# C05 – SAT solvers

Program Verification

FMI · Denisa Diaconescu · Spring 2025

# Propositional logic

# Propositional Logic

Formulas are defined by

$$\varphi \ ::= \ p \mid \top \mid \bot \mid \neg\varphi \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid \varphi \to \varphi$$

starting from propositional variables (atoms).

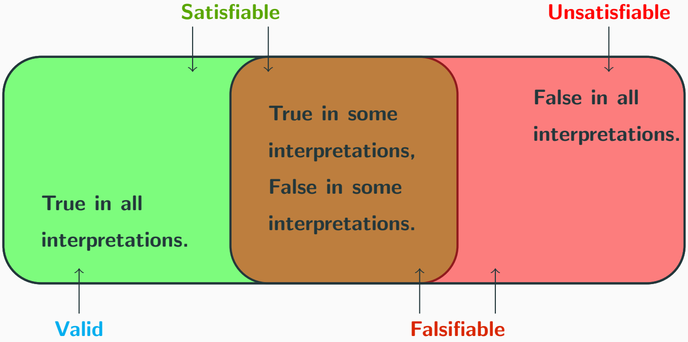
Interpretations assign truth values to propositional variables (true/false).

Further, we can compute the truth value of any formula (e.g., using truth tables).

A formula is satisfiable if there exists an interpretation which makes the formula true.

A formula is valid if is it true under all interpretations.

# Satisfiability and Validity of formulas




$$
\begin{aligned}
\text{valid} &= \text{not falsifiable} \\
\text{satisfiable} &= \text{not unsatisfiable}
\end{aligned}
$$

$$\boxed{\varphi \text{ satisfiable} \quad \text{iff} \quad \neg\varphi \text{ is falsifiable} \quad \text{iff} \quad \neg\varphi \text{ is not valid}}$$

$$\boxed{\varphi \text{ valid} \quad \text{iff} \quad \neg\varphi \text{ is unsatisfiable}}$$

# The SAT problem

The SAT problem:

*Given a propositional formula with $n$ variables,*
*can we find an interpretation to make the formula true?*

Is the formula satisfiable? If so, how?

The first known NP-complete problem, as proved by Stephen Cook in 1971.

Since SAT is NP-complete, is there hope?

# SAT solvers

A SAT solver is a program that automatically decides whether a propositional formula is satisfiable (i.e, answers the SAT problem).

If it is satisfiable, a SAT solver will produce an example of an interpretation that satisfies the formula.

Naive algorithm: enumerate all assignments to the $n$ variables in the formula ($2^n$ assignments!)

Worst case complexity is exponential (for all known algorithms).

Perhaps surprisingly, many efficient SAT solvers exist!

- average cases encountered in practice can be handled (much) faster
- real problem instances will not be random: exploit implicit structure
- some variables will be tightly correlated with other variables
- some variables will be irrelevant for the difficult parts of the search

# Applications

Where can we find SAT technology today?

- Formal methods
  - Hardware model checking
  - Software model checking
  - Termination analysis of term-rewrite systems
  - Test pattern generation (testing of software & hardware)
  - ...
- Artificial intelligence
  - Planning
  - Knowledge representation
  - Games (n-queens, sudoku, ...)

# Applications

Where can we find SAT technology today?

- Bioinformatics
  - Haplotype inference
  - Pedigree checking
  - ...
- Design automation
  - Equivalence checking
  - Fault diagnosis
  - Noise analysis
  - ...
- Security
  - Cryptanalysis
  - Inversion attacks on hash functions
  - ...

## Applications

Where can we find SAT technology today?

- Computationally hard problems
  - Graph coloring
  - Traveling salesperson
  - ...
- Mathematical problems
  - van der Waerden numbers
  - Quasigroup open problems
  - ...
- Core engine for other solvers
- Integrated into theorem provers
  - HOL
  - Isabelle
  - ...

# An example - Pythagorean Triples

Is it possible to assign to each integer $1, 2, \ldots, n$ one of two colors such that if $a^2 + b^2 = c^2$ then $a, b$, and $c$ do not all have the same color?

- Solution: nope
- for $n = 7825$ it is not possible
- the proof obtained by a SAT solver has 200 Terrabytes
- the largest Math proof ever *(see the article)*

How to encode this problem?

- for each integer $i$ we have a Boolean variable $x_i$
- $x_i = 1$ if the color of $i$ is 1 and $x_i = 0$ otherwise
- for each $a, b, c$ such that $a^2 + b^2 = c^2$ we have two clauses:

$$
\begin{aligned}
(x_a \vee x_b \vee x_c) \quad &\sim \quad \neg(\neg x_a \wedge \neg x_b \wedge \neg x_c) \\
(\neg x_a \vee \neg x_b \vee \neg x_c) \quad &\sim \quad \neg(x_a \wedge x_b \wedge x_c)
\end{aligned}
$$

# CNF - Conjunctive normal form

# CNF - Conjunctive normal form

All current fast SAT solvers work on CNF.

