Satisfiability Modulo Theories

Summer School on Formal Methods

Menlo College, 2011

Bruno Dutertre and Leonardo de Moura

`bruno@csl.sri.com, leonardo@microsoft.com`

SRI International, Microsoft Research

What's Satisfiability Modulo Theory

- Satisfiability is the problem of determining whether a formula ϕ has a model
  - If ϕ is propositional, a model is a truth assignemt to Boolean variables
  - If ϕ is a first-order formula, a model assigns values to variables and interpretations to the function and predicate symbols
- SAT Solvers: check satisfiability of propositional formulas
- SMT Solvers: check satisfiability of formulas in a decidable first-order theory (e.g., linear arithmetic, uninterpreted functions, array theory, bitvectors)

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

Arithmetic

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

Array theory

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

Uninterpreted function

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

By arithmetic, this is equivalent to

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), b)) \neq f(3)$$

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

By arithmetic, this is equivalent to

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), b)) \neq f(3)$$

then, by the array theory axiom: $\texttt{read}(\texttt{write}(v, i, x), i) = x$

$$b + 2 = c \wedge f(3) \neq f(3)$$

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

By arithmetic, this is equivalent to

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), b)) \neq f(3)$$

then, by the array theory axiom: $\texttt{read}(\texttt{write}(v, i, x), i) = x$

$$b + 2 = c \wedge f(3) \neq f(3)$$

then, the formula is unsatisfiable

Example 2

$$x \geq 0 \wedge f(x) \geq 0 \wedge y \geq 0 \wedge f(y) \geq 0 \wedge x \neq y$$

Example 2

$$x \geq 0 \wedge f(x) \geq 0 \wedge y \geq 0 \wedge f(y) \geq 0 \wedge x \neq y$$

This formula is satisfiable

Example 2

$$x \geq 0 \land f(x) \geq 0 \land y \geq 0 \land f(y) \geq 0 \land x \neq y$$

This formula is satisfiable:

Example model:

$$x \rightarrow 1$$

$$y \rightarrow 2$$

$$f(1) \rightarrow 0$$

$$f(2) \rightarrow 1$$

$$f(\ldots) \rightarrow 0$$

SMT Solving

- Input
  - a first-order formula ϕ
- Output
  - the status of ϕ: satisfiable or unsatisfiable
  - optionally, if ϕ is satisfiable, a model of ϕ
  - also optionally, if ϕ is unsatisfiable, a proof of unsatisfiability
- Main issues
  - Formula size (e.g., thousands of atoms or more)
  - Formulas with complex Boolean structure
  - Combinations of theories

Overview of SMT Solving

- SMT Solver = SAT Solver + Theory Solver
  - The SAT solver enumerates possible truth assignments
  - The theory solver is a decision procedure that checks whether the truth assignments are satisfiable in the theory (or combination of theories)
- Efficient integration uses several mechanisms
  - Theory explanations to rule out unsatisfiable truth assignments
  - Theory lemmas and theory propagation to prune the SAT solver search tree

Naïve SMT Solving

$$x + y \geq 0 \land (x = z \Rightarrow z + y = -1) \land z > 3t$$

1) Replace atoms by boolean variables

$$\begin{array}{lllllll} a & \mapsto & x + y \geq 0 & \quad & b & \mapsto & x = z \\ c & \mapsto & z + y = -1 & \quad & d & \mapsto & z > 3t \end{array}$$

2) Ask for a model of $a \land (b \Rightarrow c) \land d$ using a SAT solver

- Boolean model: $\{a, b, c, d\}$
- Convert the model back to arithmetic

$$x + y \geq 0 \land x = z \land z + y = -1 \land z > 3t$$

This is not consistent:

$$Arithmetic \models \neg(x + y \geq 0 \land x = z \land z + y = -1)$$

Naïve SMT Solving (continued)

3) Feed the explanation to the SAT solver:

- add the clause $(\neg a \vee \neg b \vee \neg c)$

4) Get a model of $(a \wedge (b \Rightarrow c) \wedge d) \wedge (\neg a \vee \neg b \vee \neg c)$

- Boolean model: $\{a, \neg b, c, d\}$
- Convert back to arithmetic:

$$x + y \geq 0 \wedge \neg(x = z) \wedge z + y = -1 \wedge z > 3t$$

- Check consistency: satisfiable

Conclusion: The original formula is satisfiable

Remainder of the Lectures

- We will cover the basics of SAT and SMT solving more precisely.
- We will not give a comprehensive survey, but a basic and rigorous introduction to some of the key ideas.
- This tutorial is not directed at experts but at potential users and developers of SMT solvers.

