Proof Complexity II: CDCL Solvers

Sam Buss

Caleidoscope Research School
Institute Henri Poincaré, Paris
June 17-18, 2019

This talk discusses:

- **DPLL and CDCL SAT solvers.**
  - 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.
  - See [Beame-Kautz-Sabharwal'04] for an introduction.
  - Also, the survey "*Proof Complexity*" [B-Nordström, in preparation]
- **DRAT and related inference systems.**
  These extend CDCL solvers to refutations systems which are strictly stronger than resolution.

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.

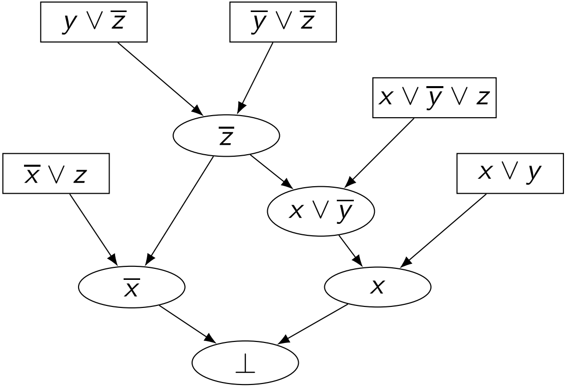
Thm: Resolution is sound and complete (for CNF refutations)

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.	$\perp$	res

First five lines are axioms;

last five 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.

Initialization: Set ρ to be the empty assignment.