- A literal is a propositional variable or its negation
  - example: $p$, $\neg q$
  - For a literal $l$ we write $\sim l$ for the negation of $l$ cancelling double negations
- A clause is a disjunction of literals
  - example: $p \vee \neg q \vee r$
  - Since $\vee$ is associative, we can represent clauses as lists of literals.
  - The empty clause (0 disjuncts) is defined to be $\bot$
  - A unit clause is a clause consisting of exactly one literal.
- A formula is in CNF if it is a conjuction of clauses
  - example: $(p \vee \neg q \vee r) \wedge (\neg p \vee s \vee t \vee \neg u)$
  - Since $\wedge$ is associative, we can represent formulas in CNF as lists of clauses.
  - The empty conjunction is defined to be $\top$

# Conversion to CNF

Any propositional formula can be transformed into an equivalent formula in CNF (need not be unique!).

Two formulas are equivalent if they are satisfied by the same interpretations.

**Example**
The formula $p$ is equivalent with the following formulas in CNF:

- $p$
- $p \wedge (p \vee q)$

## Conversion to CNF

We can rewrite the formula directly via the following equivalences:

- Remove implications: rewrite $A \to B$ to $\neg A \vee B$
- Push all negations inwards:
  - rewrite $\neg(A \vee B)$ to $\neg A \wedge \neg B$
  - rewrite $\neg(A \wedge B)$ to $\neg A \vee \neg B$
- Remove double negations: rewrite $\neg\neg A$ to $A$
- Eliminate $\top$ and $\bot$:
  - rewrite $A \vee \bot$ to $A$
  - remove clauses containing $\top$
- Distribute disjunctions over conjunctions: rewrite $A \vee (B \wedge C)$ to $(A \vee B) \wedge (A \vee C)$

# Conversion to CNF

**Example**

Applying the above rules to the formula

$$(\neg p \wedge q) \rightarrow (p \wedge (r \rightarrow q))$$

we obtain the equivalent formula in CNF:

$$(p \vee \neg q \vee p) \wedge (p \vee \neg q \vee \neg r \vee q).$$

We can further simplify:

- Remove duplicate clauses, duplicate literals from clauses
- Remove clauses in which a literal is both positive and negative
- In fact, each variable need only occur in each clause at most once!

**Example**
If we simply the CNF formula from the above example we get

$$(p \vee \neg q).$$

# CNF and Validity

**Theorem**

*A clause $L_1 \vee \ldots \vee L_m$ is valid iff there are $1 \leq i, j \leq m$ such that $L_i$ is $\neg L_j$.*

Checking validity for formulas in CNF is very easy! For each clause of the formula, check if it contains a literal and its negation.

**Example**

- $(\neg q \vee p \vee r) \wedge (\neg p \vee r) \wedge q$ is not valid
- $(\neg q \vee p \vee q) \wedge (\neg p \vee p)$ is valid

Satisfiability is not so easy!

$$\boxed{\varphi \text{ satisfiable} \quad \text{iff} \quad \neg\varphi \text{ is not valid}}$$

# Conversion to CNF

- The previous method to transform a formula into an equivalent one in CNF can blow-up exponentially!
- There exist transformations into CNF that avoid an exponential increase in size by preserving satisfiability rather than equivalence.
- Two formulas are equisatisfiable if either both formulas are satisfiable or both are not
  - Equisatisfiable formulas may disagree for a particular choice of variables.
- These transformations are guaranteed to only linearly increase the size of the formula, but introduce new variables (e.g., Tseitin transformation).

# Tseitin transformation - an example

Let us consider the formula $\varphi = ((p \vee q) \wedge r) \to (\neg s)$.

Consider all its subformulas (excluding propositional variables):

- $\neg s$
- $p \vee q$
- $(p \vee q) \wedge r$
- $((p \vee q) \wedge r) \to (\neg s)$

Introduce a new variable for each subformula:

- $x_1 \leftrightarrow \neg s$
- $x_2 \leftrightarrow p \vee q$
- $x_3 \leftrightarrow x_2 \wedge r$
- $x_4 \leftrightarrow (x_3 \to x_1)$

# Tseitin transformation - an example

Conjunct all the equivalences and $\varphi$:

$$T(\varphi) = x_4 \land (x_4 \leftrightarrow x_3 \rightarrow x_1) \land (x_3 \leftrightarrow x_2 \land r) \land (x_2 \leftrightarrow p \lor q) \land (x_1 \leftrightarrow \neg s).$$

