Selected Topics in AI

Practical SAT Solving

Martin Suda
(based on slides of Mikoláš Janota and João Marques-Silva)

4 May 2026

Outline

Propositional Logics as a Formal Language

Propositional Logics as a Formal Language

Syntax

Propositional Logics as a Formal Language

Syntax

- a set of (propositional) variables $\mathcal{V}$

Propositional Logics as a Formal Language

Syntax

- a set of (propositional) variables $\mathcal{V}$
- formula φ (defined inductively):

Propositional Logics as a Formal Language

Syntax

- a set of (propositional) variables $\mathcal{V}$
- formula φ (defined inductively):
  - φ is atomic: $\top, \bot, v$ for $v \in \mathcal{V}$

Propositional Logics as a Formal Language

Syntax

- a set of (propositional) variables $\mathcal{V}$
- formula φ (defined inductively):
  - φ is atomic: $\top, \bot, v$ for $v \in \mathcal{V}$
  - φ has one of the following forms
    $\neg\varphi, \varphi \wedge \varphi, \varphi \vee \varphi, \varphi \to \varphi, \varphi \leftrightarrow \varphi, \ldots$

Propositional Logics as a Formal Language

Syntax

- a set of (propositional) variables $\mathcal{V}$
- formula φ (defined inductively):
  - φ is atomic: $\top, \bot, v$ for $v \in \mathcal{V}$
  - φ has one of the following forms
    $\neg\varphi, \varphi \wedge \varphi, \varphi \vee \varphi, \varphi \to \varphi, \varphi \leftrightarrow \varphi, \ldots$

Semantics

- truth values: $\mathcal{B} = \{0, 1\}$

Propositional Logics as a Formal Language

Syntax

- a set of (propositional) variables $\mathcal{V}$
- formula φ (defined inductively):
  - φ is atomic: $\top, \bot, v$ for $v \in \mathcal{V}$
  - φ has one of the following forms
    $\neg\varphi, \varphi \wedge \varphi, \varphi \vee \varphi, \varphi \to \varphi, \varphi \leftrightarrow \varphi, \ldots$

Semantics

- truth values: $\mathcal{B} = \{0, 1\}$
- variable assignment: $A : \mathcal{V} \to \mathcal{B}$

Propositional Logics as a Formal Language

Syntax

- a set of (propositional) variables $\mathcal{V}$
- formula φ (defined inductively):
  - φ is atomic: $\top, \bot, v$ for $v \in \mathcal{V}$
  - φ has one of the following forms
    $\neg\varphi, \varphi \wedge \varphi, \varphi \vee \varphi, \varphi \to \varphi, \varphi \leftrightarrow \varphi, \ldots$

Semantics

- truth values: $\mathcal{B} = \{0, 1\}$
- variable assignment: $A : \mathcal{V} \to \mathcal{B}$
- assignment makes a formula true (defined recursively):

Propositional Logics as a Formal Language

Syntax

- a set of (propositional) variables $\mathcal{V}$
- formula φ (defined inductively):
  - φ is atomic: $\top, \bot, v$ for $v \in \mathcal{V}$
  - φ has one of the following forms
    $\neg\varphi, \varphi \wedge \varphi, \varphi \vee \varphi, \varphi \to \varphi, \varphi \leftrightarrow \varphi, \ldots$

Semantics

- truth values: $\mathcal{B} = \{0, 1\}$
- variable assignment: $A : \mathcal{V} \to \mathcal{B}$
- assignment makes a formula true (defined recursively):
  - $A \models \top$, not $A \models \bot$, $A \models v$ iff $A(v) = 1$,

Propositional Logics as a Formal Language

Syntax

- a set of (propositional) variables $\mathcal{V}$
- formula φ (defined inductively):
  - φ is atomic: $\top, \bot, v$ for $v \in \mathcal{V}$
  - φ has one of the following forms
    $\neg\varphi, \varphi \wedge \varphi, \varphi \vee \varphi, \varphi \to \varphi, \varphi \leftrightarrow \varphi, \ldots$

Semantics

- truth values: $\mathcal{B} = \{0, 1\}$
- variable assignment: $A : \mathcal{V} \to \mathcal{B}$
- assignment makes a formula true (defined recursively):
  - $A \models \top$, not $A \models \bot$, $A \models v$ iff $A(v) = 1$,
  - $A \models \neg\varphi$ iff not $A \models \varphi$,

Propositional Logics as a Formal Language

Syntax

- a set of (propositional) variables $\mathcal{V}$
- formula φ (defined inductively):
  - φ is atomic: $\top, \bot, v$ for $v \in \mathcal{V}$
  - φ has one of the following forms
    $\neg\varphi, \varphi \wedge \varphi, \varphi \vee \varphi, \varphi \to \varphi, \varphi \leftrightarrow \varphi, \ldots$

Semantics

- truth values: $\mathcal{B} = \{0, 1\}$
- variable assignment: $A : \mathcal{V} \to \mathcal{B}$
- assignment makes a formula true (defined recursively):
  - $A \models \top$, not $A \models \bot$, $A \models v$ iff $A(v) = 1$,
  - $A \models \neg\varphi$ iff not $A \models \varphi$,
  - $A \models \varphi_1 \wedge \varphi_2$ iff $A \models \varphi_1$ and $A \models \varphi_2$,

Propositional Logics as a Formal Language

Syntax

- a set of (propositional) variables $\mathcal{V}$
- formula φ (defined inductively):
  - φ is atomic: $\top, \bot, v$ for $v \in \mathcal{V}$
  - φ has one of the following forms
    $\neg\varphi, \varphi \wedge \varphi, \varphi \vee \varphi, \varphi \to \varphi, \varphi \leftrightarrow \varphi, \ldots$

Semantics

- truth values: $\mathcal{B} = \{0, 1\}$
- variable assignment: $A : \mathcal{V} \to \mathcal{B}$
- assignment makes a formula true (defined recursively):
  - $A \models \top$, not $A \models \bot$, $A \models v$ iff $A(v) = 1$,
  - $A \models \neg\varphi$ iff not $A \models \varphi$,
  - $A \models \varphi_1 \wedge \varphi_2$ iff $A \models \varphi_1$ and $A \models \varphi_2$,
  - $A \models \varphi_1 \to \varphi_2$ iff not $A \models \varphi_1$ or $A \models \varphi_2$,

Propositional Logics as a Formal Language

Syntax

- a set of (propositional) variables $\mathcal{V}$
- formula φ (defined inductively):
  - φ is atomic: $\top, \bot, v$ for $v \in \mathcal{V}$
  - φ has one of the following forms
    $\neg\varphi, \varphi \wedge \varphi, \varphi \vee \varphi, \varphi \to \varphi, \varphi \leftrightarrow \varphi, \ldots$

Semantics

- truth values: $\mathcal{B} = \{0, 1\}$
- variable assignment: $A : \mathcal{V} \to \mathcal{B}$
- assignment makes a formula true (defined recursively):
  - $A \models \top$, not $A \models \bot$, $A \models v$ iff $A(v) = 1$,
  - $A \models \neg\varphi$ iff not $A \models \varphi$,
  - $A \models \varphi_1 \wedge \varphi_2$ iff $A \models \varphi_1$ and $A \models \varphi_2$,
  - $A \models \varphi_1 \to \varphi_2$ iff not $A \models \varphi_1$ or $A \models \varphi_2$,
  - ...

“To Solve a Logical Formula”?

Satistifiability

A formula φ is *satisfiable* iff there is an assignment A s.t. $A \models \varphi$.

“To Solve a Logical Formula”?

Satistifiability

A formula φ is *satisfiable* iff there is an assignment A s.t. $A \models \varphi$.

Tautologyhood

A formula φ is *logically valid* (a tautology) iff for every assignment A we have $A \models \varphi$.

Further Important Notions

Entailment

A set of formulas Ψ logically entails a formula φ, written $\Psi \models \varphi$,
iff for every assignment A: if $A \models \psi$ for every $\psi \in \Psi$ then $A \models \varphi$.

Further Important Notions

Entailment

A set of formulas Ψ logically entails a formula φ, written $\Psi \models \varphi$,
iff for every assignment A: if $A \models \psi$ for every $\psi \in \Psi$ then $A \models \varphi$.

Note: We can write $\models \varphi$ in the sense that φ is a tautology.
(empty set of formulas on the l.h.s. of $\models$)

Further Important Notions

Entailment

A set of formulas Ψ logically entails a formula φ, written $\Psi \models \varphi$,
iff for every assignment A: if $A \models \psi$ for every $\psi \in \Psi$ then $A \models \varphi$.

