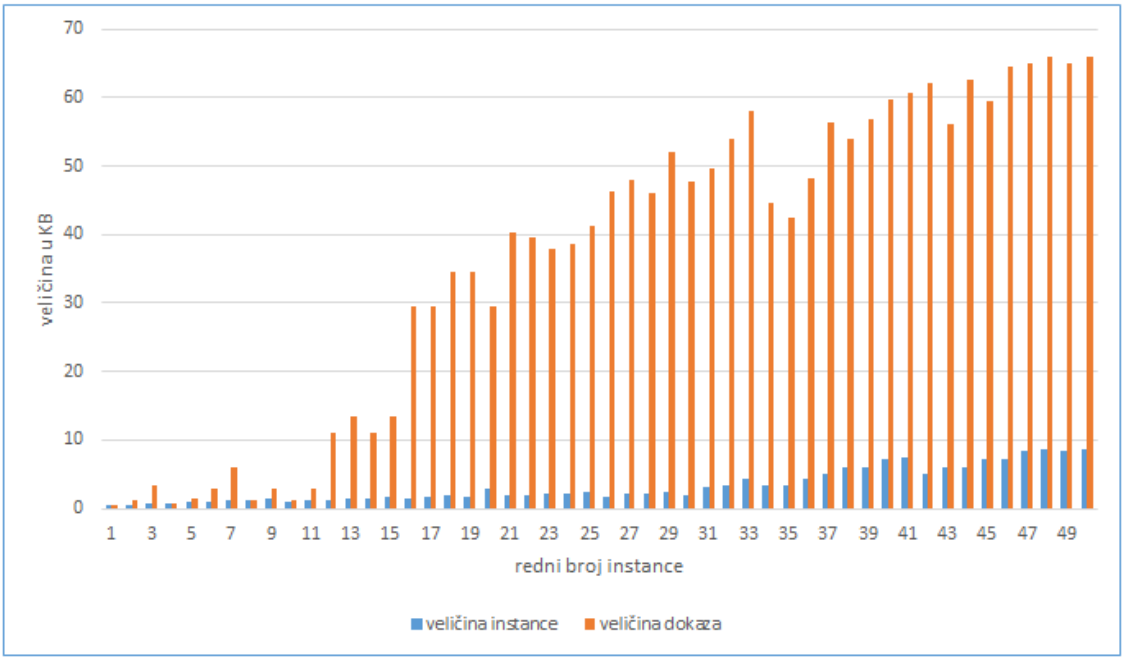
Rezultati merenja vremena izvršavanja su u skladu sa pretpostavkom da će generisanje dokaza dovesti do usporenja rada rešavača. Faktor usporenja je približno 1.6235. Međutim, ne treba uzeti ovu vrednost za empirijski dokazanu. Ona samo oslikava instance na kojima je test izvršen. Za drugačije odabrene instance kao i drugačiji računar na kome se testira, rezultat bi mogao biti značajno drugačiji. Zbog toga, dalji rad može biti nastavljen u pravcu standardizacije ne samo formata dokaza već i njegove implementacije kako bi obuhvatio što veći dijapazon primera formula EUF teorije.

	odabrane instance
ukupno vreme izvršavanja (bez generisanja dokaza)	65.56776735
ukupno vreme izvršavanja (sa generisanjem dokaza)	106.4492703

Tabela 6.2: Ukupna vremena izvršavanja

Grafikon na slici 6.2 predstavlja odnos veličine instance problema zapisanog u *.smt2* fajlu i veličine odgovarajućeg dokaza nezadovoljivosti koji je takođe smešten u *.smt2* fajl. Svaka instanca je označena rednim brojem, dok je veličina fajlova izdražena u kilobajtima. Posmatrajući grafik, može se videti da kod formula EUF teorije još više dolazi do izražaja nesrazmer veličine dokaza u odnosu na veličinu instance kojoj odgovara. Ako se kod iskaznih formula odnos veličina kretao oko faktora 2, ovaj faktor je prisutan samo

kod najmanjih formula EUF teorije. Što je formula veća i kompleksnija, to i njen dokaz raste, ponekad i desetostruko. Na odabranim primerima nije primećen eksponencijalni rast veličine dokaza u odnosu na veličinu instance.



Slika 6.2: Grafički prikaz odnosa veličina instance i njenog dokaza nezadovoljivosti

Sveukupno, rezultati evaluacije ukazuju na značajno usporenje rešavača kada je uključeno generisanje dokaza kao i na nezanemarljiv rast veličine dokaza u odnosu na veličinu polaznog problema. Stoga budući razvoj može ići u više pravaca od kojih jedan predstavlja rad na dodatnoj optimizaciji generisanja dokaza putem implementacije konstrukcije `let` u dokazu koja u trenutnoj verziji ne postoji, a za koju se pretpostavlja da bi mogla doprineti smanjenju veličine dokaza. Ipak, neka rezultati ove evaluacije ne umanje značaj koji HS format dokaza može imati u budućnosti.

