SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO-MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Matej Mihelčić

DAVIS-PUTNAMOV ALGORITAM

Diplomski rad

Zagreb, srpanj 2011.

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO-MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Matej Mihelčić

DAVIS-PUTNAMOV ALGORITAM

Diplomski rad

Voditelj rada:
Doc. dr. sc. Mladen Vuković

Zagreb, srpanj 2011.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

U ovom radu predstavljamo Davis-Putnamov algoritam za rješavanje NP-potpunog problema ispunjivosti logičke formule, kraće SAT (eng. satisfiability). Problem spada u klasu NP ako se može riješiti na nedeterminističkom Turingovom stroju u polinomnom vremenu. Za takve probleme postoji algoritam koji u polinomnom vremenu može testirati je li neko predloženo rješenje stvarno rješenje danog problema. NP-potpun problem je problem koji je sadržan u klasi NP i koji ima svojstvo da se svaki drugi problem iz klase NP može polinomno reducirati na njega. Nasuprot NP-potpunih problema su problemi rješivi nekim algoritmom u polinomnom vremenu na determinističkom Turingovom stroju i oni čine klasu P. Postavlja se pitanje: Kakav je odnos klasa P i NP? Odgovor na to pitanje je do danas nerješen problem. Njegovo rješavanje bi nam otkrilo je li moguće naći polinoman algoritam za rješavanje NP-potpunih problema ili takvi algoritmi jednostavno ne postoje.

SAT problem je trenutno jedan od najistraživanijih problema na polju računarstva, zbog čega postoje brojni algoritmi za nalaženje njegovog rješenja. Moderni algoritmi za rješavanje problema ispunjivosti logičke formule se mogu podijeliti u dvije velike klase:

- moderne varijante Davis-Putnam-Logemann-Loveland algoritma
- stohastičke algoritme lokalnog traženja.

SAT problem se javlja u raznim znanstvenim disciplinama, od verifikacije softvera i umjetne inteligencije do operacijskih istraživanja. Mnogi drugi NP-potpuni problemi, kao bojanje grafova, problemi planiranja i raspodjele poslova procesa, se mogu zakodirati kao SAT i zatim riješiti, zato je važno pronaći najbolje i najefikasnije algoritme za njegovo rješavanje.

Davis-Putnamov algoritam je algoritam za rješavanje SAT problema baziran na metodi rezolucije. Razvili su ga Martin Davis i Hilary Putnam 1960. godine, a kao njegovo poboljšanje 1962. godine Martin Davis, Hilary Putnam, George Logemann i Donald W. Loveland predstavljaju Davis-Putnam-Logemann-Loveland algoritam.

Algoritmi za rješavanje problema SAT su se razvijali od 1920. godine:

- 1920. metoda traženja rješenja preko tablica istinitosti (neučinkovito).

- 1960. klasični SAT solveri bazirani na Davis-Putnamovom algoritmu.
- 1990.-1992. uspješna implementacija stohastičkih algoritama za rješavanje teških SAT formula.
- 1993.-1994. nova hibridna stohastička strategija sa poboljšanom robusnosti i performansama.
- 1996.-1997. napredak u stohastičkim algoritmima, randomizirane sistematske metode pretraživanja.
- 1998.-2000. daljni napredak u stohastičkim algoritmima koristeći GLSM model (generalizirani strojevi lokalnog pretraživanja). Metode pretraživanja raznih algoritama se kombiniraju koristeći određeni mehanizam kontrole.
- 2001.-2004. Algoritmi dinamičkog lokalnog pretraživanja visokih performansi. Primjer takvog algoritma je self-tuning/adaptive WalkSat. (algoritam je sposoban sam odrediti svoje ulazne parametre u ovisnosti o veličini ulazne formule).

Cilj rada je dokazati korektnost Davis-Putnamovog algoritma te proučiti razna njegova poboljšanja, posebno Davis-Putnam-Logemann-Loveland algoritam. Davis-Putnamov algoritam spada u skupinu deterministčkih potpunih algoritama za rješavanje problema SAT. Uz tu skupinu postoje i stohastički algoritmi koji su uglavnom nepotpuni. U radu ćemo se osvrnuti na razne vrste stohastičkih solvera i detaljno opisati UnitWalk algoritam koji spada u tu skupinu. Predstavit ćemo generalne prednosti i mane algoritama iz svake skupine.

Poglavlje 1

Osnovne definicije i teoremi

U ovom poglavlju navodimo definicije i teoreme potrebne za razumijevanje i praćenje rezultata koje ćemo navesti u sljedećim poglavljima.

1.1 Osnovne definicije jezika logike sudova

Korištena notacija kao i definicije navedene u ovom odjeljku se mogu pronaći u [9].

Definicija 1.1.1. ***Alfabet logike sudova*** *je unija skupova A_1, A_2 i A_3. pri čemu je:*

$$A_1 = \{P_0,\ P_1,\ P_2,\ \dots\}$$

prebrojiv skup čije elemente nazivamo propozicionalne varijable;

$$A_2 = \{\neg\ , \wedge\ , \vee\ \rightarrow\ , \leftarrow\}$$

skup logičkih veznika;

$$A_3 = \{(,)\}$$

skup pomoćnih simbola (zagrade);

Definicija 1.1.2. ***Atomarna formula*** *je svaka propozicionalna varijabla. Pojam* ***formule*** *definiramo induktivno:*

1. *svaka atomarna formula je formula;*
2. *ako su A i B formule tada su i riječi $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$ i $(A \leftrightarrow B)$ također formule*
3. *riječ alfabeta logike sudova je formula ako je nastala primjenom konačno mnogo koraka 1. i 2.*

Definicija 1.1.3. *Svako preslikavanje sa skupa propozicionalnih varijabli u skup {0,1}, tj.* $I\colon \{P_0, P_1 \dots\} \to \{0, 1\}$*, nazivamo* ***totalna interpretacija*** *ili kratko* ***interpretacija****. Ako je preslikavanje definirano na podskupu skupa propozicionalnih varijabli tada kažemo da je to* ***parcijalna interpretacija****. Kažemo da je parcijalna interpretacija I* ***adekvatna*** *za formulu* $A(P_1, \dots, P_n)$ *ako je funkcija I definirana na* P_i *za sve* $i = 1, \dots, n$*.*

Definicija 1.1.4. *Neka je I interpretacija (totalna ili parcijalna). Ako se radi o parcijalnoj interpretaciji I smatramo da je I adekvatna za formule na kojima se definira njena vrijednost. Tada vrijednost interpretacije I na proizvoljnoj formuli definiramo induktivno:*

$$I(\neg A) = 1 \text{ ako i samo ako } I(A) = 0;$$

$$I(A \wedge B) = 1 \text{ ako i samo ako } I(A) = 1 \text{ i } I(B) = 1;$$

$$I(A \vee B) = 1 \text{ ako i samo ako } I(A) = 1 \text{ ili } I(B) = 1;$$

$$I(A \to B) = 1 \text{ ako i samo ako } I(A) = 0 \text{ ili } I(B) = 1;$$

$$I(A \leftrightarrow B) = 1 \text{ ako i samo ako } I(A) = I(B);$$

Ako alfabet sadrži i konstante $\top$ *i* $\bot$*, tada za svaku interpretaciju I definiramo* $I(\top) = 1$ *i* $I(\bot) = 0$*.*

Definicija 1.1.5. *Neka je S skup formula, a F neka formula. Kažemo da formula F* ***logički slijedi*** *iz skupa S, u oznaci* $S \models F$*, ako za svaku interpretaciju I, za koju je* $I(S) = 1$*, vrijedi* $I(F) = 1$*. Relaciju* $\models$ *nazivamo* ***relacija logičke posljedice****. Ako je S jednočlan skup, tj.* $S = \{A\}$*, tada činjenicu* $\{A\} \models B$ *zapisujemo i kao* $A \Rightarrow B$*.*

Definicija 1.1.6. *Kažemo da su formule A i B* ***logički ekvivalentne*** *i označavamo* $A \Leftrightarrow B$*, ako za svaku interpretaciju I vrijedi* $I(A) = I(B)$*.*

Definicija 1.1.7. *Za formulu F kažemo da je* ***ispunjiva*** *ako postoji interpretacija I takva da vrijedi* $I(F) = 1$*.*
Za formulu F kažemo da je ***oboriva*** *ako postoji interpretacija I takva da vrijedi* $I(F) = 0$*.*
Za formulu F kažemo da je ***valjana (tautologija, identički istinita)*** *ako je istinita za svaku interpretaciju.*
Za formulu F kažemo da je ***antitautologija*** *ili* ***identički neistinita*** *ako je neistinita za svaku interpretaciju.*

1.2 Normalne forme

Svi algoritmi koji rješavaju problem ispunjivosti logičkih formula kao ulazni parametar primaju formulu napisanu u konjunktivnoj normalnoj formi. Takav oblik formule omogućuje brojne optimizacije koje ubrzavaju nalaženje rješenja. U nastavku ćemo definirati normalne forme i dati osnovne rezultate koji pokazuju da svaka formula ima logički ekvivalentnu normalnu formu.

Definicija 1.2.1. *Atomarnu formulu i njezinu negaciju nazivamo **literal**. Formulu oblika $A_1 \wedge A_2 \wedge \cdots \wedge A_n$ nazivamo konjunkcija (A_i su proizvoljne formule). Formulu oblika $A_1 \vee A_2 \vee \cdots \vee A_n$ nazivamo disjunkcija. **Elementarna konjunkcija** je konjunkcija literala, a **elementarna disjunkcija** je disjunkcija literala. **Konjunktivna normalna forma** je konjunkcija elementarnih disjunkcija. **Disjunktivna normalna forma** je disjunkcija elementarnih konjunkcija.*

Neka je A neka formula, te B konjunktivna normalna forma i C disjunktivna normalna forma. Kažemo da je B konjunktivna normalna forma za A ako vrijedi $A \Leftrightarrow B$. Kažemo da je C disjunktivna normalna forma za A ako vrijedi $A \Leftrightarrow C$.

Sljedeći teorem je dokazan na kolegiju Matematička logika (vidi dokaz u [9])

Teorem 1.2.2. *Za svaku formulu logike sudova postoji konjunktivna i disjunktivna normalna forma.*

Prethodni teorem je jako važan. Garantira nam da možemo svaku formulu pretvoriti u ekvivalentnu konjunktivnu normalnu formu s kojom će naš algoritam raditi, međutim ostavlja nam preveliki izbor zato što mi za svaku formulu možemo generirati beskonačno mnogo konjunktivnih i disjunktivnih normalnih formi. Pošto se radi o algoritmima želimo da nam ulazni podaci (KNF forme) budu donekle jedinstveni.

Definicija 1.2.3. *Neka je $A(P_1, \ldots, P_n)$ konjunktivna normalna forma. Kažemo da je to **savršena konjunktivna normalna forma** ako u svakoj njezinoj elementarnoj disjunkciji svaka propozicionalna varijabla P_i nastupa točno jednom (s ili bez negacije), te su sve elementarne dijunkcije međusobno logički neekvivalentne.*
*Neka je $A(P_1, \ldots, P_n)$ disjunktivna normalna forma. Kažemo da je to **savršena disjunktivna normalna forma** ako u svakoj njezinoj elementarnoj konjunkciji svaka propozicionalna varijabla P_i nastupa točno jednom (s ili bez negacije), te su sve elementarne konjunkcije međusobno logički neekvivalentne.*

Prethodna definicija nam daje željeni rezultat.

Korolar 1.2.4. *Za svaku oborivu formulu postoji savršena konjunktivna normalna forma. Za svaku ispunjivu formulu postoji savršena disjunktivna normalna forma. Savršene forme su jedinstvene do na permutaciju varijabli u elementarnim disjunkcijama, odnosno konjunkcijama, te do na permutaciju elementarnih konjunkcija, odnosno disjunkcija.*

Prezentirane i dodatne informacije o normalnim formama mogu se vidjeti u [9].

Navodimo KNF algoritam koji pokazuje kako proizvoljnu formulu svodimo na konjunktivnu normalnu formu.

Ulaz: Proizvoljna formula F.

1. Sve podformule formule F oblika $F_1 \to F_2$ zamijenimo sa $(\neg F_1 \vee F_2)$ i sve podformule oblika $F_1 \leftrightarrow F_2$ sa $(\neg F_1 \vee F_2) \wedge (\neg F_2 \vee F_1)$. Postupak završavamo kada u F nema više podformula oblika $A \to B$ i $A \leftrightarrow B$ i prelazimo na drugi korak.

2. Rješavamo se svih dvostrukih negacija i primjenjujemo De Morganove zakone:

   $\neg\neg F_1$ zamijenimo sa F_1.

   $\neg(F_1 \wedge F_2)$ sa $(\neg F_1 \vee \neg F_2)$,

   $\neg(F_1 \vee F_2)$ sa $(\neg F_1 \wedge \neg F_2)$,

   Kada u formuli F nema više podformula takvih oblika prelazimo na treći korak.

3. Primjenjujemo svojstvo distributivnosti za $\wedge$ gdje je moguće tako da zamijenimo sve podformule oblika $(F_1 \vee (F_2 \wedge F_3))$ ili $((F_2 \wedge F_3) \vee F_1)$ sa $((F_1 \vee F_2) \wedge (F_1 \vee F_3))$.

Izlaz: Logički ekvivalentna formula u konjunktivnoj normalnoj formi.

Ilustrirat ćemo to na jednom primjeru:

Za pretvoriti formulu $(A_1 \wedge A_2 \wedge A_3) \vee (B_1 \wedge B_2 \wedge B_3)$ računamo:

$(A_1 \vee (B_1 \wedge B_2 \wedge B_3)) \wedge (A_2 \vee (B_1 \wedge B_2 \wedge B_3)) \wedge (A_3 \vee (B_1 \wedge B_2 \wedge B_3)) \Leftrightarrow (A_1 \vee B_1) \wedge (A_1 \vee B_2) \wedge (A_1 \vee B_3) \wedge (A_2 \vee B_1) \wedge (A_2 \vee B_2) \wedge (A_2 \vee B_3) \wedge (A_3 \vee B_1) \wedge (A_3 \vee B_2) \wedge (A_3 \vee B_3)$.

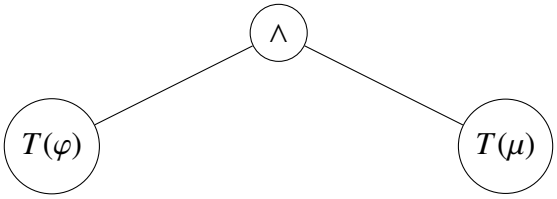
Navedeni algoritam ima eksponencijalnu složenost pošto pri primjeni svojstva distributivnosti dolazi do uzastopnog kopiranja varijabli. U koraku jedan algoritma dolazi do dupliciranja podformula pri svakoj eliminaciji ekvivalencije.

Na sreću, postoji i polinomni algoritam za pretvaranje proizvoljne formule u KNF.

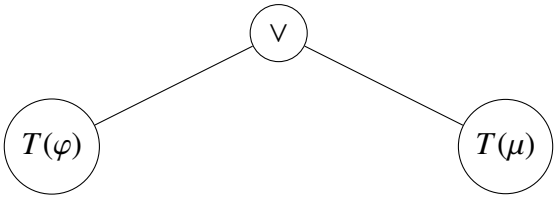
Definicija 1.2.5. *Za proizvoljnu formulu propozicionalne logike F definiramo stablo formule $T(F)$:*

(i) Ako se formula sastoji samo od propozicionalne varijable P, tada stablo formule sadrži samo jedan čvor P.

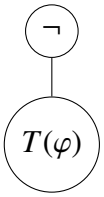
(ii) Ako su $F_1 = (\varphi \wedge \mu)$, $F_2 = (\varphi \vee \mu)$, $F_3 = (\neg\varphi)$ *formule i* $T(\varphi)$ *i* $T(\mu)$ *su pripadna stabla formula* φ *i* μ*, tada stabla formula koja odgovaraju* F_1, F_2 *i* F_3 *imaju sljedeći oblik:*



Stablo $T(F_1)$



Stablo $T(F_2)$



Stablo $T(F_3)$

U nastavku navodimo osnovne činjenice o *negacijskoj normalnoj formi (NNF)* [2].

Definicija 1.2.6. *Za formulu F koja ne sadrži* $\rightarrow$ *i* $\leftrightarrow$ *kažemo da je u negacijskoj normalnoj formi (NNF) ako i samo ako se svaka negacija javlja ispred propozicionalne varijable. Iz ovoga slijedi da formula ne smije sadržavati dvostruku negaciju. Sa NNF označavamo skup svih formula koje su u negacijskoj normalnoj formi.*

Formule u NNF možemo definirati i induktivno:

(1) Za svaku propozicionalnu varijablu A, A i $\neg A$ su formule u negacijskoj normalnoj formi.

(2) Ako su α i β formule u negacijskoj normalnoj formi, tada su i formule $(\alpha \wedge \beta)$ i $(\alpha \vee \beta)$ u negacijskoj normalnoj formi.

(3) Samo izrazi definirani pomoću (1) i (2) su formule u negacijskoj normalnoj formi.

Primjer 1.2.7. *Neka je* $F = (\neg(A \vee \neg B) \vee C) \vee \neg\neg D$ *zadana formula. Njoj ekvivalentna formula u NNF je* $F' = ((\neg A \wedge B) \vee C) \vee D)$.

Lema 1.2.8. *Za proizvoljnu formulu F postoji logički ekvivalentna formula u negacijskoj normalnoj formi.*

Dokaz. Lemu dokazujemo indukcijom po broju propozicionalnih varijabli formule.

1. Pretpostavimo da je F atomarna formula. Tada po definiciji negacijske konjunktivne forme sljedi da je F u NNF.

2. Pretpostavimo da za sve formule sa najviše n propozicionalnih varijabli postoji logički ekvivalentna formula u NNF.

3. Složena formula F sa više od n varijabli može biti oblika:

   - $F = \neg(F_1 \vee F_2)$. Primjenom de Morganovih pravila dobivamo logički ekvivalentnu formulu $(\neg F_1 \wedge \neg F_2)$, sada po pretpostavci indukcije postoje formule C_1 i C_2 koje su u NNF i logički ekvivalentne sa F_1 i F_2. Rekurzivnom primjenom de Morganovih pravila na formule $\neg C_1$ i $\neg C_2$ dobivamo logički ekvivalentne formule C_1' i C_2' koje su u NNF. Konačna formula $F' = C_1' \wedge C_2'$ je u NNF i logički je ekvivalentna sa formulom F.
   - $F = \neg(F_1 \wedge F_2)$. Analogno gornjem slučaju primjenom de Morganovih pravila dobivamo logički ekvivalentnu formulu $(\neg F_1 \vee \neg F_2)$, sada po pretpostavci indukcije postoje formule C_1 i C_2 koje su u NNF i logički ekvivalentne sa F_1 i F_2. Rekurzivnom primjenom de Morganovih pravila na formule $\neg C_1$ i $\neg C_2$ dobivamo logički ekvivalentne formule C_1' i C_2' koje su u NNF. Konačna formula $F' = C_1' \vee C_2'$ je u NNF i logički je ekvivalentna sa formulom F.
   - $F = F_1 \rightarrow F_2$, gdje su F_1 i F_2 dvije formule sa najviše n varijabli. Eliminiranjem $\rightarrow$ dobivamo $\neg F_1 \vee F_2$. Po pretpostavci indukcije za formule F_1 i F_2 postoje logički ekvivalentne formule u NNF. Označimo sa C_1 formulu u NNF koja je logički ekvivalentna formuli F_1. Na formulu $\neg C_1$ primjenimo de Morganove zakone i dobijemo formulu C_1' u NNF. Označimo s C_2 logički ekvivalentnu formulu formule F_2,tada $C_1' \vee C_2$ je formula u NNF koja je logički ekvivalentna s F .

- $F = F_1 \leftrightarrow F_2$, gdje su F_1 i F_2 dvije formule sa najviše n varijabli. Primjenimo prethodni korak na formule $F_1 \rightarrow F_2$ i $F_2 \rightarrow F_1$.

□

Proizvoljnu formulu ćemo prebacivati u KNF u tri koraka:

1. Iz proizvoljne formule logike sudova izbacujemo $\rightarrow$ i $\leftrightarrow$. Pod eliminacijom ekvivalencija smatramo zamjenu $(\alpha \leftrightarrow \beta)$ sa $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$ i zamjenu $(\alpha \rightarrow \beta)$ sa $(\neg\alpha \vee \beta)$.

   U ovom koraku moramo paziti na kopiranje podformula koje sadrže $\leftrightarrow$.

   Promotrimo formulu

   $$F = (A_n \leftrightarrow (A_{n-1} \leftrightarrow (A_{n-2} \leftrightarrow (\ldots(A_2 \leftrightarrow (A_1 \leftrightarrow A_0))\ldots)))).$$

   Ako definiramo **duljinu** formule F kao broj propozicionalnih varijabli koje sadrži, tada eliminacijom ekvivalencija iz F dobivamo formulu duljine

   $$6 \cdot 2^{n-1} - 2 \ (n \geq 1)$$

   stoga, ako u svakom koraku računamo i kopiramo podformule, naš algoritam će imati eksponencijalnu složenost.

   Nekoliko algoritama za pretvaranje proizvoljne formule u KNF se mogu vidjeti u [1].

   Primjer 1.2.9. *Promotrimo formulu* $F = A_3 \leftrightarrow (A_2 \leftrightarrow (A_1 \leftrightarrow A_0))$. *Eliminacijom ekvivalencija dobivamo:* $A_3 \rightarrow (A_2 \leftrightarrow (A_1 \leftrightarrow A_0)) \wedge (A_2 \leftrightarrow (A_1 \leftrightarrow A_0)) \rightarrow A_3$

   U sljedećem koraku dobivamo: $A_3 \rightarrow ((A_2 \rightarrow (A_1 \leftrightarrow A_0)) \wedge ((A_1 \leftrightarrow A_0) \rightarrow A_2)) \wedge (A_2 \rightarrow (A_1 \leftrightarrow A_0)) \wedge ((A_1 \leftrightarrow A_0) \rightarrow A_2) \rightarrow A_3$

   $(A_3 \rightarrow ((A_2 \rightarrow (A_1 \rightarrow A_0) \wedge (A_0 \rightarrow A_1)) \wedge ((A_1 \rightarrow A_0) \wedge (A_0 \rightarrow A_1)) \wedge A_2) \wedge ((A_2 \rightarrow ((A_1 \rightarrow A_0) \wedge (A_0 \rightarrow A_1)) \wedge ((A_1 \rightarrow A_0) \wedge (A_0 \rightarrow A_1)) \rightarrow A_2) \rightarrow A_3)$. *Vidimo da formula ima točno* $6 \cdot 2^{3-1} - 2 = 22$ *propozicionalnih varijabli, odnosno duljina formule je* 22.

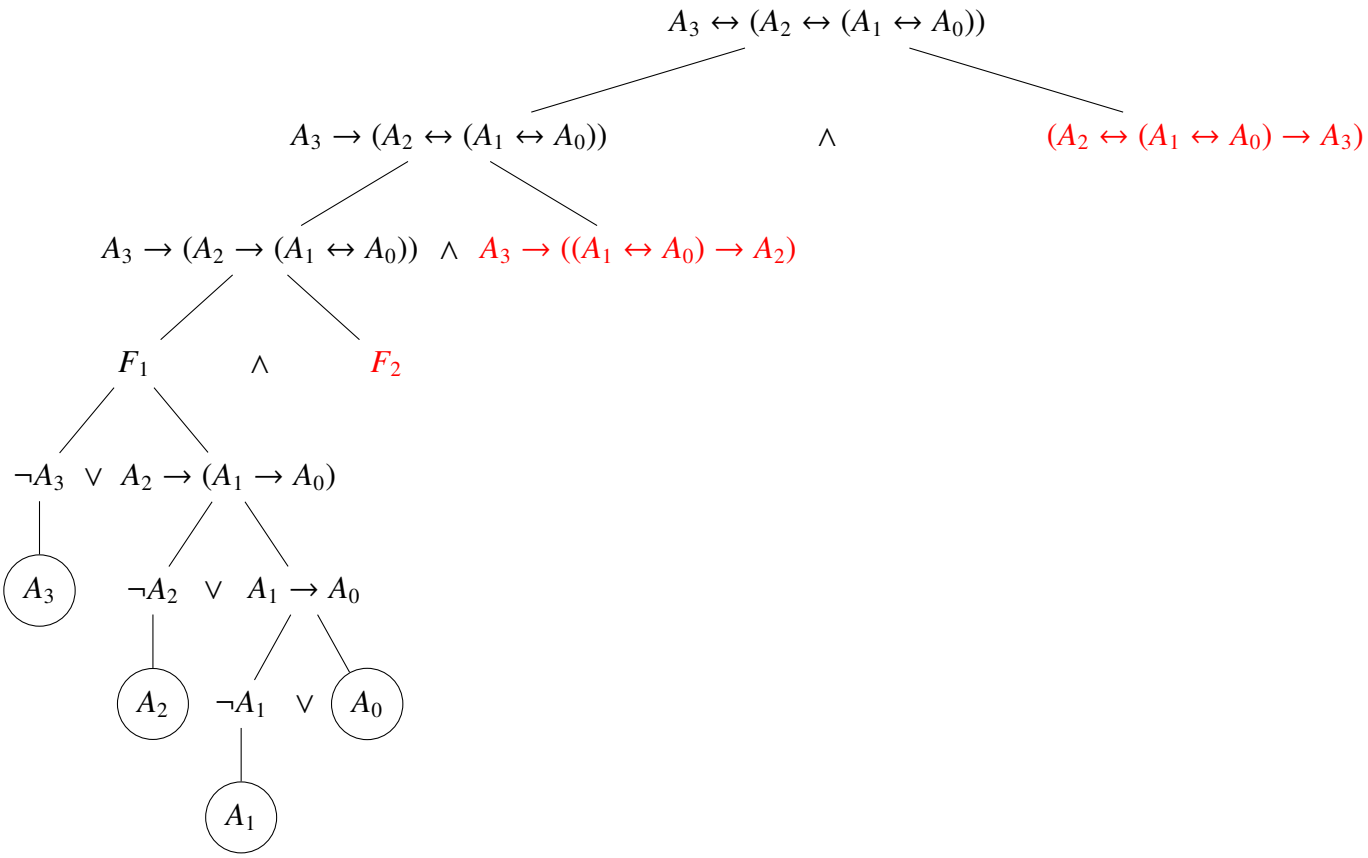
   Dajemo psudokod algoritma za pretvaranje proizvoljne formule propozicionalne logike u ekvivalentnu formulu bez $\leftrightarrow$ i $\rightarrow$ [2].

```
Algoritam Tree-SIMPLIFY.
Ulaz: Formula F reprezentirana pomoću stabla.
Izlaz: Ekvivalentna formula formuli F bez → i ↔.

  prop-formula* TREE-SIMPLIFY(prop-formula* α)
  {
    prop-formula* δ;
    switch(α){
          case α == β → γ:
           δ = ¬ TREE − SIMPLIFY(β) ∨ TREE − SIMPLIFY(γ);
          case α == β ↔ γ:
           δ = TREE−SIMPLIFY(β → γ)∧ TREE−SIMPLIFY(γ → β);
          case α == β ∧ γ:
           δ = TREE − SIMPLIFY(β) ∧ TREE − SIMPLIFY(γ);
           case α == β ∨ γ:
           δ = TREE − SIMPLIFY(β) ∨ TREE − SIMPLIFY(γ);
           case α == ¬ β:
           δ = ¬ TREE − SIMPLIFY(β);
           default: /*α je propozicionalna varijabla */
           δ = α;
          }
    return(δ);
  }
```

Primjer 1.2.10. *Demonstrirajmo rad algoritma na formuli:*

$$F = A_3 \leftrightarrow (A_2 \leftrightarrow (A_1 \leftrightarrow A_0)).$$

Nakon izvođenja algoritma dobivamo stablo:



$$* \; F_1 = A_3 \to (A_2 \to (A_1 \to A_0)).$$

$$* \; F_2 = A_3 \to (A_2 \to (A_0 \to A_1)).$$

Formule obojane crveno ne trebamo razmatrati. Ako smo izračunali formulu $F_1 = A \to B$ što je logički ekvivalentno sa $\neg A \vee B$, tada formulu $F_2 = B \to A$ možemo dobiti tako da iskoristimo račun koji smo radili za F_1 i zaključiti da je logički ekvivalentna formula od F_2 zapravo $\neg B \vee A$.

