Automatsko rezonovanje – beleške sa predavanja Iskazno rezonovanje

Milan Banković (po slajdovima Filipa Marića)

*Matematički fakultet,
Univerzitet u Beogradu

Prolećni semestar 2023/24.

Pregled

Uvod

Šta je iskazna logika?

- Iskazna logika proučava iskaze i odnose između njihovih logičkih (istinitonosnih) vrednosti
- Iskazi predstavljaju tvrđenja koja mogu biti tačna ili netačna
  - polazimo od elementarnih iskaza koje predstavljamo atomima (iskaznim slovima)
  - unutrašnju strukturu atoma ne razmatramo, već jedino njihovu istinitonosnu vrednost
  - složeni iskazi predstavljaju se iskaznim formulama koje se formiraju upotrebom logičkih veznika
  - istinitonosna vrednost složenih iskaza zavisi od pretpostavljenih vrednosti atoma koji se u njima pojavljuju
- Iskazna logika predstavlja osnov za formiranje bogatijih logika (logika prvog reda, logika višeg reda, i sl.)
  - otuda je proučavanje iskazne logike značajno sa stanovišta budućeg proučavanja bogatijih logika
- Iskazna logika je i sama za sebe veoma primenjiva u praksi

Pregled

Sintaksa iskaznih formula

Definicija

Neka je dat (najviše prebrojiv) skup atoma (iskaznih slova) P. Skup iskaznih formula nad P (u oznaci $\mathcal{F}_P$) je najmanji skup reči nad azbukom $P \cup \{\top, \bot, \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, (,)\}$ koji zadovoljava:

- *iskazna slova i logičke konstante su iskazne formule*
- *Ako je A iskazna formula, onda je i $\neg A$ iskazna formula*
- *Ako su A i B iskazne formule, onda su i $A \wedge B$, $A \vee B$, $A \Rightarrow B$ i $A \Leftrightarrow B$ iskazne formule*
- *Ako je A iskazna formula, tada je i (A) iskazna formula*

Prioritet operatora

Prioriteti i zagrade

- Prioritet operatora (opadajuće): $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$
- Asocijativnost operatora: $\wedge$ i $\vee$ su levo, a $\Rightarrow$ i $\Leftrightarrow$ desno asocijativni
- Prioritet i asocijativnost se može promeniti zagradama

Primeri iskaznih formula

Primer

$$p \wedge q \Rightarrow (r \Leftrightarrow \neg p)$$

$$x_1 \wedge (x_2 \vee ((x_1 \Rightarrow \neg x_2) \Leftrightarrow \neg x_3))$$

Napomena

U gornjem primeru, zagrade oko $x_1 \Rightarrow \neg x_2$ nisu neophodne

Literali

Literali

Često se, kao elementarne, razmatraju veoma jednostavne formule: atomi i njihove negacije.

Definicija

Literal je ili atom (pozitivan literal) ili negacija atoma (negativan literal).

Definicija

Suprotan literal atoma p je njegova negacija $\neg p$, dok je suprotan literal negaciji atoma $\neg p$ sam atom p. Suprotni literal literala l označavaćemo sa $\overline{l}$.

Reprezentacija u programskim jezicima

Reprezentacija formula u programima

- Formule date u gore opisanoj konkretnoj sintaksi se programski obrađuju tako što se parsiraju i prevode u apstraktno sintaksno stablo (AST) u programu
- Nakon toga se sve potrebne operacije vrše nad tom apstraktnom reprezentacijom
- Različiti programski jezici ovu apstraktnu reprezentaciju implementiraju na različite načine
- Ukoliko je izlaz programa ponovo neka formula, tada se apstraktno stablo te formule jednostavno ponovo prevodi u konkretnu sintaksu i štampa na izlazu

Funkcionalni jezici

Primer

Funkcionalni jezici uglavnom imaju mogućnost direktnog definisanja algebarskih (najčešće rekurzivnih) tipova podataka. Npr. u jeziku Haskell:

```
data Formula =
  TRUE |
  FALSE |
  Var Int |
  Not Formula |
  And Formula Formula |
  Or Formula Formula |
  Imp Formula Formula |
  Iff Formula Formula
  deriving (Show, Read, Eq, Ord)
```

C

Primer

```
enum formula_type {TRUE, FALSE, VAR, NOT, AND, OR, IMP, IFF};
typedef struct _formula {
   enum formula_type type;
   unsigned var_num;
   struct _formula* op1, op2;
} formula;
```

U svakom čvoru se ostavlja mogućnost smeštanja i relevantnih i nerelevantih podataka (npr. čvor `NOT` *ne koristi* `var_num` *niti* `op2`*). Moguće su i ,,štedljivije" reprezentacije koje ne čuvaju u svakom čvoru sva moguća polja (koriste se obično unije).*

C++

Primer

Svaki veznik se predstavlja zasebnom klasom, pri čemu sve klase nasleđuju apstraktnu baznu klasu `Formula`.

```
class Formula {...};

class False : public Formula {...};
class True : public Formula {...};

class Atom : public Formula {
...
private:
  unsigned var_num;
};
```

C++

Primer

```
class UnaryConnective : public Formula {
...
private:
  Formula *_op1;
};

class Not : public UnaryConnective {...};

class BinaryConnective : public Formula {
...
private:
  Formula *_op1, *_op2;
};

class And : public BinaryConnective {...};
class Or : public BinaryConnective {...};
class Imp : public BinaryConnective {...};
class Iff : public BinaryConnective {...};
```

Složenost formule

Definicija

Složenost formule $c(F)$ *definišemo rekurzivno, na sledeći način:*

- $c(\top) = c(\bot) = 0$
- *ako je* p *atom, tada je* $c(p) = 0$
- $c((F)) = c(F)$
- $c(\neg F) = 1 + c(F)$
- $c(F \wedge G) = c(F \vee G) = c(F \Rightarrow G) = c(F \Leftrightarrow G) = c(F) + c(G) + 1$

Napomena

- Intuitivno, složenost formule je jednaka broju veznika koje ona sadrži
- Složenost formule odražava memorijsku veličinu formule kao podatka u programu
- Mnoga svojstva formula se dokazuju indukcijom po složenosti formule

Valuacije

Vrednosti atoma

Formalno, istinitosne vrednosti se atomima pridružuju pomoću valuacija.

Definicija

Valuacija $v : P \to \{0, 1\}$ *je funkcija koja skup svih atoma preslikava u dvočlan skup* $\{0, 1\}$ *(ili* $\{$*tačno, netačno*$\}$, $\{T, F\}$, ...*). Vrednost valuacije atoma* p *označavamo sa* $v(p)$*.*

Napomena

Skup $\{\top, \bot\}$*, se obično izbegava kao kodomen valuacija kako bi se napravila jasna razlika između sintaksnih simbola konstanti i semantičke vrednosti formule. Atom je tačan u datoj valuaciji akko se preslikava u tačno (tj. u* 1*).*

Primer

$$p \mapsto 1, q \mapsto 0, r \mapsto 1, \ldots$$

Atomi p *i* r *su tačani u ovoj valuaciji,* q *je netačan, ...*

Valuacije

Valuacije kao skupovi tačnih atoma

Alternativno, valuaciju možemo posmatrati kao skup atoma koje smatramo tačnim (dok su ostali atomi netačni).

Definicija

Valuacija je skup atoma. Atom je tačan akko pripada valuaciji.

Primer

$$\{p, r\}$$

Atomi p i r su tačani u ovoj valuaciji, q je netačan, . . .

Valuacije

Parcijalne valuacije

Ponekad je zgodno pridružiti istinitonosne vrednosti samo nekim atomima, dok za ostale kažemo da su *nedefinisani* u toj valuaciji. Takve valuacije nazivamo *parcijalne valuacije*. Najčešće se zadaju kao skupovi literala koji su tačni u toj valuaciji.

Definicija

Parcijalna valuacija je skup literala, koji ne sadrži dva suprotna literala. Literal je tačan akko pripada valuaciji, netačan ako njemu suprotan literal pripada valuaciji, a nedefinisan inače.

Primer

$$\{p, \neg q, r\}$$

Atomi p i r su tačni, q je netačan. Dalje, npr. literal $\neg p$ je netačan, dok su literali s i $\neg s$ nedefinisani.

Vrednost formule (interpretacija)

Vrednost formule

Svaka (potpuna) valuacija v indukuje funkciju $I_v : \mathcal{F}_P \to \{0, 1\}$ na skupu svih iskaznih formula nad P (u oznaci $\mathcal{F}_P$) koju nazivamo *interpretacija*. Ova funkcija svakoj formuli pridružuje istinitonosnu vrednost, a definišemo je rekurzivno, kao u sledećoj definiciji.

Definicija

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

Zadovoljenje

Definicija

Činjenicu da je formula F tačna u valuaciji v, tj. da je $I_v(F) = 1$ označavaćemo sa $v \models F$. Kažemo još i da valuacija v zadovoljava formulu F, ili da je valuacija v model formule F.

Istinitosna vrednost u funkcionalnom jeziku

Primer

Naredni kod implementira funkciju $I_v(F)$ (kroz funkciju `eval F v`*).*

```
eval :: Formula -> (Int -> Bool) ->  Bool
eval TRUE _ = True
eval FALSE _ = False
eval (Var i) v = v i
eval (Not f) v = not (eval f v)
eval (And f g) v = (eval f v) && (eval g v)
eval (Or f g) v = (eval f v) || (eval g v)
eval (Imp f g) v = not (eval f v) || (eval g v)
eval (Iff f g) v = (eval f v) == (eval g v)
```

Istinitosna vrednost u C-u

Primer

```
int eval(formula* F, int (*v) (unsigned)) {
   switch(F->type) {
       case TRUE:  return 1;
       case FALSE: return 0;
       case VAR:  return (*v)(F->var_num);
       case NOT:   return !eval(F->op1, v);
       case AND:   return eval(F->op1, v) && eval(F->op2, v);
       case OR:    return eval(F->op1, v) || eval(F->op2, v);
       case IMP:   return !eval(F->op1, v) || eval(F->op2, v);
       case IFF:   return eval(F->op1, v) == eval(F->op2, v);
   }
}
```

Istinitosna vrednost u C++-u (1)

Primer

```
class Valuation {
public:
   bool operator () (unsigned p) const {
      std::map<unsigned, bool>::const_iterator it = _m.find(p);
      if (it == _m.end())
        throw "Valuation lookup error";
       return it->second;
   }

   void setValue(unsigned p, bool v) {
      _m[p] = v;
   }
private:
  std::map<unsigned, bool> _m;
};
```

Istinitosna vrednost u C++-u (2)

Primer

```
class Formula {
public: ...
   virtual bool eval(const Valuation & val) = 0;
};

bool  True::eval(const Valuation & v) { return true; }
bool False::eval(const Valuation & v) { return false; }
bool   Var::eval(const Valuation & v) { return v(_var_num); }
bool   Not::eval(const Valuation & v) { return !_op1->eval(v); }

bool   And::eval(const Valuation & v) {
   return _op1->eval(v) && _op2->eval(v);
}
bool    Or::eval(const Valuation & v) {
   return _op1->eval(v) || _op2->eval(v);
}
bool   Imp::eval(const Valuation & v) {
   return !_op1->eval(v) || _op2->eval(v);
}
bool   Iff::eval(const Valuation & v) {
   return _op1->eval(v) == _op2->eval(v);
}
```

Zadovoljivost, nezadovoljivost, tautologičnost, porecivost

Definicija

- *Formula je zadovoljiva ako ima bar jedan model.*
- *Formula je nezadovoljiva (kontradikcija) ako nema nijedan model.*
- *Formula je tautologija (logički valjana) ako joj je svaka valuacija model (u oznaci $\models F$)*
- *Formula je poreciva ako postoji valuacija koja joj nije model.*

Zadovoljivost, nezadovoljivost, tautologičnost, porecivost

Stav

- *Svaka tautologija je zadovoljiva.*
- *Svaka nezadovoljiva formula je poreciva.*
- *Formula je poreciva akko nije tautologija.*
- *Formula je nezadovoljiva akko nije zadovoljiva.*
- *Formula F je tautologija akko je $\neg F$ nezadovoljiva.*
- *Formula F je zadovoljiva akko je $\neg F$ poreciva.*

Model skupa formula

Model skupa formula

U nekim slučajevima, potrebno je da je više formula istovremeno zadovoljeno.

Definicija

Valuacija v zadovoljava skup formula tj. predstavlja model skupa formula Γ (što označavamo sa $v \models \Gamma$) akko je v model svake formule iz Γ.
Skup formula je zadovoljiv akko ima model.

Napomena

Obratiti pažnju da se traži da postoji jedna valuacija koja istovremeno zadovoljava sve formule.

Logičke posledice, ekvivalentne formule, ekvizadovoljive formule

Definicija

Formula F je logička posledica skupa formula Γ (što označavamo sa $\Gamma \models F$) akko je svaki model za skup Γ istovremeno i model za formulu F.
Formule F_1 i F_2 su logički ekvivalentne (što označavamo sa $F_1 \equiv F_2$) ako je svaki model formule F_1 ujedno model formule F_2 i obratno (tj. ako $A \models B$ i $B \models A$).
Formule F_1 i F_2 su ekvizadovoljive (slabo ekvivalentne) (što označavamo sa $F_1 \equiv_s F_2$) akko je F_1 zadovoljiva ako i samo ako je F_2 zadovoljiva.

Primeri

Primer

- *Formula p je logička posledica formule $p \wedge q$. Zaista, za svaku valuaciju v za koju važi $I_v(p \wedge q)$, važi i $I_v(p)$.*
- *Formula $p \wedge q$ nije logička posledica formule $p \vee q$, iako valuacija $v = \{p, q\}$ zadovoljava obe. Npr. valuacija $v = \{p, \neg q\}$ zadovoljava $p \vee q$, ali ne i $p \wedge q$.*
- *Formule $p \wedge q$ i $q \wedge p$ su logički ekvivalentne.*
- *Formule $p \wedge q$ i $r \wedge (r \Leftrightarrow p \wedge q)$ su ekvizadovoljive (obe su zadovoljive), ali nisu ekvivalentne. Npr. valuacija $v = \{p, q, \neg r\}$ zadovoljava prvu, ali ne i drugu.*

Stav

- $\bot \models F$ *i* $F \models \top$ *za svaku formulu* F
- $F_1, \ldots, F_n \models F$ *akko je* $F_1 \wedge \ldots \wedge F_n \Rightarrow F$ *tautologija.*
- $\Gamma, F \models F'$ *akko* $\Gamma \models F \Rightarrow F'$.
- $F \equiv F'$ *akko je* $F \Leftrightarrow F'$ *tautologija.*
- *Ako je* $F_1 \equiv F_1'$ *i* $F_2 \equiv F_2'$ *tada je i*
  - $\neg F_1 \equiv \neg F_1'$,
  - $F_1 \wedge F_2 \equiv F_1' \wedge F_2'$,
  - $F_1 \vee F_2 \equiv F_1' \vee F_2'$,
  - $F_1 \Rightarrow F_2 \equiv F_1' \Rightarrow F_2'$, *i*
  - $F_1 \Leftrightarrow F_2 \equiv F_1' \Leftrightarrow F_2'$.

Logički zakoni

Stav

- $\neg\top \equiv \bot$, $\neg\bot \equiv \top$, $A \wedge \top \equiv A$, $A \wedge \bot \equiv \bot$, $A \vee \top \equiv \top$, $A \vee \bot \equiv A$, $\bot \Rightarrow A \equiv \top$, $\top \Rightarrow A \equiv A$, $A \Rightarrow \bot \equiv \neg A$, $A \Rightarrow \top \equiv \top$, $\top \Leftrightarrow A \equiv A$, $\bot \Leftrightarrow A \equiv \neg A$ *(logičke konstante)*
- $A \wedge \neg A \equiv \bot$, $A \vee \neg A \equiv \top$ *(komplementarnost)*
- $A \wedge A \equiv A$, $A \vee A \equiv A$ *(idempotencija)*
- $\neg\neg A \equiv A$ *(dvojna negacija)*
- $A \wedge (B \wedge C) \equiv (A \wedge B) \wedge C$, $A \vee (B \vee C) \equiv (A \vee B) \vee C$ *(asocijativnost)*
- $A \wedge B \equiv B \wedge A$, $A \vee B \equiv B \vee A$ *(komutativnost)*
- $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$, $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$ *(distibutivnost)*
- $\neg(A \wedge B) \equiv \neg A \vee \neg B$, $\neg(A \vee B) \equiv \neg A \wedge \neg B$ *(de-Morganovi zakoni)*
- $A \wedge (A \vee B) \equiv A$, $A \vee (A \wedge B) \equiv A$ *(zakoni apsorpcije)*
- $A \Rightarrow B \equiv \neg A \vee B$
- $A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (B \Rightarrow A) \equiv (\neg A \vee B) \wedge (\neg B \vee A)$
- $A \Leftrightarrow B \equiv (A \wedge B) \vee (\neg A \wedge \neg B)$

Pregled

Vrednost formule je određena vrednošću njenih atoma

Stav

- *Ako je skup atoma beskonačan (a najčešće uzimamo da je prebrojiv) postoji beskonačno (čak neprebrojivo) mnogo različitih valuacija.*
- *Skup atoma koji se javljaju u formuli je konačan.*
- *Ukoliko se dve valuacije poklapaju na skupu atoma koji se javlja u nekoj formuli, istinitosna vrednost formule u obe valuacije je jednaka.*

Napomena

Iz gornjeg stava sledi da je, sa stanovišta jedne fiksirane formule, potrebno razmatrati konačno mnogo (parcijalnih) valuacija (ako je n broj različitih atoma u formuli, onda je dovoljno ispitati 2^n različitih valuacija). Otuda imamo sledeću važnu posledicu.

Posledica

Problem tautologičnosti (zadovoljivosti, . . .) u iskaznoj logici je odlučiv.

Procedura odlučivanja

Jedna jednostavna (ali prilično neefikasna) procedura odlučivanja za problem tautologičnosti (zadovoljivosti,. . .) je metod istinitosnih tablica.

Istinitosna tablica – primer

Primer

$(p$	$\vee$	$(q$	$\wedge$	$r))$	$\wedge$	$(\neg$	p	$\vee$	$\neg$	$r)$
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	1	0	1	0	1	0	1
0	0	1	0	0	0	1	0	1	1	0
0	1	1	1	1	1	1	0	1	0	1
1	1	0	0	0	1	0	1	1	1	0
1	1	0	0	1	0	0	1	0	0	1
1	1	1	0	0	1	0	1	1	1	0
1	1	1	1	1	0	0	1	0	0	1

Pregled

Zamena

Definicija

Formula $G[F \to F']$ je zamena formule F formulom F' u formuli G i konstruiše se tako što se sva pojavljivanja potformule F u okviru neke formule G zamenjuju sa F'.

Primer

Ako u formuli

$$(p \Rightarrow q) \wedge r \vee (p \Rightarrow q),$$

zamenjujemo $p \Rightarrow q$ sa $\neg p \vee q$, dobija se formula:

$$(\neg p \vee q) \wedge r \vee (\neg p \vee q).$$

Zamena u funkcionalnom jeziku

Primer

```
subst :: Formula -> Formula -> Formula -> Formula
subst g f f' | g == f = f'
subst TRUE _ _ = TRUE
subst FALSE _ _ = FALSE
subst (Var i) _ _ = Var i
subst (Not g') f f' = Not (subst g' f f')
subst (And g1 g2) f f' = And (subst g1 f f') (subst g2 f f')
subst (Or g1 g2) f f' = Or (subst g1 f f') (subst g2 f f')
subst (Imp g1 g2) f f' = Imp (subst g1 f f') (subst g2 f f')
subst (Iff g1 g2) f f' = Iff (subst g1 f f') (subst g2 f f')
```

Svojstva zamene

Stav

Ako je p atom, F i G formule, v valuacija, a v' valuacija koja se dobija postavljanjem vrednosti p na $I_v(G)$ u valuaciji v, onda je $I_v(F[p \to G]) = I_{v'}(F)$.

Teorema

Ako je p atom, F proizvoljna formula, a formula G je tautologija, onda je $G[p \to F]$ tautologija.

Teorema

Ako je $F \equiv F'$, onda je $G[F \to F'] \equiv G$ (Teorema o zameni).

Svojstva zamene — primeri

Primer

- $p \wedge q \Leftrightarrow q \wedge p$ *je tautologija, pa je i* $(r \vee s) \wedge (s \wedge r) \Leftrightarrow (s \wedge r) \wedge (r \vee s)$ *tautologija.*
- $p \wedge q \equiv q \wedge p$ *pa je i* $r \vee (p \wedge q) \equiv r \vee (q \wedge p)$.

Pregled

Normalne forme

Normalne forme

- Određenim transformacijama iskazne formule se mogu svesti na oblike pogodnije za određene zadatke
- Transformacije formula predstavljaju sintaksne operacije, ali se obično zahteva njihova semantička opravdanost:
  - najčešće se zahteva da dobijena formula bude logički ekvivalentna sa polaznom
  - u nekim situacijama dovoljna je i ekvizadovoljivost
- Semantičko opravdanje daje nam teorema o zameni

Eliminisanje konstanti

Zakoni za eliminaciju logičkih konstanti

Naredne logičke ekvivalencije mogu se upotrebiti za uprošćavanje formule eliminisanjem logičkih konstanti $\top$ i $\bot$ (preciznije, eliminišu se veznici koji se primenjuju na logičke konstante):

$$\neg\bot \equiv \top$$
$$\neg\top \equiv \bot$$
$$P \wedge \bot \equiv \bot$$
$$\bot \wedge P \equiv \bot$$
$$P \wedge \top \equiv P$$
$$\top \wedge P \equiv P$$
$$P \vee \bot \equiv P$$
$$\bot \vee P \equiv P$$
$$P \vee \top \equiv \top$$
$$\top \vee P \equiv \top$$

$$P \Rightarrow \bot \equiv \neg P$$
$$\bot \Rightarrow P \equiv \top$$
$$P \Rightarrow \top \equiv \top$$
$$\top \Rightarrow P \equiv P$$
$$P \Leftrightarrow \bot \equiv \neg P$$
$$\bot \Leftrightarrow P \equiv \neg P$$
$$P \Leftrightarrow \top \equiv P$$
$$\top \Leftrightarrow P \equiv P$$

Implementacija u funkcionalnom jeziku

Algoritam

```
simplify' :: Formula -> Formula
simplify' (Not FALSE) = TRUE
simplify' (Not TRUE) = FALSE
simplify' (And f TRUE) = f
simplify' (And TRUE f) = f
...
simplify' f = f

simplify :: Formula -> Formula
simplify FALSE = FALSE
simplify TRUE = TRUE
simplify (Var i) = Var i
simplify (Not f) = simplify' (Not (simplify f))
simplify (And f1 f2) = simplify' (And (simplify f1) (simplify f2))
simplify (Or f1 f2) = simplify' (Or (simplify f1) (simplify f2))
simplify (Imp f1 f2) = simplify' (Imp (simplify f1) (simplify f2))
simplify (Iff f1 f2) = simplify' (Iff (simplify f1) (simplify f2))
```

Eliminacija konstanti

Primer

Uprošćavanjem formule

$$(\top \Rightarrow (x \Leftrightarrow \bot)) \Rightarrow \neg(y \vee (\bot \wedge z))$$

dobija se

$$\neg x \Rightarrow \neg y$$

Napomena

Primenom gornjeg algoritma formula se svodi ili na $\top$ ili $\bot$, ili na formulu koja ne sadrži logičke konstante.

NNF

Definicija

NNF Formula je u negacionoj normalnoj formi (NNF) akko je sastavljena od literala korišćenjem isključivo veznika $\wedge$ i $\vee$ ili je logička konstanta ($\top$ ili $\bot$).

Primer

Formule $\top$, $\bot$, p, $p \wedge \neg q$ i $p \vee (q \wedge (\neg r) \vee s)$ su u NNF, dok $\neg\neg p$ i $p \wedge \neg(q \vee r)$ to nisu.

Transformacija u NNF formu

Svaka formula se može transformisati u ekvivalentnu NNF formu:

- eliminišemo ekvivalencije i implikacije
- spustimo negacije na nivo atoma

Uklanjanje implikacija i ekvivalencija

Zakoni za eliminaciju implikacija

$$\begin{aligned} P \Rightarrow Q &\equiv \neg P \vee Q \\ \neg(P \Rightarrow Q) &\equiv P \wedge \neg Q \end{aligned}$$

Zakoni za eliminaciju ekvivalencija

$$\begin{aligned} P \Leftrightarrow Q &\equiv (P \wedge Q) \vee (\neg P \wedge \neg Q) \\ \neg(P \Leftrightarrow Q) &\equiv (P \wedge \neg Q) \vee (\neg P \wedge Q) \end{aligned}$$

ili

$$\begin{aligned} P \Leftrightarrow Q &\equiv (P \vee \neg Q) \wedge (\neg P \vee Q) \\ \neg(P \Leftrightarrow Q) &\equiv (P \vee Q) \wedge (\neg P \vee \neg Q) \end{aligned}$$

Spuštanje negacije

Zakoni za spuštanje negacije

$$\begin{aligned} \neg(P \wedge Q) &\equiv \neg P \vee \neg Q \\ \neg(P \vee Q) &\equiv \neg P \wedge \neg Q \\ \neg\neg P &\equiv P \end{aligned}$$

Implementacija u funkcionalnom programskom jeziku

Algoritam

```
nnf' :: Formula -> Formula
nnf' FALSE = FALSE
nnf' TRUE = TRUE
nnf' (Var i) = Var i
nnf' (And f g) = And (nnf' f) (nnf' g)
nnf' (Or f g) = Or (nnf' f) (nnf' g)
nnf' (Imp f g) = Or (nnf' (Not f)) (nnf' g)
nnf' (Iff f g) = Or (And (nnf' f) (nnf' g))
                    (And (nnf' (Not f)) (nnf' (Not g)))
nnf' (Not (Not f)) = nnf' f
nnf' (Not (And f g)) = Or (nnf' (Not f)) (nnf' (Not g))
nnf' (Not (Or f g)) = And (nnf' (Not f)) (nnf' (Not g))
nnf' (Not (Imp f g)) = And (nnf' f) (nnf' (Not g))
nnf' (Not (Iff f g)) = Or (And (nnf' f) (nnf' (Not g)))
                          (And (nnf' (Not f)) (nnf' g))
nnf' (Not f) = Not f

nnf :: Formula -> Formula
nnf f = nnf' (simplify f)
```

NNF

Primer

$$NNF[(p \Leftrightarrow q) \Leftrightarrow \neg(r \Rightarrow s)] \quad \equiv$$

$$NNF[p \Leftrightarrow q] \wedge NNF[\neg(r \Rightarrow s)] \vee$$
$$NNF[\neg(p \Leftrightarrow q)] \wedge NNF[\neg\neg(r \Rightarrow s)] \quad \equiv$$

$$(p \wedge q) \vee (\neg p \wedge \neg q) \wedge (r \wedge \neg s) \vee ((p \wedge \neg q) \vee (\neg p \wedge q)) \wedge (\neg r \vee s)$$

Složenost NNF transformacije

Kombinatorna eksplozija

- Zahvaljujući uvećanju formule prilikom eliminacije ekvivalencije, u najgorem slučaju, NNF formule sa n veznika može da sadrži više od 2^n veznika.
- Ukoliko je samo cilj da se negacija spusti do nivoa atoma, i ne insistira na potpunoj izgradnji NNF, tj. ako se dopusti zadržavanje ekvivalencija u formuli, moguće je izbeći eksponencijalno uvećanje. Negacija se može spustiti kroz ekvivalenciju na osnovu npr. $\neg(p \Leftrightarrow q) \equiv \neg p \Leftrightarrow q$.