All equivalences are now transformed into CNF, eg.:

$$\begin{aligned} x_2 \leftrightarrow p \lor q \quad & \equiv \quad (x_2 \rightarrow p \lor q) \land (p \lor q \rightarrow x_2) \\ & \equiv \quad (\neg x_2 \lor p \lor q) \land (\neg(p \lor q) \lor x_2) \\ & \equiv \quad (\neg x_2 \lor p \lor q) \land ((\neg p \land \neg q) \lor x_2) \\ & \equiv \quad (\neg x_2 \lor p \lor q) \land (\neg p \lor x_2) \land (\neg q \lor x_2) \end{aligned}$$

# SAT solvers

# SAT solvers

- There are plenty SAT solvers
  - Glucose
  - MiniSAT, PicoSAT
  - RelSAT
  - GRASP
  - ...
- In order to solve the problems, you can use any SAT solver you prefer
- There are also online SAT solvers:
  - Logictools
  - ...

# DIMACS Format

- the most common input format for SAT solvers
- a way to encode CNF formulas

**Example**
The input

```
c This is a comment
c This is another comment
p cnf 6 3
1 -2 3 0
2 4 5 0
4 6 0
```

represents the CNF formula $(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_4 \vee x_5) \wedge (x_4 \vee x_6)$

# DIMACS Format

- At the beginning there can exist one or more comment line.
- Comment lines start with a c
- The following lines are information about the expression itself
- the Problem line starts with a p:

  `p FORMAT VARIABLES CLAUSES`

- `FORMAT` should be cnf
- `VARIABLES` is the number of variables in the expression
- `CLAUSES` is the number of clauses in the expression

**Example**
`p cnf 6 3` expresses that there are 6 variables and 3 clauses

# DIMACS Format

- The next `CLAUSES` lines are for the clauses themselves
- Variables are enumerated from 1 to `VARIABLES`
- A negation is represented by $-$
- Each variable information is separated by a blank space
- A 0 is added at the end to mark the end of the clause

**Example**

`1 -2 3 0` expresses the clause $(x_1 \vee \neg x_2 \vee x_3)$

# Problem 1 - A planning problem encoded in SAT

Problem.

Scheduling a meeting considering the following constraints:

- Adam can only meet on Monday and Wednesday
- Bridget cannot meet on Wednesday
- Charles cannot meet on Friday
- Darren can only meet on Thursday or Friday

# Problem 1 - A planning problem encoded in SAT

Solution.

- We represent week day *Monday*, *Tuesday*, ... as variables $x_1, x_2, \ldots$.
- We obtain the following formula in CNF:

$$
\begin{aligned}
\varphi = {} & (x_1 \vee x_3) \wedge (\neg x_3) \wedge (\neg x_5) \wedge (x_4 \vee x_5) \wedge \\
& (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_5) \wedge \\
& (\neg x_2 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_5) \wedge \\
& (\neg x_3 \vee \neg x_4) \wedge (\neg x_3 \vee \neg x_5) \wedge \\
& (\neg x_4 \vee \neg x_5)
\end{aligned}
$$

# Problem 2 - Graph Colouring encoded in SAT

Problem.

Given an undirected graph $G = (V, E)$, a graph colouring assigns a colour to each node such that all adjacent nodes have a different colour.

A graph colouring using at most $k$ colours is called a $k$-colouring.

The Graph Colouring Problem asks whether a $k$-colouring for $G$ exists.

# Problem 2 - Graph Colouring encoded in SAT

Problem.

Given an undirected graph $G = (V, E)$, a graph colouring assigns a colour to each node such that all adjacent nodes have a different colour.

A graph colouring using at most $k$ colours is called a $k$-colouring.

The Graph Colouring Problem asks whether a $k$-colouring for $G$ exists.

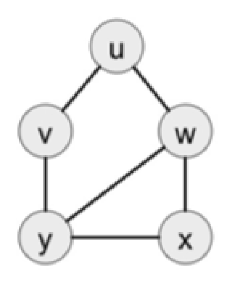
Solution

- SAT encoding: use $k \cdot |V|$ Boolean variables
- For every $v \in V$ and $1 \leq j \leq k$, variable $v_j$ is true if node $v$ gets colour $j$
- Clauses?
  - Every node gets a colour: $(v_1 \vee \ldots \vee v_k)$ for $v \in V$
  - Adjacent nodes have diff. colours: $\neg u_j \vee \neg v_j$ for $u, v \in V$, $u \neq v$, $u, v$ adjacent, $1 \leq j \leq k$
  - What about multiple colours for a node? At-most-one constraints

# Problem 2 - Graph Colouring encoded in SAT

Todo. Encode the following graph colouring problem into SAT and use a SAT solver to find a solution.