2. Formulu koja ne sadrži veznike $\to$ i $\leftrightarrow$ prebacujemo u negacijsku normalnu formu.

Definiramo algoritam za pretvaranje formule s navedenim svojstvima u NNF [2].

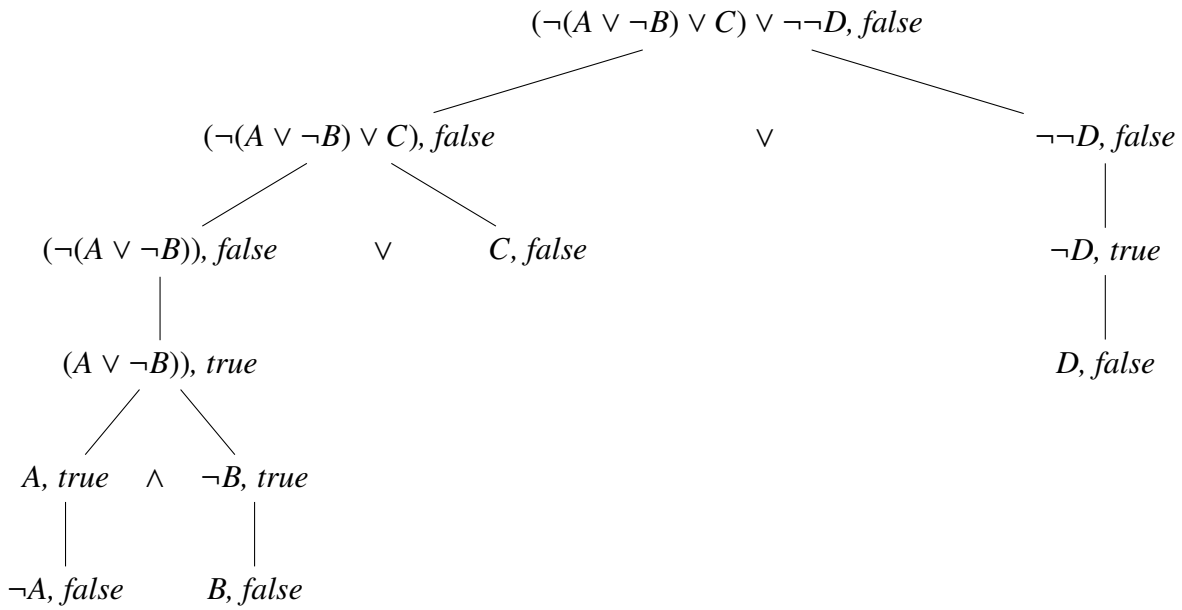
```
Algoritam Tree-NNF.
Ulaz: Formula F reprezentirana pomoću stabla koja ne sadrži
      → i ↔.
Izlaz: Ekvivalentna formula formuli F u NNF.

  prop-formula* REC-TREE-NNF(prop-formula* α, boolean negated)
  {
    prop-formula* δ;
    switch(α){
          case α == ¬ β):
           if(negated==true) negated=false;
           else(negated=true);
           δ =  REC − TREE − NNF(β, negated);
          case α == β  ∧  γ:
           if(negated==true)
           δ =  REC − TREE − NNF(β, negated)  ∨
                REC − TREE − NNF(γ, negated);
           else
           δ =  REC − TREE − NNF(β, negated)  ∧
                REC − TREE − NNF(γ, negated);
          case α == β  ∨  γ:
           if(negated==true)
           δ =  REC − TREE − NNF(β, negated)  ∧
                REC − TREE − NNF(γ, negated);
           else
           δ =  REC − TREE − NNF(β, negated)  ∨
                REC − TREE − NNF(γ, negated);
          default: /*α je propozicionalna varijabla */
            if(negated==true) δ = ¬ α;
            else δ = α;
          }
    return(δ);
  }
```

```
prop-formula* TREE-NNF(prop-formula* α)
{
     return(REC-TREE-NNF(α,false));
}
```

Demonstrirajmo rad algoritma na primjeru:

Primjer 1.2.11. *Neka je zadana formula* $F = (\neg(A \vee \neg B) \vee C) \vee \neg\neg D$.

Izvršavanjem algoritma TREE-NNF dobivamo:



Konačno dobivena formula je $((\neg A \wedge B) \vee C) \vee D$.

Vrijeme izvršavanja algoritma je linearno u odnosu na broj čvorova stabla ulazne formule. Svaki čvor posjećujemo točno jednom i za svaki posječeni čvor stvorimo najviše dva nova čvora.

3. Pretvaranje formule iz negacijske normalne forme u konjunktivnu normalnu formu. Ideja za polinomno pretvaranje formule iz NNF u KNF je sljedeća: Elementarnu disjunkciju $(\varphi \wedge \psi) \vee \pi$ zamijenimo elementarnom disjunkcijom $(\varphi \vee \neg A) \wedge (\psi \vee \neg A) \wedge (\pi \vee A)$, gdje je A nova propozicionalna varijabla formule. Varijablu A shvaćamo kao

supstituciju za podformulu $\varphi \wedge \psi$. Imamo: Formula $(\varphi \wedge \psi) \vee \pi$ je ispunjiva ako i samo ako su ispunjive formule

$$(A \leftrightarrow (\varphi \wedge \psi)) \wedge (A \vee \pi), \text{ što je ekvivalentno sa}$$

$$(A \vee \neg\varphi \vee \neg\psi) \wedge (\neg A \vee \varphi) \wedge (\neg A \vee \psi) \wedge (A \vee \pi), \text{ odnosno sa}$$

$$(\neg A \vee \varphi) \wedge (\neg A \vee \psi) \wedge (A \vee \pi)$$

Prije definiranja polinomnog algoritma za pretvaranje proizvoljne formule u KNF, definiramo dvije pomoćne funkcije koje ćemo koristiti [2]:

- **REC-DEGENERATE**
  Ova funkcija transformira disjunkcije proizvoljne klauzule u $\sigma_1 \vee (\sigma_2 \vee (\cdots \vee (\sigma_{n-1} \vee \sigma_n)\ldots))$, gdje niti jedan σ_i nije disjunkcija.
- **REC-REARRANGE**
  Ova funkcija preuređuje podformule disjunkcija koje je vratila funkcija **REC-DEGENERATE** na način da je prva podformula σ_{i_1} od $\sigma_{i_1} \vee (\sigma_{i_2} \vee (\cdots \vee (\sigma_{i_{n-1}} \vee \sigma_{i_n})\ldots))$ konjunkcija ako postoji konjunkcija u toj podformuli.

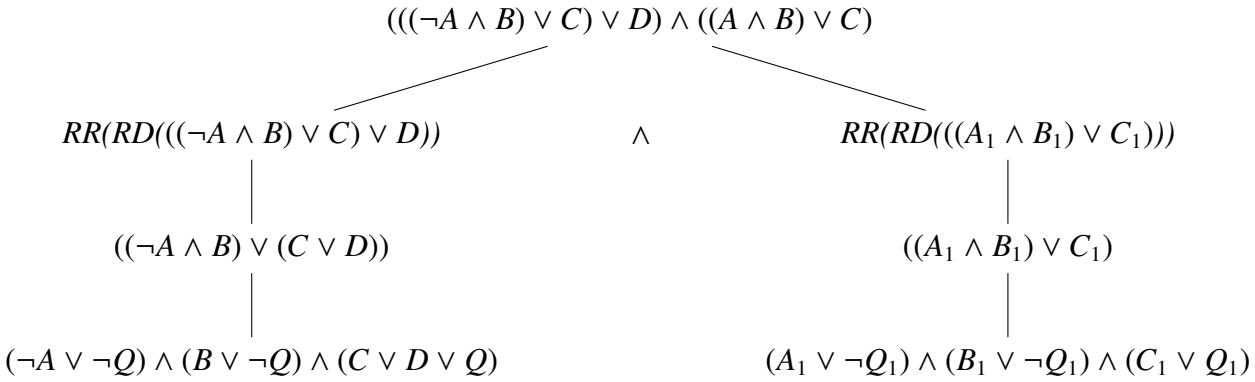
Obje funkcije se pozivaju samo na formulama koje sadrže disjunkciju. Duljina formule se ne mijenja.

```
Algoritam Tree-CNF.
Ulaz: Formula F reprezentirana pomoću stabla u NNF.
Izlaz: Ekvivalentna formula formuli F u KNF.

  prop-formula* REC-DEGENERATE(prop-formula* α) {
    switch(α){
          case α == ((β ∨  γ)  ∨  δ):
              α = β  ∨  (γ  ∨  δ);
           α =  REC − DEGENERATE(α);
          case α == β  ∨  γ: /*β nije disjunkcija*/
           α = β  ∨  REC − DEGENERATE(γ);
          default: /*α je literal, nema transformacije*/
          }
    return(α);  }
```

```
prop-formula* REC-REARRANGE(prop-formula* α)
{
  switch(α){
        case α == ((β ∧ γ) ∨ δ): /*α ima željenu strukturu
        case α == β ∨ γ: /*β nije konjunkcija*/
           γ = REC − REARRANGE(γ);
         if(γ == δ ∨ ε)
              α = δ ∨ (β ∨ ε);
         else
              α = γ ∨ β;
        default: /*α nije disjunkcija, nema transformacije*/
        }
  return(α);
}


  prop-formula* REC-TREE-CNF(prop-formula* α)
{
  switch(α){
        case α == (β ∧ γ):
             α = REC − TREE − CNF(β) ∧ REC − TREE − CNF(γ);
        case α == β ∨ γ:
             α = REC − REARRANGE(REC − DEGENERATE(α));
            if(α == (φ ∧ ψ) ∨ π){
                        A = formula(new_atom(new_name()));
                        α = (φ ∨ ¬ A) ∧ (ψ ∨ ¬ A) ∧ (π ∨ A);
                        α = REC − TREE − CNF(α);
                  }
        default: /*α je literal, nema transformacije*/
        }
  return(α);
}

 prop-formula* TREE-CNF(prop-formula* α)
{
initialise_new_names(α);
return(REC-TREE-CNF(α));
}
```

Gore navedeni algoritmi se mogu pronaći u [2].

Primjer 1.2.12. *Neka je zadana formula* $F = (((\neg A \wedge B) \vee C) \vee D) \wedge ((A_1 \wedge B_1) \vee C_1)$.

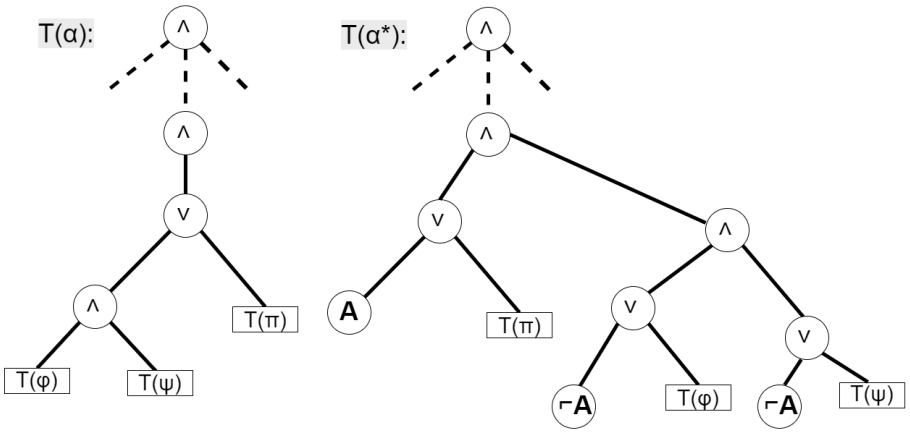


Dobivena formula u KNF je:

$$F' = (\neg A \vee \neg Q) \wedge (B \vee \neg Q) \wedge (C \vee D \vee Q) \wedge (A_1 \vee \neg Q_1) \wedge (B_1 \vee \neg Q_1) \wedge (C_1 \vee Q_1)$$

Teorem 1.2.13. *Algoritam TREE-CNF svaku formulu α duljine n pretvori u ekvivalentno ispunjivu formulu β u KNF duljine $O(n)$. Algoritam pri tome napravi $O(n^2)$ koraka.*

Dokaz. Teorem dokazujemo indukcijom po broju veznika $\wedge$ koji se nalaze na neodgovarajućim pozicijama u stablu ulazne formule. U stablu formule u NNF, veznik $\wedge$ se nalazi na neodgovarajućoj poziciji ako mu je veznik $\vee$ predak u stablu. Broj veznika $\wedge$ koji se nalaze na neodgovarajućim pozicijama je broj $\wedge$ čvorova u stablu koji su potomci $\vee$ čvora. Označimo taj broj sa $|F(\alpha)|$. Ako ulazna formula ne sadrži niti jedan čvor $\wedge$ koji se nalazi na neodgovarajućoj poziciji, tada je α već u KNF. Pretpostavimo da α ima točno $(n+1)$ čvor koji se nalazi na neodgovarajućoj poziciji. Neka je $(\varphi \wedge \psi) \vee \pi)$ ili $(\pi \vee (\varphi \wedge \psi))$ podformula od α takva da na putu od ove podformule do korijena stabla $T(\alpha)$ nema više čvorova $\wedge$ koji su na neodgovarajućoj poziciji. Tada stablo $T(\alpha)$ transformiramo u stablo $T(\alpha*)$ kroz operaciju opisanu na slici 1.1, gdje A označava novu propozicionalnu varijablu. Pošto prvo primjenjujemo funkcije $REC-DEGENERATE$ i $REC-REARRANGE$ na podformule od α koje su disjunkcije podformula, uvijek imamo situaciju opisanu na slici sa samo jednim čvorom $\vee$ iznad čvora $\wedge$ i samo čvorovima $\wedge$ na putu do korijena stabla.



Slika 1.1: Transformacija stabla formule. Slika bazirana na slici 2.4, str.33 u [2].

Očito vrijedi da je $|F(\alpha*)| = |F(\alpha)| - 1 = n$.
Pošto je $\wedge$ čvor novi korijen podstabla, ne možemo imati novih čvorova $\wedge$ koji se nalaze na neodgovarajućim pozicijama. Broj čvorova $\wedge$ koji se nalaze na neodgovarajućim pozicijama unutar podstabla $T(\varphi)$, $T(\psi)$ i $T(\pi)$ kao i preostali dio stabla $T(\alpha)$ ostaju nepromijenjeni. Stoga algoritam završava i svaka formula se pretvara u formulu u KNF. Pošto je $|F(\alpha)|$ manji ili jednak od ukupnog broja pojavljivanja čvora $\wedge$ i svaki poziv funkcije REC-DEGENERATE ili REC-REARRANGE i svaka supstitucija se mogu izvršiti u linearnom vremenu, tada je ukupno vrijeme kvadratno u broju propozicionalnih varijabli ulazne formule. Duljina konačne formule ovisi linearno o duljini ulazne formule, pošto unutar svakog koraka dodamo tri nova pojavljivanja literala u formulu. Formule $(\varphi \wedge \psi) \vee \pi$ i $(\varphi \vee \neg A) \wedge (\psi \vee \neg A) \wedge (\pi \vee A)$ su ispunjivo ekvivalentne, pošto je A nova propozicionalna varijabla koju uvodimo kao zamjenu za podformulu $\varphi \wedge \psi$ na prije navedeni način. Međutim α i novodobivena formula nisu međusobno ekvivalentne. Zbog toga koristimo činjenicu da se interpretacija za koju je α istinit može proširiti do interpretacije za koju je $TREE - CNF(\alpha)$ istinita. Restrikcija interpretacije za koju je $TREE - CNF(\alpha)$ istinita se može restringirati na interpretaciju za koju je α istinita. □

Originalna verzija iskaza prethodnog teorema i pripadajući dokaz se mogu vidjeti u [2].

1.3 Osnovne tvrdnje logike sudova

Glavna literatura korištena u izradi ovog odjeljka je [5].

Pri zapisu formula u konjunktivnoj normalnoj formi često se umjesto pojma elementarna disjunkcija koristi pojam klauzule. Klauzula se sastoji od jednog ili više literala. Pojam

literala i propozicionalne varijable je usko povezan pa se ti pojmovi često koriste u kombinaciji pri iskazima ili dokazima nekih tvrdnji. Navodimo definicije oba pojma i primjere koji bi trebali demonstrirati veze i svojstva navedenih pojmova.

Definicija 1.3.1. ***Klauzula*** *je formula oblika* $l_1 \vee l_2 \vee \cdots \vee l_k$ *gdje je* $l_j,\ 1 \leq j \leq k$ *literal.*

Definicija 1.3.2. *Sa* $|l|$ *označavamo propozicionalnu varijablu pridruženu literalu l.* ***Predznak (polarnost)*** *literala l, u oznaci* $sgn(l)$*, je 1 (pozitivna) ako je l varijabla, a 0 (negativna) ako je l negacija varijable. Sa* $\bar{l}$ *označavamo literal dualan literalu l.*

Primjer 1.3.3. *Neka je P propozicionalna varijabla pridružena literalu l. Ako je* $|l| = P$*, tada je* $sgn(l) = 1$*. Zatim, vrijedi* $|\bar{l}| = P$ *i* $sgn(\bar{l}) = 0$*.*

Definicija 1.3.4. *Za literal l koji se javlja u skupu klauzula S kažemo da je* ***čist*** *za S ako se* $\bar{l}$ *ne javlja niti u jednoj klauzuli od S.*

Primjer 1.3.5. *Neka je* $S = \{l_1 \vee l_2 \vee l_3, l_1 \vee \bar{l_3}, l_2 \vee l_3, l_1 \vee \bar{l_2} \vee \bar{l_3}\}$*, tada je literal* l_1 *čist za skup klauzula S, a literal* l_3 *nije čist za S.*

U nastavku definiramo pojmove ispunjivosti i konzistentnosti skupa formula te uvodimo pojam Kleenijevog i Postovog uređaja za uspoređivanje vrijednosti interpretacije na proizvoljnoj propozicionalnoj varijabli. Kleenijev uređaj koristimo kod dokaza korektnosti Davis-Putnamovog algoritma.

Definicija 1.3.6. *Za skup formula S kažemo da je* ***ispunjiv*** *ako postoji interpretacija I tako da za sve formule* $F \in S$ *vrijedi* $I(F) = 1$ *(označavamo* $I(S) = 1$*).*

Definicija 1.3.7. *Za skup formula S kažemo da je* ***konzistentan*** *ako za svaku varijablu* P_i *iz skupa svih propozicionalnih varijabli* A_1*, najviše jedna od varijabli* P_i *i* $\neg P_i$ *pripada skupu S.*

Označimo sa u nepoznatu vrijednost interpretacije na određenoj propozicionalnoj varijabli P, koju ćemo možda kasnije saznati i promijeniti vrijednost u 1 ili 0. Postoje dva prirodna uređaja na skupu $\{0, 1, u\}$.

1. Prvi uređaj se naziva Kleenejev uređaj, $\leq_k$. Za taj uređaj vrijedi:

$$u \leq_k 0,\ u \leq_k 1$$

   dok su 1 i 0 neuspredivi.

2. Drugi uređaj se naziva Postov uređaj i označava $\leq_p$, a definiran je sa:

$$0 \leq_p u \leq_p 1$$

Neka je $B \subseteq \{P_0, P_1, \ldots, \}$ definiramo uređaj $\leq_k$ na $\prod_{p \in B}\{0, 1, u\}$ ovako:

$$v_1 \leq_k v_2 \text{ ako za svaki } p \in B \text{ vrijedi } v_1(p) \leq_k v_2(p)$$

Primijetimo da smo upotrijebili istu notaciju za uređaj ($\leq_k$) za uspoređivanje vektora i vrijednosti interpretacije na proizvoljnoj propozicionalnoj varijabli. Zapravo smo proširili definiciju Kleenijevog uređaja na vektor.

Primjer 1.3.8. *Neka je* $B = \{p, q, r, s\}$ *i* $v_1(p) = 0,\ v_1(q) = 1,\ v_1(r) = v_1(s) = u$. *Neka je interpretacija* v_2 *definirana na* B *sa* $v_2(p) = 0,\ v_2(q) = 1,\ v_2(r) = u$ *i* $v_2(s) = 1$.
Imamo: $v_1(p) = 0 \leq_k v_2(p) = 0,\ v_1(q) = 1 \leq_k v_2(q) = 1,\ v_1(r) = u \leq_k v_2(r) = u,\ v_1(s) = u \leq_k v_2(s) = 1$. *Tada vrijedi* $v_1 \leq_k v_2$

Primjer 1.3.9. *Neka je* $B = \{p, q, r, s\}$ *i* $v_1(p) = 0,\ v_1(q) = 1,\ v_1(r) = v_1(s) = u$. *Neka je* v_2 *definirana na* B *sa* $v_2(p) = 1,\ v_2(q) = 1,\ v_2(r) = u$ *i* $v_2(s) = 1$.
Imamo: $v_1(p) = 0,\ v_2(p) = 1,$ *stoga* $v_1(p) \not\leq_k v_2(p)$ *i* $v_2(p) \not\leq_k v_1(p),\ v_1(q) = 1 \leq_k v_2(q) = 1,\ v_1(r) = u \leq_k v_2(r) = u,\ v_1(s) = u \leq_k v_2(s) = 1 \Rightarrow v_1 \not\leq_k v_2$ *i* $v_2 \not\leq_k v_1$

Nakon što smo označili nepoznatu vrijednost interpretacije sa u, pokazujemo kako možemo proširiti proizvoljnu parcijalnu interpretaciju do adekvatne interpretacije za neku formulu, definiramo samodovoljne interpretacije koje imaju specijalna svojstva na skupu formula i navodimo neke činjenice vezane uz njihova svojstva.

Definicija 1.3.10. *Neka je* $F = F(P_1, \ldots, P_n)$ *proizvoljna formula i* v *neka nije adekvatna za* F, *tada definiramo adekvatnu interpretaciju* $v'(F)$ *na sljedeći način:*

$$v'(P_i) := \begin{cases} 0, \textit{ ako } v(P_i) = 0 \\ 1, \textit{ ako } v(P_i) = 1 \\ u, \textit{ ako } v(P_i) \textit{ nedefiniran} \end{cases}$$

Definicija 1.3.11. *Neka je* v *parcijalna interpretacija i* S *neki skup formula. Kažemo da* v ***dodiruje*** *formulu* F *ako neke propozicijonalne varijable iz domene od* v *pripadaju formuli* F. *Kažemo da je* v ***samodovoljna*** *za* S *ako za sve* $F \in S$ *takve da* v *dodiruje* F *vrijedi* $v'(F) = 1$.

Primjer 1.3.12. *Neka je* $S = \{l_1 \vee l_2 \vee \bar{l}_3, l_2 \vee l_4 \vee l_5 \vee \bar{l}_6, l_5 \vee \bar{l}_6 \vee l_7, l_2 \vee l_8 \vee l_9\}$ *zadani skup formula i* v *parcijalna interpretacija definirana sa:* $v(l_1) = 1,\ v(l_6) = 0$, *tada je* v *samodovoljna interpretacija za skup* S. *Očito za svaku formulu* $F \in S$ *takvu da* v *dodiruje* F *vrijedi* $v'(F) = 1$.

Propozicija 1.3.13. *Prazna parcijalna interpretacija je samodovoljna za bilo koji skup formula* S. *Bilo koja interpretacija* I *takva da* $I(S) = 1$ *je samodovoljna za* S.

Dokaz. Tvrdnja da je prazna parcijalna interpretacija samodovoljna za bilo koji skup formula slijedi direktno iz definicije samodovoljne interpretacije. Prazna interpretacija ne dodiruje niti jednu formulu iz skupa formula stoga je ona samodovoljna za taj skup.
Pretpostavimo da imamo skup formula S i neku interpretaciju I takvu da $I(S) = 1$. Po definiciji vrijedi da je tada $I(F) = 1$ za svaku formulu $F \in S$. Odnosno, za svaku formulu $F \in S$ takvu da I dodiruje F vrijedi $I(F) = 1$, a to je upravo definicija samodovoljne interpretacije. □

Propozicija 1.3.14.

1. *Svaka neprazna klauzula je ispunjiva.*
2. *Klauzula $C = l_1 \vee \cdots \vee l_k$ je tautologija ako i samo ako za neke $i, j, 1 \leq i \leq j \leq k$, l_j je dualan literal literalu l_i.*

Dokaz.

1. Izaberemo proizvoljan literal l u klauzuli C i uzmemo proizvoljnu interpretaciju I takvu da $I(l) = 1$.
2. Ako C sadrži par dualnih literala l_i i l_j tada je C tautologija, pošto za svaku interpretaciju I vrijedi $I(l_i) = 1$ ili $I(l_j) = 1$. Ako C ne sadrži par dualnih literala, tada za sljedeću interpretaciju I vrijedi $I(C) = 0$:

$$I(p) := \begin{cases} 1, & \text{ako je polarnost literla } p \text{ negativna} \\ 0, & \text{ako je polarnost literala } p \text{ pozitivna} \\ 0, & \text{ako se } p \text{ ne pojavljuje u } C \end{cases}$$

□

Uvodimo alternativan način definiranja parcijalne interpretacije. Sa $v = \{l_1, l_2, \ldots, l_n\}$ označavamo interpretaciju $v(l_1) = v(l_2) = \cdots = v(l_n) = 1$.

