SAT Solving

Testing, Quality Assurance, and Maintenance
Winter 2020

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-Logeman-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
Boolean Satisfiability: From Theoretical Hardness to Practical Success

By Sharad Malik, Lintao Zhang
Communications of the ACM, Vol. 52 No. 8, Pages 76-82
10.1145/1536616.1536637

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 or 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

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>

Concept:

Millions of variables from HW designs

2002 - 2010

Results of the SAT competition/race winners on the SAT 2009 application benchmarks, 20mn timeout

[Le Berre'10]

Courtesy Daniel le Berre
Davis Putnam Logemann Loveland

DPLL PROCEDURE
Chapter 2: Decision Procedures for Propositional Logic
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

\[
\begin{array}{c}
C \lor p \\
\hline
C \lor D
\end{array} \\
\begin{array}{c}
D \lor \neg p
\end{array}
\]

Res(\{C, p\}, \{D, \neg p\}) = \{C, D\}

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

Lemma:
Let $F$ be a CNF formula. Let $R$ be a resolvent of two clauses $X$ and $Y$ in $F$. Then, $F \cup \{R\}$ is equivalent to $F$

\[
\begin{align*}
C \lor p & \quad \text{D} \lor \neg p \\
\hline
C \lor D
\end{align*}
\]
Resolution Theorem

Let F be a set of clauses

\[ \text{Res}(F) = F \cup \{ R \mid R \text{ is a resolvent of two clauses in } F \} \]

\[ \text{Res}^0(F) = F \]
\[ \text{Res}^{n+1}(F) = \text{Res}(\text{Res}^n(F)), \text{ for } n \geq 0 \]
\[ \text{Res}^\ast(F) = \bigcup_{n \geq 0} \text{Res}^n(F) \]

**Theorem:** A CNF F is UNAT iff \( \text{Res}^\ast(F) \) contains an empty clause
Resolution Theorem

Let F be a set of clauses

\[ Res(F) = F \cup \{ R \mid R \text{ is a resolvent of two clauses in } F \} \]

\[ Res^0(F) = F \]

\[ Res^{n+1}(F) = Res(Res^n(F)), \text{ for } n \geq 0 \]

\[ Res^*(F) = \bigcup_{n \geq 0} Res^n(F) \]

**Theorem:** A CNF F is UNAT iff \( Res^*(F) \) contains an empty clause
Proof of the Resolution Theorem

(Soundness) By Resolution Lemma, F is equivalent to Res\(i\)(F) for any \(i\). Let \(n\) be such that Res\(^{n+1}\)(F) contains an empty clause, but Res\(^n\)(F) does not. Then Res\(^n\)(F) must contain to unit clauses L and ¬L. Hence, it is UNSAT.

(Completeness) By induction on the number of different atomic propositions in F.
Base case is trivial: F contains an empty clause.
IH: Assume F has atomic propositions A\(1\), … A\(_{n+1}\)
Let F\(_0\) be the result of replacing A\(_{n+1}\) by 0
Let F\(_1\) be the result of replacing A\(_{n+1}\) by 1
Apply IH to F\(_0\) and F\(_1\). Restore replaced literals. Combine the two resolutions.
Proof System

An inference rule is a tuple \((P_1, \ldots, P_n, C)\)
- where, \(P_1, \ldots, P_n, C\) are formulas
- \(P_i\) are called premises and \(C\) is called a conclusion
- intuitively, the rules says that the conclusion is true if the premises are

A proof system \(P\) is a collection of inference rules

A proof in a proof system \(P\) is a tree (or a DAG) such that
- nodes are labeled by formulas
- for each node \(n\), \((\text{parents}(n), n)\) is an inference rule in \(P\)
Propositional Resolution

\[ C \lor p \quad \frac{}{D \lor \neg p} \quad C \lor D \]

Propositional resolution is a sound inference rule.

