DPLL
a metody pro SAT

Jiří Vyskočil

2011

Co je SAT a co nabízí

- SAT = Boolean SATisfiability problem
- Řeší problém nalezení ohodnocení proměnných v booleovské formuli bez kvantifikátorů tak, že je formule splněna.
- Jinými slovy řešíme problém pravdivosti uzavřené existenčně kvantifikované booleovské formule.
- Příklad:

  $\exists x_1 x_2 x_3 x_4 \;\; (x_1 \vee \neg x_2 \vee \neg x_3) \,\&\, (x_1 \vee x_2 \vee x_4)$

  Řešení:

  Formule je splněna když: (x_1, x_2, x_3, x_4=TRUE)
- Vstupem většiny SAT solverů je CNF (Conjunctive Normal Form).
- Nalezení řešení SAT je NP-úplný problém (Stephen Cook 1971).

Převod do CNF

- Pokud chceme převést libovolnou výrokovou formuli do CNF pouze s použitím vstupních proměnných je problém až exponenciální (výsledná formule je až exponenciálně větší než vstupní). Složitost je tedy O(exp(n)).
- Pokud povolíme vznik nových pomocných proměnných, které ale neovlivní splnitelnost původní formule (compact CNF), lze převod uskutečnit v lineární složitosti (výsledná formule je konstanta krát větší než vstupní). Časová složitost může být rovněž lineární (v závislosti na použitém algoritmu např. RBC O(n), Boy de la Tour alg. $O(n^2)$).

Proč převádět na SAT?

- Dnes existují desítky výkonných SAT solverů (např. MiniSAT, zChaff, …), které se každoročně testují na stovkách testovacích příkladů v soutěžích http://www.satcompetition.org/.
- Složitost nalezení řešení pomocí SAT solveru je řádově menší než řešení problému klasickým ATP například s úplnou rezolucí.
- SAT má dlouhodobě standardizovaný a jednoduchý formát (na rozdíl od CSP (Constraint Satisfaction Problem)), lze tedy snadno používat více různých solverů.
- Existují preprocesory pro SAT, které na některých typech úloh výrazně zvyšují výkonnost (např. SatELite)
- Pokud problém nelze vyjádřit jako SAT, existují rozšíření, které řeší složitější úlohy (např. Paradox, Equinox, některé QBF solvery, …)

Jak fungují SAT solvery

- Většina nejrychlejších současných SAT solverů používá algoritmus založený na DPLL (Davis-Putnam-Logemann-Loveland) algoritmu (1962).

- Základní operace a pojmy v tomto algoritmu jsou:

  unit clause (jednotková klauzule)

  je klausule která obsahuje právě jeden literál. Tento literál obsahuje nenastavenou proměnnou.

  unit-propagation(U, Φ)

  kde Φ je CNF, které chceme nastavit její proměnné tak, aby byla splněna a U je *unit clause* obsažená ve Φ.

  Existuje pouze jediný způsob jak může být klauzule L splněna.

  Pokud je U tvaru $\neg x$ pak musí platit, že $x = \text{false}$ jinak $x = \text{true}$.

  Nyní ekvivalentně upravíme Φ tak, aby již žádná její klauzule neobsahovala literál U s výjimkou toho původního (propagujeme jednotkovou klauzuli):

  for every clause C **in** Φ

  if C $\supseteq$ U **then** remove C from Φ

  else if C $\supseteq$ $\bar{U}$ **then** delete $\bar{U}$ from C;

  add U to Φ;

unit-propagation – příklad

- Příklad na *unit-propagation*:

  Mějme následující CNF Φ ve formě množiny klausulí

  $$\{ a \vee b, \neg a \vee c, \neg c \vee d, a \}$$

  a literál U = a

  Nyní zavoláme unit-propagation(U,Φ).

  Klausuli ($a \vee b$) odstraníme z Φ, protože klauzule obsahuje U.

  Z klausule ($\neg a \vee c$) odstraníme ¬a, protože klauzule obsahuje Ū.

  Klausuli ($\neg c \vee d$) ponecháme beze změny, protože neobsahuje ani U ani Ū.

  Klausuli (a) nemůžeme odstranit z Φ, protože bychom porušili ekvivalenci.

  Výsledná CNF Φ je $\{c, \neg c \vee d, a\}$.

  Pokud bychom ještě jednou zavolali unit-propagation na Φ a literál c, získali bychom CNF $\Phi=\{d, a, c\}$.

Jak fungují SAT solvery

pure-literal-elimination (U, Φ)

kde Φ je CNF, které chceme nastavit její proměnné tak, aby byla splněna a U je literál, který má v celé Φ tzv. *pure* výskyt. To znamená, že se všude ve Φ vyskytuje buď ve tvaru x anebo se všude ve Φ vyskytuje ve tvaru $\neg x$.

