Syntax of Propositional Logic

An *atomic formula* has a form $A_i$, where $i = 1, 2, 3 \ldots$

*Formulas* are defined inductively as follows:

- All atomic formulas are formulas
- For every formula $F$, $\neg F$ (called not $F$) is a formula
- For all formulas $F$ and $G$, $F \land G$ (called and) and $F \lor G$ (called or) are formulas

**Abbreviations**

- use $A, B, C, \ldots$ instead of $A_1, A_2, \ldots$
- use $F_1 \rightarrow F_2$ instead of $\neg F_1 \lor F_2$ (implication)
- use $F_1 \leftrightarrow F_2$ instead of $(F_1 \rightarrow F_2) \land (F_2 \rightarrow F_1)$ (iff)
Syntax of Propositional Logic (PL)

\[
\text{truth\_symbol ::= } \top (\text{true}) \mid \bot (\text{false}) \\
\text{variable ::= } p, q, r, \ldots \\
\text{atom ::= truth\_symbol \mid variable} \\
\text{literal ::= atom} \mid \neg \text{atom} \\
\text{formula ::= literal} \mid \neg \text{formula} \mid \text{formula} \land \text{formula} \mid \text{formula} \lor \text{formula} \mid \text{formula} \rightarrow \text{formula} \mid \text{formula} \leftrightarrow \text{formula}
\]
Normal Forms: CNF and DNF

A literal is either an atomic proposition \( v \) or its negation \( \sim v \)

A clause is a disjunction of literals

\[ \text{e.g.,} \ (v_1 \lor \sim v_2 \lor v_3) \]

A formula is in \textit{Conjunctive Normal Form} (CNF) if it is a conjunction of disjunctions of literals (i.e., a conjunction of clauses):

\[ \land \left( \lor_{j=1}^{m} L_{i,j} \right)_{i=1}^{n} \land_{i=1}^{n} \lor_{j=1}^{m} L_{i,j} \]

A formula is in \textit{Disjunctive Normal Form} (DNF) if it is a disjunction of conjunctions of literals

\[ \lor \left( \land_{j=1}^{m} L_{i,j} \right)_{i=1}^{n} \land_{i=1}^{n} \lor_{j=1}^{m} L_{i,j} \]
Boolean Satisfiability (CNF-SAT)

Let \( V \) be a set of variables

A \textit{literal} is either a variable \( v \) in \( V \) or its negation \( \neg v \)

A \textit{clause} is a disjunction of literals

\begin{itemize}
  \item e.g., \((v_1 \lor \neg v_2 \lor v_3)\)
\end{itemize}

A Boolean formula in \textit{Conjunctive Normal Form} (CNF) is a conjunction of clauses

\begin{itemize}
  \item e.g., \((v_1 \lor \neg v_2) \land (v_3 \lor v_2)\)
\end{itemize}

An \textit{assignment} \( s \) of Boolean values to variables \textit{satisfies} a clause \( c \) if it evaluates at least one literal in \( c \) to true

An assignment \( s \) \textit{satisfies} a formula \( C \) in CNF if it satisfies every clause in \( C \)

Boolean Satisfiability Problem (CNF-SAT):

\begin{itemize}
  \item determine whether a given CNF \( C \) is satisfiable
CNF Examples

CNF 1
  • \sim b
  • \sim a \lor \sim b \lor \sim c
  • a
  • sat: s(a) = True; s(b) = False; s(c) = False

CNF 2
  • \sim b
  • \sim a \lor b \lor \sim c
  • a
  • \sim a \lor c
  • unsat
Algorithms for SAT

SAT is NP-complete
- solution can be checked in polynomial time
- no polynomial algorithms for finding a solution are known

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
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 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 the speed-up of 2012 solver over other solvers](from M. Vardi, https://www.cs.rice.edu/~vardi/papers/highlights15.pdf)
## 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>

2002 **Concept**

2010

---

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

---

[Le Berre'10]

Courtesy Daniel le Berre
NP is the new P!

Solve any computational problem by effective reduction to SAT/SMT

• iterate as necessary
Graph k-Coloring

Given a graph $G = (V, E)$, and a natural number $k > 0$ is it possible to assign colors to vertices of $G$ such that no two adjacent vertices have the same color.

