The Power of SAT: Defying NP-Completeness in Practice

Albert Oliveras

Department of Computer Science Seminar
Universitat Politècnica de Catalunya
May 13th 2026

Outline of the Talk

Outline of the Talk

The SAT Problem

(CNF)-SAT Problem: given a CNF formula in propositional logic

$$\begin{aligned}(p \vee \neg q) \quad &\wedge \\ (\neg p \vee q \vee \neg r) \quad &\wedge \\ (\neg p \vee \neg r)\end{aligned}$$

we want to determine whether there is an assignment (i.e. **model**) that satisfies it.

Some terminology:

- variables or negations of them are **literals**,
- disjunctions of literals are **clauses**, and
- a conjuntion of clauses is a **CNF**

The SAT Problem

(CNF)-SAT Problem: given a CNF formula in propositional logic

$$\begin{aligned}(p \vee \neg q) \quad &\wedge \\ (\neg p \vee q \vee \neg r) \quad &\wedge \\ (\neg p \vee \neg r)\end{aligned}$$

we want to determine whether there is an assignment (i.e. **model**) that satisfies it.

Some terminology:

- variables or negations of them are **literals**,
- disjunctions of literals are **clauses**, and
- a conjuntion of clauses is a **CNF**

In this case, a model is $p \to \mathit{True}, \quad q \to \mathit{True}, \quad r \to \mathit{False}$

Canonical NP-complete problem [Coo71]

The SAT problem

How many models does this formula have?

$$
\begin{aligned}
(p \vee \neg q) \quad &\wedge \\
(p \vee q) \quad &\wedge \\
(\neg p \vee \neg r) \quad &\wedge \\
(\neg p \vee r \vee s) \quad &\wedge \\
(r \vee \neg s) \quad &
\end{aligned}
$$

The SAT problem

How many models does this formula have?

$$
\begin{aligned}
(p \vee \neg q) \quad & \wedge \\
(p \vee q) \quad & \wedge \\
(\neg p \vee \neg r) \quad & \wedge \\
(\neg p \vee r \vee s) \quad & \wedge \\
(r \vee \neg s) \quad &
\end{aligned}
$$

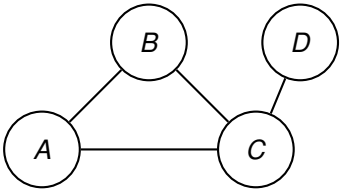
None! This is an unsatisfiable instance.

Convincing someone that this is the case seems more complex.

Indeed, we do not know whether CNF-SAT belongs to co-NP (small witnesses of unsatisfiability).

Using SAT to Encode Problems

Consider the 3-coloring problem on this graph:



We can encode it into SAT with 12 variables:

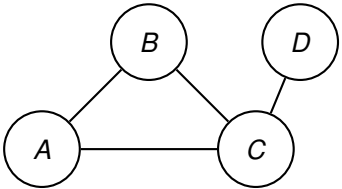
$$x_{N,C} = \text{node } N \text{ gets color } C$$

and clauses enforcing that

- Each node gets at least one color
- Each node gets at most one color
- Adjacent nodes get different color

Using SAT to Encode Problems

Consider the 3-coloring problem on this graph:



We can encode it into SAT with 12 variables:

$$x_{N,C} = \text{node } N \text{ gets color } C$$

and clauses enforcing that

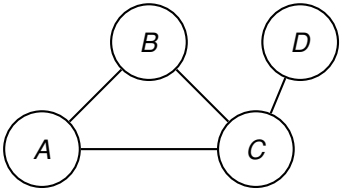
- Each node gets at least one color

$$(x_{A,1} \vee x_{A,2} \vee x_{A,3}) \quad \wedge$$
$$(x_{B,1} \vee x_{B,2} \vee x_{B,3}) \quad \wedge$$
$$(x_{C,1} \vee x_{C,2} \vee x_{C,3})$$

- Each node gets at most one color
- Adjacent nodes get different color

Using SAT to Encode Problems

Consider the 3-coloring problem on this graph:



We can encode it into SAT with 12 variables:

$$x_{N,C} = \text{node } N \text{ gets color } C$$

and clauses enforcing that

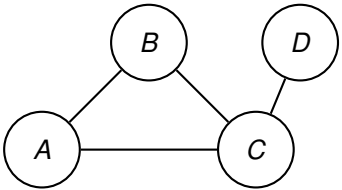
- Each node gets at least one color
- Each node gets at most one color

$$(\neg x_{A,1} \vee \neg x_{A,2}) \wedge (\neg x_{A,1} \vee \neg x_{A,3}) \wedge (\neg x_{A,2} \vee \neg x_{A,3}) \quad \wedge$$
$$(\neg x_{B,1} \vee \neg x_{B,2}) \wedge (\neg x_{B,1} \vee \neg x_{B,3}) \wedge (\neg x_{B,2} \vee \neg x_{B,3}) \quad \wedge$$
$$(\neg x_{C,1} \vee \neg x_{C,2}) \wedge (\neg x_{C,1} \vee \neg x_{C,3}) \wedge (\neg x_{C,2} \vee \neg x_{C,3})$$

- Adjacent nodes get different color

Using SAT to Encode Problems

Consider the 3-coloring problem on this graph:



We can encode it into SAT with 12 variables:

$$x_{N,C} = \text{node } N \text{ gets color } C$$

and clauses enforcing that

- Each node gets at least one color
- Each node gets at most one color
- Adjacent nodes get different color

$$(\neg x_{A,1} \vee \neg x_{B,1}) \wedge (\neg x_{A,2} \vee \neg x_{B,2}) \wedge (\neg x_{A,3} \vee \neg x_{B,3}) \wedge$$
$$(\neg x_{A,1} \vee \neg x_{C,1}) \wedge (\neg x_{A,2} \vee \neg x_{C,2}) \wedge (\neg x_{A,3} \vee \neg x_{C,3}) \wedge$$
$$(\neg x_{B,1} \vee \neg x_{C,1}) \wedge (\neg x_{B,2} \vee \neg x_{C,2}) \wedge (\neg x_{B,3} \vee \neg x_{C,3}) \wedge$$
$$(\neg x_{C,1} \vee \neg x_{D,1}) \wedge (\neg x_{C,2} \vee \neg x_{D,2}) \wedge (\neg x_{C,3} \vee \neg x_{D,3})$$

Some Extensions of SAT

- AllSAT: find all models
- #SAT: count the number of models
- MaxSAT: find an assignment that satisfies the maximum number of clauses
- Pseudo-Boolean solving (aka 0-1 ILP): clauses are replaced by constraints of the form

$$7x + 2\bar{y} + 4z + 2\bar{u} \geq 8$$

  with variables taking values in $\{0, 1\}$, and $\bar{x} := (1 - x)$.

  We might want to additionally optimize a linear function.

- Satisfiability Modulo Theories (SMT): input formula is a CNF with literals being interpreted over some theory. Example:

$$x - y \leq 2 \quad \wedge \quad (x - y \geq 4 \ \vee \ y - z \leq 3) \quad \wedge \quad x - z \geq 6$$

State-of-the-art approaches for some of these problems use SAT solvers as oracles.

Some Extensions of SAT

The CDCL(T) [NOT06] approach to SMT solves the formula as follows:

$$x - y \leq 2 \quad \wedge \quad (x - y \geq 4 \ \vee \ y - z \leq 3) \quad \wedge \quad x - z \geq 6$$

- Let us first abstract every literal of the formula by a fresh propositional variable

$$\underbrace{x - y \leq 2}_{1} \quad \wedge \quad (\underbrace{x - y \geq 4}_{2} \ \vee \ \underbrace{y - z \leq 3}_{3}) \quad \wedge \quad \underbrace{x - z \geq 6}_{4}$$

- The abstracted CNF $\{1,\ 2 \vee 3,\ 4\}$ is given to a SAT solver, returning the model $\{1, 2, 4\}$.
- The model is interpreted back to is concrete semantics, and a T-solver determines that 1 and 2 cannot be simultaneously true. Hence it must hold that $\neg 1 \vee \neg 2$.
- The CNF now becomes $\{1,\ 2 \vee 3,\ 4,\ \neg 1 \vee \neg 2\}$ and a SAT solver gives the model $\{1, \neg 2, 3, 4\}$.
- The T-solver determines that $1, 3, 4$ cannot be simultaneously true.
- The final CNF is $\{1,\ 2 \vee 3,\ 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4\}$ is UNSATISFIABLE.

Some Extensions of SAT

The CDCL(T) [NOT06] approach to SMT solves the formula as follows:

$$x - y \leq 2 \quad \wedge \quad (x - y \geq 4 \ \vee \ y - z \leq 3) \quad \wedge \quad x - z \geq 6$$

- Let us first abstract every literal of the formula by a fresh propositional variable

$$\underbrace{x - y \leq 2}_{1} \quad \wedge \quad (\underbrace{x - y \geq 4}_{2} \ \vee \ \underbrace{y - z \leq 3}_{3}) \quad \wedge \quad \underbrace{x - z \geq 6}_{4}$$

- The abstracted CNF $\{1, \ 2 \vee 3, \ 4\}$ is given to a SAT solver, returning the model $\{1, 2, 4\}$.
- The model is interpreted back to is concrete semantics, and a T-solver determines that 1 and 2 cannot be simultaneously true. Hence it must hold that $\neg 1 \vee \neg 2$.
- The CNF now becomes $\{1, \ 2 \vee 3, \ 4, \ \neg 1 \vee \neg 2\}$ and a SAT solver gives the model $\{1, \neg 2, 3, 4\}$.
- The T-solver determines that $1, 3, 4$ cannot be simultaneously true.
- The final CNF is $\{1, \ 2 \vee 3, \ 4, \ \neg 1 \vee \neg 2, \ \neg 1 \vee \neg 3 \vee \neg 4\}$ is UNSATISFIABLE.

Some Extensions of SAT

The CDCL(T) [NOT06] approach to SMT solves the formula as follows:

$$x - y \leq 2 \quad \wedge \quad (x - y \geq 4 \ \vee \ y - z \leq 3) \quad \wedge \quad x - z \geq 6$$

- Let us first abstract every literal of the formula by a fresh propositional variable

$$\underbrace{x - y \leq 2}_{1} \quad \wedge \quad (\underbrace{x - y \geq 4}_{2} \ \vee \ \underbrace{y - z \leq 3}_{3}) \quad \wedge \quad \underbrace{x - z \geq 6}_{4}$$

- The abstracted CNF $\{1, \ 2 \vee 3, \ 4\}$ is given to a SAT solver, returning the model $\{1, 2, 4\}$.
- The model is interpreted back to is concrete semantics, and a T-solver determines that 1 and 2 cannot be simultaneously true. Hence it must hold that $\neg 1 \vee \neg 2$.
- The CNF now becomes $\{1, \ 2 \vee 3, \ 4, \ \neg 1 \vee \neg 2\}$ and a SAT solver gives the model $\{1, \neg 2, 3, 4\}$.
- The T-solver determines that $1, 3, 4$ cannot be simultaneously true.
- The final CNF is $\{1, \ 2 \vee 3, \ 4, \ \neg 1 \vee \neg 2, \ \neg 1 \vee \neg 3 \vee \neg 4\}$ is UNSATISFIABLE.

