

# Decision Procedures and SAT/SMT Solvers

NAIL094

Petr Kučera

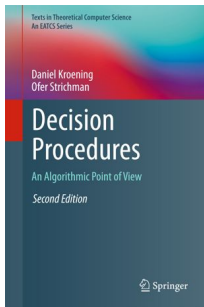
Charles University

2024/25 (1st lecture)

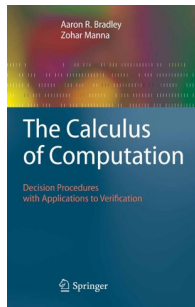
# Introduction

# Overview

- Introduction, motivation, basic notions.
- Normal forms: CNF, DNF, NNF
- Satisfiability of boolean formulas
  - Modern SAT solvers
  - Local search
  - Binary decision diagrams (BDD)
- Decision procedures for theories
  - Equality and uninterpreted functions
  - Linear arithmetic
  - Bit vectors
  - Arrays, memory, pointers
- Combination of theories
- Quantified boolean formulas (QBF)



Kroening, D., & Strichman, O. (2016). Decision procedures. Springer.



Bradley, A. R., & Manna, Z. (2007). The calculus of computation. Springer.



Handbook of satisfiability 2nd  
Edition. IOS press 2021.  
Editors: Armin Biere, Marijn Heule,  
Hans Van Maaren and Toby Walsh

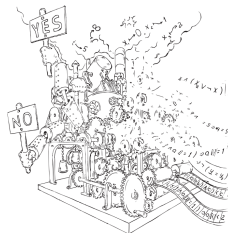


Dennis Yurichev (2024).  
SAT/SMT by Example

# Decision Procedure

## Intuition

A **decision procedure** is an algorithm that, given a logical formula, decides if it is satisfiable.



**Satisfiable**  $\rightarrow$  satisfying assignment (model)

**Unsatisfiable**  $\rightarrow$  proof of unsatisfiability

# Applications

- Model checking
- Hardware verification
  - Verifying designs of electronic circuits
- Software verification
  - Verifying that an assertion in code cannot be violated
- Compiler optimizations
  - Correctness of transformations
- Software package dependencies
  - Dependency hell
- Planning and scheduling
- Chemical reaction networks
- ...

# Propositional SAT Solving



# Language of Propositional Logic

A propositional logic formula is defined by the following grammar:

```
fla : fla ∧ fla | ¬fla | (fla) | atom
atom : boolean-identifier | true | false
```

- Other connectives can be derived using  $\wedge$  and  $\neg$ 
  - $\vee$ ,  $\implies$ ,  $\iff$ ,  $\oplus$  (XOR), ...

## Example

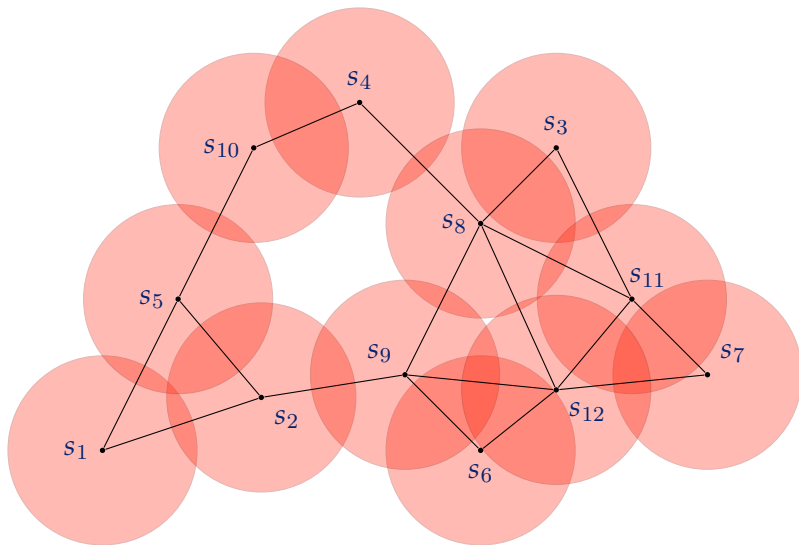
- $(x \implies (y \oplus z)) \vee (y \iff z)$
- $(y \wedge x \implies z) \iff ((y \implies z) \wedge (x \implies z))$
- $(x \vee y) \iff \neg(\neg x \wedge \neg y)$

# Motivation I — Radio Stations

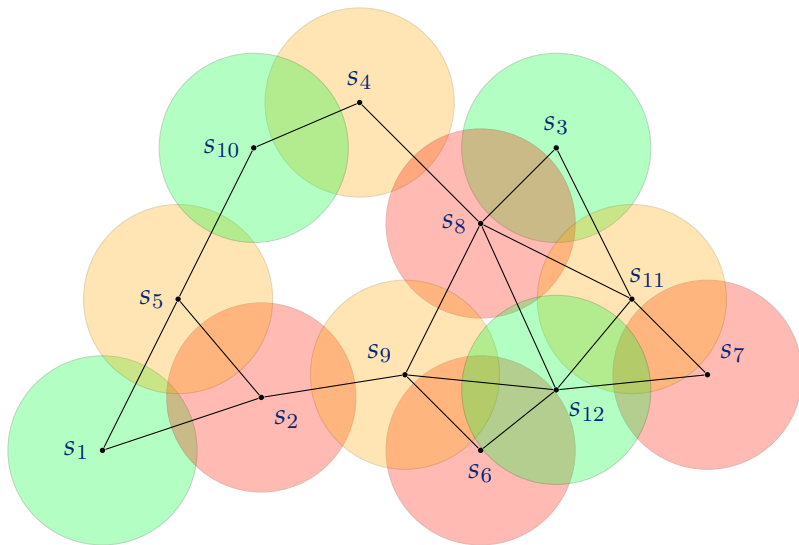
- Consider a set of radio stations  $S = \{s_1, \dots, s_n\}$
- Every station should get allocated one of  $k$  transmission frequencies for some  $k < n$
- Stations that are too close should not share the same frequency

Graph coloring problem

# Radio Stations — Example Instance



# Radio Stations — A Possible Solution



# Radio Stations — Model

We want to model the problem in propositional logic.

- Variables  $x_{i,j}$ ,  $i = 1, \dots, n$ ,  $j = 1, \dots, k$
- $x_{i,j} = 1$  — “Radio station  $s_i$  has frequency  $j$ ”
- $E$  denotes the set of pairs of stations that are too close

# Radio Stations — Formula

- Every station is assigned at least one frequency

$$\bigwedge_{i=1}^n \bigvee_{j=1}^k x_{i,j}$$

- Every station is assigned not more than one frequency

$$\bigwedge_{i=1}^n \bigwedge_{j=1}^{k-1} (x_{i,j} \implies \bigwedge_{j < t \leq k} \neg x_{i,t})$$

- Close stations are not assigned the same frequency.

$$\bigwedge_{t=1}^k (x_{i,t} \implies \neg x_{j,t}) \quad (\text{for each } (i, j) \in E)$$

# Motivation II — Code equivalence

## Snippet A

```
1  if(!a && !b) h();  
2  else  
3      if(!a) g();  
4      else f();
```

## Snippet B

```
1  if(a) f();  
2  else  
3      if(b) g();  
4      else h();
```

Are the snippets A and B equivalent?