DNF

Definicija

Formula je u disjunktivnoj normalnoj formi (DNF) ako je oblika $D_1 \vee \ldots \vee D_N$, *pri čemu je svaki disjunkt* D_i *oblika* $l_{i1} \wedge \ldots \wedge l_{im_i}$, *pri čemu su* l_{ij} *literali, tj. oblika je*

$$\bigvee_{i=1}^{N} \bigwedge_{j=1}^{m_i} l_{ij}$$

Napomena

Ako je formula u *DNF* ona je i u *NNF*, uz dodatno ograničenje da je ,,disjunkcija konjunkcija".

KNF

Definicija

Formula je u konjunktivnoj normalnoj formi (KNF)[1] *ako je oblika* $K_1 \wedge \ldots \wedge K_N$, *pri čemu je svaki konjunkt* K_i *klauza, tj. oblika* $l_{i1} \vee \ldots \vee l_{im_i}$, *pri čemu su* l_{ij} *literali, tj. oblika je*

$$\bigwedge_{i=1}^{N} \bigvee_{j=1}^{m_i} l_{ij}$$

Napomena

Ako je formula u *KNF* ona je i u *NNF*, uz dodatno ograničenje da je ,,konjunkcija disjunkcija".

[1] eng. CNF (Conjunctive Normal Form)

KNF i DNF date formule

Definicija

Za formulu F' kažemo da je DNF (KNF) formule F akko je F' u DNF (KNF) i važi $F \equiv F'$.

Stav

Svaka formula ima DNF (KNF).

Napomene

- Naglasimo da postoji više različitih DNF (KNF) za istu polaznu formulu.
- Specijalno, za formule $\top$ i $\bot$ smatraćemo da su one same u DNF (odnosno KNF).

Prevođenje pomoću zakona distribucije

Zakoni za transformaciju u DNF

NNF formula se može prevesti u DNF primenom logičkih ekvivalencija

$$\begin{aligned} P \wedge (Q \vee R) &\equiv (P \wedge Q) \vee (P \wedge R) \\ (Q \vee R) \wedge P &\equiv (Q \wedge P) \vee (R \wedge P) \end{aligned}$$

Zakoni za transformaciju u KNF

NNF formula se može prevesti u KNF primenom logičkih ekvivalencija

$$\begin{aligned} P \vee (Q \wedge R) &\equiv (P \vee Q) \wedge (P \vee R) \\ (Q \wedge R) \vee P &\equiv (Q \vee P) \wedge (R \vee P) \end{aligned}$$

Implementacija u funkcionalnom jeziku

Algoritam

```
distribdnf :: Formula -> Formula
distribdnf (And f (Or g1 g2)) = Or (distribdnf (And f g1))
                                   (distribdnf (And f g2))
distribdnf (And (Or g1 g2) f) = Or (distribdnf (And g1 f))
                                   (distribdnf (And g2 f))
distribdnf f = f

dnf' :: Formula -> Formula
dnf' (And f g) = distribdnf (And (dnf' f) (dnf' g))
dnf' (Or f g) = Or (dnf' f) (dnf' g)
dnf' f = f

dnf :: Formula -> Formula
dnf f = dnf' (nnf f)
```

Primer

Primer

$$
\begin{aligned}
& dnf\,[(p \vee (q \wedge r)) \wedge (\neg p \vee \neg r)] && \equiv \\
& distribdnf\,[dnf\,[p \vee (q \wedge r)] \wedge dnf\,[\neg p \vee \neg r]] && \equiv \\
& distribdnf\,[(dnf\,[p] \vee dnf\,[q \wedge r]) \wedge (dnf\,[\neg p] \vee dnf\,[\neg r])] && \equiv \\
& distribdnf\,[(p \vee distribdnf\,[q \wedge r]) \wedge (\neg p \vee \neg r)] && \equiv \\
& distribdnf\,[(p \vee (q \wedge r)) \wedge (\neg p \vee \neg r)] && \equiv \\
& (p \wedge \neg p) \vee ((q \wedge r) \wedge \neg p) \vee (p \wedge \neg r) \vee ((q \wedge r) \wedge \neg r)
\end{aligned}
$$

Primetimo da se dobijena DNF može dalje uprostiti (npr. $p \wedge \neg p$, kao i $q \wedge r \wedge \neg r$ su logički ekvivalentne sa $\bot$ i mogu se ukloniti).

Reprezentacija pomoću lista (skupova)

Struktura DNF (KNF) formule

- Primetimo da DNF (KNF) ima veoma jednostavnu strukturu: sastoji se iz disjunkcija (konjunkcija) koje se primenjuju na konjunkcije (disjunkcije) literala
- Otuda za predstavljanje DNF (KNF) formula nije potrebno apstraktno stablo, kao u slučaju formula proizvoljnog oblika
- Štaviše, apstraktno stablo, kako smo ga ranije definisali, podrazumeva da su konjunkcije i disjunkcije binarni veznici
  - s obzirom na asocijativnost konjunkcije i disjunkcije, ove veznike možemo smatrati i n-arnim veznicima
  - primenom distributivnih zakona dobijaju se, u suštini, n-arne konjunkcije i disjunkcije
  - otuda stablo u kome su ovi veznici predstavljeni kao binarni nije pogodna struktura za predstavljanje DNF (KNF) formula
- Jednostavna struktura DNF (KNF) formula omogućava znatno jednostavniju reprezentaciju ovakvih formula:
  - DNF (KNF) formula je lista konjunkcija (disjunkcija), a svaka konjunkcija (disjunkcija) je lista literala
  - otuda se DNF (KNF) formule mogu predstaviti jednostavno kao liste listi literala
  - npr. $(p \wedge \neg q) \vee (q \wedge \neg r \wedge s)$ se može predstaviti kao $[[p, \neg q], [q, \neg r, s]]$
- Štaviše, s obzirom na zakone idempotencije ($P \wedge P \equiv P$ i $P \vee P \equiv P$) i komutativnosti ($P \wedge Q \equiv Q \wedge P$ i $P \vee Q \equiv Q \vee P$), umesto liste listi može se koristiti skup skupova (npr. $\{\{p, \neg q\}, \{q, \neg r, s\}\}$)
- Ipak, liste se obično jednostavnije realizuju u većini programskih jezika, pa se češće koriste

Implementacija KNF (DNF) za reprezentaciju u obliku lista

Pomoćne funkcije

```
type Lit = Int

toLit :: Formula -> Lit
toLit (Var i) = i
toLit (Not (Var i)) = -i

distrib :: [[a]] -> [[a]] -> [[a]]
distrib [] l = []
distrib (h:t) l = let hl = map (\e -> h ++ e) l
                      in hl ++ distrib t l

trivial :: [Lit] -> Bool
trivial [] = False
trivial (h:t) | elem (-h) t = True
              | otherwise = trivial t

remDup :: Eq a => [a] -> [a]
remDup [] = []
remDup (h:t) | elem h t = remDup t
             | otherwise = h : (remDup t)
```

Implementacija KNF za reprezentaciju u obliku lista

Algoritam

```
type CNF = [[Lit]]

listCNF' :: Formula -> CNF
listCNF' (And f g) = (listCNF' f) ++ (listCNF' g)
listCNF' (Or f g) = distrib (listCNF' f) (listCNF' g)
listCNF' TRUE = []
listCNF' FALSE = [[]]
listCNF' l = [[toLit l]]

listCNF :: Formula -> CNF
listCNF f = map remDup (filter (not . trivial) (listCNF' (nnf f)))
```

Implementacija DNF za reprezentaciju u obliku lista (2)

Algoritam

```
type DNF = [[Lit]]

listDNF' :: Formula -> DNF
listDNF' (Or f g) = (listDNF' f) ++ (listDNF' g)
listDNF' (And f g) = distrib (listDNF' f) (listDNF' g)
listDNF' TRUE = [[]]
listDNF' FALSE = []
listDNF' l = [[toLit l]]

listDNF :: Formula -> DNF
listDNF f = map remDup (filter (not . trivial) (listDNF' (nnf f)))
```

Veza između DNF i istinitosnih tablica

Definicija

Ako je data formula F koja nije kontradikcija, i koja sadrži atome $p_1, \ldots, p_n$, njena kanonska DNF je

$$\bigvee_{v \models F} \bigwedge_{i=1}^{n} p_i^v,$$

pri čemu je

$$p_i^v = \begin{cases} p_i & ako\ v(p_i) = 1 \\ \neg p_i & inače \end{cases}$$

Veza između KNF i istinitosnih tablica

Definicija

Ako je data formula F koja nije tautologija, i koja sadrži atome p_1, ..., p_n, njena kanonska KNF je

$$\bigwedge_{v \not\models F} \bigvee_{i=1}^{n} p_i^{\hat{v}},$$

pri čemu je

$$p_i^{\hat{v}} = \begin{cases} \neg p_i & ako\ v(p_i) = 1 \\ p_i & inače \end{cases}$$

Kanonska DNF/KNF - primer

Primer

p	q	r	$(p \vee (q \wedge r)) \wedge (\neg p \vee \neg r)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Kanonska DNF:

$$(\neg p \wedge q \wedge r) \vee (p \wedge \neg q \wedge \neg r) \vee (p \wedge q \wedge \neg r)$$

Kanonska KNF:

$$(p \vee q \vee r) \wedge (p \vee q \vee \neg r) \wedge (p \vee \neg q \vee r) \wedge (\neg p \vee q \vee \neg r) \wedge (\neg p \vee \neg q \vee \neg r)$$

Minimalizacija DNF (KNF)

Minimalizacija DNF (KNF)

DNF (KNF) formule nije jednoznačno određena. Postoje tehnike koje određuju minimalni DNF (ili KNF):

- Algebarske transformacije
- Karnoove (Karnaugh) mape
- Kvin-MekKlaski (Quine–McCluskey) algoritam (prosti implikanti)

Problem pronalaženja ekvivalentnog DNF-a (KNF-a) sa minimalnim brojem disjunkta (konjunkta): NP-težak

Složenost DNF i KNF procedura

Koliko je skupa DNF (CNF) transformacija?

- S obzirom da se baziraju na NNF, jasno je da distributivne DNF i KNF mogu eksponencijalno da uvećaju formulu.
- Takođe, i pravila distributivnosti sama za sebe doprinose eksponencijalnom uvećavanju. Npr. prevođenje $(p_1 \vee q_1) \wedge \ldots \wedge (p_n \vee q_n)$ u KNF ima 2^n klauza.
- Slično, i kanonske DNF i KNF mogu da budu eksponencijalno velike u odnosu na polaznu formulu.
- Ovakvo uvećanje je neizbežno ako se insistira da dobijena formula bude ekvivalentna polaznoj.
- Navedene činjenice čine sve prethodno navedene tehnike neupotrebljive u većini praktičnih zadataka velike dimenzije.

Definiciona (Cajtinova) KNF

Da li moramo da insistiramo na ekvivalentnosti?

- U većini primena, nije neophodno da KNF bude logički ekvivalentna polaznoj — dovoljno je da bude ekvizadovoljiva.
- Ekvizadovoljiva KNF se može izgraditi tako da dobijena formula bude samo za konstantni faktor veća od polazne.
- Dualno, postoji definiciona DNF koja je ekvivaljana polaznoj.
- Ideja potiče od Cajtina (Tseitin) — za potformule uvode se novi atomi (iskazna slova).

Definiciona (Cajtinova) KNF

Primer

$$(p \vee (q \wedge r)) \wedge (\neg p \vee \neg r)$$

$$(p \vee s_1) \wedge (\neg p \vee \neg r) \wedge (s_1 \Leftrightarrow q \wedge r)$$

$$s_2 \wedge (\neg p \vee \neg r) \wedge (s_1 \Leftrightarrow q \wedge r) \wedge (s_2 \Leftrightarrow p \vee s_1)$$

$$s_2 \wedge s_3 \wedge (s_1 \Leftrightarrow q \wedge r) \wedge (s_2 \Leftrightarrow p \vee s_1) \wedge (s_3 \Leftrightarrow \neg p \vee \neg r)$$

$$s_4 \wedge (s_1 \Leftrightarrow q \wedge r) \wedge (s_2 \Leftrightarrow p \vee s_1) \wedge (s_3 \Leftrightarrow \neg p \vee \neg r) \wedge (s_4 \Leftrightarrow s_2 \wedge s_3)$$

Definiciona (Cajtinova) KNF

Primer

Definicione ekvivalencije se klasično prevode u KNF:

$$s_4 \wedge (s_1 \Leftrightarrow q \wedge r) \wedge (s_2 \Leftrightarrow p \vee s_1) \wedge (s_3 \Leftrightarrow \neg p \vee \neg r) \wedge (s_4 \Leftrightarrow s_2 \wedge s_3)$$

$$\begin{aligned}
&s_4 \\
&(\neg s_1 \vee q) \wedge (\neg s_1 \vee r) \wedge (\neg q \vee \neg r \vee s_1) \wedge \\
&(\neg s_2 \vee p \vee s_1) \wedge (\neg p \vee s_2) \wedge (\neg s_1 \vee s_2) \wedge \\
&(\neg s_3 \vee \neg p \vee \neg r) \wedge (p \vee s_3) \wedge (r \vee s_3) \wedge \\
&(\neg s_4 \vee s_2) \wedge (\neg s_4 \vee s_3) \wedge (\neg s_2 \vee \neg s_3 \vee s_4)
\end{aligned}$$

Ekvizadovoljivost definicione KNF

Teorema

Ako se s_i ne javlja u Q tada su $P[s_i \to Q]$ i $P \wedge (s_i \Leftrightarrow Q)$ ekvizadovoljive. Preciznije, svaki model za $P[s_i \to Q]$ se može proširiti do modela za $P \wedge (s_i \Leftrightarrow Q)$ dok je svaki model za $P \wedge (s_i \Leftrightarrow Q)$ ujedno i model za $P[s_i \to Q]$.

Dokaz

Neka $v \vDash P[s_i \to Q]$. Posmatrajmo valuaciju v' koja menja v samo postavljajući promenljivoj s_i vrednost na $I_v(Q)$. Važi $v' \vDash Q$ (na osnovu svojstava zamene), i važi i $v' \vDash s_i \Leftrightarrow Q$ jer je $I_{v'}(Q) = I_v(Q) = I_v(s_i)$ (pošto se s_i ne javlja u Q), te važi $v \vDash P \wedge (s_i \Leftrightarrow Q)$.
Obratno, ako $v \vDash P \wedge (s_i \Leftrightarrow Q)$, tada $v \vDash P$ i $v \vDash s_i \Leftrightarrow Q$ pa je $I_v(s_i) = I_v(Q)$. Zato $v \vDash P[s_i \to Q]$ (na osnovu svojstava zamene).

Pregled

SAT problem

Šta je SAT problem?

- Problem ispitivanja zadovoljivosti iskazne formule naziva se SAT problem (od engleske reči *satisfiability*).
- Centralni problem teorijskog računarstva.
- SAT je prvi problem za koji je dokazano da je NP kompletan (Cook-Levin, 1971.)
- Ogromne praktične primene.

Pristupi rešavanja SAT problema

Pristupi

- Naivni metodi (metod istinitosnih tablica)
- Problem se tehnički jednostavnije rešava ako je formula u nekom od specijalnih oblika (DNF, KNF, BDD)
- Za DNF se zadovoljivost trivijalno ispituje, ali je postupak prevođenja u DNF netrivijalan. Slično je i sa BDD
- Postoji efikasan postupak prevođenja formule u KNF koji čuva zadovoljivost (Cajtinova transformacija)
- Zadovoljivost formula u KNF se dalje ispituje nekim od tzv. klauzalnih algoritama:
  - DP procedura (iskazna rezolucija)
  - DPLL procedura
  - CDCL SAT rešavači (unapređena DPLL procedura)
  - Stohastički SAT rešavači (zasnovani na randomizaciji — ne garantuju uvek definitivan odgovor)
- Na žalost, za KNF formule SAT problem je i dalje NP kompletan

O većini navedenih pristupa biće reči u nastavku.

SAT rešavači

SAT rešavači

Implementacije procedura odlučivanja za SAT problem nazivaju se SAT rešavači

SAT rešavači kao dokazivači teorema

Tautologije i teoreme

- Semantički pojam tautologije (valjane formule) ima svoj deduktivni ekvivalent – pojam teoreme
- Formula je teorema ako se može dokazati u zadatom deduktivnom sistemu
  - deduktivni sistem dat je aksiomama i pravilima izvođenja
- Iako je priroda pojmova tautologije i teoreme različita, obično se deduktivni sistemi grade tako da ovi pojmovi odgovaraju jedan drugom:
  - deduktivni sistem je saglasan (engl. sound) ako je svaka teorema ujedno i valjana formula
  - deduktivni sistem je potpun (engl. complete) ako je svaka valjana formula ujedno i teorema
- Ako je deduktivni sistem saglasan i potpun, tada se semantičke procedure za ispitivanje valjanosti mogu koristiti kao procedure za ispitivanje dokazivosti (obično bez konstruisanja dokaza)

SAT rešavači kao dokazivači teorema

- Formula F je tautologija akko je $\neg F$ nezadovoljiva
- Otuda se SAT rešavači mogu koristiti i kao dokazivači teorema
  - da bismo dokazali da je F tautologija, njenu negaciju $\neg F$ svedemo na KNF i predamo SAT rešavaču
  - ako rešavač potvrdi da je $\neg F$ nezadovoljiva, tada je polazna formula F tautologija
  - ovakav način dokazivanja teorema se naziva dokazivanje pobijanjem (engl. refutation)

Svojstva procedura za ispitivanje (ne)zadovoljivosti

Saglasnost, potpunost, zaustavljanje

Imajući u vidu opisanu vezu između ispitivanja (ne)zadovoljivosti i dokazivanja teorema (pobijanjem), osobine SAT rešavača kao dokazivača teorema možemo formulisati na sledeći način:

Saglasnost – Ako algoritam prijavi nezadovoljivost, polazna formula je zaista nezadovoljiva

Potpunost – Ako je polazna formula nezadovoljiva, algoritam će prijaviti nezadovoljivost

Zaustavljanje – Za svaku ulaznu formulu, algoritam se zaustavlja nakon primene konačno mnogo koraka

Od SAT rešavača obično očekujemo da ima sve tri navedene osobine.

SAT rešavači kao procedure pretrage

SAT rešavači kao procedure pretrage

- U zavisnosti od algoritma na kome je zasnovan, SAT rešavač može na izlazu dati i zadovoljavajuću valuaciju u slučaju zadovoljive formule
  - takvi SAT rešavači se mogu koristiti i kao procedure za pretragu, a ne samo kao procedure odlučivanja
- Ovo omogućava primenu SAT rešavača u oblastima koje izlaze iz okvira dokazivanja teorema:
  - mnogi problemi se mogu izraziti na jeziku iskazne logike
  - zadovoljavajuća valuacija kodira njihovo rešenje

DIMACS CNF

DIMACS format

- DIMACS CNF — Standardni format zapisa KNF formula u tekstualne datoteke.
- Koristi se kao ulaz SAT rešavača.

Primer

$$(x_1 \vee \neg x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge x_3$$

```
p cnf 3 4
1 -2 0
-2 3 0
-1 2 -3 0
3 0
```

Pregled

Metod tabloa

O metodu tabloa

- Metod analitičkih tabloa
- Bet (Everth Beth), Hintika (Hintikka) 1955., Smaljan (Raymond Smullyan) 1971.
- Koristi se za pokazivanje nezadovoljivosti formula (može se koristiti za dokazivanje tautologičnosti pobijanjem)
- Metod tabloa se primenjuje na formulu u proizvoljnom obliku, tj. formula se ne transformiše u normalnu formu pre primene:
  - metod tabloa donekle odgovara (lenjom) prevođenju formule u DNF
- Tabloi se obično prikazuju u obliku stabla
  - dobijeno stablo se može smatrati dokazom nezadovoljivosti

Iskazni tablo – pravila

Pravila

$$\frac{\neg\neg A}{A}$$

$$\frac{A \wedge B}{\begin{array}{c} A \\ B \end{array}}$$

$$\frac{\neg(A \vee B)}{\begin{array}{c} \neg A \\ \neg B \end{array}}$$

$$\frac{\neg(A \Rightarrow B)}{\begin{array}{c} A \\ \neg B \end{array}}$$

$$\frac{\neg(A \wedge B)}{\neg A \mid \neg B}$$

$$\frac{A \vee B}{A \mid B}$$

$$\frac{A \Rightarrow B}{\neg A \mid B}$$

Slična pravila se mogu definisati i za druge logičke veznike.

Konstrukcija tabloa

Postupak konstrukcije tabloa

1. Konstrukcija tabloa kreće od stabla koje u svom jedinom čvoru (korenu) sadrži formulu čija se zadovoljivost ispituje.
2. U svakom koraku, moguće je stablo proširiti naniže primenom nekog od datih pravila na neku formulu koja se nalaze u putanji od korena do lista koji se proširuje, a na koju to pravilo ranije nije bilo primenjeno u toj grani
3. Tablo koji se ne može proširiti, naziva se zasićenim.
4. Putanja je zatvorena ukoliko sadrži formulu i njenu negaciju (obično su u pitanju kontradiktorni literali). Tablo je zatvoren ako mu je svaka putanja zatvorena.

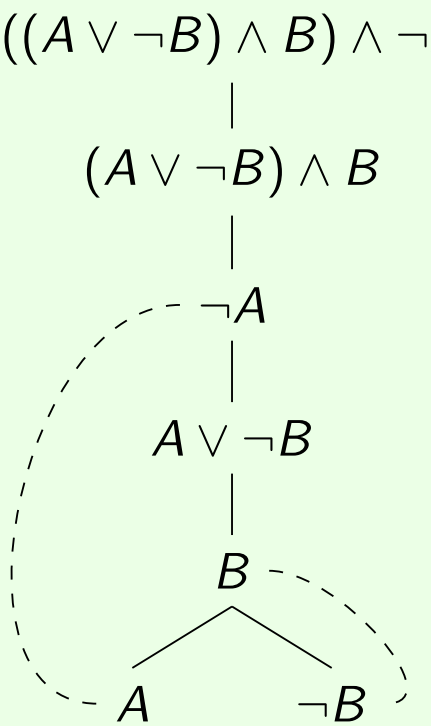
Teorema (Korektnost metoda tabloa)

Metod tabloa se zaustavlja za svaku iskaznu formulu i dobijeni tablo je zatvoren ako i samo ako je polazna iskazna formula nezadovoljiva.

Primer

Primer

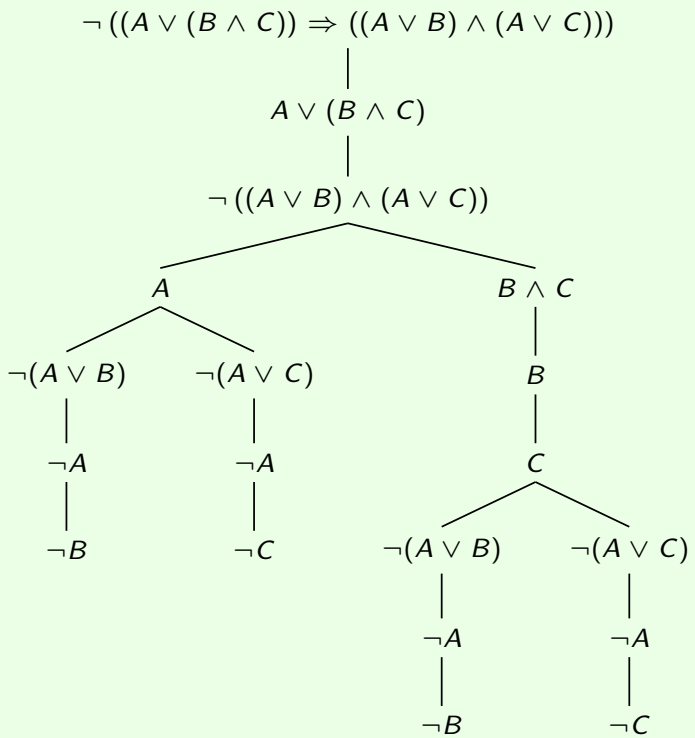
Pokažimo da je $((A \vee \neg B) \wedge B) \wedge \neg A$ *nezadovoljiva formula.*



Primer

Primer

Pokažimo da je $(A \vee (B \wedge C)) \Rightarrow ((A \vee B) \wedge (A \vee C)$ *tautologija.*



Redosled primene pravila

Heuristike

- Redosled primene pravila nije bitan za korektnost procedure.
- Bolja efikasnost se dobija ukoliko se negranajućim pravilima da prioritet.