Formally:
- does there exists a function $f : V \rightarrow [0..k)$ such that
- for every edge $(u, v)$ in $E$, $f(u) \neq f(v)$

Graph coloring for $k > 2$ is NP-complete

Problem: Encode k-coloring of $G$ into CNF
- construct CNF $C$ such that $C$ is SAT iff $G$ is $k$-colorable

https://en.wikipedia.org/wiki/Graph_coloring
**k-coloring as CNF**

Let a Boolean variable $f_{v,i}$ denote that vertex $v$ has color $i$

- if $f_{v,i}$ is true if and only if $f(v) = i$

Every vertex has at least one color

$$\bigvee_{0 \leq i < k} f_{v,i} \quad (v \in V)$$

No vertex is assigned two colors

$$\bigwedge_{0 \leq i < j < k} \neg f_{v,i} \lor \neg f_{v,j} \quad (v \in V)$$

No two adjacent vertices have the same color

$$\bigwedge_{0 \leq i < k} \neg f_{v,i} \lor \neg f_{u,i} \quad ((v, u) \in E)$$
References

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

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 \( D \):

\[
\text{Res}(\{C, p\}, \{D, \neg p\}) = \{C, D\}
\]
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 $F$.
Resolution Theorem

Let $F$ be a set of clauses

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

$Res^0(F) = F$

$Res^{n+1}(F) = Res(Res^n(F))$, 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
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

$$\neg b \land (\neg a \lor b \lor \neg c) \land a \land (\neg a \lor c)$$

$$\neg a \lor b \lor \neg c \quad a$$

$$\overline{b \lor \neg c \quad b}$$

$$\overline{\neg c \quad a \quad \neg a \lor c \quad c}$$
Proof of the Resolution Theorem

(Soundness) By Resolution Lemma, F is equivalent to $\text{Res}^i(F)$ for any i. Let n be such that $\text{Res}^{n+1}(F)$ contains an empty clause, but $\text{Res}^n(F)$ does not. Then $\text{Res}^n(F)$ must contain to unit clauses L and $\neg$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 $A_1$, $\ldots$, $A_{n+1}$
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 \textit{premises} and \(C\) is called a \textit{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

\[ C \lor p \hspace{1cm} D \lor \neg p \]

\[ C \lor D \]

Propositional resolution is a sound inference rule

Proposition resolution system consists of a single propositional resolution rule
DP Procedure: SAT solving by resolution

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

\[
\frac{F}{F, p} \quad | \quad \frac{F, \neg p}{\text{split}}
\]

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

p and \neg p are not in F

Davis, Martin; Logemann, George; Loveland, Donald (1962).
"A Machine Program for Theorem Proving".
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: Decide

Guessing (Decide)

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

\[ p, \neg q \mid p \lor q, \neg q \lor r \]
DPLL: Boolean Constraint Propagation

Deducing (Unit Propagation or BCP)

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

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

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 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

**assign**

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

**Guess 3**

<table>
<thead>
<tr>
<th>1, 2, 3</th>
<th>Deduce 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 2, 3, 4</td>
<td>Conflict</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 |
| 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</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deduce 1</td>
<td>1, 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deduce $\neg 2$</td>
<td>1, 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Guess 3</td>
<td>1, 2, 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deduce 4</td>
<td>1, 2, 3, 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Undo 3</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

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 \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 \neg 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>
</tbody>
</table>

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** \( M \parallel F, C \lor l \rightarrow M \parallel F, C \lor l \)  
\[ M \models \neg C \quad \text{I is undefined in } M \]

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

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

Repairing the assignment:

Fail: $M \parallel F, C \rightarrow \text{fail}$

- $M \vdash \neg C$
- $M$ does not contain decision literals

Backtrack: $M \downarrow^d N \parallel F, C \rightarrow M \leftarrow I \parallel F, C$

- $M \downarrow^d N \not\vdash \neg C$
- $I$ is the last decision literal
Transition Rules DPLL – Example

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

\[ 1 \ | 1 \lor 2, 2 \lor \lnot 3 \lor 4, \lnot 1 \lor \lnot 2, \lnot 1 \lor \lnot 3 \lor \lnot 4, 1 \]

\[ 1, 2 \ | 1 \lor 2, 2 \lor \lnot 3 \lor 4, \lnot 1 \lor \lnot 2, \lnot 1 \lor \lnot 3 \lor \lnot 4, 1 \]

\[ 1, 2, 3^d \ | 1 \lor 2, 2 \lor \lnot 3 \lor 4, \lnot 1 \lor \lnot 2, \lnot 1 \lor \lnot 3 \lor \lnot 4, 1 \]

\[ 1, 2, 3^d, 4 \ | 1 \lor 2, 2 \lor \lnot 3 \lor 4, \lnot 1 \lor \lnot 2, \lnot 1 \lor \lnot 3 \lor \lnot 4, 1 \]