# If-then-else

- Ternary operator **if-then-else** can be represented with a propositional formula

$$\text{"if } x \text{ then } y \text{ else } z" \equiv (x \wedge y) \vee (\neg x \wedge z)$$

- Replace boolean variables and function calls with propositional variables
- Form the corresponding propositional formulas and check their equivalence



# Motivation II — Code equivalence

## Snippet A

```
1  if  $\neg a \wedge \neg b$  then  $h$ 
2  else
3      if  $\neg a$  then  $g$ 
4      else  $f$ 
```

## Snippet B

```
1  if  $a$  then  $f$ 
2  else
3      if  $b$  then  $g$ 
4      else  $h$ 
```

# Motivation II — Code equivalence

## Snippet A

```
1  if  $\neg a \wedge \neg b$  then  $h$ 
2  else
3      if  $\neg a$  then  $g$ 
4      else  $f$ 
```

## Snippet B

```
1  if  $a$  then  $f$ 
2  else
3      if  $b$  then  $g$ 
4      else  $h$ 
```

$$\varphi_a = \neg a \wedge \neg b \wedge h \vee \\ \neg(\neg a \wedge \neg b) \wedge (\neg a \wedge g \vee a \wedge f)$$

# Motivation II — Code equivalence

## Snippet A

```
1  if  $\neg a \wedge \neg b$  then  $h$ 
2  else
3      if  $\neg a$  then  $g$ 
4      else  $f$ 
```

$$\begin{aligned}\varphi_a = & \neg a \wedge \neg b \wedge h \vee \\ & \neg(\neg a \wedge \neg b) \wedge (\neg a \wedge g \vee a \wedge f)\end{aligned}$$

## Snippet B

```
1  if  $a$  then  $f$ 
2  else
3      if  $b$  then  $g$ 
4      else  $h$ 
```

$$\begin{aligned}\varphi_b = & a \wedge f \vee \\ & \neg a \wedge (b \wedge g \vee \neg b \wedge h)\end{aligned}$$

# Motivation II — Code equivalence

## Snippet A

```
1  if  $\neg a \wedge \neg b$  then  $h$ 
2  else
3      if  $\neg a$  then  $g$ 
4      else  $f$ 
```

## Snippet B

```
1  if  $a$  then  $f$ 
2  else
3      if  $b$  then  $g$ 
4      else  $h$ 
```

$$\begin{aligned}\varphi_a = & \neg a \wedge \neg b \wedge h \vee \\ & \neg(\neg a \wedge \neg b) \wedge (\neg a \wedge g \vee a \wedge f)\end{aligned}$$

$$\begin{aligned}\varphi_b = & a \wedge f \vee \\ & \neg a \wedge (b \wedge g \vee \neg b \wedge h)\end{aligned}$$

Check the validity of  $\varphi_a \iff \varphi_b$

# Assignments and Satisfaction

- $\mathbf{x}$  denotes a set of variables  $x_1, \dots, x_n$
- $\text{lit}(\mathbf{x})$  literals over variables in  $\mathbf{x}$
- **Assignment**  $\mathbf{a}$  maps propositional variables to **true** or **false**
  - 1 or 0,  $\top$  or  $\perp$
- An assignment  $\mathbf{a}$  **satisfies formula**  $\varphi$  if  $\varphi(\mathbf{a})$  evaluates to **true**
  - $\mathbf{a}$  is a **model** of  $\varphi$
  - $\mathbf{a} \models \varphi$
- **Satisfiable** formula admits a model
- **Unsatisfiable** formula has no model
  - **Contradiction**
- **Valid** formula is satisfied by every assignment
  - **Tautology**

$\varphi$  is a tautology  $\iff \neg\varphi$  is unsatisfiable.

# Negation Normal Form

**Literal** a propositional variable  $x$  or its negation  $\neg x$

**NNF** a formula is in **negation normal form** if it uses only  $\wedge$ ,  $\vee$ ,  $\neg$  and negation is only in front of variables (in literals)

## Example

$$\varphi_1 = \left[ \bigwedge_{i=1}^n \bigwedge_{j=i+1}^n \neg(x_i \wedge \neg x_j) \right] \implies y$$

Not NNF

$$\varphi_2 = \left[ \bigvee_{i=1}^n \bigvee_{j=i+1}^n x_i \wedge \neg x_j \right] \vee y$$

NNF

$\varphi_2$  is a NNF equivalent to  $\varphi_1$

# CNF and DNF

**Term** a conjunction of literals, e.g.  $x \wedge \neg y \wedge \neg z$

**Clause** a disjunction of literals, e.g.  $x \vee \neg y \vee \neg z$

**CNF** a formula is in **conjunctive normal form** if it is a conjunction of clauses

**DNF** a formula is in **disjunctive normal form** if it is a disjunction of terms

- The empty term and empty CNF are valid —  $\top$
- The empty clause and empty DNF are contradictions —  $\perp$

## Example

$$\varphi_1 = (x \vee \neg y \vee z) \wedge (\neg x \vee y)$$

CNF

$$\varphi_2 = (x \wedge y) \vee (\neg x \wedge \neg y) \vee (\neg x \wedge z) \vee (y \wedge z)$$

an equivalent DNF

# SAT and Related Problems

Given a CNF formula  $\varphi$

**SAT** Is  $\varphi$  satisfiable?

**UNSAT** Is  $\varphi$  unsatisfiable?

**MaxSAT** Maximize the number of satisfied clauses of  $\varphi$

**#SAT** Count the models of  $\varphi$

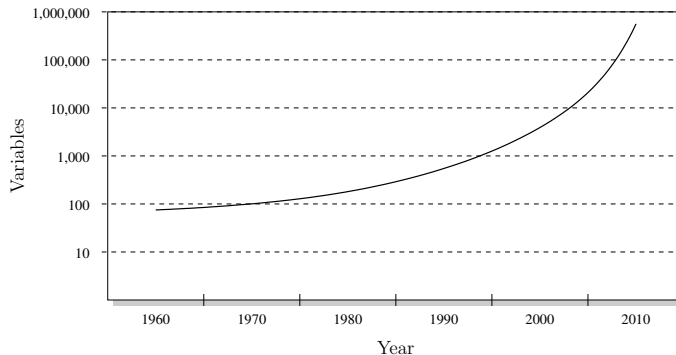
**Model Enumeration** Enumerate the models of  $\varphi$



# Why Study Propositional SAT?

- Interesting from both theoretical and practical perspective
- The first problem to be proven NP-complete
  - Cook, 1971; Levin, 1973
- Generic problem — many problems encoded into SAT
  - Hardware and software verification
  - Planning and scheduling
  - Product configuration
  - ...and many others

# Progress in SAT Solving



The size of industrial CNF formulas that are regularly solved by SAT solvers in a few hours, according to year.

Image source: Decision Procedures. Kroening D., Strichman O.

# Progress in SAT Solving

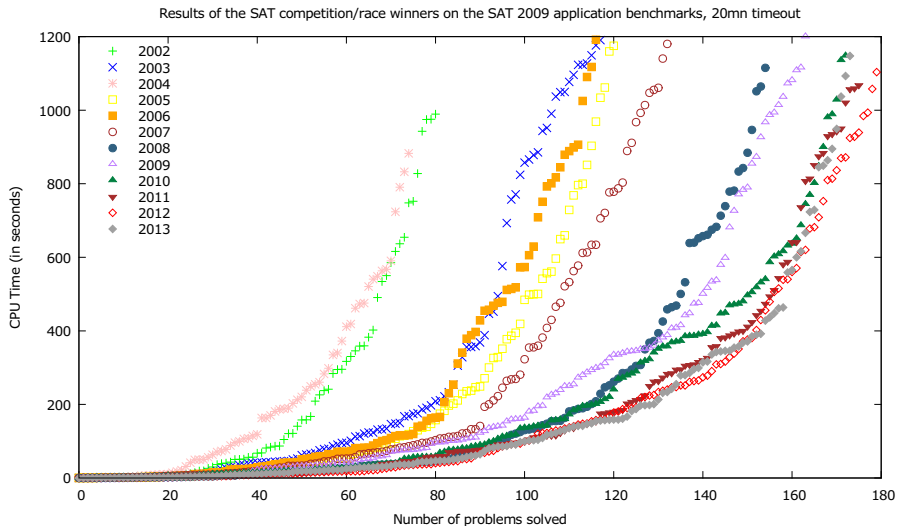
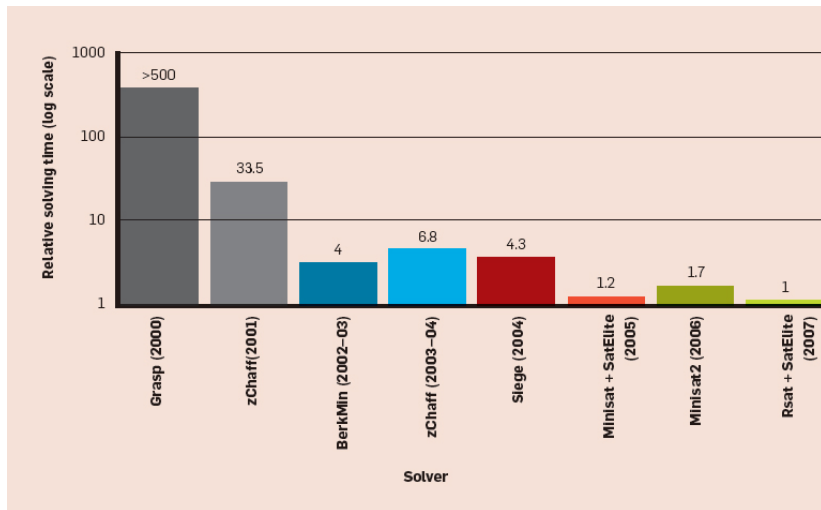


Image source: Decision Procedures. Kroening D., Strichman O.

