Proof Complexity
Part C: Resolution and CDCL

Sam Buss

Satisfiability Boot Camp
Simons Institute, Berkeley, California
January–May 2021

Part C. discusses:

- DPLL proof search
- CDCL (Conflict Driven Clause Learning) proof search
- Proof logging; RUP and (D)RAT

- CDCL solvers can be remarkably successful in solving very large instances of SAT, routinely solving SAT instances with 100,000's or even 1,000,000's of variables.
- When CDCL solvers find an instance of SAT to be unsatisfiable, they (mostly) implicitly find a resolution refutation.
- (D)RAT extends CDCL solvers to be as strong as extended resolution; hence much stronger than resolution.
- See [Beame-Kautz-Sabharwal'04] for an introduction, or the survey "*Proof Complexity*" [B-Nordström, in preparation]

SAT Solvers (Satisfiability Solvers)

Problem: Given a set Γ of clauses representing a CNF formula, determine whether Γ is satisfiable.

CDCL SAT Solvers are built on four principal components:

- **DPLL proofs:** A depth-first search for (tree-like) resolution refutations.
- **Unit propagation** (trivial resolution) guides the DPLL search and underpins clause learning.
- **Clause learning** infers new clauses that help prune the search space.
- **Restarts** interrupt a depth-first DPLL search, and start a new DPLL search.
- and many more optimizations!

Resolution

Resolution is a refutation system, refuting sets of clauses. Thus, resolution is a system for refuting CNF formulas, equivalently, a system for proving DNF formulas are tautologies.

- A literal is a variable x or a negated variable $\overline{x}$.
- A *clause* is a set of literals, interpreted as their disjunction
- A set Γ of clauses is a CNF formula
- Resolution rule: $$\frac{x, C \qquad \overline{x}, D}{C \cup D}$$
- A **resolution refutation** of Γ is a derivation of the empty clause from clauses in Γ.
- This allows resolution to be a proof system for DNF formulas.

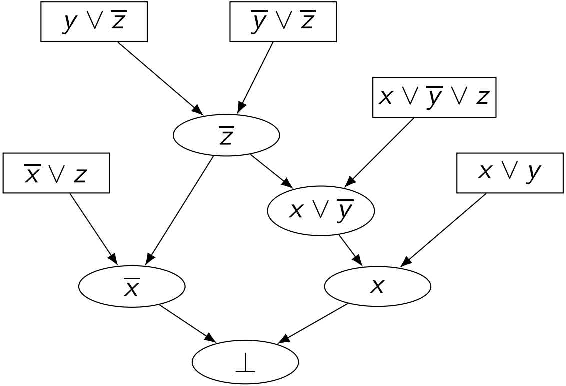
Thm: Resolution is sound and complete (for CNF refutations)

Resolution refutation — example

1.	$x \lor y$	Ax
2.	$x \lor \overline{y} \lor z$	Ax
3.	$\overline{x} \lor z$	Ax
4.	$y \lor \overline{z}$	Ax
5.	$\overline{y} \lor \overline{z}$	Ax
6.	$\overline{z}$	res
7.	$\overline{x}$	res
8.	$x \lor \overline{y}$	res
9.	x	res
10.	$\bot$	res

First five lines are axioms;

last five are inferred by resolution.



The refutation is a dag (directed cyclic graph)

It is not *regular* due to the two resolutions on y along one of the paths in the dag.

DPLL search procedure

Named after Davis-Putnam-Logemann-Loveland [DP'60, DLL'62]

Input: Γ, a set of clauses.

Goal: A satisfying assignment ρ for Γ or a refutation of Γ

The **DPLL** algorithm performs a depth-first search through the space of truth assignments, setting literals one-by-one to form a partial truth assignment ρ, backtracking when needed (namely, when some clause is falsified).

Initialization: Set ρ to be the empty assignment.

Then: Use a recursive procedure (next slide)...

