

Shema dva posmatrana literala u DPLL SAT rešavačima

Seminarski rad u okviru kursa
Automatsko rezonovanje
Matematički fakultet

Staša Đorđević 1007/2025

27. 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 klauza 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 klauza 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-kompletni 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 jediničnim propagacijama (unit propagate) i detekcijom konflikata, čime omogućava efikasno ispitivanje zadovoljivosti velikih formula. [5]

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

Ostatak ovog rada organizovan je na sledeći način. U poglavlju 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 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. Pregled osnovnih pojmova i definicija preuzet je iz [1].

2.1 Sintaksa iskazne logike

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

Formule iskazne logike definišu se rekursivno:

- 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 rekursivno 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 $I_v(F_1) = I_v(F_2)$.

Pored potpune valuacije, u praksi je često korisno definisati i **parcijalnu valuaciju**, koja dodeljuje istinitosne vrednosti samo nekim atomima, dok za ostale promenljive ostaje nedefinisana. Parcijalna valuacija se obično predstavlja kao skup literala koji su tačni; literal je netačan ako mu suprotan literal pripada valuaciji, a u suprotnom je nedefinisan.

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.

2.4 SAT problem

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

U praksi, često se koriste formule u *konjunktivnoj normalnoj formi* (KNF), 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 sistematski ispituje moguće dodele vrednosti promenljivama. Originalni DPLL algoritam menja formulu pri svakoj dodeli promenljive, dok u ovoj implementaciji koristimo unapređenu verziju zasnovanu na parcijalnoj valuaciji i shemi dva posmatrana literala, što omogućava ažuriranje samo relevantnih klauza bez transformacije celokupne formule. Pretraga se zasniva na proveru obe moguće istinitosne vrednosti svake promenljive u formuli i ažuriranju stanja formule kroz parcijalnu valuaciju.

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 klauga netačna u  $M$  then
3:   return UNSAT
4: end if
5: if sve klaube zadovoljene u  $M$  then
6:   return SAT
7: end if
8: while postoji jedinična klauga  $C$  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: Izaberi 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 i shema dva posmatrana literala

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

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

Shema dva posmatrana literala. Da bi se izbegao obilazak svih klauga, 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:

- klauga je konfliktna ako je i drugi posmatrani literal netačan,
- klauga je jedinična ako je drugi posmatrani literal nedefinisan.

Ova optimizacija značajno smanjuje broj provera potrebnih za detekciju konflikta i jediničnih klauga. Umesto obilaska svih klauga nakon svake izmene valuacije, proveravaju se samo one klaube u kojima se prati literal koji je upravo

postao netačan. Zbog toga shema dva posmatrana literala predstavlja standardnu komponentu savremenih SAT rešavača i ključni je element njihove praktične efikasnosti. [3]

3.4 Eliminacija čistih literala (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 uklanjaju, što pojednostavljuje formulu. [5]

DPLL sa parcijalnom valuacijom. Proverava se da li postoji literal l koji je nedefinisan u parcijalnoj valuaciji i čija negacija nije prisutna u nezadovoljenim klauzama. Ako postoji, valuacija se proširuje sa l , a klauze koje ga sadrže se automatski smatraju zadovoljenim.

3.5 Split (Grananje)

Kada više nije moguće primeniti jediničnu propagaciju niti eliminaciju čistih literala, bira se nedefinisan 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.

4 Implementacija

Implementacija DPLL procedure sa shemom dva posmatrana literala 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 literala:

- `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.

Takođe, u ovom modulu se nalazi i metod za čitanje formule iz DIMACS formata.

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 $\neg \text{lit}$ 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 parcijalnu valuaciju, što omogućava vraćanje unazad (backtracking) nakon neuspešne pretrage,
- `dpll()` – rekurzivna implementacija DPLL procedure opisane u poglavlju 3,
- `solve()` – javni ulaz u solver koji obrađuje inicijalne jedinične i prazne klauze pre pokretanja rekurzije.

4.2 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 literala 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 nekommentarisana linija je zaglavlje oblika `p cnf <broj_promenljivih> <broj_klauza>`, svaka naredna linija predstavlja jednu klauzulu kao listu literala završenu nulom, a linije koje počinju slovom `c` tretiraju se kao komentari.

5 Zaključak

U ovom radu prikazana je shema dva posmatrana literala 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 literala leži u tome što se proveru 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 velikim instancama sa mnogo klauza.

Implementirana procedura prati pseudokod DPLL algoritma uz tri pravila: jediničnu propagaciju, eliminaciju čistih literala i grananje sa vraćanjem unazad.

Mogući pravac daljeg razvoja je prelazak na CDCL (Conflict-Driven Clause Learning) algoritam, koji predstavlja značajno unapređenje u odnosu na klasični DPLL. CDCL zadržava tehnike već opisane u ovom radu - inkrementalnu izgradnju parcijalne evaluacije i shemu dva posmatrana literala, ali uvodi i dodatna poboljšanja: iterativnu implementaciju umesto rekurzivne, efikasne heuristike grananja, analizu konflikata zasnovanu na rezoluciji, nehnološko vraćanje unazad i učenje novih klauza iz konflikata u cilju sprečavanja sličnih konflikata u budućnost. [1] Zahvaljujući ovim unapređenjima, CDCL algoritmi postižu znatno bolje performanse u praksi i predstavljaju prirodan pravac daljeg razvoja implementacije prikazane u ovom radu.

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] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In DAC, str. 530–535, June 2001.
- [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.
- [6] Anurag Jain, *Watched Literals in a Finite Domain SAT Solver*, Master’s thesis, University of Minnesota, Faculty of the Graduate School, September 2008, dostupno na: https://d1wqtxts1xzle7.cloudfront.net/39095614/Thesis_Report-libre.pdf?1444523956&response-content-disposition=inline%3B+filename%3DWatched_Literals_in_a_Finite_Domain_SAT.pdf.