Pregled

Metod rezolucije

O metodu rezolucije

- Metod otkriven još 1960.-tih (Davis, Putnam, Robinson, . . .)
- Metod ispitivanja (ne)zadovoljivosti formule (ili skupa formula):
  - iskazne logike
  - logike prvog reda
- Nije metod pretrage – u slučaju zadovoljive formule ne daje zadovoljavajuću valuaciju
- Pogodan je za dokazivanje valjanosti formule pobijanjem
  - rezolucijski dokazi se mogu izvesti i proveriti putem nezavisnog provaravača

Pravilo rezolucije

Pravilo rezolucije

$$\frac{\mathcal{C}' \cup \{l\} \quad \mathcal{C}'' \cup \{\bar{l}\}}{\mathcal{C}' \cup \mathcal{C}''}$$

Dobijena klauza $\mathcal{C}' \cup \mathcal{C}''$ se naziva rezolventa, dok se polazne klauze $\mathcal{C}' \cup \{l\}$ i $\mathcal{C}'' \cup \{\bar{l}\}$ nazivaju njenim roditeljima.
Rezolucija klauza $\mathcal{C}_1$ i $\mathcal{C}_2$ preko literala l će biti označavana i sa $\mathcal{C}_1 \oplus_l \mathcal{C}_2$.

Napomena

Suštinski, opravdanje ovog pravila je u tranzitivnosti implikacije.

$$\frac{\overline{\mathcal{C}'} \Rightarrow l \quad l \Rightarrow \mathcal{C}''}{\overline{\mathcal{C}'} \Rightarrow \mathcal{C}''}$$

Napomena

Pravilo rezolucije podrazumeva da su klauze skupovi, tj. da u njima nema ponavljanja literala. Ako u programu klauze predstavljamo listama, tada moramo obezbediti da one ne sadrže duplikate. Alternativa je da rezolviramo istovremeno sve kopije odgovarajućeg literala (tzv. opšte pravilo rezolucije).

Metod rezolucije

Algoritam

Neka je $\mathcal{S}$ skup klauza čiju (ne)zadovoljivost ispitujemo:

- Dokle god postoje dve klauze C' i C'' iz $\mathcal{S}$ i literal l takve da rezolventa $\mathcal{C}_1 \oplus_l \mathcal{C}_2$ ne pripada $\mathcal{S}$:
  - dodaj $\mathcal{C}_1 \oplus_l \mathcal{C}_2$ u skup $\mathcal{S}$
  - ako je dodata klauza $\mathcal{C}_1 \oplus_l \mathcal{C}_2$ prazna, završi algoritam i prijavi nezadovoljivost
- Kada više ne postoje nove rezolvente koje možemo izvesti i dodati u $\mathcal{S}$, prijavljujemo zadovoljivost

Metod rezolucije — primeri

Primer

$$\{\{\neg p, \neg q, r\}, \{\neg p, q\}, \{p\}, \{\neg r\}\}$$

$\mathcal{C}_1$	:	$\{\neg p, \neg q, r\}$	
$\mathcal{C}_2$	:	$\{\neg p, q\}$	
$\mathcal{C}_3$	:	$\{p\}$	
$\mathcal{C}_4$	:	$\{\neg r\}$	
$\mathcal{C}_5$	:	$\{\neg p, r\}$	$\mathcal{C}_1 \oplus_q \mathcal{C}_2$
$\mathcal{C}_6$	:	$\{\neg p\}$	$\mathcal{C}_4 \oplus_r \mathcal{C}_5$
$\mathcal{C}_7$	:	$\{\}$	$\mathcal{C}_3 \oplus_p \mathcal{C}_6$

Zaustavljanje rezolucije

Teorema

Metod rezolucije se zaustavlja.

Dokaz

Pošto je početni skup atoma konačan, postoji konačno mnogo različitih klauza sagrađenih od njegovih elemenata (za n atoma, postoji 2^{2n} različitih klauza). U svakom koraku u skup klauza se dodaje nova klauza koja sadrži samo literale iz polaznog skupa, te može da bude samo konačno mnogo koraka.

Saglasnost rezolucije

Lema (Saglasnost pravila rezolucije)

Ako je $\mathcal{C}_1 \in \Gamma$ *i* $\mathcal{C}_2 \in \Gamma$, *skupovi klauza* Γ *i* $\Gamma \cup \{\mathcal{C}_1 \oplus_l \mathcal{C}_2\}$ *su logički ekvivalentni.*

Dokaz

Ako je v *model za* $\Gamma \cup \{\mathcal{C}_1 \oplus_l \mathcal{C}_2\}$, *tada je, trivijalno,* v *model i za* Γ. *Obratno, neka je* v *model za* Γ. *Razmotrimo sledeće slučajeve:*

- $v \not\models l$. *Tada u* $\mathcal{C}_1$ *postoji literal* l' *različit od* l *takav da* $v \models l'$. *No,* $l' \in \mathcal{C}_1 \oplus_l \mathcal{C}_2$ *te* $v \models \mathcal{C}_1 \oplus_l \mathcal{C}_2$.
- $v \models l$. *Tada* $v \not\models \bar{l}$ *pa u* $\mathcal{C}_2$ *postoji literal* l' *različit od* $\bar{l}$ *takav da* $v \models l'$. *No,* $l' \in \mathcal{C}_1 \oplus_l \mathcal{C}_2$ *te* $v \models \mathcal{C}_1 \oplus_l \mathcal{C}_2$.

Saglasnost metoda rezolucije

Teorema

Ako metod rezolucije prijavi nezadovoljivost (tj. ako izvede praznu klauzu), onda je polazni skup klauza nezadovoljiv.

Dokaz

U svakom koraku metoda rezolucije u skup klauza se dodaje rezolventa neka dva njegova člana i novodobijeni skup je ekvivalentan prethodnom, pa samim tim, na osnovu tranzitivnosti logičke ekvivalentnosti, i polaznom. Kako poslednji skup sadrži praznu klauzu, on nije zadovoljiv pa ne može biti ni njemu ekvivalentan polazni skup klauza.

Potpunost rezolucije

Teorema

Ako je polazni skup klauza nezadovoljiv, onda metod rezolucije prijavljuje nezadovoljivost (tj. izvodi praznu klauzu).

Dokaz

Skica dokaza: posmatramo parcijalne valuacije oblika $\{l_1, \ldots, l_k\}$, $k \leq n$, gde je $l_i \in \{p_i, \neg p_i\}$. Za ovakvu parcijalnu valuaciju kažemo da je pokrivena skupom klauza $\mathcal{S}$, ako postoji klauza $C \in \mathcal{S}$ koja je netačna za tu parcijalnu valuaciju. Pretpostavimo da imamo nezadovoljiv skup klauza $\mathcal{S}$ za koji smo iscrpno primenili pravilo rezolucije na sve moguće načine. Dokažimo da $\mathcal{S}$ mora sadržati praznu klauzu. Zaista, sve potpune valuacije su sigurno pokrivene skupom $\mathcal{S}$, jer je $\mathcal{S}$ nezadovoljiv. Dalje, ako imamo dve ,,susedne" parcijalne valuacije $\{l_1, \ldots, l_{k-1}, l_k\}$ i $\{l_1, \ldots, l_{k-1}, \bar{l}_k\}$, tada iz činjenice da su obe pokrivene (npr. klauzama C i C'), sledi da je pokrivena i parcijalna valuacija $\{l_1, \ldots, l_{k-1}\}$: zaista, ako je $\overline{C}$ (ili $\overline{C'}$) podskup od $\{l_1, \ldots, l_{k-1}\}$, tada ta klauza pokriva i $\{l_1, \ldots, l_{k-1}\}$; u suprotnom, klauze C i C' sadrže, respektivno, literale $\bar{l}_k$ i l_k, pa je klauza $C \oplus_{l_k} C'$ sigurno u $\mathcal{S}$, jer je rezolucija iscrpno primenjena, a ova klauza tada pokriva valuaciju $\{l_1, \ldots, l_{k-1}\}$. Na ovaj način, indukcijom možemo dokazati da su pokrivene sve parcijalne valuacije oblika $\{l_1, \ldots, l_k\}$ za svako $k \geq 0$. Za $k = 0$ imamo praznu valuaciju koja može biti pokrivena jedino praznom klauzom, pa ona mora pripadati skupu $\mathcal{S}$.

Efikasnost metoda rezolucije

Pitanje efikasnosti i strategije pretrage

- Da bi metod rezolucije imao svojstvo potpunosti, potrebno je da se rezolucija primenjuje iscrpno
  - ovakav pristup se naziva sistematski metod rezolucije
- Ipak, nije svejedno u kom poretku će se primenjivati pravilo rezolucije nad različitim klauzama (i literalima)
  - tu na scenu stupaju različite heuristike
  - ipak, složenost je u najgorem slučaju eksponencijalna
- U nekim slučajevima potpunost je moguće obezbediti čak i bez sistematske primene rezolucije:
  - npr. ako su sve klauze takve da imaju najviše jedan pozitivan literal (tzv. Hornove klauze), tada se potpunost može obezbediti linearnom ulaznom strategijom: pravilo rezolucije se uvek primenjuje nad poslednjom izvedenom rezolventom i nekom od početnih (ulaznih) klauza
  - opisanu strategiju koristi programski jezik Prolog (doduše, u logici prvog reda)

Pregled

Davis-Putnam procedura

Osnovne informacije

- DP procedura (Davis-Putnam, 1961)
- Ne mešati sa DPLL procedurom (Davis-Putnam-Logemann-Loveland, 1962)
- Suštinski zasnovana na rezoluciji

Elementi algoritma

Elementi algoritma

- Ulaz algoritma je skup klauza $\mathcal{S}$ (kao i kod metoda rezolucije)
- Postupak uključuje tri vrste operacija:
  - propagacija jediničnih klauza (unit propagation)
  - eliminacija ,,čistih" literala (pure literal)
  - eliminacija promenljive (variable elimination)
- Prve dve operacije podrazumevaju rezonovanje, tj. zaključivanja kojima se pojednostavljuje skup klauza
  - ove operacije se pojavljuju i u DPLL algoritmu, pa ćemo ih podrobnije upoznati tamo
- Suštinski korak je eliminacija promenljive i on je zasnovan na pravilu rezolucije
- Rezolucija se primenjuje sistematski i iscrpno, promenljivu po promenljivu, pri čemu se originalne klauze u svakom koraku uklanjaju i time izabrana promenljiva eliminiše iz skupa klauza

Korak eliminacije promenljive

Eliminacija promenljive

Promenljiva p se eliminiše tako što se sve klauze polaznog skupa koje sadrže p i sve klauze koje sadrže $\neg p$ rezolivraju i zamene dobijenim rezolventama.
Dakle, umesto skupa $\mathcal{S}$ posmatra se skup

$$\mathcal{S}' = \{\mathcal{C}' \oplus_p \mathcal{C}'' \mid \mathcal{C}' \in \mathcal{S}, p \in \mathcal{C}', \mathcal{C}'' \in \mathcal{S}, \neg p \in \mathcal{C}''\} \cup$$
$$\{\mathcal{C} \mid \mathcal{C} \in \mathcal{S}, p \notin \mathcal{C}, \neg p \notin \mathcal{C}\}$$

Tautologične klauze i eliminacija promenljive

Napomena

- Ukoliko polazna klauza $C \in \mathcal{S}$ sadrži i p i $\neg p$, nakon njenog rezolviranja, ne uklanja se promenljiva p.
- Ovakve tautologične klauze mogu postojati u polaznom skupu, a mogu nastati rezolucijom
- Kako su tautologične klauze uvek tačne, one se mogu ukloniti iz skupa (na početku procedure, kao i nakon svakog koraka eliminacije)
- Ovo je neophodno raditi kako bi korak eliminacije garantovano eliminisao izabranu promenljivu p

Korektnost operacije eliminacije

Teorema

Neka je

$$\mathcal{S} = \{\mathcal{C}'_i \cup \{p\} \mid 1 \leq i \leq m\} \cup \{\mathcal{C}''_j \cup \{\neg p\} \mid 1 \leq j \leq n\} \cup \mathcal{S}_0,$$

pri čemu $\mathcal{C}'_i$ ne sadrže $\neg p$, $\mathcal{C}''_j$ ne sadrže p, dok klauze iz $\mathcal{S}_0$ ne sadrže ni p ni $\neg p$.
Neka je

$$\mathcal{S}' = \{\mathcal{C}'_i \cup \mathcal{C}''_j \mid 1 \leq i \leq m, 1 \leq j \leq n\} \cup \mathcal{S}_0.$$

Tada je

$$\mathcal{S} \equiv_s \mathcal{S}'.$$

Korektnost koraka eliminacije

Dokaz

Ako je $\mathcal{S}$ zadovoljiv, na osnovu leme o pravilu rezolucije, zadovoljiv je i $\mathcal{S}'$.
Neka je v valuacija koja zadovoljava $\mathcal{S}'$. Valuacija v ili istovremeno zadovoljava sve $\mathcal{C}'_i$ ili istovremeno zadovoljava sve $\mathcal{C}''_j$. Zaista, pošto v zadovoljava sve $\mathcal{C}'_i \cup \mathcal{C}''_j$, ako v ne zadovoljava neko $\mathcal{C}'_i$, onda zadovoljava sve $\mathcal{C}''_j$, i obratno.
Ako v zadovoljava sve $\mathcal{C}'_i$, posmatrajmo valuaciju v' dobijenu od v' postavljanjem p na netačno. Sve klauze $\mathcal{C}'_i \cup \{p\}$ su zadovoljene u v' jer su i sve $\mathcal{C}'_i$, sve klauze $\mathcal{C}''_j \cup \{\neg p\}$ jer je p netačno, dok su klauze iz $\mathcal{S}_0$ zadovoljene jer pripadaju i $\mathcal{S}'$.
Ako v zadovoljava sve $\mathcal{C}''_j$, dokaz je analogan.

Zaustavljanje, saglasnost, potpunost

Zaustavljanje saglasnost, potpunost

- Nakon svakog koraka eliminacije dobijeni skup klauza sadrži jednu promenljivu manje
- Samim tim, kada eliminišemo sve promenljive, moguća su dva scenarija:
  - algoritam se zaustavlja i kao rezultat dobijamo prazan skup klauza (koji je zadovoljiv)
  - algoritam se zaustavlja i kao rezultat dobijamo skup klauza koji sadrži samo praznu klauzu (ovaj skup je nezadovoljiv)
- U svakom slučaju, algoritam se sigurno zaustavlja u konačnom broju koraka i prijavljuje nezadovoljivost akko je polazni skup klauza nezadovoljiv (na osnovu prethodne teoreme)
- Otuda algoritam ima svojstvo zaustavljanja, saglasnosti i potpunosti

Napomena

Doduše, gornja konstatacija za sada važi ako primenjujemo samo operaciju eliminacije. Kasnije ćemo videti da slične teoreme važe i za druge dve operacije koje se koriste u algoritmu.

Pregled

DPLL procedura

Pregled

- DPLL (Davis-Putnam-Logemann-Loveland, 1962)
- Ispituje zadovoljivost iskaznih formula (u KNF)
  - kao i ranije, i ovde je ulaz dat kao skup klauza, pri čemu je svaka klauza skup literala
- Izmena DP procedure (Davis-Putnam, 1961) u cilju smanjivanja utroška memorije
- Osnova savremenih CDCL SAT rešavača
- Predstavlja proceduru pretrage: ako je skup klauza zadovoljiv, vraća zadovoljavajuću valuaciju
- Razlikovaćemo DPLL pretragu i DPLL proceduru (koja uz pretragu uključuje i elemente zaključivanja kojima se sužava prostor pretrage)

DPLL procedura — ideja koraka pretrage

Pretraga

- Pretraga za zadovoljavajućom valuacijom zasniva se na ispitivanju obe moguće istinitosne vrednosti promenljivih koje se javljaju u formuli
- Postavljanje vrednosti neke promenljive indukuje vrednost oba njoj odgovarajuća literala
  - postavljanje vrednosti promenljive p na *tačno* čini da je literal p tačan, a $\neg p$ netačan
  - postavljanje vrednosti promenljive p na *netačno* čini da je literal p netačan, a $\neg p$ tačan
  - otuda ćemo umesto ,,postavljanja promenljive p na tačno (netačno)" reći ,,postavljanje literala p ($\neg p$) na tačno"
- Vrednost nekog literala l se postavi na tačno
- Postavljanje literala na tačno dovodi do uprošćavanja formule:
  - klauze koje sadrže l su tačne pa se brišu
  - iz klauza koje sadrže $\overline{l}$ se taj literal briše, jer je netačan
- za uprošćeni skup klauza se rekurzivno proverava zadovoljivost
- Ako je nakon postavljanja l na tačno dobijena nezadovoljivost, pokušava se sa postavljanjem $\overline{l}$ na tačno
- Ako se i u tom slučaju dobije nezadovoljivost, formula je nezadovoljiva

DPLL procedura — $F[[l \to \top]]$

Uprošćavanje formule

- Efekat uprošćavanja formule postavljanjem nekog literala na tačno se može postići tako što se sva njegova pojavljivanja zamene logičkom konstantom $\top$, a pojavljivanja njemu suprotnog literala zamene logičkom konstantom $\bot$, a zatim primeni eliminacija logičkih konstanti
  - rezultat je upravo ono što je već rečeno: klauze koje sadrže tačan literal se brišu, a iz klauza koje sadrže suprotan literal se briše taj literal
- Neka $F[[l \to \top]]$ označava formulu $F[l \to \top][\bar{l} \to \bot]$
- Neka $F[[l \to \bot]]$ označava formulu $F[\bar{l} \to \top][l \to \bot]$ (tj. $F[[\bar{l} \to \top]]$)

Korak split

Split

Pretraga u okviru DPLL procedure je zasnovana na koraku split koji ispitivanje zadovoljivosti formule F svodi na ispitivanje zadovoljivosti formula $F[[I \to \top]]$ i $F[[I \to \bot]]$.
Logičko opravdanje za ovo je u ekvizadovoljivosti:

$$F \equiv_s F[[I \to \top]] \vee F[[I \to \bot]]$$

DPLL i tautologične klauze

Tautologične klauze

- Klauze koje sadrže međusobno suprotne literale ne utiču na zadovoljivost formule (jer su uvek tačne)
- Ove klauze je moguće ukloniti pre primene procedure (često već u fazi prevođenja u KNF)
- Primetimo da za razliku od DP procedure, ne postoji mogućnost da se naknadno pojave nove tautologične klauze

DPLL pretraga — trivijalni slučajevi

Izlaz iz rekurzije

- Kako je DPLL pretraga rekurzivna procedura, mora imati i izlaz iz rekurzije
- Kao i kod DP procedure, mogu nastupiti dva ,,trivijalna" slučaja:
  - skup klauza postane prazan nakon uprošćavanja: DPLL pretraga prijavljuje zadovoljivost
  - prilikom uprošćavanja u skupu klauza se pojavi prazna klauza: DPLL pretraga prijavljuje nezadovoljivost

DPLL pretraga — pseudokod

Pseudokod DPLL pretrage

```
function dpll (F : CNF Formula) : (SAT, UNSAT)
begin
   if C is empty then return SAT
   else if there is an empty clause in F then return UNSAT
   else begin
      select a literal l occurring in F
      if dpll(F[[l → ⊤]]) = SAT then return SAT
      else return dpll(F[[l → ⊥]])
   end
end
```

Korektnost koraka split

Stav (Split)

Skup klauza $\mathcal{F}$ je zadovoljiv, ako i samo ako je zadovoljiv $\mathcal{F}[[l \to \top]]$ ili je zadovoljiv $\mathcal{F}[[l \to \bot]]$, gde je l proizvoljan literal.

Dokaz

Neka je v model za $\mathcal{F}$. Ako je $v \vDash l$, onda je v model i za $\mathcal{F}[[l \to \top]]$, a ako $v \nvDash l$, pošto $v \vDash \bar{l}$, onda je v model za $\mathcal{F}[[l \to \bot]]$.
Ako je v model za $\mathcal{F}[[l \to \top]]$, onda postoji v' (dobijena od v postavljanjem vrednosti l na tačno) tako da je v' model za $\mathcal{F}$.
Ako je v model za $\mathcal{F}[[l \to \bot]]$, onda postoji v' (dobijena od v postavljanjem vrednosti l na netačno) tako da je v' model za $\mathcal{F}$.

Korektnost trivijalnih slučajeva

Stav

- *Prazan skup klauza je zadovoljen u svakoj valuaciji.*
- *Prazna klauza nema model.*
- *Ako skup formula sadrži praznu klauzu on nema model (tj. nije zadovoljiv).*

Korektnost DPLL pretrage

Teorema (Zaustavljanje)

Ukoliko se korak split uvek primenjuje na literal koji je deo tekućeg skupa klauza, DPLL pretraga se zaustavlja.

Dokaz

Broj različitih promenljivih u tekućem skupu klauza se smanjuje pri svakom rekurzivnom pozivu, pa se procedura mora zaustaviti.

Teorema (Saglasnost i potpunost)

Skup klauza $\mathcal{F}$ je zadovoljiv ako i samo je `dpll` $\mathcal{F} = True$.

Dokaz

Indukcijom, na osnovu definicije funkcije `dppl`*, uz korišćenje lema o korektnosti pojedinačnih koraka.*

Elementi zaključivanja

Elementi rezonovanja (zaključivanja)

- Prethodna procedura ne uključuje nikakve elemente zaključivanja
- U nekim slučajevima može se izbeći analiziranje obe grane u split pravilu
- DPLL procedura uvodi dva ovakva koraka koji značajno mogu da smanje prostor pretrage:
  - propagacija jediničnih klauza (eng. unit propagation)
  - eliminacija ,,čistih" literala (eng. pure literal)
- Podsetnik: isti ovi koraci su postojali i u DP proceduri (tamo nisu objašnjeni)

Primeri

Primer

$$\mathcal{F}_0 = \{\{\neg x_1, \neg x_2\}, \{x_2\}, \{\neg x_2, x_3, \neg x_4\}, \{\neg x_3, x_4\}, \{x_4, x_5\}, \{\neg x_3, \neg x_4, x_1\}\}$$

Unit propagate: *valuacija koja x_2 postavlja na netačno ne može biti model.*

$$\mathcal{F}_1 = \mathcal{F}_0[[x_2 \to \top]] = \{\{\neg x_1\}, \{x_3, \neg x_4\}, \{\neg x_3, x_4\}, \{x_4, x_5\}, \{\neg x_3, \neg x_4, x_1\}\}$$

Unit propagate: *valuacija koja x_1 postavlja na tačno ne može biti model.*

$$\mathcal{F}_2 = \mathcal{F}_1[[x_1 \to \bot]] = \{\{x_3, \neg x_4\}, \{\neg x_3, x_4\}, \{x_4, x_5\}, \{\neg x_3, \neg x_4\}\}$$

Pure literal: *Ne smeta da se x_5 postavi na tačno.*

$$\mathcal{F}_3 = \mathcal{F}_2[[x_5 \to \top]] = \{\{x_3, \neg x_4\}, \{\neg x_3, x_4\}, \{\neg x_3, \neg x_4\}\}$$

Tek sada kreće pretraga.

Elementi zaključivanja

Pravila zaključivanja

Jedinična propagacija (engl. *unit propagation*) — Ako formula sadrži klauzu sa tačno jednim literalom $\{l\}$, onda taj literal mora biti tačan u eventualnoj zadovoljavajućoj valuaciji. Klauze sa samo jednim literalom nazivamo jedinične klauze (engl. unit clause).

Čisti literal (engl. *pure literal*) — Ako se u formuli javlja neki literal l, a ne javlja se njemu suprotan literal $\bar{l}$, onda taj literal može biti postavljen na tačno u eventualnoj zadovoljavajućoj valuaciji. Ovakav literal nazivamo čist literal.

DPLL procedura — pseudokod

Algoritam

```
function dpll (F : CNF Formula) : bool
begin
   if F is empty then return True
   else if there is an empty clause in F then return False
   else if there is a pure literal l in F then return dpll(F[[l → ⊤]])
   else if there is a unit clause [l] in F then return dpll(F[[l → ⊤]])
   else begin
      select a literal l occurring in F
      if dpll(F[[l → ⊤]]) = SAT then return SAT
      else return dpll(F[[l → ⊥]])
   end
end
```

Korektnost koraka propagacije jediničnih klauza

Stav (Unit propagate)

Ukoliko je $\{l\} \in \mathcal{F}$*, tada je*

$$\mathcal{F} \equiv_s \mathcal{F}[[l \to \top]].$$

Dokaz

Neka je v *model za* $\mathcal{F}$*. Pošto je* $\{l\} \in \mathcal{F}$*, važi* $v \vDash l$*, te* $v \vDash \mathcal{F}[[l \to \top]]$.
Ako je v *model za* $\mathcal{F}[[l \to \top]]$*, onda postoji* v' *(dobijena od* v *postavljanjem vrednosti* l *na tačno) tako da je* v' *model za* $\mathcal{F}$.

Korektnost koraka eliminacije čistih literala

Stav (Pure literal)

Ako se u skupu klauza $\mathcal{F}$ javlja literal l, a ne javlja literal $\bar{l}$, tada je

$$\mathcal{F} \equiv_s \mathcal{F}[[l \to \top]].$$

Dokaz

Ako je $\mathcal{F}$ zadovoljiv, zadovoljiv je i $\mathcal{F}[[l \to \top]]$ (kao njegov podskup). Ako je v model za $\mathcal{F}[[l \to \top]]$, onda postoji v' (dobijena od v postavljanjem vrednosti l na tačno) tako da je v' model za $\mathcal{F}$.

Efikasnost DPLL procedure

Efikasnost

- U najgorem slučaju, DPLL procedura zahteva eksponencijalni broj koraka u odnosu na veličinu ulazne formule (korak split generiše eksponencijalnost). Ovaj problem nije moguće razrešiti.
- Osnovna varijanta procedure je izrazito neefikasna:
  - Pri svakom koraku vrši se modifikacija formule (skupa klauza).
  - Rekurzivna implementacija značajno kvari efikasnost (formule koje mogu da budu veoma velike se prenose kao parametar funkcije i slažu na programski stek).