- $V = \{u, v, w, x, y\}$
- Colours: red (=1), green (=2), blue (=3)



# Problem 3 - Sudoku encoded in SAT

Problem. Represent a Sudoku puzzle as a SAT problem.

Solution.

- The grid for the puzzle is $9 \times 9$.
- Encoding Sudoku puzzles into CNF requires $9 \cdot 9 \cdot 9 = 729$ propositional variables.
- For each entry in the $9 \times 9$ grid $\mathcal{S}$, we associate 9 variables.
- Let us use the denotation $s_{xyz}$ to refer to variables.
- Variable $s_{xyz}$ is assigned true iff the entry in row x and column y is assigned number $z$.
- For example, $s_{483} = 1$ means that $\mathcal{S}[4, 8] = 3$.
- Naturally, the pre-assigned entries of the Sudoku grid will be represented as unit clauses.

## Problem 3 - Sudoku encoded in SAT

The add the following constraints:

- There is at least one number in each entry:

$$\bigwedge_{x=1}^{9} \bigwedge_{y=1}^{9} \bigvee_{z=1}^{9} s_{xyz}$$

- Each number appears at most once in each row:

$$\bigwedge_{y=1}^{9} \bigwedge_{z=1}^{9} \bigwedge_{x=1}^{8} \bigwedge_{i=x+1}^{9} (\neg s_{xyz} \vee \neg s_{iyz})$$

- Each number appears at most once in each columns:

$$\bigwedge_{x=1}^{9} \bigwedge_{z=1}^{9} \bigwedge_{y=1}^{8} \bigwedge_{i=y+1}^{9} (\neg s_{xyz} \vee \neg s_{xiz})$$

# Problem 3 - Sudoku encoded in SAT

The add the following constraints:

- Each number appears at most once in each $3 \times 3$ sub-grid:

$$\bigwedge_{z=1}^{9} \bigwedge_{i=0}^{2} \bigwedge_{j=0}^{2} \bigwedge_{x=1}^{3} \bigwedge_{y=1}^{3} \bigwedge_{k=y+1}^{3} (\neg s_{(3i+x)(3j+y)z} \vee \neg s_{(3i+x)(3j+k)z})$$

$$\bigwedge_{z=1}^{9} \bigwedge_{i=0}^{2} \bigwedge_{j=0}^{2} \bigwedge_{x=1}^{3} \bigwedge_{y=1}^{3} \bigwedge_{k=x+1}^{3} \bigwedge_{l=1}^{3} (\neg s_{(3i+x)(3j+y)z} \vee \neg s_{(3i+k)(3j+l)z})$$

The encoding is from the paper

I. Lynce, J. Ouaknine, *Sudoku as a SAT Problem* (link)

# SAT solvers algorithms

# Davis-Putnam algorithm

- First attempt at a better-than-brute-force SAT algorithm (1960)
  - Original algorithm tackles first-order logic
  - We present the propositional case
- We assume as input a formula $A$ in CNF
  - a set of clauses
  - a set of sets of literals
- The DP algorithm rewrites the set of clauses until
  - $A$ is $\top$ (the set is empty) then returns sat, or
  - $A$ contains an empty clause $\perp$ return unsat

# Resolution rule

$$\boxed{\frac{p \vee \alpha \quad \neg p \vee \beta}{\alpha \vee \beta} \qquad \textit{resolution}}$$

$$\boxed{\frac{p \vee p \vee \alpha}{p \vee \alpha} \qquad \textit{merging}}$$

**Example**

$$\frac{x_1 \vee x_2 \vee x_3 \quad x_1 \vee \neg x_2 \vee x_4}{x_1 \vee x_1 \vee x_3 \vee x_4} \qquad \frac{x_1 \vee x_1 \vee x_3 \vee x_4}{x_1 \vee x_3 \vee x_4}$$

# Resolution rule

If a variable $p$ occurs both positively and negatively in clauses of $A$:

- Let $C_{pos} = \{A_1 \vee p, A_2 \vee p, \ldots\}$ be the clauses in $A$ in which $p$ occurs positively
- Let $C_{neg} = \{B_1 \vee \neg p, B_2 \vee \neg p, \ldots\}$ be the clauses in $A$ in which $p$ occurs negatively
- Remove these two sets of clauses from $A$, and replace them with the new set

$$\{A_i \vee B_j \mid A_i \vee p \in C_{pos}, B_j \vee \neg p \in C_{neg}\}$$

# Davis-Putnam algorithm

- Iterately apply the following steps:
  - Select variable $x$
  - Apply resolution rule between every pair of clauses of the form $(x \vee \alpha)$ and $(\neg x \vee \beta)$
  - Remove all clauses containing either $x$ or $\neg x$
- Terminate if
  - The empty formula is derived ($\top$) and then return sat, or
  - An empty clause is derived ($\bot$) and then return unsat