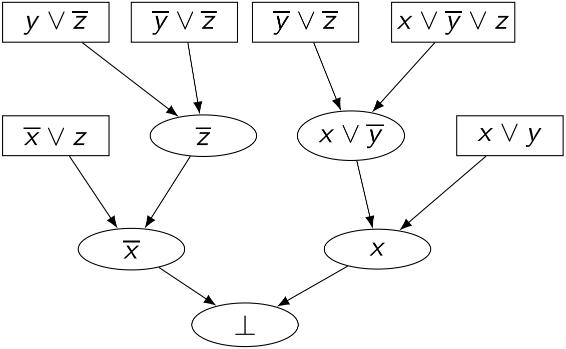
DPLL Recursive Procedure:

if *the partial assignment* ρ *falsifies some clause of* Γ **then**
  return False;
end
if ρ *satisfies* Γ **then**
  Output ρ as a satisfying assignment and terminate.
end
Pick some unset literal, x, the "*decision literal*";
Extend ρ to set x true;
Call this DPLL procedure recursively;
Update ρ to set x false;
Call this DPLL procedure recursively (again);
return False;

Either

- Terminates with a satisfying assignment, or
- Terminates with "*False*" – unsatisfiable.
  Implicitly finding a tree-like, regular proof.

A tree-like refutation from DPLL search.



Decision literals: (left-to-right, depth-first traversal)
x, $\overline{z}$, $\perp$; z, $\overline{y}$, $\perp$; y, $\perp$; $\overline{x}$, y, z, $\perp$; $\overline{z}$, $\perp$; $\overline{y}$, $\perp$; $\perp$;
"$\perp$" means, returning *False* and backtracking.

Note that the DPLL search does not need to set all variables on paths of the depth-first traversal.

Unit Propagation

- Suppose C is a clause in Γ and ρ has all but one of the literals in C false.
- Then any satisfying assignment must set the remaining literal in C true.

DPLL with UP (unit propagation): Same as the DPLL algorithm, but all possible unit propagations are carried out before choosing a decision literal. (See next slide.)

A **Unit refutation** is a resolution refutation in which each resolution inference has at least one hypothesis a unit clause.

Proposition: Γ has a unit refutation iff unit propagation finds a contradiction from Γ starting with ρ the empty assignment.

DPLL with Unit Propagation - recursive procedure

$\rho_0 \leftarrow \rho$;
Extend ρ by unit propagation for as long as possible;
if ρ *falsifies some clause of* Γ **then**
  $\rho \leftarrow \rho_0$;
  return False;
end
if ρ *satisfies* Γ **then**
  Output ρ as a satisfying assignment and terminate.
end
Pick some literal x not set by ρ (the decision literal);
Extend ρ to set x true;
Call this DPLL procedure recursively;
Update ρ to set x false;
Call this DPLL procedure recursively (again);
$\rho \leftarrow \rho_0$;
return False;

Trivial resolution

Defn' A resolution derivation of a clause D from Γ is **trivial** if

- It is an *input* refutation, i.e., every resolution inference has at least one hypothesis from Γ, and
- It is regular.

Let Γ be a set of clauses, let C be the clause $x_1 \vee \cdots \vee x_n$, and let ρ be the assignment falsifying the x_i's in C.

Theorem The following are equivalent

- There is a trivial derivation of C from Γ.
- Unit propagation with ρ and Γ yields the empty (false) clause.

Notation: This is denoted $\Gamma \vdash_1 \boldsymbol{C}$.
Or: C is inferred by **Reverse Unit Propagation (RUP)**.
Or: C is an **Asymmetric Tautology**.

The property $\Gamma \vdash_1 C$ can be checked in polynomial time (even, in linear time).

Conflict Directed Clause Learning (CDCL)

CDCL algorithms form the core of most of the modern successful SAT solvers. [Marques-Silva, Sakallah'94; MMZZM'01]

Underlying idea:

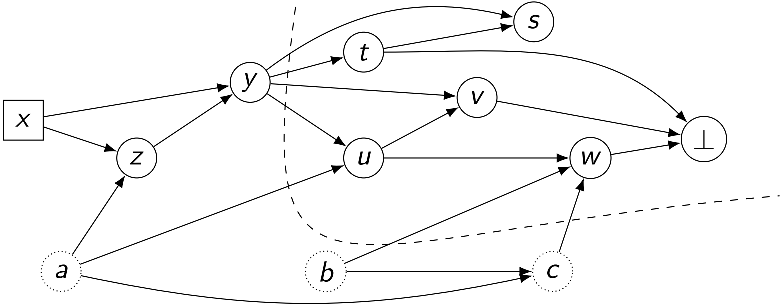
- Conflicts (falsified clauses) are found after unit propagation.
- Unit propagation gives rise to clauses that can be derived ("learned") by trivial resolution.
- These learned clauses are saved with Γ and used for future proof search.
- The learned clauses help prune the search space, in effective, reducing the need to re-traverse the same area of the search space.

