# Sure, Your Algorithm Is Really Fast, But Is It Really Correct?

Jakob Nordström

University of Copenhagen and Lund University

DIKU Bits
September 10, 2024

# This Is Me...

**Jakob Nordström**

Professor

Algorithms and Complexity Section

Department of Computer Science (DIKU)

University of Copenhagen

(and also at Lund University)

https://jakobnordstrom.se

jn@di.ku.dk

# ... And Here Are Three Problems I Get Paid for Thinking About

$$
\begin{aligned}
&(x \lor z) \land (y \lor \neg z) \land (x \lor \neg y \lor u) \land (\neg y \lor \neg u) \\
\land\ &(u \lor v) \land (\neg x \lor \neg v) \land (\neg u \lor w) \land (\neg x \lor \neg u \lor \neg w)
\end{aligned}
$$

# . . . And Here Are Three Problems I Get Paid for Thinking About

$$(x \lor z) \land (y \lor \neg z) \land (x \lor \neg y \lor u) \land (\neg y \lor \neg u)$$
$$\land\ (u \lor v) \land (\neg x \lor \neg v) \land (\neg u \lor w) \land (\neg x \lor \neg u \lor \neg w)$$

- Variables should be set to **true** or **false**
- Constraints like $(x \lor \neg y \lor z)$ means $x$ or $z$ should be true or $y$ false
- $\land$ means all constraints should hold simultaneously
- Is there a truth value assignment satisfying all constraints?

# . . . And Here Are Three Problems I Get Paid for Thinking About

$$(x \lor z) \land (y \lor \neg z) \land (x \lor \neg y \lor u) \land (\neg y \lor \neg u)$$
$$\land\ (u \lor v) \land (\neg x \lor \neg v) \land (\neg u \lor w) \land (\neg x \lor \neg u \lor \neg w)$$

$$\begin{aligned}
(1-x)(1-z) &= 0 \\
(1-y)z &= 0 \\
(1-x)y(1-u) &= 0 \\
yu &= 0 \\
(1-u)(1-v) &= 0 \\
xv &= 0 \\
u(1-w) &= 0 \\
xuw &= 0
\end{aligned}$$

For **true** $= 1$ and **false** $= 0$, is there a $\{0,1\}$-valued solution?

# . . . And Here Are Three Problems I Get Paid for Thinking About

$$(x \lor z) \land (y \lor \neg z) \land (x \lor \neg y \lor u) \land (\neg y \lor \neg u)$$
$$\land\ (u \lor v) \land (\neg x \lor \neg v) \land (\neg u \lor w) \land (\neg x \lor \neg u \lor \neg w)$$

$$\begin{aligned}
1 - x - z + xz &= 0 \\
z - yz &= 0 \\
y - xy - yu + xyu &= 0 \\
yu &= 0 \\
1 - u - v + uv &= 0 \\
xv &= 0 \\
u - uw &= 0 \\
xuw &= 0
\end{aligned}$$

For **true** $= 1$ and **false** $= 0$, is there a $\{0, 1\}$-valued solution?

# . . . And Here Are Three Problems I Get Paid for Thinking About

$$(x \lor z) \land (y \lor \neg z) \land (x \lor \neg y \lor u) \land (\neg y \lor \neg u)$$
$$\land\ (u \lor v) \land (\neg x \lor \neg v) \land (\neg u \lor w) \land (\neg x \lor \neg u \lor \neg w)$$

| | |
|---|---|
| $1 - x - z + xz = 0$ | $x + z \geq 1$ |
| $z - yz = 0$ | $y + (1 - z) \geq 1$ |
| $y - xy - yu + xyu = 0$ | $x + (1 - y) + u \geq 1$ |
| $yu = 0$ | $(1 - y) + (1 - u) \geq 1$ |
| $1 - u - v + uv = 0$ | $u + v \geq 1$ |
| $xv = 0$ | $(1 - x) + (1 - v) \geq 1$ |
| $u - uw = 0$ | $(1 - u) + w \geq 1$ |
| $xuw = 0$ | $(1 - x) + (1 - u) + (1 - w) \geq 1$ |

For **true** $= 1$ and **false** $= 0$, is there a $\{0, 1\}$-valued solution?

# . . . And Here Are Three Problems I Get Paid for Thinking About

$$(x \lor z) \land (y \lor \lnot z) \land (x \lor \lnot y \lor u) \land (\lnot y \lor \lnot u)$$
$$\land\ (u \lor v) \land (\lnot x \lor \lnot v) \land (\lnot u \lor w) \land (\lnot x \lor \lnot u \lor \lnot w)$$

$$\begin{aligned} 1 - x - z + xz &= 0 \\ z - yz &= 0 \\ y - xy - yu + xyu &= 0 \\ yu &= 0 \\ 1 - u - v + uv &= 0 \\ xv &= 0 \\ u - uw &= 0 \\ xuw &= 0 \end{aligned}$$

$$\begin{aligned} x + z &\geq 1 \\ y - z &\geq 0 \\ x - y + u &\geq 0 \\ -y - u &\geq -1 \\ u + v &\geq 1 \\ -x - v &\geq -1 \\ -u + w &\geq 0 \\ -x - u - w &\geq -2 \end{aligned}$$

For **true** $= 1$ and **false** $= 0$, is there a $\{0,1\}$-valued solution?

# Highly Concrete Applications of These Very Abstract Problems

- Software analysis, testing, and synthesis [DMB11]
- Hardware verification [Sha09]
- Air and train traffic control [ABFP12, FFH+16, ZR14]
- Smart crypto contracts [AGRS20, AGH+22]
- Gene regulatory network inference [PBD+22]
- Computational protein design [AAB+14, HD19]
- Assigning donated organs for transplants [MO12, BvdKM+21]
- Allocation of education and work opportunities [Man16, MMT17]
- Proving theorems in pure mathematics [HK17]

# Highly Concrete Applications of These Very Abstract Problems

- Software analysis, testing, and synthesis [DMB11]
- Hardware verification [Sha09]
- Air and train traffic control [ABFP12, FFH+16, ZR14]
- Smart crypto contracts [AGRS20, AGH+22]
- Gene regulatory network inference [PBD+22]
- Computational protein design [AAB+14, HD19]
- Assigning donated organs for transplants [MO12, BvdKM+21]
- Allocation of education and work opportunities [Man16, MMT17]
- Proving theorems in pure mathematics [HK17]

(Requires fleshing out quite a bit of details, but we don't have time for this in this brief talk. . . )

# Bad News

- This type of problems discussed already in Gödel's famous letter in 1956 to von Neumann ("the father of computer science")
- Topic of intense research in computer science ever since 1960s
- Problems known to be computationally very challenging (NP-complete or worse) [Coo71, Lev73]

# Bad News

- This type of problems discussed already in Gödel's famous letter in 1956 to von Neumann ("the father of computer science")
- Topic of intense research in computer science ever since 1960s
- Problems known to be computationally very challenging (NP-complete or worse) [Coo71, Lev73]
- And machine learning approaches typically do not work

# The Success of Combinatorial Solving (and the Dirty Little Secret)

- Revolution last couple of decades on so-called combinatorial solvers for, e.g.:
  - Boolean satisfiability (SAT) solving and optimization [BHvMW21]
  - Constraint programming [RvBW06]
  - Mixed integer linear programming [AW13, BR07]
  - Satisfiability modulo theories (SMT) solving [BHvMW21]
- Often solve these very hard problems extremely successfully in practice!
- Except the solvers are sometimes wrong...
  [BLB10, CKSW13, AGJ+18, GSD19, BMN22, BBN+23, Tin24]
- Even worse: No way of knowing for sure when errors happen

# What Can Be Done About Solver Bugs?

- **Software testing**
  Hard to get good test coverage for sophisticated solvers
  Progress using fuzzing and delta debugging [BB09, BLB10, KB22, NPB22, PB23]
  But testing inherently can only detect presence of bugs, not absence

# What Can Be Done About Solver Bugs?

- **Software testing**
  Hard to get good test coverage for sophisticated solvers
  Progress using fuzzing and delta debugging [BB09, BLB10, KB22, NPB22, PB23]
  But testing inherently can only detect presence of bugs, not absence

- **Formal verification**
  Prove that solver implementation adheres to formal specification
  Current techniques cannot scale to level of complexity in modern solvers

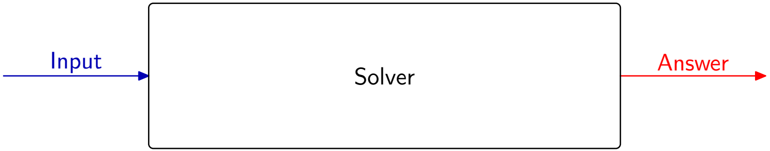
# What Can Be Done About Solver Bugs?

- **Software testing**
  Hard to get good test coverage for sophisticated solvers
  Progress using fuzzing and delta debugging [BB09, BLB10, KB22, NPB22, PB23]
  But testing inherently can only detect presence of bugs, not absence

- **Formal verification**
  Prove that solver implementation adheres to formal specification
  Current techniques cannot scale to level of complexity in modern solvers

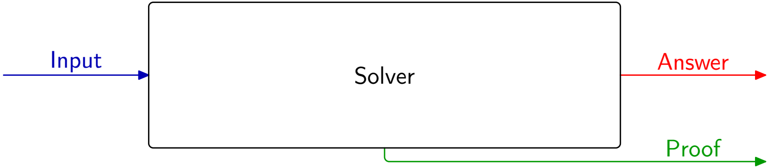
- **Proof logging**
  Make solver certifying [ABM+11, MMNS11] by adding code so that it outputs
  1. not only answer but also
  2. simple, machine-verifiable proof that answer is correct

# Proof Logging with Certifying Solvers: Workflow



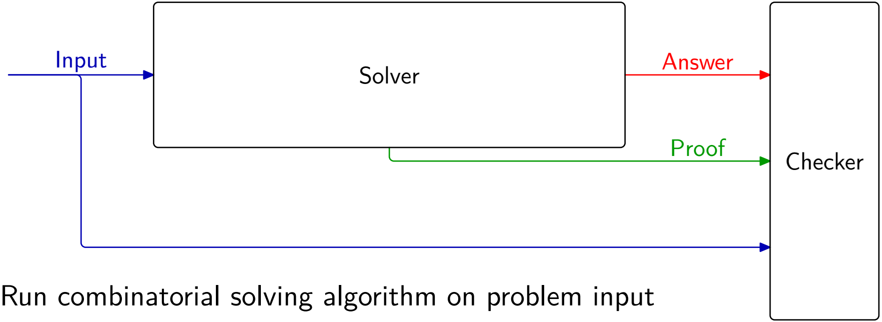

1. Run combinatorial solving algorithm on problem input

# Proof Logging with Certifying Solvers: Workflow



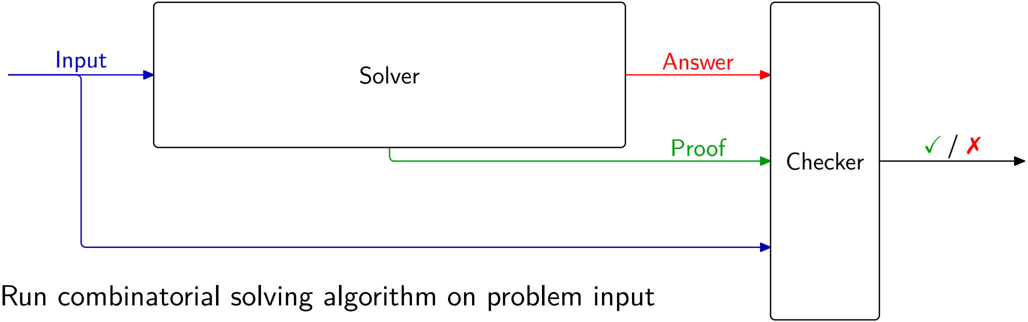

1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof

# Proof Logging with Certifying Solvers: Workflow




1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
3. Feed input + answer + proof to proof checker