Pregled

O CDCL algoritmu

CDCL – Conflict Driven Clause Learning

- CDCL algoritam je nastao početkom 21. veka, sa ciljem unapređenja efikasnosti klasičnog DPLL algoritma
- Unapređenja idu u nekoliko pravaca:
  - inkrementalna izgradnja parcijalne valuacije, umesto transformacija formule
  - iterativna implementacija umesto rekurzivne
  - efikasan obilazak skupa klauza zasnovan na pogodnim strukturama podataka
  - efikasne heuristike grananja
  - analiza konflikata zasnovana na rezoluciji i nehronološko vraćanje unazad
  - učenje klauza iz konflikata, u cilju sprečavanja sličnih konflikata u budućnosti
- Sva ova unapređenja detaljno razmatramo u nastavku

Prosleđivanje parcijalnih valuacija

Alternativni pristup – parcijalne valuacije

- Umesto zamene izabranog literala sa $\top$ u formuli (i potom pojednostavljivanja skupa klauza) moguće je da se formula ne menja, a da se literal postavlja na tačno tako što se dodaje u parcijalnu valuaciju
- Podsetnik: parcijalna valuacija može da se shvati kao skup literala koji je konzistentan tj. ne sadrži istovremeno dva suprotna literala
- Pošto je valuacija parcijalna, promenljiva, odnosno literal, mogu u njoj da budu tačni, netačni ili nedefinisani
- Sada ćemo imati samo jednu formulu $\mathcal{F}$ koja se ne menja i koja je globalno vidljiva, a rekurzivnim pozivima ćemo prosleđivati parcijalne valuacije

Pojam jedinične klauze

Jedinične klauze – uopštenje

- Ukoliko se vrši uprošćavanje formule, iz klauza se uklanjaju literali
  - otuda neka klauza može postati jedinična nakon pojednostavljivanja i ako prvobitno to nije bila
- U slučaju da se ne vrši uprošćavanje formule već izgradnja parcijalne valuacije, pojam jediničnih klauza se mora uopštiti:

Definicija

Klauza c je jedinična u odnosu na parcijalnu valuaciju v akko sadrži literal l nedefinisan u v, dok su joj svi literali različiti od l netačni u v.

Čisti literali – uopštenje

Slično se može proširiti i pojam čistih literala:

Definicija

Literal l je čist u odnosu na parcijalnu valuaciju v ako je nedefinisan u v i svaka klauza $C \in \mathcal{F}$ ili tačna u v, ili ne sadrži literal $\overline{l}$.

DPLL procedura — prosleđivanje valuacija — pseudokod

Algoritam

```
function dpll (M : Valuation) : (SAT, UNSAT)
begin
   if M ⊨¬ F (i.e., there is c ∈ F s.t. M ⊨¬ c) then return UNSAT
   else if M ⊨ F then return SAT
   else if there is a unit clause (i.e., there is a clause
      l ∨ l1 ∨ ... ∨ lk in F s.t. l, l̄ ∉ M, l̄1, ..., l̄k ∈ M) then
      return dpll(M ∪ {l})
   else if there is a pure literal l in F then
      return dpll(M ∪ {l})
   else begin
      select a literal l s.t. l ∈ F, l, l̄ ∉ M
      if dpll(M ∪ {l}) = SAT then return SAT
      else return dpll(M ∪ {l̄})
   end
end
```

Provera tačnosti u parcijalnoj valuaciji?

Problem?

- Pokazuje se da postoje efikasni postupci za proveru da li u formuli postoji netačna ili jedinična klauza, ali ne postoji efikasan postupak kojim bi se utvrdilo da li je formula tačna u nekoj parcijalnoj valuaciji.
- Može li se taj test izbeći?

Stav

Ukoliko su sve promenljive iz $\mathcal{F}$ definisane u parcijalnoj valuaciji M, tada je $\mathcal{F}$ ili tačna ili netačna u M, tj. $M \models \mathcal{F}$ ili $M \models \neg \mathcal{F}$.

DPLL procedura — prosleđivanje valuacija — pseudokod

Algoritam

```
function dpll (M : Valuation) : (SAT, UNSAT)
begin
   if M ⊨¬ F (i.e., there is c ∈ F s.t. M ⊨¬ c) then return UNSAT
   else if M is total wrt. the variables of F then return SAT
   else there is a unit clause (i.e., there is a clause
      l ∨ l1 ∨ ... ∨ lk in F s.t. l, l̄ ∉ M, l̄1, ..., l̄k ∈ M) then
      return dpll(M ∪ {l})
   else if there is a pure literal l in F then
      return dpll(M ∪ {l})
   else begin
      select a literal l s.t. l ∈ F, l, l̄ ∉ M
      if dpll(M ∪ {l}) = SAT then return SAT
      else return dpll(M ∪ {l̄})
   end
end
```

Primer

Primer

$$\mathcal{F} = \{\{\neg x_1, \neg x_2\}, \{x_2\}, \{\neg x_2, x_3, \neg x_4\}, \{\neg x_3, x_4\}, \{x_4, x_5\}\}$$

$$v = \{\}$$

Unit propagate: *Klauza* $\{x_2\}$ *je jedinična.*

$$v = \{x_2\}$$

Unit propagate: *Klauza* $\{\neg x_1, \neg x_2\}$ *je jedinična.*

$$v = \{x_2, \neg x_1\}$$

Pure literal: *Literal* x_5 *je čist.*

$$v = \{x_2, \neg x_1, x_5\}$$

Tek sada kreće pretraga.

Primer

$$\mathcal{F} = \{\{\neg x_1, \neg x_2\}, \{x_2\}, \{\neg x_2, x_3, \neg x_4\}, \{\neg x_3, x_4\}, \{x_4, x_5\}, \{\neg x_3, \neg x_4, x_1\}\}$$

dpll($\{\}$)
dpll($\{x_2\}$) — *unitPropagate* ($c = \{x_2\}$)
dpll($\{x_2, \neg x_1\}$) — *unitPropagate* ($c = \{\neg x_1, \neg x_2\}$)
dpll($\{x_2, \neg x_1, x_5\}$) – *pureLiteral* (x_5)
dpll($\{x_2, \neg x_1, x_5, x_3\}$) — *split* ($x_3$)
dpll($\{x_2, \neg x_1, x_5, x_3, x_4\}$) — *unitPropagate* ($c = \{\neg x_3, x_4\}$)
return False — $c = \{\neg x_3, \neg x_4, x_1\}$

dpll($\{x_2, \neg x_1, x_5, \neg x_3\}$) — *split* ($x_3$) – *druga grana*
dpll($\{x_2, \neg x_1, x_5, \neg x_3, \neg x_4\}$) — *unitPropagate* ($c = \{\neg x_2, x_3, \neg x_4\}$)
return True

Iterativna implementacija

Predstavljanje parcijalne valuacije pomoću steka

Rekurzivna implementacija se može zameniti iterativnom, tako što se parcijalna valuacija predstavi stekom označenih literala:

- čuvanje parcijalne valuacije u obliku steka omogućava efikasno vraćanje unazad (engl. backtracking)
- literali na steku su označeni ili kao literali odlučivanja (engl. decision literal) ili kao izvedeni literali (engl. inferred literal)
- literali odlučivanja su rezultat naših odluka i njima započinju nivoi odlučivanja
- izvedeni literali su rezultat rezonovanja (jedinične propagacije, čisti literali) i posledica su odluka koje smo doneli
- stek označenih literala se često naziva i trag (engl. trail)

Apstraktni opis procedure

Apstraktni opis procedure

- Razmatranje procedure opisane na nivou koda je obično suviše komplikovano jer sadrži dosta implementacionih detalja
- Umesto koda, moguće je opisati tekuće stanje procedure i dati pravila koja opisuju kako se može preći iz stanja u stanje
- Ovakav način apstraktnog opisa procedura omogućava lakšu analizu i dokazivanje korektnosti procedure
- U slučaju CDCL procedure, stanje je predstavljeno stekom parcijalne valuacije (tragom)

Apstraktni opis procedure

Sistem pravila za promenu stanja

Stanje rešavača

- M - označena parcijalna valuacija u vidu steka (trag)

Decide:

$$\frac{l \in F \quad l, \bar{l} \notin M}{M := M \,|\, l}$$

UnitPropagate:

$$\frac{l \vee l_1 \vee \ldots \vee l_k \in F \qquad \bar{l}_1, \ldots, \bar{l}_k \in M \qquad l, \bar{l} \notin M}{M := M \; l}$$

Backtrack:

$$\frac{M \models \neg F \quad M = M' \,|\, l \, M'' \quad \text{decisions } M'' = [\,]}{M := M' \; \bar{l}}$$

Primer

$F = [[-1, +2], [-1, -3], [-2, +4, +5], [+3, -4, -5], [-4, +5]]$

Primenjeno pravilo	*sat?*	*M*
	UNDEF	
Decide $(l = +1)$	*UNDEF*	$\mid +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\mid +1, +2$
UnitProp $(c = [-1, -3],\ l = -3)$	*UNDEF*	$\mid +1, +2, -3$
Decide $(l = +4)$	*UNDEF*	$\mid +1, +2, -3, \mid +4$
UnitProp $(c = [-4, +5],\ l = +5)$	*UNDEF*	$\mid +1, +2, -3, \mid +4, +5$
Backtrack $(M \models \neg [+3, -4, -5])$	*UNDEF*	$\mid +1, +2, -3, -4$
UnitProp $(c = [-2, +4, +5],\ l = +5)$	*UNDEF*	$\mid +1, +2, -3, -4, +5$
$M \not\models \neg F$, $(\text{vars } M) = (\text{vars } F)$	*SAT*	$\mid +1, +2, -3, -4, +5$

Primer

$F = [[-1, +2], [-1, -3], [-2, +4, +5], [+3, -4, -5], [-4, +5]]$

Primenjeno pravilo	*sat?*	M
	UNDEF	
Decide ($l = +1$)	*UNDEF*	$\mid +1$
UnitProp ($c = [-1, +2]$, $l = +2$)	*UNDEF*	$\mid +1, +2$
UnitProp ($c = [-1, -3]$, $l = -3$)	*UNDEF*	$\mid +1, +2, -3$
Decide ($l = +4$)	*UNDEF*	$\mid +1, +2, -3, \mid +4$
UnitProp ($c = [-4, +5]$, $l = +5$)	*UNDEF*	$\mid +1, +2, -3, \mid +4, +5$
Backtrack ($M \models \neg [+3, -4, -5]$)	*UNDEF*	$\mid +1, +2, -3, -4$
UnitProp ($c = [-2, +4, +5]$, $l = +5$)	*UNDEF*	$\mid +1, +2, -3, -4, +5$
$M \not\models \neg F$, (vars M) = (vars F)	*SAT*	$\mid +1, +2, -3, -4, +5$

Primer

$F = [[-1, +2], [-1, -3], [-2, +4, +5], [+3, -4, -5], [-4, +5]]$

Primenjeno pravilo	*sat?*	*M*
	UNDEF	
Decide $(l = +1)$	*UNDEF*	$\mid +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\mid +1, +2$
UnitProp $(c = [-1, -3],\ l = -3)$	*UNDEF*	$\mid +1, +2, -3$
Decide $(l = +4)$	*UNDEF*	$\mid +1, +2, -3, \mid +4$
UnitProp $(c = [-4, +5],\ l = +5)$	*UNDEF*	$\mid +1, +2, -3, \mid +4, +5$
Backtrack $(M \models \neg [+3, -4, -5])$	*UNDEF*	$\mid +1, +2, -3, -4$
UnitProp $(c = [-2, +4, +5],\ l = +5)$	*UNDEF*	$\mid +1, +2, -3, -4, +5$
$M \not\models \neg F$, $(\text{vars } M) = (\text{vars } F)$	*SAT*	$\mid +1, +2, -3, -4, +5$

Primer

$F = [[-1, +2], [-1, -3], [-2, +4, +5], [+3, -4, -5], [-4, +5]]$

Primenjeno pravilo	*sat?*	*M*
	UNDEF	
Decide $(l = +1)$	*UNDEF*	$\mid +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\mid +1, +2$
UnitProp $(c = [-1, -3],\ l = -3)$	*UNDEF*	$\mid +1, +2, -3$
Decide $(l = +4)$	*UNDEF*	$\mid +1, +2, -3, \mid +4$
UnitProp $(c = [-4, +5],\ l = +5)$	*UNDEF*	$\mid +1, +2, -3, \mid +4, +5$
Backtrack $(M \models \neg [+3, -4, -5])$	*UNDEF*	$\mid +1, +2, -3, -4$
UnitProp $(c = [-2, +4, +5],\ l = +5)$	*UNDEF*	$\mid +1, +2, -3, -4, +5$
$M \not\models \neg F$, $(\text{vars } M) = (\text{vars } F)$	*SAT*	$\mid +1, +2, -3, -4, +5$

Primer

$F = [[-1, +2], [-1, -3], [-2, +4, +5], [+3, -4, -5], [-4, +5]]$

Primenjeno pravilo	*sat?*	*M*
	UNDEF	
Decide $(l = +1)$	*UNDEF*	$\mid +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\mid +1, +2$
UnitProp $(c = [-1, -3],\ l = -3)$	*UNDEF*	$\mid +1, +2, -3$
Decide $(l = +4)$	*UNDEF*	$\mid +1, +2, -3, \mid +4$
UnitProp $(c = [-4, +5],\ l = +5)$	*UNDEF*	$\mid +1, +2, -3, \mid +4, +5$
Backtrack $(M \models \neg [+3, -4, -5])$	*UNDEF*	$\mid +1, +2, -3, -4$
UnitProp $(c = [-2, +4, +5],\ l = +5)$	*UNDEF*	$\mid +1, +2, -3, -4, +5$
$M \not\models \neg F$, $(\text{vars } M) = (\text{vars } F)$	*SAT*	$\mid +1, +2, -3, -4, +5$

Primer

$F = [[-1, +2], [-1, -3], [-2, +4, +5], [+3, -4, -5], [-4, +5]]$

Primenjeno pravilo	*sat?*	*M*
	UNDEF	
Decide $(l = +1)$	*UNDEF*	$\mid +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\mid +1, +2$
UnitProp $(c = [-1, -3],\ l = -3)$	*UNDEF*	$\mid +1, +2, -3$
Decide $(l = +4)$	*UNDEF*	$\mid +1, +2, -3, \mid +4$
UnitProp $(c = [-4, +5],\ l = +5)$	*UNDEF*	$\mid +1, +2, -3, \mid +4, +5$
Backtrack $(M \models \neg [+3, -4, -5])$	*UNDEF*	$\mid +1, +2, -3, -4$
UnitProp $(c = [-2, +4, +5],\ l = +5)$	*UNDEF*	$\mid +1, +2, -3, -4, +5$
$M \not\models \neg F$, $(\text{vars } M) = (\text{vars } F)$	*SAT*	$\mid +1, +2, -3, -4, +5$

Primer

$F = [[-1, +2], [-1, -3], [-2, +4, +5], [+3, -4, -5], [-4, +5]]$

Primenjeno pravilo	*sat?*	M
	UNDEF	
Decide $(l = +1)$	*UNDEF*	$\mid +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\mid +1, +2$
UnitProp $(c = [-1, -3],\ l = -3)$	*UNDEF*	$\mid +1, +2, -3$
Decide $(l = +4)$	*UNDEF*	$\mid +1, +2, -3, \mid +4$
UnitProp $(c = [-4, +5],\ l = +5)$	*UNDEF*	$\mid +1, +2, -3, \mid +4, +5$
Backtrack $(M \models \neg [+3, -4, -5])$	*UNDEF*	$\mid +1, +2, -3, -4$
UnitProp $(c = [-2, +4, +5],\ l = +5)$	*UNDEF*	$\mid +1, +2, -3, -4, +5$
$M \not\models \neg F$, $(\text{vars } M) = (\text{vars } F)$	*SAT*	$\mid +1, +2, -3, -4, +5$

Primer

$F = [[-1, +2], [-1, -3], [-2, +4, +5], [+3, -4, -5], [-4, +5]]$

Primenjeno pravilo	*sat?*	M
	UNDEF	
Decide $(l = +1)$	*UNDEF*	$\mid +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\mid +1, +2$
UnitProp $(c = [-1, -3],\ l = -3)$	*UNDEF*	$\mid +1, +2, -3$
Decide $(l = +4)$	*UNDEF*	$\mid +1, +2, -3, \mid +4$
UnitProp $(c = [-4, +5],\ l = +5)$	*UNDEF*	$\mid +1, +2, -3, \mid +4, +5$
Backtrack $(M \models \neg [+3, -4, -5])$	*UNDEF*	$\mid +1, +2, -3, -4$
UnitProp $(c = [-2, +4, +5],\ l = +5)$	*UNDEF*	$\mid +1, +2, -3, -4, +5$
$M \not\models \neg F$, $(\text{vars } M) = (\text{vars } F)$	*SAT*	$\mid +1, +2, -3, -4, +5$

Primer

$F = [[-1, +2], [-1, -3], [-2, +4, +5], [+3, -4, -5], [-4, +5]]$

Primenjeno pravilo	*sat?*	*M*
	UNDEF	
Decide $(l = +1)$	*UNDEF*	$\mid +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\mid +1, +2$
UnitProp $(c = [-1, -3],\ l = -3)$	*UNDEF*	$\mid +1, +2, -3$
Decide $(l = +4)$	*UNDEF*	$\mid +1, +2, -3, \mid +4$
UnitProp $(c = [-4, +5],\ l = +5)$	*UNDEF*	$\mid +1, +2, -3, \mid +4, +5$
Backtrack $(M \models \neg [+3, -4, -5])$	*UNDEF*	$\mid +1, +2, -3, -4$
UnitProp $(c = [-2, +4, +5],\ l = +5)$	*UNDEF*	$\mid +1, +2, -3, -4, +5$
$M \not\models \neg F$, $(\text{vars } M) = (\text{vars } F)$	*SAT*	$\mid +1, +2, -3, -4, +5$

Efikasnost implementacije

Kako efikasno pronaći konfliktne i jedinične klauze?

- Skup klauza može biti veoma veliki
- Prolazak kroz skup svih klauza radi pronalaženja konfliktnih i jediničnih klauza može biti veoma skup
- Efikasna implementacija: shema dva posmatrana literala (engl. two-watched-literals scheme)
- Zasniva se na observaciji da dokle god u klauzi postoje bar dva literala čija je vrednost nedefinisana u tekućoj parcijalnoj valuaciji, ona ne može biti ni jedinična ni konfliktna

Shema dva posmatrana literala

Shema dva posmatrana literala

- Za svaku klauzu C se odaberu dva trenutno nedefinisana literala za posmatranje ($watch1(C)$ i $watch2(C)$)
- Za svaki literal l održavamo listu $watched(l)$ svih klauza u kojima je l jedan od posmatranih literala
- Kada se literal $\bar{l}$ doda u parcijalnu valuaciju, obilazi se lista $watched(l)$
- Za svaku klauzu $C \in watched(l)$ traži se alternativni posmatrani literal (tj. neki literal $l' \in C$ različit od $watch1(C)$ i $watch2(C)$ koji nije netačan u toj parcijalnoj valuaciji, ako postoji)
- Ako ga pronadjemo, tada tu klauzu prebacujemo u listu $watched(l')$
- U suprotnom, ako je drugi posmatrani literal takodje netačan u tekućoj parcijalnoj valuaciji, prijavljujemo konflikt
- Inače, propagiramo drugi posmatrani literal

Analiza konflikata

Šta je uzrok konflikta?

- Kada neka klauza postane netačna, uzrok često nije u poslednjem literalu odlučivanja
  - stoga, njegovom negacijom ponovo dobijamo konflikt
- Vrlo često je uzrok u nekom ranijem literalu odlučivanja, ali se to ne vidi odmah:
  - mi rezonovanjem donosimo zaključke isključivo na osnovu tekućeg skupa klauza
  - međutim, postoje klauze koje nisu u našem skupu, ali jesu logička posledica našeg skupa (npr. klauze koje bismo mogli dobiti pravilom rezolucije)
  - da su nam te klauze na raspolaganju, mnoge konflikte otkrili bismo znatno ranije
- Ideja je da, kada se već desi konflikt, probamo da otkrijemo njegov pravi uzrok (tj. izvedenu klauzu koja je postala netačna znatno ranije):
  - ovo će nam omogućiti da se vratimo na neki dublji nivo na steku
  - ovim se izbegava nepotrebni prolazak kroz grane prostora pretrage koje sigurno ne sadrže zadovoljavajuću valuaciju

Primer

$F_0 = [[-1, +2], [-3, +4], [-1, -3, +5], [-2, -4, -5], [-2, +3, +5, -6], [-1, +3, -5, -6], [+1, -6], [+1, +7]]$.

Primenjeno pravilo	*satFlag*	*M*
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\mid +6$
UnitProp $(c = [+1, -6],\ l = +1)$	*UNDEF*	$\mid +6, +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\mid +6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\mid +6, +1, +2, \mid +7$
Decide $(l = +3)$	*UNDEF*	$\mid +6, +1, +2, \mid +7, \mid +3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\mid +6, +1, +2, \mid +7, \mid +3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\mid +6, +1, +2, \mid +7, \mid +3, +4, +5$
Backtrack $(M \models \neg [-2, -4, -5])$	*UNDEF*	$\mid +6, +1, +2, \mid +7, -3$
UnitProp $(c = [-2, +3, +5, -6],\ l = +5)$	*UNDEF*	$\mid +6, +1, +2, \mid +7, -3, +5$
Backtrack $(M \models \neg [-1, +3, -5, -6])$	*UNDEF*	$\mid +6, +1, +2, -7$
Decide $(l = +3)$	*UNDEF*	$\mid +6, +1, +2, -7, \mid +3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\mid +6, +1, +2, -7, \mid +3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\mid +6, +1, +2, -7, \mid +3, +4, +5$
Backtrack $(M \models \neg [-2, -4, -5])$	*UNDEF*	$\mid +6, +1, +2, -7, -3$
UnitProp $(c = [-2, +3, +5, -6],\ l = +5)$	*UNDEF*	$\mid +6, +1, +2, -7, -3, +5$
Backtrack $(M \models \neg [-1, +3, -5, -6])$	*UNDEF*	-6

Primer

$F_0 = [[-1, +2], [-3, +4], [-1, -3, +5], [-2, -4, -5], [-2, +3, +5, -6], [-1, +3, -5, -6], [+1, -6], [+1, +7]]$.

Primenjeno pravilo	*satFlag*	*M*
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\|+6$
UnitProp $(c = [+1, -6], l = +1)$	*UNDEF*	$\|+6, +1$
UnitProp $(c = [-1, +2], l = +2)$	*UNDEF*	$\|+6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\|+6, +1, +2, \|+7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3$
UnitProp $(c = [-3, +4], l = +4)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4$
UnitProp $(c = [-1, -3, +5], l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Backtrack $(M \models \neg [-2, -4, -5])$	*UNDEF*	$\|+6, +1, +2, \|+7, -3$
UnitProp $(c = [-2, +3, +5, -6], l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, -3, +5$
Backtrack $(M \models \neg [-1, +3, -5, -6])$	*UNDEF*	$\|+6, +1, +2, -7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, -7, \|+3$
UnitProp $(c = [-3, +4], l = +4)$	*UNDEF*	$\|+6, +1, +2, -7, \|+3, +4$
UnitProp $(c = [-1, -3, +5], l = +5)$	*UNDEF*	$\|+6, +1, +2, -7, \|+3, +4, +5$
Backtrack $(M \models \neg [-2, -4, -5])$	*UNDEF*	$\|+6, +1, +2, -7, -3$
UnitProp $(c = [-2, +3, +5, -6], l = +5)$	*UNDEF*	$\|+6, +1, +2, -7, -3, +5$
Backtrack $(M \models \neg [-1, +3, -5, -6])$	*UNDEF*	-6

Problemi

Kako otkriti pravi uzrok konflikta?

- Niz koraka nakon pretpostavke $+7$ je pokazao da ni $+3$ ni -3 nisu kompatibilni sa prethodnim pretpostavkama.
- pravilo Backtrack menja uvek samo poslednju pretpostavku.
- Isti niz koraka pokazuje da ni ovaj put ni $+3$ ni -3 nisu kompatibilni sa prethodnim pretpostavkama.
- Pretpostavka $+7$ je potpuno irelevantna za konflikt koji se dogodio i ponavljanje postupka sa -7 je gubitak vremena.

Rešenje: analiza konflikata (eng. conflict analysis) i uvođenje povratnih skokova (eng. backjumping).

Problemi

Kako otkriti pravi uzrok konflikta?

- Niz koraka nakon pretpostavke $+7$ je pokazao da ni $+3$ ni -3 nisu kompatibilni sa prethodnim pretpostavkama.
- pravilo Backtrack menja uvek samo poslednju pretpostavku.
- Isti niz koraka pokazuje da ni ovaj put ni $+3$ ni -3 nisu kompatibilni sa prethodnim pretpostavkama.
- Pretpostavka $+7$ je potpuno irelevantna za konflikt koji se dogodio i ponavljanje postupka sa -7 je gubitak vremena.

Rešenje: analiza konflikata (eng. conflict analysis) i uvođenje povratnih skokova (eng. backjumping).