An important feature is that the learned clauses help compensate for poor choices of decision literals.

Fast backtracking (backjumping) allows backtracking past decision literals that did not participate in the clause learning.

```
L ← 0 ;                                          // L is the decision level
ρ ← empty assignment;
loop
    Extend ρ by unit propagation for as long as possible;
    if ρ satisfies Γ then
        return ρ as a satisfying assignment;
    end
    if ρ falsifies some clause of Γ then
        if L == 0 then
            return "Unsatisfiable";
        end
        Optionally learn one or more clauses C and add them to Γ;
        Choose a backjumping level L′ < L;
        Unassign all literals set at levels > L′;
        L ← L′;
    else
        Pick some unset literal x (the decision literal);
        Extend ρ to set x true;
        L ← L + 1;
    end
    continue (with the next iteration of the loop);
end loop
```

Example of a conflict graph and first-UIP learning



Γ contains $\overline{x} \vee \overline{a} \vee z$, $\overline{x} \vee \overline{z} \vee y$, $\overline{y} \vee t$, $\overline{y} \vee v$, $\overline{y} \vee \overline{a} \vee u$, $\overline{y} \vee \overline{u} \vee v$, $\overline{u} \vee \overline{b} \vee \overline{c} \vee w$, $\overline{t} \vee \overline{v} \vee \overline{w}$ and $\overline{a} \vee \overline{b} \vee c$.

x is the top-level decision literal.

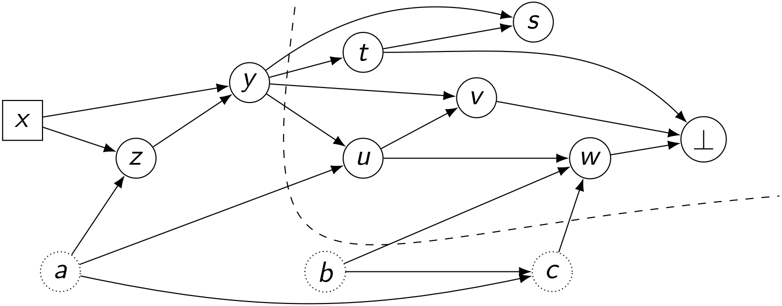
a, b, c were set at lower decision levels.

The first-UIP literal is y.

The learned clause is $\overline{\boldsymbol{a}} \vee \overline{\boldsymbol{b}} \vee \overline{\boldsymbol{c}} \vee \overline{\boldsymbol{y}}$.

(Clause minimization based on self-subsumption [Sorensson-Biere'09,Han-Somenzi'09] can learn the smaller clause $\overline{a} \vee \overline{b} \vee \overline{y}$.)

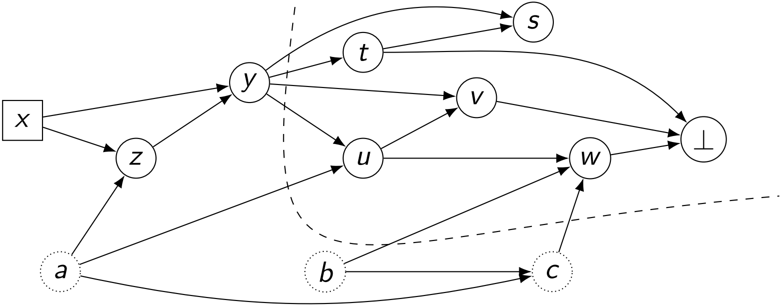
Example of a conflict graph and first-UIP learning



Γ contains $\overline{x} \vee \overline{a} \vee z$, $\overline{x} \vee \overline{z} \vee y$, $\overline{y} \vee t$, $\overline{y} \vee v$, $\overline{y} \vee \overline{a} \vee u$, $\overline{y} \vee \overline{u} \vee v$, $\overline{u} \vee \overline{b} \vee \overline{c} \vee w$, $\overline{t} \vee \overline{v} \vee \overline{w}$ and $\overline{a} \vee \overline{b} \vee c$.