Plan

- Lecture 1: principles of SAT solving
  - Logic background
  - Modern DPLL SAT solvers
- Lecture 2: SMT solving
  - DPLL + Theory Solvers
  - Theory combination
  - Equality
  - Arithmetic
- Lecture 3: applications of SMT

Roadmap

- Logic Background
- Modern SAT Solvers
- DPPL with Theory Solvers
- Theory Combination
- Equality
- Arithmetic
- Applications

Logic Basics

- Logic studies the trinity between language, interpretation, and proof.
- Language circumscribes the syntax that is used to construct sensible assertions.
- Interpretation ascribes an intended sense to these assertions by fixing the meaning of certain symbols, e.g., the logical connectives, and delimiting the variation in the meanings of other symbols, e.g., variables, functions, and predicates.
- An assertion is valid if it holds in all interpretations.
- Checking validity through interpretations is typically not feasible, so proofs in the form axioms and inference rules are used to demonstrate the validity of assertions.

Language: Signatures

- A signature Σ is a finite set of:
  - Function symbols: $\Sigma_F = \{f, g, \ldots\}$.
  - Predicate symbols: $\Sigma_P = \{p, q, \ldots\}$.
  - and an arity function: $\Sigma \mapsto \mathbb{N}$
- Function symbols with arity 0 are called constants.
- A countable set $\mathcal{V}$ of variables disjoint of Σ.

Language: Terms

- The set $T(\Sigma, \mathcal{V})$ of terms is the smallest set such that:
  - $\mathcal{V} \subset T(\Sigma, \mathcal{V})$
  - $f(t_1, \ldots, t_n) \in T(\Sigma, \mathcal{V})$ whenever $f \in \Sigma_F, t_1, \ldots, t_n \in T(\Sigma, \mathcal{V})$ and *arity*$(f) = n$.
- The set of ground terms is defined as $T(\Sigma, \emptyset)$.

Language: Atomic Formulas

- $p(t_1, \ldots, t_n)$ is an atomic formula whenever $p \in \Sigma_P$, *arity*$(p) = n$, and $t_1, \ldots, t_n \in T(\Sigma, \mathcal{V})$.
- true and false are atomic formulas.
- If $t_1, \ldots, t_n$ are ground terms, then $p(t_1, \ldots, t_n)$ is called a ground (atomic) formula.
- We assume that the binary predicate $=$ is present in Σ_P.
- A literal is an atomic formula or its negation.

Language: Quantifier Free Formulas

- The set *QFF*$(\Sigma, \mathcal{V})$ of quantifier-free formulas is the smallest set such that:
  - Every atomic formulas is in *QFF*$(\Sigma, \mathcal{V})$.
  - If $\phi \in$ *QFF*$(\Sigma, \mathcal{V})$, then $\neg\phi \in$ *QFF*$(\Sigma, \mathcal{V})$.
  - If $\phi_1, \phi_2 \in$ *QFF*$(\Sigma, \mathcal{V})$, then

$$
\begin{array}{rcl}
\phi_1 \wedge \phi_2 & \in & \mathit{QFF}(\Sigma, \mathcal{V}) \\
\phi_1 \vee \phi_2 & \in & \mathit{QFF}(\Sigma, \mathcal{V}) \\
\phi_1 \Rightarrow \phi_2 & \in & \mathit{QFF}(\Sigma, \mathcal{V}) \\
\phi_1 \Leftrightarrow \phi_2 & \in & \mathit{QFF}(\Sigma, \mathcal{V})
\end{array}
$$

Language: Formulas

- The set of first-order formulas is the closure of *QFF*$(\Sigma, \mathcal{V})$ under existential ($\exists$) and universal ($\forall$) quantification.
- Free (occurrences) of variables in a formula are those not bound by a quantifier.
- A sentence is a first-order formula with no free variables.

Models (Semantics)

- A model M is defined as:
  - Domain $|M|$: set of elements.
  - Interpretation $M(f) : |M|^n \mapsto |M|$ for each $f \in \Sigma_F$ with *arity*$(f) = n$.
  - Interpretation $M(p) \subseteq |M|^n$ for each $p \in \Sigma_P$ with *arity*$(p) = n$.
  - Assignment $M(x) \in |M|$ for every variable $x \in \mathcal{V}$.
- A formula ϕ is true in a model M if it evaluates to true under the given interpretations over the domain $|M|$.

Interpreting Terms

$$
\begin{aligned}
M[\![x]\!] &= M(x) \\
M[\![f(a_1, \ldots, a_n)]\!] &= M(f)(M[\![a_1]\!], \ldots, M[\![a_n]\!])
\end{aligned}
$$

Interpreting Formulas

The interpretation of a formula F in M, $M[\![F]\!]$, is defined as

$$
\begin{array}{rcl}
M \models a = b & \iff & M[\![a]\!] = M[\![b]\!] \\
M \models p(a_1, \ldots, a_n) & \iff & \langle M[\![a_1]\!], \ldots, M[\![a_n]\!]\rangle \in M(p) \\
M \models \neg\psi & \iff & M \not\models \psi \\
M \models \psi_1 \vee \psi_2 & \iff & M \models \psi_1 \text{ or } M \models \psi_2 \\
M \models \psi_1 \wedge \psi_2 & \iff & M \models \psi_1 \text{ and } M \models \psi_2 \\
M \models (\forall x : \psi) & \iff & M\{x \mapsto a\} \models \psi, \text{ for all } a \in |M| \\
M \models (\exists x : \psi) & \iff & M\{x \mapsto a\} \models \psi, \text{ for some } a \in |M|
\end{array}
$$