Note: We can write $\models \varphi$ in the sense that φ is a tautology.
(empty set of formulas on the l.h.s. of $\models$)

Entailment preserves satisfiability

If Ψ is satisfiable and $\Psi \models \varphi$ then φ is satisfiable.

Further Important Notions

Entailment

A set of formulas Ψ logically entails a formula φ, written $\Psi \models \varphi$,
iff for every assignment A: if $A \models \psi$ for every $\psi \in \Psi$ then $A \models \varphi$.

Note: We can write $\models \varphi$ in the sense that φ is a tautology.
(empty set of formulas on the l.h.s. of $\models$)

Entailment preserves satisfiability

If Ψ is satisfiable and $\Psi \models \varphi$ then φ is satisfiable.
(Actually, $\Psi \wedge \varphi$ is satisfiable.)

Further Important Notions

Entailment

A set of formulas Ψ logically entails a formula φ, written $\Psi \models \varphi$,
iff for every assignment A: if $A \models \psi$ for every $\psi \in \Psi$ then $A \models \varphi$.

Note: We can write $\models \varphi$ in the sense that φ is a tautology.
(empty set of formulas on the l.h.s. of $\models$)

Entailment preserves satisfiability

If Ψ is satisfiable and $\Psi \models \varphi$ then φ is satisfiable.
(Actually, $\Psi \wedge \varphi$ is satisfiable.)

Equivalence

Formulas φ and ψ are equivalent iff $\varphi \models \psi$ and $\psi \models \varphi$.

Further Important Notions

Entailment

A set of formulas Ψ logically entails a formula φ, written $\Psi \models \varphi$,
iff for every assignment A: if $A \models \psi$ for every $\psi \in \Psi$ then $A \models \varphi$.

Note: We can write $\models \varphi$ in the sense that φ is a tautology.
(empty set of formulas on the l.h.s. of $\models$)

Entailment preserves satisfiability

If Ψ is satisfiable and $\Psi \models \varphi$ then φ is satisfiable.
(Actually, $\Psi \wedge \varphi$ is satisfiable.)

Equivalence

Formulas φ and ψ are equivalent iff $\varphi \models \psi$ and $\psi \models \varphi$.

Note: Equivalent formulas have the same set of satisfying assignments.

Outline

Today: What makes modern SAT solvers tick?

A first approach:

- Solve SAT by backtracking
- For n variables 2^n assignments ... highly impractical

Today: What makes modern SAT solvers tick?

A first approach:

- Solve SAT by backtracking
- For n variables 2^n assignments ... highly impractical

What can we do?

- Identify consequences — unit propagation
  e.g. if $X = 1$ and $X \to Y$, then $Y = 1$

Today: What makes modern SAT solvers tick?

A first approach:

- Solve SAT by backtracking
- For n variables 2^n assignments ... highly impractical

What can we do?

- Identify consequences — unit propagation
  e.g. if $X = 1$ and $X \to Y$, then $Y = 1$
- "Avoid repeating same mistake twice" — clause learning

Today: What makes modern SAT solvers tick?

A first approach:

- Solve SAT by backtracking
- For n variables 2^n assignments ... highly impractical

What can we do?

- Identify consequences — unit propagation
  e.g. if $X = 1$ and $X \to Y$, then $Y = 1$
- "Avoid repeating same mistake twice" — clause learning
- ... *among other things*, e.g. restarts

Outline

Preliminaries – CNF Formulas

- Variables: $w, x, y, z, a, b, c, \ldots$
- Literals: $w, \bar{x}, \bar{y}, a, \ldots$, but also $\neg w, \neg y, \ldots$
- Clauses: disjunction of literals or set of literals
- Formula: conjunction of clauses or set of clauses
- Model (satisfying assignment): partial/total mapping from variables to $\{0, 1\}$ that satisfies formula
- Formula can be SAT/UNSAT

Preliminaries – CNF Formulas

- Variables: $w, x, y, z, a, b, c, \ldots$
- Literals: $w, \bar{x}, \bar{y}, a, \ldots$, but also $\neg w, \neg y, \ldots$
- Clauses: disjunction of literals or set of literals
- Formula: conjunction of clauses or set of clauses
- Model (satisfying assignment): partial/total mapping from variables to $\{0, 1\}$ that satisfies formula
- Formula can be SAT/UNSAT
- Example:

$$\mathcal{F} \triangleq (r) \wedge (\bar{r} \vee s) \wedge (\bar{w} \vee a) \wedge (\bar{x} \vee b) \wedge (\bar{y} \vee \bar{z} \vee c) \wedge (\bar{b} \vee \bar{c} \vee d)$$

  - Example models:
    - $\{r, s, a, b, c, d\}$
    - $\{r, s, \bar{x}, y, \bar{w}, z, \bar{a}, b, c, d\}$

Resolution

- Resolution rule: [DP60,R65]

$$\frac{(\alpha \vee x) \qquad (\beta \vee \bar{x})}{(\alpha \vee \beta)}$$

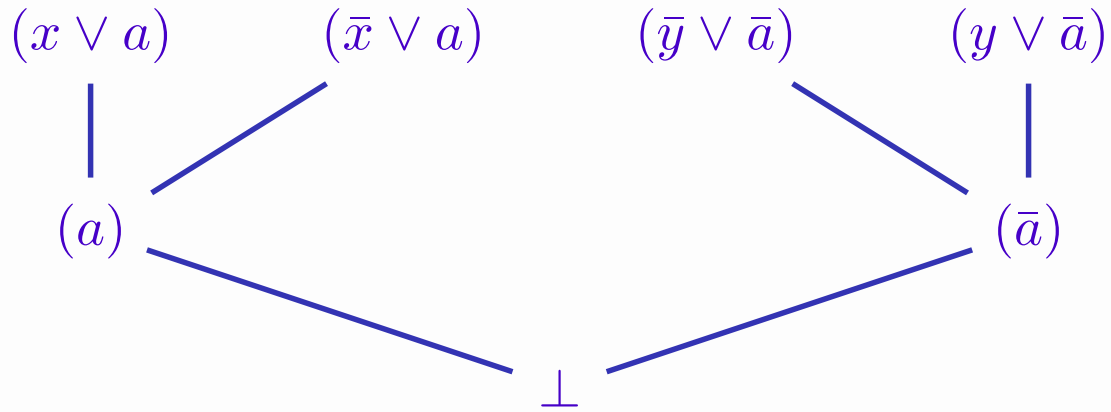
  - Refutationally complete proof system for propositional logic

Resolution

- Resolution rule: [DP60,R65]

$$\frac{(\alpha \vee x) \qquad (\beta \vee \bar{x})}{(\alpha \vee \beta)}$$

  - Refutationally complete proof system for propositional logic



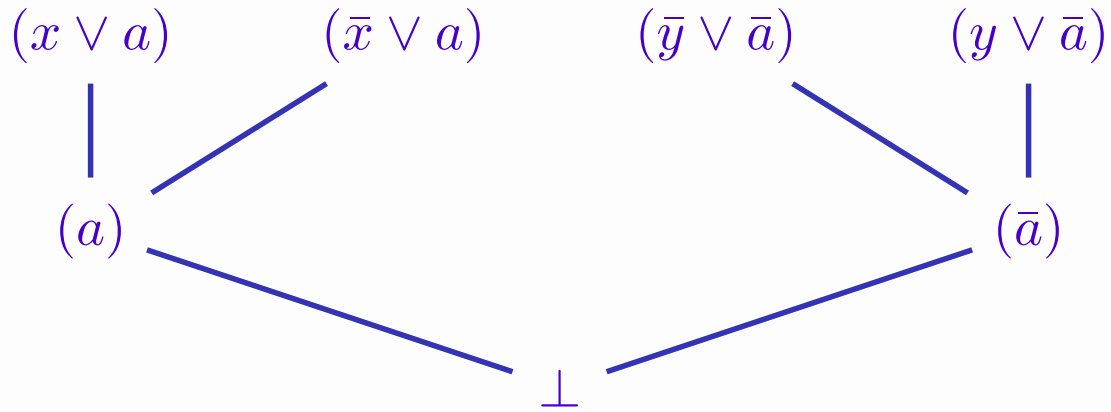
  - Extensively used with (CDCL) SAT solvers

