

Shema dva posmatrana literala u DPLL SAT

rešavačima

Seminarski rad u okviru kursa
Automatsko rezonovanje
Matematički fakultet

Staša Đorđević 1007/2025

25. februar 2026.

Sažetak

U ovom radu analizirana je i implementirana shema dva posmatrana literala u okviru DPLL algoritma za rešavanje problema zadovoljivosti iskazne logike (SAT). Opisani su teorijski osnovi metode, njena uloga u efikasnoj detekciji jediničnih kluza i konflikata, kao i detalji implementacije u programskom jeziku Python. Shema dva posmatrana literala predstavlja standardnu optimizaciju savremenih SAT rešavača koja značajno smanjuje broj provera kluza tokom jedinične propagacije.

1 Uvod

Automatsko rezonovanje je oblast računarske nauke koja se bavi razvojem metoda i algoritama za automatsko izvođenje zaključaka iz formalnih logičkih sistema. Jedan od centralnih problema u ovoj oblasti je problem zadovoljivosti iskazne logike (SAT problem), koji se sastoji u određivanju da li postoji interpretacija kojom se formula iskazne logike može učiniti istinitom. SAT problem je fundamentalni NP-kompletan problem i predstavlja osnovu za mnoge primene u verifikaciji softvera, veštačkoj inteligenciji i optimizaciji. [2]

Jedna od najefikasnijih metoda za rešavanje SAT problema je DPLL algoritam (Davis–Putnam–Logemann–Loveland), koji predstavlja unapređenje klasičnog DP (Davis–Putnam) postupka. DPLL algoritam kombinuje sistematsko pretraživanje sa jednostrukim dedukcijama na osnovu jedinica (unit propagate) i detekcijom konflikata, čime omogućava efikasno ispitivanje velikih formula. Ključni deo DPLL algoritma predstavlja izbor literalata i način na koji se donose odluke tokom pretraživanja, jer od toga zavisi brzina pronalaženja rešenja ili dokaza da rešenje ne postoji. [5]

U okviru ovog rada fokus je stavljen na analizu i implementaciju *sheme dva posmatrana literala* unutar DPLL SAT rešavača. Ova shema predstavlja optimizaciju mehanizma jedinične propagacije i detekcije konflikata, čime se smanjuje broj potrebnih provera kluza tokom propagacije. Razumevanje i pravilna implementacija ove sheme može značajno poboljšati performanse DPLL algoritma, posebno na velikim i složenim SATinstancama.

Ostatak ovog rada organizovan je na sledeći način. U poglavljju 2 uvodimo osnovne definicije i notaciju iz iskazne logike, kao i formalni opis SAT problema.

Poglavlje 3 sadrži detaljan opis DPLL algoritma i sheme dva posmatrana literala. Poglavlje 4 opisuje implementaciju ove metode u izabranom programskom jeziku Python, uključujući ključne klase i funkcije. Na kraju, poglavlje 5 daje osvrt na rezultate rada i diskusiju o efikasnosti primenjene strategije.

2 Osnove

U ovom poglavlju uvodimo osnovne pojmove i notaciju iz iskazne logike koji će biti korišćeni u ostatku rada.

2.1 Sintaksa iskazne logike

Iskazna logika se bazira na *atomskim iskazima*, koji predstavljaju osnovne logičke promenljive, i na *logičkim veznicima* koji povezuju iskaze u složenije formule.

Formule iskazne logike definišu se rekurzivno:

- Iskazna slova i logičke konstante su iskazne formule.
- Ako su F i G formule, onda su i (F) , $\neg F$ (negacija), $F \wedge G$ (konjunkcija), $F \vee G$ (disjunkcija), $F \rightarrow G$ (implikacija) i $F \leftrightarrow G$ (ekvivalencija) formule.

2.2 Semantika iskazne logike

Semantika iskaznih formula definiše se pomoću pojma valuacije i interpretacije. Valuacija je funkcija $v : P \rightarrow \{0, 1\}$ koja svakom atomu iz skupa iskaznih slova P dodeljuje logičku vrednost.

Svaka (potpuna) valuacija v indukuje funkciju $I_v : F_P \rightarrow \{0, 1\}$ na skupu svih iskaznih formula nad P (u oznaci F_P), koju nazivamo interpretacija. Ova funkcija svakoj formuli pridružuje istinitosnu vrednost i definiše se rekurzivno na sledeći način:

- $I_v(p) = 1$ akko je $v(p) = 1$;
- $I_v(\top) = 1$, $I_v(\perp) = 0$;
- $I_v(\neg F) = 1$ akko je $I_v(F) = 0$;
- $I_v(F_1 \wedge F_2) = 1$ akko je $I_v(F_1) = 1$ i $I_v(F_2) = 1$;
- $I_v(F_1 \vee F_2) = 1$ akko je $I_v(F_1) = 1$ ili $I_v(F_2) = 1$;
- $I_v(F_1 \Rightarrow F_2) = 1$ akko je $I_v(F_1) = 0$ ili $I_v(F_2) = 1$;
- $I_v(F_1 \Leftrightarrow F_2) = 1$ akko je $I_v(F_1) = I_v(F_2)$.