Propozicija 1.3.15. *Neka je S proizvoljni skup klauzula i l čisti literal za S. Tada je $v = \{l\}$ samodovoljna za S.*

Dokaz. Pošto je literal l čist za S, $\bar{l}$ se ne javlja niti u jednoj klauzuli iz S. Za sve klauzule $C \in S$ takve da v dodiruje C mora se l nalaziti u C, stoga $v'(C) = 1$. Dakle v je samodovoljna za S. □

Za zadani skup formula S, proizvoljna parcijalna interpretacija v određuje particiju $S = S'_v \cup S''_v$, pri čemu je S''_v skup formula iz S koje v dodiruje, a S'_v je skup koji sadrži preostale formule iz S.

Za skup klauzula S i literal l, uvodimo oznaku $S - l$ za skup svih klauzula iz S koje ne sadrže l.

Propozicija 1.3.16. *Neka je S skup klauzula i l čisti literal za S, S je ispunjiv skup ako i samo ako je $S - l$ ispunjiv skup.*

Dokaz. Pretpostavimo da je $S - l$ ispunjiv skup. Dakle postoji interpretacija v takva da je $v(S - l) = 1$. Definiramo interpretaciju v' na sljedeći način:

$$v'(l_i) := \begin{cases} v(l_i), & l_i \neq l \\ 1, & l_i = l \end{cases}$$

Pošto je l čisti literal za S i v_1 definirana sa $v_1(l) = 1$ samodovoljna za S, očito $v'(S) = 1$, stoga je i skup S ispunjiv.

Obrnuto, neka je S ispunjiv skup. Tada postoji interpretacija v_1 takva da $v_1(S) = 1$. Interpretacija v_1 je istinita na svim klauzulama iz S, stoga je istinita i na svim klauzulama iz S koje ne sadrže l. Definiramo interpretaciju v sa $v(l_i) = v_1(l_i),\ l_i \neq l$. Očito $v(S - l) = 1$, pa je i skup $S - l$ ispunjiv. □

Navodimo bez dokaza nekoliko propozicija koje opisuju svojstva samodovoljnih interpretacija. Dokazi se mogu pogledati u [5] str. 24-27.

Propozicija 1.3.17. *Ako je v samodovoljna interpretacija za skup formula S, tada je S ispunjiv ako i samo ako je S'_v ispunjiv.*

Neka su v i w dva konzistentna skupa literala. Definiramo $v \oplus w := v \cup \{l \in w : \bar{l} \notin v\}$.

Neka je v neka parcijalna interpretacija, Var_v domena od v i S skup klauzula. Tada sa O_v označavamo skup svih varijabli koje se javljaju u S, a ne javljaju se u Var_v.

Definicija 1.3.18. *Neka je S skup formula. Sa $\wedge S$ označavamo konjunkciju svih formula $F : F \in S$.*

Primjer 1.3.19. *Neka je $S = \{F_1, F_2, F_3\}$, tada $\wedge S = F_1 \wedge F_2 \wedge F_3$.*

Propozicija 1.3.20. *Neka je S skup nepraznih klauzula koje nisu tautologije. Netrivijalna parcijalna interpretacija v je samodovoljna za S ako i samo ako su za svaku klauzulu $C \in S$ i za svaku parcijalnu interpretaciju w takvu da $Dom(w) = O_v$ sljedeće tvrdnje ekvivalentne:*

(a) Postoji parcijalna interpretacija u takva da $Dom(u) = Var_v$ i $w \oplus u \models C$.

(b) $w \oplus v \models C$.

Propozicija 1.3.21. *Neka je S skup nepraznih klauzula koje nisu tautologije. Ako je netrivijalna parcijalna interpretacija v samodovoljna za S tada je za svaku parcijalnu interpretaciju w takvu da $Dom(w) = Var_v$ ekvivalentno:*

(a) Postoji parcijalna interpretacija u takva da $Dom(u) = Var_v$ i $w \oplus u \models \wedge F$

(b) $w \oplus v \models \wedge F$

Propozicija 1.3.22. *Neka je S skup nepraznih klauzula koje nisu tautologije. Ako je netrivijalna parcijalna interpretacija v samodovoljna za S tada je za svaku parcijalnu interpretaciju w takvu da $Dom(w) = Var_v$ ekvivalentno:*

(a) Postoji parcijalna interpretacija u takva da $Dom(u) = Var_v$ i $w \oplus u \models \wedge F$

(b) $w \models \wedge F'_v$

1.4 Mreže i fiksne točke

Glavna literatura korištena u izradi ovog odjeljka je [5].

U ovom odjeljku uvodimo pojam mreže i fiksne točke te navodimo osnovne rezultate vezane uz navedena dva pojma. Pojam mreže i fiksne točke ćemo koristiti kod algebarskog opisa svojstva operatora kojega koristi Davis-Putnam-Logemann-Loveland algoritam za pojednostavljivanje postupka pronalaska interpretacije za koju je neka zadana formula istinita.

Definicija 1.4.1. ***Mreža*** *je parcijalno uređen skup u kojem za svaki par elemenata postoji najmanja gornja ograda, te postoji najveća donja ograda. Najveću donju ogradu od x i y označavamo sa $x \wedge y$, a najmanju gornju ogradu sa $x \vee y$.*

Definicija 1.4.2. *Mreža $(L, \wedge, \vee)$ je* ***potpuna*** *ako za svaki podskup od L postoji najmanja gornja ograda i najveća donja ograda. U potpunoj mreži uvijek postoji jedinstveni najmanji element (koji označimo $\perp$) i jedinstveni najveći element ($\top$).*

Definicija 1.4.3. *Neka je $(L, \preceq)$ potpuna mreža. Kažemo da je funkcija $f : L \to L$ monotona ako za sve $x, y \in L$ vrijedi:*

$$x \preceq y \Rightarrow f(x) \preceq f(y)$$

Definicija 1.4.4. *Neka je $(L, \preceq)$ potpuna mreža. Fiksna točka funkcije $f : L \to L$ je proizvoljni $x \in L$ takav da $f(x) = x$.*

Definicija 1.4.5. *Neka je $(L, \preceq)$ potpuna mreža. Kažemo da je element $x \in L$* ***prefiksna točka*** *funkcije $f : L \to L$ ako $f(x) \preceq x$. Kažemo da je element $x \in L$* ***postfiksna točka*** *funkcije $f : L \to L$ ako $x \preceq f(x)$.*

Iskazujemo Knaster Tarskijev teorem o fiksnoj točki, koji će nam omogućiti definiranje zatvorenja formule po rezoluciji i jednu pomoćnu lemu koju ćemo koristiti u dokazu teorema.

Lema 1.4.6. *Neka je $f : L \to L$ funkcija i $(L, \preceq)$ potpuna mreža. Tada funkcija f ima prefiksnu i postfiksnu točku.*

Dokaz. Očito je $\top$ (najveći element iz L) prefiksna točka funkcije f, a $\bot$ (najmanji element skupa L) postfiksna točka funkcije f. □

Sa Pre_f i $Post_f$ ćemo označiti skupove prefiksnih točaka funkcije f i postfiksnih točaka funkcije f. Prethodna lema garantira da su oba skupa neprazna.

Navodimo iskaz i dokaz Knaster Tarskijevog teorema o fiksnoj točki.

Teorem 1.4.7 (Knaster Tarskijev teorem o fiksnoj točki)**.** *Neka je $f : L \to L$ monotona funkcija u potpunoj mreži $(F, \preceq)$. Tada:*

(a) f ima najmanju prefiksnu točku l, i l je najmanja fiksna točka od f.

(b) f ima najveću postfiksnu točku m, i m je najveća fiksna točka od f.

Dokaz.

(a) Iz leme 1.4.6 znamo da je $Pre_f \neq \emptyset$. Neka je $l = \bigwedge Pre_f$, to jest l je najveća donja ograda od Pre_f. Tvrdimo da je l prefiksna točka od f. Uzmimo proizvoljan $x \in Pre_f$. Po definiciji $l \preceq x$ i $f(x) \preceq x$. Zbog monotonosti funkcije f, pošto $l \preceq x$, tada $f(l) \preceq f(x)$. Zbog tranzitivnosti relacije $\preceq$ vrijedi $f(l) \preceq x$. Pošto je x bio proizvoljan element skupa Pre_f, tada $f(l) \preceq \bigwedge Pre_f$, odnosno $f(l) \preceq l$. Iz ovoga vidimo da je l najmanja prefiksna točka. Pokazujemo da je l fiksna točka. Primijetimo da pošto je l prefiksna točka vrijedi $f(f(l)) \preceq f(l)$, stoga je i $f(l)$ jedna prefiksna točka od f. Pošto je l najmanja prefiksna točka vrijedi $l \preceq f(l)$ pa je $l = f(l)$. Dokažimo još da je l najmanja fiksna točka. Neka je l' neka fiksna točka od f. Tada je l' prefiksna točka od f, a onda $l \preceq l'$ pošto je l najmanja prefiksna točka skupa Pre_f.

(b) Neka je $m = \bigvee Post_f$, to jest m je najmanja gornja ograda skupa $Post_f$. Tvrdimo da je m postfiksna točka od f. Promotrimo proizvoljan element $x \in Post_f$. Po definiciji $x \preceq m$ i $x \preceq f(x)$. Zbog monotonosti funkcije f vrijedi $f(x) \preceq f(m)$ pa po tranzitivnosti $x \preceq f(m)$. Pošto je x bio proizvoljan element skupa $Post_f$ tada

$\bigvee Post_f \preceq f(m)$, odnosno $m \preceq f(m)$. Stoga vrijedi da je m najveća postfiksna točka. Dokazujemo da je m fiksna točka. Primijetimo da vrijedi $f(m) \preceq f(f(m))$. Stoga je $f(m)$ također jedna postfiksna točka. Tada, pošto je m najveća postfiksna točka, vrijedi $f(m) \preceq m$. Stoga $m = f(m)$, odnosno m je fiksna točka funkcije f. Pokažimo da je m najveća fiksna točka funkcije f. Neka je m' proizvoljna fiksna točka funkcije f, tada je m' i postfiksna točka od f i $m' \preceq m$.

□

Dokaz prethodnog teorema nam ne govori kako eksplicitno izračunati fiksnu točku monotone funkcije međutim nama će u daljnjim rezultatima biti važna egzistencija fiksne točke za definiranje zatvorenja formule po rezoluciji u sljedećem odjeljku.

Poglavlje 2

Rezolucija

U ovom poglavlju definiramo rezoluciju, veoma važno pravilo u matematičkoj logici i u teoriji automatiziranog dokazivanja. Tvrdnje prezentirane u ovom poglavlju se mogu pronaći u [4, 5].

2.1 Rezolucija

U ovom odjeljku ćemo objasniti pojam rezolucije, te navesti neke primjene i tvrdnje vezane uz rezoluciju.

Neka je l proizvoljan literal, te C_1 i C_2 klauzule. Pravilo rezolucije je sljedeće pravilo izvoda:

$$\frac{l \vee C_1 \qquad \bar{l} \vee C_2}{C_1 \vee C_2}$$

Ako je klauzula $D_1 = l \vee C_1$ i $D_2 = \bar{l} \vee C_2$, tada je **rezolventa** $Res(D_1, D_2)$ definirana i dobiva se eliminacijom l i $\bar{l}$ iz D_1, D_2 i spajanjem preostalih literala u jednu klauzulu. Dakle vrijedi $Res(D_1, D_2) = C_1 \vee C_2$. Ako u D_1 i D_2 postoji više od jednog para dualnih literala tada je rezultantna klauzula tautologija, jer rezolucijom eliminiramo samo jedan par. Pretpostavljamo da se nakon izvršavanja rezolucije uklanjaju višestruka pojavljivanja nekog literala.

Primjer 2.1.1. *Pretpostavimo da je* $D_1 = p \vee q \vee s$ *i* $D_2 = \overline{p} \vee q \vee \bar{t}$*, tada je* $Res(D_1, D_2) = q \vee s \vee \bar{t}$

Primijetimo da rezolventa $Res(D_1, D_2)$ ne mora biti jedinstvena.

Primjer 2.1.2. *Pretpostavimo da je* $D_1 = p \vee q \vee \overline{q} \vee s$ *i* $D_2 = p \vee \overline{p} \vee q \vee \bar{t}$*, tada* $Res(D_1, D_2) = q \vee \overline{q} \vee s \vee \bar{t}$ *i* $Res(D_1, D_2) = p \vee \overline{p} \vee s \vee \bar{t}$

Neka je F proizvoljna, ali fiksna formula (skup klauzula), tada postoji najmanji skup klauzula G koji zadovoljava sljedeće uvjete:

1. $F \subseteq G$.
2. Ako su D_1 i D_2 dvije klauzule u G i definirana je operacija $Res(D_1, D_2)$, a rezultat je klauzula koja nije tautologija, tada $Res(D_1, D_2)$ pripada skupu G.

Tvrdnju dokazujemo nizom lema:

Lema 2.1.3. *Neka je F formula zapisana u obliku* $F = \{C_1, C_2, \ldots, C_n\}$ *gdje su* C_i *klauzule formule F. Skup svih podskupova skupa klauzula formule F je potpuna mreža.*

Dokaz. Neke je S proizvoljni podskup skupa klauzula F. Pretpostavimo da je $S = \{C_l, \ldots, C_k\}$ za proizvoljne indekse iz skupa $\{1, \ldots, n\}$. Tada je najveća donja ograda skupa S jednaka $\bigwedge S$, a najmanja gornja ograda $\bigvee S$. □

Neka je F fiksna formula i operator $res_F(.)$ definiran na potpunoj mreži podskupova skupa klauzula na sljedeći način:

$res_F(G) = F \cup \{Res(C_1, C_2) : C_1, C_2 \in F \cup G$ i $Res(C_1, C_2)$ je definirana i nije tautologija$\}$

Lema 2.1.4. *Operator* $res_F(.)$ *je monoton. Neka su G i H neka dva podskupa formule F. Ako* $G \subseteq H$ *tada* $res_F(G) \subseteq res_F(H)$*.*

Dokaz. Neka je $S \subseteq res_F(G)$ proizvoljan. Tada S može sadržavati neke klauzule iz F i neke klauzule iz skupa $\{Res(C_1, C_2) : C_1, C_2 \in F \cup G$ i $Res(C_1, C_2)$ je definiran i nije tautologija $\}$. Pošto $G \subseteq H$ tada H sadrži sve klauzule koje se nalaze u G, samim time sve klauzule $\{Res(C_1, C_2) : C_1, C_2 \in F \cup G$ i $Res(C_1, C_2)$ je definiran i nije tautologija$\}$ se nalaze i u $res_F(H)$, stoga $res_F(G) \subseteq res_F(H)$. □

Iz Knaster Tarskijevog teorema o fiksnoj točki (1.4.7) znamo da postoji najmanja fiksna točka od $res_F(.)$. Tu najmanju fiksnu točku označavamo sa $Res(F)$ i zovemo **zatvorenje** of F **po rezoluciji**.

Postavlja se pitanje je li Res potpun skup formula, to jest ako neka klauzula C logički slijedi iz F, mora li tada biti $C \in Res(F)$?
Odgovor je ne. Navedimo jedan primjer koji to pokazuje:

Primjer 2.1.5. *Neka je F formula koja se sastoji od jedne klauzule koja sadrži samo jedan literal a i neka je* $D = a \vee b$. *Očito* $F \models D$, *ali* $Res(F) = F$, *pa* $D \notin Res(F)$.

Neka je F formula u konjunktivnoj normalnoj formi. Klauzula C je **rezolucijska posljedica** od F ako $C \in Res(F)$. Klauzula C je **minimalna rezolucijska posljedica** od F ako je C rezolucijska posljedica od F ali niti jedan pravi podskup od C nije rezolucijska posljedica od F.

Sljedeća propozicija nam daje alat za pojednostavljivanje kompliciranijih formula i bržu provjeru je li generirana interpretacija istinita za zadanu formulu. Postupak se primjenjuje većinom kod stohastičkih algoritama koji koriste slučajne šetnje da bi pronašli rješenje. Propoziciju ćemo koristiti i kod dokaza teorema o potpunosti rezolucije.

Propozicija 2.1.6. *Ako su klauzule* C_1 *i* C_2 *istinite za interpretaciju v i operacija* $Res(C_1, C_2)$ *se može izvršiti, tada je* $Res(C_1, C_2)$ *istinita za v.*

Dokaz. Ako se $Res(C_1, C_2)$ može izvršiti tada postoje dvije klauzule D_1 i D_2 i literal l takav da $C_1 = D_1 \vee l$, $C_2 = D_2 \vee \bar{l}$. Ako $v(l) = 1$, tada $v(\bar{l}) = 0$, stoga $v(D_2) = 1$ stoga $v(D_1 \vee D_2) = 1$, odnosno $v(Res(C_1, C_2)) = 1$. Ako $v(l) = 0$ tada $v(D_1) = 1$ i opet slijedi $v(Res(C_1, C_2)) = 1$.

□

Pretpostavimo da imamo klauzulu C koja nije tautologija, $C = l_1 \vee \cdots \vee l_n$, $n \geq 1$. Tada postoji točno jedna parcijalna interpretacija v takva da:

1. $Dom(v) = Var_C$, (v je definirana samo na varijablama od C)
2. $v(C) = 0$

Tražena interpretacija je očito definirana sa:

$$v(x) := \begin{cases} 1, \text{ ako je } \neg x \text{ jedan od } l_1, \ldots, l_n \\ 0, \text{ ako je } x \text{ jedan od } l_1, \ldots, l_n \end{cases}$$

Pošto C po pretpostavci nije tautologija, tada je interpretacija v dobro definirana. Takav jedinstveni v označavamo v_C.

Teorem 2.1.7 (Quineov teorem)**.** *Klauzula C koja nije tautologija je posljedica skupa klauzula F ako i samo ako postoji klauzula D koja je rezolucijska posljedica od F i* $D \subseteq C$.

Quineov teorem nećemo dokazivati. Dokaz se može pogledati u [5] str. 111-113. Koristiti ćemo ga u poglavlju o primjenama rezolucije. Kao njegovu posljedicu imamo sljedeća dva korolara:

Korolar 2.1.8. *Klauzula C koja nije tautologija je logička posljedica formule u konjunktivnoj normalnoj formi F ako i samo ako za neku minimalnu rezolucijsku posljedicu D formule F, $D \subseteq C$.*

Definiramo **pravilo uključivanja** kao:

$$\frac{C}{C \vee D}$$

Pošto je glavna metoda pojednostavljivanja formule kod Davis-Putnamovog algoritma rezolucija, bitna nam je činjenica da je pravilo rezolucije potpuno. Stoga navodimo iskaz i dokaz sljedećeg korolara [5].

Korolar 2.1.9 (Teorem o potpunosti rezolucije)**.** *Neka je F formula u konjunktivnoj normalnoj formi. Formula F je ispunjiva ako i samo ako $\emptyset \notin Res(F)$.*

Dokaz. Ako je F ispunjiva tada $\emptyset \notin Res(F)$ slijedi iz propozicije 2.1.6. Da bi dokazali obrat pretpostavimo da $\emptyset \notin Res(F)$. Izaberimo novu propozicionalnu varijablu $p \notin Var(F)$ i pretpostavimo da $F \models p$. U tom slučaju se jedinična klauzula koja sadrži p mora dobiti pravilom uključivanja iz neke klauzule iz $Res(F)$. Po definiciji za svaku interpretaciju v takvu da $v(F) = 1$ vrijedi $v(p) = 1$. Pošto $p \notin Var(F)$ formula F nema nikakvog logičkog utjecaja na istinitost varijable p, stoga mora vrijediti da se p može dobiti pravilom uključivanja iz neke klauzule iz $Res(F)$. Pošto vrijedi $Var_{Res(F)} = Var_F$ jedina klauzula za koju je to moguće iz $Var_{Res(F)}$ je prazna klauzula. To je kontradikcija sa pretpostavkom. Stoga $F \not\models p$, pa je formula F ispunjiva. □

U nastavku definiramo i objašnjavamo računanje skupa $Res(S, x)$ koji čini bazu Davis-Putnamovog algoritma.

Definicija 2.1.10. *Neka je S skup klauzula i x neka propozicionalna varijabla. Definiramo novi skup kaluzula $Res(S, x)$ na sljedeći način:*

$$Res(S,x) := \begin{cases} S - l, \text{ ako je } l \text{ čist za } S \text{ i } |l| = x \\ \{C_1 \vee C_2 : x \vee C_1 \in S \text{ i } \neg x \vee C_2 \in S \text{ i } C_1 \vee C_2 \text{ nije tautologija}\} \ \bigcup \\ \{C \in S : x \text{ se ne pojavljuje u } C\} \end{cases}$$

Objasnimo računanje $Res(S, x)$:

1. U prvom slučaju kada je l čist za S jednostavno maknemo iz S sve klauzule koje sadrže l.

To možemo zato što za neku od interpretacija v_i za koju vrijedi $v_i(S) = 1$ možemo definirati:

$$v_i'(l_j) := \begin{cases} v_i(l_j), & l_j \neq l \\ 1, & l_j = l \end{cases}$$

Na taj način za sve klauzule C_k takve da $l \in C_k$ vrijedi $v_i'(C_k) = 1$ i $v_i'(S) = 1$, stoga možemo formulu za koju tražimo interpretaciju za koju je ta formula istinita smanjiti eliminacijom takvih klauzula.

2. U drugom slučaju na sve klauzule iz S koje sadrže x se provede rezolucija sa svim klauzulama iz S koje sadrže $\neg x$ što uzrokuje da se dvije klauzule poništavaju i spajaju u jednu po pravilu

$$\frac{x \vee C_i \quad \neg x \vee C_j}{C_i \vee C_j}$$

3. Nakon toga se eliminiraju tautologije. Pošto za sve C_i takve da je C_i tautologija vrijedi $v(C_i) = 1$ za proizvoljnu interpretaciju v. Takve klauzule možemo ispustiti pošto nam ne daju nikakvu informaciju prema kojoj bi mogli graditi interpretaciju za koju je S istinita.

4. Klauzule koje ne sadrže x ili $\neg x$ samo prelaze iz S u $Res(S, x)$.

Ako je $l = x$ i l je čist za S rezultat je $S - l$. Isti rezultat se dobije i ako je $l = \neg x$ čist za S. Primijetimo da ako postoji interpretacija v takva da $v(F) = 1$ tada i $v(Res(F, x)) = 1$. Ne možemo tvrditi da je veličina od $Res(S, x)$ manja od veličine skupa S, ali $Res(S, x)$ ima barem jednu varijablu manje jer je x eliminiran. U posebnim slučajevima, kada $l = x$ je čist za S, i neka varijabla y se javlja samo u klauzulama koje sadrže x, možemo eliminirati i više varijabli.
Veličina $Res(S, x)$ se može čak i povećati. Ako je $|F| = n$ tada $|Res(S, x)| \leq \frac{n^2}{4}$. Ako iteriramo operaciju $Res(\cdot, \cdot)$ selektirajući varijable koje se pojavljuju u formuli jednu po jednu, za formulu S sa m varijabli eliminirat ćemo sve varijable u najviše m koraka.

Koji je mogući krajnji ishod?

Moguća su dva ishoda. Jedan je da dobijemo prazan skup klauzula, a drugi da dobijemo neprazan skup klauzula koji sadrži samo jednu klauzulu: praznu klauzulu ($\perp$). U prvom slučaju naš skup je ispunjiv, u drugom je neispunjiv.

1. U prvom slučaju su nam klauzule bile kombinacije tautologija i klauzula koje sadrže neki čisti literal, pošto je to jedini slučaj u kojemu mićemo te klauzule iz formule. Za svaki čisti literal postavimo vrijednost trenutne interpretacije tako da vrijedi $v(l) = 1$. Stoga interpretacija dobivena u n-tom koraku predstavlja jednu intepretaciju za koju je početna formula istinita.

2. U drugom slučaju smo u jednom od koraka dobili klauzulu l i $\bar{l}$. Primjenom rezolucije na te dvije klauzule dobivamo klauzulu $\emptyset$ koju ne možemo eliminirati nikakvim postupkom iz skupa Res. Očito će za svaku interpretaciju v biti $v(S) = 0$, odnosno S nije ispunjiv.

Primijetimo da prazna klauzula, ako se dobije u nekoj od iteracija $Res(\cdot, \cdot)$, ostaje u skupu do eliminacije svih varijabli.

Navodimo dva primjera u kojima ćemo demonstrirati računanje skupa $Res(S, x)$.

Primjer 2.1.11. *Neka je* $S = \{x_1,\ \neg x_5,\ x_2 \vee x_4,\ x_3 \vee \neg x_5,\ x_5 \vee x_6,\ x_6 \vee x_7,\ x_1 \vee x_5 \vee \neg x_6,\ x_1 \vee \neg x_2 \vee x_6,\ x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4\}$ *zadani skup klauzula, računajmo* $Res(S, x_1)$. *Pošto je* l, *takav da* $|l| = x_1$, *čisti literal za* S *jednostavno iz skupa* S *izbacimo sve klauzule koje sadrže* x_1. *Tada je* $Res(S, x_1) = \{\neg x_5,\ x_2 \vee x_4,\ x_3 \vee \neg x_5,\ x_5 \vee x_6,\ x_6 \vee x_7\}$.

Promotrimo računanje skupa $Res(S, x)$ u slučaju kada l takav da $|l| = x$ nije čist za S.

Primjer 2.1.12. *Neka je* $S = \{x_1,\ \neg x_1 \vee x_2,\ x_1 \vee x_3,\ \neg x_1 \vee x_5,\ x_1 \vee x_6,\ x_1 \vee x_2 \vee \neg x_3,\ \neg x_5 \vee x_6 \vee \neg x_7\}$ *zadani skup klauzula, računajmo* $Res(S, x_1)$. *U ovom slučaju* l, *takav da* $|l| = x_1$, *nije čist za* S *stoga računamo dva skupa:*

1. *Za sve klauzule* C_i *takve da* $x_1 \vee C_i \in S$ *i sve klauzule* C_j *takve da* $\neg x_1 \vee C_j \in S$ *stavljamo klauzulu* $C_i \vee C_j$ *u skup* $Res(S, x_1)$.

   Dobivamo skup:

   $$T_1 = \{x_2,\ x_5,\ x_2 \vee x_3,\ x_2 \vee \neg x_3,\ x_2 \vee x_6,\ x_3 \vee x_4,\ x_3 \vee x_5,\ x_5 \vee x_6,\ x_2 \vee \neg x_3 \vee x_5,\ x_3 \vee x_4 \vee x_6\}.$$

   Primijetimo da smo pri računanju dobili i klauzulu $x_2 \vee \neg x_3 \vee x_3 \vee x_4$ *no ona je tautologija i ne ubacujemo ju u skup* T_1.