# Davis-Putnam algorithm

**Example**

1. $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4)$
2. $(\neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4)$
3. $(\neg x_3 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4)$
4. $(\neg x_3 \vee x_3) \wedge (x_3)$
5. $\top$

Formula is SAT

# Davis-Putnam algorithm

**Example**

1. $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_2) \wedge (\neg x_2 \vee x_3)$
2. $(\neg x_2 \vee \neg x_3) \wedge (x_2) \wedge (\neg x_2 \vee x_3)$
3. $(\neg x_3) \wedge (x_3)$
4. $\perp$

Formula is UNSAT

# Davis-Putnam algorithm

Main issues of the approach:

- In which order should the resolution steps be performed?
- In which order the variables should be selected? (variable elimination)
- Worst-case exponential in memory consumption!

# Faster algorithms

- Davis-Putnam algorithm: the refinements
  - Add specific cases to order variable elimination steps.
  - The pure literal rule and the unit propagation rule
- Davis-Putnam-Logemann-Loveland algorithm
  - Standard backtrack search
  - space efficient DP
- Conflict-Driven Clause Learning algorithm
  - An extension of DPLL with:
    - Clause learning
    - Non-chronological backtracking
  - Clause learning can be performed with various strategies
  - CDCL algorithms are use in almost all modern SAT solvers

## Davis-Putnam algorithm: the refinements

Add specific cases to order variable elimination steps.

- Iteratively apply the following steps:
  - Apply the pure literal rule and unit propagation
  - Select variable $x$
  - Apply resolution rule between every pair of clauses of the form $(x \vee \alpha)$ and $(\neg x \vee \beta)$
  - Remove all clauses containing either $x$ or $\neg x$
- Terminate if
  - The empty formula is derived ($\top$) and then return sat, or
  - An empty clause is derived ($\bot$) and then return unsat

# Pure literal rule

If a variable $p$ occurs either only positively or only negatively in $A$,
delete all clauses of $A$ in which $p$ occurs.

- A literal is pure if occurs only positively or negatively in a CNF formula
- Pure literal rule: eliminate first pure literals since no resolvant are produced!
- Applying a variable elimination step on a pure literal strictly reduced the number of clauses!
- Preserves satisfiability, not logical equivalence!

**Example**
In $(\neg x_1 \vee x_2) \wedge (x_3 \vee \neg x_2) \wedge (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$ the pure literals are $\neg x_1$ and $x_3$.

# Unit propagation rule

If $l$ is a unit clause in $A$, then update $A$ by:

- removing all clauses which have $l$ as a disjunct, and
- updating all clauses in $A$ containing $\sim l$ as a disjunct by removing that disjunct

- Specific case of resolution
- Only shorten clauses!
- a.k.a. Boolean constraint propagation or BCP
- Is arguably the key component to fast SAT solving
- Since clauses are shortened, new unit clauses may appear.
  Empty clauses also!
- Apply unit propagation while new unit clauses are produced.
- Preserves logical equivalence!

# DP algorithm

**Example**

1. $p \wedge (\neg p \vee q) \wedge (\neg q \vee r) \wedge (\neg r \vee s \vee t)$

   We apply the Pure literal rule for $s$ and $t$ and we delete the last clause.

2. $p \wedge (\neg p \vee q) \wedge (\neg q \vee r)$

   We apply the Unit propagation rule for $p$.

3. $q \wedge (\neg q \vee r)$

   We apply the Unit propagation rule for $q$.

4. $r$

   We apply the Unit propagation rule for $r$.

5. $\top$

The formula is SAT

# DP: The limits

- The approach runs easily out of memory
- The solution: using backtrack search!

## Quiz time!

https://tinyurl.com/FMI-PV2023-Quiz6

# Davis-Putnam-Logemann-Loveland algorithm

# Preliminary definitions

- Propositional variable can be assigned value False or True.
    - In some contexts variables may be unassigned
- A clause is satisfied is at least one of its literals is assigned value true
    - $(x_1 \vee \neg x_2 \vee \neg x_3)$
- A clause is unsatisfied if all of its literals are assigned value false
    - $(x_1 \vee \neg x_2 \vee \neg x_3)$
- A clause is unit if it contains one single unassigned literal and all other literals are assigned value false
    - $(x_1 \vee \neg x_2 \vee \neg x_3)$
- A formula is satisfied if all its clauses are satisfied
- A formula is unsatisfied if at least one of its clauses is unsatisfied

# DPLL

**DPLL(F, $\mathcal{I}$):**

- 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, $\mathcal{I} \cup$ x) = SAT return SAT
  - return DPLL(F, $\mathcal{I} \cup$ x)

**Notes:**

**x** We use red to denote that a variable / literal is false

**x** We use green to denote that a variable / literal is true

**Conflict** all disjuncts of a clause are assigned false.

# Pure literals in backtrack search

As before.