Primer

Primenjeno pravilo	*satFlag*	*M*
...		
Backtrack $(M \models \neg [-1, 3, -5, -6])$	*UNDEF*	-6
Decide $(l = 1)$	*UNDEF*	$-6, \lvert 1$
UnitProp $(c = [-1, 2],\ l = 2)$	*UNDEF*	$-6, \lvert 1, 2$
Decide $(l = 7)$	*UNDEF*	$-6, \lvert 1, 2, \lvert 7$
Decide $(l = 3)$	*UNDEF*	$-6, \lvert 1, 2, \lvert 7, \lvert 3$
UnitProp $(c = [-3, 4],\ l = 4)$	*UNDEF*	$-6, \lvert 1, 2, \lvert 7, \lvert 3, 4$
UnitProp $(c = [-1, -3, 5],\ l = 5)$	*UNDEF*	$-6, \lvert 1, 2, \lvert 7, \lvert 3, 4, 5$
Backtrack $(M \models \neg [-2, -4, -5])$	*UNDEF*	$-6, \lvert 1, 2, \lvert 7, -3$
Decide $(l = 4)$	*UNDEF*	$-6, \lvert 1, 2, \lvert 7, -3, \lvert 4$
UnitProp $(c = [-2, -4, -5],\ l = -5)$	*UNDEF*	$-6, \lvert 1, 2, \lvert 7, -3, \lvert 4, -5$
$M \not\models \neg F_0$, $(\text{vars } M) = (\text{vars } F_0)$	*SAT*	$-6, \lvert 1, 2, \lvert 7, -3, \lvert 4, -5$

Primer

Primenjeno pravilo	*satFlag*	*M*
...		
Backtrack $(M \models \neg [-1, 3, -5, -6])$	*UNDEF*	-6
Decide $(l = 1)$	*UNDEF*	$-6, \|1$
UnitProp $(c = [-1, 2],\ l = 2)$	*UNDEF*	$-6, \|1, 2$
Decide $(l = 7)$	*UNDEF*	$-6, \|1, 2, \|7$
Decide $(l = 3)$	*UNDEF*	$-6, \|1, 2, \|7, \|3$
UnitProp $(c = [-3, 4],\ l = 4)$	*UNDEF*	$-6, \|1, 2, \|7, \|3, 4$
UnitProp $(c = [-1, -3, 5],\ l = 5)$	*UNDEF*	$-6, \|1, 2, \|7, \|3, 4, 5$
Backtrack $(M \models \neg [-2, -4, -5])$	*UNDEF*	$-6, \|1, 2, \|7, -3$
Decide $(l = 4)$	*UNDEF*	$-6, \|1, 2, \|7, -3, \|4$
UnitProp $(c = [-2, -4, -5],\ l = -5)$	*UNDEF*	$-6, \|1, 2, \|7, -3, \|4, -5$
$M \not\models \neg F_0$, (vars M) = (vars F_0)	*SAT*	$-6, \|1, 2, \|7, -3, \|4, -5$

Problemi – nastavak

Kako da sprečimo da ponovo radimo istu stvar?

- Ista vrsta redundantnosti može da se javi i u nekom kontekstu (npr. sa literalom -6 umesto literala 6).

Rešenje: učenje iz ranijih konflikata (eng. learning).

Problemi – nastavak

Kako da sprečimo da ponovo radimo istu stvar?

- Ista vrsta redundantnosti može da se javi i u nekom kontekstu (npr. sa literalom -6 umesto literala 6).

Rešenje: učenje iz ranijih konflikata (eng. learning).

Analiza konflikata

Primer

Primenjeno pravilo	*satFlag*	*M*
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\|+6$
UnitProp $(c = [+1, -6],\ l = +1)$	*UNDEF*	$\|+6, +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\|+6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\|+6, +1, +2, \|+7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Conflict $(M \models \neg [-2, -4, -5])$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$

Zašto je došlo do konflikta? *Zbog literala $+2$, $+4$ i $+5$.*
A zbog čega je prisutan literal $+5$? *Zbog literala $+1$ i $+3$.*
A zbog čega je prisutan literal $+4$? *Zbog literala $+3$.*
Dakle, uz literale $+1$ i $+2$, ne ide literal $+3$.

Analiza konflikata

Primer

Primenjeno pravilo	*satFlag*	*M*
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\|+6$
UnitProp $(c = [+1, -6],\ l = +1)$	*UNDEF*	$\|+6, +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\|+6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\|+6, +1, +2, \|+7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Conflict $(M \models \neg [-2, -4, -5])$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$

Zašto je došlo do konflikta? *Zbog literala $+2$, $+4$ i $+5$.*

A zbog čega je prisutan literal $+5$? *Zbog literala $+1$ i $+3$.*

A zbog čega je prisutan literal $+4$? *Zbog literala $+3$.*

Dakle, uz literale $+1$ i $+2$, ne ide literal $+3$.

Analiza konflikata

Primer

Primenjeno pravilo	*satFlag*	*M*
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\|+6$
UnitProp $(c = [+1, -6],\ l = +1)$	*UNDEF*	$\|+6, +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\|+6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\|+6, +1, +2, \|+7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Conflict $(M \models \neg [-2, -4, -5])$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$

$$\{+2, +4, +5\}$$

Zašto je došlo do konflikta? *Zbog literala* $+2$, $+4$ *i* $+5$.

A zbog čega je prisutan literal $+5$*?* *Zbog literala* $+1$ *i* $+3$.

A zbog čega je prisutan literal $+4$*?* *Zbog literala* $+3$.

Dakle, uz literale $+1$ *i* $+2$*, ne ide literal* $+3$.

Analiza konflikata

Primer

Primenjeno pravilo	*satFlag*	*M*
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\|+6$
UnitProp $(c = [+1, -6],\ l = +1)$	*UNDEF*	$\|+6, +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\|+6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\|+6, +1, +2, \|+7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Conflict $(M \models \neg [-2, -4, -5])$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$

$$\{+2, +4, +5\}$$

Zašto je došlo do konflikta? *Zbog literala* $+2$, $+4$ *i* $+5$.

A zbog čega je prisutan literal $+5$*?* *Zbog literala* $+1$ *i* $+3$.

A zbog čega je prisutan literal $+4$*?* *Zbog literala* $+3$.

Dakle, uz literale $+1$ *i* $+2$*, ne ide literal* $+3$.

Analiza konflikata

Primer

Primenjeno pravilo	*satFlag*	*M*
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\|+6$
UnitProp $(c = [+1, -6],\ l = +1)$	*UNDEF*	$\|+6, +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\|+6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\|+6, +1, +2, \|+7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Conflict $(M \models \neg [-2, -4, -5])$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$

$$\{+2, +4, +5\}$$

Zašto je došlo do konflikta? *Zbog literala* $+2$, $+4$ *i* $+5$.

A zbog čega je prisutan literal $+5$*?* *Zbog literala* $+1$ *i* $+3$.

A zbog čega je prisutan literal $+4$*?* *Zbog literala* $+3$.

Dakle, uz literale $+1$ *i* $+2$*, ne ide literal* $+3$.

Analiza konflikata

Primer

Primenjeno pravilo	*satFlag*	*M*
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\|+6$
UnitProp $(c = [+1, -6],\ l = +1)$	*UNDEF*	$\|+6, +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\|+6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\|+6, +1, +2, \|+7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Conflict $(M \models \neg [-2, -4, -5])$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$

$$\{+1, +2, +3, +4\}$$

Zašto je došlo do konflikta? *Zbog literala* $+2$, $+4$ *i* $+5$.

A zbog čega je prisutan literal $+5$*?* *Zbog literala* $+1$ *i* $+3$.

A zbog čega je prisutan literal $+4$*?* *Zbog literala* $+3$.

Dakle, uz literale $+1$ *i* $+2$*, ne ide literal* $+3$.

Analiza konflikata

Primer

Primenjeno pravilo	*satFlag*	*M*
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\|+6$
UnitProp $(c = [+1, -6],\ l = +1)$	*UNDEF*	$\|+6, +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\|+6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\|+6, +1, +2, \|+7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Conflict $(M \models \neg [-2, -4, -5])$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$

$$\{+1, +2, +3, +4\}$$

Zašto je došlo do konflikta? *Zbog literala* $+2$, $+4$ *i* $+5$.

A zbog čega je prisutan literal $+5$*?* *Zbog literala* $+1$ *i* $+3$.

A zbog čega je prisutan literal $+4$*?* *Zbog literala* $+3$.

Dakle, uz literale $+1$ *i* $+2$*, ne ide literal* $+3$.

Analiza konflikata

Primer

Primenjeno pravilo	*satFlag*	*M*
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\|+6$
UnitProp $(c = [+1, -6],\ l = +1)$	*UNDEF*	$\|+6, +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\|+6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\|+6, +1, +2, \|+7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Conflict $(M \models \neg [-2, -4, -5])$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$

$$\{+1, +2, +3, +4\}$$

Zašto je došlo do konflikta? *Zbog literala* $+2$, $+4$ *i* $+5$.

A zbog čega je prisutan literal $+5$*?* *Zbog literala* $+1$ *i* $+3$.

A zbog čega je prisutan literal $+4$*?* *Zbog literala* $+3$.

Dakle, uz literale $+1$ *i* $+2$*, ne ide literal* $+3$.

Analiza konflikata

Primer

Primenjeno pravilo	*satFlag*	*M*
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\|+6$
UnitProp $(c = [+1, -6],\ l = +1)$	*UNDEF*	$\|+6, +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\|+6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\|+6, +1, +2, \|+7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Conflict $(M \models \neg [-2, -4, -5])$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$

$$\{+1, +2, +3\}$$

Zašto je došlo do konflikta? *Zbog literala* $+2$, $+4$ *i* $+5$.

A zbog čega je prisutan literal $+5$*?* *Zbog literala* $+1$ *i* $+3$.

A zbog čega je prisutan literal $+4$*?* *Zbog literala* $+3$.

Dakle, uz literale $+1$ *i* $+2$*, ne ide literal* $+3$.

Analiza konflikata

Primer

Primenjeno pravilo	*satFlag*	*M*
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\|+6$
UnitProp $(c = [+1, -6],\ l = +1)$	*UNDEF*	$\|+6, +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\|+6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\|+6, +1, +2, \|+7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Conflict $(M \models \neg [-2, -4, -5])$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$

$$\{+1, +2, +3\}$$

Zašto je došlo do konflikta? *Zbog literala* $+2$, $+4$ *i* $+5$.

A zbog čega je prisutan literal $+5$*?* *Zbog literala* $+1$ *i* $+3$.

A zbog čega je prisutan literal $+4$*?* *Zbog literala* $+3$.

Dakle, uz literale $+1$ *i* $+2$*, ne ide literal* $+3$.

Analiza konflikata

Primer

Primenjeno pravilo	*satFlag*	*M*
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\|+6$
UnitProp $(c = [+1, -6],\ l = +1)$	*UNDEF*	$\|+6, +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\|+6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\|+6, +1, +2, \|+7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Conflict $(M \models \neg[-2, -4, -5])$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Backjump $(C = [-1, -2, -3])$	*UNDEF*	$\|+6, +1, +2, -3$

$$\{+1, +2, +3\}$$

Zašto je došlo do konflikta? *Zbog literala* $+2$, $+4$ *i* $+5$.

A zbog čega je prisutan literal $+5$*?* *Zbog literala* $+1$ *i* $+3$.

A zbog čega je prisutan literal $+4$*?* *Zbog literala* $+3$.

Dakle, uz literale $+1$ *i* $+2$*, ne ide literal* $+3$.

Analiza konflikata kao rezolucija

Stablo rezolucije

$$\dfrac{\dfrac{[-2,-4,-5] \qquad [-1,-3,5]}{[-1,-2,-3,-4]} \qquad [-3,4]}{[-1,-2,-3]}$$

Dobijena klauza povratnog skoka je semantička posledica polaznog skupa klauza.

Analiza konflikata kao rezolucija

Stablo rezolucije

$$\dfrac{\dfrac{[-2,-4,-5] \qquad [-1,-3,5]}{[-1,-2,-3,-4]} \qquad [-3,4]}{[-1,-2,-3]}$$

Dobijena klauza povratnog skoka je semantička posledica polaznog skupa klauza.

Analiza konflikata kao rezolucija

Stablo rezolucije

$$\dfrac{\dfrac{[-2,-4,-5] \qquad [-1,-3,5]}{[-1,-2,-3,-4]} \qquad [-3,4]}{[-1,-2,-3]}$$

Dobijena klauza povratnog skoka je semantička posledica polaznog skupa klauza.

Analiza konflikata kao rezolucija

Stablo rezolucije

$$\dfrac{\dfrac{[-2,-4,-5] \qquad [-1,-3,5]}{[-1,-2,-3,-4]} \qquad [-3,4]}{[-1,-2,-3]}$$

Dobijena klauza povratnog skoka je semantička posledica polaznog skupa klauza.

Analiza konflikata kao rezolucija

Stablo rezolucije

$$\dfrac{\dfrac{[-2,-4,-5] \qquad [-1,-3,5]}{[-1,-2,-3,-4]} \qquad [-3,4]}{[-1,-2,-3]}$$

Dobijena klauza povratnog skoka je semantička posledica polaznog skupa klauza.

Sistem sa analizom konflikata

Stanje rešavača:

- (M, C)
- M - parcijalna valuacija
- C - konfliktna klauza (ili *no_cflt* ako nema konflikta)

Pravila:

Decide:
$$\frac{C = \mathit{no\_cflt} \quad l \in F \quad l, \bar{l} \notin M}{M := M \,|\, l}$$

UnitPropagate:
$$\frac{C = \mathit{no\_cflt} \quad l \vee l_1 \vee \ldots \vee l_k \in F \qquad \bar{l}_1, \ldots, \bar{l}_k \in M \qquad l, \bar{l} \notin M}{M := M\ l}$$

Conflict:
$$\frac{C = \mathit{no\_cflt} \quad l_1 \vee \ldots l_k \in F \qquad \bar{l}_1, \ldots, \bar{l}_k \in M}{C := l_1 \vee \ldots \vee l_k}$$

Explain:
$$\frac{\bar{l} \in C \qquad l \vee l_1 \vee \ldots \vee l_k \in F \qquad \bar{l}_1, \ldots, \bar{l}_k \prec^M l}{C := (C \setminus \bar{l}) \vee l_1 \vee \ldots \vee l_k}$$

Backjump:
$$\frac{C = l \vee l_1 \vee \ldots \vee l_k \qquad \text{level } \bar{l} > m \geq \text{level } \bar{l}_i}{M := (\text{prefixToLevel } m\ M)\ l \quad C := \mathit{no\_cflt}}$$

Učenje

Sistem sa učenjem

Stanje rešavača:

- (M, F, C)
- M - parcijalna valuacija
- F - formula koja se vremenom proširuje (učenjem klauza)
- C - konfliktna klauza (ili *no_cflt* ako nema konflikta)

Dodatno pravilo:

Learn:

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

NAPOMENA: Obično se uče isključivo klauze povratnog skoka.

Primer

$F = [[-1, +2], [-3, +4], [-1, -3, +5], [-2, -4, -5], [-2, +3, +5, -6], [-1, +3, -5, -6], [+1, -6], [+1, +7]\ ]$.

Primenjeno pravilo	*satFlag*	M/C
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\|+6$
UnitProp $(c = [+1, -6],\ l = +1)$	*UNDEF*	$\|+6, +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\|+6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\|+6, +1, +2, \|+7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Conflict $(M \models \neg [-2, -4, -5])$	*UNDEF*	$[-2, -4, -5]$
Explain $(l = +5,\ c = [-1, -3, +5])$	*UNDEF*	$[-1, -2, -3, -4]$
Explain $(l = 4,\ c = [-3, +4])$	*UNDEF*	$[-1, -2, -3]$
Learn *(C = [-1, -2, -3])*	*UNDEF*	$[-1, -2, -3]$
Backjump $(C = [-1, -2, -3])$	*UNDEF*	$\|+6, +1, +2, -3$
UnitProp $(c = [-2, +3, +5, -6],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, -3, +5$
Conflict $(M \models \neg [-1, +3, -5, -6])$	*UNDEF*	$[-1, +3, -5, -6]$
Explain $(l = +5,\ c = [-2, +3, +5, -6])$	*UNDEF*	$[-1, -2, +3, -6]$
Explain $(l = -3,\ c = [-1, -2, -3])$	*UNDEF*	$[-1, -2, -6]$
Explain $(l = +2,\ c = [-1, +2])$	*UNDEF*	$[-1, -6]$
Explain $(l = +1,\ c = [+1, -6])$	*UNDEF*	$[-6]$
Learn *(C = [-6])*	*UNDEF*	$[-6]$
Backjump $(C = [-6])$	*UNDEF*	-6

Primer

$F = [[-1, +2], [-3, +4], [-1, -3, +5], [-2, -4, -5], [-2, +3, +5, -6], [-1, +3, -5, -6], [+1, -6], [+1, +7]\ ]$.

Primenjeno pravilo	*satFlag*	M/C
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\|+6$
UnitProp $(c = [+1, -6],\ l = +1)$	*UNDEF*	$\|+6, +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\|+6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\|+6, +1, +2, \|+7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Conflict $(M \models \neg [-2, -4, -5])$	*UNDEF*	$[-2, -4, -5]$
Explain $(l = +5,\ c = [-1, -3, +5])$	*UNDEF*	$[-1, -2, -3, -4]$
Explain $(l = 4,\ c = [-3, +4])$	*UNDEF*	$[-1, -2, -3]$
Learn *(C = [-1, -2, -3])*	*UNDEF*	$[-1, -2, -3]$
Backjump $(C = [-1, -2, -3])$	*UNDEF*	$\|+6, +1, +2, -3$
UnitProp $(c = [-2, +3, +5, -6],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, -3, +5$
Conflict $(M \models \neg [-1, +3, -5, -6])$	*UNDEF*	$[-1, +3, -5, -6]$
Explain $(l = +5,\ c = [-2, +3, +5, -6])$	*UNDEF*	$[-1, -2, +3, -6]$
Explain $(l = -3,\ c = [-1, -2, -3])$	*UNDEF*	$[-1, -2, -6]$
Explain $(l = +2,\ c = [-1, +2])$	*UNDEF*	$[-1, -6]$
Explain $(l = +1,\ c = [+1, -6])$	*UNDEF*	$[-6]$
Learn *(C = [-6])*	*UNDEF*	$[-6]$
Backjump $(C = [-6])$	*UNDEF*	-6

Primer

$F = [[-1, +2], [-3, +4], [-1, -3, +5], [-2, -4, -5], [-2, +3, +5, -6], [-1, +3, -5, -6], [+1, -6], [+1, +7]\ ]$.

Primenjeno pravilo	*satFlag*	*M/C*
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\|+6$
UnitProp $(c = [+1, -6],\ l = +1)$	*UNDEF*	$\|+6, +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\|+6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\|+6, +1, +2, \|+7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Conflict $(M \models \neg [-2, -4, -5])$	*UNDEF*	$[-2, -4, -5]$
Explain $(l = +5,\ c = [-1, -3, +5])$	*UNDEF*	$[-1, -2, -3, -4]$
Explain $(l = 4,\ c = [-3, +4])$	*UNDEF*	$[-1, -2, -3]$
Learn *(C = [-1, -2, -3])*	*UNDEF*	$[-1, -2, -3]$
Backjump $(C = [-1, -2, -3])$	*UNDEF*	$\|+6, +1, +2, -3$
UnitProp $(c = [-2, +3, +5, -6],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, -3, +5$
Conflict $(M \models \neg [-1, +3, -5, -6])$	*UNDEF*	$[-1, +3, -5, -6]$
Explain $(l = +5,\ c = [-2, +3, +5, -6])$	*UNDEF*	$[-1, -2, +3, -6]$
Explain $(l = -3,\ c = [-1, -2, -3])$	*UNDEF*	$[-1, -2, -6]$
Explain $(l = +2,\ c = [-1, +2])$	*UNDEF*	$[-1, -6]$
Explain $(l = +1,\ c = [+1, -6])$	*UNDEF*	$[-6]$
Learn *(C = [-6])*	*UNDEF*	$[-6]$
Backjump $(C = [-6])$	*UNDEF*	-6

Primer

$F = [[-1, +2], [-3, +4], [-1, -3, +5], [-2, -4, -5], [-2, +3, +5, -6], [-1, +3, -5, -6], [+1, -6], [+1, +7]\ ]$.

Primenjeno pravilo	*satFlag*	M/C
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\|+6$
UnitProp $(c = [+1, -6],\ l = +1)$	*UNDEF*	$\|+6, +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\|+6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\|+6, +1, +2, \|+7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Conflict $(M \models \neg [-2, -4, -5])$	*UNDEF*	$[-2, -4, -5]$
Explain $(l = +5,\ c = [-1, -3, +5])$	*UNDEF*	$[-1, -2, -3, -4]$
Explain $(l = 4,\ c = [-3, +4])$	*UNDEF*	$[-1, -2, -3]$
Learn *(C = [-1, -2, -3])*	*UNDEF*	$[-1, -2, -3]$
Backjump $(C = [-1, -2, -3])$	*UNDEF*	$\|+6, +1, +2, -3$
UnitProp $(c = [-2, +3, +5, -6],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, -3, +5$
Conflict $(M \models \neg [-1, +3, -5, -6])$	*UNDEF*	$[-1, +3, -5, -6]$
Explain $(l = +5,\ c = [-2, +3, +5, -6])$	*UNDEF*	$[-1, -2, +3, -6]$
Explain $(l = -3,\ c = [-1, -2, -3])$	*UNDEF*	$[-1, -2, -6]$
Explain $(l = +2,\ c = [-1, +2])$	*UNDEF*	$[-1, -6]$
Explain $(l = +1,\ c = [+1, -6])$	*UNDEF*	$[-6]$
Learn *(C = [-6])*	*UNDEF*	$[-6]$
Backjump $(C = [-6])$	*UNDEF*	-6

Primer

$F = [[-1, +2], [-3, +4], [-1, -3, +5], [-2, -4, -5], [-2, +3, +5, -6], [-1, +3, -5, -6], [+1, -6], [+1, +7], [-1, -2, -3]]$.

Primenjeno pravilo	*satFlag*	*M/C*
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\|+6$
UnitProp $(c = [+1, -6], l = +1)$	*UNDEF*	$\|+6, +1$
UnitProp $(c = [-1, +2], l = +2)$	*UNDEF*	$\|+6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\|+6, +1, +2, \|+7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3$
UnitProp $(c = [-3, +4], l = +4)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4$
UnitProp $(c = [-1, -3, +5], l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Conflict $(M \models \neg [-2, -4, -5])$	*UNDEF*	$[-2, -4, -5]$
Explain $(l = +5, c = [-1, -3, +5])$	*UNDEF*	$[-1, -2, -3, -4]$
Explain $(l = 4, c = [-3, +4])$	*UNDEF*	$[-1, -2, -3]$
Learn *(C = [-1, -2, -3])*	*UNDEF*	$[-1, -2, -3]$
Backjump $(C = [-1, -2, -3])$	*UNDEF*	$\|+6, +1, +2, -3$
UnitProp $(c = [-2, +3, +5, -6], l = +5)$	*UNDEF*	$\|+6, +1, +2, -3, +5$
Conflict $(M \models \neg [-1, +3, -5, -6])$	*UNDEF*	$[-1, +3, -5, -6]$
Explain $(l = +5, c = [-2, +3, +5, -6])$	*UNDEF*	$[-1, -2, +3, -6]$
Explain $(l = -3, c = [-1, -2, -3])$	*UNDEF*	$[-1, -2, -6]$
Explain $(l = +2, c = [-1, +2])$	*UNDEF*	$[-1, -6]$
Explain $(l = +1, c = [+1, -6])$	*UNDEF*	$[-6]$
Learn *(C = [-6])*	*UNDEF*	$[-6]$
Backjump $(C = [-6])$	*UNDEF*	-6

Primer

$F = [[-1, +2], [-3, +4], [-1, -3, +5], [-2, -4, -5], [-2, +3, +5, -6], [-1, +3, -5, -6], [+1, -6], [+1, +7], [-1, -2, -3]\ ]$.

Primenjeno pravilo	*satFlag*	M/C
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\|+6$
UnitProp $(c = [+1, -6],\ l = +1)$	*UNDEF*	$\|+6, +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\|+6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\|+6, +1, +2, \|+7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Conflict $(M \models \neg [-2, -4, -5])$	*UNDEF*	$[-2, -4, -5]$
Explain $(l = +5,\ c = [-1, -3, +5])$	*UNDEF*	$[-1, -2, -3, -4]$
Explain $(l = 4,\ c = [-3, +4])$	*UNDEF*	$[-1, -2, -3]$
Learn *(C = [-1, -2, -3])*	*UNDEF*	$[-1, -2, -3]$
Backjump $(C = [-1, -2, -3])$	*UNDEF*	$\|+6, +1, +2, -3$
UnitProp $(c = [-2, +3, +5, -6],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, -3, +5$
Conflict $(M \models \neg [-1, +3, -5, -6])$	*UNDEF*	$[-1, +3, -5, -6]$
Explain $(l = +5,\ c = [-2, +3, +5, -6])$	*UNDEF*	$[-1, -2, +3, -6]$
Explain $(l = -3,\ c = [-1, -2, -3])$	*UNDEF*	$[-1, -2, -6]$
Explain $(l = +2,\ c = [-1, +2])$	*UNDEF*	$[-1, -6]$
Explain $(l = +1,\ c = [+1, -6])$	*UNDEF*	$[-6]$
Learn *(C = [-6])*	*UNDEF*	$[-6]$
Backjump $(C = [-6])$	*UNDEF*	-6

Primer

$F = [[-1, +2], [-3, +4], [-1, -3, +5], [-2, -4, -5], [-2, +3, +5, -6], [-1, +3, -5, -6], [+1, -6], [+1, +7], [-1, -2, -3]\ ]$.

Primenjeno pravilo	*satFlag*	M/C
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\|+6$
UnitProp $(c = [+1, -6],\ l = +1)$	*UNDEF*	$\|+6, +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\|+6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\|+6, +1, +2, \|+7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Conflict $(M \models \neg [-2, -4, -5])$	*UNDEF*	$[-2, -4, -5]$
Explain $(l = +5,\ c = [-1, -3, +5])$	*UNDEF*	$[-1, -2, -3, -4]$
Explain $(l = 4,\ c = [-3, +4])$	*UNDEF*	$[-1, -2, -3]$
Learn *(C = [-1, -2, -3])*	*UNDEF*	$[-1, -2, -3]$
Backjump $(C = [-1, -2, -3])$	*UNDEF*	$\|+6, +1, +2, -3$
UnitProp $(c = [-2, +3, +5, -6],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, -3, +5$
Conflict $(M \models \neg [-1, +3, -5, -6])$	*UNDEF*	$[-1, +3, -5, -6]$
Explain $(l = +5,\ c = [-2, +3, +5, -6])$	*UNDEF*	$[-1, -2, +3, -6]$
Explain $(l = -3,\ c = [-1, -2, -3])$	*UNDEF*	$[-1, -2, -6]$
Explain $(l = +2,\ c = [-1, +2])$	*UNDEF*	$[-1, -6]$
Explain $(l = +1,\ c = [+1, -6])$	*UNDEF*	$[-6]$
Learn *(C = [-6])*	*UNDEF*	$[-6]$
Backjump $(C = [-6])$	*UNDEF*	-6

Primer

$F = [[-1, +2], [-3, +4], [-1, -3, +5], [-2, -4, -5], [-2, +3, +5, -6], [-1, +3, -5, -6], [+1, -6], [+1, +7], [-1, -2, -3]\ ]$.