2.3 Zadovoljivost i tautologija

Formula F je *zadovoljiva* ako postoji barem jedna interpretacija I takva da je $I(F) = 1$. Formula je *tautologija* ako je $I(F) = 1$ za sve moguće interpretacije I . S druge strane, formula je *nezadovoljiva* ako ne postoji nijedna interpretacija koja je čini istinitom. [1]

2.4 SAT problem

Problem zadovoljivosti iskazne logike (SAT problem) je zadatak određivanja da li je data formula F zadovoljiva. SAT problem je NP-kompletan i predstavlja osnovni problem u oblasti automatskog rezonovanja. [4]

U praksi, često se koriste formule u *konjunktivnoj normalnoj formi* (CNF), gde je formula predstavljena kao konjunkcija klauza, a svaka klauza je disjunkcija literala. Literal je ili atomski iskaz ili njegova negacija.

3 Opis metode

U ovom poglavlju biće opisan DPLL algoritam za rešavanje SAT problema, zajedno sa ključnim mehanizmima koji utiču na njegovu efikasnost. Posebna pažnja biće posvećena jediničnoj propagaciji (engl. unit propagate) i shemi dva posmatrana literala, koja predstavlja standardnu optimizaciju u savremenim SAT rešavačima.

3.1 DPLL algoritam

DPLL algoritam predstavlja rekurzivnu proceduru pretrage za određivanje zadovoljivosti formule u konjunktivnoj normalnoj formi (KNF). Za razliku od originalne DP procedure, DPLL ne eliminiše promenljive primenom rezolucije, već konstruiše parcijalnu valuaciju i sistematski ispituje moguće dodele vrednosti promenljivama.

Neka je F formula u KNF obliku, a M parcijalna valuacija. Algoritam se može opisati sledećim koracima:

1. Ako postoji klauza $C \in F$ koja je netačna u parcijalnoj valuaciji M , vraća se UNSAT.
2. Ako su sve promenljive iz F dodeljene i nijedna klauza nije netačna, vraća se SAT.
3. Ako postoji jedinična klauza, primenjuje se jedinična propagacija.
4. Ako postoji čist literal, dodeljuje mu se odgovarajuća vrednost.
5. U suprotnom, bira se nedefinisani literal i algoritam se rekurzivno poziva nad dve proširene valuacije: $M \cup \{l\}$ i $M \cup \{\neg l\}$.

3.2 Pseudokod DPLL algoritma

Struktura DPLL algoritma može se prikazati sledećim pseudokodom:

Algorithm 1 DPLL procedura

```
1: function DPLL( $F, M$ )
2: if postoji klauza netačna u  $M$  then
3:   return UNSAT
4: end if
5: if sve klauze zadovoljene u  $M$  then
6:   return SAT
7: end if
8: while postoji jedinična klauza  $C$  do
9:    $l \leftarrow$  jedinični literal iz  $C$ 
10:   $M \leftarrow M \cup \{l\}$ 
11:  if nastane konflikt then
12:    return UNSAT
13:  end if
14: end while
15: if postoji čist literal  $l$  then
16:    $M \leftarrow M \cup \{l\}$ 
17:   return DPLL( $F, M$ )
18: end if
19: Izaberite nedefinisani literal  $l$ 
20: if DPLL( $F, M \cup \{l\}$ ) = SAT then
21:   return SAT
22: end if
23: return DPLL( $F, M \cup \{\neg l\}$ )
```

3.3 Jedinična propagacija (Unit Propagate Rule)

Klasični DPLL. U naivnom DPLL algoritmu koji transformiše formulu, jedinična klauza je klauza sa tačno jednim literalom. Dodeljivanjem vrednosti koja zadovoljava taj literal, sve klauze koje ga sadrže se uklanjaju, a iz preostalih klauza se uklanja njegova negacija. Postupak se ponavlja dok postoje jedinične klauze. [5]

DPLL sa parcijalnom valuacijom. U implementacijama zasnovanim na parcijalnoj valuaciji M , klauza je jedinična ako su svi literali netačni, osim jednog koji je nedefinisani. Umesto transformacije formule, valuacija se proširuje sa tim literalom, a status klauza se ažurira u odnosu na novu dodelu.