Interpretation Example

$$\begin{aligned}
\Sigma &= \{0, +, <\}, \text{ and } M \text{ such that } |M| = \{a, b, c\} \\
M(0) &= a, \\
M(+) &= \{\langle a, a \mapsto a\rangle, \langle a, b \mapsto b\rangle, \langle a, c \mapsto c\rangle, \langle b, a \mapsto b\rangle, \langle b, b \mapsto c\rangle, \\
&\quad \langle b, c \mapsto a\rangle, \langle c, a \mapsto c\rangle, \langle c, b \mapsto a\rangle, \langle c, c \mapsto b\rangle\} \\
M(<) &= \{\langle a, b\rangle, \langle a, c\rangle, \langle b, c\rangle\}
\end{aligned}$$

If $M(x) = a, M(y) = b, M(z) = c$, then

$M[\![+(+(x, y), z)]\!] =$

$M(+)(M(+)(M(x), M(y)), M(z)) = M(+)(M(+)(a, b), c) =$

$M(+)(b, c) = a$

Interpretation Example

$$
\begin{aligned}
\Sigma &= \{0, +, <\}, \text{ and } M \text{ such that } |M| = \{a, b, c\} \\
M(0) &= a, \\
M(+) &= \{\langle a, a \mapsto a\rangle, \langle a, b \mapsto b\rangle, \langle a, c \mapsto c\rangle, \langle b, a \mapsto b\rangle, \langle b, b \mapsto c\rangle, \\
&\quad \langle b, c \mapsto a\rangle, \langle c, a \mapsto c\rangle, \langle c, b \mapsto a\rangle, \langle c, c \mapsto b\rangle\} \\
M(<) &= \{\langle a, b\rangle, \langle a, c\rangle, \langle b, c\rangle\}
\end{aligned}
$$

$$M \models (\forall x : (\exists y : +(x, y) = 0))$$

$$M \not\models (\forall x : (\exists y : x < y))$$

$$M \models (\forall x : (\exists y : +(x, y) = x))$$

Validity

- A formula F is satisfiable if there is a model M such that $M \models F$.
- Otherwise, the formula F is unsatisfiable.
- If a formula is satisfiable, so is its existential closure $\exists \vec{x} : F$, where $\vec{x}$ is *vars*(F), the set of free variables in F.
- If a formula F is unsatisfiable, then the negation of its existential closure $\neg\exists \vec{x} : F$ is valid.

Theories

- A (first-order) theory $\mathcal{T}$ (over a signature Σ) is a set of (deductively closed) sentences (over Σ and $\mathcal{V}$).
- Let *DC*(Γ) be the deductive closure of a set of sentences Γ.
  - For every theory $\mathcal{T}$, *DC*$(\mathcal{T}) = \mathcal{T}$.
- A theory $\mathcal{T}$ is consistent if false $\notin \mathcal{T}$.
- We can view a (first-order) theory $\mathcal{T}$ as the class of all models of $\mathcal{T}$ (due to completeness of first-order logic).

Satisfiability and Validity

- A formula $\phi(\vec{x})$ is satisfiable in a theory $\mathcal{T}$ if there is a model of *DC*$(\mathcal{T} \cup \exists \vec{x}.\phi(\vec{x}))$. That is, there is a model M for $\mathcal{T}$ in which $\phi(\vec{x})$ evaluates to true, denoted by,

$$M \models_{\mathcal{T}} \phi(\vec{x})$$

- This is also called $\mathcal{T}$-satisfiability.
- A formula $\phi(\vec{x})$ is valid in a theory $\mathcal{T}$ if $\forall \vec{x}.\phi(\vec{x}) \in \mathcal{T}$. That is $\phi(\vec{x})$ evaluates to true in every model M of $\mathcal{T}$.
- $\mathcal{T}$-validity is denoted by $\models_{\mathcal{T}} \phi(\vec{x})$.
- The quantifier free $\mathcal{T}$-satisfiability problem restricts ϕ to be quantifier free.