Some Extensions of SAT

The CDCL(T) [NOT06] approach to SMT solves the formula as follows:

$$x - y \leq 2 \quad \wedge \quad (x - y \geq 4 \ \vee \ y - z \leq 3) \quad \wedge \quad x - z \geq 6$$

- Let us first abstract every literal of the formula by a fresh propositional variable

$$\underbrace{x - y \leq 2}_{1} \quad \wedge \quad (\underbrace{x - y \geq 4}_{2} \ \vee \ \underbrace{y - z \leq 3}_{3}) \quad \wedge \quad \underbrace{x - z \geq 6}_{4}$$

- The abstracted CNF $\{1, \ 2 \vee 3, \ 4\}$ is given to a SAT solver, returning the model $\{1, 2, 4\}$.
- The model is interpreted back to is concrete semantics, and a T-solver determines that 1 and 2 cannot be simultaneously true. Hence it must hold that $\neg 1 \vee \neg 2$.
- The CNF now becomes $\{1, \ 2 \vee 3, \ 4, \ \neg 1 \vee \neg 2\}$ and a SAT solver gives the model $\{1, \neg 2, 3, 4\}$.
- The T-solver determines that $1, 3, 4$ cannot be simultaneously true.
- The final CNF is $\{1, \ 2 \vee 3, \ 4, \ \neg 1 \vee \neg 2, \ \neg 1 \vee \neg 3 \vee \neg 4\}$ is UNSATISFIABLE.

Some Extensions of SAT

The CDCL(T) [NOT06] approach to SMT solves the formula as follows:

$$x - y \leq 2 \quad \wedge \quad (x - y \geq 4 \ \vee \ y - z \leq 3) \quad \wedge \quad x - z \geq 6$$

- Let us first abstract every literal of the formula by a fresh propositional variable

$$\underbrace{x - y \leq 2}_{1} \quad \wedge \quad (\underbrace{x - y \geq 4}_{2} \ \vee \ \underbrace{y - z \leq 3}_{3}) \quad \wedge \quad \underbrace{x - z \geq 6}_{4}$$

- The abstracted CNF $\{1, \ 2 \vee 3, \ 4\}$ is given to a SAT solver, returning the model $\{1, 2, 4\}$.
- The model is interpreted back to is concrete semantics, and a T-solver determines that 1 and 2 cannot be simultaneously true. Hence it must hold that $\neg 1 \vee \neg 2$.
- The CNF now becomes $\{1, \ 2 \vee 3, \ 4, \ \neg 1 \vee \neg 2\}$ and a SAT solver gives the model $\{1, \neg 2, 3, 4\}$.
- The T-solver determines that $1, 3, 4$ cannot be simultaneously true.
- The final CNF is $\{1, \ 2 \vee 3, \ 4, \ \neg 1 \vee \neg 2, \ \neg 1 \vee \neg 3 \vee \neg 4\}$ is UNSATISFIABLE.

Some Extensions of SAT

The CDCL(T) [NOT06] approach to SMT solves the formula as follows:

$$x - y \leq 2 \quad \wedge \quad (x - y \geq 4 \ \vee \ y - z \leq 3) \quad \wedge \quad x - z \geq 6$$

- Let us first abstract every literal of the formula by a fresh propositional variable

$$\underbrace{x - y \leq 2}_{1} \quad \wedge \quad (\underbrace{x - y \geq 4}_{2} \ \vee \ \underbrace{y - z \leq 3}_{3}) \quad \wedge \quad \underbrace{x - z \geq 6}_{4}$$

- The abstracted CNF $\{1, \ 2 \vee 3, \ 4\}$ is given to a SAT solver, returning the model $\{1, 2, 4\}$.
- The model is interpreted back to is concrete semantics, and a T-solver determines that 1 and 2 cannot be simultaneously true. Hence it must hold that $\neg 1 \vee \neg 2$.
- The CNF now becomes $\{1, \ 2 \vee 3, \ 4, \ \neg 1 \vee \neg 2\}$ and a SAT solver gives the model $\{1, \neg 2, 3, 4\}$.
- The T-solver determines that $1, 3, 4$ cannot be simultaneously true.
- The final CNF is $\{1, \ 2 \vee 3, \ 4, \ \neg 1 \vee \neg 2, \ \neg 1 \vee \neg 3 \vee \neg 4\}$ is UNSATISFIABLE.

Some Extensions of SAT

The CDCL(T) [NOT06] approach to SMT solves the formula as follows:

$$x - y \leq 2 \quad \wedge \quad (x - y \geq 4 \ \vee \ y - z \leq 3) \quad \wedge \quad x - z \geq 6$$

- Let us first abstract every literal of the formula by a fresh propositional variable

$$\underbrace{x - y \leq 2}_{1} \quad \wedge \quad (\underbrace{x - y \geq 4}_{2} \ \vee \ \underbrace{y - z \leq 3}_{3}) \quad \wedge \quad \underbrace{x - z \geq 6}_{4}$$

- The abstracted CNF $\{1, \ 2 \vee 3, \ 4\}$ is given to a SAT solver, returning the model $\{1, 2, 4\}$.
- The model is interpreted back to is concrete semantics, and a T-solver determines that 1 and 2 cannot be simultaneously true. Hence it must hold that $\neg 1 \vee \neg 2$.
- The CNF now becomes $\{1, \ 2 \vee 3, \ 4, \ \neg 1 \vee \neg 2\}$ and a SAT solver gives the model $\{1, \neg 2, 3, 4\}$.
- The T-solver determines that $1, 3, 4$ cannot be simultaneously true.
- The final CNF is $\{1, \ \ 2 \vee 3, \ \ 4, \ \ \neg 1 \vee \neg 2, \ \ \neg 1 \vee \neg 3 \vee \neg 4\}$ is UNSATISFIABLE.

The T-solver in this is case only needs to find negative cycles in a directed graph – Bellman-Ford.

Real-world Applications

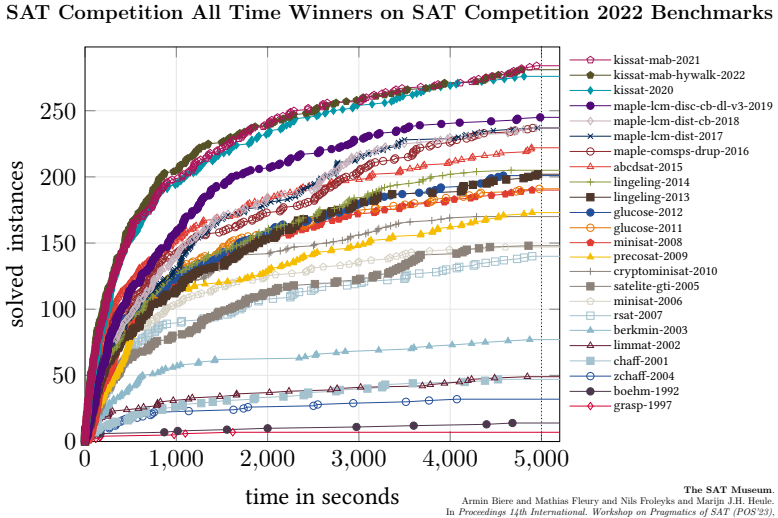
Industrial Applications:

- Amazon Research: Formal verification
- Microsoft Research: Formal verification
- Intel/NVIDIA: Formal verification
- Google: Optimization problems
- Anaconda Python Distribution: Finding the set of compatible package versions

Applications in Mathematics:

- Boolean Pythagorean Triples Conjecture
  - Impact on media
- SAT4Math repository

State-of-the-art Tools - Evolution



https://cca.informatik.uni-freiburg.de/satmuseum

The SAT Museum.
Armin Biere and Mathias Fleury and Nils Froleyks and Marijn J.H. Heule.
In *Proceedings 14th International. Workshop on Pragmatics of SAT (POS'23)*,
vol. 3545, CEUR Workshop Proceedings, pages 72-87, CEUR-WS.org 2023.
[paper - bibtex - data - zenodo - ceur - workshop - proceedings]

State-of-the-art Tools – 2026

Currently, the two most competitive SAT solvers are developed by the same team (led by Armin Biere, Univ. Freiburg):

- CaDiCaL:
  - Focus on usability: API, incrementality
  - Readable code in C++ ($\sim$70K loc)
- Kissat:
  - Focus on performance
  - Less readable code in C ($\sim$40K loc)

State-of-the-art Tools - How is this possible?

- Given a problem with $\sim$40K variables, we have $2^{40,000}$ assignments to check, which is around $1.5 \cdot 10^{12,041}$.
- If we evenly distribute the assignments among the world population ($8 \cdot 10^9$), each person gets around $2 \cdot 10^{12,031}$ assignments.
- If every person has a computer that can check 10^9 assignments per second, it takes $2 \cdot 10^{12,022}$ seconds, i.e., $6 \cdot 10^{12,012}$ centuries.
- The longest-lived stars in the universe will only live for 10^{11} centuries.

Key for efficienty: SAT solvers look both for solutions and proofs of unsatisfiability

Key for reliabilty: KISSAT and CADICAL provide unsatisfiability certificates that can later be checked by **certified** verifiers built in Lean, Coq, Isabelle, ACL2, etc.

