SAT Solving

Testing, Quality Assurance, and Maintenance
Winter 2018

Prof. Arie Gurfinkel

based on slides by Prof. Ruzica Piskac, Nikolaj Bjorner, and others
Boolean Satisfiability (CNF-SAT)

Let $V$ be a set of variables

A *literal* is either a variable $v$ in $V$ or its negation $\sim v$

A *clause* is a disjunction of literals

- e.g., $(v_1 \lor \sim v_2 \lor v_3)$

A Boolean formula in *Conjunctive Normal Form* (CNF) is a conjunction of clauses

- e.g., $(v_1 \lor \sim v_2) \land (v_3 \lor v_2)$

An *assignment* $s$ of Boolean values to variables *satisfies* a clause $c$ if it evaluates at least one literal in $c$ to true

An assignment $s$ *satisfies* a formula $C$ in CNF if it satisfies every clause in $C$

Boolean Satisfiability Problem (CNF-SAT):

- determine whether a given CNF $C$ is satisfiable
Algorithms for SAT

SAT is NP-complete

DPLL (Davis-Putnam-Logemann-Loveland, ‘60)
• smart enumeration of all possible SAT assignments
• worst-case EXPTIME
• alternate between deciding and propagating variable assignments

CDCL (GRASP ‘96, Chaff ‘01)
• conflict-driven clause learning
• extends DPLL with
  – smart data structures, backjumping, clause learning, heuristics, restarts…
• scales to millions of variables
Background Reading: SAT

COMMUNICATIONS
OF THE
ACM

Home / Magazine Archive / August 2009 (Vol. 52, No. 8) / Boolean Satisfiability: From Theoretical Hardness... / Full Text

REVIEW ARTICLES

Boolean Satisfiability: From Theoretical Hardness to Practical Success

By Sharad Malik, Lintao Zhang
Communications of the ACM, Vol. 52 No. 8, Pages 76-82
10.1145/1536615.1536637

There are many practical situations where we need to satisfy several potentially conflicting constraints. Simple examples of this abound in daily life, for example, determining a schedule for a series of games that resolves the availability of players and venues, or finding a seating assignment at dinner consistent with various rules the host would like to impose. This also applies to applications in computing, for example, ensuring that a hardware/software system functions correctly with its overall behavior constrained by the behavior of its components and their composition, or finding a plan for a robot to reach a goal that is
Some Experience with SAT Solving

Speed-up of 2012 solver over other solvers

![Graph showing speed-up of 2012 solver over other solvers.](image)

Solver

### SAT - Milestones

Problems impossible 10 years ago are trivial today.

<table>
<thead>
<tr>
<th>Year</th>
<th>Milestone</th>
</tr>
</thead>
<tbody>
<tr>
<td>1960</td>
<td>Davis-Putnam procedure</td>
</tr>
<tr>
<td>1962</td>
<td>Davis-Logeman-Loveland</td>
</tr>
<tr>
<td>1984</td>
<td>Binary Decision Diagrams</td>
</tr>
<tr>
<td>1992</td>
<td>DIMACS SAT challenge</td>
</tr>
<tr>
<td>1994</td>
<td>SATO: clause indexing</td>
</tr>
<tr>
<td>1997</td>
<td>GRASP: conflict clause learning</td>
</tr>
<tr>
<td>1998</td>
<td>Search Restarts</td>
</tr>
<tr>
<td>2001</td>
<td>zChaff: 2-watch literal, VSIDS</td>
</tr>
<tr>
<td>2005</td>
<td>Preprocessing techniques</td>
</tr>
<tr>
<td>2007</td>
<td>Phase caching</td>
</tr>
<tr>
<td>2008</td>
<td>Cache optimized indexing</td>
</tr>
<tr>
<td>2009</td>
<td>In-processing, clause management</td>
</tr>
<tr>
<td>2010</td>
<td>Blocked clause elimination</td>
</tr>
</tbody>
</table>