Roadmap

- Logic Background
- Modern SAT Solvers
- DPLL with Theory Solvers
- Theory Combination
- Equality
- Arithmetic
- Applications

SAT Solvers

- Modern Boolean SAT solvers are based on the Davis-Putnam and Davis-Logemann-Loveland (DPLL) procedures
  - Input formula is in Conjunctive Normal Form (CNF)
  - Solvers combine search with backtracking and deduction based on resolution

Clausal Form (CNF)

- In clausal form, a formula is a set (conjunction) of clauses $\bigwedge_i C_i$, and each clause C_i is a disjunction of literals.
  A literal is an atom or the negation of an atom.
- Example: $(p_1 \vee \neg p_2) \ \wedge \ (\neg p_1 \vee p_2 \vee p_3) \ \wedge \ p_3$
- Theorem: for any formula ϕ, there's a CNF formula ϕ' such that $\phi' \iff \phi$.
- But, ϕ' may be exponentially larger than ϕ. For example, if ϕ is $(p_1 \wedge q_1) \vee (p_2 \wedge q_2) \vee \ldots \vee (p_n \wedge q_n)$.
- Rather than constructing a CNF formula equivalent to ϕ, it's cheaper to construct a CNF formula ϕ' that preserves satisfiability:

ϕ is satisfiable iff ϕ' is satisfiable

Efficient Conversion to CNF

- Idea: replace a subformula ψ by a fresh variable p, then add clauses to express the constraint $p \iff \psi$

  For example, we can replace $(p_1 \wedge p_2)$ by a fresh p and add the clauses $(\neg p \vee p_1)$, $(\neg p \vee p_2)$, and $(p \vee \neg p_1 \vee \neg p_2)$

CNF-Conversion Procedure

$$
\begin{aligned}
\mathit{CNF}(p, \Delta) &= \langle p, \Delta \rangle \\
\mathit{CNF}(\neg\phi, \Delta) &= \langle \neg l, \Delta' \rangle, \text{ where } \langle l, \Delta' \rangle = \mathit{CNF}(\phi, \Delta) \\
\mathit{CNF}(\phi_1 \wedge \phi_2, \Delta) &= \langle p, \Delta' \rangle, \text{ where} \\
& \quad \langle l_1, \Delta_1 \rangle = \mathit{CNF}(\phi_1, \Delta) \\
& \quad \langle l_2, \Delta_2 \rangle = \mathit{CNF}(\phi_2, \Delta_1) \\
& \quad p \text{ is fresh} \\
& \quad \Delta' = \Delta_2 \cup \{\neg p \vee l_1, \neg p \vee l_2, \neg l_1 \vee \neg l_2 \vee p\} \\
\mathit{CNF}(\phi_1 \vee \phi_2, \Delta) &= \langle p, \Delta' \rangle, \text{ where} \ldots \\
& \quad \Delta' = \Delta_2 \cup \{\neg p \vee l_1 \vee l_2, \neg l_1 \vee p, \neg l_2 \vee p\}
\end{aligned}
$$

Theorem: ϕ and $l \wedge \Delta$ are equisatisfiable, where $\mathit{CNF}(\phi, \emptyset) = \langle l, \Delta \rangle$.

Conversion to CNF: Example

$$\mathit{CNF}(\neg(\underbrace{q_1 \wedge \overbrace{(q_2 \vee \neg q_3)}^{p_1}}_{p_2}), \emptyset) =$$

$$\begin{aligned}
\langle \neg p_2, \{ \quad & \neg p_1 \vee q_2 \vee \neg q_3, \\
& \neg q_2 \vee p_1, \\
& q_3 \vee p_1, \\
& \neg p_2 \vee q_1, \\
& \neg p_2 \vee p_1, \\
& \neg q_1 \vee \neg p_1 \vee p_2 \} \rangle
\end{aligned}$$

Conversion to CNF: Improvements

- Maximize sharing & canonicity in the input formula F.
- Cache $\phi \mapsto l$, when *CNF*$(\phi, \Delta) = \langle l, \Delta' \rangle$.
- Support for multiary $\vee$ and $\wedge$.
- $\ldots$

Resolution

- Resolution rule: take two clauses $(p \vee A)$ and $(\neg p \vee B)$ add the new clause $(A \vee B)$, called the resolvent
- In this rule, clauses are considered as sets of literals:
  - No duplicate literals in clauses.
  - $(A \vee B)$ is a tautology (can be deleted) if it contains complementary literals l and $\bar{l}$
  - The empty clause is false.
- Property: if F is a set of clauses (CNF formula), and C is the resolvent of two clauses from F, then F and $F \cup C$ are equivalent.
- So, if we can derive the empty clause from F using resolution, we know that F is unsatisfiable.

