SAT and SMT algorithms

Paul Jackson

School of Informatics
University of Edinburgh

Formal Verification
Spring 2017
Basic question

Given a propositional logic formula, is it satisfiable?
Standard to always put formulas into Conjunctive Normal Form or CNF.

▶ By introducing new variables this can be done with only constant-factor growth in formula size.

Terminology

▶ An atom \( p \) is a propositional symbol
▶ 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 \( C_1 \land \ldots \land C_m \).
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:

\( \text{fail} \) or \( M \parallel F \)

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 \( \bullet \) 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 \( M \)
- \( M^{[i]} = M_0 \bullet \cdots \bullet M_i \)
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

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

UnitPropagate

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

Fail

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

Backtrack

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

- Are many heuristics for choosing literal $l$ in Decide rule.
  - E.g. MOMS: choose the literal with the Maximum number of Occurrences in Minimum Size clauses.

- UnitPropagate applied with higher priority than Decide since it does not introduce branching in search.

- 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.
## Example execution

<table>
<thead>
<tr>
<th>$M$</th>
<th>$C_1 \overline{x}_1 \lor x_2$</th>
<th>$C_2 \overline{x}_3 \lor x_4$</th>
<th>$C_3 \overline{x}_5 \lor \overline{x}_6$</th>
<th>$C_4 x_6 \lor \overline{x}_5 \lor \overline{x}_2$</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>$u$</td>
<td>$u$</td>
<td>$u$</td>
<td>$u$</td>
<td>Decide $x_1$</td>
</tr>
<tr>
<td>$\bullet x_1$</td>
<td>$0$</td>
<td>$u$</td>
<td>$u$</td>
<td>$u$</td>
<td>UnitProp $C_1$</td>
</tr>
<tr>
<td>$\bullet x_1 x_2$</td>
<td>$0$</td>
<td>$1$</td>
<td>$u$</td>
<td>$u$</td>
<td>Decide $x_3$</td>
</tr>
<tr>
<td>$\bullet x_1 x_2 \bullet x_3$</td>
<td>$0$</td>
<td>$1$</td>
<td>$0$</td>
<td>$u$</td>
<td>UnitProp $C_2$</td>
</tr>
<tr>
<td>$\bullet x_1 x_2 \bullet x_3 x_4$</td>
<td>$0$</td>
<td>$1$</td>
<td>$0$</td>
<td>$1$</td>
<td>Decide $x_5$</td>
</tr>
<tr>
<td>$\bullet x_1 x_2 \bullet x_3 x_4 \bullet x_5$</td>
<td>$0$</td>
<td>$1$</td>
<td>$0$</td>
<td>$1$</td>
<td>UnitProp $C_3$</td>
</tr>
<tr>
<td>$\bullet x_1 x_2 \bullet x_3 x_4 \bullet x_5 \overline{x}_6$</td>
<td>$0$</td>
<td>$1$</td>
<td>$0$</td>
<td>$1$</td>
<td>Backtrack $C_4$</td>
</tr>
<tr>
<td>$\bullet x_1 x_2 \bullet x_3 x_4 \overline{x}_5$</td>
<td>$0$</td>
<td>$1$</td>
<td>$0$</td>
<td>$u$</td>
<td>Decide $\overline{x}_6$</td>
</tr>
<tr>
<td>$\bullet x_1 x_2 \bullet x_3 x_4 \overline{x}_5 \overline{x}_6$</td>
<td>$0$</td>
<td>$1$</td>
<td>$0$</td>
<td>$1$</td>
<td></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 \( l \oplus i \) 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
- Edges \( l_1 \rightarrow l, \cdots, l_n \rightarrow l \) 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
- When current assignment is conflicting with conflicting clause \( \neg l_1 \lor \cdots \lor \neg l_n \), then conflict node \( \kappa \) and edges \( l_1 \rightarrow l, \cdots, l_n \rightarrow l \) are added
  - Each edge labelled with conflicting clause
Partial Implication graph example

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

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

\[ C_4 = \overline{d} \lor e \lor g \quad C_5 = f \lor \overline{g} \]
Backjump clause inference

The implication graph enables inference of new clauses entailed by the current formula $F$ and made false by 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
  \[\neg l_1 \lor \cdots \lor \neg l_n\]
**Backjump clause inference**

If

- current assignment has form $M \bullet l N$, and
- the inferred clause has form $C' \lor l'$ where $l'$ is the only literal at the current decision level, and
- 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 $M l'$.

