SAT and SMT algorithms

Paul Jackson
Paul.Jackson@ed.ac.uk

School of Informatics
University of Edinburgh

Formal Verification
Autumn 2023

1Including contributions by Elizabeth Polgreen
Basic questions

SAT: Given a propositional logic formula, is it satisfiable?

SMT (SAT Modulo Theories): Given a logical formula over theories (e.g. \( \mathbb{Z}, \mathbb{R} \), arrays, uninterpreted functions), is it satisfiable?

- A formula is *valid* just when its negation is *unsatisfiable*.
- Hence SAT & SMT solvers are also *automatic theorem provers*.

Huge range of applications

- Reasoning engines for many kinds of formal verification tools
- **Constraint Programming**: e.g. planning, scheduling, Suduko
- Automatic test-case generation for programs
- Synthesis of programs & systems from specifications
SAT solver progress

SAT is NP-complete: no polynomial time algorithm.

Yet, huge progress has been made in size of formula that modern SAT solvers can solve:

<table>
<thead>
<tr>
<th>Year</th>
<th># Vars</th>
</tr>
</thead>
<tbody>
<tr>
<td>1960</td>
<td>80</td>
</tr>
<tr>
<td>1970</td>
<td>100</td>
</tr>
<tr>
<td>1980</td>
<td>120</td>
</tr>
<tr>
<td>1990</td>
<td>700</td>
</tr>
<tr>
<td>2000</td>
<td>3,000</td>
</tr>
<tr>
<td>2010</td>
<td>600,000</td>
</tr>
</tbody>
</table>

Size of realistic problems solved in a few hours
Terminology

- An atom $p$ is a propositional symbol. Also call an atom a propositional variable or simply a variable.

- A literal $l$ is an atom $p$ or the negation of an atom $\neg p$.

- A clause $C$ is a disjunction of literals $l_1 \lor \ldots \lor l_n$.

- A CNF formula $F$ is a conjunction of clauses $C_1 \land \ldots \land C_m$

$(\text{CNF} \equiv \text{Conjunctive Normal Form})$
Use of CNF

Standard to always first convert formulas to CNF

- Can get exponential blow-up in size.

Consider putting into CNF

\[(x_1 \land x_2) \lor \ldots \lor (x_{2n} \land x_{2n+1})\]

- If introduce a new variable for each non-terminal in a formula’s syntax tree, can get an equi-satisfiable formula with constant-factor growth in formula size (Tseitin’s encoding)

\[x_1 \Rightarrow (x_2 \land x_3)\]

becomes, with new variables \(z_1\) and \(z_2\),

\[z_1 \land (z_1 \iff (x_1 \Rightarrow z_2)) \land (z_2 \iff ((x_2 \land x_3)))\]

which can easily be converted to CNF with a constant growth-factor
Abstract rules for DPLL

Core algorithms used in SAT and SMT solvers derived from DPLL algorithm (Davis, Putnam, Logemann, Loveland) from 1962.

Here present algorithms using abstract rule-based system due to Nieuwenhuis, Oliveras and Tinelli.

- General structure of algorithms easy to see
- Can work through simple examples on paper
General approach

- Try to incrementally build a satisfying truth assignment $M$ for a CNF formula $F$

- Grow $M$ by
  - guessing truth value of a literal not assigned in $M$
  - deducing truth value from current $M$ and $F$.

- If reach a contradiction ($M \models \neg C$ for some $C \in F$), undo some assignments in $M$ and try starting to grow $M$ again in a different way.

- If all variables from $M$ assigned and no contradiction, a satisfying assignment has been found for $F$

- If exhaust possibilities for $M$ and no satisfying assignment is found, $F$ is unsatisfiable
Assignments and States

States:

\[ \text{fail} \quad \text{or} \quad M \parallel F \]

where

- M is a sequence of literals and decision points • denoting a partial truth assignment
- F is a set of clauses denoting a CNF formula

First literal after each • is called a decision literal

Decision points start suffixes of M that might be discarded when choosing new search direction

Def: If \( M = M_0 \bullet M_1 \bullet \cdots \bullet M_n \) where each \( M_i \) contains no decision points

- \( M_i \) is decision level \( i \) of \( M \)
- \( M_n \), decision level \( n \), is the current decision level
Initial and final states

Initial state

- $(\cdot) \parallel F_0$

Expected final states

- **fail** if $F_0$ is unsatisfiable
- $M \parallel G$ otherwise, where
  - $G$ is equivalent to $F_0$
  - $M$ satisfies $G$
Classic DPLL rules