Outline of the Talk

The DPLL Algorithm

Given CNF F and an assignment α we will denote by $F|_\alpha$ the result of:

- Removing all **clauses** of F that are **true** in α
- Removing all **literals** in F that are **false** in α

The starting point of most state-of-the-art SAT solvers is the classical DPLL algorithm [DP60, DLL62]:

Function `DPLL` (F: CNF, α: Assignment) **:**

if $F|_\alpha$ contains an empty clause **then return** *False*

if $F|_\alpha$ is empty **then return** *True*

$\ell :=$ *choose_literal*$(F|_\alpha)$

return *DPLL*$(F, \alpha \cup \{\ell\})$ **or** *DPLL*$(F, \alpha \cup \{\neg\ell\})$

The DPLL Algorithm

Given CNF F and an assignment α we will denote by $F|_\alpha$ the result of:

- Removing all **clauses** of F that are **true** in α
- Removing all **literals** in F that are **false** in α

The starting point of most state-of-the-art SAT solvers is the classical DPLL algorithm [DP60, DLL62]:

Function `DPLL`$(F$*:* CNF$, \alpha$*:* Assignment$)$**:**

while $F|_\alpha$ contains lit ℓ that only occurs with this polarity **do** $\alpha := \alpha \cup \{\ell\}$ `// pure lit.`

if $F|_\alpha$ contains an empty clause **then return** *False*

if $F|_\alpha$ is empty **then return** *True*

$\ell :=$ *choose_literal*$(F|_\alpha)$

return *DPLL*$(F, \alpha \cup \{\ell\})$ **or** *DPLL*$(F, \alpha \cup \{\neg\ell\})$

The DPLL Algorithm

Given CNF F and an assignment α we will denote by $F|_\alpha$ the result of:

- Removing all **clauses** of F that are **true** in α
- Removing all **literals** in F that are **false** in α

The starting point of most state-of-the-art SAT solvers is the classical DPLL algorithm [DP60, DLL62]:

Function `DPLL` (F: CNF, α: Assignment) **:**

while $F|_\alpha$ contains lit ℓ that only occurs with this polarity **do** $\alpha := \alpha \cup \{\ell\}$ `// pure lit.`

while $F|_\alpha$ contains a unit clause ℓ **do** $\alpha := \alpha \cup \{\ell\}$ `// unit propagation`

if $F|_\alpha$ contains an empty clause **then return** *False*

if $F|_\alpha$ is empty **then return** *True*

$\ell :=$ *choose_literal*$(F|_\alpha)$

return *DPLL*$(F, \alpha \cup \{\ell\})$ **or** *DPLL*$(F, \alpha \cup \{\neg\ell\})$

Moving from DPLL to CDCL

State-of-the-art solvers do not implement DPLL, but a more involved algorithm called CDCL (Conflict-Driven Clause Learning) [MLM21]:

- The pure literal rule is not usually implemented in CDCL (not worth it)
- The unit propagation rule is a critical part of CDCL. Extremely simple rule:

  Given $x \vee \overline{y} \vee \overline{z} \vee u$ and $\alpha = \{\overline{x}, z, \overline{u}\}$, we can propagate $\overline{y}$

  This is executed until fixpoint
- When unit propagation cannot be applied, a literal is chosen to be added to α, using dedicated heuristics [BF15]
- The main difference is in the backtracking step

CDCL - Unit Propagation

1. $p_6 \vee \overline{p}_7$
2. $p_6 \vee p_{13} \vee p_8$
3. $p_8 \vee \overline{p}_4 \vee p_{15}$
4. $\overline{p}_{11} \vee p_6 \vee \overline{p}_{12}$
5. $\overline{p}_{11} \vee p_{13} \vee p_5$
6. $p_{12} \vee \overline{p}_5 \vee \overline{p}_2$
7. $p_2 \vee \overline{p}_4 \vee \overline{p}_{10}$
8. $p_{10} \vee \overline{p}_8 \vee p_1$
9. $p_{10} \vee p_3$
10. $\overline{p}_1 \vee \overline{p}_3 \vee p_7 \vee p_{14}$
11. $\overline{p}_3 \vee \overline{p}_{15} \vee \overline{p}_{14}$
12. $\overline{p}_{16} \vee \overline{p}_{17}$

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Value																	
Reason																	

CDCL - Unit Propagation

1. $p_6 \vee \overline{p}_7$
2. $p_6 \vee p_{13} \vee p_8$
3. $p_8 \vee \overline{p}_4 \vee p_{15}$
4. $\overline{p}_{11} \vee p_6 \vee \overline{p}_{12}$
5. $\overline{p}_{11} \vee p_{13} \vee p_5$
6. $p_{12} \vee \overline{p}_5 \vee \overline{p}_2$
7. $p_2 \vee \overline{p}_4 \vee \overline{p}_{10}$
8. $p_{10} \vee \overline{p}_8 \vee p_1$
9. $p_{10} \vee p_3$
10. $\overline{p}_1 \vee \overline{p}_3 \vee p_7 \vee p_{14}$
11. $\overline{p}_3 \vee \overline{p}_{15} \vee \overline{p}_{14}$
12. $\overline{p}_{16} \vee \overline{p}_{17}$

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Value						F											
Reason						d											

$\overline{p}_6^d$

CDCL - Unit Propagation

1. $p_6 \vee \overline{p}_7$
2. $p_6 \vee p_{13} \vee p_8$
3. $p_8 \vee \overline{p}_4 \vee p_{15}$
4. $\overline{p}_{11} \vee p_6 \vee \overline{p}_{12}$
5. $\overline{p}_{11} \vee p_{13} \vee p_5$
6. $p_{12} \vee \overline{p}_5 \vee \overline{p}_2$
7. $p_2 \vee \overline{p}_4 \vee \overline{p}_{10}$
8. $p_{10} \vee \overline{p}_8 \vee p_1$
9. $p_{10} \vee p_3$
10. $\overline{p}_1 \vee \overline{p}_3 \vee p_7 \vee p_{14}$
11. $\overline{p}_3 \vee \overline{p}_{15} \vee \overline{p}_{14}$
12. $\overline{p}_{16} \vee \overline{p}_{17}$

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Value						F	F										
Reason						d	1										

$\overline{p}_6^d \ \overline{p}_7$

CDCL - Unit Propagation

1. $p_6 \vee \overline{p}_7$
2. $p_6 \vee p_{13} \vee p_8$
3. $p_8 \vee \overline{p}_4 \vee p_{15}$
4. $\overline{p}_{11} \vee p_6 \vee \overline{p}_{12}$
5. $\overline{p}_{11} \vee p_{13} \vee p_5$
6. $p_{12} \vee \overline{p}_5 \vee \overline{p}_2$
7. $p_2 \vee \overline{p}_4 \vee \overline{p}_{10}$
8. $p_{10} \vee \overline{p}_8 \vee p_1$
9. $p_{10} \vee p_3$
10. $\overline{p}_1 \vee \overline{p}_3 \vee p_7 \vee p_{14}$
11. $\overline{p}_3 \vee \overline{p}_{15} \vee \overline{p}_{14}$
12. $\overline{p}_{16} \vee \overline{p}_{17}$

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Value						F	F						F				
Reason						d	1						d				

$\overline{p}_6^d \ \overline{p}_7 \ \overline{p}_{13}^d$

CDCL - Unit Propagation

1. $p_6 \vee \overline{p}_7$
2. $p_6 \vee p_{13} \vee p_8$
3. $p_8 \vee \overline{p}_4 \vee p_{15}$
4. $\overline{p}_{11} \vee p_6 \vee \overline{p}_{12}$
5. $\overline{p}_{11} \vee p_{13} \vee p_5$
6. $p_{12} \vee \overline{p}_5 \vee \overline{p}_2$
7. $p_2 \vee \overline{p}_4 \vee \overline{p}_{10}$
8. $p_{10} \vee \overline{p}_8 \vee p_1$
9. $p_{10} \vee p_3$
10. $\overline{p}_1 \vee \overline{p}_3 \vee p_7 \vee p_{14}$
11. $\overline{p}_3 \vee \overline{p}_{15} \vee \overline{p}_{14}$
12. $\overline{p}_{16} \vee \overline{p}_{17}$

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Value						F	F	T					F				
Reason						d	1	2					d				

$\overline{p}_6^d \ \overline{p}_7 \ \overline{p}_{13}^d \ p_8$

CDCL - Unit Propagation

1. $p_6 \vee \overline{p}_7$
2. $p_6 \vee p_{13} \vee p_8$
3. $p_8 \vee \overline{p}_4 \vee p_{15}$
4. $\overline{p}_{11} \vee p_6 \vee \overline{p}_{12}$
5. $\overline{p}_{11} \vee p_{13} \vee p_5$
6. $p_{12} \vee \overline{p}_5 \vee \overline{p}_2$
7. $p_2 \vee \overline{p}_4 \vee \overline{p}_{10}$
8. $p_{10} \vee \overline{p}_8 \vee p_1$
9. $p_{10} \vee p_3$
10. $\overline{p}_1 \vee \overline{p}_3 \vee p_7 \vee p_{14}$
11. $\overline{p}_3 \vee \overline{p}_{15} \vee \overline{p}_{14}$
12. $\overline{p}_{16} \vee \overline{p}_{17}$

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Value				T		F	F	T					F				
Reason				d		d	1	2					d				

$\overline{p}_6^{\,d}\ \overline{p}_7\ \overline{p}_{13}^{\,d}\ p_8\ {p_4}^d$

CDCL - Unit Propagation

1. $p_6 \vee \overline{p}_7$
2. $p_6 \vee p_{13} \vee p_8$
3. $\overline{p}_8 \vee \overline{p}_4 \vee p_{15}$
4. $\overline{p}_{11} \vee p_6 \vee \overline{p}_{12}$
5. $\overline{p}_{11} \vee p_{13} \vee p_5$
6. $p_{12} \vee \overline{p}_5 \vee \overline{p}_2$
7. $p_2 \vee \overline{p}_4 \vee \overline{p}_{10}$
8. $p_{10} \vee \overline{p}_8 \vee p_1$
9. $p_{10} \vee p_3$
10. $\overline{p}_1 \vee \overline{p}_3 \vee p_7 \vee p_{14}$
11. $\overline{p}_3 \vee \overline{p}_{15} \vee \overline{p}_{14}$
12. $\overline{p}_{16} \vee \overline{p}_{17}$

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Value				T		F	F	T					F		T		
Reason				d		d	1	2					d		3		

$\overline{p}_6^{\,d}\ \overline{p}_7\ \overline{p}_{13}^{\,d}\ p_8\ p_4{}^{d}\ p_{15}$

CDCL - Unit Propagation

1. $p_6 \vee \overline{p}_7$
2. $p_6 \vee p_{13} \vee p_8$
3. $p_8 \vee \overline{p}_4 \vee p_{15}$
4. $\overline{p}_{11} \vee p_6 \vee \overline{p}_{12}$
5. $\overline{p}_{11} \vee p_{13} \vee p_5$
6. $p_{12} \vee \overline{p}_5 \vee \overline{p}_2$
7. $p_2 \vee \overline{p}_4 \vee \overline{p}_{10}$
8. $p_{10} \vee \overline{p}_8 \vee p_1$
9. $p_{10} \vee p_3$
10. $\overline{p}_1 \vee \overline{p}_3 \vee p_7 \vee p_{14}$
11. $\overline{p}_3 \vee \overline{p}_{15} \vee \overline{p}_{14}$
12. $\overline{p}_{16} \vee \overline{p}_{17}$