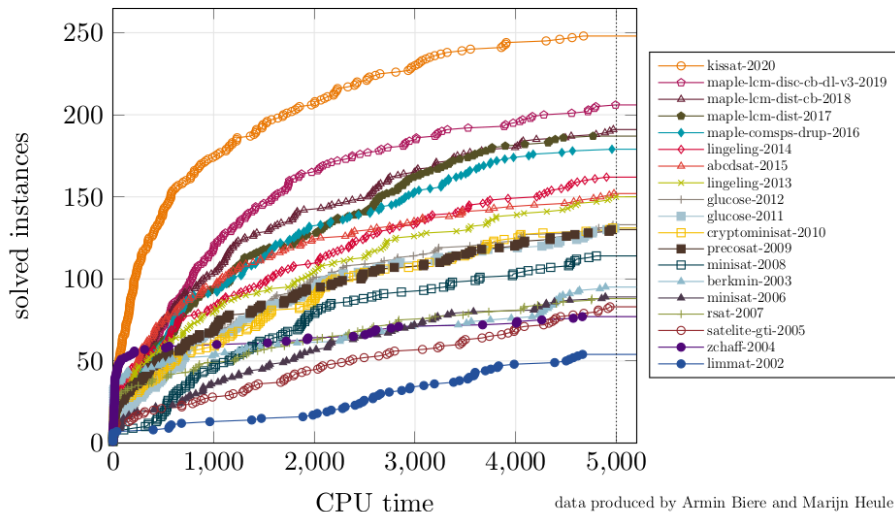
# Progress in SAT Solving



Sharad Malik, Lintao Zhang Communications of the ACM, August 2009, Vol. 52 No. 8, Pages 76–82

# Progress in SAT Solving

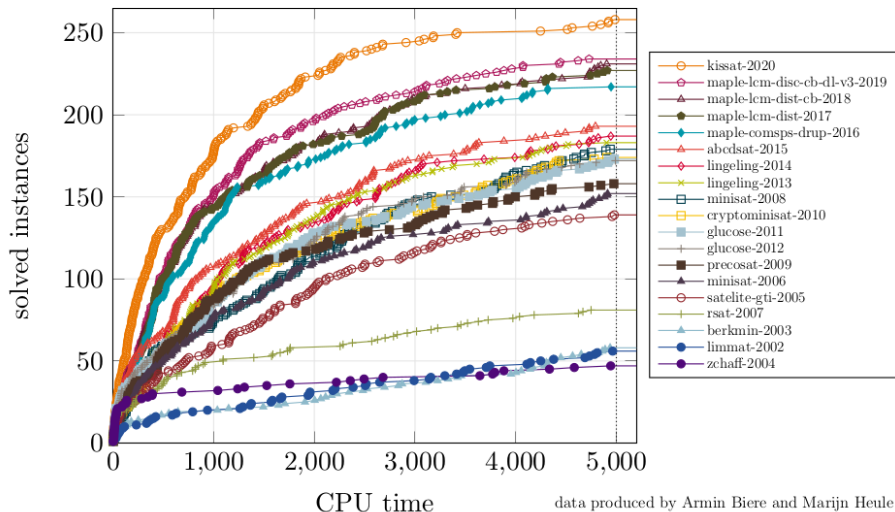
## SAT Competition Winners on the SC2020 Benchmark Suite



Source and more details: <http://fmv.jku.at/kissat>

# Progress in SAT Solving

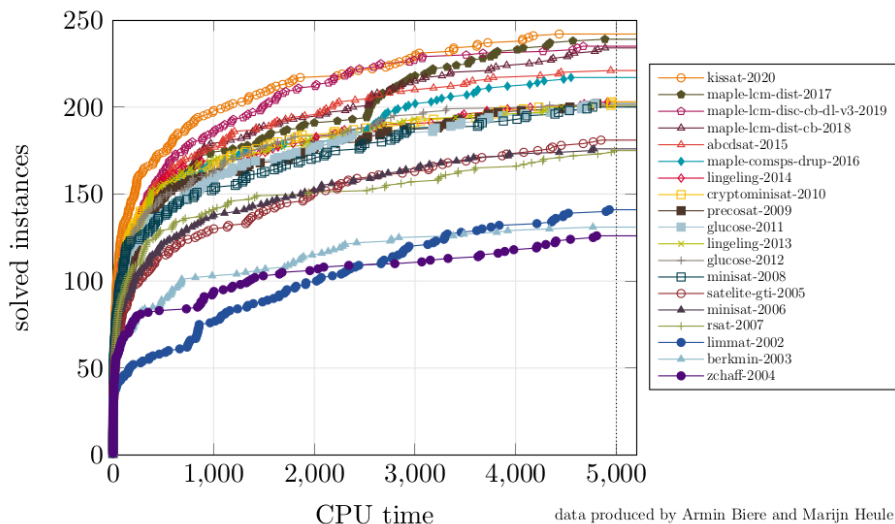
## SAT Competition Winners on the SC2019 Benchmark Suite



Source and more details: <http://fmv.jku.at/kissat>

# Progress in SAT Solving

## SAT Competition Winners on the SC2011 Benchmark Suite



Source and more details: <http://fmv.jku.at/kissat>

# SAT encodings



# Converting Formula to an Equivalent CNF

## Lemma

*To every formula we can construct an equivalent CNF.*

- 1 Convert to NNF
  - a Rewrite connectives using only  $\wedge$ ,  $\vee$ , and  $\neg$
  - b Use De Morgan's laws to propagate negations to variables
  - c Remove double negation ( $\neg\neg x = x$ )
- 2 Use distributivity to propagate disjunction over conjunction

# Converting Formula to an Equivalent CNF

## Lemma

*To every formula we can construct an equivalent CNF.*

- 1 Convert to NNF
  - a Rewrite connectives using only  $\wedge$ ,  $\vee$ , and  $\neg$
  - b Use De Morgan's laws to propagate negations to variables
  - c Remove double negation ( $\neg\neg x = x$ )
- 2 Use distributivity to propagate disjunction over conjunction

The result can be exponentially larger!

- Any CNF equivalent to  $\bigvee_{i=1}^n (x_i \wedge y_i)$  has at least  $2^n$  clauses

# Tseitin's Encoding

## Lemma (Tseitin)

*Every formula  $\varphi$  can be converted to an equisatisfiable formula  $\psi$  in CNF which is larger only by a constant factor.*

**Equisatisfiable**  $\psi$  is satisfiable if and only if  $\varphi$  is satisfiable

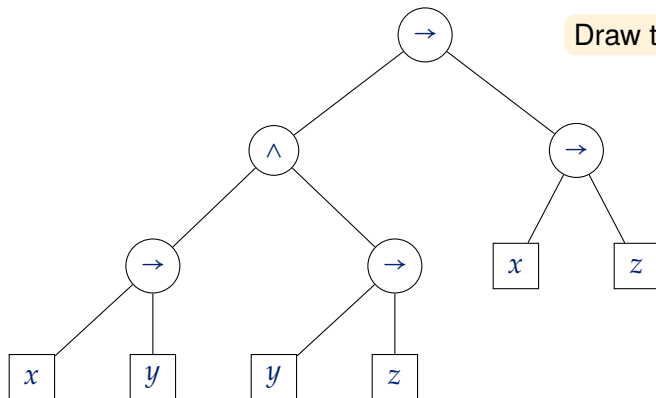
Idea:

- 1 Draw a derivation tree of the formula
- 2 Assign a fresh variable to each connective
- 3 Add clauses to define the function of each connective
- 4 Add the root variable as a unit clause

Can be used to convert a logical circuit into a CNF as well.