Resolution: Example

$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r$

Resolution: Example

$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r \qquad \Rightarrow$

$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r,\ \neg q \vee r$

Resolution: Example

$$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r \qquad \Rightarrow$$

$$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r,\ \neg q \vee r \qquad \Rightarrow$$

$$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r,\ \neg q \vee r,\ q \vee r$$

Resolution: Example

$$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r \quad \Rightarrow$$

$$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r,\ \neg q \vee r \quad \Rightarrow$$

$$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r,\ \neg q \vee r,\ q \vee r \quad \Rightarrow$$

$$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r,\ \neg q \vee r,\ q \vee r,\ r$$

Resolution: Example

$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r \quad \Rightarrow$

$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r,\ \neg q \vee r \quad \Rightarrow$

$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r,\ \neg q \vee r,\ q \vee r \quad \Rightarrow$

$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r,\ \neg q \vee r,\ q \vee r,\ r \quad \Rightarrow$

unsat

The (original) DPLL Search Procedure

- Exhaustive resolution is not practical (exponential amount of memory).
- DPLL tries to build incrementally a model M for a CNF formula F.
- M is grown by:
  - deducing the truth value of a literal from M and F, or
  - guessing the truth value of an unassigned literal
- Deducing is based on the unit-propagation rule:
  If F contains a clause $C \vee l$ and all literals of C are false in M
  then l must be true.
- If a wrong guess leads to an inconsistency, the procedure backtracks to the last guess and tries the opposite value.

Improvements to DPLL in Modern SAT solvers

- Breakthrough: Conflict-driven clause learning and backjumping:
  - When an inconsistency is detected, use resolution to construct a new (learned) clause
  - This clause is used to determine how far to backtrack
  - Benefits:
    - Backtracking can happen further than the last guess (pruning of the search tree)
    - The learned clause may avoid repeating the same conflict
- Other improvements: restarts, variable activity heuristics, clause indexing for fast propagation, preprocessing, etc.

Abstract DPLL

- During search, DPLL states are pairs $M \parallel F$ where
  - M is a truth assignment
  - F is a set of clauses (problem clauses + learned clauses)
- The truth assignment is a list of literals: either decision literals (guesses) or implied literals (by unit propagation).
  - If literal l is implied by unit propagation from clause $C \vee l$, then the clause is recorded as the explanation for l. This is written $l_{C \vee l}$ in M.
- During conflict resolution, the state is written $M \parallel F \parallel C$ where M and F are as before, and C is a clause.
  - C is false in the assigment M (written $M \models \neg C$)
  - C is either a clause of F or is derived by resolution from clauses of F.

Abstract DPLL

$M \parallel F$	$\Longrightarrow$	$M\, l \parallel F$	**if**	$\begin{cases} l \text{ or } \bar{l} \text{ occurs in } F, \\ l \text{ is undefined in } M \end{cases}$	(**Decide**)
$M \parallel F, C \vee l$	$\Longrightarrow$	$M\, l_{C \vee l} \parallel F, C \vee l$	**if**	$\begin{cases} M \models \neg C, \\ l \text{ is undefined in } M \end{cases}$	(**UnitPropagate**)
$M \parallel F, C$	$\Longrightarrow$	$M \parallel F, C \parallel C$	**if**	$M \models \neg C$	(**Conflict**)
$M \parallel F \parallel C \vee \bar{l}$	$\Longrightarrow$	$M \parallel F \parallel D \vee C$	**if**	$l_{D \vee l} \in M,$	(**Resolve**)
$M \parallel F \parallel C$	$\Longrightarrow$	$M \parallel F, C \parallel C$	**if**	$C \not\in F$	(**Learn**)
$M\, l'\, M' \parallel F \parallel C \vee l$	$\Longrightarrow$	$M\, l_{C \vee l} \parallel F$	**if**	$\begin{cases} M \models \neg C, \\ l \text{ is undefined in } M \end{cases}$	(**Backjump**)
$M \parallel F \parallel \Box$	$\Longrightarrow$	*unsat*			(**Unsat**)

Abstract DPLL: Example

$$\| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$$

Abstract DPLL: Example

$$\parallel \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(Decide)}$$

$$1 \parallel \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$$

Abstract DPLL: Example

$$\| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(Decide)}$$

$$1 \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(UnitProp)}$$

$$1\ 2_{\overline{1} \vee 2} \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$$

Abstract DPLL: Example

$$\| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(Decide)}$$

$$1 \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(UnitProp)}$$

$$1\ 2_{\overline{1} \vee 2} \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(Decide)}$$

$$1\ 2_{\overline{1} \vee 2}\ 3 \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$$

Abstract DPLL: Example

$$\| \quad \overline{1} \lor 2,\ \overline{3} \lor 4,\ \overline{5} \lor \overline{6},\ 6 \lor \overline{5} \lor \overline{2} \quad \Rightarrow \quad \text{(Decide)}$$

$$1 \quad \| \quad \overline{1} \lor 2,\ \overline{3} \lor 4,\ \overline{5} \lor \overline{6},\ 6 \lor \overline{5} \lor \overline{2} \quad \Rightarrow \quad \text{(UnitProp)}$$

$$1\ 2_{\overline{1} \lor 2} \quad \| \quad \overline{1} \lor 2,\ \overline{3} \lor 4,\ \overline{5} \lor \overline{6},\ 6 \lor \overline{5} \lor \overline{2} \quad \Rightarrow \quad \text{(Decide)}$$

$$1\ 2_{\overline{1} \lor 2}\ 3 \quad \| \quad \overline{1} \lor 2,\ \overline{3} \lor 4,\ \overline{5} \lor \overline{6},\ 6 \lor \overline{5} \lor \overline{2} \quad \Rightarrow \quad \text{(UnitProp)}$$

$$1\ 2_{\overline{1} \lor 2}\ 3\ 4_{\overline{3} \lor 4} \quad \| \quad \overline{1} \lor 2,\ \overline{3} \lor 4,\ \overline{5} \lor \overline{6},\ 6 \lor \overline{5} \lor \overline{2}$$