Shema dva posmatrana literala. Da bi se izbegao obilazak svih klauza, koristi se shema dva posmatrana literala. U svakoj klauzi dva literala se aktivno prate. Ako jedan postane netačan, pokušava se zamena drugim literalom. Ako zamena nije moguća:

- klauza je konfliktna ako je i drugi literal netačan,
- klauza je jedinična ako je drugi literal nedefinisani.

Ova optimizacija značajno smanjuje broj provera potrebnih za detekciju konflikt-a i jediničnih klauza.

3.4 Eliminacija čistih literalala (Pure Literal Rule)

Klasični DPLL. Literal l je čist ako njegova negacija $\neg l$ ne postoji ni u jednoj klauzi formule. Dodeljivanjem vrednosti koja čini l istinitim, sve klauze koje ga sadrže se uklanaju, što pojednostavljuje formulu. [5]

DPLL sa parcijalnom valuacijom. Proverava se da li postoji literal l koji još nije dodeljen i čija negacija nije prisutna u nezadovoljenim klauzama. Ako postoji, valuacija se proširuje sa l , a klauze koje ga sadrže se automatski smatraju zadovoljenim. U modernim rešavačima zasnovanim na CDCL paradigm, eliminacija čistih literalala se često izostavlja jer doprinos efikasnosti postaje manji u poređenju sa jediničnom propagacijom i učenjem klauza.

3.5 Split (Grananje)

Kada više nije moguće primeniti jediničnu propagaciju niti eliminaciju čistih literalala, bira se nedefinisani literal l i razmatraju se dve mogućnosti: $M \cup \{l\}$ i $M \cup \{\neg l\}$. Ovaj korak uvodi rekurzivnu pretragu prostora svih mogućih valuacija i ključan je za sistematsko ispitivanje formule. Efikasnost algoritma u velikoj meri zavisi od strategije izbora literala za split. [5]

3.6 Shema dva posmatrana literalala

Naivna implementacija DPLL algoritma zahteva proveru svih klauza nakon svake izmene valuacije. Shema dva posmatrana literalala uvodi optimizaciju propagacije jediničnih klauza i detekcije konflikata. Svaka klauza aktivno prati dva literalala, a klauza se proverava samo ako jedan od njih postane netačan. Ako zamena nije moguća, detektuje se konflikt ili klauza postaje jedinična. [3]

3.7 Efekat optimizacije

Primena sheme dva posmatrana literalala dovodi do značajnog poboljšanja performansi. Jedinična propagacija se realizuje uz znatno manji broj provera. Ova tehnika je standardna komponenta modernih SAT rešavača i ključni je element njihove praktične efikasnosti. [3]

4 Implementacija

Implementacija DPLL procedure sa shemom dva posmatrana literalala realizovana je u programskom jeziku Python. Kod je organizovan u dva modula: `cnf_formula.py` i `dpll_solver.py`.

4.1 Organizacija koda

Modul `cnf_formula.py` sadrži klasu `CNFFormula` koja predstavlja CNF formulu i održava strukture podataka potrebne za primenu sheme dva posmatrana literalala:

- `clauses` – lista klauza, gde je svaka klauza predstavljena listom celih brojeva. Pozitivan broj k predstavlja literal p_k , a negativan broj $-k$ predstavlja literal $\neg p_k$,

- `watched_per_clause` – mapa koja preslikava indeks klauze u par njenih posmatranih literala (w_1, w_2) ,
- `literal_to_clauses` – mapa koja preslikava literal l u skup indeksa klauza u kojima je l posmatran.

Ove dve strukture zajedno omogućavaju efikasnu propagaciju: kada literal l postane netačan, direktno se pristupa samo onim klauzama u kojima je l posmatran, bez prolaska kroz celu formulu.

Modul `dpll_solver.py` sadrži klasu `DPLLSolver` koja prima objekat tipa `CNFFormula` i održava parcijalnu valuaciju M u vidu skupa literala. Ključne metode klase su:

- `unit_propagate(lit)` – po dodavanju literala lit u M , obilazi listu klauza u kojima je $\neg lit$ posmatran i za svaku takvu klauzulu traži alternativnog posmatranog literala. Ako alternativa postoji, ažuriraju se obe mape. Ako ne postoji, a drugi posmatrani literal je nedefinisan, propagira se taj literal kao jedinična klauza. Ako je i on netačan, prijavljuje se konflikt,
- `pure_literal()` – pronalazi čist literal pregledom nezadovoljenih klauza u odnosu na tekuću valuaciju M ,
- `pick_unassigned()` – bira prvi nedefinisani literal kao literal odlučivanja,
- `_save_state()` i `_restore_state(state)` – čuvaju i vraćaju stanje solvera, tj. parcijalnu valuaciju i obe mape posmatranih literala, što omogućava vraćanje unazad (backtracking) nakon neuspešne pretrage,
- `dpll()` – rekursivna implementacija DPLL procedure opisane u poglavljju 3,
- `solve()` – javni ulaz u solver koji obrađuje inicijalne jedinične i prazne klauze pre pokretanja rekurzije.