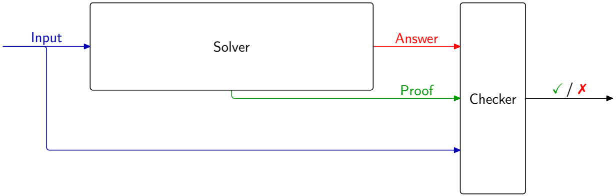
# Proof Logging with Certifying Solvers: Workflow



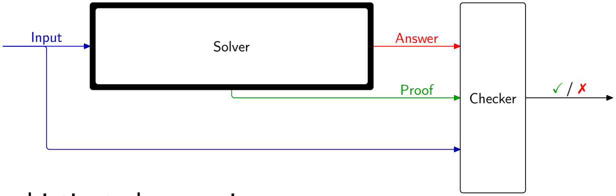

1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
3. Feed input + answer + proof to proof checker
4. Verify that proof checker says answer is correct

# Proof Logging Wishlist

Proof format for certifying solver should be



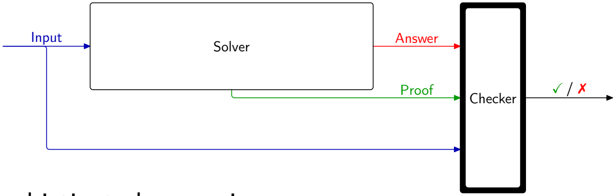
# Proof Logging Wishlist




Proof format for certifying solver should be

- **very powerful:** minimal overhead for sophisticated reasoning

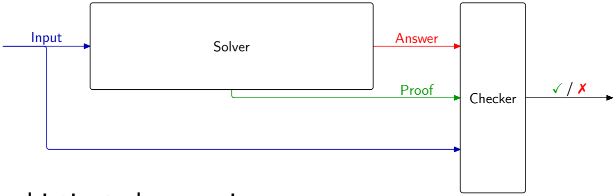
# Proof Logging Wishlist




Proof format for certifying solver should be

- **very powerful:** minimal overhead for sophisticated reasoning
- **dead simple:** checking correctness of proofs should be (almost) trivial

# Proof Logging Wishlist




Proof format for certifying solver should be

- **very powerful:** minimal overhead for sophisticated reasoning
- **dead simple:** checking correctness of proofs should be (almost) trivial

Clear conflict expressivity vs. simplicity!
Asking for both perhaps a little bit too good to be true?

# My Main Message

Proof logging for sophisticated combinatorial solvers is possible!

# My Main Message

Proof logging for sophisticated combinatorial solvers is possible!

- With single, unified method!
- Producing proofs in extremely simple format
- Implemented in VeriPB (`https://gitlab.com/MIAOresearch/software/VeriPB`)

# My Main Message

Proof logging for sophisticated combinatorial solvers is possible!

- With single, unified method!
- Producing proofs in extremely simple format
- Implemented in VeriPB (https://gitlab.com/MIAOresearch/software/VeriPB)

Purpose of this talk:

1. Marketing pitch ☺

# My Main Message

Proof logging for sophisticated combinatorial solvers is possible!

- With single, unified method!
- Producing proofs in extremely simple format
- Implemented in VeriPB (`https://gitlab.com/MIAOresearch/software/VeriPB`)

Purpose of this talk:

1. Marketing pitch ☺
2. Give an example application (but won't be able to get to my own research)

# My Main Message

Proof logging for sophisticated combinatorial solvers is possible!

- With single, unified method!
- Producing proofs in extremely simple format
- Implemented in VeriPB (`https://gitlab.com/MIAOresearch/software/VeriPB`)

Purpose of this talk:

1. Marketing pitch ☺
2. Give an example application (but won't be able to get to my own research)
3. Provide pointers for further reading
   (slides with references online at `https://jakobnordstrom.se/presentations/`)

# The SAT Problem

- Variable $x$: takes value **true** $(=1)$ or **false** $(=0)$
- Literal $\ell$: variable $x$ or its negation $\overline{x}$ (write $\overline{x}$ instead of $\neg x$ to save space)
- Clause $C = \ell_1 \vee \cdots \vee \ell_k$: disjunction of literals
  (Consider as sets, so no repetitions and order irrelevant)
- Conjunctive normal form (CNF) formula $F = C_1 \wedge \cdots \wedge C_m$: conjunction of clauses

## The SAT Problem

Given a CNF formula $F$, is it satisfiable?

For instance, what about:

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge$$
$$(x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

# Proofs for SAT

For satisfiable instances: just specify satisfying assignment

For unsatisfiability: a sequence of clauses

- Each clause follows "obviously" from everything we know so far
- Final clause is empty, meaning contradiction (written $\perp$)
- Means original formula must be inconsistent

# What Is Obvious? Unit Propagation

## Unit Propagation

Clause $C$ unit propagates $\ell$ under partial assignment $\rho$ if $\rho$ falsifies all literals in $C$ except $\ell$

# What Is Obvious? Unit Propagation

## Unit Propagation

Clause $C$ unit propagates $\ell$ under partial assignment $\rho$ if $\rho$ falsifies all literals in $C$ except $\ell$

**Example:** Unit propagate for $\rho = \{p \mapsto 0, q \mapsto 0\}$ on

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

# What Is Obvious? Unit Propagation

## Unit Propagation

Clause $C$ unit propagates $\ell$ under partial assignment $\rho$ if $\rho$ falsifies all literals in $C$ except $\ell$

**Example:** Unit propagate for $\rho = \{p \mapsto 0, q \mapsto 0\}$ on

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

# What Is Obvious? Unit Propagation

## Unit Propagation

Clause $C$ unit propagates $\ell$ under partial assignment $\rho$ if $\rho$ falsifies all literals in $C$ except $\ell$

**Example:** Unit propagate for $\rho = \{p \mapsto 0, q \mapsto 0\}$ on

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

- $p \lor \overline{u}$ propagates $u \mapsto 0$

# What Is Obvious? Unit Propagation

## Unit Propagation

Clause $C$ unit propagates $\ell$ under partial assignment $\rho$ if $\rho$ falsifies all literals in $C$ except $\ell$

**Example:** Unit propagate for $\rho = \{p \mapsto 0, q \mapsto 0\}$ on

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

- $p \lor \overline{u}$ propagates $u \mapsto 0$
- $q \lor r$ propagates $r \mapsto 1$

# What Is Obvious? Unit Propagation

## Unit Propagation

Clause $C$ unit propagates $\ell$ under partial assignment $\rho$ if $\rho$ falsifies all literals in $C$ except $\ell$

**Example:** Unit propagate for $\rho = \{p \mapsto 0, q \mapsto 0\}$ on

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

- $p \lor \overline{u}$ propagates $u \mapsto 0$
- $q \lor r$ propagates $r \mapsto 1$
- Then $\overline{r} \lor w$ propagates $w \mapsto 1$

# What Is Obvious? Unit Propagation

## Unit Propagation

Clause $C$ unit propagates $\ell$ under partial assignment $\rho$ if $\rho$ falsifies all literals in $C$ except $\ell$

**Example:** Unit propagate for $\rho = \{p \mapsto 0, q \mapsto 0\}$ on

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

- $p \lor \overline{u}$ propagates $u \mapsto 0$
- $q \lor r$ propagates $r \mapsto 1$
- Then $\overline{r} \lor w$ propagates $w \mapsto 1$
- No further unit propagations

# What Is Obvious? Unit Propagation

## Unit Propagation

Clause $C$ unit propagates $\ell$ under partial assignment $\rho$ if $\rho$ falsifies all literals in $C$ except $\ell$

**Example:** Unit propagate for $\rho = \{p \mapsto 0, q \mapsto 0\}$ on

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

- $p \vee \overline{u}$ propagates $u \mapsto 0$
- $q \vee r$ propagates $r \mapsto 1$
- Then $\overline{r} \vee w$ propagates $w \mapsto 1$
- No further unit propagations

Proof checker should know how to unit propagate until saturation

# Davis-Putman-Logemann-Loveland (DPLL) SAT Solving from the 1960s

DPLL [DP60, DLL62]: Assign variables and propagate; backtrack when clause violated

"Proof trace": when backtracking, write negation of decisions made

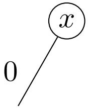
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

# Davis-Putman-Logemann-Loveland (DPLL) SAT Solving from the 1960s

DPLL [DP60, DLL62]: Assign variables and propagate; backtrack when clause violated

"Proof trace": when backtracking, write negation of decisions made

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

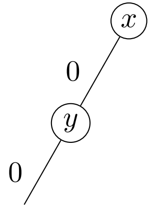


# Davis-Putman-Logemann-Loveland (DPLL) SAT Solving from the 1960s

DPLL [DP60, DLL62]: Assign variables and propagate; backtrack when clause violated

"Proof trace": when backtracking, write negation of decisions made

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

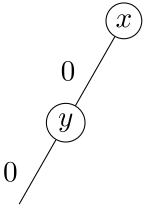


# Davis-Putman-Logemann-Loveland (DPLL) SAT Solving from the 1960s

DPLL [DP60, DLL62]: Assign variables and propagate; backtrack when clause violated

"Proof trace": when backtracking, write negation of decisions made

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



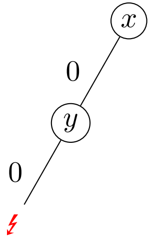
# Davis-Putman-Logemann-Loveland (DPLL) SAT Solving from the 1960s

DPLL [DP60, DLL62]: Assign variables and propagate; backtrack when clause violated

"Proof trace": when backtracking, write negation of decisions made

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

1. $x \lor y$



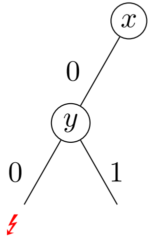
# Davis-Putman-Logemann-Loveland (DPLL) SAT Solving from the 1960s

DPLL [DP60, DLL62]: Assign variables and propagate; backtrack when clause violated

"Proof trace": when backtracking, write negation of decisions made

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

1. $x \vee y$



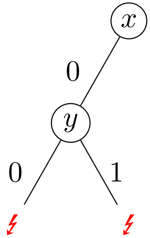
# Davis-Putman-Logemann-Loveland (DPLL) SAT Solving from the 1960s

DPLL [DP60, DLL62]: Assign variables and propagate; backtrack when clause violated

"Proof trace": when backtracking, write negation of decisions made

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

1. $x \vee y$
2. $x \vee \overline{y}$



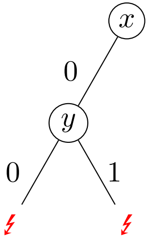
# Davis-Putman-Logemann-Loveland (DPLL) SAT Solving from the 1960s

DPLL [DP60, DLL62]: Assign variables and propagate; backtrack when clause violated

“Proof trace”: when backtracking, write negation of decisions made

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

1. $x \lor y$
2. $x \lor \overline{y}$
3. $x$



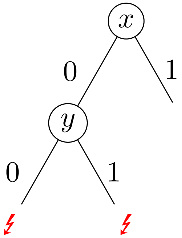
# Davis-Putman-Logemann-Loveland (DPLL) SAT Solving from the 1960s

DPLL [DP60, DLL62]: Assign variables and propagate; backtrack when clause violated