Abstract DPLL: Example

$$\begin{array}{rcll}
 & \parallel & \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \text{(Decide)} \\
1 & \parallel & \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \text{(UnitProp)} \\
1\ 2_{\overline{1} \vee 2} & \parallel & \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \text{(Decide)} \\
1\ 2_{\overline{1} \vee 2}\ 3 & \parallel & \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \text{(UnitProp)} \\
1\ 2_{\overline{1} \vee 2}\ 3\ 4_{\overline{3} \vee 4} & \parallel & \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \text{(Decide)} \\
1\ 2_{\overline{1} \vee 2}\ 3\ 4_{\overline{3} \vee 4}\ 5 & \parallel & \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} &
\end{array}$$

Abstract DPLL: Example

$$\| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(Decide)}$$

$$1 \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(UnitProp)}$$

$$1\ 2_{\overline{1} \vee 2} \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(Decide)}$$

$$1\ 2_{\overline{1} \vee 2}\ 3 \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(UnitProp)}$$

$$1\ 2_{\overline{1} \vee 2}\ 3\ 4_{\overline{3} \vee 4} \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(Decide)}$$

$$1\ 2_{\overline{1} \vee 2}\ 3\ 4_{\overline{3} \vee 4}\ 5 \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(UnitProp)}$$

$$1\ 2_{\overline{1} \vee 2}\ 3\ 4_{\overline{3} \vee 4}\ 5\ \overline{6}_{\overline{5} \vee \overline{6}} \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$$

Abstract DPLL: Example

$$
\begin{array}{rcll}
 & \| & \overline{1} \lor 2,\ \overline{3} \lor 4,\ \overline{5} \lor \overline{6},\ 6 \lor \overline{5} \lor \overline{2} & \Rightarrow \text{(Decide)} \\
1 & \| & \overline{1} \lor 2,\ \overline{3} \lor 4,\ \overline{5} \lor \overline{6},\ 6 \lor \overline{5} \lor \overline{2} & \Rightarrow \text{(UnitProp)} \\
1\ 2_{\overline{1} \lor 2} & \| & \overline{1} \lor 2,\ \overline{3} \lor 4,\ \overline{5} \lor \overline{6},\ 6 \lor \overline{5} \lor \overline{2} & \Rightarrow \text{(Decide)} \\
1\ 2_{\overline{1} \lor 2}\ 3 & \| & \overline{1} \lor 2,\ \overline{3} \lor 4,\ \overline{5} \lor \overline{6},\ 6 \lor \overline{5} \lor \overline{2} & \Rightarrow \text{(UnitProp)} \\
1\ 2_{\overline{1} \lor 2}\ 3\ 4_{\overline{3} \lor 4} & \| & \overline{1} \lor 2,\ \overline{3} \lor 4,\ \overline{5} \lor \overline{6},\ 6 \lor \overline{5} \lor \overline{2} & \Rightarrow \text{(Decide)} \\
1\ 2_{\overline{1} \lor 2}\ 3\ 4_{\overline{3} \lor 4}\ 5 & \| & \overline{1} \lor 2,\ \overline{3} \lor 4,\ \overline{5} \lor \overline{6},\ 6 \lor \overline{5} \lor \overline{2} & \Rightarrow \text{(UnitProp)} \\
1\ 2_{\overline{1} \lor 2}\ 3\ 4_{\overline{3} \lor 4}\ 5\ \overline{6}_{\overline{5} \lor \overline{6}} & \| & \overline{1} \lor 2,\ \overline{3} \lor 4,\ \overline{5} \lor \overline{6},\ 6 \lor \overline{5} \lor \overline{2} & \Rightarrow \text{(Conflict)} \\
1\ 2_{\overline{1} \lor 2}\ 3\ 4_{\overline{3} \lor 4}\ 5\ \overline{6}_{\overline{5} \lor \overline{6}} & \| & \overline{1} \lor 2,\ \overline{3} \lor 4,\ \overline{5} \lor \overline{6},\ 6 \lor \overline{5} \lor \overline{2} & \|\ 6 \lor \overline{5} \lor \overline{2}
\end{array}
$$