# Tseitin's Encoding (example)

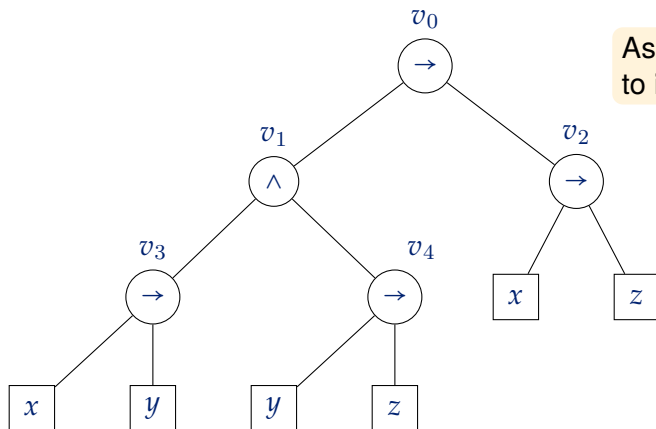
$$(x \rightarrow y) \wedge (y \rightarrow z) \rightarrow (x \rightarrow z)$$



Draw the derivation tree

# Tseitin's Encoding (example)

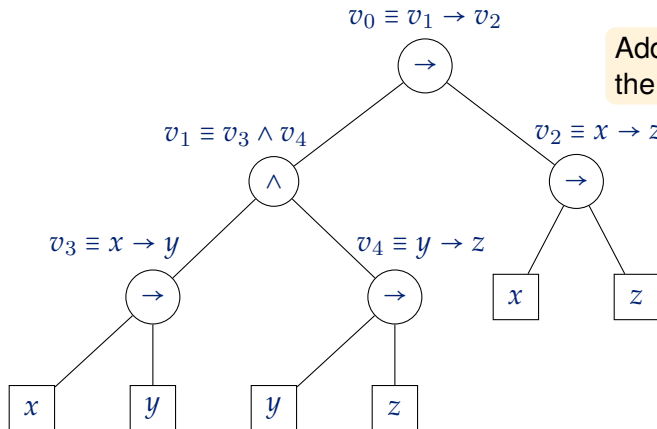
$$(x \rightarrow y) \wedge (y \rightarrow z) \rightarrow (x \rightarrow z)$$



Assign variables  
to its nodes

# Tseitin's Encoding (example)

$$(x \rightarrow y) \wedge (y \rightarrow z) \rightarrow (x \rightarrow z)$$



Add definitions of  
the new variables

# Tseitin's Encoding (example)

$$(x \rightarrow y) \wedge (y \rightarrow z) \rightarrow (x \rightarrow z)$$

- 1 Rewrite as a conjunction of definitions of new variables

$$v_0$$

$$v_0 \equiv v_1 \rightarrow v_2$$

$$v_1 \equiv v_3 \wedge v_4$$

$$v_2 \equiv x \rightarrow z$$

$$v_3 \equiv x \rightarrow y$$

$$v_4 \equiv y \rightarrow z$$

# Tseitin's Encoding (example)

$$(x \rightarrow y) \wedge (y \rightarrow z) \rightarrow (x \rightarrow z)$$

- 1 Rewrite as a conjunction of definitions of new variables
- 2 rewrite definitions only using  $\neg$ ,  $\wedge$ , and  $\vee$

$$v_0$$

$$v_0 \equiv v_1 \rightarrow v_2 \equiv \neg v_1 \vee v_2$$

$$v_1 \equiv v_3 \wedge v_4$$

$$v_2 \equiv x \rightarrow z \equiv \neg x \vee z$$

$$v_3 \equiv x \rightarrow y \equiv \neg x \vee y$$

$$v_4 \equiv y \rightarrow z \equiv \neg y \vee z$$



# Tseitin's Encoding (example)

$$(x \rightarrow y) \wedge (y \rightarrow z) \rightarrow (x \rightarrow z)$$

- 1 Rewrite as a conjunction of definitions of new variables
- 2 rewrite definitions only using  $\neg$ ,  $\wedge$ , and  $\vee$

$v_0$

$$v_0 \equiv \neg v_1 \vee v_2 \qquad (v_0 \rightarrow \neg v_1 \vee v_2) \wedge (\neg v_1 \vee v_2 \rightarrow v_0)$$

$$v_1 \equiv v_3 \wedge v_4 \qquad (v_1 \rightarrow v_3 \wedge v_4) \wedge (v_3 \wedge v_4 \rightarrow v_1)$$

$$v_2 \equiv \neg x \vee z \qquad (v_2 \rightarrow \neg x \vee z) \wedge (\neg x \vee z \rightarrow v_2)$$

$$v_3 \equiv \neg x \vee y \qquad (v_3 \rightarrow \neg x \vee y) \wedge (\neg x \vee y \rightarrow v_3)$$

$$v_4 \equiv \neg y \vee z \qquad (v_4 \rightarrow \neg y \vee z) \wedge (\neg y \vee z \rightarrow v_4)$$

# Tseitin's Encoding (example)

$$(x \rightarrow y) \wedge (y \rightarrow z) \rightarrow (x \rightarrow z)$$

- 1 Rewrite as a conjunction of definitions of new variables
- 2 rewrite definitions only using  $\neg$ ,  $\wedge$ , and  $\vee$
- 3 rewrite as a conjunction of clauses

$v_0$

$$v_0 \equiv \neg v_1 \vee v_2$$

$$v_1 \equiv v_3 \wedge v_4$$

$$v_2 \equiv \neg x \vee z$$

$$v_3 \equiv \neg x \vee y$$

$$v_4 \equiv \neg y \vee z$$

$v_0$

$$(v_0 \rightarrow \neg v_1 \vee v_2) \wedge (\neg v_1 \rightarrow v_0) \wedge (v_2 \rightarrow v_0)$$

$$(v_1 \rightarrow v_3) \wedge (v_1 \rightarrow v_4) \wedge (v_3 \wedge v_4 \rightarrow v_1)$$

$$(v_2 \rightarrow \neg x \vee z) \wedge (\neg x \rightarrow v_2) \wedge (z \rightarrow v_2)$$

$$(v_3 \rightarrow \neg x \vee y) \wedge (\neg x \rightarrow v_3) \wedge (y \rightarrow v_3)$$

$$(v_4 \rightarrow \neg y \vee z) \wedge (\neg y \rightarrow v_4) \wedge (z \rightarrow v_4)$$

# Tseitin's Encoding (example)

$$(x \rightarrow y) \wedge (y \rightarrow z) \rightarrow (x \rightarrow z)$$

- 1 Rewrite as a conjunction of definitions of new variables
- 2 rewrite definitions only using  $\neg$ ,  $\wedge$ , and  $\vee$
- 3 rewrite as a conjunction of clauses

$v_0$

$$v_0 \equiv \neg v_1 \vee v_2$$

$$v_1 \equiv v_3 \wedge v_4$$

$$v_2 \equiv \neg x \vee z$$

$$v_3 \equiv \neg x \vee y$$

$$v_4 \equiv \neg y \vee z$$

$v_0$

$$(\neg v_0 \vee \neg v_1 \vee v_2) \wedge (v_1 \vee v_0) \wedge (\neg v_2 \vee v_0)$$

$$(\neg v_1 \vee v_3) \wedge (\neg v_1 \vee v_4) \wedge (\neg v_3 \vee \neg v_4 \vee v_1)$$

$$(\neg v_2 \vee \neg x \vee z) \wedge (x \vee v_2) \wedge (\neg z \vee v_2)$$

$$(\neg v_3 \vee \neg x \vee y) \wedge (x \vee v_3) \wedge (\neg y \vee v_3)$$

$$(\neg v_4 \vee \neg y \vee z) \wedge (y \vee v_4) \wedge (\neg z \vee v_4)$$

# Tseitin's Encoding (example)

$$(x \rightarrow y) \wedge (y \rightarrow z) \rightarrow (x \rightarrow z)$$

- 1 Rewrite as a conjunction of definitions of new variables
- 2 rewrite definitions only using  $\neg$ ,  $\wedge$ , and  $\vee$
- 3 rewrite as a conjunction of clauses

$v_0$

$$v_0 \equiv \neg v_1 \vee v_2$$

$$v_1 \equiv v_3 \wedge v_4$$

$$v_2 \equiv \neg x \vee z$$

$$v_3 \equiv \neg x \vee y$$

$$v_4 \equiv \neg y \vee z$$

$v_0$

$$(\neg v_0 \vee \neg v_1 \vee v_2) \wedge (v_1 \vee v_0) \wedge (\neg v_2 \vee v_0)$$

$$(\neg v_1 \vee v_3) \wedge (\neg v_1 \vee v_4) \wedge (\neg v_3 \vee \neg v_4 \vee v_1)$$

$$(\neg v_2 \vee \neg x \vee z) \wedge (x \vee v_2) \wedge (\neg z \vee v_2)$$

$$(\neg v_3 \vee \neg x \vee y) \wedge (x \vee v_3) \wedge (\neg y \vee v_3)$$

$$(\neg v_4 \vee \neg y \vee z) \wedge (y \vee v_4) \wedge (\neg z \vee v_4)$$

Equisatisfiable, not equivalent.