UnitProp 1
UnitProp \lnot 2
Decide 3
UnitProp 4
Backtrack 3
Transition Rules DPLL – Example

<table>
<thead>
<tr>
<th>Rule</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset \models 1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1$</td>
<td></td>
</tr>
<tr>
<td>1 $\models 1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1$</td>
<td></td>
</tr>
<tr>
<td>1, 2 $\models 1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1$</td>
<td></td>
</tr>
<tr>
<td>1, 2, 3 $\models 1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1$</td>
<td></td>
</tr>
</tbody>
</table>

UnitProp 1
UnitProp 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: \[ M \parallel F, C \rightarrow M \parallel F, C \]
- \( M \parallel \neg I \)
- \( M \parallel F, C \)
- \( M I^d N \models \neg C \)
- \( I \) 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
- ECE750 with Vijay Ganesh
Conflict Directed Clause Learning

Lemma learning

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

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

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

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

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

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

\[ \neg p \lor \neg s \quad \neg q \lor s \]

\[ \neg p \lor \neg q \quad t \lor \neg p \lor q \]

\[ t \lor \neg p \]

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

<table>
<thead>
<tr>
<th>Rule</th>
<th>Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize</td>
<td>$\epsilon \mid F$</td>
</tr>
<tr>
<td>Decide</td>
<td>$M \mid F \Rightarrow M, \ell \mid F$</td>
</tr>
<tr>
<td>Propagate</td>
<td>$M \mid F, C \lor \ell \Rightarrow M, \ell^{\lor \ell} \mid F, C \lor \ell$</td>
</tr>
<tr>
<td>Sat</td>
<td>$M \mid F \Rightarrow M$</td>
</tr>
<tr>
<td>Conflict</td>
<td>$M \mid F, C \Rightarrow M \mid F, C \mid C$</td>
</tr>
<tr>
<td>Learn</td>
<td>$M \mid F \mid C \Rightarrow M \mid F, C \mid C$</td>
</tr>
<tr>
<td>Unsat</td>
<td>$M \mid F \mid \emptyset \Rightarrow \text{Unsat}$</td>
</tr>
<tr>
<td>Backjump</td>
<td>$MM' \mid F \mid C \lor \ell \Rightarrow M\ell^{\lor \ell} \mid F$</td>
</tr>
<tr>
<td>Resolve</td>
<td>$M \mid F \mid C' \lor \lnot \ell \Rightarrow M \mid F \mid C' \lor C$</td>
</tr>
<tr>
<td>Forget</td>
<td>$M \mid F, C \Rightarrow M \mid F$</td>
</tr>
<tr>
<td>Restart</td>
<td>$M \mid F \Rightarrow \epsilon \mid F$</td>
</tr>
</tbody>
</table>

- $F$ is a set of clauses
- $\ell$ is unassigned
- $C$ is false under $M$
- $F$ true under $M$
- $C$ is false under $M$
- $\bar{C} \subseteq M, \lnot \ell \in M'$
- $\ell^{\lor \ell} \in M$
- $C$ is a learned clause

[Nieuwenhuis, Oliveras, Tinelli J.ACM 06] customized
Conjunctive Normal Form

\( \varphi \iff \psi \quad \Rightarrow \text{CNF} \quad \varphi \rightarrow \psi \wedge \psi \rightarrow \varphi \)

\( \varphi \rightarrow \psi \quad \Rightarrow \text{CNF} \quad \neg \varphi \lor \psi \)

\( \neg (\varphi \lor \psi) \quad \Rightarrow \text{CNF} \quad \neg \varphi \land \neg \psi \)

\( \neg (\varphi \land \psi) \quad \Rightarrow \text{CNF} \quad \neg \varphi \lor \neg \psi \)

\( \neg \neg \varphi \quad \Rightarrow \text{CNF} \quad \varphi \)

\( (\varphi \land \psi) \lor \xi \quad \Rightarrow \text{CNF} \quad (\varphi \lor \xi) \land (\psi \lor \xi) \)

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

def cnf(\phi):
    p, F = cnf_rec(\phi)
    return p \land F

def cnf_rec(\phi):
    if is_atomic(\phi): return (\phi, True)
    elif \phi == \psi \land \xi:
        q, F_1 = cnf_rec(\psi)
        r, F_2 = cnf_rec(\xi)
        p = mk_fresh_var()
        # C is CNF for p \iff (q \land r)
        C = (\neg p \lor q) \land (\neg p \lor r) \land (p \lor \neg q \lor \neg r)
        return (p, F_1 \land F_2 \land C)
    elif \phi == \psi \lor \xi:
        ...