Glava 7

Zaključak

Ovaj rad predstavlja retrospektivu formata dokaza za SAT i SMT rešavače, sa posebnim osvrtom na HS format dokaza. U neophodnoj meri su opisani logički pojmovi neophodni za razumevanje SAT i SMT problema. Takođe, kroz pseudokod su predstavljeni CDCL i CDCL(T) algoritmi koji se koriste za rešavanje ovih problema, te je čitalac upoznat sa osnovama načina rada SAT i SMT rešavača.

Posvećena je posebna pažnja HS formatu dokaza. Data je njegova logička osnova, opisana je njegova sintaksa, te načini prevođenja logičkih pravila na jezik SMT-LIB. Opisan je način formiranja dokaza prilikom rada SMT rešavača, navedene su aksiome koje se mogu koristiti kako nad ulaznim parametrima problema tako i nad svakim ispravnim izrazom. Dat je potpun spisak aksioma koji je podeljen na celine – aksiome logičkih operatora, aksiome za kvantifikatore, aksiome EUF teorije, aksiome linearne aritmetike. HS format dokaza je detaljno prikazan kroz tri primera koji ilustruju format dokaza i njegovu složenost. Takođe, dokazi nezadovoljivosti tih primera su prikazani grafički.

Predstavljeni su neki implementacioni detalji HS formata dokaza u ArgoSMT rešavaču. Evaluiran je odnos veličine instance problema i njenog dokaza. Izvršeno je poređenje vremena rada rešavača sa i bez generisanja dokaza, te utvrđeno usporenje.

Ipak, pored svega navedenog, ostaje još dosta prostora za rad, ponajviše na polju implementacije HS formata dokaza u ArgoSMT rešavaču kako bi se kompletiralo generisanje dokaza za probleme EUF teorije te obezbedilo generisanje dokaza nezadovoljivosti za probleme formulisane na jeziku linearne aritmetike. Ostaje razmatranje da li bi izvoz dokaza implementiran na drugačiji način, korišćenjem druge strukture podataka, doveo do ubrzanja rešavača. Takođe, kako bi se smanjila zavisnost od postojećih alata za proveru dokaza za koje ne može biti garantovana dostupnost u svakom trenutku, jedan pravac rada bi predstavljao i razvoj nezavisnog proveravača dokaza.

Bibliografija

[1] Milan Banković. ArgoSMT. on-line at: `https://github.com/milanbankovic/argosmt`.

[2] Clark Barrett, David Dill, and Aaron Stump. Checking satisfiability of first-order formulas by incremental translation to sat. pages 236–249, 07 2002.

[3] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The smt-lib standard: Version 2.6, technical report. 2017.

[4] Paul Beame, Henry Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *J. Artif. Intell. Res. (JAIR)*, 22:319–351, 07 2004.

[5] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery.

[6] Ashish Darbari, Bernd Fischer, and Joao Marques-Silva. Industrial-strength formally certified sat solving. 11 2009.

[7] Evguenii Goldberg and Yakov Novikov. Verification of proofs of unsatisfiability for cnf formulas. pages 10886–10891, 01 2003.

[8] Marijn Heule, Warren Hunt, and Nathan Wetzler. Trimming while checking clausal proofs. pages 181–188, 10 2013.

[9] Marijn Heule, Warren Hunt, and Nathan Wetzler. Verifying refutations with extended resolution. pages 345–359, 06 2013.

[10] Jochen Hoenicke and Tanja Schindler. A simple proof format for SMT. volume 3185 of *CEUR Workshop Proceedings*, pages 54–70. CEUR-WS.org, 2022.

[11] Zhang Lintao and Sharad Malik. Validating sat solvers using an independent resolution-based checker: Practical implementations and other applications. volume 1, pages 10880–10885, 01 2003.

[12] João Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 133–182. IOS Press, 2021.

[13] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving sat and sat modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll(t). *J. ACM*, 53:937–977, 01 2006.

[14] Aaron Stump, Clark Barrett, and David Dill. Cvc: a cooperating validity checker. 06 2002.

[15] Aaron Stump, Duckki Oe, Andrew Reynolds, Liana Hadarean, and Cesare Tinelli. Smt proof checking using a logical framework. *Formal Methods in System Design*, 42, 02 2013.

[16] Nathan Wetzler, Marijn Heule, and Warren Hunt. Drat-trim: Efficient checking and trimming using expressive clausal proofs. 07 2014.