2. *Sve klauzule* $C \in S$ *takve da se* x *ne pojavljuje u* C *ubacimo u novi skup:*

   $$T_2 = \{\neg x_4 \vee x_5, \neg x_5 \vee x_6 \vee \neg x_7\}.$$

   $Res(S, x_1) = T_1 \cup T_2 = \{x_2,\ x_5,\ x_2 \vee x_3,\ x_2 \vee \neg x_3,\ x_2 \vee x_6,\ x_3 \vee x_4,\ x_3 \vee x_5,\ \neg x_4 \vee x_5,\ x_5 \vee x_6,\ x_2 \vee \neg x_3 \vee x_5,\ x_3 \vee x_4 \vee x_6,\ \neg x_5 \vee x_6 \vee \neg x_7\}$

   Skup $Res(S, x_1)$ *nam je u ovom primjeru dosta veći od skupa* S, *no u njemu se javlja jedna varijabla manje (eliminirali smo varijablu* x_1*).*

2.2 Baza zatvorenja formule po rezoluciji i primjene

Pretpostavimo da želimo upotrijebiti sustav dokazivanja rezolucijom i pravilom uključivanja kao alat za odgovaranje na upite. Promatramo proizvoljnu formulu F kao bazu podataka koja je spremljena u sustavu. Upiti su klauzule zadane formule. Odgovor na upit C je *da* ako $F \models C$, *ne* inače. Kada dobijemo upit C, možemo C pretvoriti u skup literala $\neg C$ ($\neg C$ je zapravo skup klauzula koje sadrže samo po jedan literal). Na negacije klauzula primjenjujemo rezoluciju. Ako pri izračunu dobijemo praznu klauzulu, odgovor na naše pitanje je *da* (negacija našeg upita nije ispunjiva, stoga je naš upit ispunjiv). U suprotnom, ako dobijemo prazan skup klauzula odgovor je *ne*. Ovakav sustav omogućuje odgovaranje na upite, međutim za svaki upit se ponovo primjenjuje rezolucija što nije učinkovito. Dodatna mana je veličina zatvorenja formule po rezoluciji [5].

Gornji sustav možemo poboljšati tako da izračunamo $Res(F)$ jednom i spremimo rezultat u memoriju. Kada dobijemo upit C provjerimo postoji li u skupu $Res(F)$ klauzula koja je podskup klauzule C. Ako takva klauzula postoji odgovor na naše pitanje je *da*, inače iz Quineovog teorema (2.1.7) slijedi da je naš odgovor *ne*.
Problem računanja ponekad velikog skupa $Res(F)$ rješavamo konstruiranjem znatno manjeg podskupa $Min \subset Res(F)$ sa istim svojstvom potpunosti: $C \in C_n(F)$ ako i samo ako je neki $D \in Min$ sadržan u C.
Algoritam za obradu upita ide tako da prvo izračunamo bazu Min. Za svaki upit C provjerimo je li neki element iz Min podskup od C. Ako takav element postoji odgovor je *da*, inače je odgovor na upit *ne*.

Definicija 2.2.1. *Za proizvoljnu interpretaciju v i skup fomula logike sudova S definiramo skup sematničkih posljedica:*

$$C_n(S) := \{\varphi : \textit{ za svaku interpretaciju v takvu da } v \models S \textit{ vrijedi } v \models \varphi\}$$

Operator C_n svakom skupu formula pridružuje skup njihovih posljedica.

Definiramo bazu skupa $Res(F)$. Baza od F je skup klauzula G sa sljedećim svojstvima:

1. $G \subseteq Res(F)$.
2. $C_n(F) = C_n(G)$.
3. G formira antilanac, ako $C_1,\ C_2 \in G$, takvi da $C_1 \subseteq C_2$, tada $C_1 = C_2$.
4. Svaki element skupa $Res(F)$ je dobiven primjenom pravila uključivanja iz nekog elementa skupa G.

Definicija 2.2.2. *Za klauzulu C kažemo da je minimalna klauzula skupa klauzula S u skladu s relacijom inkluzije, ako ne postoji klauzula $C_k \in S$ takva da $C_k \subset C$, odnosno klauzula C se ne može dobiti pravilom uključivanja iz niti jedne klauzule skupa S.*

Propozicija 2.2.3. *Neka je F skup klauzula. Tada postoji jedinstvena baza G za F.*

Dokaz. Za zadani F konstruiramo zatvorenje po rezoluciji $Res(F)$. Neka je Min skup svih minimalnih klauzula u skladu s relacijom inkluzije iz $Res(F)$. Tvrdimo da je Min jedinstveni skup klauzula koji zadovoljava uvjete (1)-(4). Provjeravamo redom uvjete:
(1) je očito po konstrukciji
(2) Pošto $Min \subseteq Res(F) \subseteq C_n(F)$, tada $C_n(Min) \subseteq C_n(F)$. Ako $C \in C_n(F)$ tada iz Quinovog teorema (2.1.7) neka klauzula $D \in Res(F)$ je podskup od C. Stoga je jedna minimalna inkluzijska klauzula $D \in F$ podskup od C. Pošto $D \in Min$ i $D \subset C$ slijedi da je $C \in C_n(Min)$.
(3) Minimalni elementi u proizvoljnoj familji skupova formiraju antilanac.
(4) Pretpostavimo da je $D \in Res(F)$. Pošto je D konačan, postoji minimalan inkluzijski element od $Res(F)$, označimo ga sa C, takav da $C \subseteq D$. Po definiciji tada $C \in Min$.

Dokazujemo da je Min jedinstvena baza za F. Pretpostavimo da je $G \neq Min$ neka baza za F.
1. slučaj $G \backslash Min \neq \emptyset$. Izaberemo $C \in G \backslash Min$. Tada postoji $D \in Min$ takav da $D \subset C$. Pošto je G baza i $D \in C_n(F)$ postoji $E \in G$, $E \subseteq D$. To povlači da $E \subset C$ što je kontradikcija sa činjenicom da je G antilanac.
2. slučaj $Min \backslash G \neq \emptyset$. Izaberemo $C \in Min \backslash G$. Tada postoji $D \in G$ takav da se C može dobiti primjenom pravila uključivanja iz D. Vrijedi $D \neq C$ zato što $D \notin Min$. Pošto $D \in C_n(F)$ tada postoji $E \in Min$ takav da $E \subseteq D$. Međutim tada $E \subset C$ i $C, E \in Min$ što je ponovo kontradikcija. □

Da bi konstruirali algoritam za konstrukciju baze Min, prvo definiramo algoritam za eliminaciju iz formule svih klauzula koje se mogu dobiti pravilom uključivanja iz neke druge klauzule te formule.

Neka je F zadana formula. U polinomnom vremenu možemo u ovisnosti o broju klauzula formule eliminirati sve klauzule dobivene pravilom uključivanja iz neke druge klauzule formule F na sljedeći način:

1. Sortiramo klauzule formule F po njihovoj veličini (broju literala u klauzulama) te unutar određene veličine uredimo klauzule leksikografski (x_i dolazi u poretku prije $\neg x_i$).

2. Za svaku klauzulu provjerimo koje klauzule formule dobivamo iz zadane pravilom uključivanja i eliminiramo iz formule (sve takve klauzule će se pojaviti kasnije u poretku od zadane klauzule).

Složenost procedure je $O(s^2)$ gdje je s broj klauzula formule F. Izlaz procedure je antilanac.

Sada dajemo nedeterministički algoritam za konstrukciju baze Min:

1. Ulaz je formula F u konjunktivnoj normalnoj formi. Zbog gore navedene procedure pretpostavljamo da se u F ne nalaze klauzule koje se mogu dobiti pravilom uključivanja iz neke druge klauzule formule F.

2. Nedeterministički izaberemo par klauzula iz F na koje se može primijeniti rezolucija i na kojima se rezolucija još nije provela. Označimo taj par sa C_1, C_2.

3. Ako u F postoji klauzula iz koje se $Res(C_1, C_2)$ može dobiti primjenom pravila uključivanja ili pak je $Res(C_1, C_2)$ tautologija izaberemo sljedeći par klauzula.

4. Ako u F ne postoji klauzula iz koje se $D := Res(C_1, C_2)$ može dobiti pravilom uključivanja tada:

   (a) Izračunamo $R := \{E \in F : D \subseteq E\}$. Eliminiramo iz F sve klauzule koje se dobiju iz D primjenom pravila uključivanja, $F := F \backslash R$.

   (b) Definiramo $F := F \cup \{D\}$.

5. Ponavljamo korake 1-4 dok se F ne promijeni.

Razmatramo korektnost algoritma za računanje Min.

Dokazujemo:

1. F je antilanac i nakon svake iteracije algoritma F će ostati antilanac.

2. Označimo sa F_0 početnu formulu koju je algoritam dobio na ulazu. Nakon svake iteracije algoritma, za svaku privremenu formulu F vrijedi $C_n(F) = C_n(F_0)$.

1. Za činjenicu da je F antilanac se brinu koraci (3) i (4) algoritma. Ako nakon računanja $Res(C_1, C_2)$ postoji neka klauzula u F iz koje se može dobiti $Res(C_1, C_2)$ tada novodobivenu klauzulu ignoriramo. Ako takva klauzula ne postoji izbacimo iz formule sve klauzule koje se mogu dobiti pravilom uključivanja iz klauzule $Res(C_1, C_2)$. Primijetimo da je ulazna formula antilanac pošto smo pretpostavili da su iz nje početno eliminirane sve klauzule koje se mogu dobiti uključivanjem iz neke druge klauzule formule. Stoga ako za neke klauzule vrijedi $C_1 \subseteq C_2$ tada mora biti $C_1 = C_2$.
2. Da bi pokazali drugu tvrdnju pretpostavimo da je F_1 vrijednost formule F prije izvođenja iteracije algoritma, a F_2 vrijednost formule poslije izvođenja iteracije algoritma.

Ako $F_2 = F_1$ tada tvrdnja trivijalno vrijedi. Pretpostavimo da je $F_2 = (F_1 \backslash R) \cup \{C\}$ i $C_n(F_1) = C_n(F)$. Pokazat ćemo da $C_n(F_2) = C_n(F)$. Pošto je $F_1 \subseteq C_n(F)$, tada $F_1 \backslash R \subseteq C_n(F)$. Zatim iz $C \in C_n(F)$ slijedi $F_2 \subseteq C_n(F)$. Tada $C_n(F_2) \subseteq C_n(F)$, $R \subseteq C_n(\{C\})$ zato što sve klauzule iz R možemo dobiti pravilom uključivanja iz klauzule C. Imamo: $C_n(F_1) = C_n((F_1 \backslash R) \cup R) \subseteq C_n((F_1 \backslash R) \cup C_n(\{C\})) \subseteq C_n((F_1 \backslash R) \cup \{C\}) = C_n(F_2)$. Zadnja inkluzija slijedi iz činjenice da $C_n(C_n(F)) = C_n(F)$.

Trebamo još provjeriti da algoritam završava u konačno mnogo koraka. Trebamo pokazati da se uklonjene klauzule neće ponovo vratiti u formulu u nekoj od narednih iteracija. Pretpostavimo da je F_1 vrijednost formule prije iteracije algoritma i D neka klauzula koja se može dobiti pravilom uključivanja iz neke klauzule iz F_1. Sa F_2 označimo vrijednost formule nakon iteracije algoritma. Ako $F_2 = F_1$ tvrdnja očito vrijedi. Ako $F_2 = (F_1 \backslash R) \cup \{C\}$ tada je moguć jedan od dva slučajeva:

1. Postoji klauzula iz $F_1 \backslash R$ iz koje se može dobiti D pravilom uključivanja. U ovom slučaju D možemo dobiti pravilom uključivanja iz F_2.

2. Klauzula D se može dobiti pravilom uključivanja iz neke klauzule iz R. Po konstrukciji, sve klauzule skupa R možemo dobiti pravilom uključivanja iz klauzule C. Stoga za svaki $E \in R$ takav da $E \subseteq D$ vrijedi $C \subseteq E \subseteq D$.

Stoga skup svih klauzula koje se eliminiraju pravilom uključivanja raste i algoritam staje u konačno mnogo koraka.

Postavlja se pitanje, jesmo li zbilja izračunali bazu zatvorenja formule po rezoluciji?
Odgovor je da.
Nakon izvršavanja algoritma zadovoljeno je $C_n(F) = C_n(F_0)$ gdje je F_0 ulazna formula algoritma i za svaku klauzulu $C \in Res(F)$ postoji $D \in F$ takav da se C može dobiti iz D primjenom pravila uključivanja.

2.3 Redukcija literala i Davis-Putnamova lema

U ovom odjeljku ćemo opisati redukciju literala, te iskazati i dokazati Davis-Putnamovu lemu [5]. Redukcija literala se koristi kao dodatni mehanizam pojednostavljivanja formule koju obrađujemo kod raznih algoritama za rješavanje problema SAT. Davis-Putnamov algoritam koristi redukciju u drugoj fazi svoga izvođenja, točnije pri traženju interpretacije za koju je ulazna formula istinita. Promotrit ćemo podjelu formule na redukcije i utjecaj istinitosti redukcija na istinitost početne formule.

Pretpostavimo da je S skup klauzula i v parcijalna interpretacija. Tada se redukcija od S u oznaci $Red(S, v)$ računa:

1. Ako za neki l u C, $v(l) = 1$, tada se klauzula C eliminira.
2. U klauzulama C koje su preživjele test (1) eliminiramo sve literale l takve da $v(l) = 0$.

Ovim postupkom reduciramo broj varijabli i klauzula u formuli S u odnosu na interpretaciju v.

Primjer 2.3.1. *Neka je S sljedeći skup klauzula:*

$$\{p \vee q \vee \bar{r}, \bar{p} \vee q \vee s, p \vee t\}$$

i neka je v parcijalna interpretacija definirana na $\{q, t\}$ sa vrijednostima: $v(q) = 0,\ v(t) = 1$. Tada $Red(S, v) = \{p \vee \bar{r}, \bar{p} \vee s\}$.
Klauzula $p \vee q \vee \bar{r}$ je reducirana primjenom pravila 2. Pošto $v(q) = 0$, tada q eliminiramo iz svih klauzula od S. Iz istoga razloga je reducirana i klauzula $\bar{p} \vee q \vee s$. Klauzula $p \vee t$ je eliminirana primjenom pravila 1 pošto $v(t) = 1$.

Računanje skupa $Red(S, v)$ nam olakšava pronalaženje interpretacije koja je definirana na svim literalima od S i za koju je S istinita. Nakon računanja redukcije, lako se vidi da je jedna od interpretacija za koju je S istinita dana sa: $v'(q) = 0,\ v'(t) = 1,\ v'(p) = 1,\ v(s) = 1,\ v(r) = 1$.

Postupak redukcije se često koristi kod stohastičkih solvera koji prvo generiraju interpretaciju definiranu na svim literalima formule pridjeljujući na slučajan način vrijednost interpretacije na svakom literalu. Primjenom redukcije se formula pojednostavljuje i lakše se uočava koje vrijednosti interpretacije bi trebalo promijeniti. Na taj način se slučajno generirana interpretacija pokušava dograditi do interpretacije za koju je početna formula istinita.

Definiramo redukciju formule F u odnosu na literal l na sljedeći način:

$$G := \{C \backslash \{\bar{l}\} : C \in F \text{ i } l \notin C\}$$

Primijetimo da literal l određuje dvije redukcije: u odnosu na l i $\bar{l}$. Za zadanu formulu F, te dvije redukcije označavamo sa F_l i $F_{\bar{l}}$.

Lema 2.3.2 (Davis-Putnam lema)**.** *Neka je F skup klauzula koje nisu tautologije i neka je l neki literal formule F. Skup klauzula F nije ispunjiv ako i smo ako nisu ispunjivi niti F_l niti $F_{\bar{l}}$.*

Dokaz. Pretpostavimo da je F_l ili $F_{\bar{l}}$ ispunjiva formula. Radi određenosti pretpostavimo da je F_l ispunjiva. Slučaj kada je formula $F_{\bar{l}}$ ispunjiva dokazuje se sasvim analogno. Pošto F ne sadrži klauzule koje su tautologije, F_l ne sadrži l i $\bar{l}$. Pošto je F_l ispunjiva postoji neka interpretacija v takva da $v(F_l) = 1$. Definiramo interpretaciju ω na sljedeći način:

$$\omega(m) = \begin{cases} v(m), & m \notin \{l, \bar{l}\} \\ 1, & m = l \end{cases}$$

Tada za sve klauzule $C \in F$ takve da $l \notin C$ vrijedi $v \models C$, stoga vrijedi i $\omega \models C$. Ako $l \in C$ tada $\omega \models C$ po konstrukciji. Iz prethodnih razmatranja vidimo da vrijedi $\omega(F) = 1$, pa je formula F ispunjiva.

Obrnuto, pretpostavimo da F_l i $F_{\bar{l}}$ nisu ispunjive, te da postoji interpretacija v takva da $v(F) = 1$. Pretpostavimo da $v(l) = 1$ i promatrajmo F_l. Ako $v(l) = 0$ promatramo $F_{\bar{l}}$ i primjenimo analogne argumente na taj skup.
Za $C \in F_l$ moguća su dva slučaja:

1. $C \in F$ i $l, \bar{l} \notin C$. U ovom slučaju redukcija po literalu l nije utjecala na klauzulu C, odnosno ona je identična kao i u formuli F. Pošto po pretpostavci $v(F) = 1$, za svaku klauzulu $C_k \in F$ vrijedi $v(C_k) = 1$, stoga $v \models C$.

2. U drugom slučaju je $C = D \backslash \{\bar{l}\}$ i $\bar{l} \in D$, $D \in F$. Sada imamo $v \models D$ i $v(\bar{l}) = 0$ zato što je $v(l) = 1$, pa mora vrijednost interpretacije v za jedan od preostalih literala u D biti 1. U tom slučaju $v \models C$.

Iz ova dva slučaja zaključujemo da $v \models F_l$ što je kontradikcija.

□

Iz prethodne leme očito slijedi tvrdnja sljedećeg korolara:

Korolar 2.3.3. *Neka je F skup klauzula koje nisu tautologije i neka je l neki literal formule F. Formula F je ispunjiva ako i samo ako je barem jedna formula od F_l, $F_{\bar{l}}$ ispunjiva.*

Sljedeća propozicija nije vezana uz Davis-Putnamov algoritam, međutim vidjet ćemo da se malo varirani oblik njenoga rezultata koristi kod UnitWalk algoritma kojega ćemo kasnije opisati.

Propozicija 2.3.4. *Neka je F skup klauzula. Ako je l čist literal za F tada je F ispunjiva ako i samo ako je F_l ispunjiva. Ako je v interpretacija takva da $v(F_l) = 1$ tada za interpretaciju ω definiranu:*

$$\omega(m) = \begin{cases} v(m), & m \notin \{l, \bar{l}\} \\ 1, & m = l \end{cases}$$

vrijedi $\omega(F) = 1$.

Dokaz. Pošto je F ispunjiva formula, postoji interpretacija v takva da $v(F) = 1$. Definiramo interpretaciju v' na sljedeći način:

$$v'(m) = v(m), \quad m \notin \{l, \bar{l}\}$$

Očito $v'(F_l) = 1$.
Obrnuto, neka je F_l ispunjiv skup, tada postoji interpretacija v takva da $v(F_l) = 1$. Definiramo interpretaciju ω na sljedeći način:

$$\omega(m) = \begin{cases} v(m), & m \notin \{l, \bar{l}\} \\ 1, & m = l \end{cases}$$

Tvrdimo da vrijedi $\omega(F) = 1$. Skup F_l sadrži sve klauzule formule F osim klauzula koje sadrže literal l. Klauzule koje sadrže literal $\bar{l}$ formule F su identične u F_l osim što ne sadrže $\bar{l}$. Pošto se ω podudara sa v na svim klauzulama osim klauzulama koje sadrže literal l, one su istinite za ω. Zbog definicije interpretacije ω na klauzulama koje sadrže literal l i te klauzule su istinite, stoga vrijedi $\omega(F) = 1$.

□

Kod UnitWalk algoritma mi također postavljamo vrijednost generirane interpretacije svih čistih literala na 1. Pošto tražimo interpretaciju v takvu da $v(F_l) = 1$ vrijednosti interpretacije na literlima koji nisu čisti određujemo na slučajan način. Ova propozicija nam govori da idemo u pravom smjeru. Kasnije ćemo vidjeti da UnitWalk ne kvari rješenje. Ako se generirana interpretacija poklapa s interpretacijom za koju je ulazna formula istinita na k varijabli, algoritam te vrijednosti neće mijenjati do na jednu varijablu u periodu.

Poglavlje 3

Davis-Putnamov algoritam

U ovom odjeljku ćemo iznijeti Davis-Putnamov algoritam i dokazati njegovu korektnost. Glavna literatura korištena za stvaranje ovog poglavlja je [5].

3.1 Dokaz korektnosti DP algoritma

Propozicija koju navodimo je baza za dokazivanje korektnosti DP algoritma [5].

Propozicija 3.1.1. *Neka je S neki skup klauzula i x neka propozicionalna varijabla. Tada vrijedi:*

(a) S je ispunjiv ako i samo ako je $Res(S, x)$ ispunjiv skup.

(b) Ako je v neka parcijalna interpretacija takva da $v(S) = 1$, tada je $v(Res(S, x)) = 1$. Ako je v parcijalna interpretacija definirana na $Var_{Res(S,x)}$ takva da $v(Res(S, x)) = 1$ tada se v može proširiti do interpretacije w tako da $w(S) = 1$.

Dokaz. Očito (b) povlači (a) stoga dokazujemo samo (b). Ako se varijabla x ne pojavljuje u S tada je očito $S = Res(S, x)$ i (b) vrijedi.
Pretpostavimo da se x javlja u nekim klauzulama iz S. Pokazujemo da $v(S) = 1$ povlači $v(Res(S, x)) = 1$. Iz korolara 2.1.8 slijedi $v(S) = 1$ povlači $v(Res(S)) = 1$.
Pošto $Res(S, x) \subseteq Res(S)$ vrijedi $v \models Res(S, x)$.
Sada pretpostavimo da $v \models Res(S, x)$ i da je v definirana na varijablama iz $Res(S, x)$.
Interpretaciju v proširujemo do ω tako da vrijedi $\omega \models S$. Promatramo dva slučaja:

1. $Var_{Res(S,x)} = \emptyset$
2. $Var_{Res(S,x)} \neq \emptyset$

Promatramo slučaj 1. Ovdje postoje tri podslučaja:

1.1 Literal x je čist za S. Tada $S - x = \emptyset$. Definiramo interpretaciju ω:

$$\omega(y) = \begin{cases} 1, & y = x \\ 0, & y \in Var_S \setminus \{x\} \end{cases}$$

Pošto je $Res(S, x)$ ispunjiv i $Var_{Res(S,x)} = \emptyset$, mora biti $Res(S, x) = \emptyset$ i svaka kluzula u S mora sadržati x pa vrijedi $\omega(S) = 1$.

1.2 Literal $\neg x$ je čist za S. Tada na isti način kao i u prethodnom slučaju, za intepretaciju ω koja je za svaku varijablu $y \in Var_S$ definirana sa $\omega(y) = 0$ vrijedi $\omega(S) = 1$.

1.3 Literali x i $\neg x$ nisu čisti za S. Ne može se dogoditi da neka klauzula $C \in S$ ne sadrži niti x niti $\neg x$, zato što bi se takva klauzula automatski našla u skupu $Res(S, x)$. Stoga za svaku klauzulu $C \in S$ postoji D takav da $C = D \vee x$ ili E takav da $C = E \vee \neg x$. Barem jedna od klauzula $D,\ E$ mora biti neprazna ili rezolucijom dobijemo $\emptyset \in Res(S, x)$ što znači da $Res(S, x)$ nije ispunjiv.

Pretpostavimo da je $C_1 = D \vee x,\ C_2 = E \vee \neg x$. Ako je D prazna klauzula, tada je rezolventa E. Pošto C_2 nije tautologija, E nije tautologija stoga je E neprazna klauzula koja nije tautologija u $Res(S, x)$ što je kontradikcija s pretpostavkom $Var_{Res(S,x)} = \emptyset$. Ekvivalentan je slučaj kada je E prazna klauzula. Iz toga zaključujemo da nisu niti klauzula D niti klauzula C prazne klauzule. Zbog $Var_{Res(S,x)} = \emptyset$ vrijedi $D \vee E \notin Res(S, x)$ što znači da je $D \vee E$ tautologija. Ako $D \vee x \in S$ i $E \vee \neg x \in S$ tada mora postojati neka propozicionalna varijabla x_i takva da $x_i \in D$ i $\neg x_i \in E$.

Izaberimo klauzulu $C \in S$ oblika $D \vee x$ i definiramo interpretaciju ω:

$$\omega(y) = \begin{cases} 1, & y = x \\ v_D(y), & y \in D \\ 0, & y \neq x \text{ i } y \notin D \end{cases}$$

Tvrdimo: $\omega \models S$. Ako klauzula $C' \in S$ sadrži varijablu x tada očito $\omega(C') = 1$. Pretpostavimo da C' sadrži varijablu $\neg x$. Sada mora postojati neka varijabla $x_i \neq x,\ x_i \neq \neg x$ takva da $x_i \in C$ i $\neg x_i \in C'$. Po definiciji interpretacije v_D vrijedi $v_D(x_i) = 0$ pa je $\omega(x_i) = 0$, odnosno $\omega(\neg x_i) = 1$. Tako smo dobili željeni rezultat $\omega(C') = 1$.

Sada promatramo drugi slučaj, $Var_{Res(S,x)} \neq \emptyset$. Konstrukciju provodimo slijedeći istu strategiju kao i u prvom slučaju. Pretpostavljamo da imamo interpretaciju v definiranu samo na varijablama od $Res(S, x)$ i želimo proširiti tu interpretaciju do interpretacije ω takve da $\omega \models S$.

Promatrajmo skup $G = Red(S, v)$. Primijetimo da za sve klauzule eliminirane iz S u koraku (1) redukcije vrijedi $v(C) = 1$, stoga to vrijedi i za svako proširenje interpretacije v. Primijetimo da je svaka klauzula iz G podklauzula neke klauzule iz S. Za svaki $C \in S$ takav da $v(C) = 0$, redukcija klauzule C po v pripada skupu $Red(S, v)$. Dovoljno je pokazati da za konstruiranu interpretaciju ω vrijedi $\omega(G) = 1$ što povlači $\omega(S) = 1$.

Promatramo četiri slučaja:

2.1 Skup G je prazan. To znači da za sve klauzule C iz S vrijedi $v(C) = 1$. Pošto moramo definirati interpretaciju koja je definirana na svim varijablama iz S, definiramo:

$$\omega(y) = \begin{cases} v(y), & y \in Dom(v) \\ 0, & y \in Var_F \backslash Var_G \end{cases}$$

Očito $v \preceq_k \omega$, stoga vrijedi $\omega(S) = 1$.