**Example**
$(\neg x_1 \vee x_2) \wedge (x_3 \vee \neg x_2) \wedge (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$

becomes

$(x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$

# Unit propagation in backtrack search

Unit clause rule in backtrack search:

> Given a unit clause, its only unassigned literal must be assigned value true for the clause to be satisfied.

**Example**
For unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, the variable $x_3$ must be assigned value false.

Unit propogation rule: Iterated application of the unit clause rule.

# Unit propagation in backtrack search

**Example (1)**

- $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$
- $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$
- $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$

# Unit propagation in backtrack search

**Example (2)**

- $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4)$
- $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4)$
- $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4)$

Conflict!

Unit propagation can satisfy clauses but can also unsatisfy clauses (i.e. conflicts)

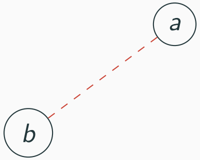
# An example of DPLL

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

# An example of DPLL

Select decision variable: $a$

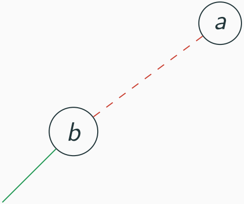
$(a \vee \neg b \vee d) \wedge$
$(a \vee \neg b \vee e) \wedge$
$(\neg b \vee \neg d \vee \neg e) \wedge$
$(a \vee b \vee c \vee d) \wedge$
$(a \vee b \vee c \vee \neg d) \wedge$
$(a \vee b \vee \neg c \vee e) \wedge$
$(a \vee b \vee \neg c \vee \neg e)$



# An example of DPLL

Select decision variable: $b$

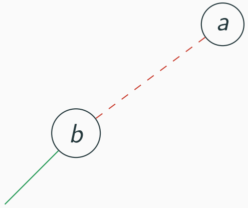
$(a \vee \neg b \vee d)\ \wedge$
$(a \vee \neg b \vee e)\ \wedge$
$(\neg b \vee \neg d \vee \neg e)\ \wedge$
$(a \vee b \vee c \vee d)\ \wedge$
$(a \vee b \vee c \vee \neg d)\ \wedge$
$(a \vee b \vee \neg c \vee e)\ \wedge$
$(a \vee b \vee \neg c \vee \neg e)$



# An example of DPLL

Unit propagation: $d$

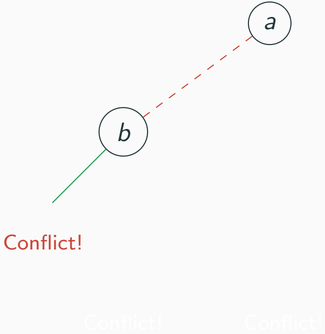
$(a \vee \neg b \vee d) \wedge$
$(a \vee \neg b \vee e) \wedge$
$(\neg b \vee \neg d \vee \neg e) \wedge$
$(a \vee b \vee c \vee d) \wedge$
$(a \vee b \vee c \vee \neg d) \wedge$
$(a \vee b \vee \neg c \vee e) \wedge$
$(a \vee b \vee \neg c \vee \neg e)$



# An example of DPLL

Unit propagation: $e$

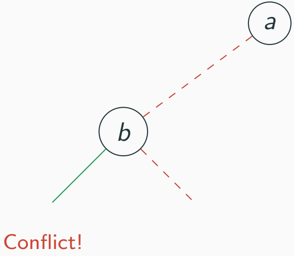
$(a \lor \neg b \lor d) \land$
$(a \lor \neg b \lor e) \land$
$(\neg b \lor \neg d \lor \neg e) \land$
$(a \lor b \lor c \lor d) \land$
$(a \lor b \lor c \lor \neg d) \land$
$(a \lor b \lor \neg c \lor e) \land$
$(a \lor b \lor \neg c \lor \neg e)$