Primenjeno pravilo	*satFlag*	*M/C*
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\mid +6$
UnitProp $(c = [+1, -6],\ l = +1)$	*UNDEF*	$\mid +6, +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\mid +6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\mid +6, +1, +2, \mid +7$
Decide $(l = +3)$	*UNDEF*	$\mid +6, +1, +2, \mid +7, \mid +3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\mid +6, +1, +2, \mid +7, \mid +3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\mid +6, +1, +2, \mid +7, \mid +3, +4, +5$
Conflict $(M \models \neg [-2, -4, -5])$	*UNDEF*	$[-2, -4, -5]$
Explain $(l = +5,\ c = [-1, -3, +5])$	*UNDEF*	$[-1, -2, -3, -4]$
Explain $(l = 4,\ c = [-3, +4])$	*UNDEF*	$[-1, -2, -3]$
Learn *(C = [-1, -2, -3])*	*UNDEF*	$[-1, -2, -3]$
Backjump $(C = [-1, -2, -3])$	*UNDEF*	$\mid +6, +1, +2, -3$
UnitProp $(c = [-2, +3, +5, -6],\ l = +5)$	*UNDEF*	$\mid +6, +1, +2, -3, +5$
Conflict $(M \models \neg [-1, +3, -5, -6])$	*UNDEF*	$[-1, +3, -5, -6]$
Explain $(l = +5,\ c = [-2, +3, +5, -6])$	*UNDEF*	$[-1, -2, +3, -6]$
Explain $(l = -3,\ c = [-1, -2, -3])$	*UNDEF*	$[-1, -2, -6]$
Explain $(l = +2,\ c = [-1, +2])$	*UNDEF*	$[-1, -6]$
Explain $(l = +1,\ c = [+1, -6])$	*UNDEF*	$[-6]$
Learn *(C = [-6])*	*UNDEF*	$[-6]$
Backjump $(C = [-6])$	*UNDEF*	-6

Primer

$F = [[-1, +2], [-3, +4], [-1, -3, +5], [-2, -4, -5], [-2, +3, +5, -6], [-1, +3, -5, -6], [+1, -6], [+1, +7], [-1, -2, -3]\ ]$.

Primenjeno pravilo	*satFlag*	M/C
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\|+6$
UnitProp $(c = [+1, -6],\ l = +1)$	*UNDEF*	$\|+6, +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\|+6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\|+6, +1, +2, \|+7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Conflict $(M \models \neg [-2, -4, -5])$	*UNDEF*	$[-2, -4, -5]$
Explain $(l = +5,\ c = [-1, -3, +5])$	*UNDEF*	$[-1, -2, -3, -4]$
Explain $(l = 4,\ c = [-3, +4])$	*UNDEF*	$[-1, -2, -3]$
Learn *(C = [-1, -2, -3])*	*UNDEF*	$[-1, -2, -3]$
Backjump $(C = [-1, -2, -3])$	*UNDEF*	$\|+6, +1, +2, -3$
UnitProp $(c = [-2, +3, +5, -6],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, -3, +5$
Conflict $(M \models \neg [-1, +3, -5, -6])$	*UNDEF*	$[-1, +3, -5, -6]$
Explain $(l = +5,\ c = [-2, +3, +5, -6])$	*UNDEF*	$[-1, -2, +3, -6]$
Explain $(l = -3,\ c = [-1, -2, -3])$	*UNDEF*	$[-1, -2, -6]$
Explain $(l = +2,\ c = [-1, +2])$	*UNDEF*	$[-1, -6]$
Explain $(l = +1,\ c = [+1, -6])$	*UNDEF*	$[-6]$
Learn *(C = [-6])*	*UNDEF*	$[-6]$
Backjump $(C = [-6])$	*UNDEF*	-6

Primer

$F = [[-1, +2], [-3, +4], [-1, -3, +5], [-2, -4, -5], [-2, +3, +5, -6], [-1, +3, -5, -6], [+1, -6], [+1, +7], [-1, -2, -3]\ ]$.

Primenjeno pravilo	*satFlag*	M/C
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\|+6$
UnitProp $(c = [+1, -6],\ l = +1)$	*UNDEF*	$\|+6, +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\|+6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\|+6, +1, +2, \|+7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Conflict $(M \models \neg [-2, -4, -5])$	*UNDEF*	$[-2, -4, -5]$
Explain $(l = +5,\ c = [-1, -3, +5])$	*UNDEF*	$[-1, -2, -3, -4]$
Explain $(l = 4,\ c = [-3, +4])$	*UNDEF*	$[-1, -2, -3]$
Learn *(C = [-1, -2, -3])*	*UNDEF*	$[-1, -2, -3]$
Backjump $(C = [-1, -2, -3])$	*UNDEF*	$\|+6, +1, +2, -3$
UnitProp $(c = [-2, +3, +5, -6],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, -3, +5$
Conflict $(M \models \neg [-1, +3, -5, -6])$	*UNDEF*	$[-1, +3, -5, -6]$
Explain $(l = +5,\ c = [-2, +3, +5, -6])$	*UNDEF*	$[-1, -2, +3, -6]$
Explain $(l = -3,\ c = [-1, -2, -3])$	*UNDEF*	$[-1, -2, -6]$
Explain $(l = +2,\ c = [-1, +2])$	*UNDEF*	$[-1, -6]$
Explain $(l = +1,\ c = [+1, -6])$	*UNDEF*	$[-6]$
Learn *(C = [-6])*	*UNDEF*	$[-6]$
Backjump $(C = [-6])$	*UNDEF*	-6

Primer

$F = [[-1, +2], [-3, +4], [-1, -3, +5], [-2, -4, -5], [-2, +3, +5, -6], [-1, +3, -5, -6], [+1, -6], [+1, +7], [-1, -2, -3]\ ]$.

Primenjeno pravilo	*satFlag*	M/C
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\|+6$
UnitProp $(c = [+1, -6],\ l = +1)$	*UNDEF*	$\|+6, +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\|+6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\|+6, +1, +2, \|+7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Conflict $(M \models \neg [-2, -4, -5])$	*UNDEF*	$[-2, -4, -5]$
Explain $(l = +5,\ c = [-1, -3, +5])$	*UNDEF*	$[-1, -2, -3, -4]$
Explain $(l = 4,\ c = [-3, +4])$	*UNDEF*	$[-1, -2, -3]$
Learn *(C = [-1, -2, -3])*	*UNDEF*	$[-1, -2, -3]$
Backjump $(C = [-1, -2, -3])$	*UNDEF*	$\|+6, +1, +2, -3$
UnitProp $(c = [-2, +3, +5, -6],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, -3, +5$
Conflict $(M \models \neg [-1, +3, -5, -6])$	*UNDEF*	$[-1, +3, -5, -6]$
Explain $(l = +5,\ c = [-2, +3, +5, -6])$	*UNDEF*	$[-1, -2, +3, -6]$
Explain $(l = -3,\ c = [-1, -2, -3])$	*UNDEF*	$[-1, -2, -6]$
Explain $(l = +2,\ c = [-1, +2])$	*UNDEF*	$[-1, -6]$
Explain $(l = +1,\ c = [+1, -6])$	*UNDEF*	$[-6]$
Learn *(C = [-6])*	*UNDEF*	$[-6]$
Backjump $(C = [-6])$	*UNDEF*	-6

Primer

$F = [[-1, +2], [-3, +4], [-1, -3, +5], [-2, -4, -5], [-2, +3, +5, -6], [-1, +3, -5, -6], [+1, -6], [+1, +7], [-1, -2, -3]\ ]$.

Primenjeno pravilo	*satFlag*	*M/C*
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\|+6$
UnitProp $(c = [+1, -6],\ l = +1)$	*UNDEF*	$\|+6, +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\|+6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\|+6, +1, +2, \|+7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Conflict $(M \models \neg [-2, -4, -5])$	*UNDEF*	$[-2, -4, -5]$
Explain $(l = +5,\ c = [-1, -3, +5])$	*UNDEF*	$[-1, -2, -3, -4]$
Explain $(l = 4,\ c = [-3, +4])$	*UNDEF*	$[-1, -2, -3]$
Learn *(C = [-1, -2, -3])*	*UNDEF*	$[-1, -2, -3]$
Backjump $(C = [-1, -2, -3])$	*UNDEF*	$\|+6, +1, +2, -3$
UnitProp $(c = [-2, +3, +5, -6],\ l = +5)$	*UNDEF*	$\|+6, +1, +2, -3, +5$
Conflict $(M \models \neg [-1, +3, -5, -6])$	*UNDEF*	$[-1, +3, -5, -6]$
Explain $(l = +5,\ c = [-2, +3, +5, -6])$	*UNDEF*	$[-1, -2, +3, -6]$
Explain $(l = -3,\ c = [-1, -2, -3])$	*UNDEF*	$[-1, -2, -6]$
Explain $(l = +2,\ c = [-1, +2])$	*UNDEF*	$[-1, -6]$
Explain $(l = +1,\ c = [+1, -6])$	*UNDEF*	$[-6]$
Learn *(C = [-6])*	*UNDEF*	$[-6]$
Backjump $(C = [-6])$	*UNDEF*	-6

Primer

$F = [[-1, +2], [-3, +4], [-1, -3, +5], [-2, -4, -5], [-2, +3, +5, -6], [-1, +3, -5, -6], [+1, -6], [+1, +7], [-1, -2, -3]$, $[-6]$].

Primenjeno pravilo	*satFlag*	*M/C*
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\|+6$
UnitProp $(c = [+1, -6], l = +1)$	*UNDEF*	$\|+6, +1$
UnitProp $(c = [-1, +2], l = +2)$	*UNDEF*	$\|+6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\|+6, +1, +2, \|+7$
Decide $(l = +3)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3$
UnitProp $(c = [-3, +4], l = +4)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4$
UnitProp $(c = [-1, -3, +5], l = +5)$	*UNDEF*	$\|+6, +1, +2, \|+7, \|+3, +4, +5$
Conflict $(M \models \neg [-2, -4, -5])$	*UNDEF*	$[-2, -4, -5]$
Explain $(l = +5, c = [-1, -3, +5])$	*UNDEF*	$[-1, -2, -3, -4]$
Explain $(l = 4, c = [-3, +4])$	*UNDEF*	$[-1, -2, -3]$
Learn *(C = [-1, -2, -3])*	*UNDEF*	$[-1, -2, -3]$
Backjump $(C = [-1, -2, -3])$	*UNDEF*	$\|+6, +1, +2, -3$
UnitProp $(c = [-2, +3, +5, -6], l = +5)$	*UNDEF*	$\|+6, +1, +2, -3, +5$
Conflict $(M \models \neg [-1, +3, -5, -6])$	*UNDEF*	$[-1, +3, -5, -6]$
Explain $(l = +5, c = [-2, +3, +5, -6])$	*UNDEF*	$[-1, -2, +3, -6]$
Explain $(l = -3, c = [-1, -2, -3])$	*UNDEF*	$[-1, -2, -6]$
Explain $(l = +2, c = [-1, +2])$	*UNDEF*	$[-1, -6]$
Explain $(l = +1, c = [+1, -6])$	*UNDEF*	$[-6]$
Learn *(C = [-6])*	*UNDEF*	$[-6]$
Backjump $(C = [-6])$	*UNDEF*	-6

Primer

$F = [[-1, +2], [-3, +4], [-1, -3, +5], [-2, -4, -5], [-2, +3, +5, -6], [-1, +3, -5, -6], [+1, -6], [+1, +7], [-1, -2, -3]\ , [-6]\ ]$.

Primenjeno pravilo	*satFlag*	*M/C*
	UNDEF	
Decide $(l = +6)$	*UNDEF*	$\mid +6$
UnitProp $(c = [+1, -6],\ l = +1)$	*UNDEF*	$\mid +6, +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$\mid +6, +1, +2$
Decide $(l = +7)$	*UNDEF*	$\mid +6, +1, +2, \mid +7$
Decide $(l = +3)$	*UNDEF*	$\mid +6, +1, +2, \mid +7, \mid +3$
UnitProp $(c = [-3, +4],\ l = +4)$	*UNDEF*	$\mid +6, +1, +2, \mid +7, \mid +3, +4$
UnitProp $(c = [-1, -3, +5],\ l = +5)$	*UNDEF*	$\mid +6, +1, +2, \mid +7, \mid +3, +4, +5$
Conflict $(M \models \neg [-2, -4, -5])$	*UNDEF*	$[-2, -4, -5]$
Explain $(l = +5,\ c = [-1, -3, +5])$	*UNDEF*	$[-1, -2, -3, -4]$
Explain $(l = 4,\ c = [-3, +4])$	*UNDEF*	$[-1, -2, -3]$
Learn *(C = [-1, -2, -3])*	*UNDEF*	$[-1, -2, -3]$
Backjump $(C = [-1, -2, -3])$	*UNDEF*	$\mid +6, +1, +2, -3$
UnitProp $(c = [-2, +3, +5, -6],\ l = +5)$	*UNDEF*	$\mid +6, +1, +2, -3, +5$
Conflict $(M \models \neg [-1, +3, -5, -6])$	*UNDEF*	$[-1, +3, -5, -6]$
Explain $(l = +5,\ c = [-2, +3, +5, -6])$	*UNDEF*	$[-1, -2, +3, -6]$
Explain $(l = -3,\ c = [-1, -2, -3])$	*UNDEF*	$[-1, -2, -6]$
Explain $(l = +2,\ c = [-1, +2])$	*UNDEF*	$[-1, -6]$
Explain $(l = +1,\ c = [+1, -6])$	*UNDEF*	$[-6]$
Learn *(C = [-6])*	*UNDEF*	$[-6]$
Backjump $(C = [-6])$	*UNDEF*	-6

Primer

$F = [[-1, +2], [-3, +4], [-1, -3, +5], [-2, -4, -5], [-2, +3, +5, -6], [-1, +3, -5, -6], [+1, -6], [+1, +7]\ , [-1, -2, -3]\ , [-6]\ ]$.

Primenjeno pravilo	*satFlag*	M/C
Backjump $(C = [-6])$	*UNDEF*	-6
Decide $(l = +1)$	*UNDEF*	$-6, \vert +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$-6, \vert +1, +2$
UnitProp $(c = [-1, -2, -3],\ l = -3)$	*UNDEF*	$-6, \vert +1, +2, -3$
Decide $(l = +4)$	*UNDEF*	$-6, \vert +1, +2, -3, \vert 4$
UnitProp $(c = [-2, -4, -5],\ l = -5)$	*UNDEF*	$-6, \vert +1, +2, -3, \vert 4, -5$
Decide $(l = +7)$	*UNDEF*	$-6, \vert +1, +2, -3, \vert 4, -5, \vert +7$
$M \not\models \neg F$, $(\text{vars } M) = (\text{vars } F)$	*SAT*	$-6, \vert +1, +2, -3, \vert 4, -5, \vert +7$

Primer

$F = [[-1, +2], [-3, +4], [-1, -3, +5], [-2, -4, -5], [-2, +3, +5, -6], [-1, +3, -5, -6], [+1, -6], [+1, +7]\ , [-1, -2, -3]\ , [-6]\ ]$.

Primenjeno pravilo	*satFlag*	M/C
Backjump $(C = [-6])$	*UNDEF*	-6
Decide $(l = +1)$	*UNDEF*	$-6, \vert +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$-6, \vert +1, +2$
UnitProp $(c = [-1, -2, -3],\ l = -3)$	*UNDEF*	$-6, \vert +1, +2, -3$
Decide $(l = +4)$	*UNDEF*	$-6, \vert +1, +2, -3, \vert 4$
UnitProp $(c = [-2, -4, -5],\ l = -5)$	*UNDEF*	$-6, \vert +1, +2, -3, \vert 4, -5$
Decide $(l = +7)$	*UNDEF*	$-6, \vert +1, +2, -3, \vert 4, -5, \vert +7$
$M \not\models \neg F$, $(\text{vars } M) = (\text{vars } F)$	*SAT*	$-6, \vert +1, +2, -3, \vert 4, -5, \vert +7$

Primer

$F = [[-1, +2], [-3, +4], [-1, -3, +5], [-2, -4, -5], [-2, +3, +5, -6], [-1, +3, -5, -6], [+1, -6], [+1, +7]\ , [-1, -2, -3]\ , [-6]\ ]$.

Primenjeno pravilo	*satFlag*	M/C
Backjump $(C = [-6])$	*UNDEF*	-6
Decide $(l = +1)$	*UNDEF*	$-6, \vert +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$-6, \vert +1, +2$
UnitProp $(c = [-1, -2, -3],\ l = -3)$	*UNDEF*	$-6, \vert +1, +2, -3$
Decide $(l = +4)$	*UNDEF*	$-6, \vert +1, +2, -3, \vert 4$
UnitProp $(c = [-2, -4, -5],\ l = -5)$	*UNDEF*	$-6, \vert +1, +2, -3, \vert 4, -5$
Decide $(l = +7)$	*UNDEF*	$-6, \vert +1, +2, -3, \vert 4, -5, \vert +7$
$M \not\models \neg F$, $(\text{vars } M) = (\text{vars } F)$	*SAT*	$-6, \vert +1, +2, -3, \vert 4, -5, \vert +7$

Primer

$F = [[-1, +2], [-3, +4], [-1, -3, +5], [-2, -4, -5], [-2, +3, +5, -6], [-1, +3, -5, -6], [+1, -6], [+1, +7]\ , [-1, -2, -3]\ , [-6]\ ]$.

Primenjeno pravilo	*satFlag*	M/C
Backjump $(C = [-6])$	*UNDEF*	-6
Decide $(l = +1)$	*UNDEF*	$-6, \vert +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$-6, \vert +1, +2$
UnitProp $(c = [-1, -2, -3],\ l = -3)$	*UNDEF*	$-6, \vert +1, +2, -3$
Decide $(l = +4)$	*UNDEF*	$-6, \vert +1, +2, -3, \vert 4$
UnitProp $(c = [-2, -4, -5],\ l = -5)$	*UNDEF*	$-6, \vert +1, +2, -3, \vert 4, -5$
Decide $(l = +7)$	*UNDEF*	$-6, \vert +1, +2, -3, \vert 4, -5, \vert +7$
$M \not\models \neg F$, $(\text{vars } M) = (\text{vars } F)$	*SAT*	$-6, \vert +1, +2, -3, \vert 4, -5, \vert +7$

Primer

$F = [[-1, +2], [-3, +4], [-1, -3, +5], [-2, -4, -5], [-2, +3, +5, -6], [-1, +3, -5, -6], [+1, -6], [+1, +7]\ , [-1, -2, -3]\ , [-6]\ ]$.

Primenjeno pravilo	*satFlag*	M/C
Backjump $(C = [-6])$	*UNDEF*	-6
Decide $(l = +1)$	*UNDEF*	$-6, \vert +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$-6, \vert +1, +2$
UnitProp $(c = [-1, -2, -3],\ l = -3)$	*UNDEF*	$-6, \vert +1, +2, -3$
Decide $(l = +4)$	*UNDEF*	$-6, \vert +1, +2, -3, \vert 4$
UnitProp $(c = [-2, -4, -5],\ l = -5)$	*UNDEF*	$-6, \vert +1, +2, -3, \vert 4, -5$
Decide $(l = +7)$	*UNDEF*	$-6, \vert +1, +2, -3, \vert 4, -5, \vert +7$
$M \not\models \neg F$, $(\text{vars } M) = (\text{vars } F)$	*SAT*	$-6, \vert +1, +2, -3, \vert 4, -5, \vert +7$

Primer

$F = [[-1, +2], [-3, +4], [-1, -3, +5], [-2, -4, -5], [-2, +3, +5, -6], [-1, +3, -5, -6], [+1, -6], [+1, +7]\ , [-1, -2, -3]\ , [-6]\ ]$.

Primenjeno pravilo	*satFlag*	M/C
Backjump $(C = [-6])$	*UNDEF*	-6
Decide $(l = +1)$	*UNDEF*	$-6, \vert +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$-6, \vert +1, +2$
UnitProp $(c = [-1, -2, -3],\ l = -3)$	*UNDEF*	$-6, \vert +1, +2, -3$
Decide $(l = +4)$	*UNDEF*	$-6, \vert +1, +2, -3, \vert 4$
UnitProp $(c = [-2, -4, -5],\ l = -5)$	*UNDEF*	$-6, \vert +1, +2, -3, \vert 4, -5$
Decide $(l = +7)$	*UNDEF*	$-6, \vert +1, +2, -3, \vert 4, -5, \vert +7$
$M \not\models \neg F$, $(\text{vars } M) = (\text{vars } F)$	*SAT*	$-6, \vert +1, +2, -3, \vert 4, -5, \vert +7$

Primer

$F = [[-1, +2], [-3, +4], [-1, -3, +5], [-2, -4, -5], [-2, +3, +5, -6], [-1, +3, -5, -6], [+1, -6], [+1, +7]\ , [-1, -2, -3]\ , [-6]\ ]$.

Primenjeno pravilo	*satFlag*	M/C
Backjump $(C = [-6])$	*UNDEF*	-6
Decide $(l = +1)$	*UNDEF*	$-6, \mid +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$-6, \mid +1, +2$
UnitProp $(c = [-1, -2, -3],\ l = -3)$	*UNDEF*	$-6, \mid +1, +2, -3$
Decide $(l = +4)$	*UNDEF*	$-6, \mid +1, +2, -3, \mid 4$
UnitProp $(c = [-2, -4, -5],\ l = -5)$	*UNDEF*	$-6, \mid +1, +2, -3, \mid 4, -5$
Decide $(l = +7)$	*UNDEF*	$-6, \mid +1, +2, -3, \mid 4, -5, \mid +7$
$M \not\models \neg F$, $(\text{vars } M) = (\text{vars } F)$	*SAT*	$-6, \mid +1, +2, -3, \mid 4, -5, \mid +7$

Primer

$F = [[-1, +2], [-3, +4], [-1, -3, +5], [-2, -4, -5], [-2, +3, +5, -6], [-1, +3, -5, -6], [+1, -6], [+1, +7]\ , [-1, -2, -3]\ , [-6]\ ]$.

Primenjeno pravilo	*satFlag*	M/C
Backjump $(C = [-6])$	*UNDEF*	-6
Decide $(l = +1)$	*UNDEF*	$-6, \vert +1$
UnitProp $(c = [-1, +2],\ l = +2)$	*UNDEF*	$-6, \vert +1, +2$
UnitProp $(c = [-1, -2, -3],\ l = -3)$	*UNDEF*	$-6, \vert +1, +2, -3$
Decide $(l = +4)$	*UNDEF*	$-6, \vert +1, +2, -3, \vert 4$
UnitProp $(c = [-2, -4, -5],\ l = -5)$	*UNDEF*	$-6, \vert +1, +2, -3, \vert 4, -5$
Decide $(l = +7)$	*UNDEF*	$-6, \vert +1, +2, -3, \vert 4, -5, \vert +7$
$M \not\models \neg F$, (vars M) = (vars F)	*SAT*	$-6, \vert +1, +2, -3, \vert 4, -5, \vert +7$

Heuristike grananja

Kako odabrati literal odlučivanja?

- Izbor atoma za grananje: Variable State Independent Decaying Sum (VSIDS)
  - Prilikom analize konflikta svim literalima klauza koje učestvuju u konfliktu povećava se skor za 1
  - Svi skorovi se periodično množe nekom konstantom manjom od 1
  - Na ovaj način do izražaja dolaze atomi koji su učestvovali u skorijim konfliktima
- Izbor polariteta: Saved polarity
  - Biramo onaj polaritet koji je promenljiva imala prethodnog puta u parcijalnoj valuaciji

Zaboravljanje

Zaboravljanje naučenih klauza?

- Ukoliko broj naučenih klauza postane preveliki, pronalaženje jediničnih i konfliktnih klauza se usporava.
- Neke naučene klauze se posle određenog vremena uklanjaju.
- Zaboravljanje je kontrolisano heuristikama.

Otpočinjanje iznova

Ponovo pokretanje pretrage

- U nekim trenucima je korisno pretragu prekinuti i započeti iznova (engl. restart).
- Postoji nada da će nas klauze koje su u međuvremenu naučene odvesti u neku drugu (lakšu) granu stabla pretrage.
- Otpočinjanje iznova pokazuje dobra svojstva kada se koristi uz određenih procenat slučajnih odluka u toku pretrage.
- Otpočinjanje iznova je kontrolisano heuristikama.

Pouzdanost SAT rešavača

Kako verovati SAT rešavaču?

- CDCL zasnovani SAT rešavači su veoma kompleksan softver
- Na izlazu dobijamo odgovor *SAT* ili *UNSAT*:
  - bilo koja greška u implementaciji rešavača može proizvesti pogrešan odgovor za neki ulaz
  - kako mu možemo verovati?
- U slučaju SAT odgovora, dobijamo i zadovoljavajuću valuaciju
  - lako se može proveriti da li dobijena valuacija zaista zadovoljava formulu
- Šta u slučaju UNSAT odgovora?