Resolution

- Resolution rule: [DP60,R65]

$$\frac{(\alpha \vee x) \qquad (\beta \vee \bar{x})}{(\alpha \vee \beta)}$$

  - Refutationally complete proof system for propositional logic



  - Extensively used with (CDCL) SAT solvers

- Self-subsuming resolution (with $\alpha' \subseteq \alpha$): [e.g. SP04,EB05]

$$\frac{(\alpha \vee x) \qquad (\alpha' \vee \bar{x})}{(\alpha)}$$

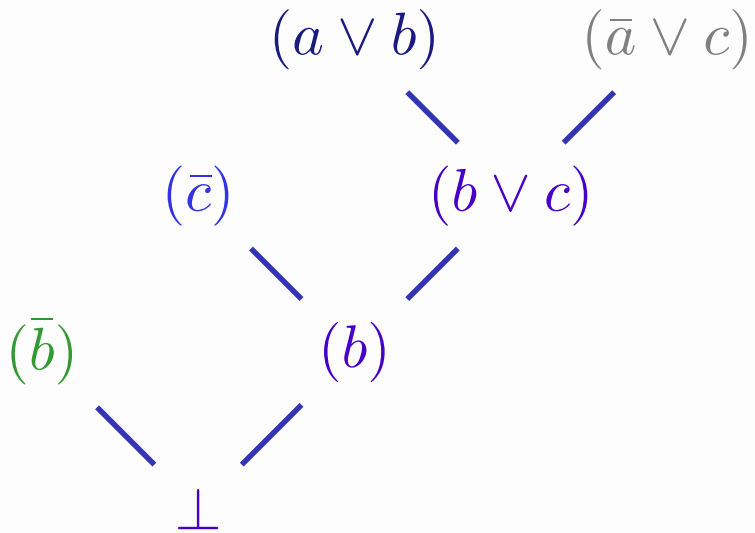
  - (α) subsumes $(\alpha \vee x)$

Resolution Proofs

- Refutation of unsatisfiable formula by iterated resolution operations produces resolution proof
- An example:
  $\mathcal{F} = (\bar{c}) \wedge (\bar{b}) \wedge (\bar{a} \vee c) \wedge (a \vee b) \wedge (a \vee \bar{d}) \wedge (\bar{a} \vee \bar{d})$

Resolution Proofs

- Refutation of unsatisfiable formula by iterated resolution operations produces resolution proof
- An example:
  $\mathcal{F} = (\bar{c}) \wedge (\bar{b}) \wedge (\bar{a} \vee c) \wedge (a \vee b) \wedge (a \vee \bar{d}) \wedge (\bar{a} \vee \bar{d})$
- Resolution proof:



- SAT solvers can generate resolution proofs from learned clauses

[ZM03]

Unit Propagation

$$
\begin{array}{rcl|l}
\mathcal{F} & = & (r) & \mathcal{A} \\
 & \wedge & (\bar{r} \vee s) & \mathcal{B} \\
 & \wedge & (\bar{w} \vee a) & \mathcal{C} \\
 & \wedge & (\bar{x} \vee \bar{a} \vee b) & \mathcal{D} \\
 & \wedge & (\bar{y} \vee \bar{z} \vee c) & \mathcal{E} \\
 & \wedge & (\bar{b} \vee \bar{c} \vee d) & \mathcal{F}
\end{array}
$$

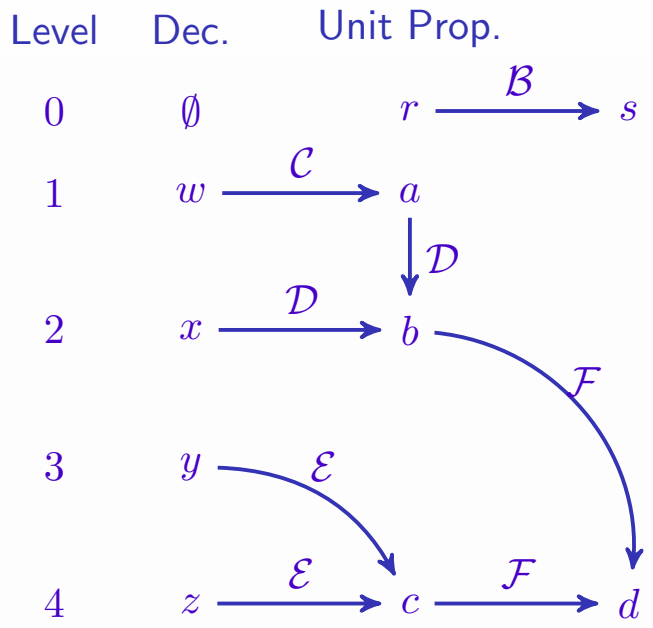
Unit Propagation

$$
\begin{array}{rcl|l}
\mathcal{F} & = & (r) & \mathcal{A} \\
 & \wedge & (\bar{r} \vee s) & \mathcal{B} \\
 & \wedge & (\bar{w} \vee a) & \mathcal{C} \\
 & \wedge & (\bar{x} \vee \bar{a} \vee b) & \mathcal{D} \\
 & \wedge & (\bar{y} \vee \bar{z} \vee c) & \mathcal{E} \\
 & \wedge & (\bar{b} \vee \bar{c} \vee d) & \mathcal{F}
\end{array}
$$

- Decisions / Variable Branchings:
  $w = 1, x = 1, y = 1, z = 1$

Unit Propagation

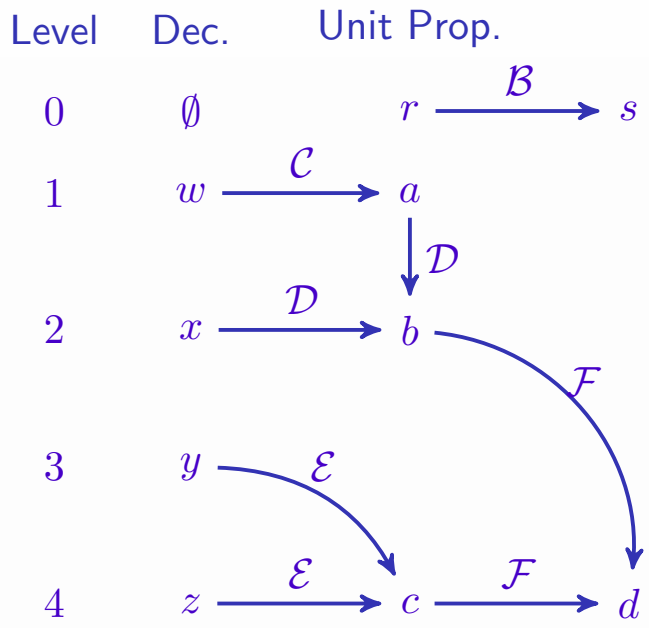
$$
\begin{array}{rcl|l}
\mathcal{F} & = & (r) & \mathcal{A} \\
 & \wedge & (\bar{r} \vee s) & \mathcal{B} \\
 & \wedge & (\bar{w} \vee a) & \mathcal{C} \\
 & \wedge & (\bar{x} \vee \bar{a} \vee b) & \mathcal{D} \\
 & \wedge & (\bar{y} \vee \bar{z} \vee c) & \mathcal{E} \\
 & \wedge & (\bar{b} \vee \bar{c} \vee d) & \mathcal{F}
\end{array}
$$



- Decisions / Variable Branchings:
  $w = 1, x = 1, y = 1, z = 1$

Unit Propagation

$$
\begin{array}{rcl|l}
\mathcal{F} & = & (r) & \mathcal{A} \\
 & \wedge & (\bar{r} \vee s) & \mathcal{B} \\
 & \wedge & (\bar{w} \vee a) & \mathcal{C} \\
 & \wedge & (\bar{x} \vee \bar{a} \vee b) & \mathcal{D} \\
 & \wedge & (\bar{y} \vee \bar{z} \vee c) & \mathcal{E} \\
 & \wedge & (\bar{b} \vee \bar{c} \vee d) & \mathcal{F}
\end{array}
$$



- Decisions / Variable Branchings:
  $w = 1, x = 1, y = 1, z = 1$

- Additional definitions:
  - Antecedent (or reason) of an implied assignment
    - $(\bar{b} \vee \bar{c} \vee d)$ for d
  - Associate assignment with decision levels
    - $w = 1 @ 1$, $x = 1 @ 2$, $y = 1 @ 3$, $z = 1 @ 4$
    - $r = 1 @ 0$, $d = 1 @ 4$, ...