# An example of DPLL

Select decision variable: $b$

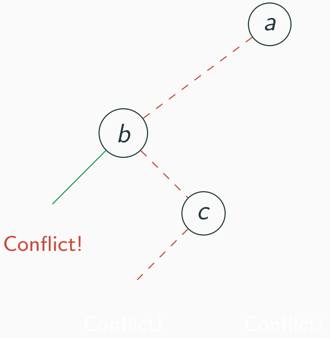
$(a \lor \neg b \lor d) \land$
$(a \lor \neg b \lor e) \land$
$(\neg b \lor \neg d \lor \neg e) \land$
$(a \lor b \lor c \lor d) \land$
$(a \lor b \lor c \lor \neg d) \land$
$(a \lor b \lor \neg c \lor e) \land$
$(a \lor b \lor \neg c \lor \neg e)$



# An example of DPLL

Select decision variable: $c$

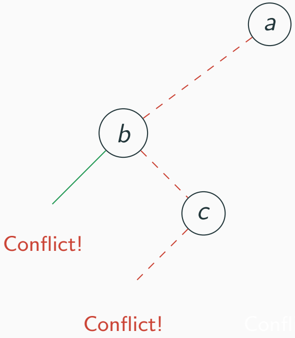
$(a \vee \neg b \vee d) \wedge$
$(a \vee \neg b \vee e) \wedge$
$(\neg b \vee \neg d \vee \neg e) \wedge$
$(a \vee b \vee c \vee d) \wedge$
$(a \vee b \vee c \vee \neg d) \wedge$
$(a \vee b \vee \neg c \vee e) \wedge$
$(a \vee b \vee \neg c \vee \neg e)$



# An example of DPLL

Unit propagation: $d$

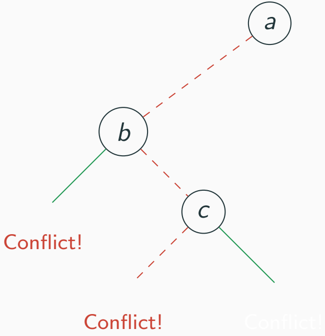
$(a \vee \neg b \vee d) \wedge$
$(a \vee \neg b \vee e) \wedge$
$(\neg b \vee \neg d \vee \neg e) \wedge$
$(a \vee b \vee c \vee d) \wedge$
$(a \vee b \vee c \vee \neg d) \wedge$
$(a \vee b \vee \neg c \vee e) \wedge$
$(a \vee b \vee \neg c \vee \neg e)$



# An example of DPLL

Select decision variable: $c$

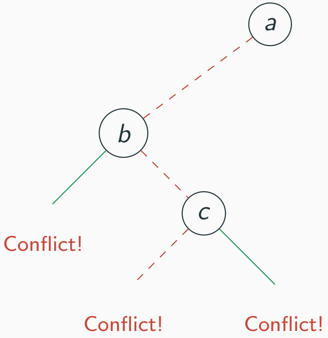
$(a \lor \neg b \lor d) \land$
$(a \lor \neg b \lor e) \land$
$(\neg b \lor \neg d \lor \neg e) \land$
$(a \lor b \lor c \lor d) \land$
$(a \lor b \lor c \lor \neg d) \land$
$(a \lor b \lor \neg c \lor e) \land$
$(a \lor b \lor \neg c \lor \neg e)$



# An example of DPLL

Unit propagation: $e$

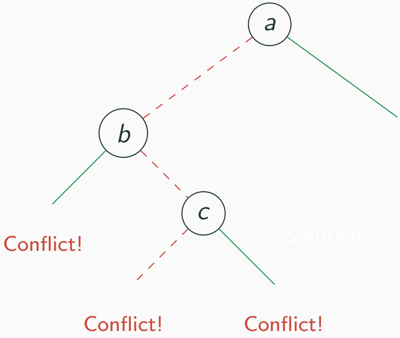
$(a \vee \neg b \vee d) \wedge$
$(a \vee \neg b \vee e) \wedge$
$(\neg b \vee \neg d \vee \neg e) \wedge$
$(a \vee b \vee c \vee d) \wedge$
$(a \vee b \vee c \vee \neg d) \wedge$
$(a \vee b \vee \neg c \vee e) \wedge$
$(a \vee b \vee \neg c \vee \neg e)$



# An example of DPLL

Select decision variable: $a$

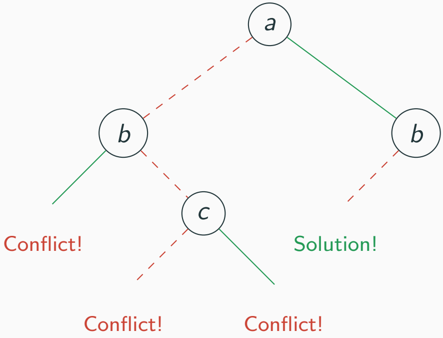
$(a \vee \neg b \vee d) \wedge$
$(a \vee \neg b \vee e) \wedge$
$(\neg b \vee \neg d \vee \neg e) \wedge$
$(a \vee b \vee c \vee d) \wedge$
$(a \vee b \vee c \vee \neg d) \wedge$
$(a \vee b \vee \neg c \vee e) \wedge$
$(a \vee b \vee \neg c \vee \neg e)$



# An example of DPLL

Select decision variable: $b$

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



# Conflict-Driven Clause Learning algorithm

# Conflict-Driven Clause Learning algorithm

- CDCL
- An extension of DPLL with:
  - Clause learning
  - Non-chronological backtracking
- Clause learning can be performed with various strategies
- CDCL algorithms are use in almost all modern SAT solvers

# Clause learning

During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict.