Sertifikati

Sertifikat

- Tvrdnja da je formula nezadovoljiva se mora potkrepiti dokazom
- U slučaju CDCL SAT rešavača taj dokaz je rezolucijski
  - kada se desi konflikt na nultom nivou, može se izvršiti analiza konflikta koja će kao rezultat izvesti praznu klauzu
  - zajedno sa ranijim rezolucijskim izvodjenjima naučenih klauza, dobijamo rezolucijski dokaz da je polazni skup klauza nezadovoljiv
- Na zahtev, SAT rešavac može na izlazu dati ovaj rezolucijski dokaz u odgovarajućem formatu
- Ovaj dokaz se obično naziva sertifikat
  - sertifikat se može proveriti nezavisnim softverom koji je mnogo jednostavniji od SAT rešavača i lakše mu se može verovati
- Različiti formati (DRUP, DRAT, GRAT, GRIT, ...)
- Primer alata: drat-trim

Primer

Primer

$F = [[1, -3], [2, 3, -1], [-1, -3, 2], [-1, -2], [1, 3]]$

Primenjeno pravilo	*satFlag*	M/C
	UNDEF	
Decide $(l = +1)$	*UNDEF*	$\vert +1$
UnitProp $(c = [-1, -2], l = -2)$	*UNDEF*	$\vert +1, -2$
UnitProp $(c = [-1, -3, 2], l = -3)$	*UNDEF*	$\vert +1, -2, -3$
Conflict $(M \models \neg [2, 3, -1])$	*UNDEF*	$[2, 3, -1]$
Explain $(l = -3, c = [-1, -3, 2])$	*UNDEF*	$[-1, 2]$
Explain $(l = -2, c = [-1, -2])$	*UNDEF*	$[-1]$
Learn $(C = [-1])$	*UNDEF*	$[-1]$
Backjump $(C = [-1])$	*UNDEF*	-1
UnitProp $(c = [1, -3])$	*UNDEF*	$-1, -3$
Conflict $(M \models \neg [1, 3])$	*UNSAT*	$[1, 3]$
Explain $(l = -3, c = [1, -3])$	*UNSAT*	$[1]$
Explain $(l = -1, c = [-1])$	*UNSAT*	$[\,]$
Learn $(C = [\,])$	*UNSAT*	$[\,]$

Primer

Primer

$F = [[1, -3], [2, 3, -1], [-1, -3, 2], [-1, -2], [1, 3], [-1]]$

Primenjeno pravilo	*satFlag*	M / C
	UNDEF	
Decide $(l = +1)$	*UNDEF*	$\mid +1$
UnitProp $(c = [-1, -2],\ l = -2)$	*UNDEF*	$\mid +1, -2$
UnitProp $(c = [-1, -3, 2],\ l = -3)$	*UNDEF*	$\mid +1, -2, -3$
Conflict $(M \models \neg [2, 3, -1])$	*UNDEF*	$[2, 3, -1]$
Explain $(l = -3,\ c = [-1, -3, 2])$	*UNDEF*	$[-1, 2]$
Explain $(l = -2,\ c = [-1, -2])$	*UNDEF*	$[-1]$
Learn $(C = [-1])$	*UNDEF*	$[-1]$
Backjump $(C = [-1])$	*UNDEF*	-1
UnitProp $(c = [1, -3])$	*UNDEF*	$-1, -3$
Conflict $(M \models \neg [1, 3])$	*UNSAT*	$[1, 3]$
Explain $(l = -3,\ c = [1, -3])$	*UNSAT*	$[1]$
Explain $(l = -1,\ c = [-1])$	*UNSAT*	$[\,]$
Learn $(C = [\,])$	*UNSAT*	$[\,]$

Primer

Primer

$F = [[1, -3], [2, 3, -1], [-1, -3, 2], [-1, -2], [1, 3], [-1], [\,]]$

Primenjeno pravilo	*satFlag*	M/C
	UNDEF	
Decide $(l = +1)$	*UNDEF*	$\mid +1$
UnitProp $(c = [-1, -2],\ l = -2)$	*UNDEF*	$\mid +1, -2$
UnitProp $(c = [-1, -3, 2],\ l = -3)$	*UNDEF*	$\mid +1, -2, -3$
Conflict $(M \models \neg [2, 3, -1])$	*UNDEF*	$[2, 3, -1]$
Explain $(l = -3,\ c = [-1, -3, 2])$	*UNDEF*	$[-1, 2]$
Explain $(l = -2,\ c = [-1, -2])$	*UNDEF*	$[-1]$
Learn $(C = [-1])$	*UNDEF*	$[-1]$
Backjump $(C = [-1])$	*UNDEF*	-1
UnitProp $(c = [1, -3])$	*UNDEF*	$-1, -3$
Conflict $(M \models \neg [1, 3])$	*UNSAT*	$[1, 3]$
Explain $(l = -3,\ c = [1, -3])$	*UNSAT*	$[1]$
Explain $(l = -1,\ c = [-1])$	*UNSAT*	$[\,]$
Learn $(C = [\,])$	*UNSAT*	$[\,]$

Primer - nastavak

Rezolucijski dokaz nezadovoljivosti

$$\dfrac{\dfrac{\dfrac{[2,3,-1] \qquad [-1,-3,2]}{[-1,2]} \qquad [-1,-2]}{[-1]} \qquad \dfrac{[1,3] \qquad [1,-3]}{[1]}}{[\,]}$$

U listovima su klauze polazne formule. Plavi deo predstavlja izvođenje naučene klauze povratnog skoka $[-1]$ prilikom prvog konflikta. Ostatak predstavlja izvođenje prazne klauze prilikom konflikta na nultom nivou.

Pregled

Kompaktnost

Šta ako razmatramo zadovoljivost beskonačnih skupova formula?

- Opisane metode za ispitivanje zadovoljivosti se mogu primeniti samo na konačne skupove formula
- Slučaj beskonacnih skupova se može svesti na konačne skupove primenom svojstva kompaktnosti

Teorema (Kompaktnost iskazne logike)

- *Skup iskaznih formula je zadovoljiv akko je svaki njegov konačan podskup zadovoljiv.*
- *Skup iskaznih formula je nezadovoljiv akko postoji njegov konačan podskup koji je nezadovoljiv.*

Napomena

Kasnije ćemo videti da se svojstvo kompaktnosti prenosi i na logiku prvog reda, što će tamo imati daleko veći značaj.

Pregled

Primene iskazne logike

Izražajnost iskazne logike

Jezik iskazne logike je veoma jednostavan:

- sve promenljive mogu uzimati samo vrednosti tačno ili netačno
- zbog toga se svaki problem mora modelovati u terminima logičkih vrednosti
- iskazi koji nama imaju neko intuitivno značenje se u iskaznoj logici razmatraju apstraktno
  - zbog toga se sve logičke veze među iskazima moraju izraziti logičkim veznicima
  - npr. važi $x < y \wedge y < z \Rightarrow x < z$, ali ako ove iskaze apstrahujemo iskaznim slovima p,q i r, moramo ,,ručno" da zadamo uslov $p \wedge q \Rightarrow r$, jer se to više ne podrazumeva
- ovo ponekad zahteva modelovanje na veoma niskom nivou, a dobijeni modeli mogu biti veoma glomazni
- ipak, veoma široka klasa problema se može predstaviti na ovaj način

Svođenje na SAT

Kada se problem svede na jezik iskazne logike, tada se rešavanje problema može svoditi na:

- ispitivanje tautologičnosti dobijene formule
- ispitivanje zadovoljivosti dobijene formule

U praksi se obično razmatra zadovoljivost, dok se problem tautologičnosti obično svodi na (ne)zadovoljivost negacije:

- razlog za ovo leži u izrazitoj efikasnosti modernih SAT rešavača
- glomaznost iskaznih modela obično nije problem za savremene SAT rešavače
- otuda je svođenje problema na SAT postalo izuzetno popularno u zadnje dve decenije

Dve vrste primena

SAT rešavači kao dokazivači teorema

Tipična primena je u verifikaciji formalnih sistema (najčešće hardvera):

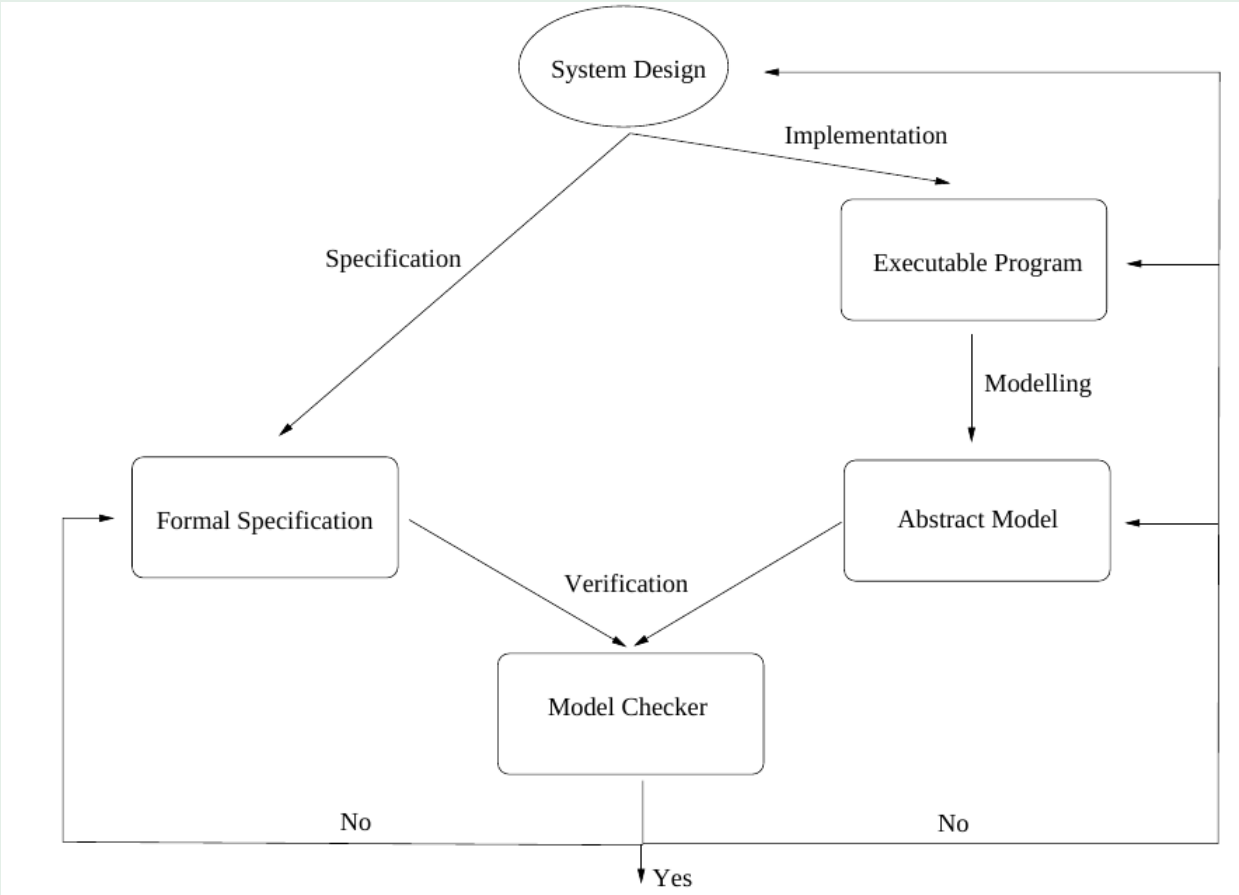
- rad sistema se predstavi iskaznom formulom P
- svojstvo koje mora da važi se predstavi formulom Q
- sada $P \Rightarrow Q$ mora da bude tautologija
- ekvivalentno, $P \wedge \neg Q$ mora da bude nezadovoljiva

SAT rešavači i rešavanje problema kombinatorne pretrage

- uslovi koje rešenje problema mora da zadovoljava predstave se klauzama KNF formule
- zadovoljavajuća valuacija koju SAT rešavač pronalazi kodira traženo rešenje

Proveravanje modela

Strukturni prikaz



Proveravanje modela

Opis komponenti

- Modelovanje: proces formulisanja apstraktnog modela sistema na osnovu konkretne implementacije
- Apstraktni model: tranzicioni sistem (skup stanja i relacija prelaska nad tim skupom)
- Formalna specifikacija: precizan opis svojstava koje apstraktni model treba da zadovolji (na jeziku matematičke logike)
- Proveravač modela: alat koji automatski proverava da li apstraktni model zadovoljava formalnu specifikaciju
- U slučaju da ne zadovoljava, proveravač daje kontraprimer u vidu putanje u modelu koja narušava zadato svojstvo
- Kontraprimer se koristi za otkrivanje greške, korekciju modela i/ili specifikacije

Tranzicioni sistem

Tranzicioni sistem

Tranzicioni sistem je uređena petorka $(S, \longrightarrow, I, \mathcal{V}, \lambda)$, gde je:

- S skup stanja
- $\longrightarrow \subseteq S \times S$ relacija prelaska (u oznaci $s_i \longrightarrow s_j$)
- $I \subseteq S$ skup početnih stanja
- $\mathcal{V}$ skup iskaznih promenljivih (skup predikata)
- $\lambda : S \to \mathbb{P}\mathcal{V}$ funkcija mapiranja (svakom stanju pridružuje skup predikata koji važe u tom stanju)

Napomene

- Skup stanja može biti konačan ili beskonačan (tipično je konačan, ali veoma veliki)
- Bez ograničenja opštosti, pretpostavljamo da je relacija $\longrightarrow$ *totalna*, tj. $(\forall s \in S)(\exists s' \in S)(s \longrightarrow s')$
- Sistem može biti deterministički (ako $(\forall s \in S)(\exists! s' \in S)(s \longrightarrow s')$) i nedeterministički

Tranzicioni sistem

Putanja u sistemu

Putanja u sistemu $T = (S, \longrightarrow, I, \mathcal{V}, \lambda)$ je beskonačni niz stanja $\sigma = s_0 s_1 s_2 \ldots$ takav da važi $s_i \longrightarrow s_{i+1}$ za svako $i \geq 0$. Pri tom ćemo koristiti oznaku $\sigma_i := s_i$ za i-to stanje putanje i $\sigma_{|i} := s_i s_{i+1} s_{i+2} \ldots$ za ,,rep" putanje počev od i-tog stanja.
Izvršavanje u sistemu T je bilo koja putanja σ takva da je $\sigma_0 \in I$.
Za stanje $s \in S$ kažemo da je dostižno ako postoji izvršavanje σ takvo da $s \in \sigma$.

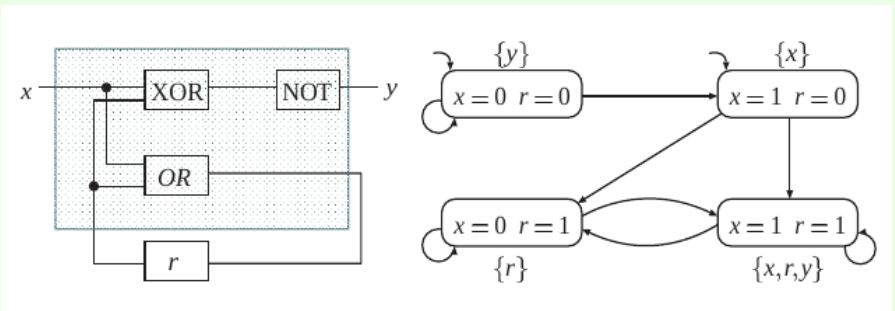
Hardver kao tranzicioni sistem

Modelovanje hardvera kao tranzicionog sistema

- Skup stanja je određen stanjem registara (memorijskih komponenti, flip-flopova) u datom hardverskom sistemu i vrednostima bitova na ulazu
  - ulazi dolaze iz spoljašnjeg okruženja (druge komponente, interakcija sa korisnikom), i njihove vrednosti se ne mogu predvideti (nedeterminizam)
  - tipično imamo veliki broj stanja (eksponencijalan u odnosu na broj bitova registara i bitova ulaza)
  - određene početne konfiguracije hardvera se smatraju početnim stanjima
- Relacija prelaska opisuje ponašanje kola u svakom ciklusu časovnika
- Skup predikata $\mathcal{V}$ obično odgovara bitovima ulaza, bitovima registara, kao i bitovima izlaza
  - $\lambda(s)$ je tada skup svih bitova koji imaju vrednost 1 u stanju s

Primer

Primer



- *x – jednobitni ulaz*
- *y – jednobitni izlaz*
- *r – jednobitni registar*
- *Stanje: par bitova* (x, r)
- *Početna stanja: za* $r = 0$
- *Graf relacije prelaska je prikazan na desnoj slici*
- *Ulaz x uvodi nedeterminizam*
- $\mathcal{V} = \{x, r, y\}$
- *Skup* $\lambda(s)$ *je prikazan uz svako stanje u grafu*

Formalna specifikacija

Svojstva sistema

Formalna specifikacija predstavlja skup svojstava izraženih na nekom formalnom jeziku koja sistem mora da zadovoljava:

- svojstva se izražavaju u terminima predikata koji važe ili ne važe u dostižnim stanjima sistema
- svojstva se mogu odnositi na pojedinačna stanja: npr. ,,u svakom stanju važi svojstvo P"
- svojstva se mogu odnositi i na više stanja odjednom: npr. ,,ako u nekom stanju važi P, onda u sledećem stanju mora važiti Q"
- neka najčešće razmatrane vrste svojstava:
  - invarijante – svojstva koja moraju da važe u svim dostižnim stanjima sistema
  - sigurnosna svojstva – da se neka negativna pojava nikada neće desiti
  - svojstva živosti – neka pozitivna pojava će se sigurno desiti u nekom narednom stanju
  - svojstva pravednosti – neka pojava će se dešavati beskonačno puta u nastavku izvršavanja
- za izražavanje se tipično koristi neki vid temporalne logike koji omogućava zadavanje vremenskih odrednica:
  - npr. *važi u svakom dostižnom stanju*, *desiće se pre ili kasnije*, *važi počev od nekog stanja* i sl.
- pod određenim uslovima, svojstva se mogu aproksimirati iskaznim formulama:
  - npr. ne možemo reći ,,u svakom dostižnom stanju važi svojstvo P", ali možemo reći ,,u prvih k stanja izvršavanja važi svojstvo P"

Opis na jeziku iskazne logike

Opis izvršavanja tranzicionog sistema $(S, \longrightarrow, I, \mathcal{V}, \lambda)$

Neka je $\mathcal{V} = \{p_1, \ldots, p_n\}$ i neka je dato proizvoljno izvršavanje $s_0 s_1 s_2 \ldots$ u tranzicionom sistemu.

- Za svako stanje s_i imaćemo skup iskaznih slova $p_1^i, \ldots, p_n^i$ (kopije predikata u i-tom stanju)
- Skup početnih stanja opisujemo iskaznom formulom $I(s_0)$ nad promenljivama $p_1^0, \ldots, p_n^0$:
  - npr. možemo imati formulu $\bigvee_{s \in I} \bigwedge_j l_j^s$, gde je $l_j^s = p_j^0$ ili $l_j^s = \neg p_j^0$, u zavisnosti od toga da li je $p_j \in \lambda(s)$
  - često postoji i jednostavniji način da se izraze početni uslovi
- Prelazak $s_i \longrightarrow s_{i+1}$ predstavljamo iskaznom formulom $R(s_i, s_{i+1})$ nad promenljivama $p_1^i, \ldots, p_n^i, p_1^{i+1}, \ldots, p_n^{i+1}$
  - npr. možemo imati formulu $\bigwedge_{s \in S}((\bigwedge_j l_j^s) \Rightarrow (\bigvee_{s' \mid s \longrightarrow s'}(\bigwedge_j l_j^{s'})))$, gde je $l_j^s = p_j^i$ ili $l_j^s = \neg p_j^i$, a $l_j^{s'} = p_j^{i+1}$ ili $l_j^{s'} = \neg p_j^{i+1}$, u zavisnosti od toga da li predikat p_j važi u stanjima s, odnosno s', respektivno
  - opet, u konkretnom sistemu, opis relacije prelaska može biti i jednostavniji

Ograničena provera modela

Ograničena provera modela (bounded model checking (BMC))

Ideja je da proverimo da li neko svojstvo važi u prvih k koraka izvršavanja:

- ispostavi se da se sva ranije spomenuta svojstva tada mogu izraziti na jeziku iskazne logike
- ovde ćemo razmatrati invarijante, jer su one najjednostavnije, budući da se odnose na pojedinačna stanja

Provera invarijante

Neka je invarijanta data iskaznom formulom $J(s)$ nad predikatima $p_1, \ldots, p_n$ u stanju s. Da bismo proverili da li invarijanta mora da važi u prvih k koraka proizvoljnog izvršavanja, ispitujemo zadovoljivost sledeće formule:

$$I(s_0) \wedge R(s_0, s_1) \wedge \ldots \wedge R(s_{k-1}, s_k) \wedge \neg(J(s_0) \wedge \ldots \wedge J(s_k))$$

- ako je formula nezadovoljiva, tada invarijanta sigurno važi u prvih k koraka
- ako je formula zadovoljiva, tada zadovoljavajuća valuacija daje kontraprimer, tj. kodira izvršavanje u kome invarijanta neće važiti (ovo olakšava pronalaženje grešaka u modelu)

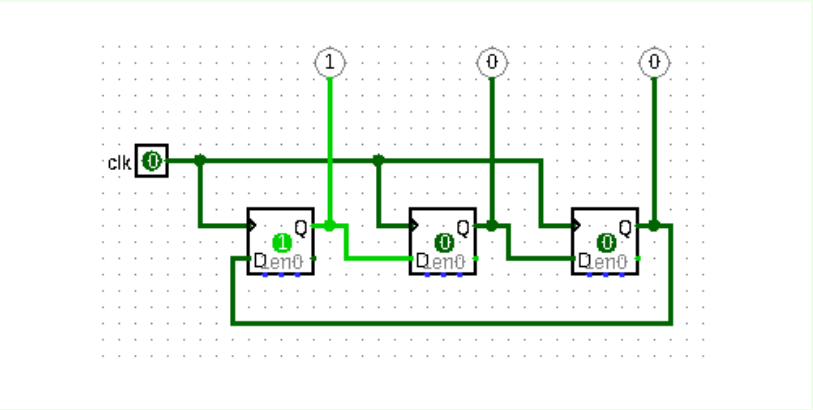
Napomena

Iako ograničena provera modela nije potpuna metoda, ona u praksi može otkriti veliki broj grešaka.

Primer

Primer

Na slici je dat je trobitni ciklični registar sa početnom vrednošću 100. *Dokazati svojstvo: ,,U svakom trenutku je tačno jedan bit registra uključen".*



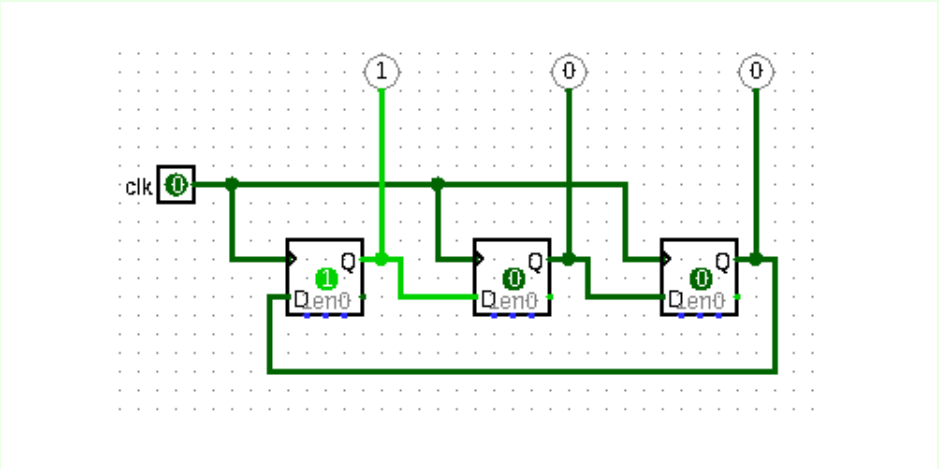
Imamo model:

- *Skup stanja:* $S = \{0,1\}^3$ *(sve trojke bitova pqr)*
- *Relacija prelaska:* $pqr \longrightarrow rpq$
- *Početno stanje:* 100
- *Predikati:* $\mathcal{V} = \{p, q, r\}$
- $\lambda(s)$ *sadrži jedini bit koji je uključen u stanju* s

Primer (2)

Primer

(nastavak) Pretpostavimo da vršimo ograničenu proveru modela za $k = 3$.

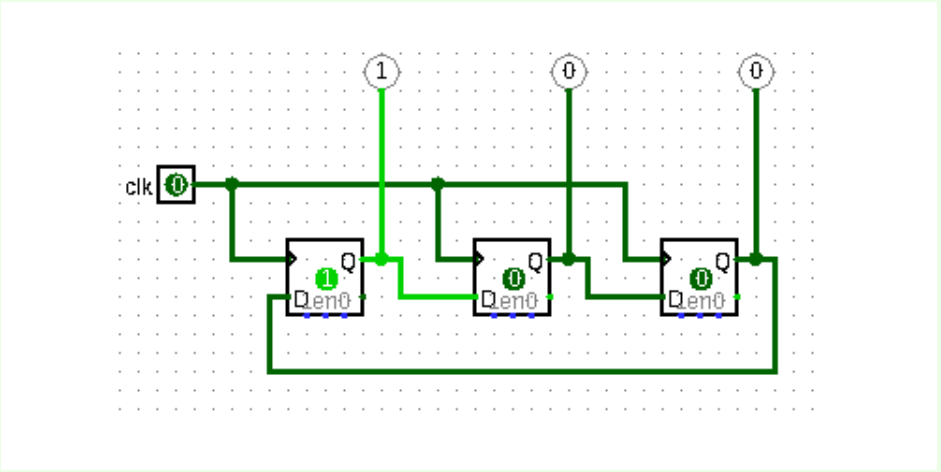


- $I(s_0) \equiv (p^0 \wedge \neg q^0 \wedge \neg r^0)$ *(početno stanje je* 100*)*
- $R(s_i, s_{i+1}) \equiv (p^{i+1} \Leftrightarrow r^i) \wedge (q^{i+1} \Leftrightarrow p^i) \wedge (r^{i+1} \Leftrightarrow q^i)$
- *Rad kola u prva* 3 *koraka se opisuje formulom:*
  $I(s_0) \wedge R(s_0, s_1) \wedge R(s_1, s_2) \wedge R(s_2, s_3)$

Primer (3)

Primer

(nastavak) Kodiramo svojstvo ,,u svakom trenutku je tačno jedan bit uključen":



- $J(s_i) \equiv (p^i \vee q^i \vee r^i) \wedge (\neg p^i \vee \neg q^i) \wedge (\neg p^i \vee \neg r^i) \wedge (\neg q^i \vee \neg r^i)$
- *Formula čija se zadovoljivost ispituje:*
  $I(s_0) \wedge R(s_0, s_1) \wedge R(s_1, s_2) \wedge R(s_2, s_3) \wedge \neg(J(s_0) \wedge J(s_1) \wedge J(s_2) \wedge J(s_3))$

Dokazivanje indukcijom

Dokazivanje indukcijom

Jedan način da se obezbedi potpunost (tj. da invarijanta važi za svako dostižno stanje) je da se koristi indukcija. Baza indukcije glasi:

$$I(s_0) \wedge \neg J(s_0)$$

Intuitivno, nezadovoljivost ove formule znači da za svako početno stanje invarijanta J važi. Sada imamo induktivni korak:

$$J(s_0) \wedge R(s_0, s_1) \wedge \neg J(s_1)$$

Intuitivno, nezadovoljivost ove formule znači da kad god invarijanta J važi u nekom stanju s_0, tada važi i u stanju s_1 koje može slediti za stanjem s_0 u datom tranzicionom sistemu.