Decide

\[
M \parallel F \implies M \bullet l \parallel F \quad \text{if} \begin{cases} 
  l \text{ or } \neg l \text{ in clause of } F, \\
  l \text{ is undefined in } M
\end{cases}
\]

Backtrack

\[
M \bullet l N \parallel F, C \implies M \neg l \parallel F, C \quad \text{if} \begin{cases} 
  M \bullet l N \models \neg C \\
  \bullet \notin N
\end{cases}
\]

Fail

\[
M \parallel F, C \implies \text{fail} \quad \text{if} \begin{cases} 
  M \models \neg C, \\
  \bullet \notin M
\end{cases}
\]

UnitPropagate

\[
M \parallel F, C \lor l \implies M l \parallel F, C \lor l \quad \text{if} \begin{cases} 
  M \models \neg C, \\
  l \text{ is undefined in } M
\end{cases}
\]
Strategies for applying rules

- After each Decide or UnitPropagate should check for a conflicting clause, a clause $C$ for which

$$M \models \neg C.$$  

If there is a conflicting clause, Backtrack or Fail are applied immediately to avoid pointless search.

- UnitPropagate applied with higher priority than Decide since it does not introduce branching in search
  - Typically many UnitPropagate applications for each Decide
  - BCP (Boolean Constraint Propagation): repeated application of UnitPropagate
Are many heuristics for choosing literal \( l \) in \textit{Decide} rule.

- **DLIS (Dynamic Largest Individual Sums):** choose the unassigned literal that satisfies the largest number of currently unsatisfied clauses.
- **MOMS:** choose literal with the Maximum number of Occurrences in Minimum Size clauses.
- **VSIDS (Variable State Independent Decaying Sum):** choose literal that has most frequently been involved in recent conflict clauses.

Heuristics striving for choice with maximum impact
## Example execution

<table>
<thead>
<tr>
<th>$M$</th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$C_3$</th>
<th>$C_4$</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>$\bar{x}_1 \lor x_2$</td>
<td>$\bar{x}_3 \lor x_4$</td>
<td>$\bar{x}_5 \lor \bar{x}_6$</td>
<td>$x_6 \lor \bar{x}_5 \lor \bar{x}_2$</td>
<td>$u \ u \ u \ u$</td>
</tr>
<tr>
<td>$\bullet x_1$</td>
<td>$0 \ u$</td>
<td>$u \ u$</td>
<td>$u \ u$</td>
<td>$u \ u \ u$</td>
<td>Decide $x_1$</td>
</tr>
<tr>
<td>$\bullet x_1 x_2$</td>
<td>$0 \ 1$</td>
<td>$u \ u$</td>
<td>$u \ u$</td>
<td>$u \ u \ 0$</td>
<td>Decide $x_3$</td>
</tr>
<tr>
<td>$\bullet x_1 x_2 \bullet x_3$</td>
<td>$0 \ 1$</td>
<td>$0 \ u$</td>
<td>$u \ u$</td>
<td>$u \ u \ 0$</td>
<td>UnitProp $C_1$</td>
</tr>
<tr>
<td>$\bullet x_1 x_2 \bullet x_3 x_4$</td>
<td>$0 \ 1$</td>
<td>$0 \ 1$</td>
<td>$u \ u$</td>
<td>$u \ u \ 0$</td>
<td>UnitProp $C_2$</td>
</tr>
<tr>
<td>$\bullet x_1 x_2 \bullet x_3 x_4 \bullet x_5$</td>
<td>$0 \ 1$</td>
<td>$0 \ 1$</td>
<td>$0 \ u$</td>
<td>$u \ 0 \ 0$</td>
<td>Decide $x_5$</td>
</tr>
<tr>
<td>$\bullet x_1 x_2 \bullet x_3 x_4 \bullet x_5 \bar{x}_6$</td>
<td>$0 \ 1$</td>
<td>$0 \ 1$</td>
<td>$0 \ 1$</td>
<td>$0 \ 0 \ 0$</td>
<td>UnitProp $C_3$</td>
</tr>
<tr>
<td>$\bullet x_1 x_2 \bullet x_3 x_4 \bar{x}_5$</td>
<td>$0 \ 1$</td>
<td>$0 \ 1$</td>
<td>$1 \ u$</td>
<td>$u \ 1 \ 0$</td>
<td>Backtrack $C_4$</td>
</tr>
<tr>
<td>$\bullet x_1 x_2 \bullet x_3 x_4 \bar{x}_5 \bar{x}_6$</td>
<td>$0 \ 1$</td>
<td>$0 \ 1$</td>
<td>$1 \ 1$</td>
<td>$0 \ 1 \ 0$</td>
<td>Decide $\bar{x}_6$</td>
</tr>
</tbody>
</table>