Jinými slovy: literál je *pure*, pokud se vykytuje v celé Φ právě s jednou polaritou.

Pokud je U tvaru $\neg x$ potom nastavením $x := \text{false}$ jinak $x := \text{true}$ jistě neporušíme splnitelnost původní formule Φ.

Na základě předchozí úvahy ekvivalentně upravíme Φ tak, aby již neobsahovala žádnou klausuli s *pure* literálem U a navíc pro zachování ekvivalence přidáme U jako unit clause do Φ:

for every clause C **in** Φ

    if C $\supseteq$ U **then** remove C from Φ;

add U to Φ;

funkce *pure-literal-elimination* se často u moderních SAT solverů vypouští kvůli nízkému zastoupení případů, kde se uplatní a kvůli relativně vysokým nákladům na detekci vlastnosti *pure* pro každou proměnnou.

DPLL

```
function DPLL(Φ : CNF) : boolean
    if Φ is a consistent set of literals then return true;
    if Φ contains an empty clause then return false;
    for every unit clause U in Φ
            Φ := unit-propagation(U, Φ);
    for every literal P that occurs pure in Φ
            Φ := pure-literal-elimination(P, Φ);
    L := choose-literal(Φ);
    return DPLL(Φ ∪ {¬L}) or DPLL(Φ ∪ {L});
```

Podmínka je splněna, pokud Φ obsahuje pouze *unit* klauzule a pokud nejsou žádné dvě klauzule ve sporu (tj. neobsahuje zároveň klauzule x a ¬x).

Splňující ohodnocení proměnných lze zjistit z aktuálního stavu Φ.

Vybereme libovolný literál z Φ. Efektivita celého DPLL algoritmu je tímto výběrem značně ovlivněna.

MiniSat

- autoři: Niklas Eén a Niklas Sörensson
- implementován v C
- MIT licence (lze tudíž použit zdarma v dalším software)
- velmi výkonný
  - vítězství v roce 2005 ve všech kategoriích: Industrial
  - vítězství v roce 2006 v SAT-Race
- dobrá dokumentace
- minimalistický návrh vhodný pro další rozšiřování
- současná další rozšíření: MiniSat+, SatELite, ...

MiniSat – celková struktura

```
Solve() {
    while (true) {
        boolean_constraint_propagation();
        if (no_conflict) {
I           if (no_unassigned_variable) return SAT;
            decision_level++;
            make_decision();
        } else {
            if  (no_decisions_made) return UNSAT;
            decision_level = analyze_conflict();
            undo_assignments(decision_level);
        }
    }
}
```

MiniSat – analýza konfliktu

- Konflikt nastane pokud se nějaká klauzule stane nesplnitelnou během propagace *unit* klauzule (boolean_constraint_propagation()).
- **Příklad:**

  Předpokládejme, že x_4 je false (v DPLL by to odpovídalo přidání $\neg x_4$ do problému).

  Po propagaci x_4 na následující problém

  $(x_1 \vee x_4)$ &

  $(\neg x_1 \vee x_2)$ &

  $(\neg x_2 \vee x_3)$ &

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

  se stane poslední klauzule nesplnitelnou.

- Následně analyzujeme konfliktní klauzuli a postupně z ní odvozujeme novou, která je konzistentní s původním problémem, ale způsobuje konflikt dříve.
- Výsledná odvozená klauzule je jako nová znalost přidána do problému

MiniSat – analýza konfliktu

```
analyze_conflict() {
    learnt_clause = find_confclicting_clause();

    while (! stop_criterion_met(learnt_clause)) {
        lit = choose_literal(learnt_clause);
        var = get_variable_of_literal(lit);
        ante = antecedent(var);
        learnt_clause = resolve(learnt_clause, ante, var);
    }
    add_clause_to_database(learnt_clause);

    backtrack_decision_level = clause_asserting_level(learnt_clause);
    return backtrack_decision_level;
}
```

Opakuj dokud neplatí, že learnt_clause je nesplnitlená a learnt_clause obsahuje právě jeden literál jehož hodnota je již známa.

Vybere naposledy vypropagovaný literál (funkcí boolean_constraint_propagation()) z literálů v learnt_clause.

Vrátí klauzuli, ze které byla hodnota proměnné var vypropagována funkcí boolean_constraint_propagation()

Provede rezoluci (aplikuje rezoluční pravidlo) na klauzule learnt_clause a ante podle proměnné var.

Vrátí nejmenší decision_level, ve kterém je learnt_clause *unit* klauzule.