"Proof trace": when backtracking, write negation of decisions made

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

1. $x \lor y$
2. $x \lor \overline{y}$
3. $x$



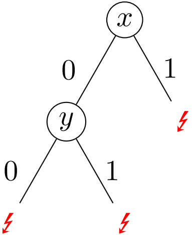
# Davis-Putman-Logemann-Loveland (DPLL) SAT Solving from the 1960s

DPLL [DP60, DLL62]: Assign variables and propagate; backtrack when clause violated

“Proof trace”: when backtracking, write negation of decisions made

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

1. $x \lor y$
2. $x \lor \overline{y}$
3. $x$
4. $\overline{x}$



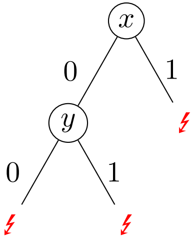
# Davis-Putman-Logemann-Loveland (DPLL) SAT Solving from the 1960s

DPLL [DP60, DLL62]: Assign variables and propagate; backtrack when clause violated

"Proof trace": when backtracking, write negation of decisions made

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

1. $x \lor y$
2. $x \lor \overline{y}$
3. $x$
4. $\overline{x}$
5. $\bot$



# Reverse Unit Propagation (RUP)

To make this a proof, need backtrack clauses to be easily verifiable

# Reverse Unit Propagation (RUP)

To make this a proof, need backtrack clauses to be easily verifiable

## Reverse unit propagation (RUP) clause [GN03, Van08]

$C$ is a reverse unit propagation (RUP) clause with respect to $F$ if

- assigning $C$ to false
- then unit propagating on $F$ until saturation
- leads to contradiction

If so, $F$ clearly implies $C$, and this condition is easy to verify efficiently

# Reverse Unit Propagation (RUP)

To make this a proof, need backtrack clauses to be easily verifiable

### Reverse unit propagation (RUP) clause [GN03, Van08]

$C$ is a reverse unit propagation (RUP) clause with respect to $F$ if

- assigning $C$ to false
- then unit propagating on $F$ until saturation
- leads to contradiction

If so, $F$ clearly implies $C$, and this condition is easy to verify efficiently

### Fact

Backtrack clauses from DPLL solver generate a RUP proof

# What About Modern Conflict-Driven Clause Learning (CDCL)?

Run CDCL SAT solver [BS97, MS99, MMZ+01] on our favourite CNF formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

# What About Modern Conflict-Driven Clause Learning (CDCL)?

Run CDCL SAT solver [BS97, MS99, MMZ+01] on our favourite CNF formula:

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

**Decision**

Free choice to assign value to variable

Notation $p \stackrel{\mathsf{d}}{=} 0$

# What About Modern Conflict-Driven Clause Learning (CDCL)?

Run CDCL SAT solver [BS97, MS99, MMZ+01] on our favourite CNF formula:

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$

**Decision**

Free choice to assign value to variable

Notation $p \stackrel{\mathsf{d}}{=} 0$

# What About Modern Conflict-Driven Clause Learning (CDCL)?

Run CDCL SAT solver [BS97, MS99, MMZ[+]01] on our favourite CNF formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$\boxed{p \overset{\mathsf{d}}{=} 0}$

**Decision**

Free choice to assign value to variable

Notation $p \overset{\mathsf{d}}{=} 0$

**Unit propagation**

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \lor \overline{u}$ forces $u = 0$

Notation $u \overset{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

# What About Modern Conflict-Driven Clause Learning (CDCL)?

Run CDCL SAT solver [BS97, MS99, MMZ+01] on our favourite CNF formula:

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$

$u \stackrel{p \vee \overline{u}}{=} 0$

**Decision**

Free choice to assign value to variable

Notation $p \stackrel{\mathsf{d}}{=} 0$

**Unit propagation**

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$

Notation $u \stackrel{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ is reason clause)

# What About Modern Conflict-Driven Clause Learning (CDCL)?

Run CDCL SAT solver [BS97, MS99, MMZ+01] on our favourite CNF formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$\boxed{p \overset{\mathsf{d}}{=} 0}$

$\boxed{u \overset{p \lor \overline{u}}{=} 0}$

$\boxed{q \overset{\mathsf{d}}{=} 0}$

**Decision**

Free choice to assign value to variable

Notation $p \overset{\mathsf{d}}{=} 0$

**Unit propagation**

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \lor \overline{u}$ forces $u = 0$

Notation $u \overset{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

# What About Modern Conflict-Driven Clause Learning (CDCL)?

Run CDCL SAT solver [BS97, MS99, MMZ+01] on our favourite CNF formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$

$u \stackrel{p \lor \overline{u}}{=} 0$

$\boxed{q \stackrel{\mathsf{d}}{=} 0}$

$r \stackrel{q \lor r}{=} 1$

**Decision**

Free choice to assign value to variable

Notation $p \stackrel{\mathsf{d}}{=} 0$

**Unit propagation**

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \lor \overline{u}$ forces $u = 0$

Notation $u \stackrel{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

# What About Modern Conflict-Driven Clause Learning (CDCL)?

Run CDCL SAT solver [BS97, MS99, MMZ+01] on our favourite CNF formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$

$u \stackrel{p \lor \overline{u}}{=} 0$

$\boxed{q \stackrel{\mathsf{d}}{=} 0}$

$r \stackrel{q \lor r}{=} 1$

$w \stackrel{\overline{r} \lor w}{=} 1$

**Decision**

Free choice to assign value to variable

Notation $p \stackrel{\mathsf{d}}{=} 0$

**Unit propagation**

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \lor \overline{u}$ forces $u = 0$

Notation $u \stackrel{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

# What About Modern Conflict-Driven Clause Learning (CDCL)?

Run CDCL SAT solver [BS97, MS99, MMZ+01] on our favourite CNF formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$

$u \stackrel{p \lor \overline{u}}{=} 0$

$\boxed{q \stackrel{\mathsf{d}}{=} 0}$

$r \stackrel{q \lor r}{=} 1$

$w \stackrel{\overline{r} \lor w}{=} 1$

$\boxed{x \stackrel{\mathsf{d}}{=} 0}$

**Decision**

Free choice to assign value to variable

Notation $p \stackrel{\mathsf{d}}{=} 0$

**Unit propagation**

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \lor \overline{u}$ forces $u = 0$

Notation $u \stackrel{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

# What About Modern Conflict-Driven Clause Learning (CDCL)?

Run CDCL SAT solver [BS97, MS99, MMZ+01] on our favourite CNF formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$p \overset{\mathsf{d}}{=} 0$

$u \overset{p \lor \overline{u}}{=} 0$

$q \overset{\mathsf{d}}{=} 0$

$r \overset{q \lor r}{=} 1$

$w \overset{\overline{r} \lor w}{=} 1$

$x \overset{\mathsf{d}}{=} 0$

$y \overset{u \lor x \lor y}{=} 1$

**Decision**

Free choice to assign value to variable

Notation $p \overset{\mathsf{d}}{=} 0$

**Unit propagation**

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \lor \overline{u}$ forces $u = 0$

Notation $u \overset{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

# What About Modern Conflict-Driven Clause Learning (CDCL)?

Run CDCL SAT solver [BS97, MS99, MMZ+01] on our favourite CNF formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$

$u \stackrel{p \lor \overline{u}}{=} 0$

$\boxed{q \stackrel{\mathsf{d}}{=} 0}$

$r \stackrel{q \lor r}{=} 1$

$w \stackrel{\overline{r} \lor w}{=} 1$

$\boxed{x \stackrel{\mathsf{d}}{=} 0}$

$y \stackrel{u \lor x \lor y}{=} 1$

$z \stackrel{x \lor \overline{y} \lor z}{=} 1$

**Decision**

Free choice to assign value to variable

Notation $p \stackrel{\mathsf{d}}{=} 0$

**Unit propagation**

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \lor \overline{u}$ forces $u = 0$

Notation $u \stackrel{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

# What About Modern Conflict-Driven Clause Learning (CDCL)?

Run CDCL SAT solver [BS97, MS99, MMZ⁺01] on our favourite CNF formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$

$u \stackrel{p \lor \overline{u}}{=} 0$

$\boxed{q \stackrel{\mathsf{d}}{=} 0}$

$r \stackrel{q \lor r}{=} 1$

$w \stackrel{\overline{r} \lor w}{=} 1$

$\boxed{x \stackrel{\mathsf{d}}{=} 0}$

$y \stackrel{u \lor x \lor y}{=} 1$

$z \stackrel{x \lor \overline{y} \lor z}{=} 1$

$\stackrel{\overline{y} \lor \overline{z}}{\bot}$

**Decision**

Free choice to assign value to variable

Notation $p \stackrel{\mathsf{d}}{=} 0$

**Unit propagation**

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \lor \overline{u}$ forces $u = 0$

Notation $u \stackrel{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

# What About Modern Conflict-Driven Clause Learning (CDCL)?

Run CDCL SAT solver [BS97, MS99, MMZ⁺01] on our favourite CNF formula:

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

$\boxed{p \overset{\mathsf{d}}{=} 0}$

$u \overset{p \vee \overline{u}}{=} 0$

decision level 1

$\boxed{q \overset{\mathsf{d}}{=} 0}$

$r \overset{q \vee r}{=} 1$

$w \overset{\overline{r} \vee w}{=} 1$

decision level 2

$\boxed{x \overset{\mathsf{d}}{=} 0}$

$y \overset{u \vee x \vee y}{=} 1$

$z \overset{x \vee \overline{y} \vee z}{=} 1$

$\overset{\overline{y} \vee \overline{z}}{\bot}$

decision level 3

**Decision**

Free choice to assign value to variable

Notation $p \overset{\mathsf{d}}{=} 0$

**Unit propagation**

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$

Notation $u \overset{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

# Conflict Analysis

Time to analyse this conflict and learn from it!

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$\boxed{p \overset{\mathsf{d}}{=} 0}$
$u \overset{p \lor \overline{u}}{=} 0$
decision level 1

$\boxed{q \overset{\mathsf{d}}{=} 0}$
$r \overset{q \lor r}{=} 1$
$w \overset{\overline{r} \lor w}{=} 1$
decision level 2

$\boxed{x \overset{\mathsf{d}}{=} 0}$
$y \overset{u \lor x \lor y}{=} 1$
$z \overset{x \lor \overline{y} \lor z}{=} 1$
$\overset{\overline{y} \lor \overline{z}}{\bot}$
decision level 3

# Conflict Analysis

Time to analyse this conflict and learn from it!

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$$\left.\begin{array}{l} \boxed{p \overset{\mathsf{d}}{=} 0} \\ u \overset{p \lor \overline{u}}{=} 0 \end{array}\right\} \text{decision level 1}$$

$$\left.\begin{array}{l} \boxed{q \overset{\mathsf{d}}{=} 0} \\ r \overset{q \lor r}{=} 1 \\ w \overset{\overline{r} \lor w}{=} 1 \end{array}\right\} \text{decision level 2}$$

$$\left.\begin{array}{l} \boxed{x \overset{\mathsf{d}}{=} 0} \\ y \overset{u \lor x \lor y}{=} 1 \\ z \overset{x \lor \overline{y} \lor z}{=} 1 \\ \overset{\overline{y} \lor \overline{z}}{\bot} \end{array}\right\} \text{decision level 3}$$

Could backtrack by erasing conflict level & flipping last decision

# Conflict Analysis

Time to analyse this conflict and learn from it!