![Graph showing SAT competition results](image)

Concept: 2002 - 2010

Millions of variables from HW designs

Courtesy Daniel le Berre
DPLL PROCEDURE
Decision Procedure for Satisfiability

Algorithm that in some finite amount of computation decides if a given propositional logic (PL) formula F is satisfiable

• NP-complete problem

Modern decision procedures for PL formulae are called SAT solvers

Naïve approach

• Enumerate models (i.e., truth tables)
• Enumerate resolution proofs

Modern SAT solvers

• DPLL algorithm
  – Davis-Putnam-Logemann-Loveland
• Combines model- and proof-based search
• Operates on Conjunctive Normal Form (CNF)
Propositional Resolution

Given two clauses \((C, p)\) and \((D, !p)\) that contain a literal \(p\) of different polarity, create a new clause by taking the union of literals in \(C\) and \(D\):

\[
\text{Res} \{\{C, p\}, \{D, !p\}\} = \{C, D\}
\]
SAT solving by resolution (DP)

Assume that input formula $F$ is in CNF

1. Pick two clauses $C_1$ and $C_2$ in $F$ that can be resolved
2. If the resolvent $C$ is an empty clause, return UNSAT
3. Otherwise, add $C$ to $F$ and go to step 1
4. If no new clauses can be resolved, return SAT

**Termination:** finitely many derived clauses
DPLL: David Putnam Logemann Loveland

Combines pure resolution-based search with case splitting on decisions
Proof search is restricted to unit resolution
  • can be done very efficiently (polynomial time)
Case split restores completeness

DPLL can be described by the following two rules
  • F is the input formula in CNF

\[
\frac{F}{F, p \mid F, \neg p} \quad \text{split} \quad p \text{ and } \neg p \text{ are not in } F
\]

\[
\frac{F, C \lor \ell, \neg \ell}{F, C, \neg \ell} \quad \text{unit}
\]

The original DPLL procedure

Incrementally builds a satisfying truth assignment $M$ for the input CNF formula $F$

$M$ is grown by

- deducing the truth value of a literal from $M$ and $F$, or
- guessing a truth value

If a wrong guess for a literal leads to an inconsistency, the procedure backtracks and tries the opposite value.
DPLL: Illustration

\[ M \, | \, F \]

- Partial model
- Set of clauses
DPLL: Illustration

Guessing

\[ p \mid p \lor q, \neg q \lor r \]

\[ p, \neg q \mid p \lor q, \neg q \lor r \]
DPLL: Illustration

Deducing

\[ p \mid p \lor q, \neg p \lor s \]

\[ p, s \mid p \lor q, \neg p \lor s \]
DPLL: Illustration

Backtracking

\[ p, \neg s, q \mid p \lor q, s \lor q, \neg p \lor \neg q \]

\[ p, s \mid p \lor q, s \lor q, \neg p \lor \neg q \]
Pure Literals

A literal is pure if only occurs positively or negatively.

Example:

\[ \varphi = (\neg x_1 \lor x_2) \land (x_3 \lor \neg x_2) \land (x_4 \lor \neg x_5) \land (x_5 \lor \neg x_4) \]

\( \neg x_1 \) and \( x_3 \) are pure literals

Pure literal rule:
Clauses containing pure literals can be removed from the formula (i.e. just satisfy those pure literals)

\[ \varphi_{\neg x_1, x_3} = (x_4 \lor \neg x_5) \land (x_5 \lor \neg x_4) \]

Preserve satisfiability, not logical equivalency!
DPLL (as a procedure)

- Standard backtrack search
- **DPLL(F):**
  - Apply unit propagation
  - If conflict identified, return **UNSAT**
  - Apply the pure literal rule
  - If F is satisfied (empty), return **SAT**
  - Select decision variable \( x \)
    - If \( \text{DPLL}(F \land x) = \text{SAT} \) return **SAT**
    - return \( \text{DPLL}(F \land \neg x) \)
### The Original DPLL Procedure – Example