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Value				T		F	F	T					F		T	T	
Reason				d		d	1	2					d		3	d	

$\overline{p}_6^{\,d}\ \overline{p}_7\ \overline{p}_{13}^{\,d}\ p_8\ p_4{}^d\ p_{15}\ p_{16}{}^d$

CDCL - Unit Propagation

1. $p_6 \vee \overline{p}_7$
2. $p_6 \vee p_{13} \vee p_8$
3. $p_8 \vee \overline{p}_4 \vee p_{15}$
4. $\overline{p}_{11} \vee p_6 \vee \overline{p}_{12}$
5. $\overline{p}_{11} \vee p_{13} \vee p_5$
6. $p_{12} \vee \overline{p}_5 \vee \overline{p}_2$
7. $p_2 \vee \overline{p}_4 \vee \overline{p}_{10}$
8. $p_{10} \vee \overline{p}_8 \vee p_1$
9. $p_{10} \vee p_3$
10. $\overline{p}_1 \vee \overline{p}_3 \vee p_7 \vee p_{14}$
11. $\overline{p}_3 \vee \overline{p}_{15} \vee \overline{p}_{14}$
12. $\overline{p}_{16} \vee \overline{p}_{17}$

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Value				T		F	F	T					F		T	T	F
Reason				d		d	1	2					d		3	d	12

$\overline{p}_6^{\,d}\ \overline{p}_7\ \overline{p}_{13}^{\,d}\ p_8\ p_4{}^d\ p_{15}\ p_{16}{}^d\ \overline{p}_{17}$

CDCL - Unit Propagation

1. $p_6 \vee \overline{p}_7$
2. $p_6 \vee p_{13} \vee p_8$
3. $p_8 \vee \overline{p}_4 \vee p_{15}$
4. $\overline{p}_{11} \vee p_6 \vee \overline{p}_{12}$
5. $\overline{p}_{11} \vee p_{13} \vee p_5$
6. $p_{12} \vee \overline{p}_5 \vee \overline{p}_2$
7. $p_2 \vee \overline{p}_4 \vee \overline{p}_{10}$
8. $p_{10} \vee \overline{p}_8 \vee p_1$
9. $p_{10} \vee p_3$
10. $\overline{p}_1 \vee \overline{p}_3 \vee p_7 \vee p_{14}$
11. $\overline{p}_3 \vee \overline{p}_{15} \vee \overline{p}_{14}$
12. $\overline{p}_{16} \vee \overline{p}_{17}$

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Value				T		F	F	T			T		F		T	T	F
Reason				d		d	1	2			d		d		3	d	12

$\overline{p}_6^{\,d}\ \overline{p}_7\ \overline{p}_{13}^{\,d}\ p_8\ p_4{}^d\ p_{15}\ p_{16}{}^d\ \overline{p}_{17}\ p_{11}{}^d$

CDCL - Unit Propagation

1. $p_6 \vee \overline{p}_7$
2. $p_6 \vee p_{13} \vee p_8$
3. $p_8 \vee \overline{p}_4 \vee p_{15}$
4. $\overline{p}_{11} \vee p_6 \vee \overline{p}_{12}$
5. $\overline{p}_{11} \vee p_{13} \vee p_5$
6. $p_{12} \vee \overline{p}_5 \vee \overline{p}_2$
7. $p_2 \vee \overline{p}_4 \vee \overline{p}_{10}$
8. $p_{10} \vee \overline{p}_8 \vee p_1$
9. $p_{10} \vee p_3$
10. $\overline{p}_1 \vee \overline{p}_3 \vee p_7 \vee p_{14}$
11. $\overline{p}_3 \vee \overline{p}_{15} \vee \overline{p}_{14}$
12. $\overline{p}_{16} \vee \overline{p}_{17}$

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Value				T		F	F	T			T	F	F		T	T	F
Reason				d		d	1	2			d	4	d		3	d	12

$\overline{p}_6^{\,d}\ \overline{p}_7\ \overline{p}_{13}^{\,d}\ p_8\ {p_4}^d\ p_{15}\ {p_{16}}^d\ \overline{p}_{17}\ {p_{11}}^d\ \overline{p}_{12}$

CDCL - Unit Propagation

1. $p_6 \vee \overline{p}_7$
2. $p_6 \vee p_{13} \vee p_8$
3. $p_8 \vee \overline{p}_4 \vee p_{15}$
4. $\overline{p}_{11} \vee p_6 \vee \overline{p}_{12}$
5. $\overline{p}_{11} \vee p_{13} \vee p_5$
6. $p_{12} \vee \overline{p}_5 \vee \overline{p}_2$
7. $p_2 \vee \overline{p}_4 \vee \overline{p}_{10}$
8. $p_{10} \vee \overline{p}_8 \vee p_1$
9. $p_{10} \vee p_3$
10. $\overline{p}_1 \vee \overline{p}_3 \vee p_7 \vee p_{14}$
11. $\overline{p}_3 \vee \overline{p}_{15} \vee \overline{p}_{14}$
12. $\overline{p}_{16} \vee \overline{p}_{17}$

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Value				T	T	F	F	T			T	F	F		T	T	F
Reason				d	5	d	1	2			d	4	d		3	d	12

$\overline{p}_6^{\,d}\ \overline{p}_7\ \overline{p}_{13}^{\,d}\ p_8\ p_4{}^d\ p_{15}\ p_{16}{}^d\ \overline{p}_{17}\ p_{11}{}^d\ \overline{p}_{12}\ p_5$

CDCL - Unit Propagation

1. $p_6 \vee \overline{p}_7$
2. $p_6 \vee p_{13} \vee p_8$
3. $p_8 \vee \overline{p}_4 \vee p_{15}$
4. $\overline{p}_{11} \vee p_6 \vee \overline{p}_{12}$
5. $\overline{p}_{11} \vee p_{13} \vee p_5$
6. $p_{12} \vee \overline{p}_5 \vee \overline{p}_2$
7. $p_2 \vee \overline{p}_4 \vee \overline{p}_{10}$
8. $p_{10} \vee \overline{p}_8 \vee p_1$
9. $p_{10} \vee p_3$
10. $\overline{p}_1 \vee \overline{p}_3 \vee p_7 \vee p_{14}$
11. $\overline{p}_3 \vee \overline{p}_{15} \vee \overline{p}_{14}$
12. $\overline{p}_{16} \vee \overline{p}_{17}$

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Value		F		T	T	F	F	T			T	F	F		T	T	F
Reason		6		d	5	d	1	2			d	4	d		3	d	12

$\overline{p}_6^{\,d}\ \overline{p}_7\ \overline{p}_{13}^{\,d}\ p_8\ p_4{}^d\ p_{15}\ p_{16}{}^d\ \overline{p}_{17}\ p_{11}{}^d\ \overline{p}_{12}\ p_5\ \overline{p}_2$

CDCL - Unit Propagation

1. $p_6 \vee \overline{p}_7$
2. $p_6 \vee p_{13} \vee p_8$
3. $p_8 \vee \overline{p}_4 \vee p_{15}$
4. $\overline{p}_{11} \vee p_6 \vee \overline{p}_{12}$
5. $\overline{p}_{11} \vee p_{13} \vee p_5$
6. $p_{12} \vee \overline{p}_5 \vee \overline{p}_2$
7. $p_2 \vee \overline{p}_4 \vee \overline{p}_{10}$
8. $p_{10} \vee \overline{p}_8 \vee p_1$
9. $p_{10} \vee p_3$
10. $\overline{p}_1 \vee \overline{p}_3 \vee p_7 \vee p_{14}$
11. $\overline{p}_3 \vee \overline{p}_{15} \vee \overline{p}_{14}$
12. $\overline{p}_{16} \vee \overline{p}_{17}$

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Value		F		T	T	F	F	T		F	T	F	F		T	T	F
Reason		6		d	5	d	1	2		7	d	4	d		3	d	12

$\overline{p}_6^{\,d}\ \overline{p}_7\ \overline{p}_{13}^{\,d}\ p_8\ p_4{}^d\ p_{15}\ p_{16}{}^d\ \overline{p}_{17}\ p_{11}{}^d\ \overline{p}_{12}\ p_5\ \overline{p}_2\ \overline{p}_{10}$

CDCL - Unit Propagation

1. $p_6 \vee \overline{p}_7$
2. $p_6 \vee p_{13} \vee p_8$
3. $p_8 \vee \overline{p}_4 \vee p_{15}$
4. $\overline{p}_{11} \vee p_6 \vee \overline{p}_{12}$
5. $\overline{p}_{11} \vee p_{13} \vee p_5$
6. $p_{12} \vee \overline{p}_5 \vee \overline{p}_2$
7. $p_2 \vee \overline{p}_4 \vee \overline{p}_{10}$
8. $p_{10} \vee \overline{p}_8 \vee p_1$
9. $p_{10} \vee p_3$
10. $\overline{p}_1 \vee \overline{p}_3 \vee p_7 \vee p_{14}$
11. $\overline{p}_3 \vee \overline{p}_{15} \vee \overline{p}_{14}$
12. $\overline{p}_{16} \vee \overline{p}_{17}$

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Value	T	F		T	T	F	F	T		F	T	F	F		T	T	F
Reason	8	6		d	5	d	1	2		7	d	4	d		3	d	12

$\overline{p}_6^{\,d}\ \overline{p}_7\ \overline{p}_{13}^{\,d}\ p_8\ p_4{}^d\ p_{15}\ p_{16}{}^d\ \overline{p}_{17}\ p_{11}{}^d\ \overline{p}_{12}\ p_5\ \overline{p}_2\ \overline{p}_{10}\ p_1$

CDCL - Unit Propagation

1. $p_6 \vee \overline{p}_7$
2. $p_6 \vee p_{13} \vee p_8$
3. $p_8 \vee \overline{p}_4 \vee p_{15}$
4. $\overline{p}_{11} \vee p_6 \vee \overline{p}_{12}$
5. $\overline{p}_{11} \vee p_{13} \vee p_5$
6. $p_{12} \vee \overline{p}_5 \vee \overline{p}_2$
7. $p_2 \vee \overline{p}_4 \vee \overline{p}_{10}$
8. $p_{10} \vee \overline{p}_8 \vee p_1$
9. $p_{10} \vee p_3$
10. $\overline{p}_1 \vee \overline{p}_3 \vee p_7 \vee p_{14}$
11. $\overline{p}_3 \vee \overline{p}_{15} \vee \overline{p}_{14}$
12. $\overline{p}_{16} \vee \overline{p}_{17}$

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Value	T	F	T	T	T	F	F	T		F	T	F	F		T	T	F
Reason	8	6	9	d	5	d	1	2		7	d	4	d		3	d	12