# Resolution

# Certifying Unsatisfiability

- CNF  $\varphi$  is satisfiable  $\implies$  SAT solver returns a model
  - easy to check its correctness
- $\varphi$  is unsatisfiable  $\implies$  SAT solver returns UNSAT
  - how to check that the answer is correct?

# Certifying Unsatisfiability

- CNF  $\varphi$  is satisfiable  $\implies$  SAT solver returns a model
  - easy to check its correctness
- $\varphi$  is unsatisfiable  $\implies$  SAT solver returns UNSAT
  - how to check that the answer is correct?
- SAT solvers can return the proof of unsatisfiability
  - resolution refutation of  $\varphi$

# Resolution

Given clauses  $A$ ,  $B$ , and a variable  $x$

$$\begin{array}{c} \text{parent clauses} \\ \hline A \vee x \qquad B \vee \neg x \\ \hline A \vee B \\ \hline \text{resolvent} \end{array}$$

## Definition

**Resolution derivation** of a clause  $C$  from a CNF  $\varphi$  is a sequence of clauses  $C_1, \dots, C_k$  such that  $C_k = C$  and for each  $i = 1, \dots, k$  either

- $C_i \in \varphi$ , or
- $C_i$  is a resolvent of some clauses preceding it in the list.

**Resolution refutation** of  $\varphi$  is the resolution derivation of the empty clause  $\perp$  (contradiction).

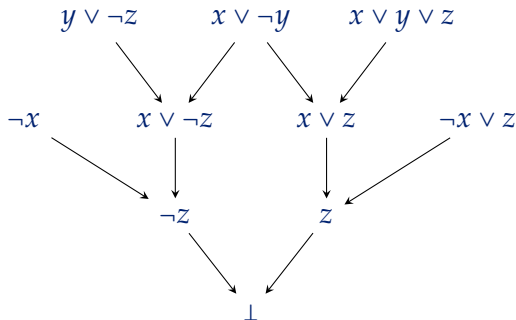


# Resolution Graph

- **Resolution graph** represents a resolution derivation
- A directed acyclic graph (DAG)
  - nodes** clauses
  - leaves** clauses of  $\varphi$
  - inner nodes** resolvents
  - edges** from parent clauses to resolvents
  - sink node** the derived clause ( $\perp$  in case of a refutation)

# Resolution Graph (example)

$$\varphi = \neg x \wedge (y \vee \neg z) \wedge (x \vee \neg y) \wedge (x \vee y \vee z) \wedge (\neg x \vee z)$$

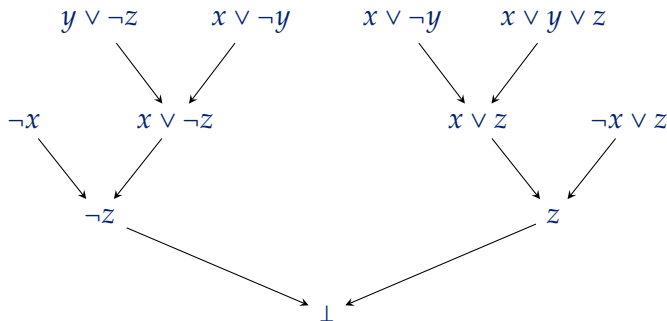


# Tree Resolution

**Tree resolution refutation** resolution graph is a tree

- clauses of  $\varphi$  can be in several leaves

$$\varphi = \neg x \wedge (y \vee \neg z) \wedge (x \vee \neg y) \wedge (x \vee y \vee z) \wedge (\neg x \vee z)$$



# Resolution and Tree Resolution

$\varphi \models C$  every model of  $\varphi$  is a model of  $C$  as well.

- Clause  $C$  is an **implicate** of  $\varphi$
- Equivalent to  $\varphi \wedge \bigwedge_{l \in C} \neg l \models \perp$

$\varphi \models \perp$  if and only if  $\varphi$  is unsatisfiable

$\varphi \vdash C$  clause  $C$  can be derived by resolution from  $\varphi$ .

(Tree) Resolution is sound and complete

$\varphi \models \perp$  if and only if  $\varphi \vdash \perp$  for every CNF formula  $\varphi$ .

- Resolution refutations can have exponential length
  - Pigeon hole principle formulas
- Tree resolution refutations may be exponentially longer than general resolution refutations

# Unit Resolution

**unit resolution** one of the parent clauses is a unit clause (single literal)

$\varphi \vdash_1 C$  clause  $C$  can be derived from  $\varphi$  by unit resolution

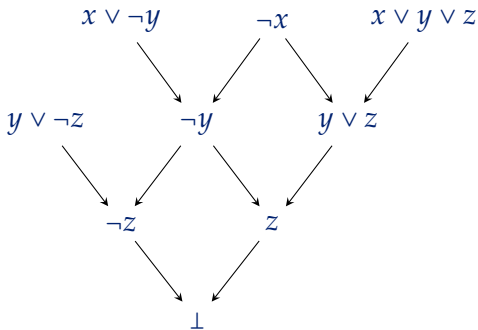
- Unit resolution is **sound** but **incomplete**

$$\varphi \equiv (x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$$

- Unsatisfiable
  - Has no unit resolution refutation
- 
- Unit resolution refutation can be found in linear time if it exists

# Unit Resolution (Example)

$$\varphi = \neg x \wedge (y \vee \neg z) \wedge (x \vee \neg y) \wedge (x \vee y \vee z) \wedge (\neg x \vee z)$$



# Partial Assignments — Notation

$\mathbf{x}$  set of variables

$\text{lit}(\mathbf{x})$  literals over variables in  $\mathbf{x}$

**partial assignment** a non-contradictory set of literals, considered as a conjunction of literals

$\varphi[\alpha]$  Application of a partial assignment  $\alpha \subseteq \text{lit}(\mathbf{x})$ :

- Removed clauses containing a literal from  $\alpha$
- Removed the negations of the literals in  $\alpha$  from the remaining clauses

$$\varphi = (y \vee \neg z) \wedge (x \vee \neg y) \wedge (x \vee y \vee z) \wedge (\neg x \vee z)$$

$$\varphi[\neg x] = (y \vee \neg z) \wedge (\neg y) \wedge (y \vee z)$$

$$\varphi[x, \neg z] = \perp \quad (\text{The empty clause — contradiction})$$

$$\varphi[x, y, z] = \top \quad (\text{The empty CNF — satisfied})$$

# Unit Propagation Algorithm

---

**Function**  $\text{UnitProp}(\varphi)$

---

**Input:** CNF formula  $\varphi$  on variables  $x$

**Output:**  $(\alpha, \psi)$  where  $\alpha$  is a set of literals which can be derived by unit resolution from  $\varphi$ ,  $\psi = \varphi[\alpha]$ .

**if**  $\perp \in \varphi$  **then return**  $(\alpha, \perp)$

$\alpha \leftarrow \emptyset$

**while**  $\varphi$  contains a unit clause  $l$  **do**

$\alpha \leftarrow \alpha \cup \{l\}$

$\varphi \leftarrow \varphi[l]$

**if**  $\perp \in \varphi$  **then return**  $(\alpha, \perp)$

**return**  $(\alpha, \varphi)$

---

- Efficient procedure
  - Linear time implementation
  - Efficient data structures (watched literals)
- Used very often in SAT solvers



# Unit Propagation Example

1  $\varphi = \underbrace{\neg x}_{\text{unit clause}} \wedge (y \vee \neg z) \wedge (x \vee \neg y) \wedge (x \vee y \vee z) \wedge (\neg x \vee z), \alpha = \emptyset$

# Unit Propagation Example

1  $\varphi = \underbrace{\neg x}_{\text{unit clause}} \wedge (y \vee \neg z) \wedge (x \vee \neg y) \wedge (x \vee y \vee z) \wedge (\neg x \vee z), \alpha = \emptyset$

2  $\varphi = \underbrace{\neg x \wedge (y \vee \neg z)}_{\text{unit clause}} \wedge (\underbrace{x \vee \neg y}_{\text{unit clause}}) \wedge (x \vee y \vee z) \wedge (\neg x \vee z), \alpha = \{\neg x\}$

# Unit Propagation Example

1  $\varphi = \underbrace{\neg x}_{\text{unit clause}} \wedge (y \vee \neg z) \wedge (x \vee \neg y) \wedge (x \vee y \vee z) \wedge (\neg x \vee z), \alpha = \emptyset$

2  $\varphi = \underbrace{\neg x \wedge (y \vee \neg z)}_{\text{unit clause}} \wedge (\underbrace{x \vee \neg y}_{\text{unit clause}}) \wedge (x \vee y \vee z) \wedge (\neg x \vee z), \alpha = \{\neg x\}$