4.2 Napomena o povratku unazad

Kada se primenjuje shema dva posmatrana literala, povratak unazad zahteva poništavanje ne samo parcijalnih dodela već i promena u strukturama `watched_per_clause` i `literal_to_clauses`. U implementaciji se to rešava smanjnjem komplettnog stanja sva tri objekta pre svake odluke i vraćanjem tog stanja u slučaju neuspeha. Alternativni pristup bio bi eksplicitno praćenje izmena u obliku traga (engl. trail), ali za potrebe ovog rada odabran je jednostavniji pristup kopiranjem stanja.

4.3 Pokretanje programa

Za pokretanje programa potrebno je imati instaliran Python interpreter. Nisu potrebne dodatne biblioteke jer se koriste isključivo standardne Python biblioteke.

Program se pokreće iz datoteke `main.py` koja se nalazi u korenu repozitorijuma, pri čemu se putanja do ulaznog `.cnf` fajla prosleđuje kao argument komandne linije:

```
python3 main.py <putanja_do_fajla.cnf>
```

U repozitorijumu se nalaze tri primera ulaznih .cnf fajlova koji se mogu koristiti za testiranje. Na primer:

```
python3 main.py proba.cnf
```

Primer ulaznog fajla proba.cnf:

```
c ovo je neki komentar
p cnf 3 4
1 0
2 0
2 -3 0
3 1 0
```

Odgovarajući izlaz programa:

```
SAT
model: 1 2 -3
```

Metoda `solve` vraća listu literalata koji čine model ako je formula zadovoljiva, ili `None` u suprotnom. Metoda `print_result` ispisuje SAT ili UNSAT i, u slučaju zadovoljive formule, jedan njen model.

Ulazni fajlovi moraju biti u standardnom DIMACS CNF formatu. Prva nekomentarisana linija je zaglavje oblika `p cnf <broj_promenljivih> <broj_klauza>`, svaka naredna linija predstavlja jednu klauzulu kao listu literalata završenu nulom, a linije koje počinju slovom `c` tretiraju se kao komentari.

5 Zaključak

U ovom radu prikazana je shema dva posmatrana literalata kao optimizacija DPLL algoritma za rešavanje SAT problema. Opisani su teorijski osnovi metode, njena uloga u efikasnoj detekciji jediničnih klauza i konflikata, kao i detalji implementacije u programskom jeziku Python.

Ključna prednost sheme dva posmatrana literalata leži u tome što se provera klauza ne vrši globalno nakon svake dodele, već samo za one klauze u kojima je netačan literal bio posmatran. Time se značajno smanjuje broj potrebnih operacija tokom jedinične propagacije, što je naročito izraženo na velikiminstancama sa mnogo klauza.

Implementirana procedura prati pseudokod DPLL algoritma uz tri pravila: jediničnu propagaciju, eliminaciju čistih literalata i grananje sa vraćanjem unazad. Povratak unazad zahteva poništavanje promena i u strukturama posmatranih literalata, što je rešeno snimanjem i vraćanjem stanja pre svake odluke.

Mogući pravac daljeg razvoja je prelazak na CDCL (Conflict-Driven Clause Learning) paradigmu, koja uz shemu dva posmatrana literalata koristi i učenje novih klauza iz konflikata, što dodatno ubrzava pretragu. Savremeni SAT rešavači poput MiniSat i Glucose zasnovani su upravo na ovoj kombinaciji tehnika.

Literatura

- [1] M. Banković, *Automatsko rezonovanje – iskazna logika*, prezentacija sa kursa, Matematički fakultet Univerziteta u Beogradu, dostupno na: <https://poincare.matf.bg.ac.rs/~milan.bankovic/preuzimanje/ar/ar-iskazna-logika.pdf>.
- [2] F. Marić, *Flexible Implementation of SAT Solvers*, Matematički fakultet Univerziteta u Beogradu, dostupno na: <https://poincare.matf.bg.ac.rs/~filip/phd/sat-flexible-implementation.pdf>.
- [3] H. Jain i E. M. Clarke, “Efficient SAT Solving for Non-Clausal Formulas Using DPLL, Graphs, and Watched Cuts”, u: *Proceedings of the 46th Annual Design Automation Conference (DAC '09)*, ACM, 2009, str. 563–568.
- [4] A. Biere, M. Heule i H. van Maaren (ur.), *Handbook of Satisfiability*, IOS Press, 2009.
- [5] M. Davis i H. Putnam, “A Computing Procedure for Quantification Theory”, *Journal of the ACM*, 7(3), 1960, str. 201–215.