<table>
<thead>
<tr>
<th>assign</th>
<th>1 ∨ 2, 2 ∨ ¬3 ∨ 4, ¬1 ∨ ¬2, ¬1 ∨ ¬3 ∨ ¬4, 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deduce 1</td>
<td>1</td>
</tr>
<tr>
<td>Deduce ¬2</td>
<td>1, 2</td>
</tr>
<tr>
<td>Guess 3</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>Deduce 4</td>
<td>1, 2, 3, 4</td>
</tr>
<tr>
<td>Conflict</td>
<td>1 ∨ 2, 2 ∨ ¬3 ∨ 4, ¬1 ∨ ¬2, ¬1 ∨ ¬3 ∨ ¬4, 1</td>
</tr>
</tbody>
</table>
The Original DPLL Procedure – Example

assign

Deduce 1
1
Deduce ¬2
1, 2

Guess 3
1, 2, 3
Deduce 4
1, 2, 3, 4

Undo 3

1 ∨ 2, 2 ∨ ¬3 ∨ 4, ¬1 ∨ ¬2,
¬1 ∨ ¬3 ∨ ¬4, 1

1 ∨ 2, 2 ∨ ¬3 ∨ 4, ¬1 ∨ ¬2,
¬1 ∨ ¬3 ∨ ¬4, 1

1 ∨ 2, 2 ∨ ¬3 ∨ 4, ¬1 ∨ ¬2,
¬1 ∨ ¬3 ∨ ¬4, 1

1 ∨ 2, 2 ∨ ¬3 ∨ 4, ¬1 ∨ ¬2,
¬1 ∨ ¬3 ∨ ¬4, 1

1 ∨ 2, 2 ∨ ¬3 ∨ 4, ¬1 ∨ ¬2,
¬1 ∨ ¬3 ∨ ¬4, 1

1 ∨ 2, 2 ∨ ¬3 ∨ 4, ¬1 ∨ ¬2,
¬1 ∨ ¬3 ∨ ¬4, 1

1 ∨ 2, 2 ∨ ¬3 ∨ 4, ¬1 ∨ ¬2,
¬1 ∨ ¬3 ∨ ¬4, 1
The Original DPLL Procedure – Example

<table>
<thead>
<tr>
<th>assign</th>
<th>1 v 2, 2 v ¬3 v 4, ¬1 v ¬2, ¬1 v ¬3 v ¬4, 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deduce 1</td>
<td>1</td>
</tr>
<tr>
<td>Deduce ¬2</td>
<td>1 v 2, 2 v ¬3 v 4, ¬1 v ¬2, ¬1 v ¬3 v ¬4, 1</td>
</tr>
<tr>
<td>1, 2</td>
<td>1 v 2, 2 v ¬3 v 4, ¬1 v ¬2, ¬1 v ¬3 v ¬4, 1</td>
</tr>
</tbody>
</table>

Guess ¬3

| 1, 2, 3 | 1 v 2, 2 v ¬3 v 4, ¬1 v ¬2, ¬1 v ¬3 v ¬4, 1 |

Model Found
An Abstract Framework for DPLL

The DPLL procedure can be described declaratively by simple sequent-style calculi.

Such calculi, however, cannot model meta-logical features such as backtracking, learning, and restarts.

We model DPLL and its enhancements as transition systems instead.