Unit Propagation & Conflict

- Deriving the empty clause is called a conflict
- The clause that became empty is called the conflict clause

Unit Propagation & Conflict

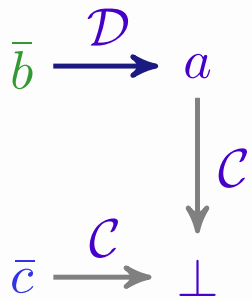
- Deriving the empty clause is called a conflict
- The clause that became empty is called the conflict clause

$$\begin{array}{rcl|l} \mathcal{F} & = & (\bar{c}) & \mathcal{A} \\ & \wedge & (\bar{b}) & \mathcal{B} \\ & \wedge & (\bar{a} \vee c) & \mathcal{C} \\ & \wedge & (a \vee b) & \mathcal{D} \\ & \wedge & (a \vee \bar{d}) & \mathcal{E} \\ & \wedge & (\bar{a} \vee \bar{d}) & \mathcal{F} \end{array}$$

Unit Propagation & Conflict

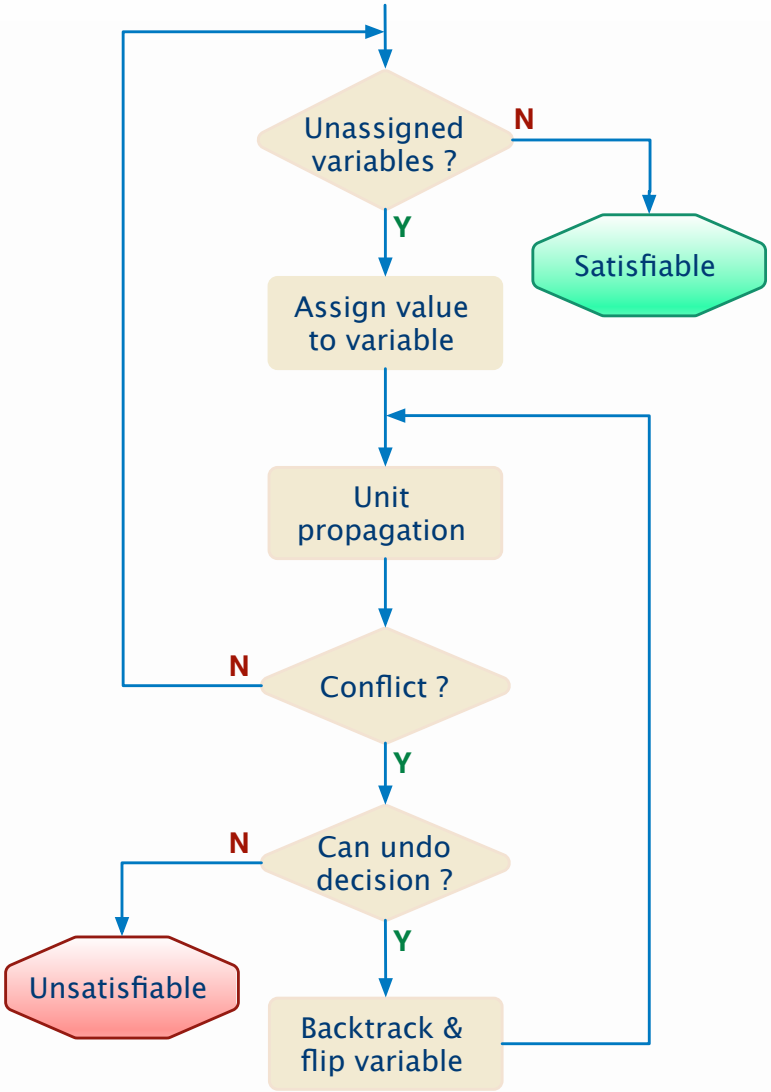
- Deriving the empty clause is called a conflict
- The clause that became empty is called the conflict clause

$$\begin{array}{rcl|l} \mathcal{F} & = & (\bar{c}) & \mathcal{A} \\ & \wedge & (\bar{b}) & \mathcal{B} \\ & \wedge & (\bar{a} \vee c) & \mathcal{C} \\ & \wedge & (a \vee b) & \mathcal{D} \\ & \wedge & (a \vee \bar{d}) & \mathcal{E} \\ & \wedge & (\bar{a} \vee \bar{d}) & \mathcal{F} \end{array}$$