- Last state here is final – no further rules apply
- Derivation shows that $C_1 \land C_2 \land C_3 \land C_4$ is satisfiable
- Final $M$ is a satisfying assignment
Implication graphs

An implication graph describes the dependencies between literals in an assignment

- 1 node per assigned literal
  - Node label \( l@i \) indicates literal \( l \) is assigned true at decision level \( i \).
- Roots of graph (nodes without in-edges) are literals in \( M_0 \) and decision literals
- \( I \) in-edges \( l_1 \rightarrow l, \ldots, l_n \rightarrow l \) added if unit propagation with clause \( \neg l_1 \lor \cdots \lor \neg l_n \lor l \) sets literal \( l \)
  - Each edge labelled with clause
  - Edges indicate that \((l_1 \land \cdots \land l_n) \Rightarrow l\)
- When current assignment is conflicting with conflicting clause \( \neg l_1 \lor \cdots \lor \neg l_n \), then conflict node \( \kappa \) and \( \kappa \) in-edges \( l_1 \rightarrow \kappa, \ldots, l_n \rightarrow \kappa \) added
  - Each edge labelled with conflicting clause
  - Edges indicate that \((l_1 \land \cdots \land l_n) \Rightarrow \text{false}\)
Partial Implication graph example

Only shows current decision-level nodes and immediately-preceding nodes.

\[
C_1 = \overline{a} \vee \overline{b} \vee c \\
C_2 = \overline{c} \vee d \\
C_3 = \overline{d} \vee \overline{f} \\
C_4 = \overline{d} \vee e \vee g \\
C_5 = f \vee \overline{g}
\]
Backjump clause inference

The implication graph enables inference of new clauses that are
1. entailed by the current formula $F$, and
2. conflicting clauses under the current assignment.

▶ Consider any cut of an implication graph with
  ▶ On right: conflicting node $\kappa$
  ▶ On left: decision literal for current level and all literals at lower levels

▶ If literals on immediate left of cut are $l_1, \ldots, l_n$, then can infer the new clause

\[(l_1 \land \cdots \land l_n) \Rightarrow \text{false}\]

or equivalently

\[\neg l_1 \lor \cdots \lor \neg l_n\]
Clause inference example

\[ C_1 = \overline{a} \lor \overline{b} \lor c \quad C_2 = \overline{c} \lor d \quad C_3 = \overline{d} \lor \overline{f} \]
\[ C_4 = \overline{d} \lor e \lor g \quad C_5 = f \lor \overline{g} \]

Decision literal \( \rightarrow \)
Backjump clause: \( \overline{b} \lor \overline{a} \lor e \) \( \overline{d} \lor e \)
Backjumping

If

- current assignment has form $M \cdot l \cdot N$,
- there is some conflicting clause under this assignment,
- an inferred clause has form $C' \lor l'$ where $l'$ is the only literal at the current decision level,
- all literals of $C'$ are assigned in $M$,

then it is legitimate to

- **backjump**, set the assignment to $M$, and
- noting that $C' \lor l'$ has exactly one literal unassigned in $M$, to apply unit propagation to extend the assignment to $M \cdot l'$.

The clause $C' \lor l'$ is called a **backjump clause** and the literal $l'$ is called a **unique implication point (UIP)**.

- One UIP is the decision literal from the current level
- More generally, a UIP is any literal at the current level that appears on every path from from the current decision literal to the conflict node $\kappa$.
- Often the UIP closest to $\kappa$ is chosen.
Backjump rule

Replaces and generalises Backtrack rule in modern DPLL implementations

Backjump

\[ M \bullet I \  N \parallel F, C \rightarrow M \  I' \parallel F, C \]\n
if

\[
M \bullet I \  N \models \neg C, \text{ and there is some clause } C' \lor I' \text{ such that:}
\]

- \( F, C \models C' \lor I' \),
- \( M \models \neg C' \),
- \( I' \) is undefined in \( M \), and
- \( I' \) or \( \neg I' \) occurs in \( F \) or in \( M \bullet I \  N \)

- \( C \) is the conflicting clause
- \( C' \lor I' \) is the backjump clause
Learning

Learn

\[ M \parallel F \implies M \parallel F, C \quad \text{if} \quad \begin{cases} \text{each atom of } C \text{ occurs in} \\ F \text{ or in } M, \\ F \models C \end{cases} \]