Such a clause $C' \lor l'$ is called a backjump clause.

A backjump clause can always be formed using the decision literal from the current level.

Smaller backjump clauses can sometimes be discovered that exploit unique implication points (UIPs), literals on every path from the current decision literal to the conflict node $\kappa$. 
Backjump 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

Cut 1

Cut 2

Backjump clause: \( \overline{b} \lor \overline{a} \lor e \)

Backjump clause: \( \overline{d} \lor e \)
Backjumping rule

Replaces and generalises Backtrack rule in modern DPLL implementations

Backjump

\[ M \bullet l \, l' \parallel F, C \implies M \bullet l' \parallel F, C \]

if

\[
\begin{align*}
M \bullet l \, l' \parallel F, C \implies M \bullet l' \parallel F, C & \quad \text{if} \\
M \bullet l \, l' \parallel F, C & \equiv \neg C, \text{ and there is some clause } C' \lor l' \text{ such that:} \\
- F, C & \models C' \lor l', \\
- M & \models \neg C', \\
- l' & \text{ is undefined in } M, \\
& \text{ and} \\
- l' \text{ or } \neg l' & \text{ occurs in } F \text{ or in } M \bullet l \, l' \parallel F, C
\end{align*}
\]

- \( C \) is the conflicting clause
- \( C' \lor l' \) is the backjump clause
Learning and forgetting

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.

Forget

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

- 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

**Restart**

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

- 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 \( F \mathrel{=\!\!\!\implies}^* M \mathrel{\parallel} F' \). then

1. \( F \) and \( F' \) are equivalent
2. If \( M \) is of the form \( M_0 \mathrel{\bullet} l_1 M_1 \cdots \mathrel{\bullet} l_n M_n \) where all \( M_i \) are \( \bullet \) 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 \( F \mathrel{=\!\!\!\implies}^* S \) and \( S \) is final (no further transitions possible), then either

1. \( S = \text{fail} \), or
2. \( S = M \mathrel{\parallel} F' \) 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 fail state)

If () || $F \implies^* S$ and $S$ is final, then

1. if $S$ is fail, then $F$ is unsatisfiable
2. if $F$ is unsatisfiable then $S$ is fail

Proof.

1. We have () || $F \implies^* M$ || $F' \implies$ fail. By Fail rule definition, there is a $C \in F'$ s.t. $M \models \neg C$. Since $M$ is • 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 Lemma 2.
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 $T$)
Theory learning

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

- One use is for catching when \( M \) is inconsistent from \( T \) point of view.
  - Say \( \{l_1, \ldots, l_n\} \subseteq M \) such that \( F \models l_1 \land \cdots \land l_n \Rightarrow \text{false} \)
  - 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.
Theory propagation

Guiding growth of $M$ rather than just detecting when it is $T$-inconsistent.

TheoryPropagate

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

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

Use of $T$-Learn and TheoryPropagate 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 need ability to efficiently retract 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>$x_1 = x_2$</td>
<td>$x_1 = x_2$</td>
</tr>
<tr>
<td></td>
<td>$a_1 = a_2$</td>
<td>$a_1 = a_2$</td>
</tr>
<tr>
<td></td>
<td>$a_1 = x_3$</td>
<td>$a_1 = a_2$</td>
</tr>
<tr>
<td></td>
<td>$\text{false}$</td>
<td>$\text{false}$</td>
</tr>
</tbody>
</table>

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

The basic Nelson-Oppen procedure relies on combined theories being convex.

- Linear real arithmetic and EUF (Equality and Uninterpreted Functions) are convex.
- Linear integer arithmetic and bit-vector theories are not.

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

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


   These slides draw mainly on 3 and part of 2. Tinelli’s presentation in 4 also expands on the Abstract DPLL approach to SAT and SMT.