SAT and SMT

Marc Chevalier

DI ENS

March 20, 2019

A First Encounter With SAT Problem

Solving, the Historic Landmark Way

Solving, a New Way

Improving DPLL

From any formula

SMT

TP

A First Encounter With SAT Problem

Solving, the Historic Landmark Way

Solving, a New Way

Improving DPLL

From any formula

SMT

TP

What is SAT?

- SAT = SATisfiability problem
- Theoretically: Cook's theorem. Central role in complexity theory (not only for NP, but also for a lot of variants).
- In practice:
  - a lot of problems are easily encoded in SAT,
  - solving SAT is not that bad (current status: billions on variables),
  - $\Rightarrow$ a lot of people use SAT-solvers and a lot of people develop better and better solvers.

What is SAT?

We want to solve a formula in conjunctive normal form (CNF):

$$(\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_2} \vee x_5) \wedge x_1$$

4 nested levels:

- Variables: x_1, ..., x_n;
- Literals: x_1, ..., x_n, $\overline{x_1}$, ..., $\overline{x_n}$
- Clauses: $\alpha_1 \vee \cdots \vee \alpha_n$;
- Formulas: $\Phi = C_1 \wedge \cdots \wedge C_n$.

Is a formula satisfiable? ie. Is there a assignation of variables such that a formula evaluates to $\top$?
Edges cases:

- Empty clause $\Rightarrow$
- Empty formula $\Rightarrow$

What is SAT?

We want to solve a formula in conjunctive normal form (CNF):

$$(\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_2} \vee x_5) \wedge x_1$$

4 nested levels:

- Variables: x_1, ..., x_n;
- Literals: x_1, ..., x_n, $\overline{x_1}$, ..., $\overline{x_n}$
- Clauses: $\alpha_1 \vee \cdots \vee \alpha_n$;
- Formulas: $\Phi = C_1 \wedge \cdots \wedge C_n$.

Is a formula satisfiable? ie. Is there a assignation of variables such that a formula evaluates to $\top$?
Edges cases:

- Empty clause $\Rightarrow$ Unsatisfiable
- Empty formula $\Rightarrow$

What is SAT?

We want to solve a formula in conjunctive normal form (CNF):

$$(\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_2} \vee x_5) \wedge x_1$$

4 nested levels:

- Variables: x_1, ..., x_n;
- Literals: x_1, ..., x_n, $\overline{x_1}$, ..., $\overline{x_n}$
- Clauses: $\alpha_1 \vee \cdots \vee \alpha_n$;
- Formulas: $\Phi = C_1 \wedge \cdots \wedge C_n$.

Is a formula satisfiable? ie. Is there a assignation of variables such that a formula evaluates to $\top$?
Edges cases:

- Empty clause $\Rightarrow$ Unsatisfiable
- Empty formula $\Rightarrow$ Satisfiable

Is CNF necessary?

- No but sufficient.
- A lot of good things happens with CNF when solving
- DNF sucks:

$$(\overline{x_1} \wedge x_2 \wedge x_3) \vee (\overline{x_2} \wedge x_5) \vee x_1$$

A First Encounter With SAT Problem

Solving, the Historic Landmark Way

Solving, a New Way

Improving DPLL

From any formula

SMT

TP

The Setup

The levels:

- A set $\mathbb{V}$ of variables.
- $\mathbb{L} := \mathbb{V} \times \{+, -\}$ the set of literals.
- $\mathbb{C} := \mathcal{P}(\mathbb{L})$ the set of clauses.
- $\mathbb{F} := \mathcal{P}(\mathbb{C})$ the set of formulas.

Empty formula $= \emptyset$, a formula with one empty clause $= \{\emptyset\}$.
$\Phi_1 \vDash \Phi_2$ means "any assignment satisfying Φ_1 satisfies Φ_2".
If Φ_2 is not satisfiable, Φ_1 neither.

Resolution

Resolution:

$$\{\{x, \alpha_1, \dots, \alpha_n\}, \{\overline{x}, \beta_1, \dots, \beta_n\}\} \vDash \{\{\alpha_1, \dots, \alpha_n, \beta_1, \dots, \beta_n\}\}$$

Let's illustrate on

$$\underbrace{(\overline{x} \vee z)}_{(1)} \wedge \underbrace{(\overline{y} \vee z)}_{(2)} \wedge \underbrace{\overline{z}}_{(3)} \wedge \underbrace{(x \vee y)}_{(4)}$$

- $(1), (3) \vDash \overline{x}$ (5)
- $(4), (5) \vDash y$ (6)
- $(2), (3) \vDash \overline{y}$ (7)
- $(6), (7) \vDash \{\emptyset\}$ Game over!