2.2 Literal x je čist u G. U tom slučaju definiramo dvije interpretacije ω' i ω. Interpretacija ω' je definirana na x i domeni od v. Interpretacija ω je definirana na svim varijablama od S. Ovdje je definicija od ω':

$$\omega'(y) = \begin{cases} v(y), & y \in Dom(v) \\ 1, & y = x \end{cases}$$

Tvrdimo da za sve klauzule $C \in S$ vrijedi $\omega'(C) = 1$. Neka je $C \in S$. Ako se x i $\neg x$ ne nalaze u C tada $C \in Res(S, x)$. Stoga $v(C) = 1$ i $\omega'(C) = 1$. Ako se x nalazi u C tada $\omega'(C) = 1$ po konstrukciji. Preostala mogućnost je da se $\neg x$ nalazi u C. Pošto je x čist u $Red(S, v)$ i x nije u domeni od v mora vrijediti da je C eliminiran u koraku (1) redukcije po v. Stoga za barem jednu propozicionalnu varijablu $y \in C$ vrijedi $v(y) = 1$. Stoga i $\omega'(C) = 1$. Interpretaciju ω' trivijalno proširimo do interpretacije ω definirane na svim varijablama iz S:

$$\omega(y) = \begin{cases} \omega'(y), & y \in Dom(\omega') \\ 0, & \text{inače} \end{cases}$$

2.3 Literal $\neg x$ je čist u G. Definiramo interpretacije ω' i ω. Interpretacija ω' je definirana na $\neg x$ i domeni od v, a ω je definirana na svim varijablama od S. Ovdje je prvo definicija interpretacije ω':

$$\omega'(y) = \begin{cases} v(y), & y \in Dom(v) \\ 1, & y = \neg x \end{cases}$$

Analogno slučaju 2.2 pokazujemo da za sve klauzule $C \in S$ vrijedi $\omega'(C) = 1$. Neka je $C \in S$. Ako se x i $\neg x$ ne nalaze u C tada $C \in Res(S, \neg x)$. Stoga $v(C) = 1$ i $\omega'(C) = 1$. Ako se $\neg x$ nalazi u C tada $\omega'(C) = 1$ po konstrukciji. Preostala mogućnost je da se x nalazi u C. Pošto je $\neg x$ čist u $Red(S, v)$ i x nije u domeni od v mora vrijediti da je C eliminiran u koraku (1) redukcije po v. Stoga za barem jednu propozicionalnu varijablu $y \in C$ vrijedi $v(y) = 1$. Stoga i $\omega'(C) = 1$. Analogno interpretaciju ω' trivijalno proširimo do interpretacije ω definirane na svim varijablama iz S na sljedeći način:

$$\omega(y) = \begin{cases} \omega'(y), & y \in Dom(\omega') \\ 0, & \text{inače} \end{cases}$$

2.4 Literali x i $\neg x$ nisu čisti za G. Da bi konstruirali ω u ovom slučaju, izaberemo neku klauzulu koja sadrži x, recimo $x \vee C_1 \in G$. Definiramo ω na sljedeći način:

$$\omega(y) = \begin{cases} v(y), & y \in Dom(v) \\ 1, & y = x \\ v_{C_1}(y), & y \in Dom(v_{C_1}) \\ 0, & \text{inače} \end{cases}$$

Prvo pokazujemo da je ω dobro definirana. Pošto se x ne nalazi u domeni od v, prva dva slučaja u definiciji nisu u konfliktu. Treći slučaj ne stvara kontradikciju pošto je klauzula $x \vee C_1 \in Red(S, v)$. Stoga varijable iz C_1 nisu u domeni od v. Pošto $x \vee C_1$ nije tautologija varijabla x se ne nalazi u C_1.

Sada trebamo pokazati da vrijedi $\omega \models S$. Pretpostavimo da $D \in S$ i $v(D) = 1$. Tada i $\omega(D) = 1$. Ako se x nalazi u D tada je $\omega(D) = 1$ pošto $\omega(x) = 1$. U slučaju da se x i $\neg x$ ne nalaze u D, D pripada $Res(S, x)$. Stoga vrijedi $v(D) = 1$ i $\omega(D) = 1$. Preostaje nam slučaj kada se $\neg x$ nalazi u D i za sve varijable iz D koje su u $Dom(v)$ vrijedi $v(x_i) = 0$. Tada imamo:

$$D = \neg x \vee D_1 \vee D_2,$$

te u D_1 se ne javlja x niti jedna druga varijabla na kojoj je v definirana i vrijedi $v'(D_2) = 0$. Promotrimo klauzulu $x \vee C_1$ koja pripada skupu $G = Red(S, v)$. To znači da postoji klauzula C oblika:

$$C = x \vee C_1 \vee C_2,$$

te je redukcijom nastala klauzula $x \vee C_1$, gdje za svaku varijablu $x_i \in C_2$ vrijedi $v(x_i) = 0$. Na klauzule $x \vee C_1 \vee C_2$ i $\neg x \vee D_1 \vee D_2$ možemo primijeniti rezoluciju. Rezultat je klauzula $E = C_1 \vee C_2 \vee D_1 \vee D_2$. Primijetimo da klauzula E ne

pripada skupu $Res(S, x)$. Kada bi E pripadao skupu $Res(S, x)$ tada bi vrijedilo $v(E) = 1$. Međutim $v(C_2) = v(D_2) = 0$ i v nije definirana niti na jednoj varijabli iz klauzula C_1 i D_1. Mora vrijediti da je klauzula E tautologija. Pošto vrijedi $v(C_2) = v(D_2) = 0$ par komplementarnih varijabli zbog kojih je E tautologija se mora nalaziti u $C_1 \vee D_1$. To znači da za proizvoljnu klauzulu $D \in S$ koja sadrži $\neg x$ mora postojati varijabla $x_i \neq x,\ x_i \neq \neg x$ takva da se x_i javlja u C_1 i $\neg x_i$ u D. Pošto ω proširuje v_{C_1} tada $\omega(x_i) = 0$, pa $\omega(\neg x_i) = 1$ što povlači $\omega(D) = 1$.

□

Ključni element gornje propozicije je što daje konstruktivni način proširenja svake interpretacije v za koju je istinit $Res(S, x)$ do interpretacije za koju je istinita formula S.

Kod dokaza propozicije počinjemo s interpretacijom v koja je definirana samo na varijablama iz $Res(S, x)$ i za koju je $Res(S, x)$ istinit. Naš cilj je proširiti v do ω, koja je definirana samo na varijablama od S tako da je S istinita za ω. Da bi proširili v do ω dovoljno je znati skup svih klauzula iz S koje nisu elementi skupa $Res(S, x)$. Proceduru opisanu u dokazu propozicije 3.1.1 ćemo zvati `expand(`G`,`v`,`x`)`. Procedura ima tri ulaza: parcijalnu interpretaciju v, varijablu x koja je u domeni od v i skup klauzula G takav da se x ili $\neg x$ pojavljuju u svakoj klauzuli iz G. Intuitivno G je skup klauzula koje su eliminirane iz S pri računanju $Res(S, x)$, gdje je x varijabla po kojoj smo vršili eliminaciju, v je interpretacija za koju je $Res(S, x)$ istinit. Procedura `expand(`G`,`v`,`x`)` vraća parcijalnu interpretaciju ω koja proširuje v. Kada je v parcijalna interpretacija definirana samo na varijablama iz $Res(S, x)$ i $G = S \setminus Res(S, x)$ tada je formula S istinita za interpretaciju $\omega =$ `expand(`G`,`v`,`x`)`.
Kada pri računanju $Res(S, x)$ dobijemo praznu klauzulu ($\emptyset$), tada se ta klauzula prenosi u svim daljnjim kalkulacijama, pošto prolazi uvjet (1) pri računanju skupa Res. Na kraju kada eliminiramo sve varijable rezultat će biti neprazan i neispunjiv skup, stoga svaki puta kada izračunamo prazan skup moramo naznačiti da nismo uspjeli pronaći rješenje.
Uvodimo i funkciju `SelectVar` koja kao argument prima skup klauzula S sa barem jednom varijablom i vraća jednu od varijabli iz S.

Dajemo pseudokod verzije Davis-Putnam algoritma koji računa je li ulazna formula ispunjiva [5]. Nazivamo ju `DPtest`.

```
Algoritam DPtest.
Ulaz: formula F u KNF.
Izlaz: Odluka je li F ispunjiva.

 if(F == ∅ )
  {return('Ulazna formula je ispunjiva')}
 elseif(F sadrži praznu klauzulu ∅)
```

```
  {return('Ulazna klauzula nije ispunjiva')}
else
 { x=SelectVar(F);
   F = Res(F, x);
   return(DPtest(F))}
```

Dokazujemo korektnost navedenog algoritma:

1. Ako je F ispunjiv skup klauzula imamo dva slučaja:
   (a) F je prazan skup i algoritam vraća da je F ispunjiva.
   (b) F ne sadrži praznu klauzulu ($\emptyset$). Kada je F neprazan skup i ne sadrži praznu klauzulu tada mora sadržavati barem jednu varijablu. Korektnost sada slijedi indukcijom pošto se algoritam rekurzivno primjenjuje na skup klauzula koje imaju jednu varijablu manje. Ako algoritam vrati 'ulazna formula je ispunjiva' za ulaz F tada mora biti F prazna ili je algoritam odredio da je $Res(F, x)$ ispunjiva. U drugom slučaju je F ispunjiva zbog propozicije 3.1.1.
2. Ako F nije ispunjiv skup klauzula, tada će po korolaru 2.1.9 vrijediti $\emptyset \in Res(F, x)$ i algoritam će vratiti 'ulazna formula nije ispunjiva'.

Kod algoritma `DPtest` trebali smo održavati samo trenutnu vrijednost varijable F i ispitati je li prazna i sadrži li praznu klauzulu. Broj logičkih veznika varijable F u `DPtest` algoritmu je eksponencijalan u ovisnosti o veličini ulazne formule, ograničen je s 3^n gdje je n broj varijabli u F.

Funkcija `DPtest` samo ispituje je li funkcija ispunjiva, no nas zanima i interpretacija za koju je istinita. U nastavku definiramo dodatne podatke koje trebamo pamtiti i opisujemo algoritam kojim dolazimo do interpretacije za koju je ulazne formula istinita, pod pretpostavkom da je ulazna formula ispunjiva.

U verziji DP algoritma koja zbilja pronalazi interpretaciju za koju je formula F istinita trebamo održavati složenije strukture podataka:

1. Da bi izračunali proširenje parcijalne interpretacije v u ω moramo znati skup klauzula koje su bile eliminirane kada smo računali skup $Res(F, x)$. Ti skupovi su oblika $F \backslash Res(F, x)$. Pošto se i F i x mijenjaju kako algoritam napreduje, moramo ih spremiti u memoriju.
2. Nakon što izračunamo sve takve skupove (pod pretpostavkom da nismo došli do kontradikcije, odnosno izračunali praznu klauzulu) krenemo računati u suprotnom smjeru izračunavajući uzastopne parcijalne interpretacije. Krajnja interpretacija je interpretacija za koju je ulazna formula istinita.

3. Moramo znati i koja varijabla je korištena u rezoluciji pa pamtimo i slijed tih varijabli.

Na kraju prve faze već znamo je li formula ispunjiva ili nije. Ako je formula ispunjiva u drugom dijelu računamo interpretaciju za koju je formula istinita računajući unatrag.

Definiramo algoritam `DPsearch` koji se sastoji od dvije faze:

1. U fazi I (koja je slična algoritmu `DPsearch`) izračunamo i spremimo podatke koji će se koristiti u fazi II.
2. U fazi II izračunavamo interpretaciju za koju je početna formula istinita.

U fazi I računamo dva polja:

- Polje `layerSeq`, koje sadrži skupove G izračunate u svakom koraku.
- Polje `varSeq` koje sadrži varijable koje su korištene za rezoluciju (računanje skupova oblika $Res(F, x)$).

Od funkcija potrebne su nam:

- Pretpostavljamo da imamo definiranu funkciju koja implementira operaciju $Res(F, x)$.
- Pretpostavljamo da imamo definiranu heurističku funkciju `selectVar` koja na skupu klauzula G, takvom da $Var_G \neq \emptyset$ vraća jednu od varijabli koje se javljaju u G.
- Trebamo i proceduru `Forward`. Ova procedura prima četiri parametra: skup klauzula G, polje `varSeq`, polje `layerSeq` i integer i. Procedura iterativno proširuje polja računajući varijable i odgovarajuće slojeve (klauzule koje smo odbacili primjenom rezolucije, u svakom koraku, na ulaznu formulu). Procedura pazi na poredak i stavlja izračunate vrijednosti točno na pravo mjesto u polju.

```
Algoritam Forward(G, varSeq, layerSeq,i).

 if(G sadrži praznu kluzulu ∅)
  { return({p,  p̄ })}
 elseif(G je prazan)
   { return(varSeq,layerSeq) }
 else          {
    x=SelectVar(G);
    varSeq[i]=x;
    layerSeq[i]=G\ Res(G, x) ;
```

```
    G = Res(G, x);
    i++;
  return(Forward(G,varSeq,layerSeq,i))}
```

Konstruirani algoritam `Forward` iteriramo na ulaznoj formuli F, integeru i inicijaliziranom na 0 i dva inicijalno prazna polja.

```
Algoritam DPSearchPhaseOne

Ulaz: Formula F u KNF koja sadrži najmanje jednu varijablu.
Izlaz: Nekonzistentan skup literala ili dva polja:
       polje slojeva i polje varijabli.

G=F;
varSeq=∅;
layerSeq=∅;
i=0;
Forward(G,varSeq,layerSeq,i)
```

Nakon završetka rada algoritma `DPSearchPhaseOne` znamo da ulazna formula F ili nije ispunjiva ili na izlazu dobijemo dva polja, polje slojeva i polje varijabli. Ta dva polja su ulaz za fazu II.
U toj fazi koristimo proceduru `expand`(G, v, x) da bi konstruirali interpretaciju za koju je ulazna formula F istinita. Algoritam `DPsearchPhaseTwo` koristi polja izračunata u fazi I, ali se izvršava u suprotnom smjeru, računajući od zadnjeg elementa u poljima prema prvome. Zbog toga ćemo smatrati da imamo proceduru `reverse` koja prima polje i vraća polje sa istim elementima raspoređenim u obrnutom redosljedu.

```
Algoritam DPsearchPhaseTwo

Ulaz: Dva polja, varSeq i layerSeq
      koja vrati algoritam DPSearchPhaseOne.
Izlaz: Interpretacija za koju je istinita ulazna formula
       F iz faze I.

layerSeq=reverse(layerSeq);
varSeq=reverse(varSeq);
v=∅;
j=length(layerSeq);
```

```
for(i = 0; i < j; i++)
 {G =layerSeq[i];
 x=varSeq[i];
 v=expand(G,v,x);}
 return(v);
```

Uzastopnim ponavljanjem argumenta propozicije 3.1.1ulazna formula F je istinita za interpretaciju koju vrati algoritam `DPsearchPhaseTwo`. Pseudokodovi navedenih algoritama i dokaz korektnosti `DPtest` algoritma može se pronaći u [5].

Primjer 3.1.2. *Neka je F sljedeći skup klauzula:* $\{C_1, \ldots, C_7\}$*, pri čemu je:*

$C_1 : \ p \vee \bar{t},$
$C_2 : \ p \vee s,$
$C_3 : \ q \vee r \vee s,$
$C_4 : \ \bar{q} \vee r \vee s,$
$C_5 : \ \bar{r} \vee \bar{s},$
$C_6 : \ u \vee \omega,$
$C_7 : \ s \vee \bar{u} \vee \bar{\omega} \vee x$

1. *Na početku pridružimo privremenoj varijabli G vrijednost ulazne formule F.*

   - *Pretpostavimo da je p=*`selectVar(G)`*, tada stavljamo* `varSeq[0]`*=p. Razumno je odabrati p pošto je taj literal čist za F i sljedeći skup klauzula je manji od početnog.* `layerSeq[0]` *je* $\{C_1, C_2\}$*, a G postaje* $\{C_3, C_4, C_5, C_6, C_7\}$*.*
   - *Pretpostavimo da je sada u=*`selectVar(G)`*. Primjetimo da se* `selectVar` *primjenjuje na formulu G, a ne na ulaznu formulu F.* `varSeq[1]`*=u i* `layerSeq[1]` *je jednak* $\{C_6, C_7\}$*. Promotrimo što se u ovom koraku događa sa G. Nakon računanja skupa* $Res(G, u)$ *kreira se nova klauzula* $s \vee \omega \vee \bar{\omega} \vee x$*, međutim ova klauzula se ne nalazi u G zato što je tautologija. G je sada* $\{C_3, C_4, C_5\}$*.*
   - *Neka je q=*`selectVar(G)`*, sada je* `varSeq[2]`*=q,* `layerSeq[2]`*=* $\{C_3, C_4\}$*. Nakon provođenja rezolucije na klauzule* C_3, C_4 *kreiramo novu klauzulu* $C_8 : r \vee s$ *(duplikati literala unutar klauzule se eliminiraju). Sada je* $G = \{C_5, C_8\}$*.*
   - *Neka je sada r =*`selectVar(G)`*,* `varSeq[3]`*= r,* `layerSeq[3]`*=* $\{C_5, C_8\}$*. Rezolventa klauzula* C_5 *i* C_8 *je* $s \vee \bar{s}$ *što je tautologija pa naša formula G postaje prazna,* $G = \emptyset$*.*

Pošto nismo izračunali praznu klauzulu niti u jednom od koraka algoritma faza I vraća polja `varSeq`$=< p,\ u,\ q,\ r >$ *i* `layerSeq`$=< \{C_1,\ C_2\},\ \{C_6,\ C_7\},\ \{C_3,\ C_4\},\ \{C_5,\ C_8\} >$.

2. *Krećemo u fazu II algoritma.*

   - *Prvi korak je preokretanje redoslijeda pojavljivanja elemenata u poljima, stoga dobivamo:*

     $< r,\ q,\ u,\ p > i < \{C_5,\ C_8\},\ \{C_3,\ C_4\},\ \{C_6,\ C_7\},\ \{C_1,\ C_2\} >$.

   - *Prvo inicijaliziramo parcijalnu interpretaciju h na* $\emptyset$.

   - *Sada iterativno koristimo funkciju* `expand`. *Interpretacija h dobivena u prvom koraku je:* $h(r) = 1,\ h(s) = 0$. *Redukcijom sljedećeg sloja dobijemo praznu redukciju. Stavimo* $h(q) = 1$. *U nastavku reduciramo sloj klauzula* `layerSeq[2]` *po h, dobivamo dvije klauzule* $u \vee \omega$ *i* $\bar{u} \vee \bar{\omega} \vee x$ *(zato što je* $h(s) = 0$*). Stavimo* $h(u) = 1,\ h(\omega) = 0$ *i* $h(x) = 0$. *Sada reduciramo sloj* `layerSeq[3]` *po interpretaciji h. Reducirani skup sadrži klauzulu* $p \vee \bar{t}$ *i jedinične klauzule p. Pošto je p čist, stavimo* $h(p) = 1,\ h(t) = 0$.

Druga faza DP algoritma kao izlaz daje interpretaciju:

$$\begin{pmatrix} p & q & r & s & t & u & \omega & x \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Faza II DP algoritma dopušta dosta slobode. U proizvoljnoj iteraciji konstrukcije možemo uključiti neku drugu parcijalnu interpretaciju (ako je odgovarajući dio formule F istinit za tu interpretaciju). Dok izvršavamo prvu fazu DP algoritma možemo ga prekinuti i riješiti ostatak formule F s nekim drugim SAT algoritmom. Na taj način dobijemo parcijalnu interpretaciju v na koju primijenimo drugu fazu algoritma (pošto druga faza računa u suprotnom smjeru od prve faze imamo izračunate sve slojeve i varijable potrebne za izračunati konačnu interpretaciju). Zbog toga razloga DP algoritam može služiti kao pred i post procesor za DPLL algoritam koji ćemo promatrati u slijedećem poglavlju. Da bi DP algoritam radio na taj način, možemo pozvati DPLL algoritam ako se veličina skupa $Res(F, x)$ poveća iznad nekog definiranog praga.

Poglavlje 4

Davis-Putnam-Logemann-Loveland algoritam

U ovom poglavlju predstavljamo DPLL algoritam. Dokazujemo njegovu korektnost i istražujemo moguća poboljšanja. Za više detalja pogledajte [5].

4.1 Boolovsko ograničeno širenje

U ovom odjeljku istražit ćemo mehanizme pojednostavljivanja koje koristi DPLL algoritam za ispitivanje ispunjivosti logičke formule, koji ćemo opisati u sljedećem odjeljku. Ova tehnika računa skup literala koji direktno slijede iz ulaznog skupa klauzula.

Primjer 4.1.1. *Primjer se sastoji od dvije varijacije na istu temu:*
U prvom primjeru koristimo činjenicu da postoje literali koji se mogu direktno izračunati iz ulaznog skupa klauzula. Drugi primjer je isti kao i prvi samo uz skup klauzula imamo zadanu i ulaznu parcijalnu interpretaciju.

(a) Neka je F skup klauzula: $\{C_1, \ldots, C_7\}$*, pri čemu je:*

$C_1 : \ \bar{p} \vee \bar{q} \vee \bar{s},$
$C_2 : \ r \vee s \vee \bar{z},$
$C_3 : \ \bar{z} \vee u,$
$C_4 : \ z \vee h \vee l,$
$C_5 : \ p,$
$C_6 : \ q,$
$C_7 : \ \bar{r}$

Pošto F sadrži klauzule sa samo jednim literalom (jedinične klauzule) C_5 i C_6 jedini način da te klauzule budu istinite istovremeno sa klauzulom C_1 je da pridružimo vrijednost 0 literalu s. Nakon toga jedini način da C_2 bude istinita je da pridružimo 0 literalu z, također zbog klauzule C_7 moramo pridružiti vrijednost 0 literalu r. U ovom trenutku klauzula C_3 je istinita bez obzira na vrijednost koju pridružimo literalu u. Zbog praktičnih razloga klauzulu C_4 (čiju vrijednost još ne znamo) reduciramo na $h \vee l$. Da bi pronašli interpretaciju za koju je F istinita prisiljeni smo pridjeliti vrijednosti odgovarajućim literalima kao u sljedećoj parcijalnoj interpretaciji: $v(p) = v(q) = 1,\ v(r) = v(s) = v(z) = 0$.

(b) Promotrimo skup klauzula F' koji se sastoji od klauzula $C_1,\ C_2,\ C_3$ i C_4 i parcijalnu interpretaciju v' definiranu sa $v'(p) = v'(q) = 1,\ v'(z) = 0$. Uz F' i v' zadane,svaka parcijalna interpretacija ω koja proširuje interpretaciju v' i za koju je F istinita mora proširiti i v (interpretaciju dobivenu u (a) dijelu). Argument za to je identičan kao i u (a). Skup klauzula F' se kao u (a) reducira u skup klauzula koji sadrži samo jednu klauzulu $h \vee l$.

Razmišljajući algebarski, u gornjem primjeru smo izračunali fiksnu točku određene operacije. Definiramo operaciju bcp_F. Ova operacija na ulazu prima skup literala (ako je konzistentan radi se o parcijalnoj interpretaciji, ali ne treba biti) i kao izlaz daje drugi skup literala (koji može biti nekonzistentan).

$$bcp_F(S) = \{l :\ \text{Postoji klauzula } C :\ l_1 \vee \cdots \vee l_{j-1} \vee l,\ C \in F,\ \bar{l}_1, \ldots, \bar{l}_{j-1} \in S.\}$$

Propozicija 4.1.2. *Neka je zadana formula F u KNF. Operator $bcp_F(.)$ je monoton operator u potpunoj mreži $\mathcal{P}(Lit)$. Sa Lit označavamo skup svih literala formule F.*

Dokaz. Pretpostavimo da su S i G dva skupa literala takva da je $S \subset G$ i F proizvoljna formula. Neka je $l \in bsp_F(S)$ proizvoljan literal. Po definiciji tada postoji klauzula $C = l_1 \vee \cdots \vee l_{j-1} \vee l$ takva da $C \in F$ i $\bar{l}_1, \ldots, \bar{l}_{j-1} \in S$. Pošto $S \subset G$ tada $\bar{l}_1, \ldots, \bar{l}_{j-1} \in G$ pa je i $l \in bcp_F(G)$. Od tuda slijedi $bcp_F(S) \subset bcp_F(G)$. Pretpostavimo sada da je $bcp_F(S) = bcp_F(G)$ i $S \subset G$. Tada vrijedi $\forall l,\ l \in bcp_F(S) \Leftrightarrow l \in bcp_F(G)$. Stoga za svaki literal l postoji $C \in F$ takva da $C = l_1 \vee \cdots \vee l_{j-1} \vee l$ i $\bar{l}_1, \ldots, \bar{l}_{j-1} \in S$. Pošto $l \in bcp_F(G)$ postoji klauzula $C' \in F$ takva da $\bar{l}_1, \ldots, \bar{l}_{j-1} \in G$. Očito vrijedi $S = G$ što je u suprotnosti sa pretpostavkom. □

Po Knaster Tarskijevom teoremu o fiksnoj točci (1.4.7) za svaki skup klauzula F operacija bcp_F sadrži najmanju fiksnu točku. Najmanja fiksna točka se označava $BCP(F)$.

$BCP(F)$ možemo definirati i u terminima logike ograničavajući formu rezolucijskih dokaza koje prihvaćamo. Računamo skup literala koji se mogu izračunati iz F koristeći rezoluciju gdje je jedna od klauzula na koju primjenjujemo rezoluciju uvijek jedinična klauzula. Takav ograničeni oblik rezolucije nazivamo **jedinična rezolucija**.

Definiramo operator $BCP(F)$ koristeći jediničnu rezoluciju:

$$BCP'(F) = \{l : \text{ za } \{l\} \text{ postoji stablo dokaza jediničnom rezolucijom iz } F\}$$

U nastavku dokazujemo da za sve skupove klauzula F operatori $BCP(F)$ i $BCP'(F)$ moraju vraćati isti skup literala.

Propozicija 4.1.3. *Za sve skupove klauzula F vrijedi $BCP(F) = BCP'(F)$.*

Dokaz. Da bi pokazali da vrijedi $BCP(F) \subseteq BCP'(F)$ trebamo dokazati da $BCP'(F)$ sadrži najmanju fiksnu točku iz bcp_F. Induktivno ćemo pokazati da je svaka iteracija $bcp_F^n(\emptyset)$ sadržana u $BCP'(F)$. Baza je očita. Pretpostavimo da je $l \in bcp_F^{n+1}(\emptyset)$. Tada postoji klauzula $C : \ l_1 \vee \cdots \vee l_{j-1} \vee l$ i $\bar{l}_1, \ldots, \bar{l}_{j-1} \in bcp^n(\emptyset)$. Tada po induktivnoj pretpostavci svaki od $\bar{l}_1, \ldots, \bar{l}_{j-1}$ se može dobiti jediničnom rezolucijom. Kombiniramo rezolucije na sljedeći način (klauzula $C_i = l_i \vee \cdots \vee l_j \vee l$) :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{C}\quad\dfrac{}{\bar{l}_1}}{C_1}\quad\dfrac{}{\bar{l}_2}}{\ddots}\quad\dfrac{}{\bar{l}_{j-1}}}{C_{j-1}}}{l}$$

Na taj način smo dokazali inkluziju $BCP(F) \subseteq BCP'(F)$.
Pretpostavimo obrnuto da $l \in BCP'(F)$. Indukcijom po broju primjena jediničnih rezolucija pokazujemo da $l \in BCP(F)$. Baza je opet očita. Pretpostavimo da imamo rezolucijsko stablo kao u dokazu prve inkluzije i da $l_1, \ldots, l_{j-1}$ svi pripadaju skupu $bcp^n(\emptyset)$, vidimo da l pripada skupu $bcp^{n+1}(\emptyset)$. To dokazuje obratnu inkluziju. □

Moramo imati na umu da $BCP(F)$ može biti nekonzistentan (sadržavati praznu klauzulu).