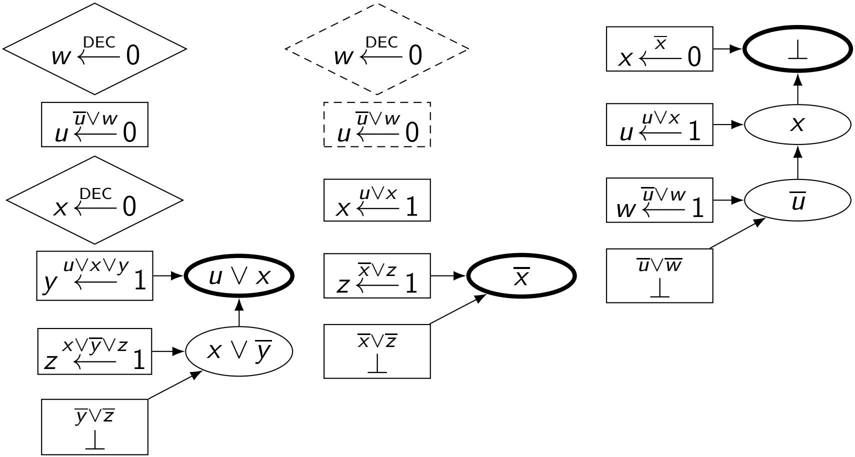
Once x, a, b, c have been set, unit propagation gives successively z, y, t, s, u, v, w, and finally $\perp$.

Example of a conflict graph and first-UIP learning



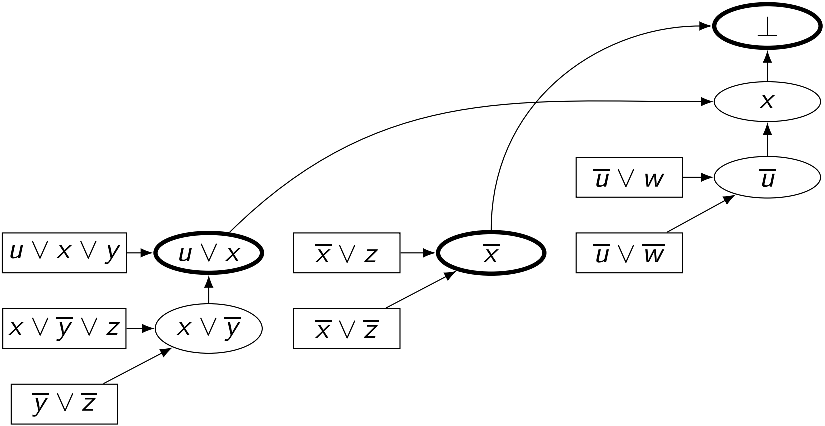
Γ contains $\overline{x} \vee \overline{a} \vee z$, $\overline{x} \vee \overline{z} \vee y$, $\overline{y} \vee t$, $\overline{y} \vee v$, $\overline{y} \vee \overline{a} \vee u$, $\overline{y} \vee \overline{u} \vee v$, $\overline{u} \vee \overline{b} \vee \overline{c} \vee w$, $\overline{t} \vee \overline{v} \vee \overline{w}$ and $\overline{a} \vee \overline{b} \vee c$.

By backtracking to the maximum decision level of a, b, c, the learned clause $\overline{a} \vee \overline{b} \vee \overline{c} \vee \overline{y}$ becomes **asserting**, allowing $\overline{y}$ to be inferred by unit propagation.
This in turn can trigger further unit propagation.



CDCL refutation of $\{(\overline{u} \vee w), (u \vee x \vee y), (x \vee \overline{y} \vee z), (\overline{y} \vee \overline{z}), (\overline{x} \vee z), (\overline{x} \vee \overline{z}), (\overline{u} \vee w), (\overline{u} \vee \overline{w})\}$.

Decision literals inside diamonds, learned clauses inside bold ovals.



The corresponding resolution refutation.

Restarts

- A **restart** backtracks the CDCL proof search back to level zero, where no decision literals have been.
- Learned clauses can be maintained after a restart.
- Perhaps surprisingly, restarts are extremely effective in the practical use of CDCL SAT solvers.

Theorem (Pipatsrisawat-Darwiche,11; Atserias-Fichte-Thurley'11; Beame-Kautz-Sabharwal'04)

CDCL + Restarts can p-simulate resolution.