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

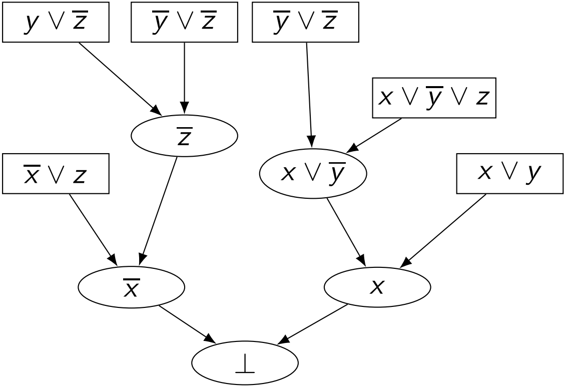
DPLL Recursive Procedure:

```
if the partial assignment $\rho$ falsifies some clause of $\Gamma$ then
    return False;
end
if $\rho$ satisfies $\Gamma$ then
    Output $\rho$ as a satisfying assignment and terminate.
end
Pick some unset literal, $x$, the "decision literal";
Extend $\rho$ to set $x$ true;
Call this DPLL procedure recursively;
Update $\rho$ 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

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

Theorem The following are equivalent

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

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

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:

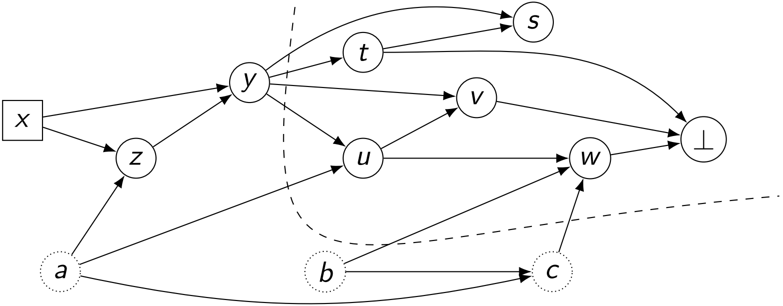
- Conflicts (falsified clauses) are found after unit propagation.
- Unit propagation gives rise to clauses that can be derived ("learned") by trivial resolution.
- These learned clauses are saved with Γ and used for future proof search.
- The learned clauses help prune the search space, in effective,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

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

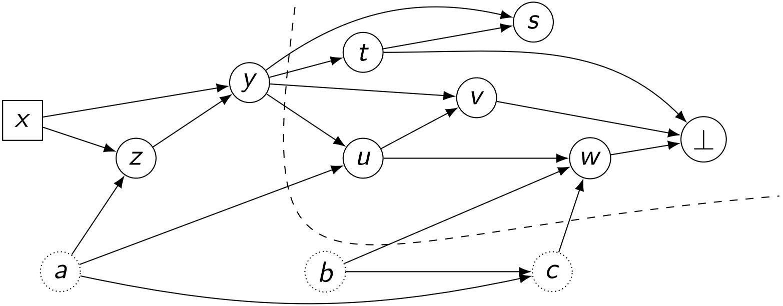
a, b, c were set at lower decision levels.

The first-UIP literal is y.

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

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

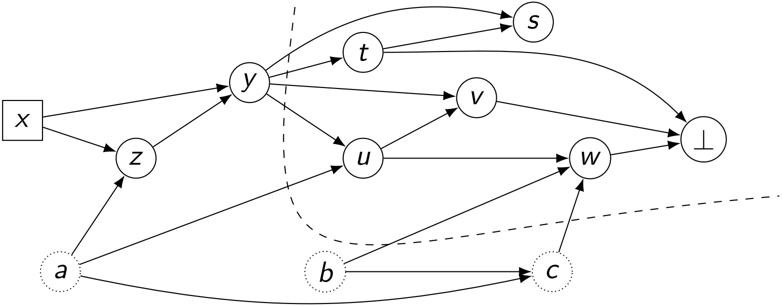
Example of a conflict graph and first-UIP learning



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 $\overline{a} \vee \overline{b} \vee c$.

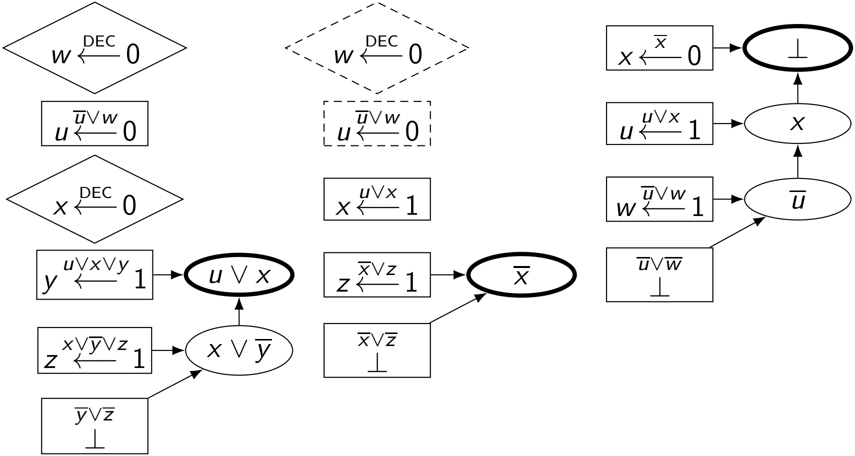
Once x, a, b, c have been set, unit propagation gives successively z, y, t, s, u, v, w, and finally $\perp$.

Example of a conflict graph and first-UIP learning

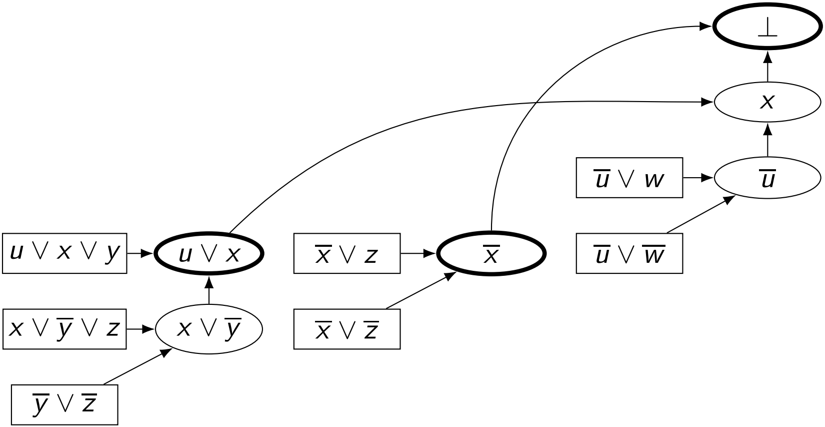


Γ contains $\overline{x} \vee \overline{a} \vee z$, $\overline{x} \vee \overline{z} \vee y$, $\overline{y} \vee t$, $\overline{y} \vee v$, $\overline{y} \vee \overline{a} \vee u$, $\overline{y} \vee \overline{u} \vee v$, $\overline{u} \vee \overline{b} \vee \overline{c} \vee w$, $\overline{t} \vee \overline{v}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


$w \xleftarrow{\text{DEC}} 0$
$u \xleftarrow{\overline{u}\vee w} 0$
$x \xleftarrow{\text{DEC}} 0$
$y \xleftarrow{u\vee x\vee y} 1$
$u \vee x$
$z \xleftarrow{x\vee\overline{y}\vee z} 1$
$x \vee \overline{y}$
$\overset{\overline{y}\vee\overline{z}}{\bot}$
$w \xleftarrow{\text{DEC}} 0$
$u \xleftarrow{\overline{u}\vee w} 0$
$x \xleftarrow{u\vee x} 1$
$z \xleftarrow{\overline{x}\vee z} 1$
$\overline{x}$
$\overset{\overline{x}\vee\overline{z}}{\bot}$
$x \xleftarrow{\overline{x}} 0$
$\bot$
$u \xleftarrow{u\vee x} 1$
x
$w \xleftarrow{\overline{u}\vee w} 1$
$\overline{u}$
$\overset{\overline{u}\vee\overline{w}}{\bot}$

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 p-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).

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.