Primjer 4.1.4. *Neka je $F = \{p, \neg p \vee q, \neg p \vee \neg q\}$. Jediničnom rezolucijom možemo dobiti q i $\neg q$, stoga i praznu klauzulu.*

Definicija 4.1.5. *Definiramo redukciju formule F po skupu literala S_l na sljedeći način:*

$$G := \{C \backslash \{\bar{l}_i\} : C \in F, \text{ za svaki literal } l_i \in S_l \, , l_i \notin C\}$$

Promotrimo efekt računanja $BCP(F)$. Ako smo utvrdili da interpretacija na literalu l mora imati vrijednost 1, tada svaka klauzula C u kojoj se pojavljuje literal l je istinita bez obzira na dodjeljivanje vrijednosti interpretacije na ostalim literalima iz C u budućnosti. Stoga možemo eliminirati C iz budućih razmatranja. Ukoliko dođemo do kontradikcije pri takvom izboru vrijednosti literala, mijenjamo vrijednost literala, vraćamo u formulu sve klauzule izbačene pomoću l te eliminiramo sve klauzule koje su postale istinite nakon promjene vrijednosti literala. Ako se literal $\bar{l}$ pojavi u nekoj klauzuli C, tada $\bar{l}$ izbacimo iz takvih klauzula pošto on ne doprinosi mjenjanju istinitosti klauzule C. Možemo provesti redukciju formule F po $BCP(F)$. Ako je $BCP(F)$ nekonzistentan, tada je i F nekonzistentna (obrnuta implikacija ne vrijedi) i moramo vratiti poruku da ulazna formula nije ispunjiva. Redukcija formule F po skupu literala $BCP(F)$ je ispunjiva ako i samo ako je F ispunjiva formula. Dokazat ćemo to svojstvo u idućoj propoziciji.

Propozicija 4.1.6. *Neka je F formula u KNF. Tada je F ispunjiva ako i samo ako je $BCP(F)$ konzistentan i redukcija formule F po $BCP(F)$ je ispunjiva.*

Dokaz. Ako je v interpretacija za koju je F istinita, tada je vrijednost v na svim literalima iz $BCP(F)$ jednaka 1. To slijedi odmah iz ispravnosti rezolucije, točnije jedinične rezolucije kao specijalnog slučaja. Neka je R redukcija formule F po $BCP(F)$. Ako $D \in R$ tada za neke literale $l_1, \ldots, l_k \in BCP(F)$ klauzula $C : \; D \vee \bar{l}_1 \vee \cdots \vee \bar{l}_k$ pripada formuli F. Međutim $v(\bar{l}_1) = \cdots = v(\bar{l}_k) = 0$ stoga mora vrijediti $v(D) = v(C)$. Pošto $v(C) = 1$ mora biti i $v(D) = 1$. Pošto je D bio proizvoljna klauzula iz redukcije, dokazali smo da vrijedi $v(BCP(F)) = 1$.
Pretpostavimo da je $BCP(F)$ konzistentan i da je redukcija R formule F po $BCP(F)$ ispunjiva. Neka je v parcijalna interpretacija za koju je R istinit. Primijetimo da se po konstrukciji ne nalazi u R niti jedna varijabla (sa ili bez negacije) koja se nalazi u $BCP(F)$. Pri izvođenju redukcije iz formule F u R neke varijable su možda u potpunosti eliminirane, ne javljaju se niti u R niti u $BCP(F)$. Moramo pridružiti neku vrijednost interpretaciji i za te varijable. Pošto je $BCP(F)$ konzistentna, postoji interpretacija v' takva da je $BCP(F)$ istinita za nju. Definiramo novu interpretaciju ω na sljedeći način:

$$\omega(p) = \begin{cases} v(p), & \text{ako se } p \text{ nalazi u } R \\ v'(p), & \text{ako se } p \text{ nalazi u } BCP(F) \\ 0, & \text{inače} \end{cases}$$

Tvrdimo da je F istinita za ω. Neka je C neka klauzula iz F. Ako je C eliminirana u prvom djelu računanja redukcije R (neki literal l iz $BCP(F)$ pripada klauzuli C) tada $\omega(C) = 1$

zato što $\omega(l) = v'(l) = 1$. U suprotnom $C = D \vee l_1 \vee \cdots \vee l_k$ i $\bar{l}_1, \ldots, \bar{l}_k \in BCP(F),\ D \in R$. Međutim $v(D) = 1$, stoga za neki literal l koji se nalazi u D, $v(l) = 1$. Tada $\omega(l) = 1$, pošto se l nalazi u C, $\omega(C) = 1$, stoga vrijedi $\omega(F) = 1$. □

U ovom odjeljku smo opisali tehniku jedinične rezolucije kao metodu poboljšavanja osnovnog Davis-Putnamovog algoritma. Napomenuli smo da je rezolucija jako dobra tehnika pojednostavljivanja i eliminacije određene varijable iz formule. Primjenom rezolucije dobijemo istinitosno ekvivalentnu formulu koja sadrži barem jednu varijablu manje, što uvelike smanjuje prostor rješenja koji moramo pretražiti. Problem kod računanja rezolventa je veliki porast u broju klauzula dobivene formule što nam otežava daljnje računanje i produljuje vrijeme potrebno da pronađemo interpretaciju za koju je ulazna formula istinita.
Primjenom jedinične rezolucije mi iz zadanog skupa klauzula, odnosno formule, lociramo klauzule koje sadrže samo jedan literal i čija je istinitosna vrijednost stoga određena zadanom formulom. Ako ne definiramo vrijednost varijable u toj klauzulu na točno određenu, ulazna formula neće biti istinita. Pomoću takvih klauzula primjenom postupka jedinične rezolucije pojednostavljujemo preostale klauzule iz zadanog skupa. Postupak jedinične rezolucije ne povečava broj klauzula u skupu, a tako dobiveni skup je istinitosno ekvivalentan kao početni. Postupak računanja jedinične rezolucije se kombinira sa postupkom redukcije formule opisanim u poglavlju 2.3 kojim vršimo eliminacije svih klauzula koje su istinite za trenutnu parcijalnu interpretaciju i eliminiramo sve literale iz klauzula koje ne doprinose istinitosnoj vrijednosti promatrane klauzule.

4.2 Dokaz korektnosti DPLL algoritma

U ovom odjeljku ćemo opisati DPLL algoritam i dokazati njegovu korektnost [5].

Prije nego što dokažemo korektnost DPLL algoritma dokazat ćemo jednu jednostavnu propoziciju čiji dokaz čini bazu za dokaz korektnosti DPLL algoritma.

Propozicija 4.2.1. *Neka je F skup klauzula, A_1 skup svih propozicionalnih varijabli i $p \in A_1$ neka propozicionalna varijabla. F je ispunjiva ako i samo ako je ispunjiva jedna od formula $F \cup \{p\}$ i $F \cup \{\neg p\}$.*

Dokaz. Neka je v neka totalna interpretacija. Ako $v \models F$, tada pošto je v definirana na svim varijablama iz A_1, $v \models F \cup \{p\}$ ili $v \models F \cup \{\neg p\}$. Obrnuto, ako $v \models F \cup \{p\}$ ili $v \models F \cup \{\neg p\}$ tada $v \models F$. □

Korektnost DPLL algoritma direktno slijedi iz Propozicija 4.1.6 i 4.2.1.

Naš pseudokod DPLL algoritma podrazumijeva da imamo implementiranu funkciju `BCP` koja računa skup BCP zadanog skupa klauzula F. Skup BCP smo definirali u prethodnom odjeljku. Pretpostavljamo da imamo funkciju `selectLit(F)` koja vraća literal iz skupa Lit_F kada je Var_F neprazan. Ovu funkciju ćemo koristiti da bi pronašli literal kojemu još nije pridružena vrijednost. Treba nam i funkcija `reduce(F,v)` koja za ulazni skup klauzula F i skup literala v vrši redukciju skupa F po v. Već smo napomenuli da skup BCP ne mora biti konzistentan. U tome slučaju će nam redukcija formule sadržavati prazan skup.

Navodimo psudokod DPLL algoritma koji ispituje je li zadana formula logike sudova ispunjiva međutim ne pronalazi interpretaciju za koju je ona istinita [5].

```
Algoritam DPLLdec

Ulaz: Skup klauzula F.
Izlaz: Odluka, je li F ispunjiva.
G = F;
if(F sadrži praznu klauzulu)
    return('Ulazna formula nije ispunjiva');
  v=BCP(G);
if(v sadrži par suprotnih literala (l, l̄))
    return('Ulazna formula nije ispunjiva');
  G=reduce(G,v);
  if(G = ∅)
    return('Ulazna formula je ispunjiva');
  else{
       l =selsectLit(G);
       DPLLdec(G ∪ {l});
       DPLLdec(G ∪ {l̄});
      }
```

U sljedećoj propoziciji dokazujemo da `DPLL` algoritam točno ispituje istinitost logičke formule.

Propozicija 4.2.2. *Ako je F ispunjiv skup klauzula tada `DPLLdec(F)` vraća 'Ulazna formula je ispunjiva'. Inače vraća 'Ulazna formula nije ispunjiva'.*

Dokaz. Propoziciju dokazujemo indukcijom po n, broju varijabli iz skupa F kojima nije dodijeljena istinitosna vrijednost.

1. Dokažimo da je izlaz algoritma točan za $n = 0$. Ako je $F = \emptyset$ algoritam vraća 'Ulazna formula je ispunjiva'. Ako je $F = \{\emptyset\}$ algoritam vraća 'Ulazna formula nije ispunjiva'.

2. Pretpostavimo da algoritam vraća točan odgovor za sve ulazne formule u KNF s najviše n varijabli kojima još nije dodijeljena istinitosna vrijednost.

3. Dokažimo da algoritam vraća točan dogovor za sve ulazne formule u KNF koje imaju $n + 1$ varijablu kojima još nije dodijeljena istinitosna vrijednost. Moramo paziti na to je li skup $BCP(F)$ konzistentan ili nije, stoga za $n > 0$ imamo dvije mogućnosti:

   - Skup $BCP(G)$ je konzistentan i $BCP(G) \neq \emptyset$. U ovom slučaju se broj varijabli u redukciji $G =$`reduce`(G, v), stoga i u $G \cup \{x\}$ i $G \cup \{\bar{x}\}$ smanjuje. Primjenom pretpostavke indukcije zaključujemo da će algoritam `DPLLdec` vratiti ispravan odgovor.

     Ako je $BCP(G) = \emptyset$ tada su $BCP(G \cup \{x\})$ i $BCP(G \cup \{\bar{x}\})$ neprazni. Pri računanju redukcije opada broj varijabli kojima nije dodijeljena istinitosna vrijednost i možemo primijeniti pretpostavku indukcije.
   - Moguće je da je skup BCP nekonzistentan. U takvom slučaju F nije ispunjiv i algoritam vraća 'Ulazna formula nije ispunjiva'.

□

U nastavku definiramo varijantu DPLL algoritma koja za ispunjivu formulu pronalazi interpretaciju za koju je ona istinita (više detalja može se pronaći u [5]). Kod ovoga algoritma moramo održavati i parcijalnu interpretaciju koja se mijenja kako se krećemo po stablu pretraživanja. Definirati ćemo algoritam `DPLLsearch`(F, v) koji kao ulaz prima formulu u KNF i parcijalnu interpretaciju v. Algoritam traži proširenje parcijalne interpretacije v u parcijalnu interpretaciju ω takvu da:

1. $v \leq_k \omega$
2. $\omega(F) = 1$

`Algoritam DPLLsearch`

`Ulaz: Skup klauzula` F`, parcijalna interpretacija` v`.`
`Izlaz: Nekonzistentan skup literala ako ne postoji proširenje` ω `od` v
`takvo da vrijedi` $\omega(C) = 1$ `za svaku klauzulu` $C \in F$ `ili`
`parcijalna interpretacija` ω `koja proširuje` v `i za koju vrijedi`

```
       ω(C) = 1 za svaku klauzulu C ∈ F.
if(F sadrži praznu klauzulu)
    return {p, p̄};
 G =Red(F, v);
 if(G = ∅)
     return v
 else{
     l =selectLit(G);
      if(DPLLsearch(G ∪ {l}, v) ne vraća kontradikciju)
                    return DPLLsearch(F, v ∪ {l});
       else
                    return DPLLsearch(F, v ∪ {l̄});

           }
```

Korektnost algoritma dokazujemo u sljedećoj propoziciji:

Propozicija 4.2.3. *Algoritam* `DPLLsearch`*(F, v) pronalazi proširenje ω parcijalne interpretacije v takvo da je svaka klauzula iz skupa klauzula F istinita za ω, ako takva interpretacija postoji. Posebno, za ulaz v = ∅, algoritam* `DPLLsearch`*(F, v) pronalazi parcijalnu interpretaciju ω takvu da je svaka klauzula iz skupa klauzula F istinita za ω, ako takva interpretacija postoji.*

Dokaz. Propoziciju dokazujemo indukcijom po broju varijabli kojima još nije pridjeljena istinitosna vrijednost.

1. Za $n = 0$ svim varijablama je pridružena istinitosna vrijednost. Pri računanju

   $G =$`Red`(F, v), dobivamo $G = \emptyset$, ili $G = \{\emptyset\}$. Ukoliko dobijemo prvi slučaj, formula je ispunjiva i algoritam vraća parcijalnu interpretaciju v za koju je F istinita. U drugom slučaju algoritam vraća nekonzistentni skup literala.

2. Pretpostavimo da algoritam vraća točan odgovor za sve formule u KNF sa najviše n varijabli kojima nije pridružena istinitosna vrijednost.

3. Dokažimo da tvrdnja vrijedi za $n + 1$. Pošto kod $G =$`Red`(F, v) smanjujemo broj varijabli formule barem za jedan, primjenom koraka indukcije dobivamo da algoritam vraća točni izlaz.

□

Primijetimo da navedeni algoritam vraća parcijalnu interpretaciju koja ima svojstvo da su sve klauzule iz F istinite za nju. Ova parcijalna interpretacija ne mora biti definirana na svim varijablama iz F, ali možemo ju trivijalno dodefinirati do interpretacije ω' koja je definirana na svim varijablama iz F. Jedna moguća definicija interpretacije ω' na varijablama skupa klauzula F bi bila:

$$\omega'(p) = \begin{cases} v(p), & \text{ako je } v \text{ definiran u } p \\ 1, & \text{inače} \end{cases}$$

4.3 Poboljšanja DPLL algoritma

U ovom odjeljku ćemo promatrati moguća poboljšanja DPLL algoritma [5]. Navesti ćemo četiri moguća poboljšanja algoritma. Sva četiri načina daju izvrsne rezultate u praksi.

1. Promatranje funkcije `select`. Proban je cijeli niz heuristika za odabir literala i njihovih polarnosti. Sve te heuristike pokušavaju pronaći dio stabla pretraživanja u kojemu ćemo najvjerojatnije pronaći rješenje. Način odabira varijabli utječe na dinamičku konstrukciju stabla pretraživanja.

2. Svaki puta kada dođemo do kontradikcije pri računanju vraćamo se na zadnju točku odabira polariteta literala u stablu. Formalno to znači da $F \cup \{l_1, \ldots, l_k\} \models \bot$. To je ekvivalentno sa $F \models \bar{l}_1 \vee \cdots \vee \bar{l}_k$. Time smo pronašli klauzulu koja je logička posljedica formule F. Ova klauzula nam ne pomaže direktno, međutim možda postoje neke klauzule koje se dobiju pravilom proširivanja iz nje i koje su logičke posljedice formule F. Takve klauzule možemo tada dodati formuli F. Ovo ne mijenja veličinu stabla pretraživanja ili skup varijabli, ali mijenja ponašanje skupa klauzula. Možemo pronaći da su djelovi stabla reducirani, što je pokazano i u praksi. Ovaj postupak ima i svojih mana, svaki dolazak do kontradikcije i vraćanje do čvora u kojemu smo izabirali polaritet literala dovodi do pronalaženja nove klauzule. Broj klauzula stoga raste i zahtijeva dodatnu memoriju, koja je ograničena. Zbog toga postoje razne tehnike kojima se obrađuju tako nastale nove klauzule.

3. Bliska sa učenjem je i tehnika 'backjump' (skok unatrag). Tijekom analize uvjeta, koja se zove **analiza konflikta**, možemo uočiti da je pravi razlog nailaska našeg algoritma na kontradikciju jedan od izbora napravljen prije zadnjega. U tom slučaju možemo reducirati cijeli dio stabla pretraživanja ispod takvog lošeg odabira. Time izbjegavamo barem jedan povratak u čvor odabira što je ogromna vremenska ušteda pošto je stablo pretraživanja eksponencijalno veliko u broju varijabli formule.

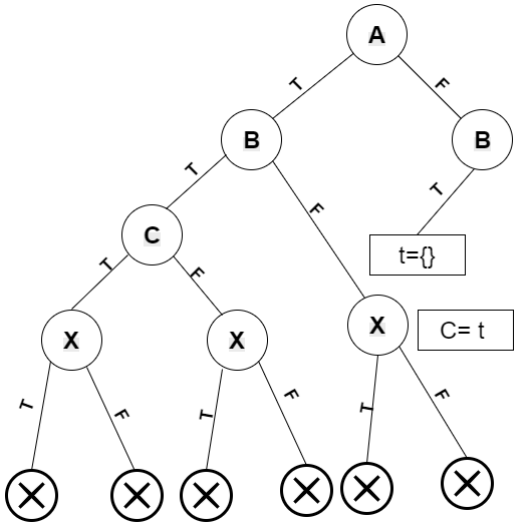
4. Najviše vremena u računanju DPLL algoritma trošimo na računanje skupa *BCP*. Računamo skup *BCP* nakon svakog pridruživanja istinitosne vrijednosti u čvoru odlučivanja. Reduciranjem broja pretraživanja klauzula u potrazi za klauzulom u kojoj su dodijeljene vrijednosti svima osim jednom literalu mogli bi ubrzati rad našega algoritma. Ideja je da promatramo dva literala iz svake klauzule. Dok niti jednom od ta dva literala nije pridijeljena vrijednost, odgovarajuća klauzula neće pomoći pri računanju skupa *BCP*, stoga je pri računanju toga skupa ne trebamo posjetiti. Ako je jedan od ta dva literala poništen (neistinit je za trenutnu interpretaciju), možemo imati jedan od sljedećih slučajeva:

   - Možemo promatrati nova dva literala iz te klauzule, koja još nisu poništena.
   - Ako točno jedan literal u klauzuli nije poništen tada ta klauzula doprinosi računanju skupa *BCP*.
   - Ako su svi literali u klauzuli poništeni, moramo se vratiti po stablu do čvora izabira polariteta litera i izabrati drugu vrijednost.

Navedena poboljšanja, i neka druga, kao razne varijante *lookahead* (promatranje unaprijed), korištenje boljih struktura podataka za spremanje klauzula, uvelike su doprinjeli ogromnom napretku u testiranju ispunjivosti u posljednjih godina.

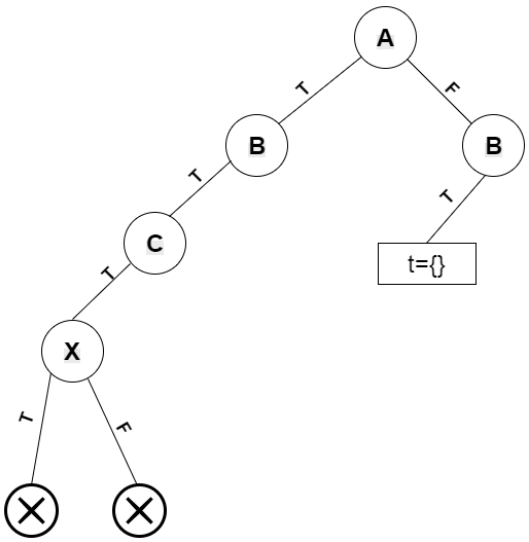
Primjer 4.3.1. *Demonstrirajmo analizu konflikata na jednom primjeru. Neka je zadana formula* $F = (A \vee B) \wedge (B \vee C) \wedge (\neg A \vee \neg X \vee Y) \wedge (\neg A \vee X \vee Z) \wedge (\neg A \vee \neg Y \vee Z) \wedge (\neg A \vee X \vee \neg Z) \wedge (\neg A \vee \neg Y \vee \neg Z)$.

DPLL algoritam bez analize konflikta bi napravio sljedeće korake:



Slika 4.1. Kronološko pretraživanje. Slika je bazirana na slici 3.6 iz [1].

Primjenjujući nekronološko pretraživanje nakon definiranja $I(A) = 1$, $I(B) = 1$, $I(C) = 1$ *i* $I(X) = 1$, *jediničnom rezolucijom dobivamo* $I(Y) = 1$, $I(Z) = 1$ *i primjetimo da smo dobili praznu klauzulu. Definiramo konfliktni skup (skup interpretacija koje su dovele do kontradikcije) kao* $\{I(A) = 1,\ I(X) = 1,\ I(Y) = 1,\ I(Z) = 1\}$. *Nekronološko pretraživanje se vraća u čvor X i proba definirati drugu vrijednost toj varijabli. Sada dolazimo do novog konflikta. Konfliktni skup će sada biti* $\{I(A) = 1,\ I(X) = 1,\ I(Z) = 1\}$. *Pošto smo ispitali sve mogućnosti za varijablu X, nekronološko pretraživanje ispituje vrijednosti varijable A i definira* $I(A) = 0$. *Dobiveno stablo je:*



Slika 4.2. Nekronološko pretraživanje.

4.4 Pojednostavljivanje traženja odgovarajuće interpretacije

U ovom odjeljku ćemo reducirati problem traženja interpretacije za koju je ulazna formula istinita na problem odlučivanja je li ulazna formula istinita.

Naš cilj je pronaći interpretaciju za koju je ulazni skup klauzula F istinit, ako takva interpretacija postoji. Pretpostavimo da imamo algoritam $\mathcal{A}$ koji ispituje je li ulazna formula

F ispunjiva, ali ne vraća interpretaciju za koju je ona istinita. Promatrat ćemo kako pretvoriti algoritam $\mathcal{A}$ u algoritam koji računa takvu interpretaciju. Na početku pokrenemo algoritam $\mathcal{A}$ da bi ispitali je li ulazna formula F ispunjiva. Ako $\mathcal{A}$ vrati 'nije ispunjiva', mi također vratimo 'nije ispunjiva'. Inicijaliziramo skup U varijabli kojima nije pridružena istinitosna vrijednost u Var_F. Ako $\mathcal{A}$ vrati 'ispunjiva', izaberemo literal l iz U i formiramo jediničnu klauzulu $\{l\}$ koju dodajemo skupu klauzula F. Brišemo literal l iz U. Pokrećemo $\mathcal{A}$ na toj novoj formuli. Ako $\mathcal{A}$ vrati 'ispunjiva', supstituiramo F sa $F \cup \{\{l\}\}$ i definiramo $v(l) = 1$. Ako $\mathcal{A}$ vrati 'nije ispunjiva', supstituiramo F u skup klauzula $F \cup \{\{\bar{l}\}\}$ i stavljamo $v(l) = 0$. Postupak nastavljamo dok U ne postane prazan skup. Dobivena interpretacija v je interpretacija za koju je formula F istinita.

Ovu tehniku smo spomenuli pošto je zatvorenje po rezoluciji primjer algoritma koji ispituje je li ulazna formula ispunjiva i ne računa interpretaciju za koju je ta formula istinita. Ovo je jedan od načina kojim možemo takav algoritam pretvoriti u algoritam koji računa interpretaciju za koju je ulazna formula istinita.

4.5 Vizualizacija DPLL algoritma

U ovom odjeljku ćemo vizualizirati rad DPLL algoritma koristeći aplikaciju DPvis razvijenu na Sveučilištu u Tübingenu, Njemačka [10].

Definicija 4.5.1. *Neusmjereni graf je uređeni par* (G, R)*, gdje je* G *neprazan skup, a* $R, \subseteq G \times G$ *simetrična relacija.*

Definicija 4.5.2. *Neka je* F *proizvoljna formula. Graf međuovisnosti* $G(F)$ *formule* F *je neusmjereni graf koji sadrži jedan čvor za svaku propozicionalnu formulu formule* F *i brid koji spaja svaka dva čvora za koje se odgovarajuće varijable nalaze u istoj klauzuli formule* F*.*

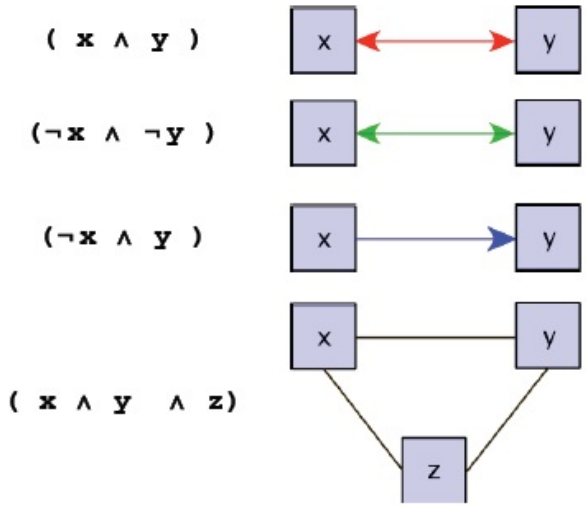
Bridovi u grafu međuovisnosti koji spajaju čvorove čije odgovarajuće varijable se nalaze u klauzulama veličine dva su označeni na poseban način, pošto je $2 - SAT$ rješiv u polinomnom vremenu.

1. Čvorovi u grafu koji odgovaraju nenegiranim varijablama i koji se nalaze u klauzuli veličine dva, su povezani bridom prikazanim kao crvena dvostrana strelica.

2. Čvorovi u grafu koji odgovaraju negiranim varijablama i koji se nalaze u klauzuli veličine dva su povezani bridom prikazanim kao zelena dvostrana strelica.