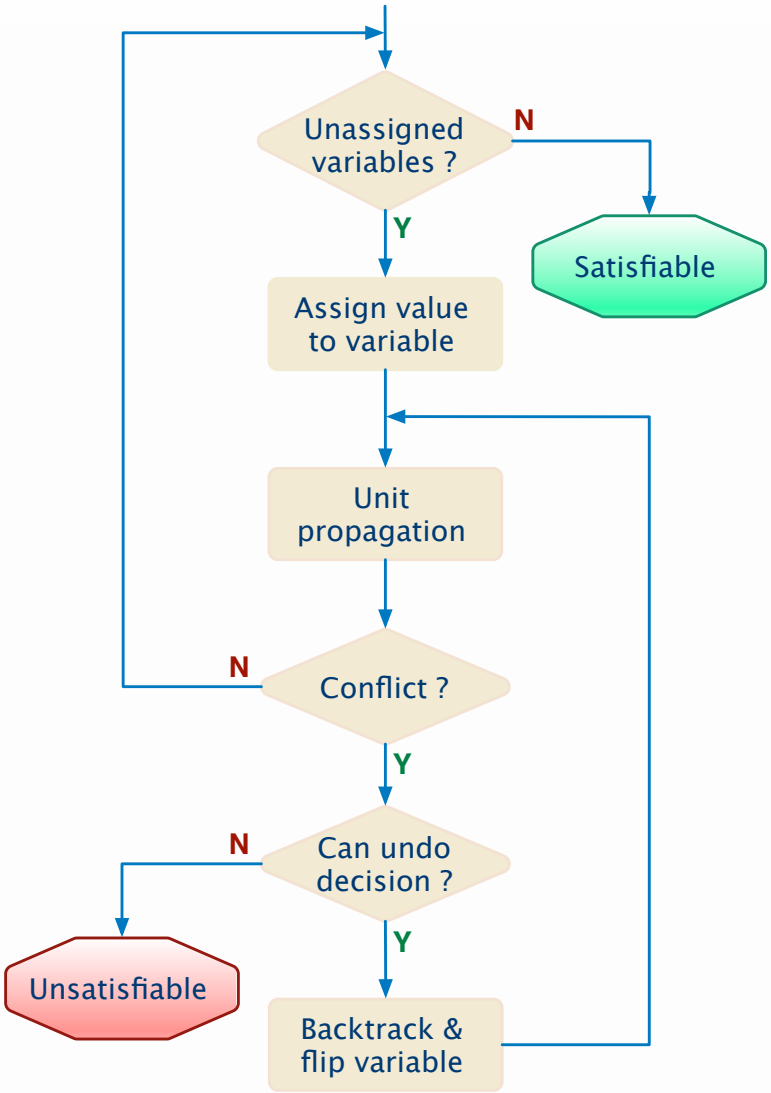
Level	Dec.	Unit Prop.
0	$\emptyset$	

Outline

The DPLL Algorithm

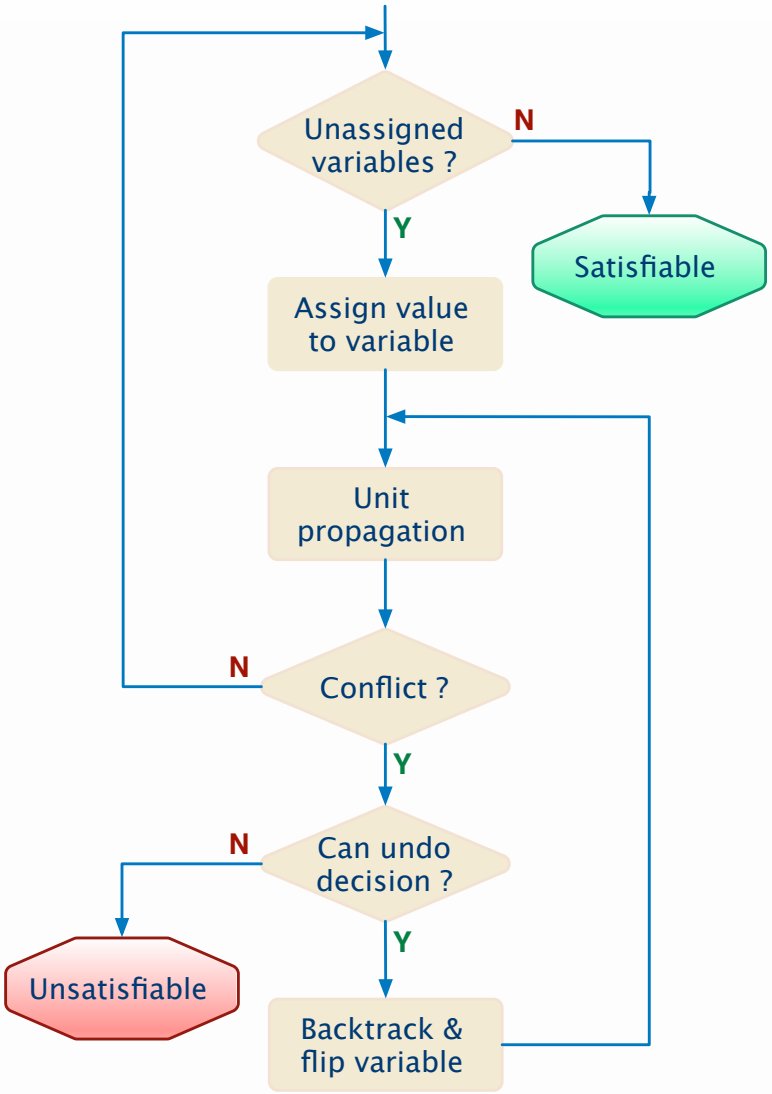


The DPLL Algorithm



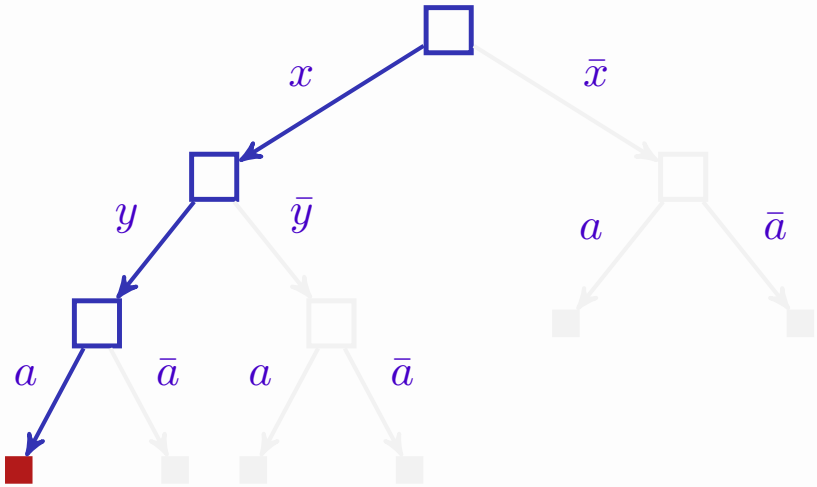
$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

The DPLL Algorithm

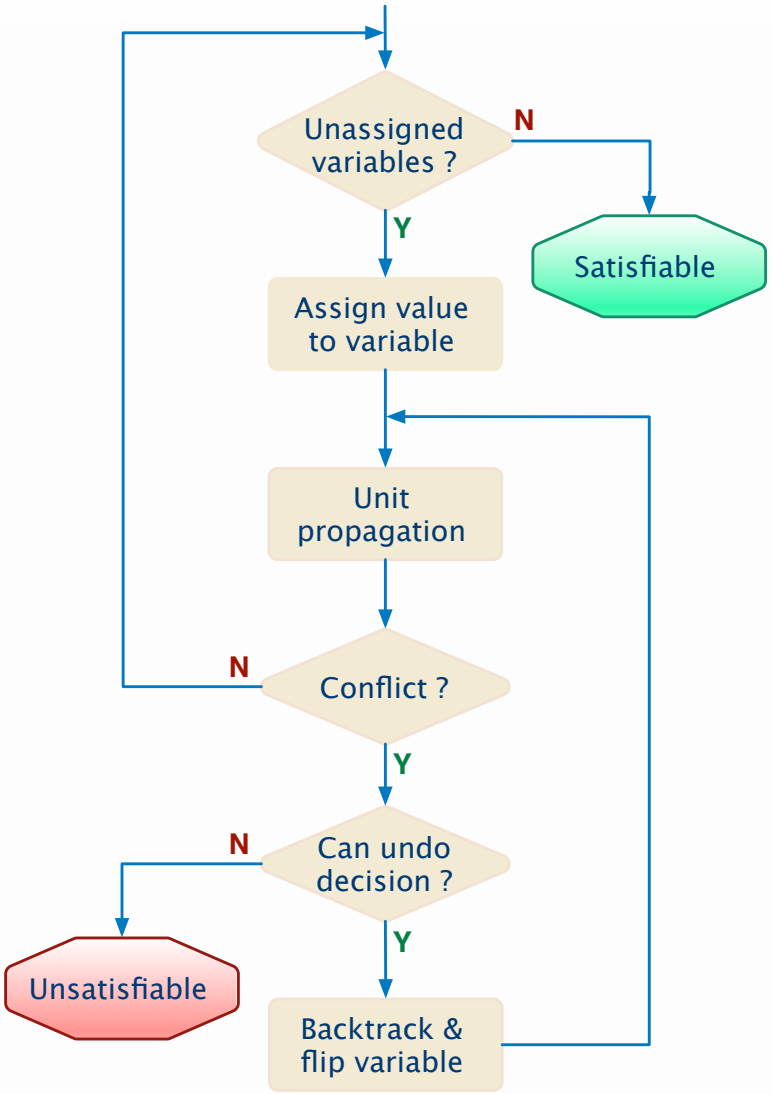


$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

Level	Dec.	Unit Prop.
0	$\emptyset$	
1	x	
2	y	
3	a	$a \rightarrow b \rightarrow \bot$

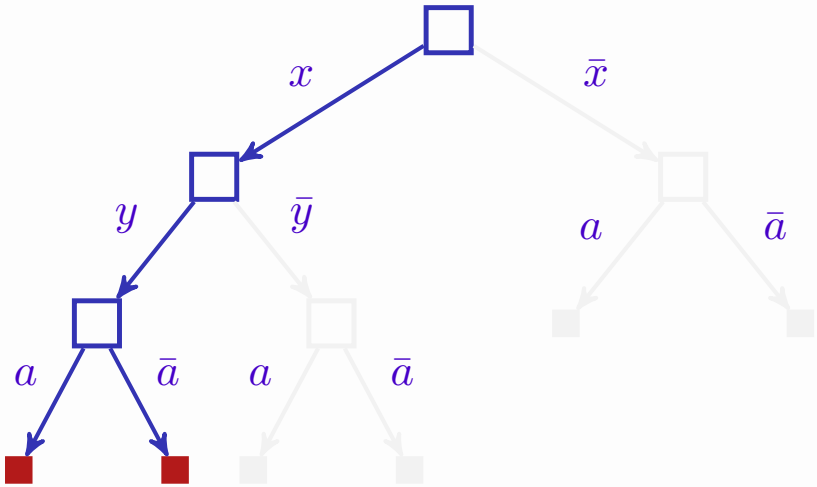


The DPLL Algorithm

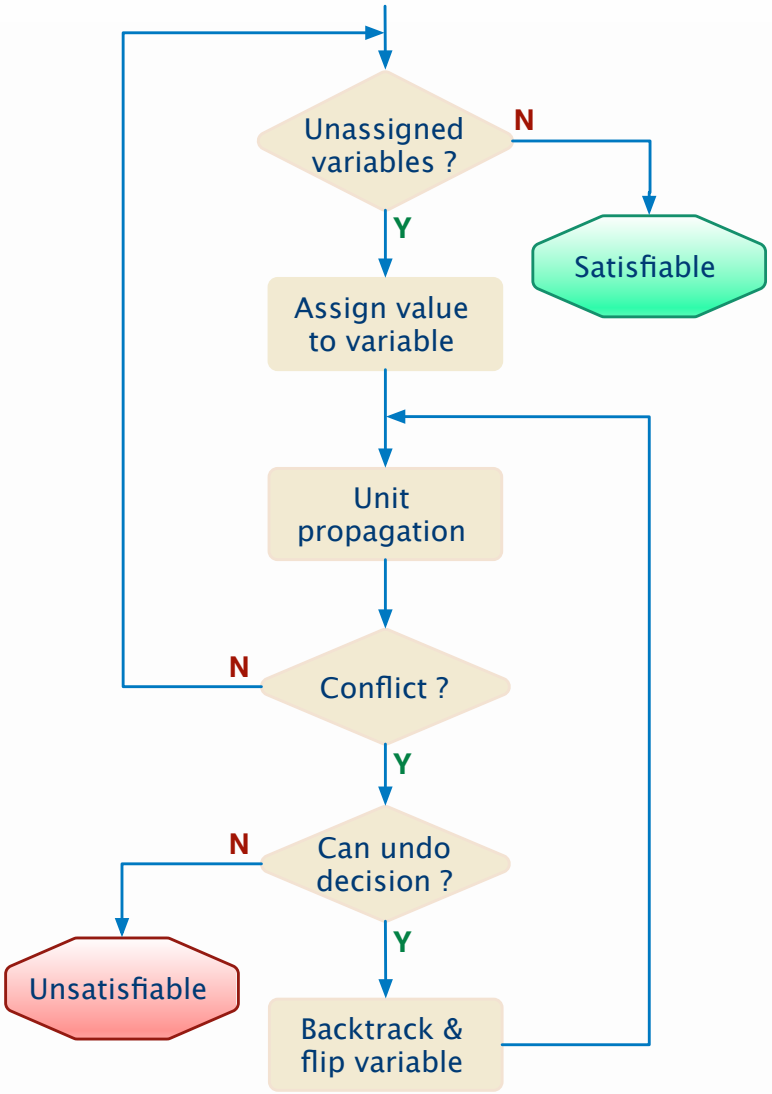


$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

Level	Dec.	Unit Prop.
0	$\emptyset$	
1	x	
2	y	
3	$\bar{a}$	$\rightarrow \bar{b} \rightarrow \bot$