3  $\varphi = \neg x \wedge (y \vee \neg z) \wedge (x \vee \neg y) \wedge (\underbrace{x \vee y \vee z}_{\text{unit clause}}) \wedge (\neg x \vee z),$   
 $\alpha = \{\neg x, \neg y\}$

# Unit Propagation Example

1  $\varphi = \underbrace{\neg x}_{\text{unit clause}} \wedge (y \vee \neg z) \wedge (x \vee \neg y) \wedge (x \vee y \vee z) \wedge (\neg x \vee z), \alpha = \emptyset$

2  $\varphi = \neg x \wedge (y \vee \neg z) \wedge \underbrace{(x \vee \neg y)}_{\text{unit clause}} \wedge (x \vee y \vee z) \wedge (\neg x \vee z), \alpha = \{\neg x\}$

3  $\varphi = \neg x \wedge (y \vee \neg z) \wedge (x \vee \neg y) \wedge \underbrace{(x \vee y \vee z)}_{\text{unit clause}} \wedge (\neg x \vee z),$

$$\alpha = \{\neg x, \neg y\}$$

4  $\varphi = \neg x \wedge \underbrace{(y \vee \neg z)}_{\text{empty clause}} \wedge (x \vee \neg y) \wedge (x \vee y \vee z) \wedge (\neg x \vee z),$

$$\alpha = \{\neg x, \neg y, z\}$$

# Unit Propagation Example

1  $\varphi = \underbrace{\neg x}_{\text{unit clause}} \wedge (y \vee \neg z) \wedge (x \vee \neg y) \wedge (x \vee y \vee z) \wedge (\neg x \vee z), \alpha = \emptyset$

2  $\varphi = \neg x \wedge (y \vee \neg z) \wedge \underbrace{(x \vee \neg y)}_{\text{unit clause}} \wedge (x \vee y \vee z) \wedge (\neg x \vee z), \alpha = \{\neg x\}$

3  $\varphi = \neg x \wedge (y \vee \neg z) \wedge (x \vee \neg y) \wedge \underbrace{(x \vee y \vee z)}_{\text{unit clause}} \wedge (\neg x \vee z),$

$$\alpha = \{\neg x, \neg y\}$$

4  $\varphi = \neg x \wedge \underbrace{(y \vee \neg z)}_{\text{empty clause}} \wedge (x \vee \neg y) \wedge (x \vee y \vee z) \wedge (\neg x \vee z),$

$$\alpha = \{\neg x, \neg y, z\}$$

Empty clause derived — Unit propagation returns  $(\{\neg x, \neg y, z\}, \perp)$ .

DPLL

- DP algorithm — existential quantification (Davis and Putnam, 1960) by DP-elimination
  - High space complexity (can soon blow up exponentially)
- Davis Putnam Logemann Loveland (Davis, Logemann, and Loveland, 1962)
- Branch and bound algorithm
  - Branch on values of a variable
  - Use unit propagation to prune the search tree
  - Polynomial space complexity

# DPLL Algorithm

---

**Function** DPLL (CNF  $\varphi$ )

---

**Output:** A set of literals (partial model) or UNSAT

$(\alpha, \psi) \leftarrow \text{UnitProp}(\varphi)$

**if**  $\psi = \emptyset$  **then return**  $\alpha$

**if**  $\perp \in \psi$  **then return** UNSAT

$l \leftarrow$  a literal in  $\psi$

$\beta \leftarrow \text{DPLL}(\psi[l])$

**if**  $\beta \neq \text{UNSAT}$  **then**

**return**  $\alpha \cup \beta \cup \{l\}$

$\beta \leftarrow \text{DPLL}(\psi[\neg l])$

**if**  $\beta \neq \text{UNSAT}$  **then**

**return**  $\alpha \cup \beta \cup \{\neg l\}$

**return** UNSAT

---



# DPLL — running example

$$\varphi = (x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$

1 Decide  $x_1$

$$(x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\underbrace{\neg x_1 \vee \neg x_3}_{\text{unit clause}}) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$

# DPLL — running example

$$\varphi = (x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$

1 Decide  $x_1$

$$(x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\underbrace{\neg x_1 \vee \neg x_3}_{\text{unit clause}}) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$

2 Derive  $\neg x_3$  by unit propagation

$$(x_1 \vee x_3 \vee x_4) \wedge (\underbrace{\neg x_1 \vee x_2 \vee x_3}_{\text{unit clause}}) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (\underbrace{x_3 \vee x_4}_{\text{unit clause}})$$

# DPLL — running example

$$\varphi = (x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$

1 Decide  $x_1$

$$(x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\underbrace{\neg x_1 \vee \neg x_3}_{\text{unit clause}}) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$

2 Derive  $\neg x_3$  by unit propagation

$$(x_1 \vee x_3 \vee x_4) \wedge (\underbrace{\neg x_1 \vee x_2 \vee x_3}_{\text{unit clause}}) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (\underbrace{x_3 \vee x_4}_{\text{unit clause}})$$

3 Derive  $x_2, x_4$  by unit propagation

$$(x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\underbrace{\neg x_2 \vee \neg x_4}_{\text{empty clause}}) \wedge (x_3 \vee x_4)$$

# DPLL — running example

$$\varphi = (x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$

1 Decide  $\neg x_1$

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

# DPLL — running example

$$\varphi = (x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$

1 Decide  $\neg x_1$

$$(\textcolor{red}{x_1} \vee x_3 \vee x_4) \wedge (\textcolor{green}{\neg x_1} \vee x_2 \vee x_3) \wedge (\textcolor{green}{\neg x_1} \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$

2 Decide  $x_2$

$$(\textcolor{red}{x_1} \vee x_3 \vee x_4) \wedge (\textcolor{green}{\neg x_1} \vee \textcolor{green}{x_2} \vee x_3) \wedge (\textcolor{green}{\neg x_1} \vee \neg x_3) \wedge (\textcolor{red}{\neg x_2} \vee \neg x_4) \wedge (x_3 \vee x_4)$$

⏟  
unit clause

# DPLL — running example

$$\varphi = (x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$

1 Decide  $\neg x_1$

$$(\textcolor{red}{x_1} \vee x_3 \vee x_4) \wedge (\neg \textcolor{red}{x_1} \vee x_2 \vee x_3) \wedge (\neg \textcolor{red}{x_1} \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$

2 Decide  $x_2$

$$(\textcolor{red}{x_1} \vee x_3 \vee x_4) \wedge (\neg \textcolor{red}{x_1} \vee \textcolor{green}{x_2} \vee x_3) \wedge (\neg \textcolor{red}{x_1} \vee \neg x_3) \wedge (\underbrace{\neg \textcolor{red}{x_2} \vee \neg x_4}_{\text{unit clause}}) \wedge (x_3 \vee x_4)$$

3 Derive  $\neg x_4$  by unit propagation

$$(\textcolor{red}{x_1} \vee x_3 \vee \textcolor{red}{x_4}) \wedge (\neg \textcolor{red}{x_1} \vee \textcolor{green}{x_2} \vee x_3) \wedge (\neg \textcolor{red}{x_1} \vee \neg x_3) \wedge (\neg \textcolor{red}{x_2} \vee \textcolor{green}{\neg x_4}) \wedge (\underbrace{x_3 \vee \textcolor{red}{x_4}}_{\text{unit clause}})$$

# DPLL — running example

$$\varphi = (x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$

1 Decide  $\neg x_1$

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

2 Decide  $x_2$

$$(x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\underbrace{\neg x_2 \vee \neg x_4}_{\text{unit clause}}) \wedge (x_3 \vee x_4)$$

3 Derive  $\neg x_4$  by unit propagation

$$(x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \underbrace{\neg x_4}_{\text{unit clause}}) \wedge (x_3 \vee x_4)$$

4 Derive  $x_3$  by unit propagation

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

# DPLL — running example

$$\varphi = (x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$

1 Decide  $\neg x_1$

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

2 Decide  $x_2$

$$(x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\underbrace{\neg x_2 \vee \neg x_4}_{\text{unit clause}}) \wedge (x_3 \vee x_4)$$

3 Derive  $\neg x_4$  by unit propagation

$$(x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \underbrace{\neg x_4}_{\text{unit clause}}) \wedge (x_3 \vee x_4)$$

4 Derive  $x_3$  by unit propagation

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

$\alpha = \{\neg x_1, x_2, x_3, \neg x_4\}$  is a satisfying assignment of  $\varphi$