- Common \( C \) are backjump clauses from the Backjump rule.
- Learned clauses record information about parts of search space to be avoided in future search
- CDCL (Conflict Driven Clause Learning)
  \( = \) Backjump + Learn
Forgetting

\[ M \| F, C \implies M \| F \text{ if } F \models C \]

- Applied to $C$ considered less important.
- Essential for controlling growth of required storage.
- Performance can degrade as $F$ grows, so shrinking $F$ can improve performance.
Restarting

\[ M \parallel F \Rightarrow (\) \parallel F \]

- Only used if \( F \) grown using learning.
- Additional knowledge causes Decide heuristics to work differently and often explore search space in more compact way.
- To preserve completeness, applied repeatedly with increasing periodicity.
Why is DPLL correct? 1

Lemma (1 - nature of reachable states)

Assume \( (F_0 \parallel F \Rightarrow^* M \parallel F') \). then

1. \( F \) and \( F' \) are equivalent
2. If \( M \) is of the form \( M_0 \bullet l_1 M_1 \cdots \bullet l_n M_n \) where all \( M_i \) are free, then \( F, l_1, \ldots l_i \models M_i \) for all \( i \) in \( 0 \ldots n \).

Lemma (2 - nature of final states)

If \( (F_0 \parallel F \Rightarrow^* S) \) and \( S \) is final (no further transitions possible), then either

1. \( S = \text{fail} \), or
2. \( S = M \parallel F' \) where \( M \models F \)
Why is DPLL correct? 2

Lemma (3 - transition sequences never go on for ever)
Every derivation \( \parallel F \rightarrow S_1 \rightarrow S_2 \rightarrow \cdots \) is finite

Proof.
Given \( M \) of form \( M_0 \bullet M_1 \cdots \bullet M_n \) where all \( M_i \) are \( \bullet \) free, define the *rank of \( M \), \( \rho(M) \) as \( \langle r_0, r_1, \ldots, r_n \rangle \) where \( r_i = |M_i| \). Every derivation must be finite as each basic DPLL rule strictly increases the rank in a lexicographic order and the image of \( \rho \) is finite. \( \square \)
Why is DPLL correct? 3

Theorem (1 - termination in \textbf{fail} state)
If () || $F \Rightarrow^{*} S$ and $S$ is final, then
1. if $S$ is \textbf{fail}, then $F$ is unsatisfiable
2. if $F$ is unsatisfiable then $S$ is \textbf{fail}
Why is DPLL correct? 4

Proof.

1. We have ( \parallel F \implies^* M \parallel F' \implies \text{fail} ).

   By Fail rule definition, there is a $C \in F'$ s.t. $M \models \neg C$.

   Since $M$ is $\bullet$ free, we have by Lemma 1(2) that $F \models M$, and therefore $F \models \neg C$.

   However, $F' \models C$ and by Lemma 1(1) $F \models C$.

   Hence, $F$ must be unsatisfiable.

2. By Lemmas 2 and 3.
Abstract DPLL modulo theories

Start just with one theory $T$. E.g.

- Equality with uninterpreted functions
- Linear arithmetic over $\mathbb{Z}$ or $\mathbb{R}$.

Propositional atoms now both

- Propositional symbols
- Atomic relations over $T$ involving individual expressions.
  E.g. $f(g(a)) = b$ or $3a + 5b \leq 7$.

Previous rules (e.g. Decide, UnitPropagate) and $\models$ (propositional entailment) treat syntactically distinct atoms as distinct

New rules involve $\models_T$ (entailment in theory $T$)

$\models_T$ is more general.
E.g. $\models_T x \leq 2 \lor x \geq 1$ but $\not\models x \leq 2 \lor x \geq 1$
Theory learning

**T-Learn**

\[ M \parallel F \iff M \parallel F, C \quad \text{if} \]

- each atom of \( C \) occurs in \( F \) or in \( M \),
- \( F \models_T C \)

- One use is for catching when \( M \) is inconsistent from \( T \) point of view.
  - Say \( \{l_1, \ldots, l_n\} \subseteq M \) such that \( F \models_T (l_1 \land \cdots \land l_n) \Rightarrow \text{false} \)
  - Then add \( C = \neg l_1 \lor \cdots \lor \neg l_n \)
  - As \( C \) is conflicting, the Backjump or Fail rule is enabled
  - Theory solvers can identify unsat cores, small subsets of literals sufficient for creating a conflicting clause

- Frequency of checks \( F \models_T C \) needs careful regulation, as cost might be far higher than basic DPLL steps.