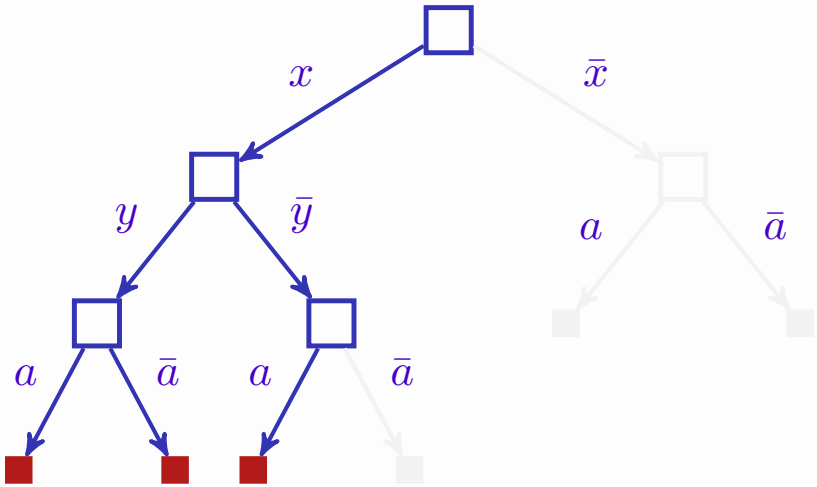


The DPLL Algorithm

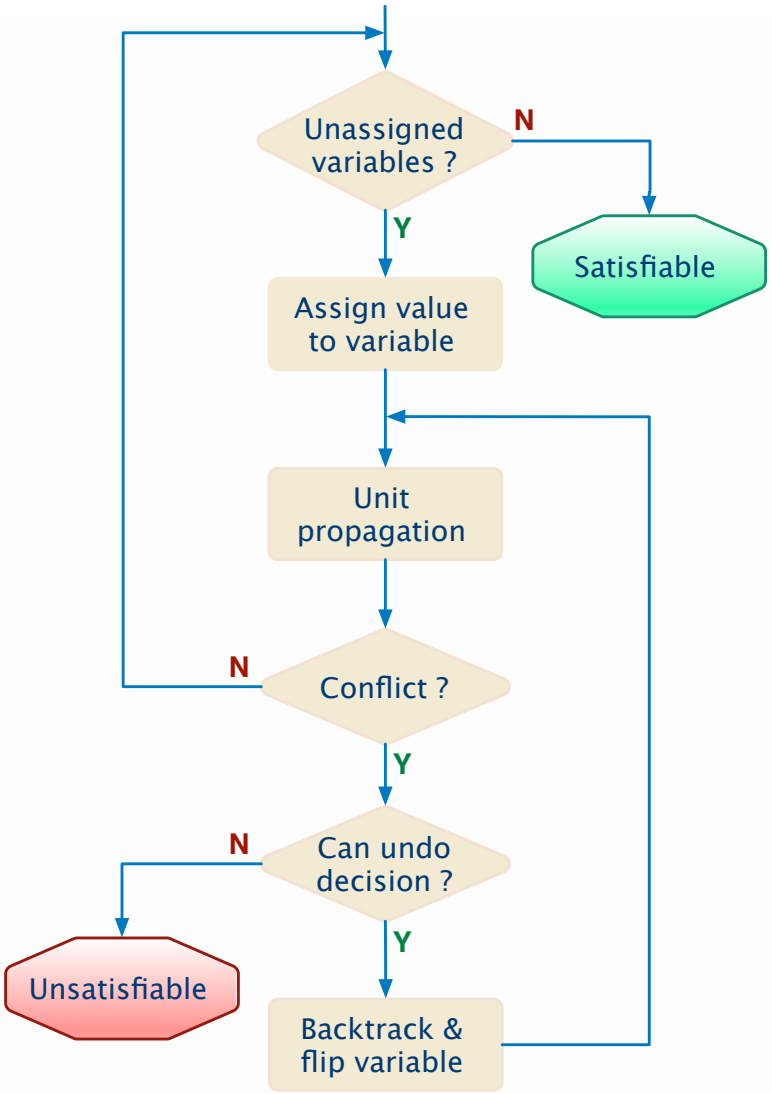


$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

Level	Dec.	Unit Prop.
0	$\emptyset$	
1	x	
2	$\bar{y}$	
3	a	$a \rightarrow b \rightarrow \bot$

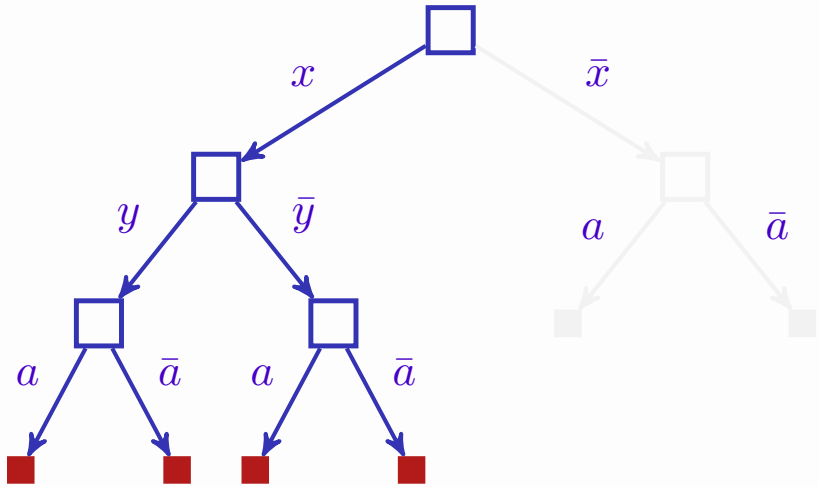


The DPLL Algorithm

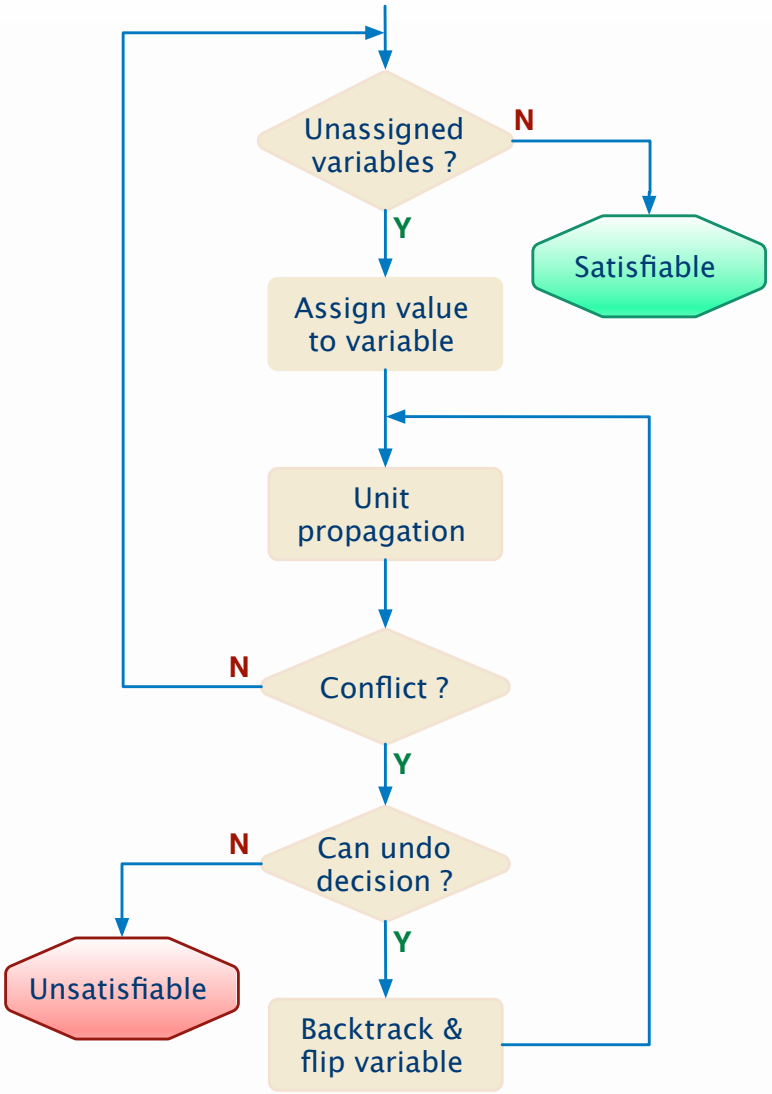


$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

Level	Dec.	Unit Prop.
0	$\emptyset$	
1	x	
2	$\bar{y}$	
3	$\bar{a}$	$\rightarrow \bar{b} \rightarrow \bot$