$\overline{p}_6^{\,d}\ \overline{p}_7\ \overline{p}_{13}^{\,d}\ p_8\ p_4{}^d\ p_{15}\ p_{16}{}^d\ \overline{p}_{17}\ p_{11}{}^d\ \overline{p}_{12}\ p_5\ \overline{p}_2\ \overline{p}_{10}\ p_1\ p_3$

CDCL - Unit Propagation

1. $p_6 \vee \overline{p}_7$
2. $p_6 \vee p_{13} \vee p_8$
3. $p_8 \vee \overline{p}_4 \vee p_{15}$
4. $\overline{p}_{11} \vee p_6 \vee \overline{p}_{12}$
5. $\overline{p}_{11} \vee p_{13} \vee p_5$
6. $p_{12} \vee \overline{p}_5 \vee \overline{p}_2$
7. $p_2 \vee \overline{p}_4 \vee \overline{p}_{10}$
8. $p_{10} \vee \overline{p}_8 \vee p_1$
9. $p_{10} \vee p_3$
10. $\overline{p}_1 \vee \overline{p}_3 \vee p_7 \vee p_{14}$
11. $\overline{p}_3 \vee \overline{p}_{15} \vee \overline{p}_{14}$
12. $\overline{p}_{16} \vee \overline{p}_{17}$

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Value	T	F	T	T	T	F	F	T		F	T	F	F	T	T	T	F
Reason	8	6	9	d	5	d	1	2		7	d	4	d	10	3	d	12

$\overline{p}_6^{\,d}\ \overline{p}_7\ \overline{p}_{13}^{\,d}\ p_8\ p_4{}^d\ p_{15}\ p_{16}{}^d\ \overline{p}_{17}\ p_{11}{}^d\ \overline{p}_{12}\ p_5\ \overline{p}_2\ \overline{p}_{10}\ p_1\ p_3\ p_{14}$

CDCL - Unit Propagation

1. $p_6 \vee \overline{p}_7$
2. $p_6 \vee p_{13} \vee p_8$
3. $p_8 \vee \overline{p}_4 \vee p_{15}$
4. $\overline{p}_{11} \vee p_6 \vee \overline{p}_{12}$
5. $\overline{p}_{11} \vee p_{13} \vee p_5$
6. $p_{12} \vee \overline{p}_5 \vee \overline{p}_2$
7. $p_2 \vee \overline{p}_4 \vee \overline{p}_{10}$
8. $p_{10} \vee \overline{p}_8 \vee p_1$
9. $p_{10} \vee p_3$
10. $\overline{p}_1 \vee \overline{p}_3 \vee p_7 \vee p_{14}$
11. $\overline{p}_3 \vee \overline{p}_{15} \vee \overline{p}_{14}$
12. $\overline{p}_{16} \vee \overline{p}_{17}$

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Value	T	F	T	T	T	F	F	T		F	T	F	F	T	T	T	F
Reason	8	6	9	d	5	d	1	2		7	d	4	d	10	3	d	12

$\overline{p}_6^{\,d}\ \overline{p}_7\ \overline{p}_{13}^{\,d}\ p_8\ p_4{}^d\ p_{15}\ p_{16}{}^d\ \overline{p}_{17}\ p_{11}{}^d\ \overline{p}_{12}\ p_5\ \overline{p}_2\ \overline{p}_{10}\ p_1\ p_3\ p_{14}$

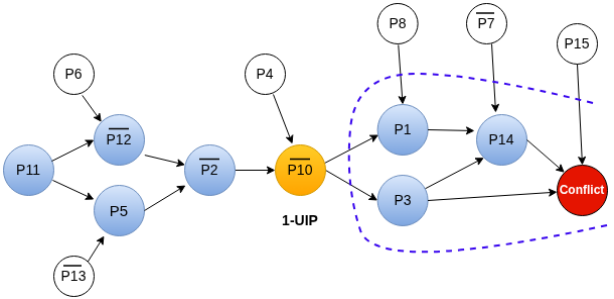
But now clause 11 is conflicting. Standard **backtracking** would move to

$\overline{p}_6^{\,d}\ \overline{p}_7\ \overline{p}_{13}^{\,d}\ p_8\ p_4{}^d\ p_{15}\ p_{16}{}^d\ \overline{p}_{17}\ \overline{p}_{11}$

CDCL Execution - Conflict Analysis

Assignment stack: $\overline{p}_6^{\,d}\ \overline{p}_7\ \overline{p}_{13}^{\,d}\ p_8\ p_4{}^{d}\ p_{15}\ p_{16}{}^{d}\ \overline{p}_{17}\ p_{11}{}^{d}\ \overline{p}_{12}\ p_5\ \overline{p}_2\ \overline{p}_{10}\ p_1\ p_3\ p_{14}$

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Value	T	F	T	T	T	F	F	T		F	T	F	F	T	T	T	F
Reason	8	6	9	d	5	d	1	2		7	d	4	d	10	3	d	12

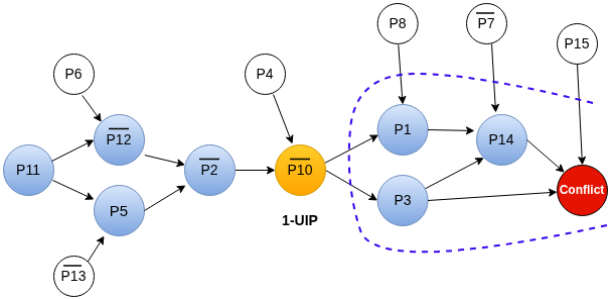


1. $p_6 \vee \overline{p}_7$
2. $p_6 \vee p_{13} \vee p_8$
3. $\overline{p}_8 \vee \overline{p}_4 \vee p_{15}$
4. $\overline{p}_{11} \vee p_6 \vee \overline{p}_{12}$
5. $\overline{p}_{11} \vee p_{13} \vee p_5$
6. $p_{12} \vee \overline{p}_5 \vee \overline{p}_2$
7. $p_2 \vee \overline{p}_4 \vee \overline{p}_{10}$
8. $p_{10} \vee \overline{p}_8 \vee p_1$
9. $p_{10} \vee p_3$
10. $\overline{p}_1 \vee \overline{p}_3 \vee p_7 \vee p_{14}$
11. $\overline{p}_3 \vee \overline{p}_{15} \vee \overline{p}_{14}$
12. $\overline{p}_{16} \vee \overline{p}_{17}$

CDCL Execution - Conflict Analysis

Assignment stack: $\overline{p}_6^{\,d}\ \overline{p}_7\ \overline{p}_{13}^{\,d}\ p_8\ p_4{}^{d}\ p_{15}\ p_{16}{}^{d}\ \overline{p}_{17}\ p_{11}{}^{d}\ \overline{p}_{12}\ p_5\ \overline{p}_2\ \overline{p}_{10}\ p_1\ p_3\ p_{14}$

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Value	T	F	T	T	T	F	F	T		F	T	F	F	T	T	T	F
Reason	8	6	9	d	5	d	1	2		7	d	4	d	10	3	d	12



1. $p_6 \vee \overline{p}_7$
2. $p_6 \vee p_{13} \vee p_8$
3. $\overline{p}_8 \vee \overline{p}_4 \vee p_{15}$
4. $\overline{p}_{11} \vee p_6 \vee \overline{p}_{12}$
5. $\overline{p}_{11} \vee p_{13} \vee p_5$
6. $p_{12} \vee \overline{p}_5 \vee \overline{p}_2$
7. $p_2 \vee \overline{p}_4 \vee \overline{p}_{10}$
8. $p_{10} \vee \overline{p}_8 \vee p_1$
9. $p_{10} \vee p_3$
10. $\overline{p}_1 \vee \overline{p}_3 \vee p_7 \vee p_{14}$
11. $\overline{p}_3 \vee \overline{p}_{15} \vee \overline{p}_{14}$
12. $\overline{p}_{16} \vee \overline{p}_{17}$

We can learn $p_{10} \vee \overline{p}_8 \vee p_7 \vee \overline{p}_{15}$

This allows to backump to state $\overline{p}_6^{\,d}\ \overline{p}_7\ \overline{p}_{13}^{\,d}\ p_8\ p_4{}^{d}\ p_{15}\ p_{10}$

CDCL Execution - Conflict Analysis

Benefits of conflict analysis:

- It allows us to backjump, as opposed to only backtrack
- We can learn a clause (called *lemma*) that might be useful in the future
- In fact, we have the guarantee that the lemma makes unit propagation stronger
- CDCL terminates by either
  - finding a **model** or
  - finding a **falsified clause with only unit propagation** (no decision)

The CDCL Algorithm

$F :=$ initial CNF formula;
$\alpha := \emptyset$
while *true* **do**
  $\alpha := \mathit{unitPropagate}(F, \alpha)$
  if *some clause is false in* α **then**
    if *no decision in* α **then** return UNSAT
    $C := \mathit{conflictAnalyze}(F, \alpha)$
    $F := F \wedge C$
    $\alpha := \mathit{backjump}(C, \alpha)$
  else
    if *all variables are assigned* **then** return SAT
    $\alpha := \alpha \cup \mathit{decide}()$

The CDCL Algorithm

$F :=$ initial CNF formula;
$\alpha := \emptyset$
while *true* **do**
  $\alpha := \mathit{unitPropagate}(F, \alpha)$
  if *some clause is false in* α **then**
    if *no decision in* α **then** return UNSAT
    $C := \mathit{conflictAnalyze}(F, \alpha)$
    $F := F \wedge C$
    $\alpha := \mathit{backjump}(C, \alpha)$
  else
    if *all variables are assigned* **then** return SAT
    clauses_reduction_if_applicable()
    restart_if_applicable()
    pre/inprocessing_if_applicable()
    $\alpha := \alpha \cup \mathit{decide}()$

- Periodically, the solver
  - Removes some lemmas
  - Restarts (keeping all lemmas)
  - Applies preprocessing techniques on the formula
- Note that the solver reports a model when
  - No clause is falsified (no conflict)
  - All variables are defined in α

  Hence we only have to detect clauses that are falsified or propagated
- How can we construct a proof of unsatisfiability?