**Exercise:** Complete cases for

\phi = \psi \lor \xi, \phi = \neg \psi, \phi = \psi \iff \xi

*mk_fresh_var()* returns a fresh variable not used anywhere before.
Tseitin Transformation: Example

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

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

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

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

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

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

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

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

\[ G: e_0 \land (\neg e_0 \lor \neg p \lor e_1) \land (\neg e_0 \lor p \lor \neg e_1) \land (e_0 \lor p \lor e_1) \land (e_0 \lor \neg p \lor \neg e_1) \land (\neg e_1 \lor \neg q \lor r) \land (e_1 \lor q) \land (e_1 \lor \neg r) \]
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
DIMACS CNF File Format

Textual format to represent CNF-SAT problems

c start with comments
c

c
p cnf 5 3
1 -5 4 0
-1 5 3 4 0
-3 -4 0

Format details
• comments start with c
• header line: p cnf nbvar nbclauses
  – nbvar is # of variables, nbclauses is # of clauses
• each clause is a sequence of distinct numbers terminating with 0
  – positive numbers are variables, negative numbers are negations
BOUNDDED MODEL CHECKING
SAT-based Model Checking

Main idea
Translate the model and the specification to propositional formulas

Reduce the model checking problem to satisfiability of propositional formulas

Use efficient tools (SAT solvers) for solving the satisfiability problem
Finite-State System is modeled as \((V, \text{INIT}, T)\):

- \(V\) – finite set of Boolean variables
  - Boolean variables: \(a\ b\ c\) \(\rightarrow\) 8 states: 000,001,...
- \(\text{INIT}(V)\) – describes the set of initial states
  - \(\text{INIT} = \neg a \land \neg b\)
- \(T(V,V')\) – describes the set of transitions
  - \(T(a,b,c,a',b',c') = (c' \leftrightarrow (a \land b) \lor c)\)

Property:

- \(p(V)\) - describes the set of states satisfying \(p\)
  - \(p = a \lor \neg c\) \(\text{ (Bad} = \neg p = \neg a \land c \text{)\)
Modeling in CNF (Tseitin encoding)

Each circuit element is a constraint

\[ g = a \land b \]

\[ p = g \lor c \]

\[ c' = p \]

\[ T(a, b, c, g, p, a', b', c') = \]

\[ g \leftrightarrow a \land b, \]

\[ p \leftrightarrow g \lor c, \]

\[ c' \leftrightarrow p \]

Each circuit element is a constraint
Bounded model checking (BMC) for checking $\text{AG} p$

**Given**

- A finite transition system $M = (V, \text{INIT}(V), T(V, V'))$
- A safety property $\text{AG} p$, where $p = p(V)$
- A bound $k$

**Determine**

- Does $M$ contain a counterexample to $p$ of $k$ transitions (or fewer)?

* BMC can handle all of LTL formulas

A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, Y. Zhu, DAC'99
Bounded model checking for checking AGp

Unwind the model for k levels, i.e., construct all computations of length k. If a state satisfying $\neg p$ is encountered, then produce a counterexample.

The method is suitable for falsification, not verification.

Can be translated to a SAT problem.
Bounded model checking with SAT

Construct a formula $f_{M,k}$ describing all possible computations of M of length k

$g = a \land b$

$p = g \lor c$

$c' = p$

$T(a, b, c, a', b', c') =$

$g \leftrightarrow a \land b,$

$p \leftrightarrow g \lor c,$

$c' \leftrightarrow p$
Bounded model checking with SAT

Construct a formula $f_{M,k}$ describing all possible computations of M of length k

$$f_{M,k} = INIT_0 \land T_0 \land T_1 \land \ldots \land T_{k-1}$$

$INIT_0 = INIT(V_0)$

$T_i = T(V_i, V_{i+1})$
Bounded model checking with SAT

Construct a formula $f_{M,k}$ describing all possible computations of $M$ of length $k$

Construct a formula $f_{\varphi,k}$ expressing that $\varphi = EF\neg p$ holds within $k$ computation steps

$$f_{\varphi,k} = V_{i=0..k} (\neg p_i) \quad [\text{Sometimes } f_{\varphi,k} = \neg p_k]$$

$$p_i = p(V_i)$$
Bounded model checking with SAT

Construct a formula $f_{M,k}$ describing all possible computations of $M$ of length $k$

Construct a formula $f_{\varphi,k}$ expressing that $\varphi=\text{EF}\neg p$ holds within $k$ computation steps

Check whether $f = f_{M,k} \land f_{\varphi,k}$ is satisfiable

If $f$ is satisfiable then $M \not\models \text{AGp}$

The satisfying assignment is a counterexample
BMC for checking AG p with SAT

Unfold the model k times:

• $U = T^{<0>} \land T^{<1>} \land ... \land T^{<k-1>}$

$\text{Biere, et al. TACAS99}$

$I^{<0>} = I(V_0)$

$T^{<i>} = T(V_i, V_{i+1})$

$p^{<k>} = p(V_k)$

• Use SAT solver to check satisfiability of

$I^{<0>} \land U \land \neg p^{<k>}$

• If satisfiable: the satisfying assignment describes a counterexample of length k

• If unsatisfiable: property has no counterexample of length k
Example – shift register

Shift register of 3 bits: \( \langle x, y, z \rangle \)

**Transition relation:**

\[
T(x, y, z, x', y', z') = x' \leftrightarrow y \land y' \leftrightarrow z \land z' = 1
\]

\[\text{error}\]

**Initial condition:**

\[
\text{INIT}(x, y, z) = x = 0 \lor y = 0 \lor z = 0
\]

**Specification:** \( \text{AG} (x = 0 \lor y = 0 \lor z = 0) \)
Propositional formula for $k=2$

$$f_{M,2} = (x_0=0 \lor y_0=0 \lor z_0=0) \land$$
$$\quad (x_1 \leftrightarrow y_0 \land y_1 \leftrightarrow z_0 \land z_1=1) \land$$
$$\quad (x_2 \leftrightarrow y_1 \land y_2 \leftrightarrow z_1 \land z_2=1)$$

$$f_{\varphi,2} = V_{i=0,..2} (x_i=1 \land y_i=1 \land z_i=1)$$

Satisfying assignment: 101 011 111

This is a counterexample!
A remark

In order to describe a computation of length $k$ by a propositional formula we need $k+1$ copies of the state variables.

With BDDs we use only two copies: for current and next states.
BMC for checking $\varphi=\text{AG}p$

1. $k=1$

2. Build a propositional formula $f_M^k$ describing all prefixes of length $k$ of paths of $M$ from an initial state

3. Build a propositional formula $f_\varphi^k$ describing all prefixes of length $k$ of paths satisfying $F\neg p$

4. If $(f_M^k \land f_\varphi^k)$ is satisfiable, return the satisfying assignment as a counterexample

5. Otherwise, increase $k$ and return to 2.
Bounded Model Checking

\[ \text{INIT}(V^0) \land T(V^0, V^1) \land \neg p(V^1) \]
Bounded Model Checking

\[\text{INIT}(V^0) \land T(V^0,V^1) \land T(V^1,V^2) \land \neg p(V^2)\]
Bounded Model Checking

\[ \text{INIT}(V^0) \land T(V^0, V^1) \land \ldots \land T(V^{k-1}, V^k) \land \neg p(V^k) \]
BMC for checking $\text{AFp} (\varphi = \text{EG} \neg p)$

Is there an infinite path in M

- From an initial state
- all of its states satisfying $\neg p$
- Over $k+1$ states?

If exists, there must also exist a lasso
BMC for checking AFp ($\varphi = \text{EG} \neg p$)

An infinite path in M, from an initial state, over $k+1$ states, all satisfying $\neg p$:

$$f_M^k (V_0, \ldots, V_k) =$$

$$\text{INIT}(V_0) \land \bigwedge_{i=0, \ldots, k-1} T(V_i, V_{i+1}) \land V_{i=0, \ldots, k-1} (V_k = V_i)$$

- $V_k = V_i$ means bitwise equality: $\bigwedge_{j=0, \ldots n} (v_{kj} \leftrightarrow v_{ij})$

$$f_{\varphi}^k (V_0, \ldots, V_k) = \bigwedge_{i=0, \ldots, k} \neg p(V_i)$$

Remark: BMC can handle all of LTL formulas
Bounded model checking

Can handle all of LTL formulas

Can be used for verification by choosing $k$ which is large enough

- Need bound on length of the shortest counterexample.

  - Diameter bound. The diameter is the maximum length of the shortest path between any two states.

Using such $k$ is often not practical due to the size of the model

- Worst case diameter is exponential. Obtaining better bounds is sometimes possible, but generally intractable.
Bounded Model Checking

Terminates
• with a counterexample or
• with time- or memory-out

=> The method is suitable for falsification, not verification

Can be used for verification by choosing k which is large enough
• Need bound on length of the shortest counterexample.
  – diameter bound. The diameter is the maximum length of the shortest path between any two states.

Using such k is often not practical
  – Worst case diameter is exponential. Obtaining better bounds is sometimes possible, but generally intractable.