# DPLL Variants and Extensions

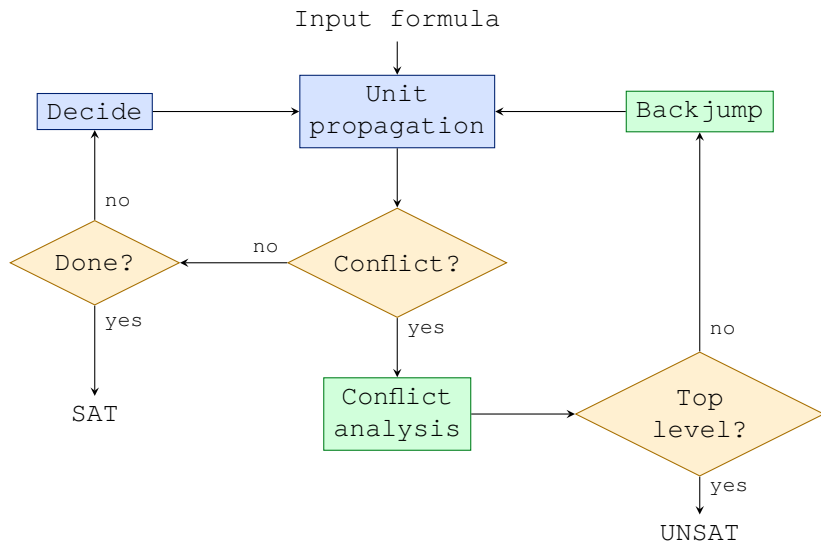
- CDCL
  - Decide quickly
  - Quickly arrive at a conflict (empty clause)
  - Learn from conflicts
- Look-ahead solvers
  - Spend more time with decisions
  - Simplify formula between decision (e.g. eliminate pure literals)
- Cube and Conquer
  - Use look-ahead solver to split into subproblems
  - Solve the subproblems using a CDCL solver
- Model counters and enumerators
- DPLL(T) — Satisfiability modulo theory (SMT)

# Conflict Driven Clause Learning (part 1)

# CDCL Extensions to DPLL

- Non-chronological backtracking (**backjumping**)
- GRASP, Marques-Silva and Sakallah, **1997**
  - Learning new clauses from conflicts
  - Exploiting structure of conflicts during clause learning
- Chaff, Moskewicz et al., **2001**
  - Using lazy data structures for the representation of formulas
  - Branching heuristics must have low computational overhead and must receive feedback from backtrack search
- Periodic restarts (Gomes, Selman, Kautz, et al., **1998**)
- Deletion policies for learnt clauses (BerkMin, Goldberg and Novikov, **2007**)

# CDCL Solver Structure



# Values

Value of a literal  $l \in \text{lit}(\mathbf{x})$  relative to a partial assignment  $\alpha$

$$\text{val}(\alpha, l) = \begin{cases} 1 & l \in \alpha \\ 0 & \neg l \in \alpha \\ * & \text{otherwise} \end{cases}$$

- 1 literal is **defined** and **satisfied** (**true**)
- 0 literal is **defined** and **unsatisfied** (**false**)
- \* literal is **undefined** (also **unassigned**)

# Clause State

State of a clause  $C$  relative to a partial assignment  $\alpha$

**Unsatisfied** all literals in  $C$  are false in  $\alpha$

**Satisfied**  $C$  contains a satisfied literal from  $\alpha$

**Unit**  $C$  is not satisfied and  
exactly one literal in  $C$  is undefined in  $\alpha$

**Unresolved**  $C$  is not satisfied and  
two or more literals in  $C$  are undefined in  $\alpha$

$$\begin{aligned}\alpha &= \{\neg x_1, \neg x_2, x_3\} \\ \varphi &= \overbrace{(x_1 \vee x_2 \vee \neg x_3)}^{\text{unsatisfied}} \wedge \overbrace{(\neg x_2 \vee \neg x_3 \vee x_4)}^{\text{satisfied}} \\ &\quad \wedge \underbrace{(x_2 \vee \neg x_4)}_{\text{unit}} \wedge \underbrace{(x_1 \vee x_4 \vee \neg x_5)}_{\text{unresolved}}\end{aligned}$$

# Variable Types

- Decision variable** the value is set heuristically when picking the next literal for branching
- Decision literal** literal specifying the value of a decision variable
- Implied variable** the value is derived by unit propagation following previous decisions
- Undefined** the value has not been fixed yet (also **unassigned**)

# Decision Level

- Values of literals are put into a **value stack**
  - Removed when backtracking
- **Current decision level** = number of decisions in the value stack
- Each variable  $x_i$  has a **decision level**  $\delta(x_i)$ 
  - $x_i$  is undefined  $\Rightarrow \delta(x_i) = -1$
  - $x_i$  is a  $d$ -th decision variable in the value stack  $\Rightarrow \delta(x_i) = d$
  - $x_i$  is an implied variable  $\Rightarrow \delta(x_i) = \text{number of decision variables before } x_i \text{ in the value stack}$
- 👁  $\delta(x_i) = 0 \Leftrightarrow x_i$  is implied by the input formula
- 👁  $\delta(x_i) > 0$  and  $\text{ant}(x_i) = \text{NIL} \Leftrightarrow x_i$  is a decision variable

$l @ d$  — value of literal  $l$  was set at decision level  $d$



# Assertive Clause







- Assume a CNF  $\varphi$ , a value stack  $\alpha$ , and current decision level  $d$
- Clause  $C$  is **assertive** if
  - $\varphi \models C$  ( $C$  is an **implicate** of  $\varphi$ )
  - $C$  is false under  $\alpha$
  - $C$  has exactly one literal at the current decision level
- Assertion level** of  $C$  is defined as
  - the second highest decision level of literals in  $C$ , or
  - 0 if  $C$  is unit

Assertive clause is unit after backtracking to assertion level

# Assertive Clauses

$$\varphi = (\neg x_{11} \vee \neg x_{31}) \wedge (x_2 \vee x_3) \wedge (x_{31} \vee \neg x_{42} \vee \neg x_{62}) \wedge \\ (x_{31} \vee \neg x_{42} \vee x_{73} \vee x_{83}) \wedge (x_{73} \vee \neg x_{83}) \wedge (\neg x_{73} \vee \neg x_{83})$$

$$\alpha = (x_{11} @ 1, \neg x_{31} @ 1, x_{42} @ 2, x_{62} @ 2, \neg x_{73} @ 3, x_{83} @ 3)$$

Clause	Assertive	
$(\neg x_{11} \vee x_{31} \vee \neg x_{83})$		assertion level 1
$(\neg x_{83})$		assertion level 0
$(\neg x_{11} \vee x_{31} \vee x_{73} \vee \neg x_{83})$		$\neg x_{73} @ 3, x_{83} @ 3$
$(x_{11} \vee x_{31} \vee \neg x_{83})$		satisfied
$(x_2 \vee x_{31} \vee \neg x_{83})$		unit
$(\neg x_{42} \vee x_{73})$		not implicate

---

## Function DPLL+ (CNF $\varphi$ )

---

**Output:** A set of literals (partial model) or UNSAT

```
 $\alpha \leftarrow ()$  // empty value stack  
 $\Gamma \leftarrow \{\}$  // empty set of learned clauses  
 $d \leftarrow 0$  // decision level
```

**while true do**

```
 $(\beta, \psi) \leftarrow \text{UnitProp}(\varphi \wedge \Gamma \wedge \alpha)$ 
```

```
if  $\psi = \perp$  then // backtrack to assertion level
```

```
    if  $d = 0$  then return UNSAT
```

```
     $C \leftarrow$  an assertive clause
```

```
     $d \leftarrow$  assertion level of  $C$ 
```

```
     $\alpha \leftarrow \alpha \setminus \{l@m \mid l \in \alpha \wedge m > d\}$ 
```

```
     $\Gamma \leftarrow \Gamma \cup \{C\}$  // learn clause  $C$ 
```

```
else // contradiction not detected
```

```
     $\alpha \leftarrow \alpha \cup \{l@d \mid l \in \beta \setminus \alpha\}$ 
```

```
    if  $\psi$  is empty then return  $\alpha$ 
```

```
     $l \leftarrow$  a literal in  $\psi$  // new decision literal
```

```
     $d \leftarrow d + 1$  // Increase the decision level
```

```
     $\alpha \leftarrow \alpha \cup \{l@d\}$  // Add decision to the value stack
```

---

# DPLL+ — running example

$$\varphi = (x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$

# DPLL+ — running example

$$\varphi = (x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$

1 Decide  $x_1$  @ 1:

$$(x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\underbrace{\neg x_1 \vee \neg x_3}_{\text{unit clause}}) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$