Proposition resolution system consists of a single propositional resolution rule.
Example of a resolution proof

A refutation of \( \neg p \lor \neg q \lor r, p \lor r, q \lor r, \neg r \):
Resolution Proof Example

Show by resolution that the following CNF is UNSAT

\[ \neg b \land (\neg a \lor b \lor \neg c) \land a \land (\neg a \lor c) \]

1. \[ \neg a \lor b \lor \neg c \]
2. \[ a \]
   \[ \frac{\neg a \lor b \lor \neg c}{b \lor \neg c} \]
   \[ \frac{b \lor \neg c}{b} \]
   \[ \frac{a \quad \neg a \lor c}{c} \]

18
Entailment and Derivation

A set of formulas $F$ entails a set of formulas $G$ iff every model of $F$ and is a model of $G$

\[ F \models G \]

A formula $G$ is derivable from a formula $F$ by a proof system $P$ if there exists a proof whose leaves are labeled by formulas in $F$ and the root is labeled by $G$

\[ F \vdash_P G \]
Soundness and Completeness

A proof system $P$ is **sound** iff

$$(F \vdash_P G) \implies (F \models G)$$

A proof system $P$ is **complete** iff

$$(F \models G) \implies (F \vdash_P G)$$
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

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

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

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 (decide)

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

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

Deducing (unit propagate)

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

\[ p, s \mid p \lor q, \neg p \lor s \]
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 DPLL(F \land x) = SAT return SAT
    - return DPLL(F \land \neg x)
### The Original DPLL Procedure – Example

<table>
<thead>
<tr>
<th>assign</th>
<th>(1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deduce 1</td>
<td>(1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1)</td>
</tr>
<tr>
<td>1</td>
<td>(1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1)</td>
</tr>
<tr>
<td>Deduce (\neg 2)</td>
<td>(1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1)</td>
</tr>
<tr>
<td>1, 2</td>
<td>(1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1)</td>
</tr>
<tr>
<td>Guess 3</td>
<td>(1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1)</td>
</tr>
<tr>
<td>1, 2, 3</td>
<td>(1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1)</td>
</tr>
<tr>
<td>Deduce 4</td>
<td>(1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1)</td>
</tr>
<tr>
<td>1, 2, 3, 4</td>
<td>(1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1)</td>
</tr>
<tr>
<td>Conflict</td>
<td>(1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1)</td>
</tr>
</tbody>
</table>
**The Original DPLL Procedure – Example**

<table>
<thead>
<tr>
<th>assign</th>
<th>1</th>
<th>Deduce ( \neg 2 )</th>
<th>1, 2</th>
<th>Guess 3</th>
<th>1, 2, 3</th>
<th>Deduce 4</th>
<th>1, 2, 3, 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1, 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1, 2, 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1, 2, 3, 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

\[
1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \\
\neg 1 \lor \neg 3 \lor \neg 4, 1
\]
The Original DPLL Procedure – Example

<table>
<thead>
<tr>
<th>assign</th>
<th>1, 2, 3</th>
<th>1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deduce 1</td>
<td>1</td>
<td>1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1</td>
</tr>
<tr>
<td>Deduce \neg 2</td>
<td>1, 2</td>
<td>1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1</td>
</tr>
<tr>
<td>Guess \neg 3</td>
<td>1, 2, 3</td>
<td>1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1</td>
</tr>
</tbody>
</table>

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

**Decide** \( M \parallel F, C \rightarrow M I^d \parallel F, C \)  
\[ I \text{ or } \neg I \text{ occur in } C \] 
\( I \) is undefined in \( M \)

Notation: \( I^d \) 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: \[ M \upair N \parallel F, C \rightarrow M \neg I \parallel F, C \]

- \( M \upair N \models \neg C \)
- \( I \) is the last decision literal
Transition Rules DPLL – Example

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

1 \| 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 \| 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 \| 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, 4 \| 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 DPLL – Example

∅ ∥ 1 ∨ 2, 2 ∨ ¬3 ∨ 4, ¬1 ∨ ¬2, ¬1 ∨ ¬3 ∨ ¬4, 1

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

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

1, 2, 3^d ∥ 1 ∨ 2, 2 ∨ ¬3 ∨ 4, ¬1 ∨ ¬2, ¬1 ∨ ¬3 ∨ ¬4, 1

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

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

**UnitProp**  
\[ M \parallel F, C \lor l \rightarrow M \parallel F, C \lor l \]  
\[ M \models \neg C \]  
l is undefined in M

**Decide**  
\[ M \parallel F, C \rightarrow M \parallel^d F, C \]  
\[ l \text{ or } \neg l \text{ occur in } C \]  
l is undefined in M

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

**Backtrack**  
\[ M \parallel^d N \parallel F, C \rightarrow M \parallel^d \neg l \parallel F, C \]  
\[ M \parallel^d N \models \neg C \]  
l 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 $M \parallel F$, $M \vDash F$

**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

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

\[ \downarrow \]

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

\[ \downarrow \]

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

\[ \downarrow \]