The caveat for this is that the CDCL+Restarts must make the correct (nondeterministic) choices to simulate resolution. It does not mean it can be done in practice. (This is an open question.) Conversely,

Theorem

Resolution can p-simulate CDCL(+restarts).

Without restarts

Open: Can the CDCL proof search without retarts p-simulate resolution?

To formalize this open question, formalize CDCL-without-restarts as either

- Pool resolution [van Gelder'05], or
- RegWRTI [B-Hoffmann-Johannsen'08]

Pool resolution refuation: A resolution refutation that, viewed as a dag, admits a depth-first, regular traversal.

Proof Traces (Refutations from SAT solvers)

As CDCL solvers become more complicated, soundness is a serious problem. Even without "bugs", solvers use many techniques, many optimizations; they interact in subtle ways that can be unsound.

Hence: desirable for SAT solvers to output refutations that can be verified independently.

[Van Gelder'03; Goldberg-Novikov'08] Output the refutation as series of RUP clauses.

A *RUP proof* is a sequence $C_1, \ldots, C_k$ with

- $\Gamma_0 = \Gamma$ and $\Gamma_{\ell+1} = \Gamma_\ell \cup \{C_{\ell+1}\}$
- $\Gamma_\ell \vdash_1 C_{\ell+1}$
- C_k is the empty clause.

A "Deletion-RUP" (DRUP) proof allows the inclusion of deletion rules to remove clauses. This can greatly improve the verification time.

Non-implicational inferences

CDCL solvers also frequently infer clauses C that are *not* implied by Γ. For example:

Pure literal: If p appears in Γ but $\overline{p}$ does not, then infer p.

Extension rule: For a new variable x infer three new clauses expressing $x \leftrightarrow q \wedge r$:

$$\overline{q} \vee \overline{r} \vee x, \qquad q \vee \overline{x}, \qquad r \vee \overline{x}.$$

A useful way to think about these are as "wlog" inferences. [Rebola-Pardo,Suda'18]
Namely, "wlog p is true" or "wlog $x \leftrightarrow q \wedge r$ holds".

Equisatisfiability: These inferences do not change the (un)satisfiability of the set of clauses.

RAT - Resolution Asymmetric Tautology

[Heule-Hunt-Wetzler'13; s. Kullmann'99]

Definition (Resolution Asymmetric Tautology (RAT))

Let $C := C' \vee p$. Then C is RAT wrt p and Γ if, for each clause $\overline{p} \vee D'$ in Γ, the resolvent $C' \vee D'$ is an "asymmetric tautology"; i.e., $\Gamma \vdash_1 C' \vee D'$. (I.e., follows from trivial resolution)

Definition (RAT inference)

If C is RAT w.r.t. Γ, then C may be inferred by a RAT inference.

Theorem (Equisatisfiability under RAT)

In this case, Γ is satisfiable iff $\Gamma \cup \{C\}$ is satisfiable.

Proof idea: Consider the first step of the Davis-Putnam procedure (applied to p).

Proof Traces [HHW'13, WHH'14]

DRAT Proof Trace system:

DRAT (= 'D' + 'RAT') Proof Trace (Refutation) consists of a sequence of clauses updating the current set Γ of clauses with two rules:

- RAT inferences: Introduce C by RAT.
- Deletion (D): Remove any clause C.

Inferences preserve satisfiability, so the system is sound.

Often takes longer to verify refutations than generate them. (!) Deletions help prune the unit propagation search space.

THE LARGEST MATH PROOF

Resolved the Pythogorean Triples Problem (false for 7825)
DRAT proof size 200TB; compressed to 14TB (clause compression plus bzip2), then to 68GB by special encoding.
Run time: 2 days wall clock time, 37100 CPU hours.
Verification time: About 16000 CPU hours.
[Heule-Kullmann-Marek'16]

Thm: [Kullmann'99; Kiesl–Rebola-Pardo—Heule'18]
The DRAT proof system and extended resolution can p-simulate each other.

Proof idea: (For DRAT p-simulates extended resolution) The three clauses of the extension rule for $x \leftrightarrow q \wedge r$:

$$\overline{q} \vee \overline{r} \vee x, \qquad q \vee \overline{x}, \qquad r \vee \overline{x}$$

can be introduced one at a time as RAT clauses. □

Thus, DRAT provides a very strong proof system! However, it is open problem how to extend CDCL solvers to exploit the full strength of DRAT.

End of part C!