Resolution is

- correct: new clauses does not change the set of satisfying assignments;
- not complete: some correct clauses cannot be obtained by resolution;
- refutation-complete: if the formula is not satisfiable, we can always get the empty clause.

David-Putnam Algorithm

- We ordered the variables: $x_1 < \cdots < x_n$.
- Idea: resolving as much as possible for the biggest variable and iterate.
- $2n$ buckets: $B_1^+, \dots, B_n^+$, $B_1^-, \dots, B_n^-$.
- A clause is put in bucket B_i^* if:
  - it contains literal x_i^*
  - all other variables are in $x_1, \dots, x_{i-1}$
  - (if x_i and $\overline{x_i}$ are both in a clause, it is trivial and removed)
- We apply resolution to all pairs of clauses in $B_k^+ \times B_k^-$.
  - If one of the buckets is empty, the literal is set to $\top$ and we remove it.
- Stop if we get an empty clause or when all resolutions are done.

A First Encounter With SAT Problem

Solving, the Historic Landmark Way

Solving, a New Way

Improving DPLL

From any formula

SMT

TP

Davis-Putnam-Logemann-Loveland

Let's try if $x_1 = \top$ (and recursively) It works? Great! It does not work? Let's try $x_1 = \bot$ (and recursively).
But enumerating all assignment... is not great. So:

- We precociously detect conflicts, as soon as they appear.
- A conflict rules out a set of assignments.
- Each time we try to guess, we also deduce some necessary assignments. Eg. with the clause $(x \vee y)$ if we bet that $x = \bot$, we can deduce that $y = \top$.

So we do not enumerate all assignments.
3 mains steps: deduce, choose and backtrack.

Deducing

2 simplifications:

- If a clause is only a literal, it must be $\top$.
- If $\alpha = \top$ in the current assignment,
  - remove all clauses that contains α (they are satisfied),
  - remove $\overline{\alpha}$ everywhere it appears.

(Iterate both until fixpoint)
The last operation can create empty clauses (conflict). Time to backtrack!

Example

Other slides

A First Encounter With SAT Problem

Solving, the Historic Landmark Way

Solving, a New Way

Improving DPLL

From any formula

SMT

TP

Improving DPLL

- Clever choice for the next variable to assign.
- Clause learning.
- Deep backtrack.
- Reset.
- Choosing an assignment and tweaking it.

Choosing the next literal

2 problems: choosing a variable and a polarity.

- Simple: first unassigned variable, starting with $\top$;
- Random: a randomly chosen unassigned variable, starting with $\top$;
- Smart: a randomly chosen unassigned variable, starting with the most common polarity;
- MOMS (Maximum Occurrences in Clauses of Minimal Size): the most common literal in minimal size clauses because it is the most constraint place;
- DLIS (Dynamic Largest Individual Sum): choosing the literal that satisfies as many clauses as possible;
- A variant of DLIS: maximizing $\sum_{\alpha \in C \in \Phi} 2^{-|C|}$ over α.

Personal experiments: Simple, random and smart suck equally; MOMS and DLIS are much better, DLIS seems a bit better.

Clause learning

- $\vdots$
- For some reasons, α_1
- I have $\overline{\alpha_1} \vee \alpha_2 \vee \underbrace{\beta_1 \vee \cdots \vee \beta_k}_{\text{all } \bot}$
- So α_2
- I have $\overline{\alpha_2} \vee \alpha_3 \vee \underbrace{\gamma_1 \vee \cdots \vee \gamma_l}_{\text{all } \bot}$
- So α_3
- $\overline{\alpha_3} \vee \underbrace{\delta_1 \vee \cdot \vee \delta_m}_{\text{all } \bot}$.
- Bam! Conflict!

I could have used resolution here to get $\overline{\alpha_1} \vee \beta_1 \vee \cdots \vee \beta_k \vee \alpha_3 \vee \gamma_1 \vee \cdots \vee \gamma_l$, $\overline{\alpha_2} \vee \gamma_1 \vee \cdots \vee \gamma_l \vee \delta_1 \vee \cdot \vee \delta_m$, so $\overline{\alpha_1} \vee \beta_1 \vee \cdots \vee \beta_k \vee \gamma_1 \vee \cdots \vee \gamma_l \vee \delta_1 \vee \cdot \vee \delta_m$. The last one is the most interesting because we could have deduced the conflict much sooner (as soon as all β_i, γ_i and δ_i are fixed).