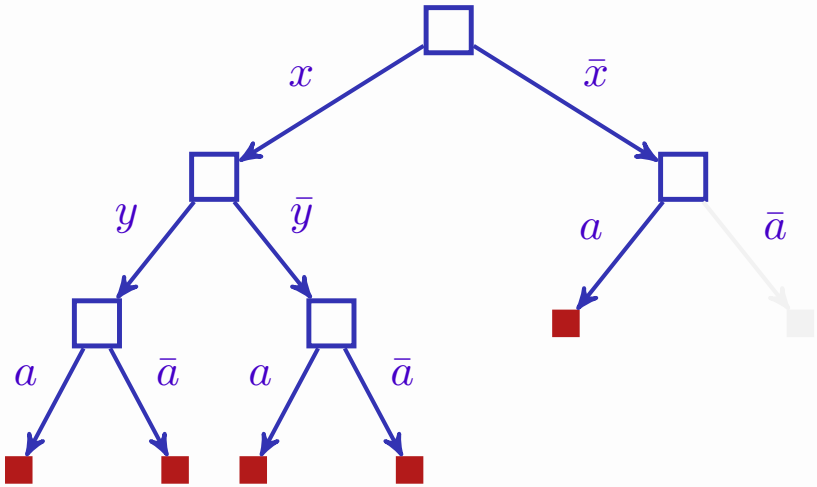


The DPLL Algorithm

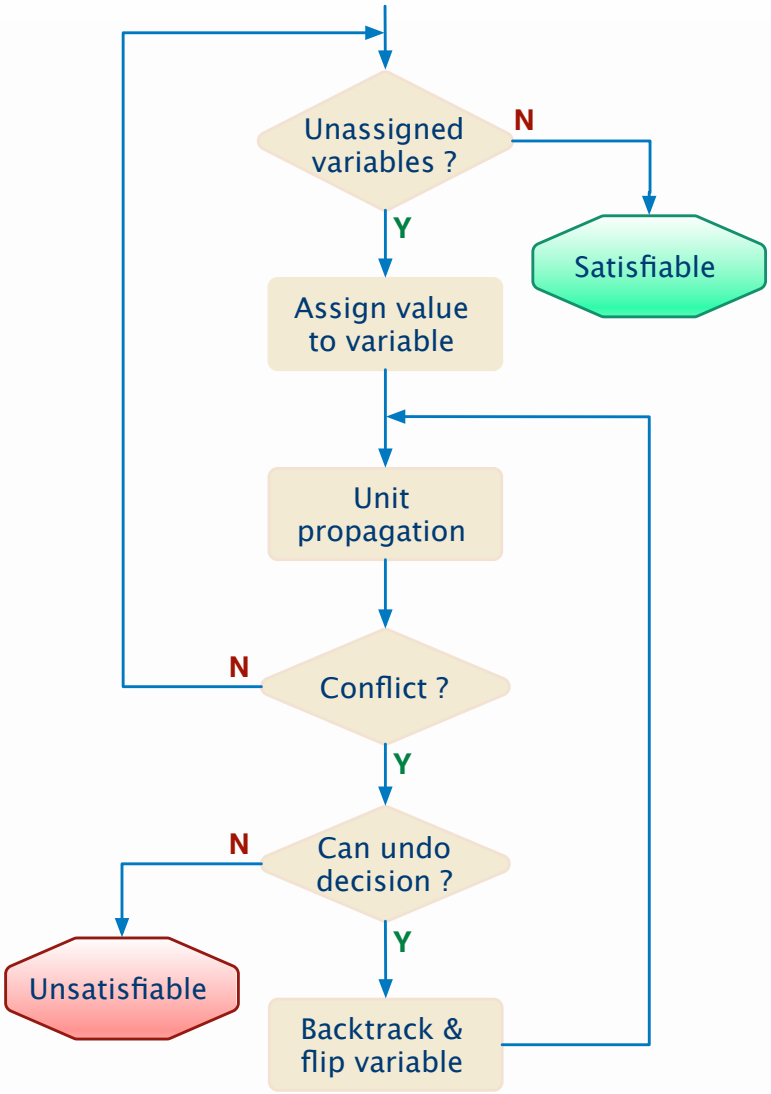


$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

Level	Dec.	Unit Prop.
0	$\emptyset$	
1	$\bar{x}$	$\rightarrow y$
2	a	$\rightarrow b \rightarrow \bot$

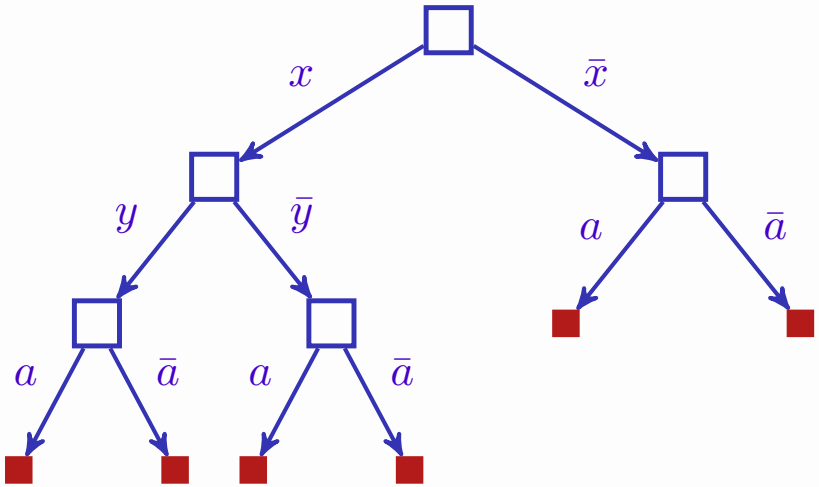


The DPLL Algorithm



$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

Level	Dec.	Unit Prop.
0	$\emptyset$	
1	$\bar{x}$	$\rightarrow y$
2	$\bar{a}$	$\rightarrow \bar{b} \rightarrow \bot$