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$$\left.\begin{array}{l} \boxed{p \stackrel{\mathsf{d}}{=} 0} \\ u \stackrel{p \lor \overline{u}}{=} 0 \end{array}\right\} \text{decision level 1}$$

$$\left.\begin{array}{l} \boxed{q \stackrel{\mathsf{d}}{=} 0} \\ r \stackrel{q \lor r}{=} 1 \\ w \stackrel{\overline{r} \lor w}{=} 1 \end{array}\right\} \text{decision level 2}$$

$$\left.\begin{array}{l} \boxed{x \stackrel{\mathsf{d}}{=} 0} \\ y \stackrel{u \lor x \lor y}{=} 1 \\ z \stackrel{x \lor \overline{y} \lor z}{=} 1 \\ \stackrel{\overline{y} \lor \overline{z}}{\bot} \end{array}\right\} \text{decision level 3}$$

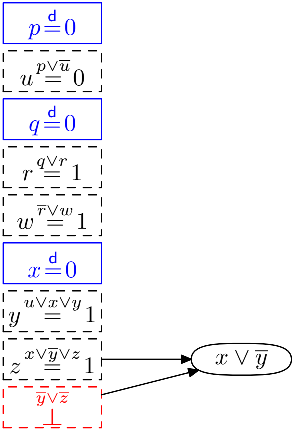
Could backtrack by erasing conflict level & flipping last decision

But want to learn from conflict and cut away as much of search space as possible

# Conflict Analysis

Time to analyse this conflict and learn from it!

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$




Could backtrack by erasing conflict level & flipping last decision

But want to learn from conflict and cut away as much of search space as possible

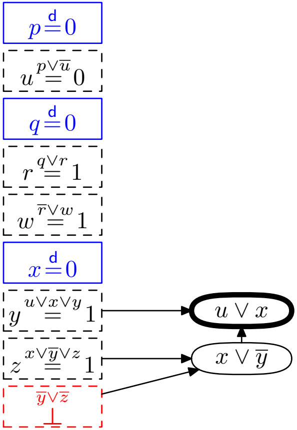
Case analysis over $z$ for last two clauses:

- $x \lor \overline{y} \lor z$ wants $z = 1$
- $\overline{y} \lor \overline{z}$ wants $z = 0$
- Merge clauses & remove $z$ — must satisfy $x \lor \overline{y}$

# Conflict Analysis

Time to analyse this conflict and learn from it!

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$




Could backtrack by erasing conflict level & flipping last decision

But want to learn from conflict and cut away as much of search space as possible

Case analysis over $z$ for last two clauses:

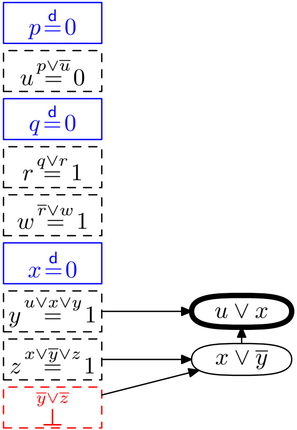
- $x \lor \overline{y} \lor z$ wants $z = 1$
- $\overline{y} \lor \overline{z}$ wants $z = 0$
- Merge clauses & remove $z$ — must satisfy $x \lor \overline{y}$

Repeat until UIP clause with only $1$ variable at conflict level after last decision — learn and backjump

# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

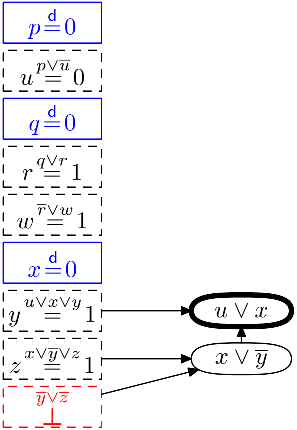
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$





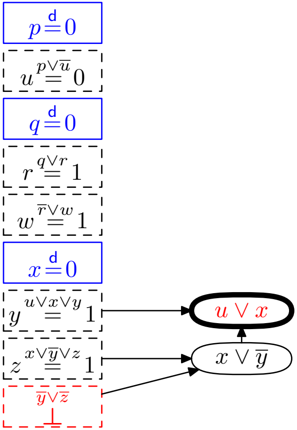


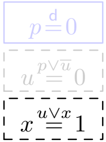
Assertion level $1$ (2nd largest level in learned clause) — trim trail to that level

# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



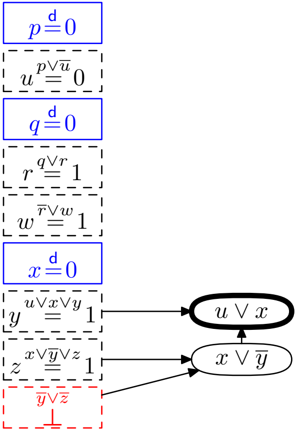

Assertion level $1$ (2nd largest level in learned clause) — trim trail to that level

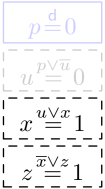
Now UIP literal guaranteed to flip (assert) — but this is a propagation, not a decision

# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$




Assertion level $1$ (2nd largest level in learned clause) — trim trail to that level

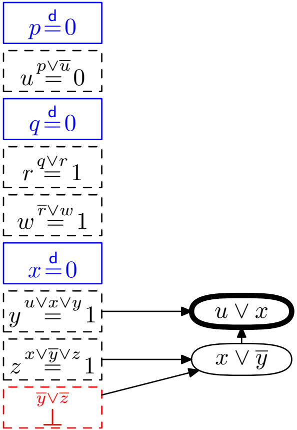
Now UIP literal guaranteed to flip (assert) — but this is a propagation, not a decision

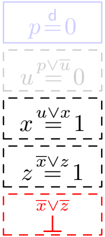
Then continue as before...

# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

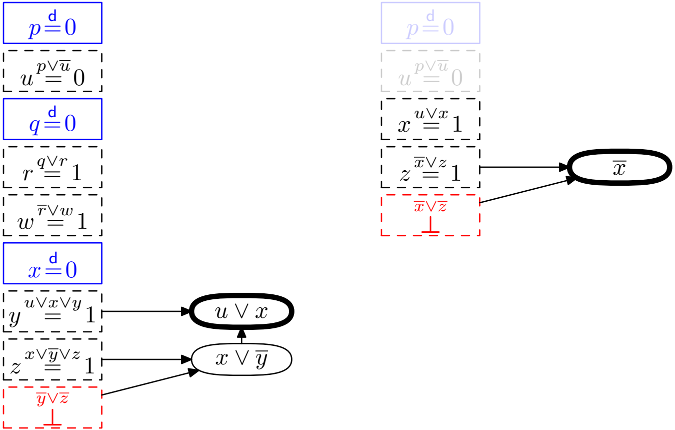


# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

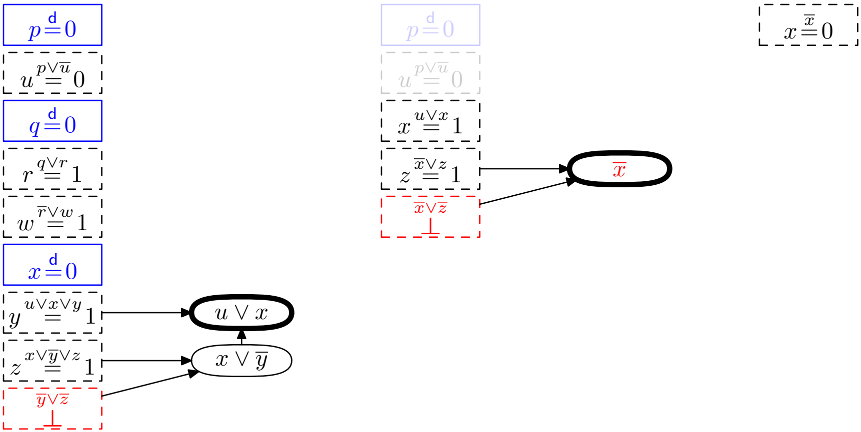
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

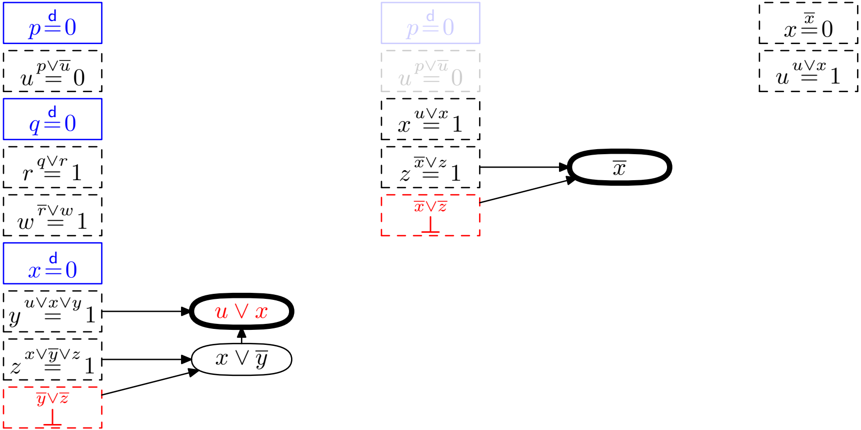
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

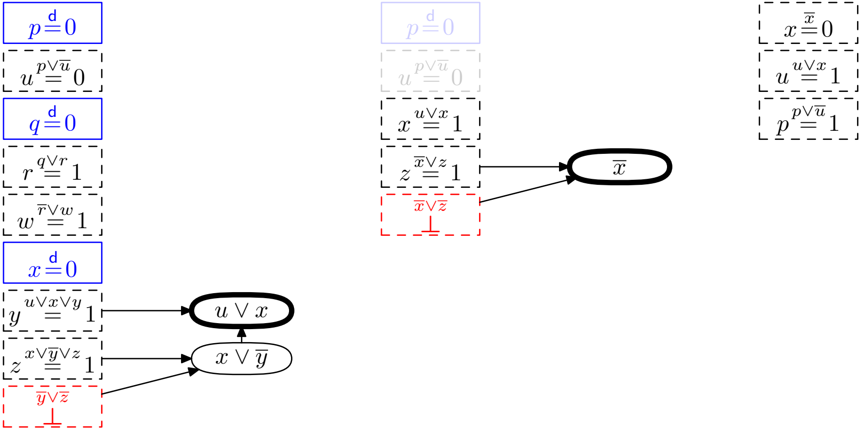
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

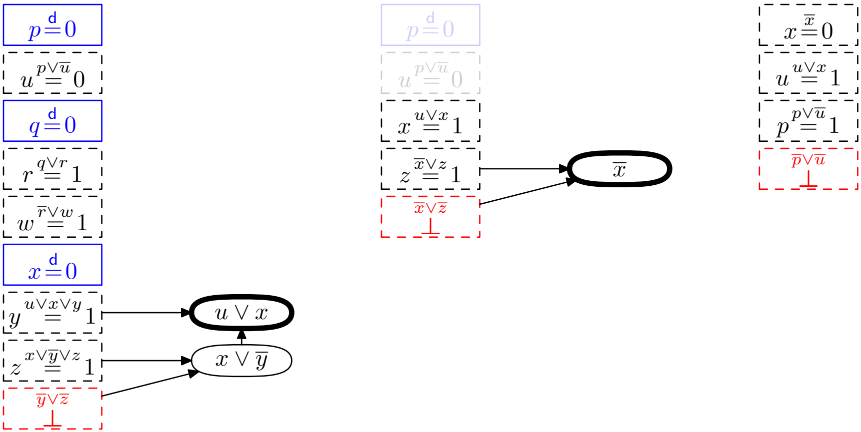
$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$



# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

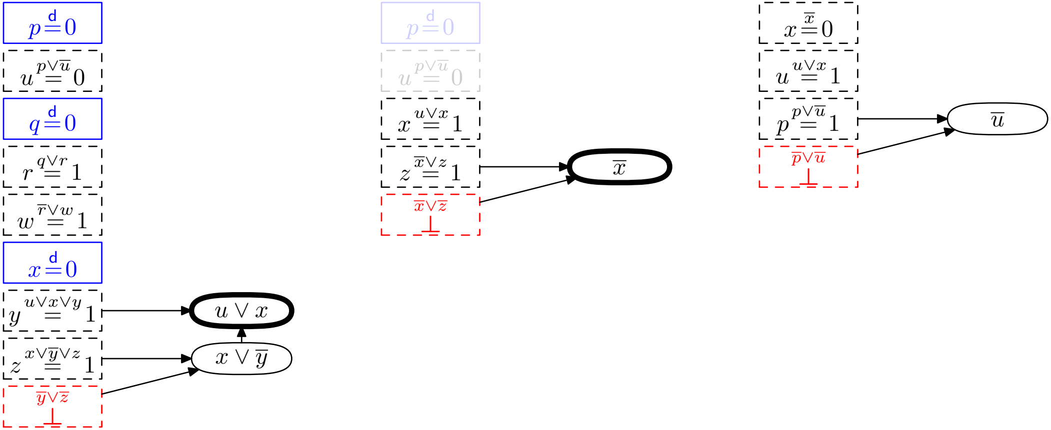
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

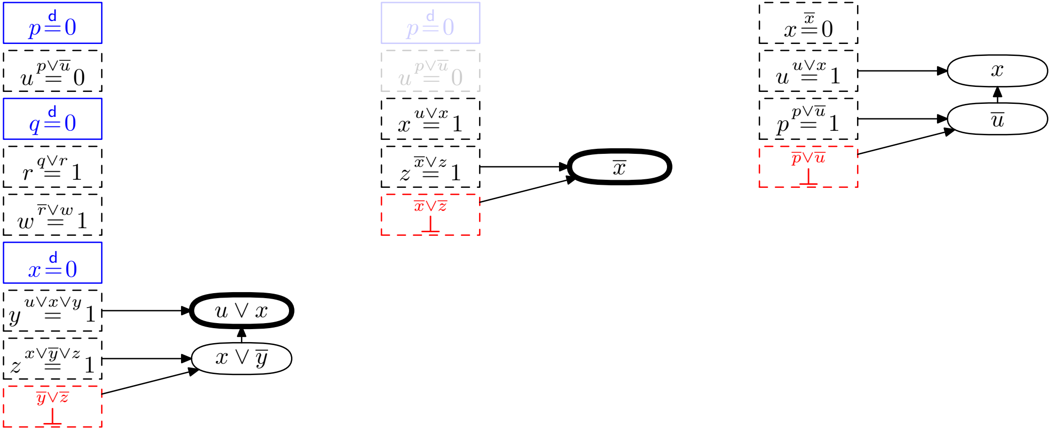
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

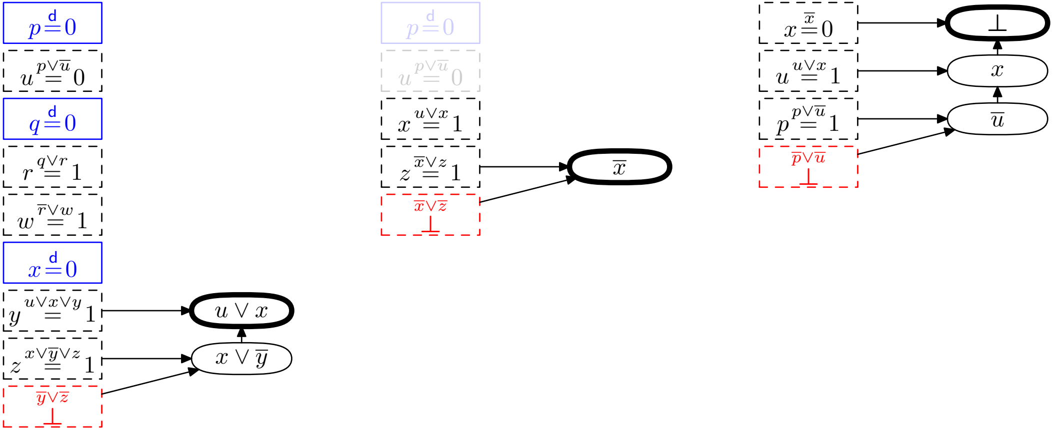
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



# CDCL Reasoning and the Resolution Proof System

To describe CDCL reasoning, need formal proof system for unsatisfiable formulas

# CDCL Reasoning and the Resolution Proof System

To describe CDCL reasoning, need formal proof system for unsatisfiable formulas

## Resolution proof system [Bla37, Rob65]

- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

$$\frac{C \lor x \qquad D \lor \overline{x}}{C \lor D}$$

- Done when contradiction $\perp$ in form of empty clause derived

# CDCL Reasoning and the Resolution Proof System

To describe CDCL reasoning, need formal proof system for unsatisfiable formulas

## Resolution proof system [Bla37, Rob65]

- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

$$\frac{C \lor x \qquad D \lor \overline{x}}{C \lor D}$$

- Done when contradiction $\perp$ in form of empty clause derived

When run on unsatisfiable formula, CDCL generates resolution proof*

# CDCL Reasoning and the Resolution Proof System

To describe CDCL reasoning, need formal proof system for unsatisfiable formulas

## Resolution proof system [Bla37, Rob65]

- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

$$\frac{C \lor x \qquad D \lor \overline{x}}{C \lor D}$$

- Done when contradiction $\perp$ in form of empty clause derived

When run on unsatisfiable formula, CDCL generates resolution proof*

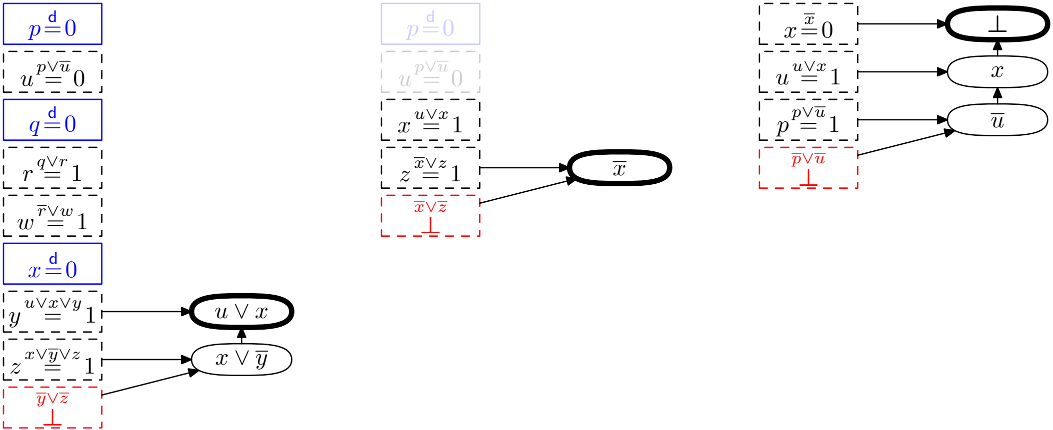
(*) Ignores pre- and inprocessing, but we don't have time to go into this level of detail

# Resolution Proofs from CDCL Executions

Obtain resolution proof. . .

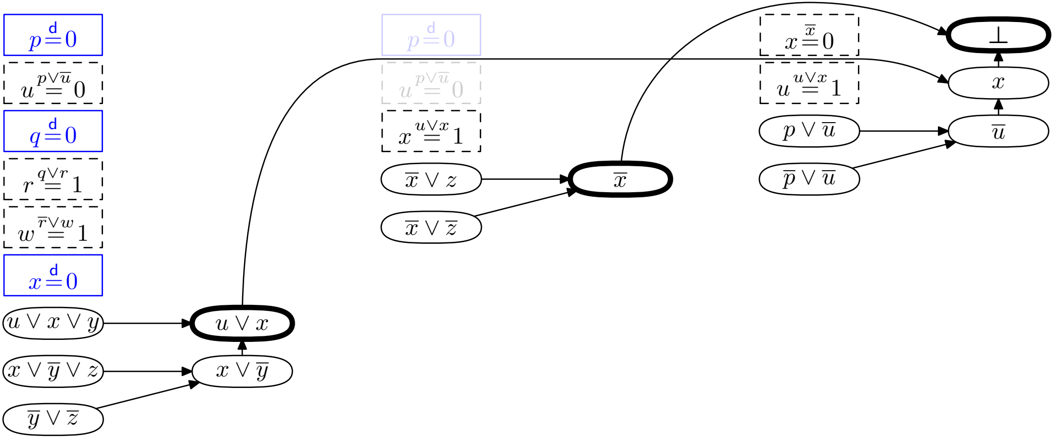
# Resolution Proofs from CDCL Executions

Obtain resolution proof from our example CDCL execution...



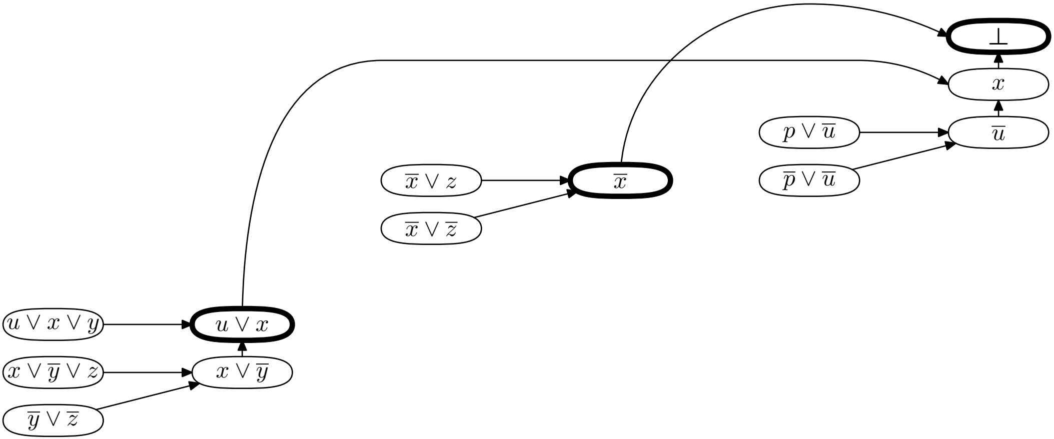
# Resolution Proofs from CDCL Executions

Obtain resolution proof from our example CDCL execution by stringing together conflict analyses:



# Resolution Proofs from CDCL Executions

Obtain resolution proof from our example CDCL execution by stringing together conflict analyses:



# RUP Proofs and CDCL

But it turns out we can be lazier. . .

**Fact**

All learned clauses generated by CDCL solver are RUP clauses

# RUP Proofs and CDCL

But it turns out we can be lazier...

**Fact**

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. $\overline{x}$
3. $\perp$

# RUP Proofs and CDCL

But it turns out we can be lazier. . .

**Fact**

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \lor x$
2. $\overline{x}$
3. $\perp$

# RUP Proofs and CDCL

But it turns out we can be lazier...

**Fact**

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \lor x$
2. $\overline{x}$
3. $\perp$

# RUP Proofs and CDCL

But it turns out we can be lazier...

**Fact**

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. $\overline{x}$
3. $\perp$

# RUP Proofs and CDCL

But it turns out we can be lazier. . .

**Fact**

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. $\overline{x}$
3. $\perp$

# RUP Proofs and CDCL

But it turns out we can be lazier...