Blocked Clauses – the inference rule

[Kullmann'99]

Definition (Blocked Clause (BC))

Let $C := C' \vee p$. Then C is BC wrt p and Γ if, for each clause $\overline{p} \vee D'$ in Γ, the resolvent $C' \vee D'$ is a tautology.

Definition (BC inference)

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

Theorem (Equisatisfiability under BC)

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).

RAT - Resolution Asymmetric Tautology

[Heule-Hunt-Wetzler'13]

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]

BC $\equiv$ RAT $\equiv$ Extended Resolution

Theorem (BC simulates ER [Kullmann'99])

An extension rule can be polynomially simulated by BC inferences. Hence also by RAT inferences.

Proof: For x new, the extension clauses are blocked w.r.t. x:

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

Theorem ([Kiesl–Rebola-Pardo–Heule'18])

Extended resolution polynomially simulates RAT proofs.

Proofs:
(1) [KRPH'18] give a direct simulation. Or
(2) The bounded arithmetic theory S^1_2 proves that RAT inferences preserve satisfiability. Thus, follows by Cook's about translations from PV to $e\mathcal{F}$.

PR, SPR - (Subset) Propagation Redundancy

For next definitions:

- Γ is a set of clauses.
- $C := p \vee C'$ is a clause.
- α is $\overline{C}$: the minimal partial assignment falsifying C.

Definition (PR - Propagation Redundant [Heule-Kiesl-Biere'17])

C is *Propagation Redundant (PR)* wrt Γ if, for some partial assignment τ,

$$\Gamma\restriction\alpha \vDash_1 (\{C\} \cup \Gamma)\restriction\tau.$$

Notation: $\Gamma \vDash_1 \Delta$ means that, for each $D \in \Delta$, $\Gamma \vDash_1 D$.

Definition (SPR - Subset Propagation Redundant [HKB'17])

C is *Subset Propagation Redundant (SPR)* wrt Γ if, it is PR with $dom(\tau) = dom(\alpha)$.

LPR $\equiv$ RAT

Recall $C := p \vee C'$ is a clause, and α is $\overline{C}$.

Definition (LPR - Literal Propagation Redundant [HKB'17])

C is *Literal Propagation Redundant (LPR)* wrt p and Γ if C is SPR and α and τ's truth values differ only on p.

Theorem (LPR $\equiv$ RAT [HKB'17])

A clause C is LPR wrt p and Γ iff it is RAT wrt p and Γ.

SR - Substitution Redundancy

Recall $C := p \vee C'$ is a clause, and α is $\overline{C}$.

Definition (SR - Substitution Redundant [B.-Thapen'19])

C is *Substitution Redundant (SR)* wrt Γ if, for some partial substitution τ,

$$\Gamma{\upharpoonright}\alpha \vDash_1 (\{C\} \cup \Gamma){\upharpoonright}\tau.$$

A **substitution** maps variables to 0 or 1 or to a literal x.

Theorem

BC, LPR/RAT, SPR, LPR, SR are increasing in applicability. They all preserve (un)satisfiability.

Proof systems

Using the inference rules BC, RAT, SPR, PR, SR, define proof systems

- Without deletion:

  BC, RAT, SPR, PR, SR

- With deletion (D):

  DBC, DRAT, DSPR, DPR, DSR.

Theorem

All of these systems are polynomially equivalent to extended resolution.

Proof: BC is the weakest, and polynmomially simulates extended resolution. Conversely, S^1_2 proves the soundness of DSR. □