MiniSat – analýza konfliktu

- Příklad:

  (¬f ∨ e) &

  (¬g ∨ f) &

  (¬h ∨ g) &

  (b ∨ a ∨ e) &

  (c ∨ e ∨ f ∨ ¬b) &

  (d ∨ ¬b ∨ h) &

  (¬b ∨ ¬c ∨ ¬d) &

  (c ∨ d)

MiniSat – analýza konfliktu

- Příklad:

  (¬f ∨ e) &

  (¬g ∨ f) &

  (¬h ∨ g) &

  (b ∨ a ∨ e) &

  (c ∨ e ∨ f ∨ ¬b) &

  (d ∨ ¬b ∨ h) &

  (¬b ∨ ¬c ∨ ¬d) &

  (c ∨ d)

Přiřazení proměnných		Hodnota antecedent()
e=false	decision_level=1	předpoklad
f=false		¬f ∨ e
g=false		¬g ∨ f
h=false		¬h ∨ g
a=false	decision_level=2	Předpoklad
b=true		b ∨ a ∨ e
c=true		c ∨ e ∨ f ∨ ¬b
d=true		d ∨ ¬b ∨ h

learnt_clause = (¬b ∨ ¬c ∨ ¬d)

konfliktní klauzule

MiniSat – analýza konfliktu

- Příklad:

  (¬f ∨ e) &

  (¬g ∨ f) &

  (¬h ∨ g) &

  (b ∨ a ∨ e) &

  (c ∨ e ∨ f ∨ ¬b) &

  (d ∨ ¬b ∨ h) &

  (¬b ∨ ¬c ∨ ¬d) &

  (c ∨ d)

Přiřazení proměnných		Hodnota antecedent()
e=false	decision_level=1	předpoklad
f=false		¬f ∨ e
g=false		¬g ∨ f
h=false		¬h ∨ g
a=false	decision_level=2	Předpoklad
b=true		b ∨ a ∨ e
c=true		c ∨ e ∨ f ∨ ¬b
d=true		d ∨ ¬b ∨ h

learnt_clause = (¬b ∨ ¬c ∨ h)

Výsledek rezoluce na (¬b ∨ ¬c ∨ ¬d) a (d ∨ ¬b ∨ h) podle d.

MiniSat – analýza konfliktu

- Příklad:

  (¬f v e) &

  (¬g v f) &

  (¬h v g) &

  (b v a v e) &

  (c v e v f v ¬b) &

  (d v ¬b v h) &

  (¬b v ¬c v ¬d) &

  (c v d)

Přiřazení proměnných		Hodnota antecedent()
e=false	decision_level=1	předpoklad
f=false		¬f v e
g=false		¬g v f
h=false		¬h v g
a=false	decision_level=2	Předpoklad
b=true		b v a v e
c=true		c v e v f v ¬b
d=true		d v ¬b v h

learnt_clause = (¬b v e v f v h)

Výsledek rezoluce na (¬b v ¬c v h) a (c v e v f v ¬b) podle c.

MiniSat – analýza konfliktu

- Příklad:

  (¬f v e) &

  (¬g v f) &

  (¬h v g) &

  (b v a v e) &

  (c v e v f v ¬b) &

  (d v ¬b v h) &

  (¬b v ¬c v ¬d) &

  (c v d)

Přiřazení proměnných		Hodnota antecedent()
e=false	decision_level=1	předpoklad
f=false		¬f v e
g=false		¬g v f
h=false		¬h v g
a=false	decision_level=2	Předpoklad
b=true		b v a v e
c=true		c v e v f v ¬b
d=true		d v ¬b v h

learnt_clause = (¬b v e v f v h)

Splněna podmínka zastavení cyklu.
learnt_clause přidáme do problému.

MiniSat – analýza konfliktu

- Příklad:

  (¬f ∨ e) &

  (¬g ∨ f) &

  (¬h ∨ g) &

  (b ∨ a ∨ e) &

  (c ∨ e ∨ f ∨ ¬b) &

  (d ∨ ¬b ∨ h) &

  (¬b ∨ ¬c ∨ ¬d) &

  (c ∨ d) &

  (¬b ∨ e ∨ f ∨ h)

MiniSat – analýza konfliktu

- Příklad:

  (¬f v e) &

  (¬g v f) &

  (¬h v g) &

  (b v a v e) &

  (c v e v f v ¬b) &

  (d v ¬b v h) &

  (¬b v ¬c v ¬d) &

  (c v d) &

  (¬b v e v f v h)

Přiřazení proměnných		Hodnota antecedent()
e=false		předpoklad
f=false		¬f v e
g=false	decision_level=1	¬g v f
h=false		¬h v g
b=false		¬b v e v f v h
…		