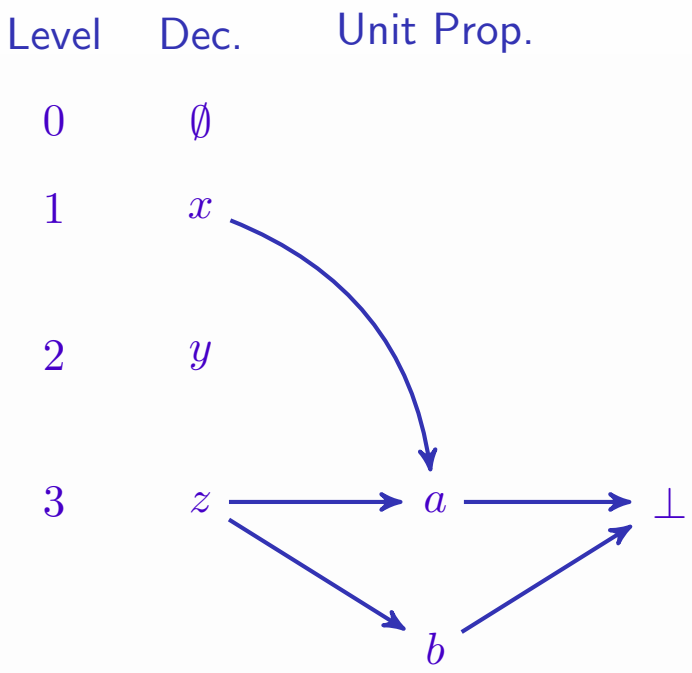


Outline

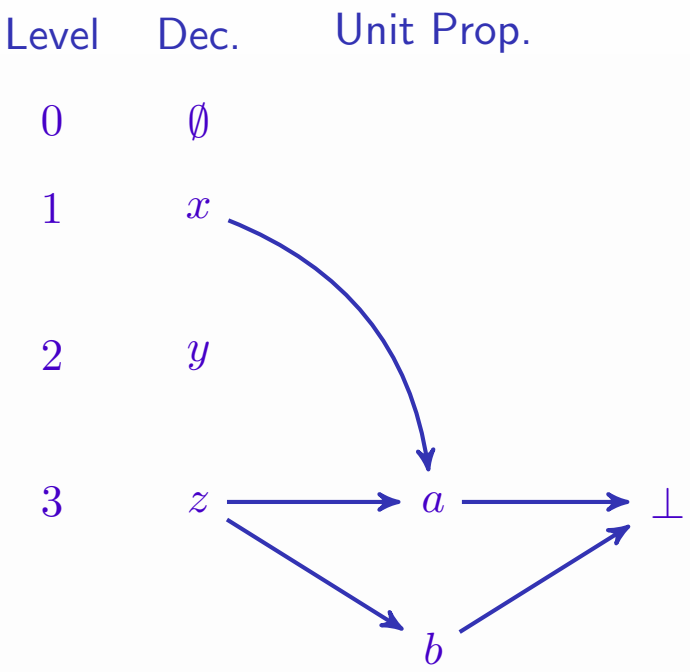
What is a CDCL SAT Solver?

- Extend DPLL SAT solver with: [DP60,DLL62]
  - Clause learning & non-chronological backtracking [MSS96,BS97,Z97]
    - Exploit UIPs [MSS96,SSS12]
    - Minimize learned clauses [SB09,VG09]
    - Opportunistically delete clauses [MSS96,MSS99,GN02]
  - Search restarts [GSK98,BMS00,H07,B08]
  - Lazy data structures
    - Watched literals [MMZZM01]
  - Conflict-guided branching
    - Lightweight branching heuristics [MMZZM01]
    - Phase saving [PD07]
  - ...

Conflict-Driven Clause Learning

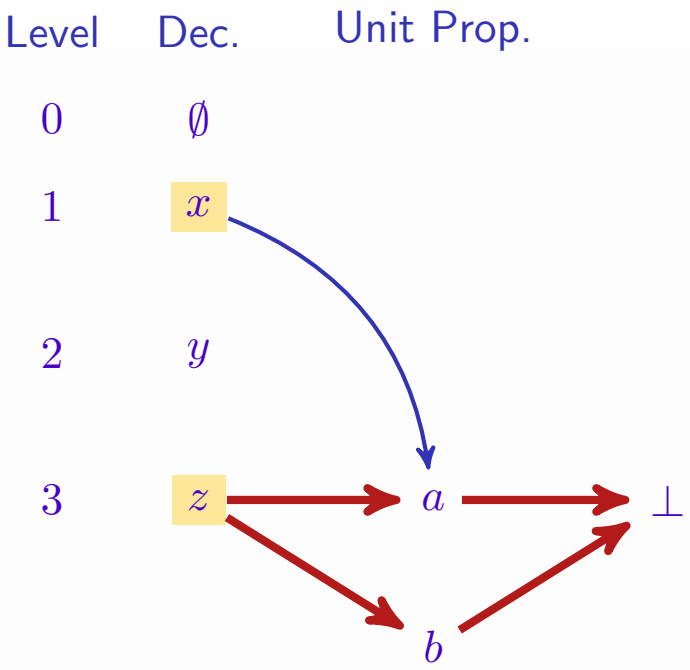


Conflict-Driven Clause Learning



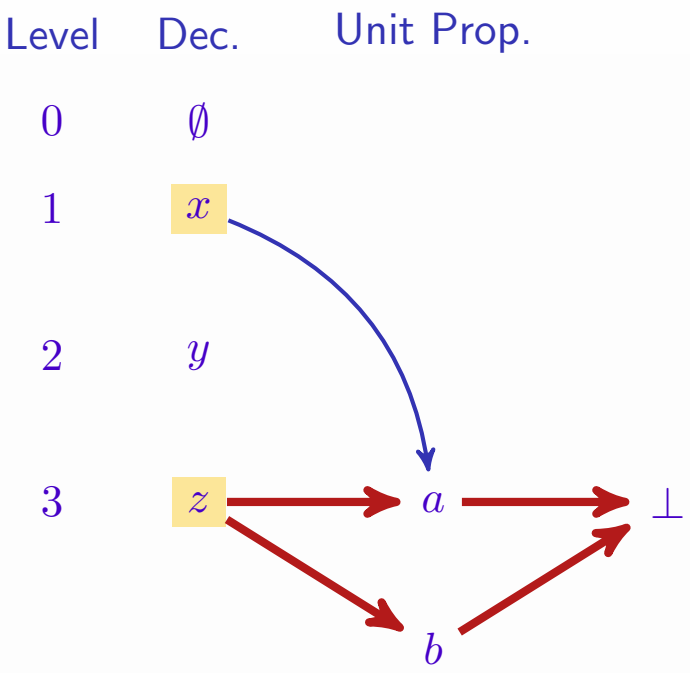
- Analyze conflict

Conflict-Driven Clause Learning



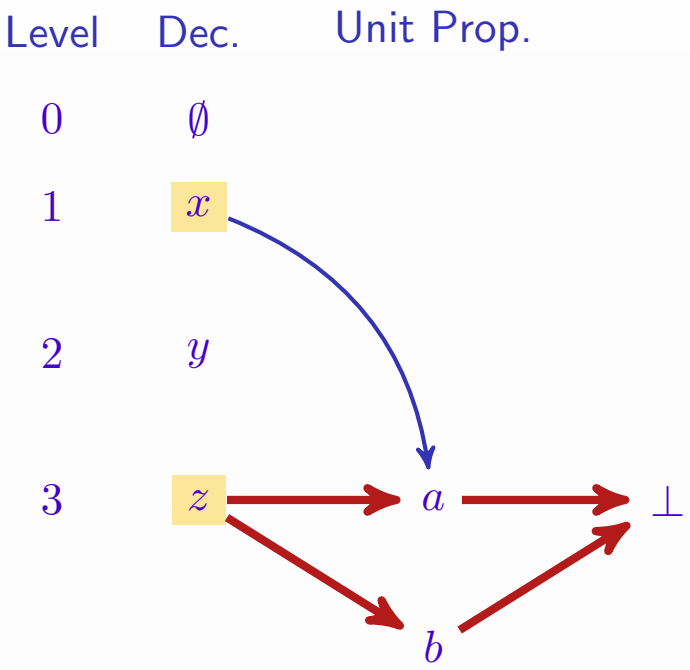
- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels

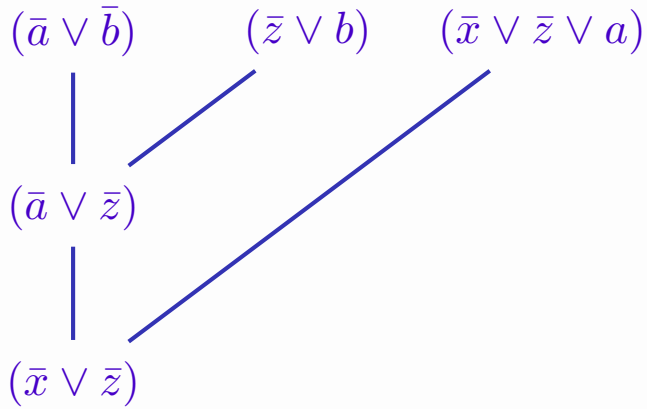
Conflict-Driven Clause Learning



- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels
  - Create **new** clause: $(\bar{x} \vee \bar{z})$

Conflict-Driven Clause Learning





- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution
  - Learned clauses result from (**selected**) resolution operations

Practicals: Exercise

Encode into CNF the k-graph coloring problem

- k is a parameter of the encoding
- use provided colab as a starting point

Practicals: Exercise

Encode into CNF the k-graph coloring problem

- k is a parameter of the encoding
- use provided colab as a starting point

Erdős–Rényi graph generator

- given number of vertices N and a parameter $p \in (0, 1)$
- each edge is present with probability p

Practicals: Exercise

Encode into CNF the k-graph coloring problem

- k is a parameter of the encoding
- use provided colab as a starting point

Erdős–Rényi graph generator

- given number of vertices N and a parameter $p \in (0, 1)$
- each edge is present with probability p

Advanced: encode the K-clique problem

- i.e., does the input graph has a clique of size K?
- hint: look at `pysat`'s `CardEnc.atmost(lits=[...], bound=k, encoding=...)`