Not using new variables

The strength of extended resolution depends strongly on the ability to introduce new variables.

Likewise the simulations of extended resolution by systems BC through DSR depend on the ability to introduce new variables.

However, for practical SAT solvers, we do not yet have any good hueristics for how to introduce new variables with extension.

This raises the question: **What are the power of systems such as BC, RAT, etc. when restricted to not allow new variables to be introduced?**

Notation: BC^-, RAT^-, SPR^-, PR^-, SR^- denote the systems restricted to not use new variables.

Theorem ([Kiesl-Rebola-Pardo-Heule'18])

Without new variables, DBC^- *polynomially simulates* DRAT^-.

Proof requires introducing and deleting clauses to make the "blocked" condition hold, then undoing the extra introductions and deletions. □

Theorem ([B.-Thapen'19])

Without new variables, DRAT^- *polynomially simulates* DPR^-.

Proof idea: Use one step of the Davis-Putnam procedure to eliminate the use of one variable from a PR refutation. Then use a simulation of [Heule-Biere'18]. Result is complex, but still polynomial size. □

Corollary (p-simulations without new variables)

$$\mathit{ER}^- \longrightarrow \mathit{DSR}^- \longrightarrow \mathit{DPR}^- \longleftrightarrow \mathit{DSPR}^- \longleftrightarrow \mathit{DRAT}^- \longleftrightarrow \mathit{DBC}^-$$

Short proofs without new variables

Theorem ([Heule-Kiesl-Biere'17])

The PHP_n^{n+1} *clauses have polynomial size refutations in* PR^-*.*

Theorem ([B.-Thapen'19])

The following have short proofs in SPR^- *(hence* DBC^-*):*

- *Parity principles*
- *Clique-Coloring principles*
- *Tseitin tautologies on degree* d *expander graphs*
- *Bit pigeonhole principles (Bit-PHP,* BPHP*)*
- *Or-ification and Xor-ification.*

These cover nearly all of the propositional principles for which lower bounds are known for constant depth Frege. Hence these systems without new variables are very strong.

Some lower bounds without Deletion

Theorem ([Kullmann'99])

BC^- *requires exponential size refutations for* PHP_n^{n+1}.

Theorem ([B.-Thapen'19])

RAT^- *requires exponential size refutations for* BPHP_n^{n+1}.

BPHP is the "bit" pigeonhole principle with the variables representing the bit graph of the pigeon-hole mapping This gives an exponential separation between SPR$^-$ and RAT$^-$.

Proof idea: A random restriction applied to a short RAT$^-$ refutation gives a narrow width refutation. In a narrow refutation, RAT$^-$ inferences can be replaced by narrow width resolution derivations. This is not possible for BPHP_n^{n+1} (the "bit" pigeonhole principle). □

Open Questions

- Can DSR$^-$ (without new variables) polynomially simulate extended resolution?
- [Heule-Kiesl-Seidel-Biere'17] have been able to automatically generate short refutations of the PHP using SDSL (Satisfaction-Driven Clause Learning). Can this method be made more broadly applicable?

End of second part!