CDCL - Proof Production

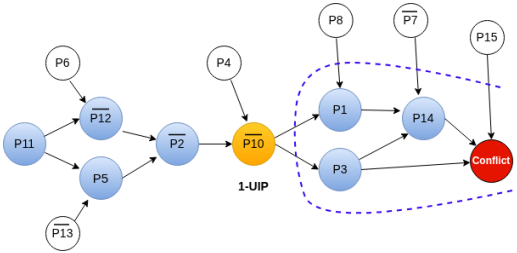
There is a well-known inference rule called resolution [Rob65]:

$$\frac{C \vee \ell \quad D \vee \neg\ell}{C \vee D} \qquad \frac{p \vee \neg q \quad s \vee \neg p \vee r}{\neg q \vee s \vee r}$$

Some theoretical results:

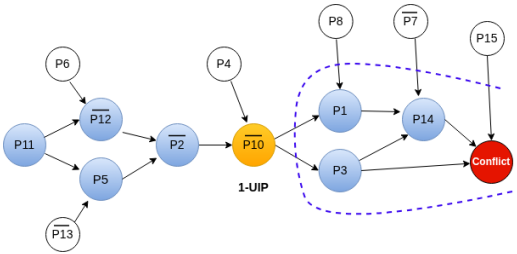
- A CNF F is unsatisfiable iff the empty clause belongs to the closure of F under resolution
- There are formulas $F(n)$ for which any resolution proof has size exponential in n [Hak85]
  - Example: Pigeon-hole principe – PHP(n) – Try to place $n+1$ pigeons into n holes.

CDCL - Proof Production



The analysis on the conflict graph allowed us to learn clause $\overline{p}_{15} \vee p_7 \vee p_{10} \vee \overline{p}_8$

CDCL - Proof Production



The analysis on the conflict graph allowed us to learn clause $\overline{p}_{15} \vee p_7 \vee p_{10} \vee \overline{p}_8$

But this can always be done via a series of resolution steps:

$$\dfrac{(\mathit{conf}):\ \overline{p}_3 \vee \overline{p}_{15} \vee \overline{p}_{14} \qquad (\mathit{reason}_{14}):\ \overline{p}_1 \vee \overline{p}_3 \vee p_7 \vee p_{14}}{\dfrac{\overline{p}_3 \vee \overline{p}_{15} \vee \overline{p}_1 \vee p_7 \qquad (\mathit{reason}_3):\ p_{10} \vee p_3}{\dfrac{\overline{p}_{15} \vee \overline{p}_1 \vee p_7 \vee p_{10} \qquad (\mathit{reason}_1):\ p_{10} \vee \overline{p}_8 \vee p_1}{\overline{p}_{15} \vee p_7 \vee p_{10} \vee \overline{p}_8}}}$$

Proof production

How can we reconstruct the proof of unsatisfiability?

- Unsatisfiability is detected when we have learned enough lemmas that allow us to find a conflict using only unit propagation
- The lemmas we have learned can be derived from the input clauses via resolution
- Unit propagation is a particular case of resolution
- For efficiency reasons, things are a little bit different in practice

Proof complexity results (vaguely stated):

- We can extract resolution proofs from CDCL-based SAT solvers (thus, anything hard for resolution is hard for CDCL)
- The opposite is also true [AFT11, PD11]: if there is a resolution proof of size s, there is a CDCL execution proving unsatisfiability in time polynomial in s

Implementing Unit Propagation

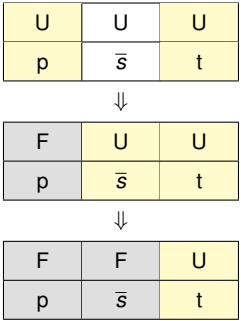
Unit Propagation: Detect all clauses that propagate or are falsified. (up to 80% of the runtime)

Implementing Unit Propagation

Unit Propagation: Detect all clauses that propagate or are falsified. (up to 80% of the runtime)

Clause: $(p \vee \overline{s} \vee t)$

- *Two-watched literal scheme:* [MMZ+01]
  If **two non-false** literals exist, there is no propagation or conflict.



Watch List (ℓ): a list of pointers to clauses where ℓ is being watched.

Whenever ℓ becomes false, traverse its watch list to:

- Try to watch a replacement.
- Propagation is possible **only** when all literals are false **except for one**.
- Unwatched literals cause no work.
- End up watching inactive literals.
- No work upon backtracking.

Gory details: The watch list is implemented as a vector (contigous memory!)

Dereferencing the pointer to the clause is the most time-consuming operation of the solver (almost surely a cache miss)

To avoid the cache miss, a literal from the clause is stored in the watch list with the hope of it being true and not having to load the clause:

···	ptr1	lit1	ptr2	lit2	ptr3	lit3	···

Parallel SAT Solving

CDCL is inherentely sequential, difficult to parallelize (unit propagation is P-complete).

However, two parallel frameworks have been devised to take advantage of multicore/clusters.

- **Distributed Clause-Sharing:**
  - Several CDCL solvers (or configurations of them) are run in parallel on the same problem
  - The only information they share are short learned lemmas
  - General purpose algorithm: works well on most problems
  - MALLOBSAT [SS24] is the state-of-the-art solver
- **Cube-and-Conquer:**
  - A look-ahead solver builds conjunctions of literals (cubes) that simplify the problem as much as possible
  - The set of cubes should cover the whole search space (e.g. $\{\{x\}, \{\neg x, y\}, \{\neg x, \neg y\}\}$)
  - Cubes are not splitted further when it is expected that the formula conjuncted with the cube is tractable with a CDCL solver
  - Each subproblem is sent in parallel to a different worker
  - Works well on highly combinatorial problems (e.g. Boolean Pythagorean Triples)

Parallel SAT Solving

CDCL is inherentely sequential, difficult to parallelize (unit propagation is P-complete).

However, two parallel frameworks have been devised to take advantage of multicore/clusters.

- **Distributed Clause-Sharing:**
  - Several CDCL solvers (or configurations of them) are run in parallel on the same problem
  - The only information they share are short learned lemmas
  - General purpose algorithm: works well on most problems
  - MALLOBSAT [SS24] is the state-of-the-art solver
- **Cube-and-Conquer:**
  - A look-ahead solver builds conjunctions of literals (cubes) that simplify the problem as much as possible
  - The set of cubes should cover the whole search space (e.g. $\{\{x\}, \{\neg x, y\}, \{\neg x, \neg y\}\}$)
  - Cubes are not splitted further when it is expected that the formula conjuncted with the cube is tractable with a CDCL solver
  - Each subproblem is sent in parallel to a different worker
  - Works well on highly combinatorial problems (e.g. Boolean Pythagorean Triples)

Parallel SAT Solving

CDCL is inherentely sequential, difficult to parallelize (unit propagation is P-complete).

However, two parallel frameworks have been devised to take advantage of multicore/clusters.

- **Distributed Clause-Sharing:**
  - Several CDCL solvers (or configurations of them) are run in parallel on the same problem
  - The only information they share are short learned lemmas
  - General purpose algorithm: works well on most problems
  - MALLOBSAT [SS24] is the state-of-the-art solver
- **Cube-and-Conquer:**
  - A look-ahead solver builds conjunctions of literals (cubes) that simplify the problem as much as possible
  - The set of cubes should cover the whole search space (e.g. $\{\{x\}, \{\neg x, y\}, \{\neg x, \neg y\}\}$)
  - Cubes are not splitted further when it is expected that the formula conjuncted with the cube is tractable with a CDCL solver
  - Each subproblem is sent in parallel to a different worker
  - Works well on highly combinatorial problems (e.g. Boolean Pythagorean Triples)

Outline of the Talk

Need to go beyond CDCL

Current state of SAT solving:

- Most successful SAT solvers essentially implement the CDCL algorithm
- It is well known that CDCL is equivalent to general resolution
- Hence, hard problems for resolution are hard for CDCL
  (pigeon-hole, Tseitin, clique, etc.)

How can we make CDCL-based solvers stronger?

- Allowing them to use stronger proof systems:
  - Cutting planes (e.g. CDCL-based PB solvers)
  - Extended Resolution
  - Propagation Redundancy (e.g. SDCL-based solvers)

Need to go beyond CDCL

Current state of SAT solving:

- Most successful SAT solvers essentially implement the CDCL algorithm
- It is well known that CDCL is equivalent to general resolution
- Hence, hard problems for resolution are hard for CDCL
  (pigeon-hole, Tseitin, clique, etc.)

How can we make CDCL-based solvers stronger?

- Allowing them to use stronger proof systems:
  - Cutting planes (e.g. CDCL-based PB solvers)
  - Extended Resolution
  - Propagation Redundancy (e.g. SDCL-based solvers)

Extended Resolution

The Extended Resolution proof (ER) system consists of two inference rules:

Resolution

$$\frac{\ell \vee C \quad \neg\ell \vee D}{C \vee D}$$

Extension rule

$$\frac{}{x \leftrightarrow \ell_1 \wedge \ell_2}$$

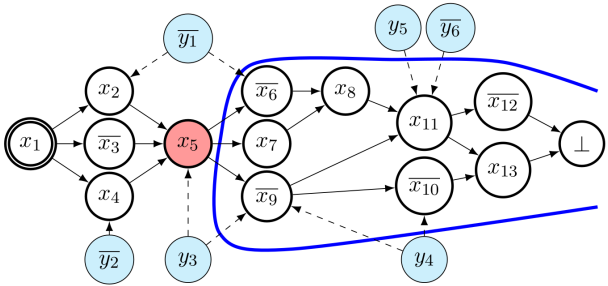
x is a fresh variable, ℓ_1, ℓ_2 are not

It is known that ER has polynomial proofs for some problems that only have exponential proofs in Resolution [Coo76]

If we want to augment CDCL with the power of ER, we need to know:

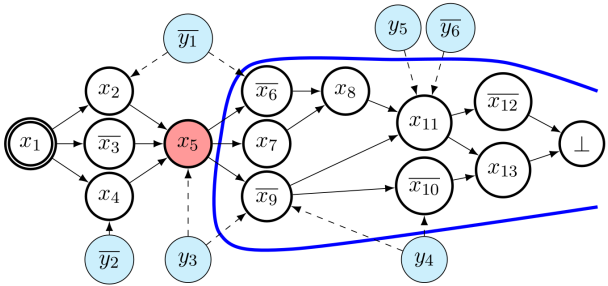
- Which extension variables to introduce?
- When to introduce an extension variable?

CDCL Conflict Analysis



- Conflict analysis in CDCL is based on the 1UIP learning scheme
- Red node x_5 is the 1UIP (i.e. the closest dominator to the conflict: any path from x_1 to conflict ($\bot$) necessarily goes through x_5)
- Blue nodes are literals from previous decision levels
- Double-circled node x_1 is the last decision
- Lemma learned is $\overline{x}_5 \vee y_1 \vee \overline{y}_3 \vee \overline{y}_5 \vee y_6 \vee \overline{y}_4$

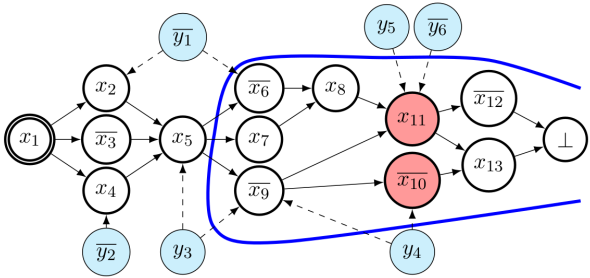
Dual Implication Points



We could generalize the idea and identify pairs of nodes $\{n_1, n_2\}$ such that every path from the decision to the conflict necessarily goes through n_1 or n_2.