Abstract DPLL: Example

$$
\begin{array}{rcll}
 & \| & \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \text{(Decide)} \\
1 & \| & \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \text{(UnitProp)} \\
1\ 2_{\overline{1} \vee 2} & \| & \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \text{(Decide)} \\
1\ 2_{\overline{1} \vee 2}\ 3 & \| & \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \text{(UnitProp)} \\
1\ 2_{\overline{1} \vee 2}\ 3\ 4_{\overline{3} \vee 4} & \| & \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \text{(Decide)} \\
1\ 2_{\overline{1} \vee 2}\ 3\ 4_{\overline{3} \vee 4}\ 5 & \| & \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \text{(UnitProp)} \\
1\ 2_{\overline{1} \vee 2}\ 3\ 4_{\overline{3} \vee 4}\ 5\ \overline{6}_{\overline{5} \vee \overline{6}} & \| & \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \text{(Conflict)} \\
1\ 2_{\overline{1} \vee 2}\ 3\ 4_{\overline{3} \vee 4}\ 5\ \overline{6}_{\overline{5} \vee \overline{6}} & \| & \underbrace{\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}}_{F} & \|\ \ 6 \vee \overline{5} \vee \overline{2}
\end{array}
$$

Abstract DPLL: Example (cont.)

$$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}}\ \ \|\ \ F \qquad\qquad \|\ \ 6 \vee \overline{5} \vee \overline{2}$$

Abstract DPLL: Example (cont.)

$$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}}\ \parallel\ F\ \parallel\ 6\vee\overline{5}\vee\overline{2}\quad\Rightarrow\quad\text{(Resolve)}$$

$$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}}\ \parallel\ F\ \parallel\ \overline{5}\vee\overline{2}$$

Abstract DPLL: Example (cont.)

$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}}$	$\parallel\ F$	$\parallel\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(Resolve)
$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}}$	$\parallel\ F$	$\parallel\ \overline{5} \vee \overline{2}$	$\Rightarrow$	(Learn)
$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}}$	$\parallel\ F,\ \overline{5} \vee \overline{2}$	$\parallel\ \overline{5} \vee \overline{2}$		

Abstract DPLL: Example (cont.)

$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}}$	$\|$ F	$\|$ $6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(Resolve)
$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}}$	$\|$ F	$\|$ $\overline{5} \vee \overline{2}$	$\Rightarrow$	(Learn)
$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}}$	$\|$ $F,\ \overline{5} \vee \overline{2}$	$\|$ $\overline{5} \vee \overline{2}$	$\Rightarrow$	(Resolve)
$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}}$	$\|$ $F,\ \overline{5} \vee \overline{2}$	$\|$ $\overline{5} \vee \overline{1}$		

Abstract DPLL: Example (cont.)

$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}}$	$\parallel\ F$	$\parallel\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Resolve)
$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}}$	$\parallel\ F$	$\parallel\ \overline{5}\vee\overline{2}$	$\Rightarrow$	(Learn)
$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}}$	$\parallel\ F,\ \overline{5}\vee\overline{2}$	$\parallel\ \overline{5}\vee\overline{2}$	$\Rightarrow$	(Resolve)
$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}}$	$\parallel\ F,\ \overline{5}\vee\overline{2}$	$\parallel\ \overline{5}\vee\overline{1}$	$\Rightarrow$	(Backjump)
$1\ 2_{\overline{1}\vee 2}\ \overline{5}_{\overline{5}\vee\overline{1}}$	$\parallel\ F,\ \overline{5}\vee\overline{2}$			

Abstract DPLL: Termination

- Each decision defines a new scope level.
- Metric: number of assigned literals per level

$$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}} \quad \mapsto \quad (2,2,2)$$

$$1\ 2_{\overline{1}\vee 2}\ \overline{5}_{\overline{5}\vee\overline{1}} \quad \mapsto \quad (3)$$

- Order: lexicographic ordering on the metric: (e..g., $(3,1) > (2,2,4,1)$)
- **Decide**, **UnitPropagate**, and **Backjump** increase the metric.
- It can not increase forever (finite number of variables).
- Conflict resolution rules (**Conflict**, **Resolve**, **Learn**) are also terminating.

Abstract DPLL: Strategy

- Abstract DPLL is very flexible.
- Basic Strategy:
  - Only apply **Decide** if **UnitPropagate** and **Conflict** cannot be applied.
- Conflict Resolution:
  - Learn only one clause per conflict (the clause used in **Backjump**).
  - Use **Backjump** as soon as possible (FUIP).
  - Use the rightmost (applicable) literal in M when applying **Resolve**.

$$M \parallel F \parallel C \vee \bar{l} \quad \Longrightarrow \quad M \parallel F \parallel D \vee C \qquad \textbf{if} \;\; l_{D \vee l} \in M, \qquad (\textbf{Resolve})$$