Deep backtrack

In the previous case, once we have learned
$\overline{\alpha_1} \vee \beta_1 \vee \cdots \vee \beta_k \vee \gamma_1 \vee \cdots \vee \gamma_l \vee \delta_1 \vee \cdot \vee \delta_m$, we can see the conflict earlier.
Corollary: we can backtrack directly to this point since conflict is sure since.

A First Encounter With SAT Problem

Solving, the Historic Landmark Way

Solving, a New Way

Improving DPLL

From any formula

SMT

TP

The Naive Way: De Morgan

All formulas are not CNF. So, to make our solver works on any formula, we need to translate them to CNF.
With De Morgan's laws:

$$(p \wedge q) \vee r \to (p \vee r) \wedge (q \vee r) \qquad \neg(p \wedge q) \to \neg p \vee \neg q$$
$$\neg(p \vee q) \to \neg p \wedge \neg q \quad p \Rightarrow q \to \neg p \vee q \qquad \neg\neg p \to p$$

- Necessary if we want to preserve the formula semantics.
- Exponential.

In fact, we do not care about semantics, but only about satisfiability.

Better: Tseitin

For each subformula p, we introduce ξ_p, and inductively:

$$
\begin{aligned}
[p = p_1 \vee p_2] &= (\neg\xi_p \vee \xi_{p_1} \vee \xi_{p_2}) \wedge (\xi_p \vee \neg\xi_{p_1}) \wedge (\xi_p \vee \neg\xi_{p_2}) \\
[p = p_1 \wedge p_2] &= (\neg\xi_p \vee \xi_{p_1}) \wedge (\neg\xi_p \vee \xi_{p_2}) \wedge (\xi_p \vee \neg\xi_{p_1} \vee \neg\xi_{p_2}) \\
[p = \neg p_1] &= (\neg\xi_p \vee \neg\xi_{p_1}) \wedge (\xi_p \vee \xi_{p_1}) \\
[p = x] &= (\neg\xi_p \vee x) \wedge (\xi_p \vee \neg x)
\end{aligned}
$$

A formula P is translated into

$$\xi_P \wedge \bigwedge_{p \in P} [p]$$

Better: Tseitin – Properties

- The transformation can be computed in linear time.
- The satisfiability is preserved.

A First Encounter With SAT Problem

Solving, the Historic Landmark Way

Solving, a New Way

Improving DPLL

From any formula

SMT

TP

What is SMT?

SMT is SAT where we are opening logic variables.
For instance:

- Congruence:

$$\underbrace{f(a, g(b)) = X}_{\alpha_1} \Rightarrow \underbrace{g(g(X)) \neq g(Y)}_{\alpha_2} \wedge \underbrace{f(X, Y) = f(Z, X)}_{\alpha_3}$$

- Affine inequalities:

$$(\underbrace{k_2 \leqslant k_5 + 3}_{\alpha_1} \wedge \underbrace{k_2 > k_1}_{\alpha_2}) \Rightarrow (\underbrace{k_5 \leqslant k_7 - 2}_{\alpha_3} \vee \underbrace{k_7 > k_1 + 1}_{\alpha_4})$$

For SAT, atoms are literals. Literals are thus not independent!

Solving a SMT problem

- We manage to compile directly the theory into SAT. Compilation. (eg. graph coloring)
-
  - offline: limited collaboration, static problem.
  - online: extended collaboration, dynamic enrichment.

Offline SMT

We try to solve the SAT problem.
Once we have a solution, we check if it is consistent in the theory. If it is, we won!
If it is not, we look for another solution of the SAT problem.
$\Rightarrow$ Enumeration of solutions.

Online SMT

We try to solve the SAT problem.
If a fail happens in the theory, the theory solver generate new literals to explain the conflict.
What we expect from the theory solver:

- check if a set of literals is satisfiable
- if satisfiable, it must be able to generate a model.

And optionally:

- Incremental: deciding a new literal or backtracking should not reset the computation.
- Explaining conflicts.
- Deducing in the theory: deducing the value of a literal of the initial formula.

TP

- Implementing Davis-Putnam
- Implementing DPLL
- Variable choosing heuristic
- Clause learning, if you feel it!