Satisfiability Solvers

Carla P. Gomes, Henry Kautz, Ashish Sabharwal, Bart Selman

Satisfiability (SAT) solvers zijn algorithmes die kunnen bepalen of een logische propositie vervuld kan worden door 'true' en 'false' in te vullen. SAT solvers kunnen ook gebruikt worden door problemen die normaal gesproken niet als propositioneel beschouwd worden op te lossen als deze naar een Boolean satisfiability problem kan worden omgeschreven.

Het 'Boolean satisfiability problem' (SAT) is als volgt: Gegeven een CNF (conjuctive normal form) formule, is het mogelijk om waar of onwaar aan de variabelen toe te kennen zodanig dat de gehele propositie waar is. Een formule is in CNF als deze bestaat uit conjuncties van clausules waar de clausules disjuncties zijn. Een voorbeeld is $F = (a \vee \neg b) \wedge (\neg a \vee c \vee d) \wedge (b \vee d)$. Bij SAT problemen is niet alleen het resultaat belangrijk (satisfiable of niet), maar ook de toewijzingen van de variabelen waar de uitkomst 'true' is, als deze bestaat.

Een 'complete' methode voor een SAT solver is er een waar er of een 'satisfying' toewijzing is voor de variabelen, of waar wordt bewezen dat de formule 'unsatisfiable' is. De beste complete SAT solvers zijn varianten op de DPLL (Davis–Putnam–Logemann–Loveland) procedure. Een belangrijk kenmerk van DPLL is dat het efficiënt de zoekruimte 'snoeit' op basis van 'false' clausules.

Figuur 1 laat de procedure van DPLL op CNF formules zien. De stap waar de waarde van ℓ wordt toegewezen (true of false) heet een 'decision' en is geassocieerd met een 'decision level' gelijk aan de diepte van de recusie. Het einde van elke recursive call wordt 'backtracking' genoemd.

Algorithm 2.1: `DPLL-recursive`(F, ρ)

Input : A CNF formula F and an initially empty partial assignment ρ
Output : UNSAT, or an assignment satisfying F
begin
  $(F, \rho) \leftarrow$ `UnitPropagate`(F, ρ)
  if F contains the empty clause **then return** UNSAT
  if F has no clauses left **then**
    Output ρ
    return SAT
  $\ell \leftarrow$ a literal not assigned by ρ `// the branching step`
  if `DPLL-recursive`$(F|_\ell, \rho \cup \{\ell\}) = SAT$ **then return** SAT
  return `DPLL-recursive`$(F|_{\neg\ell}, \rho \cup \{\neg\ell\})$
end

`sub UnitPropagate`(F, ρ)
begin
  while F contains no empty clause but has a unit clause x **do**
    $F \leftarrow F|_x$
    $\rho \leftarrow \rho \cup \{x\}$
  return (F, ρ)
end

Figuur 1: $DPLL - recursive(F, \rho)$

De efficiëntie van SAT solvers is afhankelijk van verschillende kenmerken. Enkelen worden hieronder genoemd.

Variable (and value) selection heuristic: ook wel 'decision strategy'. Verschillende strategieën om de keuzes te maken.

Clause learning: het vinden van causes met conflicten en het gebruiken van deze informatie om het zoeken efficiënter te maken.

Watched literals scheme: het 'bekijken' van twee speciale literals voor elke clausule die nog niet is opgelost die bij de huidige toewijzing ofwel true zijn ofwel nog geen toewijzing hebben.

Fast backjumping: laat de solver direct naar een lagere decision level d gaan wanneer er ook maar één branch leidt tot een conflict in level d of lager.

Assignment stack shrinking: Als een conflict ontstaan door schending van clausule C' en de resulterende conflictclausule C een bepaalde lengte overschrijden, springt de solver terug naar een van de hoogste decision levels van de literals in C.

Conflict clause minimization: zo veel mogelijk literals van clausule C waarvan wordt geïmpliceerd dat ze false zijn als de rest van de literals uit C ook false zijn worden geïdentificeerd en verwijderd.

Randomized restarts: laat het algorithme het zoeken willekeurig stoppen en herstarten vanaf decision level nul.

In figuur 2 is de top level structuur van een DPLL-based SAT solver met clause learning te zien. Het algoritme is hier in iteratieve vorm (in plaats van recursief) wat de meest gebruikte vorm is. Een clause learning algoritme stopt en verklaart een formule unsatisfiable wanneer eenheidspreiding leidt tot een conflict op decision level nul.

Algorithm 2.2: `DPLL-ClauseLearning-Iterative`

```
Input  : A CNF formula
Output : UNSAT, or SAT along with a satisfying assignment
begin
    while TRUE do
        DecideNextBranch
        while TRUE do
            status ← Deduce
            if status = CONFLICT then
                blevel ← AnalyzeConflict
                if blevel = 0 then return UNSAT
                Backtrack (blevel)
            else if status = SAT then
                Output current assignment stack
                return SAT
            else break
end
```

Figuur 2: $DPLL - ClauseLearning - Iterative$