**Fact**

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \lor x$
2. $\overline{x}$
3. $\perp$

# RUP Proofs and CDCL

But it turns out we can be lazier...

**Fact**

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. $\overline{x}$
3. $\perp$

# RUP Proofs and CDCL

But it turns out we can be lazier...

**Fact**

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \lor x$
2. $\overline{x}$
3. $\perp$

# RUP Proofs and CDCL

But it turns out we can be lazier. . .

**Fact**

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. $\overline{x}$
3. $\perp$

# RUP Proofs and CDCL

But it turns out we can be lazier. . .

**Fact**

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \lor x$
2. $\overline{x}$
3. $\perp$

# More Ingredients in Proof Logging for SAT

**Fact**

RUP proofs can be viewed as shorthand for resolution proofs

See survey chapter [BN21] for more on this and connections to SAT solving

But RUP and resolution are not enough for preprocessing, inprocessing, and some other kinds of advanced SAT solving techniques

# Extension Variables

Suppose we want a variable $a$ encoding

$$a \leftrightarrow (x \wedge y)$$

## Extended resolution [Tse68]

Resolution rule plus extension rule introducing clauses

$$a \vee \overline{x} \vee \overline{y} \qquad \overline{a} \vee x \qquad \overline{a} \vee y$$

for fresh variable $a$ (this is fine since $a$ doesn't appear anywhere previously)

# Extension Variables

Suppose we want a variable $a$ encoding

$$a \leftrightarrow (x \wedge y)$$

### Extended resolution [Tse68]

Resolution rule plus extension rule introducing clauses

$$a \vee \overline{x} \vee \overline{y} \qquad \overline{a} \vee x \qquad \overline{a} \vee y$$

for fresh variable $a$ (this is fine since $a$ doesn't appear anywhere previously)

### Fact

Extended resolution (RUP + definition of new variables) is essentially equivalent to the $\mathrm{DRAT}$ proof logging system [HHW13] most commonly used in proof logging for SAT solving

# Why Aren't We Done?

Practical limitations of current SAT proof logging technology:

- Difficulties dealing with stronger reasoning efficiently (even for SAT solving)
- Clausal proofs can't easily reflect what algorithms for other problems do

# Why Aren't We Done?

Practical limitations of current SAT proof logging technology:

- Difficulties dealing with stronger reasoning efficiently (even for SAT solving)
- Clausal proofs can't easily reflect what algorithms for other problems do

Surprising claim: Solve these problems changing format to 0-1 integer linear inequalities (a.k.a. pseudo-Boolean constraints):

$$\sum_i a_i \ell_i \geq A$$

- $a_i, A \in \mathbb{Z}$
- literals $\ell_i$: $x_i$ or $\overline{x}_i$ (where $x_i + \overline{x}_i = 1$)

# Why Aren't We Done?

Practical limitations of current SAT proof logging technology:

- Difficulties dealing with stronger reasoning efficiently (even for SAT solving)
- Clausal proofs can't easily reflect what algorithms for other problems do

Surprising claim: Solve these problems changing format to 0-1 integer linear inequalities (a.k.a. pseudo-Boolean constraints):

$$\sum_i a_i \ell_i \geq A$$

- $a_i, A \in \mathbb{Z}$
- literals $\ell_i$: $x_i$ or $\overline{x}_i$ (where $x_i + \overline{x}_i = 1$)

Use pseudo-Boolean reasoning with (extension of) the cutting planes proof system [CCT87]

# Successful Applications of Pseudo-Boolean Proof Logging

Surprisingly, pseudo-Boolean reasoning is sufficient to efficiently certify wide range of combinatorial solving techniques:

# Successful Applications of Pseudo-Boolean Proof Logging

Surprisingly, pseudo-Boolean reasoning is sufficient to efficiently certify wide range of combinatorial solving techniques:

1. Boolean satisfiability (SAT) solving including advanced techniques such as
   - Gaussian elimination [GN21]
   - symmetry breaking [BGMN23]
2. SAT-based optimization (MaxSAT) [VDB22, BBN+23, BBN+24, IOT+24]
3. (Linear) Pseudo-Boolean solving [GMNO22]
4. Subgraph solving (max clique, subgraph isomorphism, max common connected subgraph) [GMN20, GMM+20, GMM+24]
5. Dynamic programming and decision diagrams [DMM+24]
6. Presolving in $0$–$1$ integer linear programming [HOGN24]
7. Constraint programming [EGMN20, GMN22, MM23, MMN24]

# The Sales Pitch For Proof Logging

1. Certifies correctness of computed results
2. Detects errors even if due to compiler bugs, hardware failures, or cosmic rays
3. Provides debugging support [GMM+20, KM21, BBN+23, EG23, Tin24]
4. Facilitates performance analysis
5. Helps identify potential for further improvements
6. Enables auditability
7. Serves as stepping stone towards explainability

# The Sales Pitch For Proof Logging

1. Certifies correctness of computed results
2. Detects errors even if due to compiler bugs, hardware failures, or cosmic rays
3. Provides debugging support [GMM+20, KM21, BBN+23, EG23, Tin24]
4. Facilitates performance analysis
5. Helps identify potential for further improvements
6. Enables auditability
7. Serves as stepping stone towards explainability

**Opportunities for thesis projects:**

- requires mathematical maturity
- plus excellent programming skills
- quite challenging, but potential for real impact

# Pointers for Further Study

VeriPB tutorial at *CP '22* [BMN22]

- video at `youtu.be/s_5BIi4I22w`
- updated slides for *IJCAI '23* tutorial [BMN23]

Description of VeriPB for SAT 2023 competition [BMM+23]

- Available at `satcompetition.github.io/2023/checkers.html`

Specific details on different proof logging techniques covered in research papers [EGMN20, GMN20, GMM+20, GN21, GMN22, GMNO22, VDB22, BBN+23, BGMN23, MM23, BBN+24, DMM+24, GMM+24, HOGN24, IOT+24, MMN24]

Lots of concrete example files at `gitlab.com/MIAOresearch/software/VeriPB`

# Summing up

- Combinatorial solving and optimization is a true success story
- But ensuring correctness is a crucial, and not yet satisfactorily addressed, concern
- Certifying solvers producing machine-verifiable proofs of correctness seems like most promising approach
- Cutting planes reasoning with pseudo-Boolean constraints seems to hit a sweet spot between simplicity and expressivity
- **Action point:** What problems can VeriPB solve for you? ☺

# Summing up

- Combinatorial solving and optimization is a true success story
- But ensuring correctness is a crucial, and not yet satisfactorily addressed, concern
- Certifying solvers producing machine-verifiable proofs of correctness seems like most promising approach
- Cutting planes reasoning with pseudo-Boolean constraints seems to hit a sweet spot between simplicity and expressivity
- **Action point:** What problems can VeriPB solve for you? ☺

*Thank you for your attention!*

# Some Types of Pseudo-Boolean Constraints

1. Clauses

$$x_1 \vee \overline{x}_2 \vee x_3 \quad \Leftrightarrow \quad x_1 + \overline{x}_2 + x_3 \geq 1$$

2. Cardinality constraints

$$x_1 + x_2 + x_3 + x_4 \geq 2$$

3. General pseudo-Boolean constraints

$$x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7$$

# Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

**Input/model axioms**

From the input

# Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

**Input/model axioms** From the input

**Literal axioms** $$\frac{}{\ell_i \geq 0}$$

# Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

**Input/model axioms** — From the input

**Literal axioms**

$$\frac{}{\ell_i \geq 0}$$

**Addition**

$$\frac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (a_i + b_i)\ell_i \geq A + B}$$

# Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

**Input/model axioms** From the input

**Literal axioms**
$$\frac{}{\ell_i \geq 0}$$

**Addition**
$$\frac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (a_i + b_i)\ell_i \geq A + B}$$

**Multiplication** for any $c \in \mathbb{N}^+$
$$\frac{\sum_i a_i \ell_i \geq A}{\sum_i c a_i \ell_i \geq cA}$$

# Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

**Input/model axioms** — From the input

**Literal axioms**

$$\frac{}{\ell_i \geq 0}$$

**Addition**

$$\frac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (a_i + b_i)\ell_i \geq A + B}$$

**Multiplication** for any $c \in \mathbb{N}^+$

$$\frac{\sum_i a_i \ell_i \geq A}{\sum_i c a_i \ell_i \geq cA}$$

**Division** for any $c \in \mathbb{N}^+$
(assumes normalized form)

$$\frac{\sum_i a_i \ell_i \geq A}{\sum_i \left\lceil \frac{a_i}{c} \right\rceil \ell_i \geq \left\lceil \frac{A}{c} \right\rceil}$$

# Cutting Planes Toy Example

$$w + 2x + y \geq 2$$

# Cutting Planes Toy Example

Multiply by $2$ $$\frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4}$$

# Cutting Planes Toy Example

Multiply by 2 $\dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4}$ $\quad w + 2x + 4y + 2z \geq 5$

# Cutting Planes Toy Example

$$\dfrac{\dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4}\ \text{Multiply by } 2 \qquad w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9}\ \text{Add}$$

# Cutting Planes Toy Example

$$\text{Add}\ \dfrac{\text{Multiply by } 2\ \dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \qquad w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9} \qquad \overline{z} \geq 0$$

# Cutting Planes Toy Example

$$\text{Add}\ \dfrac{\text{Multiply by } 2\ \dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \qquad w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9} \qquad \dfrac{\overline{z} \geq 0}{2\overline{z} \geq 0}\ \text{Multiply by } 2$$

# Cutting Planes Toy Example

$$
\dfrac{\dfrac{\dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4}\text{ Multiply by 2} \qquad w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9}\text{ Add} \qquad \dfrac{\overline{z} \geq 0}{2\overline{z} \geq 0}\text{ Multiply by 2}}{3w + 6x + 6y + 2z + 2\overline{z} \geq 9}\text{ Add}
$$

# Cutting Planes Toy Example

$$\text{Add}\ \dfrac{\text{Add}\ \dfrac{\text{Multiply by } 2\ \dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \qquad w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9} \qquad \dfrac{\overline{z} \geq 0}{2\overline{z} \geq 0}\ \text{Multiply by } 2}{3w + 6x + 6y + 2 \qquad \geq 9}$$

# Cutting Planes Toy Example

$$
\dfrac{\dfrac{\dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4}\,\text{Multiply by } 2 \qquad w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9}\,\text{Add} \qquad \dfrac{\overline{z} \geq 0}{2\overline{z} \geq 0}\,\text{Multiply by } 2}{3w + 6x + 6y \geq 7}\,\text{Add}
$$

# Cutting Planes Toy Example

$$
\text{Divide by } 3\ \dfrac{\text{Add}\ \dfrac{\text{Add}\ \dfrac{\text{Multiply by } 2\ \dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \qquad w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9} \qquad \dfrac{\overline{z} \geq 0}{2\overline{z} \geq 0}\ \text{Multiply by } 2}{3w + 6x + 6y \geq 7}}{w + 2x + 2y \geq 2\frac{1}{3}}
$$

# Cutting Planes Toy Example

$$
\text{Add}\ \dfrac{\text{Multiply by } 2\ \dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \qquad w + 2x + 4y + 2z \geq 5}{\text{Divide by } 3\ \dfrac{\text{Add}\ \dfrac{3w + 6x + 6y + 2z \geq 9 \qquad \dfrac{\overline{z} \geq 0}{2\overline{z} \geq 0}\ \text{Multiply by } 2}{3w + 6x + 6y \geq 7}}{w + 2x + 2y \geq 3}}
$$