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

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

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

\[
\begin{align*}
\neg p \lor \neg s & \quad \neg q \lor s \\
\hline \\
\neg p \lor \neg q & \quad t \lor \neg p \lor q \\
\hline \\
t \lor \neg p
\end{align*}
\]

Trivial Resolution: at every resolution step, at least one clause is an input clause.
Modern CDCL: Abstract Rules

Initialize  \( \epsilon \mid F \)  \( F \) is a set of clauses

Decide  \( M \mid F \Rightarrow M, \ell \mid F \)  \( \ell \) is unassigned

Propagate  \( M \mid F, C \lor \ell \Rightarrow M, \ell^{C \lor \ell} \mid F, C \lor \ell \)  \( C \) is false under \( M \)

Sat  \( M \mid F \Rightarrow M \)  \( F \) true under \( M \)

Conflict  \( M \mid F, C \Rightarrow M \mid F, C \mid C \)  \( C \) is false under \( M \)

Learn  \( M \mid F \mid C \Rightarrow M \mid F, C \mid C \)

Unsat  \( M \mid F \mid \emptyset \Rightarrow \text{Unsat} \)

Backjump  \( MM' \mid F \mid C \lor \ell \Rightarrow M\ell^{C \lor \ell} \mid F \)  \( \bar{C} \subseteq M, \neg \ell \in M' \)

Resolve  \( M \mid F \mid C' \lor \neg \ell \Rightarrow M \mid F \mid C' \lor C \)  \( \ell^{C \lor \ell} \in M \)

Forget  \( M \mid F, C \Rightarrow M \mid F \)  \( C \) is a learned clause

Restart  \( M \mid F \Rightarrow \epsilon \mid F \)  [Nieuwenhuis, Oliveras, Tinelli J.ACM 06] customized
SAT Algorithm

\[
\text{while(true) \{ } \\
\text{ if (!Decide())) } \\
\text{ return SAT;} \\
\text{ while (!BCP())) } \\
\text{ if (!ResolveConflict())) } \\
\text{ return UNSAT} \\
\text{ \}}
\]

- Apply unit resolution while it is applicable. Return FALSE if reached a conflict.
- Choose next variable and value. Return FALSE if all variables are assigned.
- Backtrack until no conflict. Return FALSE if impossible.
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
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, F_1 = cnf_rec (ψ)
        r, F_2 = cnf_rec (ξ)
        p = mk_fresh_var ()
        # C is CNF for p←(q∧r)
        C = (¬p∨q)∧(¬p∨r)∧(p∨¬q∨¬r)
        return (p, F_1∧F_2∧C)
    elif ϕ == ψ∨ξ:
        ...

Exercise: Complete cases for
φ == ψ∨ξ, φ==¬ψ, 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

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

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

\[
\begin{align*}
e_0 & \leftrightarrow (p \leftrightarrow e_1) \\
= & (e_0 \rightarrow (p \leftrightarrow e_1)) \land ((p \leftrightarrow e_1) \rightarrow e_0) \\
= & (e_0 \rightarrow (p \rightarrow e_1)) \land (e_0 \rightarrow (e_1 \rightarrow p)) \land \\
& (((p \land e_1) \lor (\neg p \land \neg e_1)) \rightarrow e_0) \\
= & (\neg e_0 \lor \neg p \lor e_1) \land (\neg e_0 \lor \neg e_1 \lor p) \land \\
& (\neg p \lor \neg e_1 \lor e_0) \land (p \lor e_1 \lor e_0)
\end{align*}
\]
Tseitin Transformation: Example

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

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

\[ G : e_0 \land (\neg e_0 \lor p \lor e_1) \land (\neg e_0 \lor p \lor \neg e_1) \land (e_0 \lor p \lor e_1) \land (e_0 \lor p \lor \neg e_1) \land \\
(\neg e_1 \lor \neg q \lor \neg r) \land (e_1 \lor q) \land (e_1 \lor \neg r) \]
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
MiniSat

MiniSat is one of the most famous modern SAT-solvers

• written in C++
• designed to be easily understandable and customizable
• many new SAT-solvers use MiniSAT as their base

Web page: http://minisat.se/

Good references for understanding SAT solving details

• MiniSat architecture: http://minisat.se/downloads/MiniSat.pdf
• Donald Knuth’s SAT13 (also based on MiniSat)
  – http://www-CS-faculty.stanford.edu/~knuth/programs/sat13.w