**Example**

$$\varphi \;=\; (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \ldots$$

# Clause learning

During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict.

**Example**

$$\varphi \ = \ (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

- Assume decision $c = \mathit{False}$ and $f = \mathit{False}$

# Clause learning

During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict.

**Example**

$$\varphi \;=\; (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

- Assume decision $c = \mathit{False}$ and $f = \mathit{False}$
- Assign $a = \mathit{False}$

# Clause learning

During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict.

**Example**

$$\varphi \;=\; (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

- Assume decision $c = \mathit{False}$ and $f = \mathit{False}$
- Assign $a = \mathit{False}$
- Unit propagation $b$

# Clause learning

During backtrack search, for each conflict learn new clause,
which explains and prevents repetition of the same conflict.

**Example**

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

- Assume decision $c = False$ and $f = False$
- Assign $a = False$
- Unit propagation $b$
- Unit propagation $d$

# Clause learning

During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict.

**Example**

$$\varphi \;=\; (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

- Assume decision $c = \mathit{False}$ and $f = \mathit{False}$
- Assign $a = \mathit{False}$
- Unit propagation $b$
- Unit propagation $d$ and $e$
- A conflict is reached: $(\neg d \vee \neg e \vee f)$ is unsatisfied
- $\varphi \wedge \neg a \wedge \neg c \wedge \neg f \to \bot$, therefore $\varphi \to a \vee c \vee f$
- Learn new clause $(a \vee c \vee f)$

# Non-chronological backtracking

- aka conflict directed backjumping
- During backtrack search, for each conflict backtrack to one of the causes of conflict.

# Non-chronological backtracking

**Example**

$$\begin{aligned}\varphi \;=\; & (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge \\ & (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)\end{aligned}$$

# Non-chronological backtracking

**Example**

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge$$
$$(a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)$$

- Assume decision $c = \mathit{False}$, $f = \mathit{False}$, $h = \mathit{False}$ and $i = \mathit{False}$
- Assignment $a = \mathit{False}$ cause conflict. Learnt clause $(a \vee c \vee f)$ implies $a$
- Unit propagation $g$

# Non-chronological backtracking

**Example**

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge$$
$$(a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)$$

- Assume decision $c = False$, $f = False$, $h = False$ and $i = False$
- Assignment $a = False$ cause conflict. Learnt clause $(a \vee c \vee f)$ implies $a$
- Unit propagation $g$, $b$

# Non-chronological backtracking

**Example**

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge$$
$$(a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)$$

- Assume decision $c = \mathit{False}$, $f = \mathit{False}$, $h = \mathit{False}$ and $i = \mathit{False}$
- Assignment $a = \mathit{False}$ cause conflict. Learnt clause $(a \vee c \vee f)$ implies $a$
- Unit propagation $g$, $b$, $d$

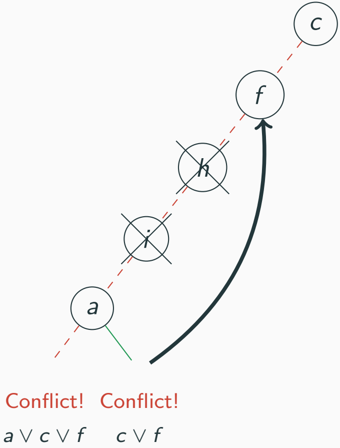
# Non-chronological backtracking

**Example**

$$\varphi \;=\; (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge$$
$$(a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)$$

- Assume decision $c = False$, $f = False$, $h = False$ and $i = False$
- Assignment $a = False$ cause conflict. Learnt clause $(a \vee c \vee f)$ implies $a$
- Unit propagation $g$, $b$, $d$, and $e$
- A conflict is again reached: $(\neg d \vee \neg e \vee f)$ is unsatisfied
- Learn new clause $(c \vee f)$

# Non-chronological backtracking




- Learnt clause: $c \vee f$
- Need to backtrack, given new clause
- Backtrack to decision $f = \mathit{false}$
- Clause learning and non-chronological backtracking are hallmarks of modern SAT solvers