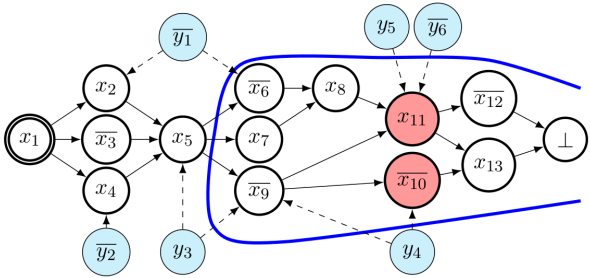
That is, we could look for 2-dominators (in graph terminology).

Dual Implication Points



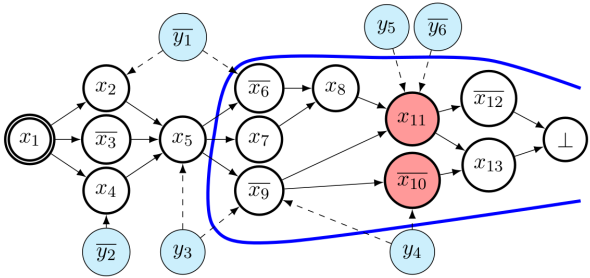
- $\{x_{11}, \overline{x}_{10}\}$ is such a pair (Dual Implication Point - DIP)
- If we merged the two nodes we would have a UIP closer to the conflict
- Add $z \leftrightarrow \overline{x}_{10} \wedge x_{11}$ and merge $\overline{x}_{10}$ and x_{11}
- We can learn unit clause $\neg z$
- This is the essence of DIP-based Extended Resolution [BCGO24].

Dual Implication Points



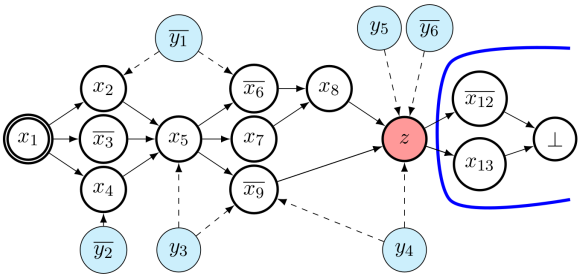
- $\{x_{11}, \overline{x}_{10}\}$ is such a pair (Dual Implication Point - DIP)
- If we merged the two nodes we would have a UIP closer to the conflict
- Add $z \leftrightarrow \overline{x}_{10} \wedge x_{11}$ and merge $\overline{x}_{10}$ and x_{11}
- We can learn unit clause $\neg z$
- This is the essence of DIP-based Extended Resolution [BCGO24].

Dual Implication Points



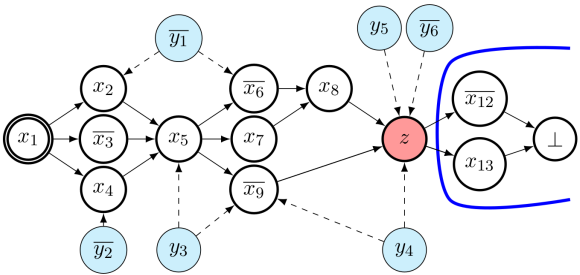
- $\{x_{11}, \overline{x}_{10}\}$ is such a pair (Dual Implication Point - DIP)
- If we merged the two nodes we would have a UIP closer to the conflict
- Add $z \leftrightarrow \overline{x}_{10} \wedge x_{11}$ and merge $\overline{x}_{10}$ and x_{11}
- We can learn unit clause $\neg z$
- This is the essence of DIP-based Extended Resolution [BCGO24].

Dual Implication Points



- $\{x_{11}, \overline{x}_{10}\}$ is such a pair (Dual Implication Point - DIP)
- If we merged the two nodes we would have a UIP closer to the conflict
- Add $z \leftrightarrow \overline{x}_{10} \wedge x_{11}$ and merge $\overline{x}_{10}$ and x_{11}
- We can learn unit clause $\neg z$
- This is the essence of DIP-based Extended Resolution [BCGO24].

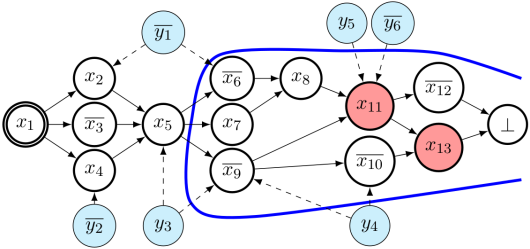
Dual Implication Points

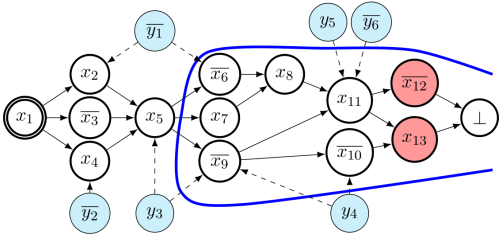


- $\{x_{11}, \overline{x}_{10}\}$ is such a pair (Dual Implication Point - DIP)
- If we merged the two nodes we would have a UIP closer to the conflict
- Add $z \leftrightarrow \overline{x}_{10} \wedge x_{11}$ and merge $\overline{x}_{10}$ and x_{11}
- We can learn unit clause $\neg z$
- This is the essence of DIP-based Extended Resolution [BCGO24].

Dual Implication Points

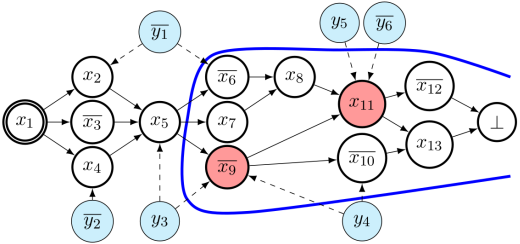
Note that there might be many DIPs (quadratic in the worst case)

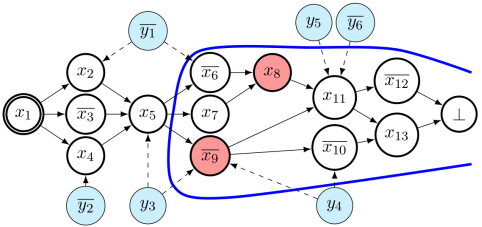




Dual Implication Points

Note that there might be many DIPs (quadratic in the worst case)





Experimental Results

- However, there is an algorithm [BGO26] to compute all of them in time linear in the size of the graph
- Experimental results reveal the this approach outperforms CDCL on a limited number of problems (namely the ones with XOR constraints)

Redundancy Notions

Another proof system that is more powerful than resolution is based on redundancy:

Definition ([HKB17])

A clause C is **redundant** wrt F if and only if F and $F \wedge C$ are equisatisfiable

Theorem ([HKB17])

A clause C is **redundant** *wrt F if and only if there exists an assignment ω such that $\omega \models C$ and $F \wedge \neg C \models F|_\omega$ (where $F|_\omega$ is the result of removing from F all clauses satisfied by ω and all literals falsified by ω, i.e. simplify F wrt ω)*

Intuitively, this is a *"without loss of generality reasoning"*. We can safely add C to F, because conditions $F \wedge \neg C \models F|_\omega$ and $\omega \models C$ ensure that any model of $F \wedge \neg C$ can be converted into a model of $F \wedge C$.

But, even if we know ω (witness), it is hard to check that $F \wedge \neg C \models F|_\omega$.

Redundancy Notions

Another proof system that is more powerful than resolution is based on redundancy:

Definition ([HKB17])

A clause C is **redundant** wrt F if and only if F and $F \wedge C$ are equisatisfiable

Theorem ([HKB17])

A clause C is **redundant** *wrt F if and only if there exists an assignment ω such that $\omega \models C$ and $F \wedge \neg C \models F|_\omega$ (where $F|_\omega$ is the result of removing from F all clauses satisfied by ω and all literals falsified by ω, i.e. simplify F wrt ω)*

Intuitively, this is a *"without loss of generality reasoning"*. We can safely add C to F, because conditions $F \wedge \neg C \models F|_\omega$ and $\omega \models C$ ensure that any model of $F \wedge \neg C$ can be converted into a model of $F \wedge C$.

But, even if we know ω (witness), it is hard to check that $F \wedge \neg C \models F|_\omega$.

Redundancy Notions

Another proof system that is more powerful than resolution is based on redundancy:

Definition ([HKB17])

A clause C is **redundant** wrt F if and only if F and $F \wedge C$ are equisatisfiable

Theorem ([HKB17])

A clause C is **redundant** *wrt F if and only if there exists an assignment ω such that $\omega \models C$ and $F \wedge \neg C \models F|_\omega$ (where $F|_\omega$ is the result of removing from F all clauses satisfied by ω and all literals falsified by ω, i.e. simplify F wrt ω)*

Intuitively, this is a *"without loss of generality reasoning"*. We can safely add C to F, because conditions $F \wedge \neg C \models F|_\omega$ and $\omega \models C$ ensure that any model of $F \wedge \neg C$ can be converted into a model of $F \wedge C$.

But, even if we know ω (witness), it is hard to check that $F \wedge \neg C \models F|_\omega$.

Redundancy Notions

Another proof system that is more powerful than resolution is based on redundancy:

Definition ([HKB17])

A clause C is **redundant** wrt F if and only if F and $F \wedge C$ are equisatisfiable

Theorem ([HKB17])

A clause C is **redundant** *wrt F if and only if there exists an assignment ω such that $\omega \models C$ and $F \wedge \neg C \models F|_\omega$ (where $F|_\omega$ is the result of removing from F all clauses satisfied by ω and all literals falsified by ω, i.e. simplify F wrt ω)*

Intuitively, this is a *"without loss of generality reasoning"*. We can safely add C to F, because conditions $F \wedge \neg C \models F|_\omega$ and $\omega \models C$ ensure that any model of $F \wedge \neg C$ can be converted into a model of $F \wedge C$.

But, even if we know ω (witness), it is hard to check that $F \wedge \neg C \models F|_\omega$.