3. Čvorovi u grafu za koje vrijedi da je jedan pridružen nenegiranoj varijabli, a drugi negiranoj i oni se nalaze u klauzuli veličine dva su povezani bridom prikazanim kao plava jednostrana strelica koja pokazuje iz negirane varijable u nenegiranu.

4. Čvorovi u grafu koji odgovaraju varijablama formule koje se nalaze u klauzuli veličine veće od dva su povezani crnim neusmjerenim bridom.

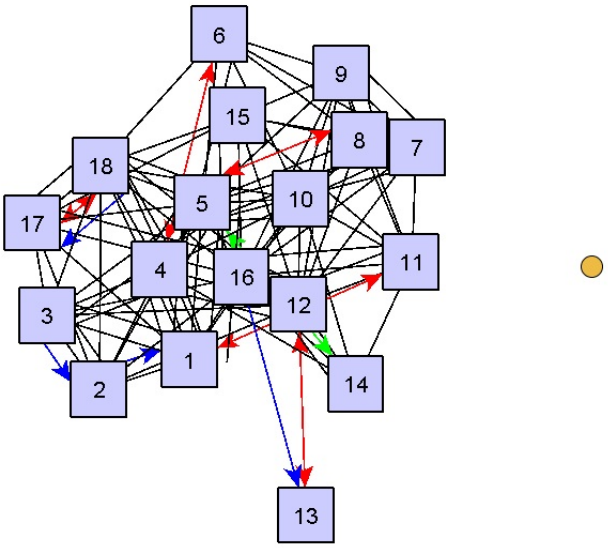


Slika 4.3. Prikaz klauzula formule u grafu međuovisnosti

Primjer 4.5.3. *Vizualizirajmo rad algoritma na formuli:*
$F = (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_1 \vee x_3 \vee \neg x_4 \vee x_5) \wedge (\neg x_1 \vee x_4 \vee \neg x_5) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6 \vee x_7 \vee \neg x_8 \vee x_9 \vee \neg x_{10}) \wedge (x_{10} \vee x_{11} \vee \neg x_{12} \vee \neg x_{14} \vee x_{16}) \wedge (x_1 \vee x_{11}) \wedge (x_{12} \vee x_{13}) \wedge (x_4 \vee \neg x_{12} \vee x_{14}) \wedge (\neg x_2 \vee x_3 \vee \neg x_4 \vee x_{12}) \wedge (x_{15} \vee \neg x_{16} \vee \neg x_{17} \vee \neg x_{18}) \wedge (x_{17} \vee x_{18}) \wedge (x_{17} \vee \neg x_{18}) \wedge (x_5 \vee \neg x_7 \vee x_8) \wedge (\neg x_4 \vee \neg x_6 \vee x_{18}) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee \neg x_5 \vee x_{16} \vee x_{17} \vee x_{18}) \wedge (x_7 \vee \neg x_8 \vee x_9 \vee \neg x_{11} \vee \neg x_{12} \vee \neg x_{15} \vee x_{16}) \wedge (x_{13} \vee \neg x_{16}) \wedge (\neg x_5 \vee \neg x_{14}) \wedge (x_6 \vee x_8) \wedge (x_1 \vee x_4 \vee x_5 \vee x_{10} \vee \neg x_{18})$.

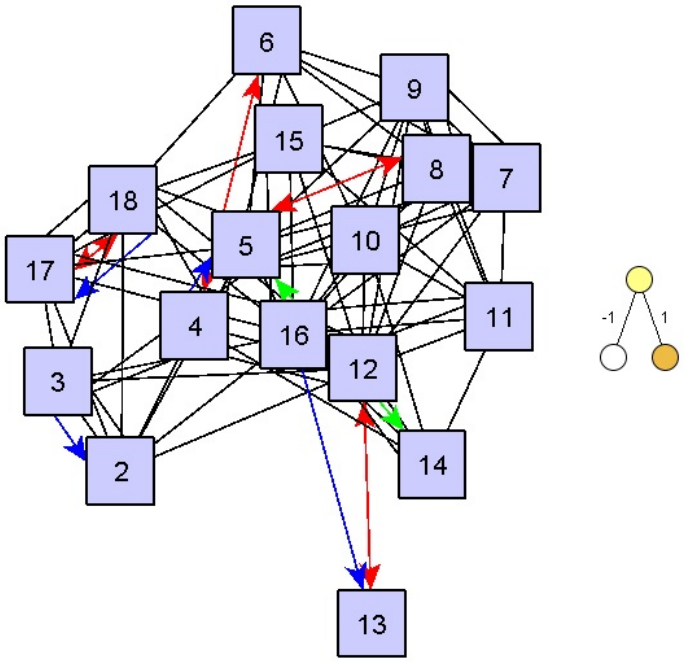
Promatrati ćemo izgled grafa međuovisnosti i stablo pretraživanja koje dobijemo pri pridjeljivanju vrijednosti pojedinoj varijabli.

- *Početni izgled za zadanu formulu je:*



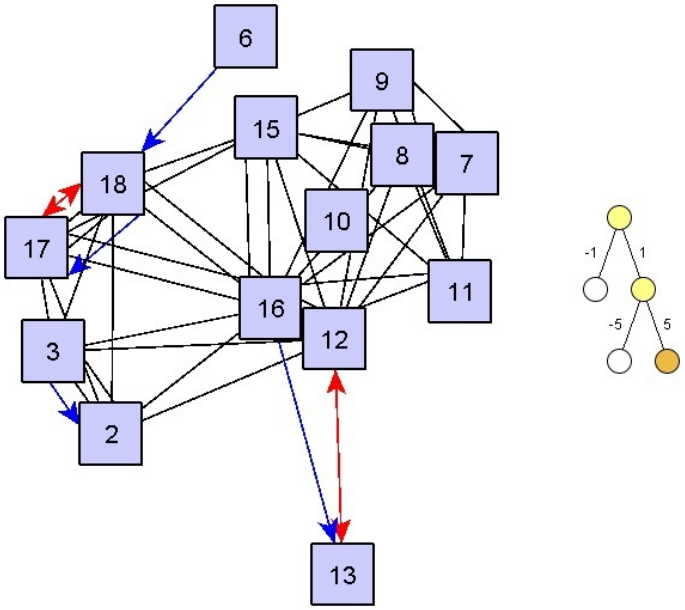
Slika 4.4. Prikaz početnog stanja

- *Definiramo* $I(x_1) = 1$.



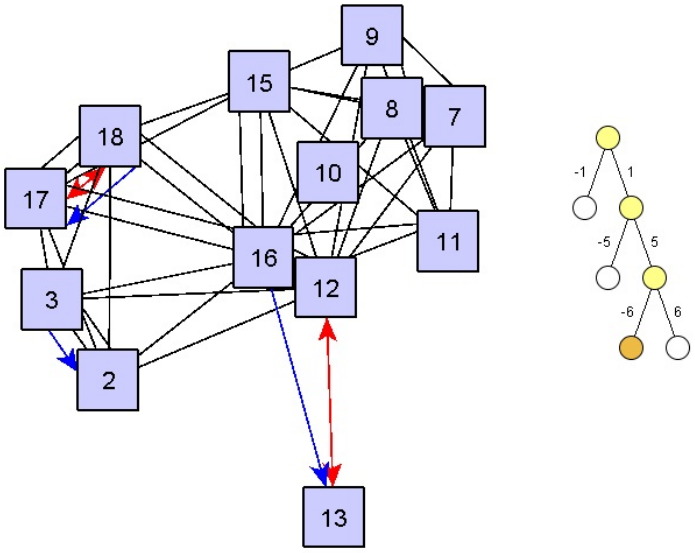
Slika 4.5. Prikaz stanja nakon definiranja $I(x_1) = 1$

- *Definiramo* $I(x_5) = 1$.



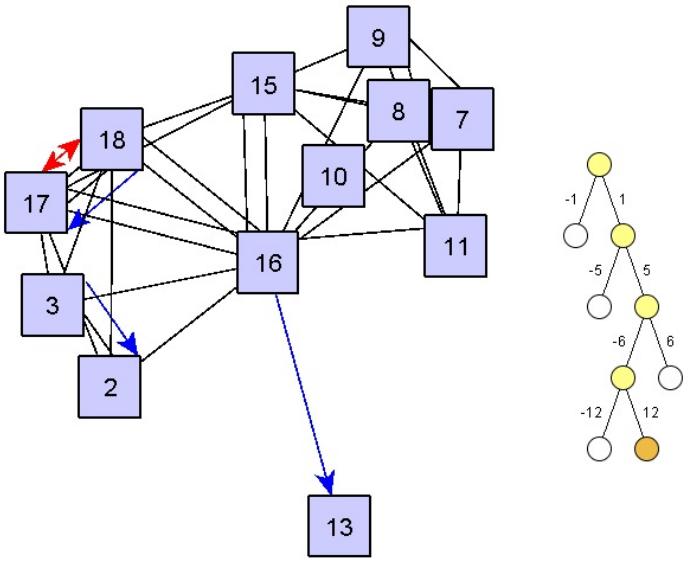
Slika 4.6. Prikaz stanja nakon definiranja $I(x_5) = 1$

- *Iz grafa međuovisnosti vidimo da se varijabla* x_6 *javlja u trenutnom koraku u samo jednoj klauzuli i to negirana. Definiranjem* $I(x_6) = 0$ *ćemo eliminirati tu klauzulu iz formule.*



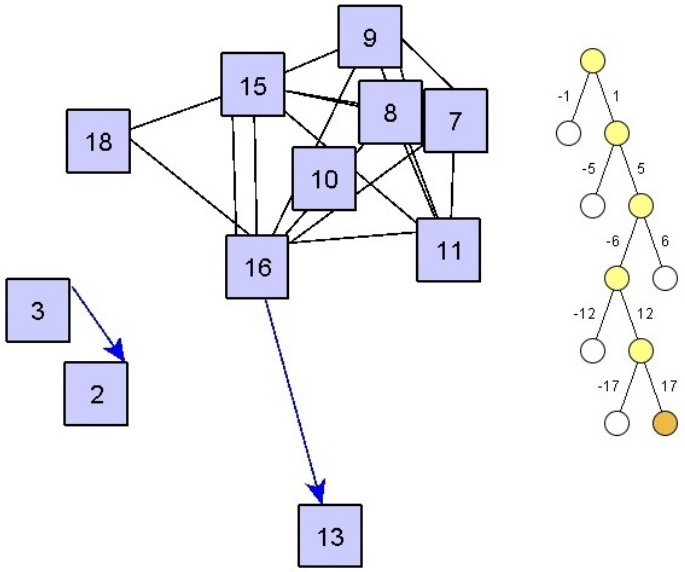
Slika 4.7. Prikaz stanja nakon definiranja $I(x_6) = 0$

- *Definiramo* $I(x_{12}) = 1$.



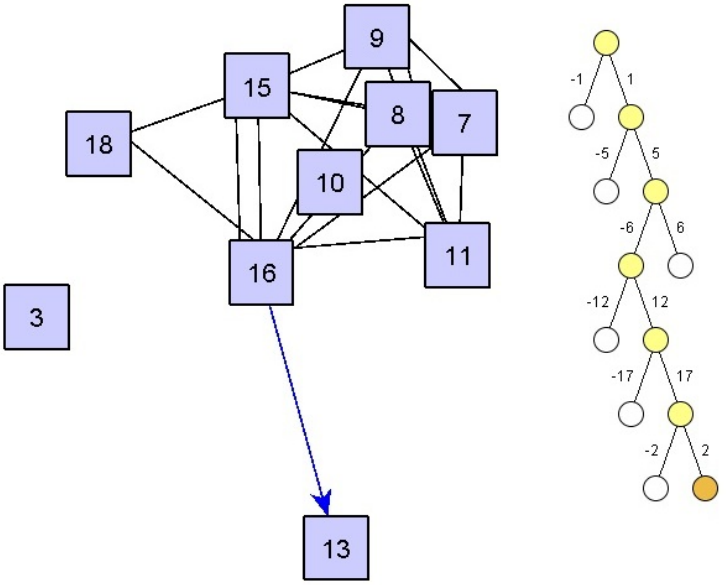
Slika 4.8. Prikaz stanja nakon definiranja $I(x_{12}) = 1$

- *Iz grafa međuovisnosti vidimo da postoji klauzula veličine dva u kojoj se varijable* x_{17} *i* x_{18} *obje javljaju bez negacije. Međutim, postoji i klauzula u kojoj se varijabla* x_{18} *javlja negirana i* x_{17} *bez negacije. Definiranjem* $I(x_{17}) = 1$ *eliminirat ćemo obje klauzule iz formule.*



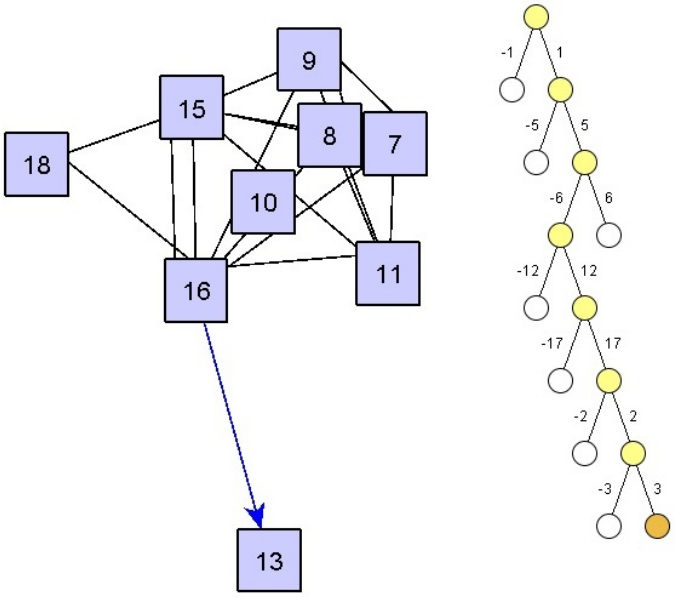
Slika 4.9. Prikaz stanja nakon definiranja $I(x_{17}) = 1$

- *Definiranjem* $I(x_2) = 1$ *sigurno eliminiramo dodatnu klauzulu iz formule pošto u grafu međuovisnosti imamo plavi usmjereni brid iz čvora* 3 *u čvor* 2.



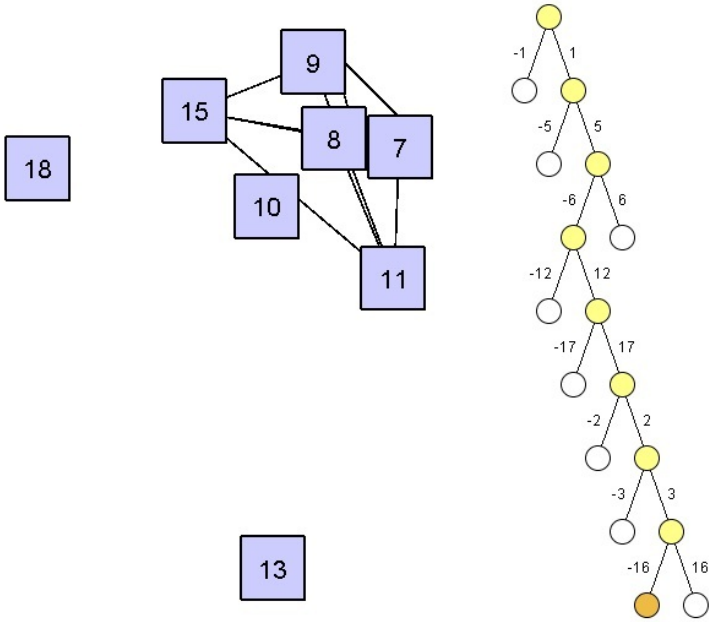
Slika 4.10. Prikaz stanja nakon definiranja $I(x_2) = 1$

- *Na grafu uočavamo da čvor* 3 *nije povezan s niti jednim drugim čvorom u grafu, stoga definiramo* $I(x_3) = 1$.



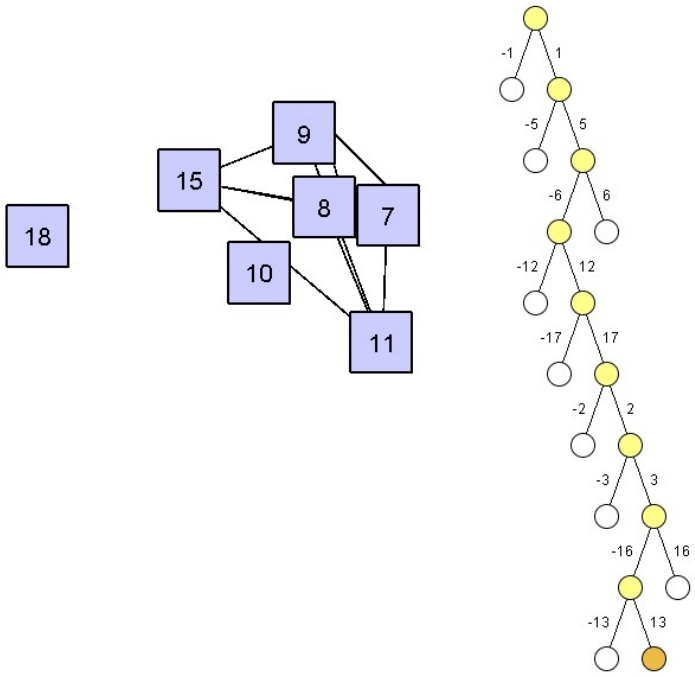
Slika 4.11. Prikaz stanja nakon definiranja $I(x_3) = 1$

- *Definiranjem* $I(x_{16}) = 0$ *eliminiramo barem jednu klauzulu iz formule.*



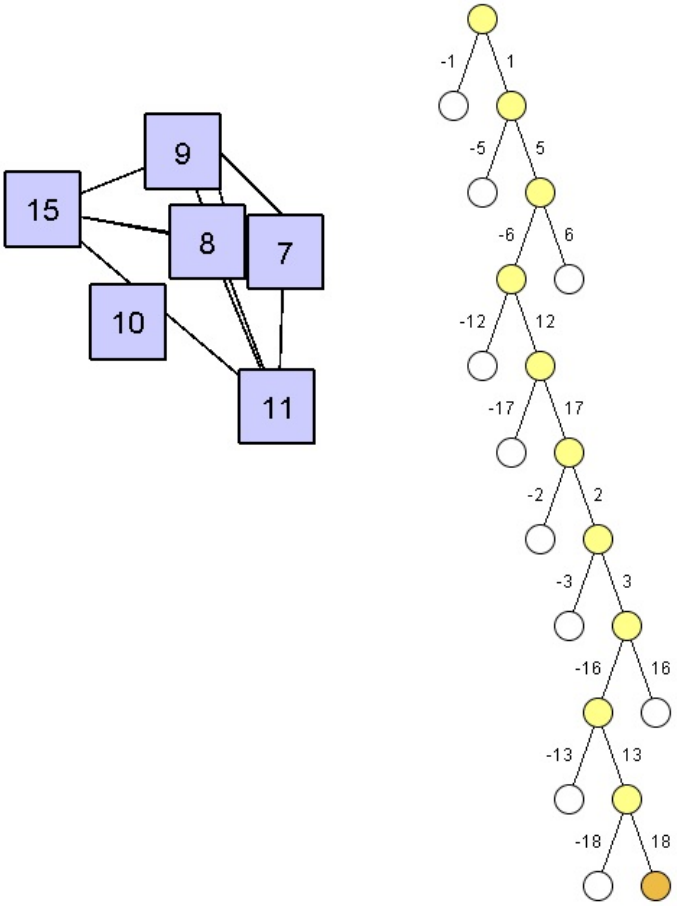
Slika 4.12. Prikaz stanja nakon definiranja $I(x_{16}) = 0$

- *Definiramo* $I(x_{13}) = 1$.



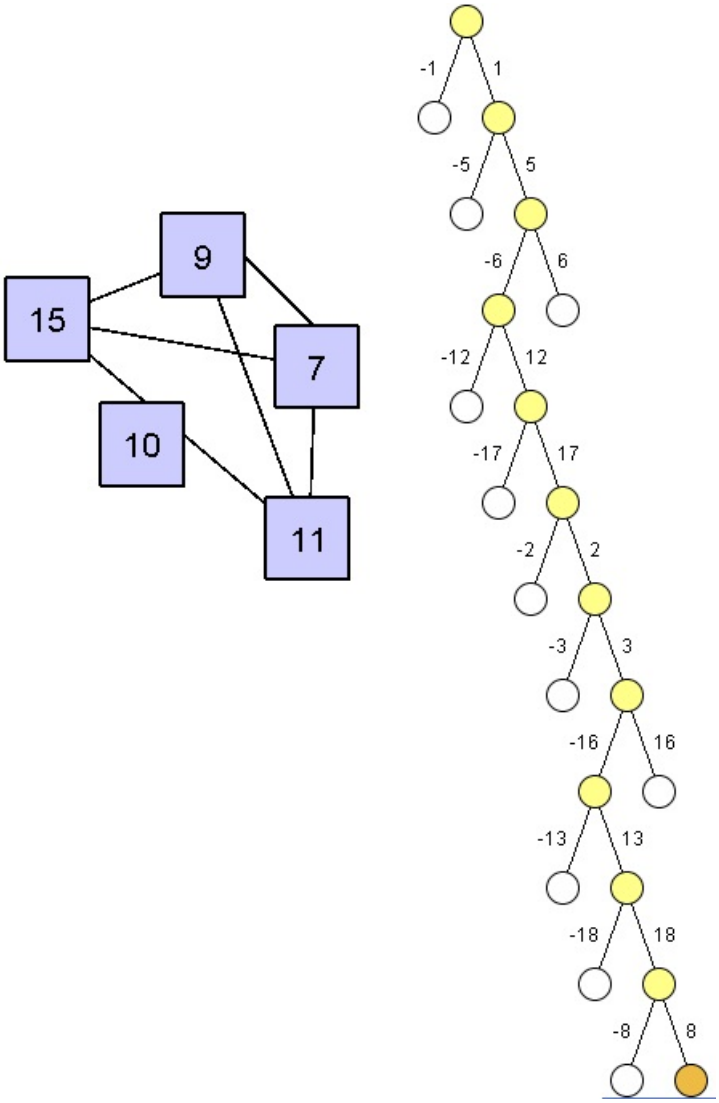
Slika 4.13. Prikaz stanja nakon definiranja $I(x_{13}) = 1$

- *Definiramo* $I(x_{18}) = 1$.



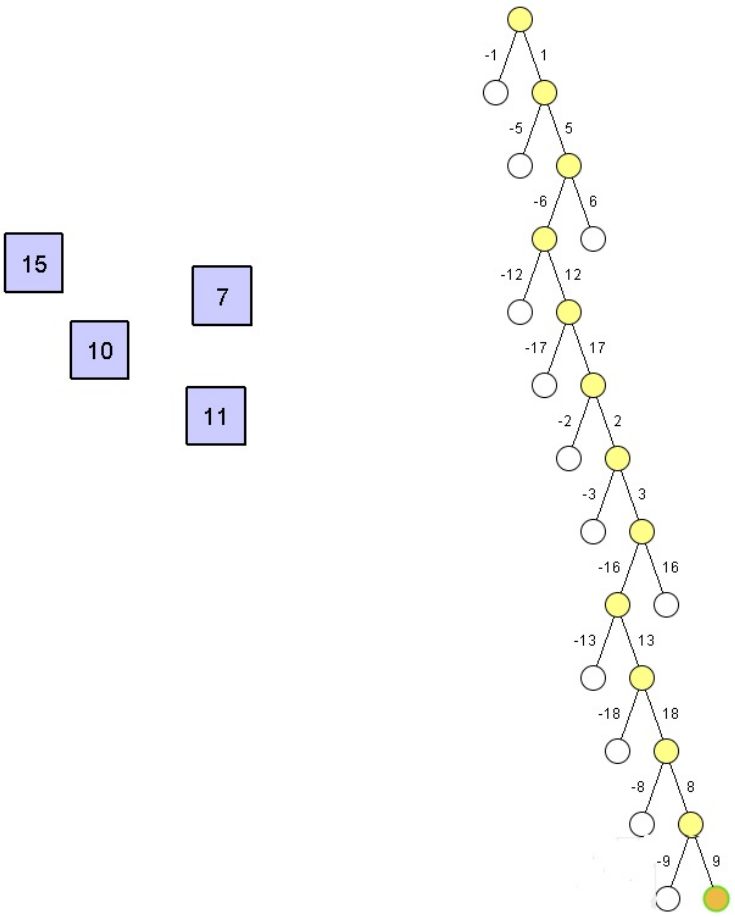
Slika 4.14. Prikaz stanja nakon definiranja $I(x_{18}) = 1$

- *U ovome trenutku formula ne sadrži niti jednu klauzulu koja sadrži manje od tri varijable. Definiramo* $I(x_8) = 1$.



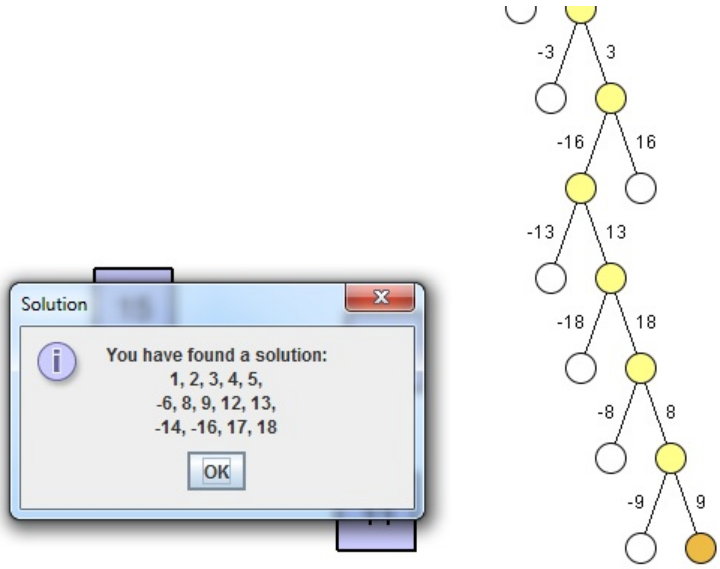
Slika 4.15. Prikaz stanja nakon definiranja $I(x_8) = 1$

- *Definiramo* $I(x_9) = 1$.



Slika 4.16. Prikaz stanja nakon definiranja $I(x_9) = 1$

- *Time smo pronašli interpretaciju za koju je ulazna formula istinita. Istinitost formule ovisi samo o varijablama kojima smo pridjelili vrijednost, za preostale varijable možemo definirati* $I(x_i) = 1$.



Slika 4.17. Dobiveno rješenje.

Poglavlje 5

Stohastički algoritmi

5.1 UnitWalk algoritam

U ovom odjeljku predstavljamo UnitWalk algoritam.

UnitWalk algoritam je nepotpuni algoritam za rješavanje problema SAT. UnitWalk algoritam koristi lokalno traženje, kao kod eksperimentalno najboljih algoritama iz skupine WalkSAT algoritama, u kombinaciji s eliminacijom jediničnih klauzula (klauzule koja sadrži samo jedan literal).