Dokazivanje indukcijom (2)

k-indukcija

Ponekad je potrebno znati da je invarijanta važila u prethodnih k stanja da bismo mogli da zaključimo da važi i u sledećem stanju. Tada prethodna formula za bazu indukcije dobija oblik:

$$I(s_0) \wedge R(s_0, s_1) \wedge \ldots \wedge R(s_{k-2}, s_{k-1}) \wedge \neg(J(s_0) \wedge \ldots \wedge J(s_{k-1}))$$

Intuitivno, nezadovoljivost ove formule znači da će u prvih k stanja proizvoljnog izvršavanja invarijanta J sigurno važiti (ovo je ista formula koju smo imali i za BMC). Induktivni korak dobija sledeći oblik:

$$J(s_0) \wedge J(s_1) \wedge \ldots \wedge J(s_{k-1}) \wedge R(s_0, s_1) \wedge \ldots \wedge R(s_{k-1}, s_k) \wedge \neg J(s_k)$$

Intuitivno, nezadovoljivost ove formule znači da ako invarijanta važi u prethodnih k stanja, tada važi i u sledećem stanju.

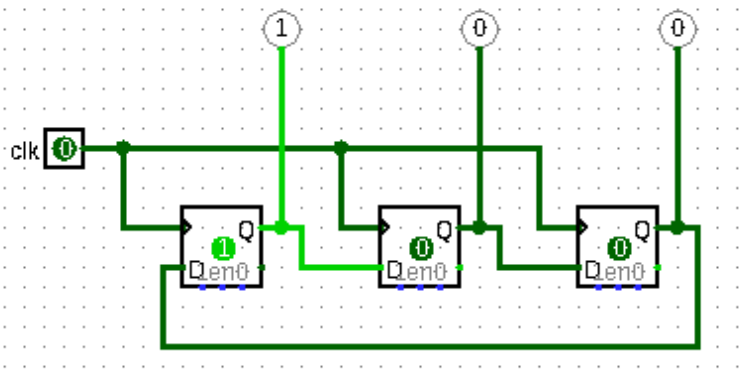
Napomena

- u slučaju da je formula za bazu indukcije zadovoljiva, tada imamo kontraprimer dužine k
- u slučaju da je formula za induktivni korak zadovoljiva, tada je potrebno probati za veće k

Primer

Primer

Primenimo indukciju na prethodni primer:



U ovom primeru, indukcija će proći za $k = 1$*, tj. imaćemo nezadovoljivost formule* $I(s_0) \wedge \neg J(s_0)$*, kao i formule* $J(s_0) \wedge R(s_0, s_1) \wedge \neg J(s_1)$*.*

Problem zadovoljenja ograničenja

Definicija

Problem zadovoljenja ograničenja (engl. *Constraint Satisfaction Problem (CSP)*) je uređena trojka $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, pri čemu je:

- $\mathcal{X} = (x_1, \ldots, x_n)$ niz promenljivih
- $\mathcal{D} = (D_1, \ldots, D_n)$ niz odgovarajućih domena promenljivih, pri čemu je D_i domen promenljive x_i (što čemo pisati kao $D(x_i) = D_i$)
- $\mathcal{C} = (C_1, \ldots, C_k)$ niz ograničenja, pri čemu je svako ograničenje C_i podskup od $D_{i_1} \times \ldots \times D_{i_r}$, za neki rastući niz indeksa $i_1 < \ldots < i_r$. Kažemo da je C_i ograničenje *nad promenljivama* $x_{i_1}, \ldots, x_{i_r}$ (u oznaci $\mathcal{X}(C_i) = (x_{i_1}, \ldots, x_{i_r})$). Broj r nazivamo *arnost* ograničenja C

Rešenje CSP problema $\mathcal{P}$ je uređena n-torka $(d_1, \ldots, d_n) \in D_1 \times \ldots \times D_n$, takva da za svako ograničenje $C_i \in \mathcal{C}$ za koje je $\mathcal{X}(C_i) = (x_{i_1}, \ldots, x_{i_r})$, r-torka $(d_{i_1}, \ldots, d_{i_r})$ pripada C_i.

Napomena

Ograničenja ćemo obično zapisivati simbolički (npr. $x = y$, $x + y < z$ i sl.). Rešenje $(d_1, \ldots, d_n)$ ćemo često zapisivati i kao dodelu $\{x_1 = d_1, \ldots, x_n = d_n\}$

Napomena

Mi ćemo se ograničiti na CSP probleme nad konačnim domenima.

Primer – latinski kvadrat

Primer

Razmotrimo primer latinskog kvadrata:

1			
			3
	4		
		2	

Prazna polja u kvadratu treba popuniti brojevima od 1 *do* 4*, tako da u svakoj vrsti i svakoj koloni svi brojevi budu međusobno različiti.*

Primer – latinski kvadrat (2)

Primer

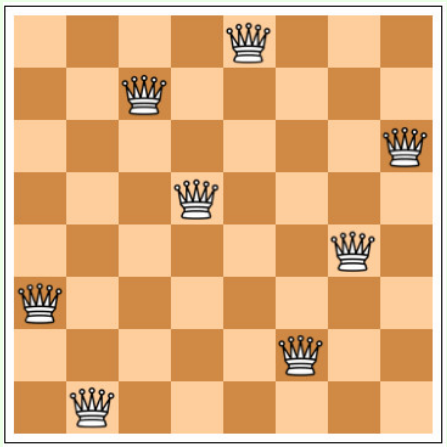
(nastavak) Latinski kvadrat se može predstaviti kao CSP problem na sledeći način:

- *skup promenljivih:* x_{ij} *za polje u* i*-toj vrsti i* j*-toj koloni*
- *domeni svih promenljivih su* $\{1, 2, 3, 4\}$
- *ograničenja su:*
  - $x_{ij_1} \neq x_{ij_2}$ *za* $1 \leq j_1 < j_2 \leq 4$ *i* $i \in \{1, 2, 3, 4\}$
  - $x_{i_1 j} \neq x_{i_2 j}$ *za* $1 \leq i_1 < i_2 \leq 4$ *i* $j \in \{1, 2, 3, 4\}$

Primer – 8 kraljica

Primer

Razmotrimo problem 8 kraljica: potrebno je rasporediti 8 kraljica na šahovsku tablu tako da se međusobno ne napadaju. Jedno moguće rešenje ovog problema dato je na slici:



Primer – 8 kraljica (2)

Primer

(nastavak) Problem 8 kraljica se može predstaviti kao CSP problem na sledeći način:

- *imamo promenljive* $x_1, \ldots, x_8$*, sa domenom* $\{1, \ldots, 8\}$*, pri čemu promenljiva* x_i *označava redni broj kolone u kojoj se nalazi kraljica iz* i*-te vrste (jasno je da su sve kraljice u različitim vrstama)*
- *imamo ograničenja:*
  - $x_i \neq x_j$ *za* $i \neq j$ *– nikoje dve kraljice se ne smeju nalaziti u istoj koloni*
  - *za* $i \neq j$ *mora da važi* $|x_i - x_j| \neq |i - j|$*, tj. nikoje dve kraljice ne smeju biti na istoj dijagonali*

Primer – Problem ranca

Primer

Razmotrimo čuveni problem ranca (engl. knapsack problem): na raspolaganju nam je ranac fiksiranog kapaciteta K, kao i skup predmeta, pri čemu svaki predmet ima svoju veličinu i svoju vrednost. Zadatak je spakovati što više predmeta u ranac tako da se ne prekorači zadati kapacitet, a da vrednost spakovanih predmeta bude što veća.

NAPOMENA: Umesto ove, optimizacione varijante problema, možemo posmatrati problem odlučivanja: da li postoji rešenje u kome je vrednost spakovanih predmeta veća od neke zadate vrednosti V?

Primer – Problem ranca (2)

Primer

Problem ranca se može predstaviti kao CSP problem na sledeći način: numerišimo najpre predmete redom sa $1, 2 \ldots, n$. *Neka je* v_i *vrednost* i*-tog predmeta, a* s_i *njegova veličina. Uvedimo promenljive* $x_1, \ldots, x_n$ *sa domenima* $D = \{0, 1\}$. *Intuitivno, promenljiva* x_i *biće jednaka* 1 *akko je* i*-ti predmet spakovan u ranac. Sada imamo ograničenje:*

$$s_1 x_1 + s_2 x_2 + \ldots + s_n x_n \leq K$$

kojim se obezbeđuje da se kapacitet ranca ne prekorači. Najzad, da bismo obezbedili da vrednost spakovanih predmeta bude bar V, *imamo ograničenje:*

$$v_1 x_1 + v_2 x_2 + \ldots + v_n x_n \geq V$$

SAT problem kao CSP problem

SAT kao CSP

SAT problem se može razumeti kao poseban tip CSP problema, gde sve promenljive imaju dvočlani domena $\{true, false\}$ i gde su sva ograničenja klauze.

Klauza kao ograničenje

Klauza $C = l_1 \vee l_2 \vee \ldots \vee l_k$ gde je $l_i \in \{p_i, \neg p_i\}$, se može razumeti kao ograničenje nad promenljivama $p_1, \ldots, p_k$:

$$C = \{true, false\}^k \setminus \{(v_1, \ldots, v_k) \mid v_i = false \text{ ako } l_i = p_i, true \text{ inače}\}$$

Opšti CSP rešavači i odnos sa SAT rešavačima

Opšti CSP rešavači

Opšti CSP rešavači su zasnovani na sličnim principima kao i DPLL algoritam:

- rezonovanje zasnovano na složenim pravilima propagacije
- pretraga zasnovana na podeli domena promenljive i razmatranju odgovarajućih podproblema

Odnos SAT i CSP rešavača

SAT rešavači su po pravilu daleko efikasniji:

- jednostavnost domena i uniformnost ograničenja omogućava efikasne tehnike implementacije
  - u opštem slučaju ne postoje tako efikasne tehnike poput sheme dva posmatrana literala
  - takodje, ne postoje heuristike grananja koje su tako efikasne u opštem slučaju
- Značajna algoritamska poboljšanja takođe nije tako lako realizovati u slučaju opšteg CSP-a:
  - nehronološko vraćanje unazad je korišćeno i u opštem slučaju, ali sa znatno manje uspeha
  - nogood learning: tehnika analogna učenju klauza

Zbog toga je veoma čest pristup u novije vreme da se CSP problemi rešavaju svođenjem na SAT.

Svođenje CSP problema na SAT

SAT rešavači i opšti CSP problemi

CSP problemi nad konačnim domenima se mogu svesti na SAT. Najpre treba kodirati domene promenljivih:

- ako je domen promenljive x_i jednak $\{v_1, \ldots, v_k\}$, tada možemo uvesti iskazno slovo p_{ij} sa značenjem $x_i = v_j$
  - klauzama $p_{i1} \vee p_{i2} \vee \ldots \vee p_{ik}$ i $\neg p_{ij_1} \vee \neg p_{ij_2}$ (za $j_1 < j_2$) obezbeđujemo da tačno jedan od atoma p_{ij} bude tačan, tj. da x_i uzme tačno jednu vrednost iz svog domena
  - opisano kodiranje naziva se direktno kodiranje (engl.direct encoding)
- alternativno, ako je domen uređen $(v_1 < \ldots < v_k)$ možemo uvesti iskazna slova q_{ij} sa značenjem $x_i \leq v_j$
  - možemo uvesti klauze: $\neg q_{ij_1} \vee q_{ij_2}$ za $j_1 < j_2$, kao i jediničnu klauzu q_{ik}
  - ovakvo kodiranje se naziva kodiranje poretka (engl. order encoding)

Nakon toga je potrebno svako ograničenje kodirati skupom klauza (za različite tipove ograničenja postoje različita kodiranja).

Primer – Latinski kvadrat (3)

Primer

(nastavak) Da bismo Latinski kvadrat sveli na SAT, uvedimo iskazna slova p_{ijk} sa značenjem $x_{ij} = k$. Sada imamo klauze:

- $\bigvee_k p_{ijk}$ *za svako* i, j, *kao i* $\neg p_{ijk_1} \vee \neg p_{ijk_2}$ *za svako* i, j *i* $k_1 < k_2$ *(uslovi domena)*
- $\neg p_{ij_1k} \vee \neg p_{ij_2k}$ *za svako* i, k *i* $j_1 < j_2$ *(dva polja u* i*-toj vrsti ne smeju imati istu vrednost* k*)*
- $\neg p_{i_1jk} \vee \neg p_{i_2jk}$ *za svako* j, k *i* $i_1 < i_2$ *(dva polja u* j*-toj vrsti ne smeju imati istu vrednost* k*)*

Primer – 8 kraljica (3)

Primer

(nastavak) Da bismo problem 8 kraljica sveli na SAT, uvodimo promenljive p_{ij} sa značenjem da je i-ta kraljica u j-toj koloni (tj. da je $x_i = j$). Kao i ranije, imaćemo klauze $\bigvee_j p_{ij}$ za svako i, kao i $\neg p_{ij_1} \vee \neg p_{ij_2}$ za $j_1 < j_2$ i svako i. Ove klauze kodiraju uslov da se svaka kraljica nalazi tačno u jednoj koloni. Sada je potrebno obezbediti da za svako j imamo najviše jedno p_{ij} koje je tačno (u svakoj koloni imamo tačno jednu kraljicu). Ovo postižemo klauzama oblika $\neg p_{i_1 j} \vee \neg p_{i_2 j}$ za $i_1 < i_2$ i svako j. Najzad, za svaki par polja (i_1, j_1) i (i_2, j_2) za koji važi $|i_1 - i_2| = |j_1 - j_2|$ (tj. koja se nalaze na istoj dijagonali) imamo da ne sme biti $x_{i_1} = j_1 \wedge x_{i_2} = j_2$, tj. mora da važi $\neg p_{i_1 j_1} \vee \neg p_{i_2 j_2}$.

Primer – Ranac (3)

Primer

U problemu ranca, promenljive x_i imaju domene $\{0, 1\}$ pa se već mogu smatrati iskaznim atomima. Pritom, ograničenja koja se u ovom problemu pojavljuju:

$$s_1 x_1 + s_2 x_2 + \ldots + s_n x_n \leq K$$

i

$$v_1 x_1 + v_2 x_2 + \ldots + v_n x_n \geq V$$

spadaju u posebnu vrstu ograničenja koja nazivamo pseudo-bulovska ograničenja, a o kojima govorimo detaljnije u nastavku.

Pseudo-bulovska ograničenja

Definicija

Ograničenje oblika:

$$c_1 l_1 + c_2 l_2 + \ldots + c_n l_n \bowtie d$$

gde su c_i i d celi brojevi, l_i su literali, a $\bowtie \in \{=, <, >, \leq, \geq, \neq\}$ nazivaju se pseudo-bulovksa ograničenja (PB). Pritom, prilikom sabiranja vrednost literala l_i računamo kao 1 ako je l_i tačan, a 0 u suprotnom.

Napomena

Specijalni tip ovih ograničenja se dobija kada su svi koeficienti c_i jednaki 1:

$$l_1 + l_2 + \ldots + l_n \bowtie d$$

Ovakva ograničenja nazivaju se ograničenja kardinalnosti, jer se njima ograničava broj literala $l_1, \ldots, l_n$ koji su tačni.

Napomena

Primetimo da se svaka klauza:

$$l_1 \vee \ldots \vee l_n$$

može smatrati ograničenjem kardinalnosti:

$$l_1 + \ldots + l_n \geq 1$$

Otuda se problem zadovoljenja ograničenja kardinalnosti (pa i pseudo-bulovskih ograničenja) može smatrati uopštenjem KNF SAT problema.

Primer

Primer

Setimo se da smo u latinskom kvadratu zahtevali da među promenljivama p_{ijk} za fiksirano i i k najviše jedna bude tačna (tj. da najviše jedno polje u i-toj vrsti ima vrednost k). Ovo se može jednostavno predstaviti ograničenjem kardinalnosti:

$$p_{i1k} + p_{i2k} + p_{i3k} + p_{i4k} \leq 1$$

Ovo ograničenje se jednostavno kodiralo skupom klauza $\neg p_{ij_1k} \vee \neg p_{ij_2k}$ za svako $j_1 < j_2$. Međutim, da li je tako jednostavno kodirati ovakva ograničenja u opštem slučaju (npr. kada je desna strana veća od 1)?

Pseudo-bulovska ograničenja

Kodiranje pseudo-bulovskih ograničenja na jeziku iskazne logike

- Direktno kodiranje pseudo-bulovskih ograničenja pomoću klauza može zahtevati eksponencijalno mnogo klauza
- Postoje efikasnija kodiranja koja podrazumevaju uvođenje dodatnih iskaznih atoma (slično kao kod Cajtina)
- Jedno kodiranje se može dobiti simulacijom binarnih sabirača:
  - razmatramo ograničenje $c_1 l_1 + c_2 l_2 + \ldots + c_n l_n \geq d$, gde je $c_i \geq 0$ i $d \geq 0$ (svaki PB problem možemo svesti na ovakva ograničenja)
  - neka je $S_2 = c_1 l_1 + c_2 l_2$
  - zbir S_2 se predstavi binarnim sabiračem, gde se u binarnim zapisima brojeva c_1 i c_2 jedinice zamenjuju literalima l_1 i l_2, respektivno
  - u binarnom sabiraču se uvode dodatne promenljive koje označavaju prenose između pozicija, a onda se vrši prevođenje u KNF
  - nakon toga se uvodi $S_3 = S_2 + c_3 l_3$ i predstavlja se sabiračem na sličan način, i td. ($S_i = S_{i-1} + c_i l_i$)
  - na kraju se realizuje binarni komparator koji ispituje da li je $S_n \geq d$ i on se prevodi u KNF

Optimizacioni PB problemi

Optimizacija PB izraza

Često je potrebno rešiti pseudo-bulovski problem, tako da rešenje maksimizuje (ili minimizuje) dati pseudo-bulovski izraz: $c_1 l_1 + \ldots c_n l_n$:

- pretpostavimo da je u pitanju minimizacija i da smo pronašli rešenje za koje je vrednost datog izraza K
- dodajemo ograničenje $c_1 l_1 + \ldots c_n l_n < K$ i ponovo rešavamo problem
- ako je sada problem nezadovoljiv, tada je K upravo minimalna vrednost izraza $c_1 l_1 + \ldots + c_n l_n$
- u suprotnom neka je vrednost datog izraza u pronađenom rešenju K' (koje je sigurno manje od K)
- sada dodajemo ograničenje $c_1 l_1 + \ldots c_n l_n < K'$ i ponovo rešavamo problem, i td.

Opisana strategija se naziva linearna strategija. Alternativa je tzv. binarna strategija, gde se optimalna vrednost traži binarnom pretragom:

- u svakom trenutku imamo neki interval $[L, U]$ u kome tražimo minimalnu vrednost funkcije
- za K uzimamo sredinu intervala $(L + U)/2$ i dodajemo ograničenje $c_1 l_1 + \ldots + c_n l_n < K$
- ako je problem zadovoljiv, tada pretragu nastavljamo u intervalu $[L, K]$, a ako je nezadovoljiv, nastavljamo u $[K, U]$

U praksi se obično koristi linearna strategija, jer je efikasnija:

- iako binarna strategija konvergira brže, ona zahteva rešavanje većeg broja nezadovoljivih problema
- sa druge strane, linearna strategija podrazumeva veći broj zadovoljivih i jedan nezadovoljiv problem
- nezadovoljivi problemi obično zahtevaju više vremena za rešavanje
- otuda linearna strategija obično zahteva manje vremena za pronalaženje optimalnog rešenja

MaxSAT problem

Definicija

Za datu KNF formulu, problem MaxSAT se sastoji u pronalaženju valuacije u kojoj je zadovoljen najveći mogući broj klauza.

Napomena

MaxSAT problem se može razumeti kao uopštenje SAT problema: KNF formula je zadovoljiva akko je najveći mogući broj zadovoljenih klauza u nekoj valuaciji jednak ukupnom broju klauza.

Definicija

Uopštenje MaxSAT problema je težinski MaxSAT: svakoj klauzi C_i se dodeljuje težinski koeficient $w_i \geq 0$. Sada je cilj da se pronađe valuacija takva da je zbir težina klauza koje su tačne u toj valuaciji maksimalan.

MaxSAT problem

Svođenje težinskog MaxSAT problema na PB problem

- Neka je dat skup klauza $\{C_1, \ldots, C_m\}$, pri čemu klauza C_i ima težinu w_i
- Za svaku klauzu $C_i \equiv l_{i1} \vee l_{i2} \vee \ldots \vee l_{in_i}$ uvodimo novo iskazno slovo r_i
- Klauzu C_i zamenjujemo uslovom $r_i \Rightarrow C_i$, tj. klauzom $\neg r_i \vee l_{i1} \vee \ldots \vee l_{in_i}$
- Sada je cilj maksimizovati PB izraz $w_1 r_1 + w_2 r_2 + \ldots + w_m r_m$

Linearna ograničenja

Linearna ograničenja

Daljim uopštenjem pseudo-bulovskih ograničenja dolazimo do linearnog ograničenja:

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n \bowtie b$$

gde su a_i i b celobrojne konstante, x_i su promenljive čiji su domeni konačni celobrojni intervali $[l_i, u_i]$, a $\bowtie \in \{=, \neq, \leq, \geq, <, >\}$.

Napomena

Lako se može pokazati da se CSP koji uključuje linearna ograničenja uvek može transformisati u ekvivalentan CSP u kome sva linearna ograničenja imaju oblik:

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n \leq b$$

Primer – magični kvadrat

Primer

Dat je magični kvadrat:

	12		
	8	15	
7		2	
4			11

Potrebno je prazna polja popuniti celim brojevima iz intervala $[1, 16]$, *tako da svi brojevi u tablici budu različiti, a zbir po svim vrstama, kolonama i velikim dijagonalama bude 34.*

Primer – magični kvadrat (2)

Primer

(nastavak) Magični kvadrat se jednostavno predstavlja kao CSP problem:

- *Uvodimo promenljive* x_{ij} *za polje u* i*-toj vrsti i* j*-toj koloni* $(1 \leq i, j \leq 4)$
- *Sve promenljive imaju domen* $\{1, 2, \ldots, 16\}$
- *Skup ograničenja:*
  - $x_{i1} + \ldots + x_{i4} = 34$, *za svako* $i \in \{1, 2, 3, 4\}$
  - $x_{1j} + \ldots + x_{4j} = 34$, *za svako* $j \in \{1, 2, 3, 4\}$
  - $x_{11} + x_{22} + x_{33} + x_{44} = 34$
  - $x_{41} + x_{32} + x_{23} + x_{14} = 34$
  - $x_{12} = 12$, $x_{22} = 8$, $x_{23} = 15$, $x_{31} = 7$, $x_{33} = 2$, $x_{41} = 4$, $x_{44} = 11$
  - $x_{ij} \neq x_{i'j'}$ *za svaka dva različita polja* (i, j) *i* (i', j')

Linearna ograničenja

Svođenje na SAT

Neka je dato ograničenje:

$$a_1 x_1 + a_2 x_2 + \ldots + a_n x_n \leq b$$

Jednostavnosti radi, pretpostavimo da su svi a_i nenegativni. Ako je $x_i \geq c_i$ (za $i = 1, 2, \ldots, n-1$), tada mora biti $x_n \leq c_n$, gde je

$$c_n = \left\lfloor \left(b - \sum_{1 \leq i < n} a_i c_i \right) / a_n \right\rfloor$$

Otuda imamo implikaciju $x_1 \geq c_1 \wedge x_2 \geq c_2 \wedge \ldots \wedge x_{n-1} \geq c_{n-1} \Rightarrow x_n \leq c_n$, koja je ekvivalentna sa: $x_1 \leq c_1 - 1 \vee x_2 \leq c_2 - 1 \vee \ldots \vee x_{n-1} \leq c_{n-1} - 1 \vee x_n \leq c_n$. Ako uvedemo iskazno slovo q_i^k sa značenjem $x_i \leq k$ ($k \in [l_i, u_i]$), sada se ovaj uslov može zapisati kao iskazna klauza (dakle, koristimo kodiranje poretka za predstavljanje domena). Ovakve klauze treba napraviti za sve moguće vrednosti c_i iz domena odgovarajućih promenljivih.

Linearna ograničenja

Svođenje na SAT

(nastavak) Ukupan broj klauza dobijen na prethodni način je $O(d^{n-1})$, gde je d veličina najvećeg domena, a n broj promenljivih u linearnom ograničenju. Ovaj broj se može značajno smanjiti, ako se najpre linearno ograničenje transformiše na sledeći način:

- uvedemo promenljive $y_2 = a_1x_1 + a_2x_2$, $y_3 = y_2 + a_3x_3$, $\ldots$, $y_{n-2} = y_{n-3} + a_{n-2}x_{n-2}$
- ograničenje $y_i = y_{i-1} + a_ix_i$ možemo zameniti parom ograničenja $y_{i-1} + a_ix_i - y_i \leq 0$ i $-y_{i-1} - a_ix_i + y_i \leq 0$
- dodatno, imamo ograničenje $y_{n-2} + c_{n-1}x_{n-1} + c_nx_n \leq b$

Dakle, sveli smo ovo ograničenje na konjunkciju linearnih ograničenja sa po tri promenljive. Sada svako od tih ograničenja predstavimo klauzama na prethodno opisan način. Broj klauza će sada biti $O(nd^2)$, što je obično značajno manje.

Optimizacioni problemi sa linearnim ograničenjima

Primedba

Optimizacioni problemi kod kojih je potrebno minimizovati (maksimizovati) linearni celobrojni izraz $a_1x_1 + \ldots + a_nx_n$ se mogu rešavati na sličan način kao i optimizacioni pseudo-bulovski problemi (korišćenjem linearne ili binarne strategije).