# Cutting Planes Toy Example

$$\dfrac{\dfrac{\dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4}\text{Multiply by } 2 \qquad w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9}\text{Add} \qquad \dfrac{\overline{z} \geq 0}{2\overline{z} \geq 0}\text{Multiply by } 2}{\dfrac{3w + 6x + 6y \geq 7}{w + 2x + 2y \geq 3}\text{Divide by } 3}\text{Add}$$

By naming constraints by integers and literal axioms by the literal involved as

$$\begin{aligned} \text{Constraint 1} &\doteq 2x + y + w \geq 2 \\ \text{Constraint 2} &\doteq 2x + 4y + 2z + w \geq 5 \\ \sim\mathbf{z} &\doteq \overline{z} \geq 0 \end{aligned}$$

# Cutting Planes Toy Example

$$\text{Divide by } 3\ \dfrac{\text{Add}\ \dfrac{\text{Add}\ \dfrac{\text{Multiply by } 2\ \dfrac{w+2x+y\geq 2}{2w+4x+2y\geq 4} \qquad w+2x+4y+2z\geq 5}{3w+6x+6y+2z\geq 9} \qquad \dfrac{\overline{z}\geq 0}{2\overline{z}\geq 0}\ \text{Multiply by } 2}{3w+6x+6y \qquad \geq 7}}{w+2x+2y\geq 3}$$

By naming constraints by integers and literal axioms by the literal involved as

$$\begin{aligned}
\text{Constraint 1} &\doteq 2x+y+w\geq 2\\
\text{Constraint 2} &\doteq 2x+4y+2z+w\geq 5\\
\sim\!\mathtt{z} &\doteq \overline{z}\geq 0
\end{aligned}$$

such a calculation is written in the proof log in reverse Polish notation as

```
pol 1 2 * 2 + ~z 2 * + 3 d
```

# Resolution and Cutting Planes

To simulate resolution step such as

$$\frac{\overline{y} \lor \overline{z} \qquad x \lor \overline{y} \lor z}{x \lor \overline{y}}$$

we can perform the cutting planes steps

$$\text{Add}\ \frac{\overline{y} + \overline{z} \geq 1 \qquad x + \overline{y} + z \geq 1}{\text{Divide by } 2\ \dfrac{x + 2\overline{y} \geq 1}{x + \overline{y} \geq 1}}$$

# Resolution and Cutting Planes

To simulate resolution step such as

$$\frac{\overline{y} \lor \overline{z} \qquad x \lor \overline{y} \lor z}{x \lor \overline{y}}$$

we can perform the cutting planes steps

$$\text{Add}\ \frac{\overline{y} + \overline{z} \geq 1 \qquad x + \overline{y} + z \geq 1}{\text{Divide by } 2\ \frac{x + 2\overline{y} \geq 1}{x + \overline{y} \geq 1}}$$

Given that the premises are clauses $7$ and $5$ in our example CNF formula, using references

$$\text{Constraint 7} \doteq \overline{y} + \overline{z} \geq 1$$

$$\text{Constraint 5} \doteq x + \overline{y} + z \geq 1$$

we can write this in the proof log as

```
pol 7 5 + 2 d
```

# Design Principles for Proof Logging

## Proof logging implementation

- Don't change solver
- Just add proof logging print statements (plus some book-keeping) to solver code

# Design Principles for Proof Logging

## Proof logging implementation

- Don't change solver
- Just add proof logging print statements (plus some book-keeping) to solver code

## Performance goals

- Proof logging overhead small constant fraction of running time ($\lessapprox 10\%$)
- Proof checking time within constant factor of solving time (current aim $\lessapprox \times 10$)

# Design Principles for Proof Logging

## Proof logging implementation

- Don't change solver
- Just add proof logging print statements (plus some book-keeping) to solver code

## Performance goals

- Proof logging overhead small constant fraction of running time ($\lesssim 10\%$)
- Proof checking time within constant factor of solving time (current aim $\lesssim \times 10$)

## Proof system

- Keep language simple — no XOR constraints, CP propagators, symmetries, ...
- But reason efficiently about such notions using power of proof system
- Combine proof logging with formally verified proof checker

# Pseudo-Boolean Proof Logging — How and Why?

If problem is (special case of) 0-1 integer linear program

- just do proof logging [basically: add print statements to solver code]

# Pseudo-Boolean Proof Logging — How and Why?

If problem is (special case of) 0-1 integer linear program

- just do proof logging [basically: add print statements to solver code]

Otherwise

- do trusted or verified translation to 0-1 ILP
- do proof logging for 0-1 ILP formulation [but solver still works with original input]

# Pseudo-Boolean Proof Logging — How and Why?

If problem is (special case of) 0-1 integer linear program

- just do proof logging [basically: add print statements to solver code]

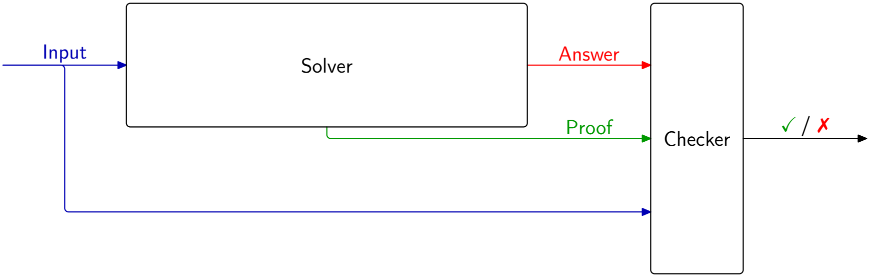
Otherwise

- do trusted or verified translation to 0-1 ILP
- do proof logging for 0-1 ILP formulation [but solver still works with original input]

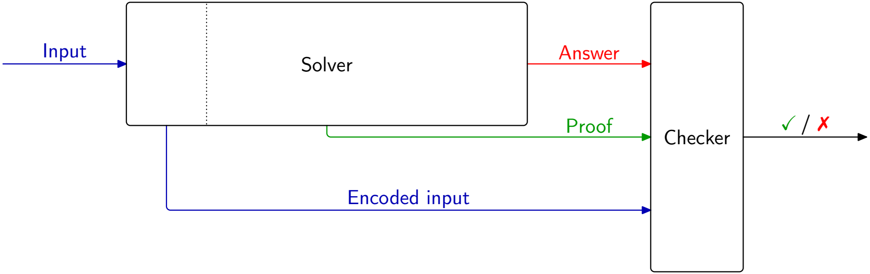
**Goldilocks compromise** between expressivity and simplicity:

1. 0-1 ILP expressive formalism for combinatorial problems (including objective)
2. Powerful reasoning capturing many combinatorial arguments

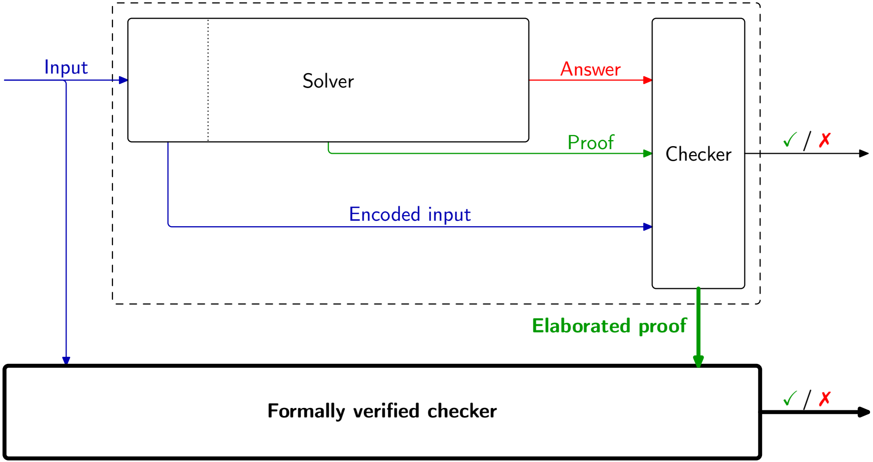
# Proof Logging with Formally Verified Checking: Full Workflow



# Proof Logging with Formally Verified Checking: Full Workflow



# Proof Logging with Formally Verified Checking: Full Workflow



# References I


[AAB+14] David Allouche, Isabelle André, Sophie Barbe, Jessica Davies, Simon de Givry, George Katsirelos, Barry O'Sullivan, Steve Prestwich, Thomas Schiex, and Seydou Traoré. Computational protein design as an optimization problem. *Artificial Intelligence*, 212(1):59–79, July 2014.

[ABFP12] Cyril Allignol, Nicolas Barnier, Pierre Flener, and Justin Pearson. Constraint programming for air traffic management: A survey. *The Knowledge Engineering Review*, 27(3):361–392, July 2012.

[ABM+11] Eyad Alkassar, Sascha Böhme, Kurt Mehlhorn, Christine Rizkallah, and Pascal Schweitzer. An introduction to certifying algorithms. *it - Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik*, 53(6):287–293, December 2011.

[AGH+22] Elvira Albert, Pablo Gordillo, Alejandro Hernández-Cerezo, Clara Rodríguez-Núñez, and Albert Rubio. Using automated reasoning techniques for enhancing the efficiency and security of (Ethereum) smart contracts. In *Proceedings of the 11th International Joint Conference on Automated Reasoning (IJCAR '22)*, volume 13385 of *Lecture Notes in Computer Science*, pages 3–7. Springer, August 2022.

[AGJ+18] Özgür Akgün, Ian P. Gent, Christopher Jefferson, Ian Miguel, and Peter Nightingale. Metamorphic testing of constraint solvers. In *Proceedings of the 24th International Conference on Principles and Practice of Constraint Programming (CP '18)*, volume 11008 of *Lecture Notes in Computer Science*, pages 727–736. Springer, August 2018.

# References II


[AGRS20] Elvira Albert, Pablo Gordillo, Albert Rubio, and Maria A. Schett. Synthesis of super-optimized smart contracts using Max-SMT. In *Proceedings of the 32nd International Conference on Computer Aided Verification (CAV '20)*, volume 12224 of *Lecture Notes in Computer Science*, pages 177–200. Springer, July 2020.

[AW13] Tobias Achterberg and Roland Wunderling. Mixed integer programming: Analyzing 12 years of progress. In Michael Jünger and Gerhard Reinelt, editors, *Facets of Combinatorial Optimization*, pages 449–481. Springer, 2013.

[BB09] Robert Brummayer and Armin Biere. Fuzzing and delta-debugging SMT solvers. In *Proceedings of the 7th International Workshop on Satisfiability Modulo Theories (SMT '09)*, pages 1–5, August 2009.

[BBN+23] Jeremias Berg, Bart Bogaerts, Jakob Nordström, Andy Oertel, and Dieter Vandesande. Certified core-guided MaxSAT solving. In *Proceedings of the 29th International Conference on Automated Deduction (CADE-29)*, volume 14132 of *Lecture Notes in Computer Science*, pages 1–22. Springer, July 2023.

[BBN+24] Jeremias Berg, Bart Bogaerts, Jakob Nordström, Andy Oertel, Tobias Paxian, and Dieter Vandesande. Certifying without loss of generality reasoning in solution-improving maximum satisfiability. In *Proceedings of the 30th International Conference on Principles and Practice of Constraint Programming (CP '24)*, volume 307 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:28, September 2024.

# References III


[BGMN23] Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified dominance and symmetry breaking for combinatorial optimisation. *Journal of Artificial Intelligence Research*, 77:1539–1589, August 2023. Preliminary version in *AAAI '22*.

[BHvMW21] Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2nd edition, February 2021.

[Bla37] Archie Blake. *Canonical Expressions in Boolean Algebra*. PhD thesis, University of Chicago, 1937.