- Given size of \( F \) often just check \( \models_T C \). In this case \( C \) is called a **theory lemma**.
Guiding growth of $M$ rather than just detecting when it is $T$-inconsistent.

TheoryPropagate

\[ M \parallel F \quad \Rightarrow \quad M \parallel F \quad \text{if} \quad \begin{cases} M \models_T l, \\ l \text{ or } \neg l \text{ occurs in } F \\ l \text{ is undefined in } M \end{cases} \]

- If applied well, can dramatically increase performance
- Worth applying exhaustively in some cases before resorting to Decide
Integration of SAT and theory solvers

Further new rules **T-Backjump** and **T-Forget** which generalise **Backjump** and **Forget** are also needed.

Use of theory-sensitive rules rules requires close integration of SAT and theory solvers

- SAT solvers need modification to be able to call out to theory solvers
- Useful to have theory solvers *incremental*, able to be rerun efficiently when input is some small increment on previous input
  - Also theory solvers need to support efficient retraction of blocks of input to cope with backjumping
Handling multiple theories

Consider formula $F$ mixing theories of linear real arithmetic and uninterpreted functions:

\[
f(x_1, 0) \geq x_3 \land f(x_2, 0) \leq x_3 \land \\
x_1 \geq x_2 \land x_2 \geq x_2 \land \\
x_3 - f(x_1, 0) \geq 1
\]

The popular Nelson-Oppen combination procedure involves first purifying, adding additional variables and creating an equisatisfiable formula with each atom over just one of the theories.

Formula $F$ above is equisatisfiable with $F_1 \land F_2$, where

\[
F_1 = a_1 \geq x_3 \land a_2 \leq x_3 \land x_1 \geq x_2 \land x_2 \geq x_1 \land \\
x_3 - a_1 \geq 1 \land a_0 = 0
\]

\[
F_2 = a_1 = f(x_1, a_0) \land a_2 = f(x_2, a_0)
\]

$F_1$ just involves linear real arithmetic and $F_2$ just involves an uninterpreted function
Nelson-Oppen example

Separate theory solvers can work on $F_1$ and $F_2$, exchanging equalities

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R arith</td>
<td>EUF</td>
</tr>
<tr>
<td>Original $F_i$</td>
<td>$a_1 \geq x_3$</td>
<td>$a_1 = f(x_1, a_0)$</td>
</tr>
<tr>
<td></td>
<td>$a_2 \leq x_3$</td>
<td>$a_2 = f(x_2, a_0)$</td>
</tr>
<tr>
<td></td>
<td>$x_1 \geq x_2$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$x_2 \geq x_1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$x_3 - a_1 \geq 1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$a_0 = 0$</td>
<td></td>
</tr>
<tr>
<td>Deduced atoms</td>
<td>$x_1 = x_2$ (*)&amp; $x_1 = x_2$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$a_1 = a_2$</td>
<td>$a_1 = a_2$ (*)</td>
</tr>
<tr>
<td></td>
<td>$a_1 = x_3$ (<em>)&amp; false(</em>)</td>
<td></td>
</tr>
</tbody>
</table>

The (*) marks indicate when inference is in the respective theory
Nelson-Oppen

The basic Nelson-Oppen procedure relies on each theory $T$ being combined being **convex**:

For any set of literals $L$, if $L \models_T s_1 = t_1 \lor \cdots \lor s_n = t_n$ then $L \models_T s_i = t_i$ for some $i$.

- Linear real arithmetic and EUF (Equality and Uninterpreted Functions) are convex.

- Linear integer arithmetic and bit-vector theories are not.

  If $L$ is $\{0 \leq x, x \leq 1\}$, then $L \models_{\mathbb{Z}} x = 0 \lor x = 1$, but $L \not\models_{\mathbb{Z}} x = 0$ and $L \not\models_{\mathbb{Z}} x = 1$.

Extensions of Nelson-Oppen can handle a number of non-convex theories.

In general, a combination of decidable theories might be undecidable.
Further reading

1. *Solving SAT and SAT Modulo Theories: From an Abstract Davis–Putnam–Logemann–Loveland Procedure to DPLL(T)*

   Main source for Abstract DPLL approach adopted in slides

2. Slides and videos from the 2012 SAT/SMT Summer School

   Tinelli’s presentation uses refined version of Abstract DPLL

3. SAT/SMT/AR/CP Summer Schools, 2011-2022

   See later schools for an introduction to recent work and applications.


   Online from Learn Resource List.

   Additional source for slides. Does not do Abstract DPLL.

   Good reference for recent work.