# DPLL+ — running example

$$\varphi = (x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$

1 Decide  $x_1$  @ 1:

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

unit clause

2 Derive  $\neg x_3$  @ 1 by unit propagation:

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

unit clause                      unit clause

# DPLL+ — running example

$$\varphi = (x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$

1 Decide  $x_1$  @ 1:

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

unit clause

2 Derive  $\neg x_3$  @ 1 by unit propagation:

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

unit clause                      unit clause

3 Derive  $x_2$  @ 1,  $x_4$  @ 1 by unit propagation:

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

empty clause

# DPLL+ — running example

$$\varphi = (x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$

1 Decide  $x_1 @ 1$ :

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

unit clause

2 Derive  $\neg x_3 @ 1$  by unit propagation:

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

unit clause                      unit clause

3 Derive  $x_2 @ 1$ ,  $x_4 @ 1$  by unit propagation:

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

empty clause

4 Learn assertive clause  $\neg x_1$ , backtrack to assertion level 0



# DPLL+ — running example

$$\varphi = (x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$

- 1 Decide  $x_1 @ 1$ :

$$(x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\underbrace{\neg x_1 \vee \neg x_3}_{\text{unit clause}}) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$

- 2 Derive  $\neg x_3 @ 1$  by unit propagation:

$$(x_1 \vee x_3 \vee x_4) \wedge (\underbrace{\neg x_1 \vee x_2 \vee x_3}_{\text{unit clause}}) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (\underbrace{x_3 \vee x_4}_{\text{unit clause}})$$

- 3 Derive  $x_2 @ 1$ ,  $x_4 @ 1$  by unit propagation:

$$(x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\underbrace{\neg x_2 \vee \neg x_4}_{\text{empty clause}}) \wedge (x_3 \vee x_4)$$

- 4 Learn assertive clause  $\neg x_1$ , backtrack to assertion level 0

$$\varphi = (x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$

$$\Gamma = (\neg x_1)$$

# DPLL+ — running example

$$\varphi = (x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$

$$\Gamma = (\neg x_1)$$

# DPLL+ — running example

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

- 1 Derive  $\neg x_1$  @ 0 by unit propagation (not decided like in DPLL!):  
 $(x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4) \wedge (\neg x_1)$

# DPLL+ — running example

$$\varphi = (x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$
$$\Gamma = (\neg x_1)$$

- 1 Derive  $\neg x_1$  @ 0 by unit propagation (not decided like in DPLL!):

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

- 2 Decide  $x_2$  @ 1:

$$(x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\underbrace{\neg x_2 \vee \neg x_4}_{\text{unit clause}}) \wedge (x_3 \vee x_4)$$

# DPLL+ — running example

$$\varphi = (x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$
$$\Gamma = (\neg x_1)$$

- 1 Derive  $\neg x_1$  @ 0 by unit propagation (not decided like in DPLL!):

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

- 2 Decide  $x_2$  @ 1:

$$((x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\underbrace{\neg x_2 \vee \neg x_4}_{\text{unit clause}}) \wedge (x_3 \vee x_4))$$

- 3 Derive  $\neg x_4$  @ 1 by unit propagation:

$$((x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \underbrace{\neg x_4}_{\text{unit clause}}) \wedge (x_3 \vee x_4))$$

# DPLL+ — running example

$$\varphi = (x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$
$$\Gamma = (\neg x_1)$$

- 1 Derive  $\neg x_1$  @ 0 by unit propagation (not decided like in DPLL!):

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

- 2 Decide  $x_2$  @ 1:

$$((x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\underbrace{\neg x_2 \vee \neg x_4}_{\text{unit clause}}) \wedge (x_3 \vee x_4))$$

- 3 Derive  $\neg x_4$  @ 1 by unit propagation:

$$((x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \underbrace{\neg x_4}_{\text{unit clause}}) \wedge (x_3 \vee x_4))$$

- 4 Derive  $x_3$  @ 1 by unit propagation:

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

# DPLL+ — running example

$$\varphi = (x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4)$$
$$\Gamma = (\neg x_1)$$

1 Derive  $\neg x_1$  @ 0 by unit propagation (not decided like in DPLL!):

$$(\textcolor{red}{x_1} \vee x_3 \vee x_4) \wedge (\neg \textcolor{red}{x_1} \vee x_2 \vee x_3) \wedge (\neg \textcolor{red}{x_1} \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_3 \vee x_4) \wedge (\neg \textcolor{red}{x_1})$$

2 Decide  $x_2$  @ 1:

$$(\textcolor{red}{x_1} \vee x_3 \vee x_4) \wedge (\neg \textcolor{red}{x_1} \vee \textcolor{green}{x_2} \vee x_3) \wedge (\neg \textcolor{red}{x_1} \vee \neg x_3) \wedge (\underbrace{\neg \textcolor{red}{x_2} \vee \neg x_4}_{\text{unit clause}}) \wedge (x_3 \vee x_4)$$

3 Derive  $\neg x_4$  @ 1 by unit propagation:

$$(\textcolor{red}{x_1} \vee x_3 \vee \textcolor{red}{x_4}) \wedge (\neg \textcolor{red}{x_1} \vee \textcolor{green}{x_2} \vee x_3) \wedge (\neg \textcolor{red}{x_1} \vee \neg x_3) \wedge (\neg \textcolor{red}{x_2} \vee \neg \textcolor{green}{x_4}) \wedge (\underbrace{x_3 \vee \textcolor{red}{x_4}}_{\text{unit clause}})$$

4 Derive  $x_3$  @ 1 by unit propagation:

$$(\textcolor{red}{x_1} \vee \textcolor{green}{x_3} \vee \textcolor{red}{x_4}) \wedge (\neg \textcolor{red}{x_1} \vee \textcolor{green}{x_2} \vee \textcolor{green}{x_3}) \wedge (\neg \textcolor{red}{x_1} \vee \neg \textcolor{red}{x_3}) \wedge (\neg \textcolor{red}{x_2} \vee \neg \textcolor{green}{x_4}) \wedge (\textcolor{green}{x_3} \vee \textcolor{red}{x_4})$$

$\alpha = \{\neg x_1, x_2, x_3, \neg x_4\}$  is a satisfying assignment of  $\varphi$

# What remains?

- ① How to find assertive clauses?
  - Conflict-driven clauses
  - Resolution based on the implication graph
    - Directed graph defined based on current values of variables and their antecedents
- ② How to backtrack quickly?
  - Lazy data structures in unit propagation
  - Watched literals
- ③ How to manage learned clauses?



# References

# References I



Cook, Stephen A. (1971). “The complexity of theorem-proving procedures.” In: STOC '71: Proceedings of the third annual ACM symposium on Theory of Computing, Shaker Heights, Ohio, United States: ACM, pp. 151–158. doi: <http://doi.acm.org/10.1145/800157.805047>.



Davis, Martin, George Logemann, and Donald Loveland (July 1962). “A machine program for theorem-proving.” In: Commun. ACM 5.7, pp. 394–397. issn: 0001-0782. doi: [10.1145/368273.368557](https://doi.org/10.1145/368273.368557).



Davis, Martin and Hilary Putnam (July 1960). “A Computing Procedure for Quantification Theory.” In: J. ACM 7.3, pp. 201–215. issn: 0004-5411. doi: [10.1145/321033.321034](https://doi.org/10.1145/321033.321034).



Goldberg, Eugene and Yakov Novikov (2007). “BerkMin: A fast and robust SAT-solver.” In: Discrete Applied Mathematics 155.12, pp. 1549–1561.

# References II



Gomes, Carla P, Bart Selman, Henry Kautz, et al. (1998). “Boosting combinatorial search through randomization.” In: AAAI/IAAI 98, pp. 431–437.



Levin, L. A. (1973). “Universal’nye zadachi perebora.” In: Probl. peredachi inform. 9.3, pp. 115–116.



Marques-Silva, João P. and Karem A. Sakallah (1997). “GRASP—a New Search Algorithm for Satisfiability.” In: Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design ICCAD ’96. San Jose, California, USA: IEEE Computer Society, pp. 220–227. ISBN: 0818675977.

# References III



Moskewicz, Matthew W. et al. (2001). “Chaff: Engineering an Efficient SAT Solver.” In: Proceedings of the 38th Annual Design Automation Conference. DAC '01. Las Vegas, Nevada, USA: Association for Computing Machinery, pp. 530–535. ISBN: 1581132972. DOI: 10.1145/378239.379017.