Heuristika za promjenu vrijednosti generirane interpretacije na nekoj varijabli formule je razvijena prema algoritmu Paturi, Pudlák, Saks i Zane-a ([7]). Jezgru toga algoritma čini funkcija koju ćemo sada opisati.
Za ulaznu formulu F generiramo interpretaciju na slučajan način i generiramo slučajnu permutaciju varijabli formule F. Uzmimo varijable po redoslijedu određenom permutacijom i za svaku izabranu varijablu, ako se varijabla v nalazi u jediničnoj klauzuli tada namjestimo vrijednost interpretacije I tako da vrijedi $I(v) = 1$. Vrijednost interpretacije je određena jediničnom klauzulom. U suprotnom uzmemo vrijednost slučajno generirane interpretacije. Napravimo odgovarajuću supstituciju u formuli u oznaci $F[v \leftarrow True]$ ili $F[v \leftarrow False]$ tako da iz formule eliminiramo sve klauzule koje sadrže varijablu v i za koje vrijedi $I(v) = 1$, a iz svih klauzula koje sadrže varijablu $\neg v$ eliminiramo tu varijablu iz klauzule.

Paturi, Pudlák i Zane ([6]) pokazuju da ta procedura pronalazi interpretaciju za koju je ulazna formula istinita, za 3-KNF formulu s vjerojatnošću najmanje $O(2^{-2n/3})$. Ponavljajući proceduru $O(2^{2n/3})$ puta dobivamo konstantnu vjerojatnost greške. Algoritam se može derandomizirati tako da ga pokrenemo na $2^{2n/3}$ interpretacija i polinomni broj per-

mutacija.

Navedenu proceduru koristimo u UnitWalk algoritmu za poboljšavanje slučajno generirane interpretacije. Ako nam se pri izračunavanju pojave jedinične klauzule odmah ih obrađujemo. Ako imamo više od jedne jedinične klauzule tada jediničnu klauzulu koju ćemo obrađivati biramo na slučajan način. Ako postoje dvije jedinične klauzule suprotnih istinitosnih vrijednosti tada ne mijenjamo interpretaciju jer takva promjena nas ne bi dovela do rješenja, formula bi i dalje bila neistinita za modificiranu interpretaciju. Ako smo pridružili vrijednost svim varijablama formule i nismo modificirali interpretaciju niti jednom, izaberemo slučajnu varijablu i promijenimo vrijednost interpretacije na njoj. Promjena vrijednosti interpretacije na varijabli v se definira kao: $I(v) = 1 - I(v)$.

Osnovni UnitWalk algoritam

UnitWalk algoritam su razvili Edward A. Hirsch, *Steklov Institute of Mathematics at St. Petersburg* i Arist Kojevnikov *St. Petersbutg State Univerity, Department of Mathematics and Mechanics, St. Petersburg* ([3]).

Kao tipični algoritam lokalnog pretraživanja algoritam generira na slučajan način početnu interpretaciju i modificira je korak po korak. Glavna razlika u odnosu na ostale algoritme koji koriste slučajne šetnje je taj što UnitWalk algoritam modificira i ulaznu formulu. Slučajna šetnja je podijeljena na periode. Tijekom jednog perioda se napravi barem jedna modifikacija generirane interpretacije, odnosno flip. Period počinje izborom slučajne permutacije varijabli. Nakon toga algoritam korak po korak modificira kopiju ulazne formule i generiranu interpretaciju. U svakom koraku algoritam supstituira vrijednost varijable u formulu, mijenja formulu G u $G[v \leftarrow t]$ za varijablu v i istinitosnu vrijednost t. Ako postoje jedinične klauzule tada uzimamo v iz jedne od jediničnih klauzula. Ako nema jediničnih klauzula, algoritam supstituira vrijednost varijable koja je sljedeća po generiranoj permutaciji. Ako se varijabla v nalazi u jediničnoj klauzuli i vrijedi $I(v) = 1$ za trenutnu interpretaciju i ne postoji klauzula u kojoj se nalazi $\neg v$, tada mijenjamo vrijednost interpretacije na v, odnosno definiramo $I(v) = 1 - I(v)$. Ako završi period bez mijenjanja vrijednosti generirane interpretacije algoritam bira varijablu na slučajan način i mijenja vrijednost interpretacije na njoj. Nakon završetka perioda algoritam generira novu slučajnu permutaciju varijabli, postavlja trenutnu formulu (u koracima algoritma mijenjamo privremenu kopiju ulazne formule) na ulaznu i izračunava novi period s trenutno generiranom permutacijom. Broj perioda je ograničen s MAX_PERIODS(F) do $+\infty$ (u ovom slučaju algoritam neće nikada stati ako ulazna formula nije ispunjiva).

Sada navodimo pseudokod osnovnog UnitWalk algoritma [3].

```
Algoritam UnitWalk.
Ulaz: formula F u KNF.
Izlaz: Interpretacija I takva da I(F) = 1 ili rješenje nije pronađeno.

 for i = 1 to MAX_TRIES(F)
             A=slučajno generirana interpretacija
               za formulu koja ima n varijabli;
              for j = 1 to MAX_PERIODS(F)
                  π := slučajna permutacija od 1... n;
                  G:=F;
                  f:=0;
                 for p = 1 to n do
                        while G sadrži jedinične klauzule radi:
                              -Izaberi jediničnu klauzulu {x_j} ili
                               {¬ x_j} iz G na slučajan način;
                              -Ako je vrijednost interpretacije A
                                na toj klauzuli 0 i G ne sadrži
                                suprotnu jediničnu klauzulu, tada
                                A[j] = 1 − A[j] i f := 1;
                              -G := G[x_j ← A[j]];
                        Ako se varijabla x_π[i] još javlja u
                        formuli G, tada G := G[x_π[i] ← A[π[i]]].
                  Ako G ne sadrži niti jednu klauzulu
                  I(G) = 1, return A;
                  Ako f = 0, izaberi j na slučajan način
                  iz 1... n i napravi zamjenu A[j] = 1 − A[j].
   return Rješenje nije pronađeno!
```

U nastavku iskazujemo i dokazujemo teorem koji pokazuje da je osnovni UnitWalk algoritam vjerojatnosno aproksimativno potpun, ako stavimo MAX_PERIODS(F) na $+\infty$ i MAX_TRIES(F) na 1. Tada za svaku ispunjivu formulu i svaku početno generiranu interpretaciju, UnitWalk pronalazi rješenje problema s vjerojatnošću 1.

Definicija 5.1.1. *Kažemo da je neki algoritam P za rješavanje problema ispunjivosti logičke formule* ***vjerojatnosno aproksimativno kompletan*** *ako P za svaku ispunjivu formulu F, sa vjerojatnošću* 1*, u konačno mnogo koraka pronalazi interpretaciju za koju je F istinita.*

Definicija 5.1.2. ***Hammingova udaljenost*** *između dvije interpretacije A i B je broj varijabli na kojima se vrijednosti A i B razlikuju.*

Primjer 5.1.3. *Pretpostavimo da imamo dvije interpretacije definirane s:* $A = \{p, q, \neg z, k, \neg l\}$, $B = \{\neg p, q, \neg z, k, l\}$. *Tada je Hammingova udaljenost* $d(A, B) = 2$ *pošto se vrijednost interpretacija A i B razlikuje na varijablama p i l.*

Teorem 5.1.4. *Algoritam UnitWalk je vjerojatnosno aproksimativno potpun.*

Dokaz. Dokazujemo da se za proizvoljnu interpretaciju A i za svaku interpretaciju S za koju je formula F istinita tijekom jednog perioda ili Hammingova udaljenost između A i S smanjuje s odozdo ograničenom vjerojatnošću ili algoritam vraća interpretaciju za koju je F istinita. Promatrajmo proizvoljnu interpretaciju S za koju je formula F istinita i interpretaciju A koja je na Hammingovoj udaljenosti d od interpretacije S. Konstruiramo permutaciju π takvu da ako je izabrana na početku perioda i početna interpretacija je A, tada će interpretacija dobivena na kraju perioda biti bliže interpretaciji S od interpretacije A (primijetimo da se svaka permutacija izabire s vjerojatnošću $\frac{1}{n!}$). Neka je $\pi[1] = i_1, \ldots, \pi[n-d] = i_{n-d}$, gdje $x_{i_1}, \ldots, x_{i_{n-d}}$ predstavljaju varijable na kojima interpretacije A i S imaju identične vrijednosti. Očito se vrijednosti tih varijabli neće mijenjati kroz period. Vrijednosti interpretacija se na ostalim varijablama razlikuju. Ako promijenimo vrijednost interpretacije A na barem jednoj varijabli na kojoj se razlikuje od interpretacije S kroz period, dokazali smo tvrdnju. Jedino na tim varijablama može biti vrijednost interpretacije A uvjetovana jediničnim klauzulama. Ako nije promijenjena vrijednost interpretacije tijekom perioda niti na jednoj varijabli i nije pronađena interpretacija za koju je F istinita, UnitWalk izabire varijablu na slučajan način i mijenja vrijednost interpretacije A za tu varijablu. S vjerojatnošću najmanje $\frac{1}{n}$ algoritam izabire varijablu čija vrijednost u A je različita od vrijednosti odgovarajuće varijable u S. Svaki period smanjuje Hammingovu udaljenost između A i S s vjerojatnošću $\frac{1}{n \cdot n!}$. Vjerojatnost da ćemo dobiti interpretaciju za koju je ulazna formula istinita u najviše n koraka je stoga najmanje $p(n) = \frac{1}{(n \cdot n!)^n}$ (bez obzira na početno generiranu interpretaciju). Ukupna vjerojatnost da ćemo dobiti interpretaciju za koju je ulazna formula istinita je najmanje $p(n)+(1-p(n))p(n)+(1-p(n))^2 p(n)+\cdots = 1$. □

Primjer 5.1.5. *Demonstrirat ćemo rad osnovnog UnitWalk algoritma na formuli:* $F = (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg x_5) \wedge (\neg x_2 \vee x_4 \vee x_5) \wedge (x_2 \vee x_4 \vee x_5)$.

Pretpostavimo da je početno $MAX\_TRIES = 2$, $MAX\_PERIODS = 5$. *Zbog lakše vizualizacije, interpretaciju ćemo reprezentirati kao string* 0 *i* 1 *takav da bit na poziciji i u stringu predstavlja trenutnu istinitosnu vrijednost varijable* x_i.

+ *Neka je* $I = 00101$ *slučajno generirana interpretacija.*
  - *Neka je* $\pi = (15423)$ *slučajno generirana interpretacija u prvom periodu.*
  - $G := F$.

* *Vršimo redukciju formule varijablom* x_1 *i dobivamo* $G = (x_2 \vee x_3 \vee \neg x_5) \wedge (\neg x_2 \vee x_4 \vee x_5) \wedge (x_2 \vee x_4 \vee x_5)$.
* *Vršimo redukciju formule varijablom* x_5 *i dobivamo* $G = (x_2 \vee x_3)$.
* *Varijabla* x_4 *je već eliminirana iz formule pa nastavljamo s postupkom eliminacije.*
* *Vršimo redukciju formule varijablom* x_2 *i dobivamo* $G = (x_3)$.
  ** *U ovom trenutku formula G sadrži jednu jediničnu klauzulu* (x_3)*, stoga ulazimo u while petlju algoritma. Pošto vrijedi* $I(x_3) = 1$*, vršimo eliminaciju formule* x_3 *iz formule G i dobivamo* $G = ()$.

- *Pošto G ne sadrži niti jednu klauzulu, algoritam vraća rješenje I.*

Demonstrirajmo rad algoritma u slučaju da ulazna formula nije istinita za slučajno generiranu interpretaciju u periodu.

Primjer 5.1.6. *Demonstrirat ćemo rad osnovnog UnitWalk algoritma na formuli:* $F = (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg x_5) \wedge (\neg x_2 \vee x_4 \vee x_5) \wedge (x_2 \vee x_4 \vee x_5)$.

Pretpostavimo da je početno $MAX\_TRIES = 2$, $MAX\_PERIODS = 5$. *Zbog lakše vizualizacije, interpretaciju ćemo reprezentirati kao string* 0 *i* 1 *takav da bit na poziciji i u stringu predstavlja trenutnu istinitosnu vrijednost varijable* x_i.

+ *Neka je* $I = 10101$ *slučajno generirana interpretacija.*

  - *Neka je* $\pi = (15423)$ *slučajno generirana interpretacija u prvom periodu.*
  - $G := F$.
    * *Vršimo redukciju formule varijablom* x_1 *i dobivamo* $G = (x_2 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg x_5) \wedge (\neg x_2 \vee x_4 \vee x_5) \wedge (x_2 \vee x_4 \vee x_5)$.
    * *Vršimo redukciju formule varijablom* x_5 *i dobivamo* $G = (x_2 \vee \neg x_3) \wedge (x_2 \vee x_3)$.
    * *Varijabla* x_4 *je već eliminirana iz formule pa nastavljamo s postupkom eliminacije.*
    * *Vršimo redukciju formule varijablom* x_2 *i dobivamo* $G = (\neg x_3) \wedge (x_3)$.
      ** *U ovom trenutku formula G sadrži dvije jedinične klauzule* (x_3) *i* $(\neg x_3)$*, stoga ulazimo u while petlju algoritma. Na slučajan način biramo jediničnu klauzulu čiju varijablu ćemo eliminirati u sljedećem koraku, pretpostavimo da je ta klauzula* (x_3)*. Pošto postoji negacija varijable* x_3 *u formuli G vršimo eliminaciju varijable iz formule G te dobivamo* $G = \{\emptyset\}$.

- *Pošto je* $G = \emptyset$ *i nismo promijenili vrijednost niti jednoj varijabli, izaberemo varijablu na slučajan način i promijenimo joj vrijednost te ulazimo u novi period algoritma. Pretpostavimo da smo promijenili vrijednost varijable* x_4 *odnosno* $I(x_4) = 1$*, stoga je sada* $I = 10111$.
- *Neka je* $\pi = (5213)$ *novo generirana permutacija u drugom periodu.*

* *Vršimo redukciju formule varijablom* x_5 *i dobivamo* $G = (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee x_3)$.

* *Vršimo redukciju formule varijablom* x_2 *i dobivamo* $G = (\neg x_1 \vee \neg x_3) \wedge (x_3)$.

** *U ovom trenutku formula G sadrži jediničnu klauzulu* (x_3) *stoga ulazimo u while petlju algoritma. Pošto postoji negacija varijable* x_3 *u formuli G vršimo eliminaciju varijable iz formule G te dobivamo* $G = (\neg x_1)$.

** *Pošto formula G i u ovom koraku sadrži jediničnu klauzulu ostajemo u while petlji i vršimo eliminaciju varijable* x_1*. Pošto se u formuli G ne javlja* x_1 *i* $\neg x_1$ *i* $I(\neg x_1) = 0$*, definiramo* $I(\neg x_1) = 1$*, odnosno* $I(x_1) = 0$ *te vršimo eliminaciju varijable* x_1 *iz formule G. U ovom koraku dobivamo* $G = ()$.

- *Pošto vrijedi* $I(F) = 1$ *vraćamo interpretaciju* $I = 00111$ *kao rješenje.*

U radu smo dokazali korektnost dva potpuna algoritma za rješavanje problema ispunjivosti logičke formule, Davis-Putnamovog algoritma i Davis-Putnam-Logemann-Lovelandovog algoritma. Davis-Putnamov algoritam je bio jedan od prvih algoritama za ispitivanje ispunjivosti logičke formule. Algoritam koristi rezoluciju kao primarnu metodu pojednostavljivanja formule. Zbog slabih računala i kombinatorne eksplozije do koje dolazi učestalom primjenom rezolucije, rezultati dobiveni primjenom ovoga algoritma su bili dosta skromni.

U današnje vrijeme koristeći sofisticiranija računala i poboljšane metode izračunavanja (DPLL algoritam i algoritmi bazirani na njemu) rješavanje problema ispunjivosti logičke formule se sve više koristi u raznim disciplinama kao što su verifikacija hardvera, verifikacija softvera, provjera modela, automatizirano dokazivanje teorema i brojni drugi. Razni teški problemi se mogu zakodirati u obliku formula propozicionalne logike i zatim riješiti nekim od algoritama za rješavanje problema ispunjivosti logičke formule.

Unatoč činjenici da su razne tehnike dovele do znatnog smanjenja prostora pretraživanja kod potpunih algoritama za rješavanje problema ispunjivosti logičke formule, oni u najgorem slučaju imaju eksponencijalnu vremensku složenost. Za neke praktične probleme u kojima imamo formule i sa nekoliko desetaka tisuća varijabli i sa nekoliko stotina tisuća klauzula potpuni algoritmi mogu još uvijek biti nepraktični.

Zbog toga se paralelno razvija grana heurističkih (stohastičkih) algoritama za rješavanje problema ispunjivosti logičke formule. Heuristički algoritmi su brzi i često dosta jednostavni algoritmi koji su dosta teški za teorijsku analizu. Ti algoritmi pokazuju izvrsne rezultate u praksi. U radu smo se osvrnuli na UnitWalk algoritam, nepotpuni heuristički algoritam koji kombinira slučajne šetnje sa eliminacijom jediničnih klauzula. Iz toga primjera vidimo da se neke metode i tehnike korištene kod potpunih algoritama primjenjuju i na stohastičkim algoritmima, međutim i u potpune algoritme se sve više uvode elementi slučajnosti pri izabiru grana pretraživanja s ciljem bržeg pronalaska rješenja. Navedeni UnitWalk algoritam za neke formule nađe rješenje i do 200 puta brže od algoritma miniSAT koji je baziran na DPLL algoritmu i koji se smatra jednim od vodećih potpunih algoritama današnjice.

Heuristički algoritmi se djele na:

- Prirodom inspirirane algoritme, kao što su particle swarm algoritam, ant colony optimization algoritam
- Evolucijske algoritme, kao što su genetski algoritmi
- Slučajne šetnje, kao što su WalkSAT, UnitWalk, Novelty, RNovelty i brojni drugi
- Pohlepne algoritme, kao što su GSAT, GSAT$^+$ algoritmi

Sve navedene vrste algoritama se upotrebljavaju i za rješavanje problema ispunjivosti logičke formule.

Problem kod heurističkih algoritama je što oni nemaju jednake performanse na svim formulama. Također, oni ne mogu sa sto postotnom sigurnošću utvrditi je li formula neispunjiva odnosno antitautologija. Ono što možemo zaključiti takvim algoritmima je da je formula sa visokom vjerojatnošću neispunjiva ako heuristički algoritmi ne pronađu rješenje u dovoljnom broju uzastopnih iteracija.

S porastom broja logičkih i fizičkih jezgara u računalu i pojavom jakih grafičkih procesnih elemenata istražuju se i moguće metode paralelizacije potpunih algoritama što bi dodatno ubrzalo njihovo izvršavanje. Razni heuristički algoritmi se izvršavaju konkurentno da bi se mogla brzo riješiti veća i raznolikija klasa formula.

Nažalost, s povećanjem dimenzije problema i takvi paralelni algoritmi brzo postaju neučinkoviti.

Sve se više istažuju novi modeli izračunavanja kao što su kvantna i DNA računala. Vjeruje se da bi razina paralelnosti koju donose ovi modeli mogla znatno ubrzati rješavanje problema ispunjivosti logičke formule. Postoje indicije da bi se na takvim računalima neki problemi za koje se smatra da spadaju u klasu složenosti NP mogli riješiti u polinomnom vremenu. Dok su kvantna računala još uvijek samo teoretski model, prva DNA računala su već konstruirana.

Automatizirano DNA računalo nazvano EDNAC koristeći DNA biblioteku i komplementarne sonde rješava problem ispunjivosti 3-KNF formula sa n varijabli. Računalo je pokrenuto na 3-KNF formuli sa 10 varijabli. Računanje je trajalo 28 sati. EDNAC je točno izračunao vrijednost devet od deset varijabli dok je vrijednost desete varijable ostala dvosmislena.

Iz svega ovoga možemo zaključiti da je problem ispunjivosti logičke formule jedan od najistraživanijih problema današnjice. Težina problema je navela stručnjake iz polja matematike, računarstva, fizike, biologije i kemije na suradnju u njegovom rješavanju. Dosadašnji, ali i budući rezultati u istraživanju ovoga interesantnog problema će sigurno dovesti do značajnih poboljšanja u mnogim znanstvenim poljima današnjice.

Bibliografija

[1] A. Biere, M. Heule, H. van Maaren, T. Walsh, *Handbook of Satisfiability*, IOS Press, 2009.

[2] H. Böning, T. Lettmann, *Propositional Logic: Deduction and Algorithms*, Cambridge University Press, 1999.

[3] E. A. Hirsc, A. Kojevnikov, *UnitWalk: A new SAT solver that uses local search guided by unit clause elimination*
http://www.cs.ubc.ca/labs/beta/Courses/CPSC532D-03/Resources/HirKoj02b.pdf

[4] I. Ivanuš, *Rezolucija u logici sudova*, Diplomski rad, PMF-MO, Zagreb, 2008.

[5] V. W. Marek, *Introduction to Mathematics of Satistiability*, CRC Press, 2009.

[6] R. Paturi, P. Pudlák, F. Zane, *Satisfiability coding lemma*, Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, FOCS'97. str. 566-574.

[7] R. Paturi, P. Pudlák, M. E. Saks, F. Zane, *An improved exponential-time algorithm for k-SAT*, Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, FOCS'98. str. 628-637.

[8] M. Sipser, *Introduction to the Theory of Computation, Second Edition*, Thomson Course Technology, 2006.

[9] M. Vuković, *Matematička Logika*, Element, Zagreb, 2009.

[10] *DPvis alat za vizualizaciju DPLL algoritma*
http://www-sr.informatik.uni-tuebingen.de/ sinz/DPvis/

Sažetak

U ovom diplomskom radu proučavamo dva potpuna algoritma za rješavanje problema ispunjivosti logičke formule, Davis-Putnamov algoritam i Davis-Putnam-Logemann-Lovelandov algoritam.

U uvodnom dijelu uvodimo osnovne pojmove logike sudova potrebne za razumijevanje rezultata navedenih u radu. Većina algoritama za rješavanje problema ispunjivosti logičke formule na ulaz prima formule u konjunktivnoj normalnoj formi. Znamo da je intuitivni algoritam za prebacivanje proizvoljne formule u konjunktivnu normalnu formu, koji koristi samo de Morganova pravila, eksponencijalne vremenske složenosti. Stoga smo opisali polinomni algoritam za dobivanje formule koja nije logički ekvivalentna početnoj, ali je ispunjiva ako i samo ako je ispunjiva i početna formula.

Detaljno smo opisali pravilo izvoda rezoluciju, naveli neke važnije tvrdnje i prikazali njezinu praktičnu primjenu na sustavu za automatsko odgovaranje na upite.

Glavni dio rada čine dokazi korektnosti Davis-Putnamovog (DP) i Davis-Putnam-Logemann-Lovelandovog (DPLL) algoritma. Naveli smo pseudokodove algoritama te na primjerima demonstrirali njihov rad. Promatrali smo koja poboljšanja donosi DPLL algoritam u odnosu na DP algoritam te koja daljnja poboljšanja DPLL algoritma su dovela do toga da je on baza svih modernih algoritama za rješavanje problema ispunjivosti logičke formule. Rad DPLL algoritma smo vizualizirali preko aplikacije DPvis na jednom primjeru. Pokazali smo kako graf međuovisnosti olakšava razumijevanje strukture formule.

Unatoč navedenim poboljšanjima, potpuni algoritmi imaju u najgorem slučaju eksponencijalnu vremensku složenost. Zbog toga se paralelno razvijaju i heuristički algoritmi. Glavna odlika tih algoritama je da relativno brzo dolaze do dovoljno dobrih rješenja. Ako zadana formula nije ispunjiva tada to ne možemo utvrditi heurističkim algoritmima. U radu smo opisali jedan heuristički algoritam za rješavanja problema ispunjivosti logičke formule, UnitWalk algoritam. Dokazali smo da je UnitWalk algoritam vjerojatnosno aproksimativno potpun, što znači da će za svaku ispunjivu formulu s vjerojatnošću 1 pronaći interpretaciju za koju je ona istinita. Na dva primjera smo prikazali rad UnitWalk algoritma.

Summary

In this graduate thesis we study two complete algorithms for solving Boolean satisfiability problem. This algorithms are the Davis-Putnam and the Davis-Putnam-Logemann-Loveland algorithm.

In the opening part of the thesis we introduce basic concepts of propositional logic necessary to fully understand presented results. Since algorithms for solving satisfiability problem generaly work with formulas in conjunctive normal form, we describe polynomial algorithm for transforming formula of propositional logic to conjunctive normal form.

We describe the resolution rule, state some important facts and show it's application on automated query answering.

Main part of the thesis consists of correctness proofs of the Davis-Putnam (DP) and the Davis-Putnam-Logemann-Loveland (DPLL) algorithm. We demonstrate these algorithms on examples and give their pseudocode. We observe improvements of DP and DPLL algorithms. We visualized the process of satisfiability solving of DPLL algorithm with DPvis application. We show on example that interaction graphs are useful for understanding structure of a formula.

Despite given improvements, complete algorithms have exponential worst case upper bounds. That is the reason for huge study of heuristic algorithms. Main advantage of heuristic algorithms is that thay are relatively quick in finding good enough solutions. Heuristic algorithms however, can't determine that a given formula is unsatisfiable. We describe one heuristic algorithm for solving satisfiability problem, the UnitWalk algorithm. We prove that the UnitWalk algorithm is probabilistically approximately complete, wich means that for any satisfiable formula it finds a solution with probability 1. We give two examples of formulas wich we solve using UnitWalk algorithm.

Životopis

Rođen sam 15. ožujka 1988. godine u Zagrebu. U Delnicama 1994. godine upisujem Osnovnu školu Ivana Gorana Kovačića. Godinu kasnije upisujem Osnovnu glazbenu školu Ive Tijardovića u Delnicama. Od hobija se intenzivno bavim šahom. Srednju školu Delnice, Prirodoslovno-matematičku gimnaziju upisujem 2002. godine, a godinu kasnije srednju glazbenu školu Ivana Matetića Ronjgova u Rijeci. Godine 2006. sam maturirao s odličnim uspjehom u gimnaziji i oslobođen sam mature. Iste godine upisujem Prirodoslovno-matematički fakultet, Matematički odsjek Sveučilišta u Zagrebu. Godine 2007. sam maturirao s odličnim uspjehom u srednjoj glazbenoj školi gdje sam također oslobođen mature. Godine 2009. nakon završenog preddiplomskog studija matematike upisujem Diplomski svučilišni studij Računarstva i matematike na istom odsjeku. Godine 2011. dobio sam Rektorovu nagradu za rad *Nova paralelna heuristika za rješavanje problema ispunjivosti logičke formule* koji sam radio zajedno s kolegom Tihomirom Lolićem.