Redundancy Notions (2)

This is why a weaker notion is considered:

Definition

A clause C is **Propagation Redundant (PR)** wrt F if and only if there exists an assignment ω such that $\omega \models C$ and $F \wedge \neg C \vdash_1 F|_\omega$
(i.e. $\mathit{UnitProp}(F \wedge \neg C \wedge \neg F|_\omega)$ = conflict)

Particular cases of PR clauses:

- SPR: ω assigns only variables of C
- LPR/RAT: ω assigns only vars of C and satisfies exactly one lit of C
- Set-blocked [KSTB16]: particular case of SPR, syntactic definition
- Blocked [JBH10]: particular case of set-blocked, syntactic definition

From CDCL to SDCL

Satisfiability-Driven Clause-Learning (**SDCL**) [HKSB17]) is a way to implement a redundancy-based proof procedure:

$\alpha := \emptyset$
while *true* **do**
  $\alpha := \mathit{unitPropagate}(F, \alpha)$
  if *conflict found* **then**
    $C := \mathit{analyzeConflict}()$
    $F := F \wedge C$
    if *C is the empty clause* **then** return UNSAT
    $\alpha := \mathit{backjump}(C, \alpha)$
  else if $\neg\alpha$ *is redundant wrt F* **then**
    $C := \mathit{decisions}(\neg\alpha)$
    $F := F \wedge C$
    $\alpha := \mathit{backjump}(C, \alpha)$
  else
    if *all variables are assigned* **then** return SAT
    $\alpha := \alpha \cup \mathit{Decide}()$

From CDCL to SDCL

SDCL is a way to implement a redundancy-based proof procedure:

$\alpha := \emptyset$
while *true* **do**
  $\alpha := \mathit{unitPropagate}(F, \alpha)$
  if *conflict found* **then**
    $C := \mathit{analyzeConflict}()$
    $F := F \wedge C$
    if *C is the empty clause* **then** return UNSAT
    $\alpha := \mathit{backjump}(C, \alpha)$
  else if $P_\alpha(F)$ *is satisfiable* **then**
    $C := \mathit{decisions}(\neg\alpha)$
    $F := F \wedge C$
    $\alpha := \mathit{backjump}(C, \alpha)$
  else
    if *all variables are assigned* **then** return SAT
    $\alpha := \alpha \cup \mathit{Decide}()$

Definition ([HKSB17])

A **pruning predicate** for F and α is a formula $P_\alpha(F)$ such that, if $P_\alpha(F)$ is satisfiable then clause $\neg\alpha$ is redundant wrt F

Pruning Predicates

Definition

A **pruning predicate** for F and α is a formula $P_\alpha(F)$ such that, if $P_\alpha(F)$ is satisfiable then clause $\neg\alpha$ is redundant wrt F

Do we know how to construct pruning predicates?

- Purely positive reduct: formula $\neg\alpha \wedge G$, where $G = \{\mathit{satisfied}_\alpha(D) \mid D \in F \text{ and } \alpha \models D\}$.
- Positive reduct: formula $\neg\alpha \wedge G$, where $G = \{\mathit{touched}_\alpha(D) \mid D \in F \text{ and } \alpha \models D\}$.
- Filtered positive reduct: formula $\neg\alpha \wedge G$, where $G = \{\mathit{touched}_\alpha(D) \mid D \in F \text{ and } F \wedge \alpha \not\vdash_1 \mathit{untouched}_\alpha(D)\}$.

Proposition ([OLW+25])

The purely positive reduct is satisfiable if and only if $\neg\alpha$ is blocked in F.

Pruning Predicates

Definition

A **pruning predicate** for F and α is a formula $P_\alpha(F)$ such that, if $P_\alpha(F)$ is satisfiable then clause $\neg\alpha$ is redundant wrt F

Do we know how to construct pruning predicates?

- Purely positive reduct: formula $\neg\alpha \wedge G$, where $G = \{\mathit{satisfied}_\alpha(D) \mid D \in F \text{ and } \alpha \models D\}$.
- Positive reduct: formula $\neg\alpha \wedge G$, where $G = \{\mathit{touched}_\alpha(D) \mid D \in F \text{ and } \alpha \models D\}$.
- Filtered positive reduct: formula $\neg\alpha \wedge G$, where $G = \{\mathit{touched}_\alpha(D) \mid D \in F \text{ and } F \wedge \alpha \not\vdash_1 \mathit{untouched}_\alpha(D)\}$.

Proposition ([HKSB17])

The positive reduct is satisfiable if and only if $\neg\alpha$ is set-blocked in F.

Pruning Predicates

Definition

A **pruning predicate** for F and α is a formula $P_\alpha(F)$ such that, if $P_\alpha(F)$ is satisfiable then clause $\neg\alpha$ is redundant wrt F

Do we know how to construct pruning predicates?

- Purely positive reduct: formula $\neg\alpha \wedge G$, where $G = \{satisfied_\alpha(D) \mid D \in F \text{ and } \alpha \models D\}$.
- Positive reduct: formula $\neg\alpha \wedge G$, where $G = \{touched_\alpha(D) \mid D \in F \text{ and } \alpha \models D\}$.
- Filtered positive reduct: formula $\neg\alpha \wedge G$, where $G = \{touched_\alpha(D) \mid D \in F \text{ and } F \wedge \alpha \not\vdash_1 untouched_\alpha(D)\}$.

Proposition ([HKB19])

The filtered positive reduct is satisfiable if and only if $\neg\alpha$ is SPR in F.

SDCL p-simulates Redundancy

An analogous result [OLW+25] to the equivalence between CDCL and resolution can be obtained for SDCL:

- Consider a redundancy notion $\mathcal{R}$ for which a corresponding reduct $p_{\alpha,\mathcal{R}}(F)$ exists
- Consider the proof system P consisting of resolution and addition of clauses satisfying $\mathcal{R}$
- Given a refutation Π in P for a formula F, an SDCL solver that uses the $p_{\alpha,\mathcal{R}}(F)$ reduct can produce a refutation for F in P whose size is at most polynomially longer than the size of Π
- However, since checking the satisfiability of $p_{\alpha,\mathcal{R}}(F)$ can take exponential time, we cannot guarantee that the refutation is found in polynomial time

Thank you for your attention!

References I

[AFT11] Albert Atserias, Johannes Klaus Fichte, and Marc Thurley, *Clause-learning algorithms with many restarts and bounded-width resolution*, J. Artif. Intell. Res. **40** (2011), 353–373.

[BCGO24] Sam Buss, Jonathan Chung, Vijay Ganesh, and Albert Oliveras, *Extended resolution clause learning via dual implication points*, CoRR **abs/2406.14190** (2024), available at 2406.14190.

[BF15] Armin Biere and Andreas Fröhlich, *Evaluating CDCL variable scoring schemes*, Theory and applications of satisfiability testing - SAT 2015 - 18th international conference, austin, tx, usa, september 24-27, 2015, proceedings, 2015, pp. 405–422.

[BGO26] Sam Buss, Vijay Ganesh, and Albert Oliveras, *A linear time algorithm for two-vertex bottlenecks*, 2026. In preparation. Preliminary version available at https://math.ucsd.edu/~sbuss/ResearchWeb/TwoVertBottlenecks.

[Coo71] Stephen A. Cook, *The Complexity of Theorem-Proving Procedures*, Proceedings of the 3rd annual ACM symposium on theory of computing, 1971, pp. 151–158.

[Coo76] Stephen A. Cook, *A short proof of the pigeon hole principle using extended resolution*, SIGACT News **8** (1976), no. 4, 28–32.

[DLL62] Martin Davis, George Logemann, and Donald Loveland, *A Machine Program for Theorem-Proving*, Communications of the ACM, CACM **5** (July 1962), no. 7, 394–397.

References II

[DP60] M. Davis and H. Putnam, *A Computing Procedure for Quantification Theory*, Journal of the ACM, JACM **7** (1960), no. 3, 201–215.

[Hak85] Armin Haken, *The intractability of resolution*, Theoretical Computer Science **39** (1985), no. 2 & 3, 297–308.

[HKB17] Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere, *Short proofs without new variables*, Automated deduction - CADE 26 - 26th international conference on automated deduction, gothenburg, sweden, august 6-11, 2017, proceedings, 2017, pp. 130–147.

[HKB19] Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere, *Encoding redundancy for satisfaction-driven clause learning*, Tools and algorithms for the construction and analysis of systems - 25th international conference, TACAS 2019, held as part of the european joint conferences on theory and practice of software, ETAPS 2019, prague, czech republic, april 6-11, 2019, proceedings, part I, 2019, pp. 41–58.

[HKSB17] Marijn J. H. Heule, Benjamin Kiesl, Martina Seidl, and Armin Biere, *Pruning through satisfaction*, Hardware and software: Verification and testing - 13th international haifa verification conference, HVC 2017, haifa, israel, november 13-15, 2017, proceedings, 2017, pp. 179–194.

References III

[JBH10] Matti Järvisalo, Armin Biere, and Marijn Heule, *Blocked clause elimination*, Tools and algorithms for the construction and analysis of systems, 16th international conference, TACAS 2010, held as part of the joint european conferences on theory and practice of software, ETAPS 2010, paphos, cyprus, march 20-28, 2010. proceedings, 2010, pp. 129–144.

[KSTB16] Benjamin Kiesl, Martina Seidl, Hans Tompits, and Armin Biere, *Super-blocked clauses*, Automated reasoning - 8th international joint conference, IJCAR 2016, coimbra, portugal, june 27 - july 2, 2016, proceedings, 2016, pp. 45–61.

[MLM21] João Marques-Silva, Inês Lynce, and Sharad Malik, *Conflict-driven clause learning SAT solvers*, Handbook of satisfiability - second edition, 2021, pp. 133–182.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik, *Chaff: engineering an efficient SAT solver*, Proceedings of the 38th annual design automation conference, 2001, pp. 530–535.

[NOT06] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli, *Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T)*, Journal of the ACM, JACM **53** (2006), no. 6, 937–977.

[OLW+25] Albert Oliveras, Chunxiao (Ian) Li, Darryl Wu, Jonathan Chung, and Vijay Ganesh, *Improving and understanding the power of satisfaction-driven clause learning*, J. Artif. Intell. Res. **83** (2025).

References IV

[PD11] Knot Pipatsrisawat and Adnan Darwiche, *On the power of clause-learning SAT solvers as resolution engines*, Artif. Intell. **175** (2011), no. 2, 512–525.

[Rob65] J.A. Robinson, *A Machine-Oriented Logic Based on the Resolution Principle*, Journal of the ACM, JACM **12** (1965), no. 1, 23–41.

[SS24] Dominik Schreiber and Peter Sanders, *Mallobsat: Scalable SAT solving by clause sharing*, J. Artif. Intell. Res. **80** (2024), 1437–1495.