[BLB10] Robert Brummayer, Florian Lonsing, and Armin Biere. Automated testing and debugging of SAT and QBF solvers. In *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT '10)*, volume 6175 of *Lecture Notes in Computer Science*, pages 44–57. Springer, July 2010.

[BMM+23] Bart Bogaerts, Ciaran McCreesh, Magnus O. Myreen, Jakob Nordström, Andy Oertel, and Yong Kiam Tan. Documentation of VeriPB and CakePB for the SAT competition 2023. Available at `https://satcompetition.github.io/2023/checkers.html`, March 2023.

# References IV

[BMN22] Bart Bogaerts, Ciaran McCreesh, and Jakob Nordström. Solving with provably correct results: Beyond satisfiability, and towards constraint programming. Tutorial at the *28th International Conference on Principles and Practice of Constraint Programming*. Slides available at `https://jakobnordstrom.se/presentations/`, August 2022.

[BMN23] Bart Bogaerts, Ciaran McCreesh, and Jakob Nordström. Combinatorial solving with provably correct results. Tutorial at the *32nd International Joint Conference on Artificial Intelligence*. Slides available at `https://jakobnordstrom.se/presentations/`, August 2023.

[BN21] Samuel R. Buss and Jakob Nordström. Proof complexity and SAT solving. In Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, chapter 7, pages 233–350. IOS Press, 2nd edition, February 2021. Available at `http://www.jakobnordstrom.se/publications/`.

[BR07] Robert Bixby and Edward Rothberg. Progress in computational mixed integer programming—A look back from the other side of the tipping point. *Annals of Operations Research*, 149(1):37–41, February 2007.

[BS97] Roberto J. Bayardo Jr. and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI '97)*, pages 203–208, July 1997.

# References V


[BvdKM+21] Péter Biró, Joris van de Klundert, David F. Manlove, William Pettersson, Tommy Andersson, Lisa Burnapp, Pavel Chromy, Pablo Delgado, Piotr Dworczak, Bernadette Haase, Aline Hemke, Rachel Johnson, Xenia Klimentova, Dirk Kuypers, Alessandro Nanni Costa, Bart Smeulders, Frits C. R. Spieksma, María O. Valentín, and Ana Viana. Modelling and optimisation in European kidney exchange programmes. *European Journal of Operational Research*, 291(2):447–456, June 2021.

[CCT87] William Cook, Collette Rene Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, November 1987.

[CKSW13] William Cook, Thorsten Koch, Daniel E. Steffy, and Kati Wolter. A hybrid branch-and-bound approach for exact rational mixed-integer programming. *Mathematical Programming Computation*, 5(3):305–344, September 2013.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC '71)*, pages 151–158, May 1971.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962.

# References VI


[DMB11] Leonardo De Moura and Nikolaj Bjørner. Satisfiability modulo theories: Introduction and applications. *Communications of the ACM*, 54(9):69–77, September 2011.

[DMM+24] Emir Demirović, Ciaran McCreesh, Matthew McIlree, Jakob Nordström, Andy Oertel, and Konstantin Sidorov. Pseudo-Boolean reasoning about states and transitions to certify dynamic programming and decision diagram algorithms. In *Proceedings of the 30th International Conference on Principles and Practice of Constraint Programming (CP '24)*, volume 307 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:21, September 2024.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[EG23] Leon Eifler and Ambros Gleixner. A computational status update for exact rational mixed integer programming. *Mathematical Programming*, 197(2):793–812, February 2023.

[EGMN20] Jan Elffers, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Justifying all differences using pseudo-Boolean reasoning. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI '20)*, pages 1486–1494, February 2020.

# References VII


[FFH+16] Andreas Falkner, Gerhard Friedrich, Alois Haselböck, Gottfried Schenner, and Herwig Schreiner. Twenty-five years of successful application of constraint technologies at Siemens. *AI Magazine*, 37(4):67–80, 2016.

[GMM+20] Stephan Gocht, Ross McBride, Ciaran McCreesh, Jakob Nordström, Patrick Prosser, and James Trimble. Certifying solvers for clique and maximum common (connected) subgraph problems. In *Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP '20)*, volume 12333 of *Lecture Notes in Computer Science*, pages 338–357. Springer, September 2020.

[GMM+24] Stephan Gocht, Ciaran McCreesh, Magnus O. Myreen, Jakob Nordström, Andy Oertel, and Yong Kiam Tan. End-to-end verification for subgraph solving. In *Proceedings of the 368h AAAI Conference on Artificial Intelligence (AAAI '24)*, pages 8038–8047, February 2024.

[GMN20] Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Subgraph isomorphism meets cutting planes: Solving with certified solutions. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI '20)*, pages 1134–1140, July 2020.

[GMN22] Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. An auditable constraint programming solver. In *Proceedings of the 28th International Conference on Principles and Practice of Constraint Programming (CP '22)*, volume 235 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:18, August 2022.

# References VIII


[GMNO22] Stephan Gocht, Ruben Martins, Jakob Nordström, and Andy Oertel. Certified CNF translations for pseudo-Boolean solving. In *Proceedings of the 25th International Conference on Theory and Applications of Satisfiability Testing (SAT '22)*, volume 236 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:25, August 2022.

[GN03] Evgueni Goldberg and Yakov Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '03)*, pages 886–891, March 2003.

[GN21] Stephan Gocht and Jakob Nordström. Certifying parity reasoning efficiently using pseudo-Boolean proofs. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI '21)*, pages 3768–3777, February 2021.

[GSD19] Xavier Gillard, Pierre Schaus, and Yves Deville. SolverCheck: Declarative testing of constraints. In *Proceedings of the 25th International Conference on Principles and Practice of Constraint Programming (CP '19)*, volume 11802 of *Lecture Notes in Computer Science*, pages 565–582. Springer, October 2019.

[HD19] Mark A. Hallen and Bruce Randall Donald. Protein design by provable algorithms. *Communications of the ACM*, 62(10):76–84, October 2019.

# References IX


[HHW13] Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan Wetzler. Verifying refutations with extended resolution. In *Proceedings of the 24th International Conference on Automated Deduction (CADE-24)*, volume 7898 of *Lecture Notes in Computer Science*, pages 345–359. Springer, June 2013.

[HK17] Marijn J. H. Heule and Oliver Kullmann. The science of brute force. *Communications of the ACM*, 60(8):70–79, August 2017.

[HOGN24] Alexander Hoen, Andy Oertel, Ambros Gleixner, and Jakob Nordström. Certifying MIP-based presolve reductions for $0$–$1$ integer linear programs. In *Proceedings of the 21st International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR '24)*, volume 14742 of *Lecture Notes in Computer Science*, pages 310–328. Springer, May 2024.

[IOT+24] Hannes Ihalainen, Andy Oertel, Yong Kiam Tan, Jeremias Berg, Matti Järvisalo, Magnus O. Myreen, and Jakob Nordström. Certified MaxSAT preprocessing. In *Proceedings of the 12th International Joint Conference on Automated Reasoning (IJCAR '24)*, volume 14739 of *Lecture Notes in Computer Science*, pages 396–418. Springer, July 2024.

[KB22] Daniela Kaufmann and Armin Biere. Fuzzing and delta debugging and-inverter graph verification tools. In *Proceedings of the 16th International Conference on Tests and Proofs (TAP '22)*, volume 13361 of *Lecture Notes in Computer Science*, pages 69–88. Springer, July 2022.

# References X


[KM21] Sonja Kraiczy and Ciaran McCreesh. Solving graph homomorphism and subgraph isomorphism problems faster through clique neighbourhood constraints. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI '21)*, pages 1396–1402, August 2021.

[Lev73] Leonid A. Levin. Universal sequential search problems. *Problemy peredachi informatsii*, 9(3):115–116, 1973. In Russian. Available at `http://mi.mathnet.ru/ppi914`.

[Man16] David F. Manlove. Hospitals/residents problem. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*, pages 926–930. Springer New York, 2016.

[MM23] Matthew McIlree and Ciaran McCreesh. Proof logging for smart extensional constraints. In *Proceedings of the 29th International Conference on Principles and Practice of Constraint Programming (CP '23)*, volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:17, August 2023.

[MMN24] Matthew McIlree, Ciaran McCreesh, and Jakob Nordström. Proof logging for the circuit constraint. In *Proceedings of the 21st International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR '24)*, volume 14743 of *Lecture Notes in Computer Science*, pages 38–55. Springer, May 2024.

# References XI


[MMNS11] Ross M. McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Certifying algorithms. *Computer Science Review*, 5(2):119–161, May 2011.

[MMT17] David F. Manlove, Iain McBride, and James Trimble. "Almost-stable" matchings in the hospitals / residents problem with couples. *Constraints*, 22(1):50–72, January 2017.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC '01)*, pages 530–535, June 2001.

[MO12] David F. Manlove and Gregg O'Malley. Paired and altruistic kidney donation in the UK: Algorithms and experimentation. In *Proceedings of the 11th International Symposium on Experimental Algorithms (SEA '12)*, volume 7276 of *Lecture Notes in Computer Science*, pages 271–282. Springer, June 2012.

[MS99] João P. Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, May 1999. Preliminary version in *ICCAD '96*.

[NPB22] Aina Niemetz, Mathias Preiner, and Clark W. Barrett. Murxla: A modular and highly extensible API fuzzer for SMT solvers. In *Proceedings of the 34th International Conference on Computer Aided Verification (CAV '22)*, volume 13372 of *Lecture Notes in Computer Science*, pages 92–106. Springer, August 2022.

# References XII


[PB23] Tobias Paxian and Armin Biere. Uncovering and classifying bugs in MaxSAT solvers through fuzzing and delta debugging. In *Proceedings of the 14th International Workshop on Pragmatics of SAT*, volume 3545 of *CEUR Workshop Proceedings*, pages 59–71. CEUR-WS.org, July 2023.

[PBD+22] Lise Pomiès, Céline Brouard, Harold Duruflé, Élise Maigné, Clément Carré, Louise Gody, Fulya Trösser, George Katsirelos, Brigitte Mangin, Nicolas B. Langlade, and Simon de Givry. Gene regulatory network inference methodology for genomic and transcriptomic data acquired in genetically related heterozygote individuals. *Bioinformatics*, 38(17):4127–4134, September 2022.

[Rob65] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.

[RvBW06] Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006.

[Sha09] Natarajan Shankar. Automated deduction for verification. *ACM Computing Surveys*, 41(4):20:1–20:56, October 2009.

[Tin24] Cesare Tinelli. Scalable proof production and checking in SMT. In *Proceedings of the 27th International Conference on Theory and Applications of Satisfiability Testing (SAT '24)*, volume 305 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:2, August 2024.

# References XIII


[Tse68] Grigori Tseitin. On the complexity of derivation in propositional calculus. In A. O. Silenko, editor, *Structures in Constructive Mathematics and Mathematical Logic, Part II*, pages 115–125. Consultants Bureau, New York-London, 1968.

[Van08] Allen Van Gelder. Verifying RUP proofs of propositional unsatisfiability. In *10th International Symposium on Artificial Intelligence and Mathematics (ISAIM '08)*, 2008. Available at `http://isaim2008.unl.edu/index.php?page=proceedings`.

[VDB22] Dieter Vandesande, Wolf De Wulf, and Bart Bogaerts. QMaxSATpb: A certified MaxSAT solver. In *Proceedings of the 16th International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR '22)*, volume 13416 of *Lecture Notes in Computer Science*, pages 429–442. Springer, September 2022.

[ZR14] Yang Zhao and Kristin Yvonne Rozier. Formal specification and verification of a coordination protocol for an automated air traffic control system. *Science of Computer Programming*, 96:337–353, December 2014.