A transition system is a binary relation over states, induced by a set of conditional transition rules.
An Abstract Framework for DPLL

State
• **fail** or \( M \parallel F \)
• where
  – \( F \) is a CNF formula, a set of clauses, and
  – \( M \) is a sequence of annotated literals denoting a partial truth assignment

Initial State
• \( \emptyset \parallel F \), where \( F \) is to be checked for satisfiability

Expected final states:
• **fail** if \( F \) is unsatisfiable
• \( M \parallel G \)
  where
  – \( M \) is a model of \( G \)
  – \( G \) is logically equivalent to \( F \)
Transition Rules for DPLL

Extending the assignment:

**UnitProp**  \[ M \parallel F, C \lor l \rightarrow M \parallel F, C \lor l \]

\[ M \models \neg C \]
\[ l \text{ is undefined in } M \]

\[ l \text{ or } \neg l \text{ occur in } C \]
\[ l \text{ is undefined in } M \]

**Decide**  \[ M \parallel F, C \rightarrow M l^d \parallel F, C \]

Notation: \( l^d \) is a decision literal
Transition Rules for DPLL

Repairing the assignment:

**Fail**

\[ M \parallel F, C \rightarrow \text{fail} \]

\[ M \models \neg C \]

M does not contain decision literals

**Backtrack**

\[ M \mathcal{I}^d N \parallel F, C \rightarrow M \neg I \parallel F, C \]

\[ M \mathcal{I}^d N \models \neg C \]

I is the last decision literal
### Transition Rules DPLL – Example

<table>
<thead>
<tr>
<th>Transition</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1, 2</td>
<td></td>
</tr>
<tr>
<td>1, 2, 3d</td>
<td></td>
</tr>
<tr>
<td>1, 2, 3d, 4</td>
<td></td>
</tr>
</tbody>
</table>

- **UnitProp**: 1, 2
- **Decide**: 3
- **UnitProp**: 4
- **Backtrack**: 3
Transition Rules DPLL – Example

\[ \emptyset \vdash 1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1 \]

1 \vdash 1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1

1, 2 \vdash 1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1

1, 2, 3^d \vdash 1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1

1, 2, 3 \vdash 1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1

UnitProp 1
UnitProp \neg 2
Decide 3
UnitProp 4
Backtrack 3
Transition Rules for DPLL (on one slide)

- **UnitProp**
  \[ M \parallel F, C \lor I \rightarrow M \parallel F, C \lor I \]
  - \( M \models \neg C \)
  - \( I \) is undefined in \( M \)

- **Decide**
  \[ M \parallel F, C \rightarrow M \parallel F, C \]
  - \( I \) or \( \neg I \) occur in \( C \)
  - \( I \) is undefined in \( M \)

- **Fail**
  \[ M \parallel F, C \rightarrow \text{fail} \]
  - \( M \models \neg C \)
  - \( M \) does not contain decision literals

- **Backtrack**
  \[ M \parallel F, C \rightarrow M \parallel F, C \]
  - \( M \models \neg C \)
  - \( I \) is the last decision literal
The DPLL System – Correctness

Some terminology

• Irreducible state: state to which no transition rule applies.
• Execution: sequence of transitions allowed by the rules and starting with states of the form $\emptyset \parallel F$.
• Exhausted execution: execution ending in an irreducible state

**Proposition** (Strong Termination) Every execution in DPLL is finite

**Proposition** (Soundness) For every exhausted execution starting with $\emptyset \parallel F$ and ending in $M \parallel F$, $M \models F$

**Proposition** (Completeness) If $F$ is unsatisfiable, every exhausted execution starting with $\emptyset \parallel F$ ends with fail

Maintained in more general rules + theories
Modern DPLL: CDCL

**Conflict Driven Clause Learning**

- two watched literals – efficient index to find clauses that can be used in unit resolution
- periodically restart backtrack search
- activity-based decision heuristic to choose decision variable
- conflict resolution via clausal learning

We will briefly look at clausal learning

More details on CDCL are available in

- Chapter 2 of Decision Procedures book
- [http://gauss.ececs.uc.edu/SAT/articles/FAIA185-0131.pdf](http://gauss.ececs.uc.edu/SAT/articles/FAIA185-0131.pdf)
Conflict Directed Clause Learning