Abstract DPLL: Decision Strategy

- Decision heuristic:
  - Associate a score with each boolean variable.
  - Select the variable with highest score when **Decide** is used.
  - Increase by δ the score of *var*(l) when **Resolve** is used:

$$M \parallel F \parallel C \vee \bar{l} \quad \Longrightarrow \quad M \parallel F \parallel D \vee C \qquad \text{if } \; l_{D \vee l} \in M, \qquad \textbf{(Resolve)}$$

  - Increase the score of every variable in the clause $C \vee l$ when **Backjump** is used:

$$M\, l'\, M' \parallel F \parallel C \vee l \quad \Longrightarrow \quad M\, l_{C \vee l} \parallel F' \qquad \text{if } \begin{cases} M \models \neg C, \\ l \text{ is undefined in } M \end{cases} \qquad \textbf{(Backjump)}$$

  - After each conflict: slightly increase the value of δ.
  - From time to time renormalize the scores and δ to avoid overflows.

Abstract DPLL: Phase Selection

- Assume p was selected by a decision strategy.

  Should we assign p or $\neg p$ in **Decide**?

  Always False Guess $\neg p$ (works well in practice).

  Always True Guess p.

  Score Associate a score with each literal instead of each variable.

  Pick the phase with highest score.

  Caching Caches the last phase of variables during conflict resolution. Improvement: except for variables in the last decision level.

  Greedy Select the phase that satisfies most clauses.

Abstract DPLL: Extra Rules

- Extra rules:

$M \parallel F, C$	$\Longrightarrow$	$M \parallel F$	**if** C is a learned clause	(**Forget**)
$M \parallel F$	$\Longrightarrow$	$\parallel F$		(**Restart**)

- **Forget** in practice:
  - Associate a score with each learned clause C.
  - Increase by δ_c the score of $D \vee l$ when **Resolve** is used.

    $M \parallel F \parallel C \vee \bar{l} \quad \Longrightarrow \quad M \parallel F \parallel D \vee C \qquad$ **if** $l_{D \vee l} \in M$, (**Resolve**)

  - From time to time use **Forget** to delete learned clauses with low score.

Abstract DPLL: Restart Strategies

No restarts

Linear Restart after every k conflicts, update $k := k + \delta$.

Geometric Restart after every k conflicts, update $k := k \times \delta$.

Inner-Out Geometric “Two dimensional pattern” that increases in both dimensions.

- Initially $k := x$, the inner loop multiplies k by δ at each restart.
- When $k > y$, $k := x$ and $y := y \times \delta$.

Luby Restarts are performed according to the following series: $1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, \ldots$, multiplied by a constant c (e.g., 100, 256, 512).

$$\textit{luby}(i) = \begin{cases} 2^{k-1}, & \text{if } \exists k.\ i = 2^k - 1 \\ \textit{luby}(i - 2^{k-1} + 1), & \text{if } \exists k.\ 2^{k-1} \leq i < 2^k - 1 \end{cases}$$

Indexing

- Indexing techniques are very important.
- How to implement **UnitPropagate** and **Conflict**?
- Scanning the set of clauses will not scale.
- Simple index: mapping from literals to clauses (occurrences).
  - *watch*$(l) = \{C_1, \ldots, C_n\}$, where $\bar{l} \in C_i$
  - If l is assigned, check each clause $C \in$ *watch*(l) for **UnitPropagate** and **Conflict**.
  - Most of the time C has more than one unassigned literal.
  - Improvement: associate a counter u with each clause (number of unassigned literals).
  - Problem: counters must be decremented when literals are assigned, and restored during **Backjump**.

Indexing: Two Watch Literal

- Insight:
  - No need to include clause C in every set *watch*(l) where $\bar{l} \in C$.
  - It suffices to include C in at most 2 such sets.
- Invariant:

  If some literal l in C is not assigned to false, then $C \in$ *watch*(l') of some l' that is not assigned to false.

Indexing: Two watch Literal

- Maintain 2-watch invariant:
  - Whenever l is assigned.
  - For each clause $C \in \textit{watch}(l)$
    - If the other watch literal l' ($C \in \textit{watch}(l')$) is assigned to true, then do nothing.
    - Else if some other literal l' is true or unassigned

$$\begin{aligned} \textit{watch}(l') &:= \textit{watch}(l') \cup \{C\} \\ \textit{watch}(l) &:= \textit{watch}(l) \setminus \{C\} \end{aligned}$$

    - Else if all literals in C are assigned to false, then **Backjump**.
    - Else (all but one literal in C is assigned to false) **Propagate**.

Preprocessing

- Preprocessing is very important for industrial benchmarks.
- Example simplification rules
  - Apply subsumption to remove clauses: C subsumes D if $C \subseteq D$, then D can be removed.
  - Apply resolution to eliminate variables provided this does not create too many new clauses:
    - *occs*$(l) = \{$clauses that contain $l\}$
    - $|$*occs*$(p)| * |$*occs*$(\neg p)| < k$
    - $|$*occs*$(p)| = 1$ or $|$*occs*$(\neg p)| = 1$