Lemma learning

\[ \neg t, p, q, s \mid t \lor \neg p \lor q, \neg q \lor s, \neg p \lor \neg s \]

\[ \neg t, p, q, s \mid t \lor \neg p \lor q, \neg q \lor s, \neg p \lor \neg s \mid \neg p \lor \neg s \]

\[ \neg t, p, q, s \mid t \lor \neg p \lor q, \neg q \lor s, \neg p \lor \neg s \mid \neg p \lor \neg q \]

\[ \neg t, p, q, s \mid t \lor \neg p \lor q, \neg q \lor s, \neg p \lor \neg s \mid \neg p \lor t \]
Learned Clause by Resolution

A new clause is learned by resolving the conflicting clause with clauses deduced from the last decision

\[
\frac{t \lor \neg p \lor q \quad \neg q \lor s}{t \lor \neg p \lor s} \quad \frac{\neg p \lor \neg s}{\neg p \lor t}
\]
# Modern CDCL: Abstract Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Rule Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initialize</strong></td>
<td>$\epsilon</td>
</tr>
<tr>
<td></td>
<td>$F$ is a set of clauses</td>
</tr>
<tr>
<td><strong>Decide</strong></td>
<td>$M</td>
</tr>
<tr>
<td></td>
<td>$\ell$ is unassigned</td>
</tr>
<tr>
<td><strong>Propagate</strong></td>
<td>$M</td>
</tr>
<tr>
<td></td>
<td>$C$ is false under $M$</td>
</tr>
<tr>
<td><strong>Sat</strong></td>
<td>$M</td>
</tr>
<tr>
<td></td>
<td>$F$ true under $M$</td>
</tr>
<tr>
<td><strong>Conflict</strong></td>
<td>$M</td>
</tr>
<tr>
<td></td>
<td>$C$ is false under $M$</td>
</tr>
<tr>
<td><strong>Learn</strong></td>
<td>$M</td>
</tr>
<tr>
<td><strong>Unsat</strong></td>
<td>$M</td>
</tr>
<tr>
<td><strong>Backjump</strong></td>
<td>$MM'</td>
</tr>
<tr>
<td></td>
<td>$\bar{C} \subseteq M, \neg \ell \in M'$</td>
</tr>
<tr>
<td><strong>Resolve</strong></td>
<td>$M</td>
</tr>
<tr>
<td></td>
<td>$\ell^{C\lor\ell} \in M$</td>
</tr>
<tr>
<td><strong>Forget</strong></td>
<td>$M</td>
</tr>
<tr>
<td></td>
<td>$C$ is a learned clause</td>
</tr>
<tr>
<td><strong>Restart</strong></td>
<td>$M</td>
</tr>
<tr>
<td></td>
<td>[Nieuwenhuis, Oliveras, Tinelli J.ACM 06] customized</td>
</tr>
</tbody>
</table>
Conjunctive Normal Form

\[
\begin{align*}
\phi & \leftrightarrow \psi \quad \Rightarrow \text{CNF} \quad \phi \rightarrow \psi \land \psi \rightarrow \phi \\
\phi & \rightarrow \psi \quad \Rightarrow \text{CNF} \quad \neg \phi \lor \psi \\
\neg (\phi \lor \psi) & \quad \Rightarrow \text{CNF} \quad \neg \phi \land \neg \psi \\
\neg (\phi \land \psi) & \quad \Rightarrow \text{CNF} \quad \neg \phi \lor \neg \psi \\
\neg \neg \phi & \quad \Rightarrow \text{CNF} \quad \phi \\
(\phi \land \psi) \lor \xi & \quad \Rightarrow \text{CNF} \quad (\phi \lor \xi) \land (\psi \lor \xi)
\end{align*}
\]

Every propositional formula can be put in CNF

**PROBLEM:** (potential) exponential blowup of the resulting formula
Tseitin Transformation – Main Idea

Introduce a fresh variable $e_i$ for every subformula $G_i$ of $F$

- intuitively, $e_i$ represents the truth value of $G_i$

Assert that every $e_i$ and $G_i$ pair are equivalent

- $e_i \leftrightarrow G_i$
- and express the assertion as CNF

Conjoin all such assertions in the end
Formula to CNF Conversion

```python
def cnf (ϕ):
    p, F = cnf_rec (ϕ)
    return p ∧ F

def cnf_rec (ϕ):
    if is_atomic (ϕ): return (ϕ, True)
    elif ϕ == ψ ∧ ξ:
        q, F₁ = cnf_rec (ψ)
        r, F₂ = cnf_rec (ξ)

        p = mk_fresh_var ()
        # C is CNF for p↔(q∧r)
        C = (¬pVq)∧(¬pVr)∧(pV¬qV¬r)
        return (p, F₁∧F₂∧C)
    elif ϕ == ψVξ:
        ...
```

**Exercise:** Complete cases for

\[ φ = ψVξ, \quad φ = ¬ψ, \quad φ = ψ↔ξ \]
Tseitin Transformation: Example

\[ G : p \leftrightarrow (q \rightarrow r) \]

\[ G : e_0 \land (e_0 \leftrightarrow (p \leftrightarrow e_1)) \land (e_1 \leftrightarrow (q \rightarrow r)) \]

\[ e_1 \leftrightarrow (q \rightarrow r) \]
\[ = (e_1 \rightarrow (q \rightarrow r)) \land ((q \rightarrow r) \rightarrow e_1) \]
\[ = (\neg e_1 \lor \neg q \lor r) \land ((\neg q \lor r) \rightarrow e_1) \]
\[ = (\neg e_1 \lor \neg q \lor r) \land (\neg q \rightarrow e_1) \land (r \rightarrow e_1) \]
\[ = (\neg e_1 \lor \neg q \lor r) \land (q \lor e_1) \land (\neg r \lor e_1) \]
Tseitin Transformation: Example

\[ G : p \leftrightarrow (q \rightarrow r) \]

\[ G : e_0 \land (e_0 \leftrightarrow (p \leftrightarrow e_1)) \land (e_1 \leftrightarrow (q \rightarrow r)) \]

\[ e_0 \leftrightarrow (p \leftrightarrow e_1) \]
\[ = (e_0 \rightarrow (p \leftrightarrow e_1)) \land ((p \leftrightarrow e_1)) \rightarrow e_0) \]
\[ = (e_0 \rightarrow (p \rightarrow e_1)) \land (e_0 \rightarrow (e_1 \rightarrow p)) \land \]
\[ (((p \land e_1) \lor (\neg p \land \neg e_1)) \rightarrow e_0) \]
\[ = (\neg e_0 \lor \neg p \lor e_1) \lor (\neg e_0 \lor \neg e_1 \lor p) \lor \]
\[ (\neg p \lor \neg e_1 \lor e_0) \lor (p \lor e_1 \lor e_0) \]
Tseitin Transformation: Example

\[ G : p \iff (q \rightarrow r) \]

\[ G : e_0 \land (e_0 \leftrightarrow (p \leftrightarrow e_1)) \land (e_1 \leftrightarrow (q \rightarrow r)) \]

\[ G : e_0 \land (\neg e_0 \lor \neg p \lor e_1) \land (\neg e_0 \lor p \lor \neg e_1) \land (e_0 \lor p \lor e_1) \land (e_0 \lor \neg p \lor \neg e_1) \land (\neg e_1 \lor \neg q \lor r) \land (e_1 \lor q) \land (e_1 \lor \neg r) \]
Tseitin Transformation [1968]

Used in practice
- No exponential blow-up
- CNF formula size is linear with respect to the original formula

Does not produce an equivalent CNF

However, given F, the following holds for the computed CNF F’:
- F’ is equisatisfiable to F
- Every model of F’ can be translated (i.e., projected) to a model of F
- Every model of F can be translated (i.e., completed) to a model of F’

No model is lost or added in the conversion