Proof Complexity
CDCL and Restarts; (Semi)Algebraic Systems; Constant depth Frege

Sam Buss

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

The Satisfiability Problem

- A literal is a variable x or a negated variable $\overline{x}$.
- A *clause* is a set of literals, interpreted as their disjunction
- An instance of satisfiability (SAT-instance) is a set Γ of clauses. This represents a CNF formula.
- The ***satisfiability problem*** is the problem of either finding a satisfying assignment for Γ or showing Γ to be unsatisfiable.

One way to show unsatisfiable: Implicitly find a resolution refutation, often using a CDCL solver.

Resolution

Resolution is a refutation system for sets of clauses (or, a proof system for DNF formulas).

- Resolution rule: $\dfrac{x, C \qquad \overline{x}, D}{C \cup D}$

- A **resolution refutation** of Γ is a derivation of the empty clause $\perp$ from clauses in Γ.
- This allows resolution to be a proof system for DNF formulas.
- Resolution is sound and complete as a refutation/proof system.

Conflict-Driven Clause Learning (CDCL) Solvers

Problem: Satisfy or refute a set Γ of clauses.

CDCL SAT Solvers are built on ~~four~~ five principal components:

- **DPLL proofs:** A depth-first search for (tree-like) resolution refutations.
- **Unit propagation** guides the DPLL search and underpins clause learning.
- **Clause learning** infers new clauses that help prune the search space.
- **Backjumping** (Nonchronological backtracking) backs up the depth first traversal to where a learned clause is asserting.
- **Restarts** interrupt a depth-first DPLL search, and start a new DPLL search.
- and many more optimizations!

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).

Unit Propagation

Unit Propagation (UP)

- Suppose C is a clause in Γ, and ρ has all but one of the literals in C false.
- Then any satisfying assignment extending ρ 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.

DPLL+UP uses a recursive procedure.
- Initialize ρ to the empty partial truth assignment.
- Then call the recursive procedure (next slide)

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+UP procedure recursively;
Update ρ to set x false;
Call this DPLL+UP procedure recursively (again);
$\rho \leftarrow \rho_0$;
return False;

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 effect 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);
        L ← L + 1;
        Extend ρ to set x true;
    end
    continue (with the next iteration of the loop);
end loop
```

Learned clauses: Typically, the following holds.

- There is a single learned clause C, a **first-UIP clause**.
- We have $\boldsymbol{\Gamma \vdash_1 C}$; namely, letting σ be the assignment falsifying the literals in C,

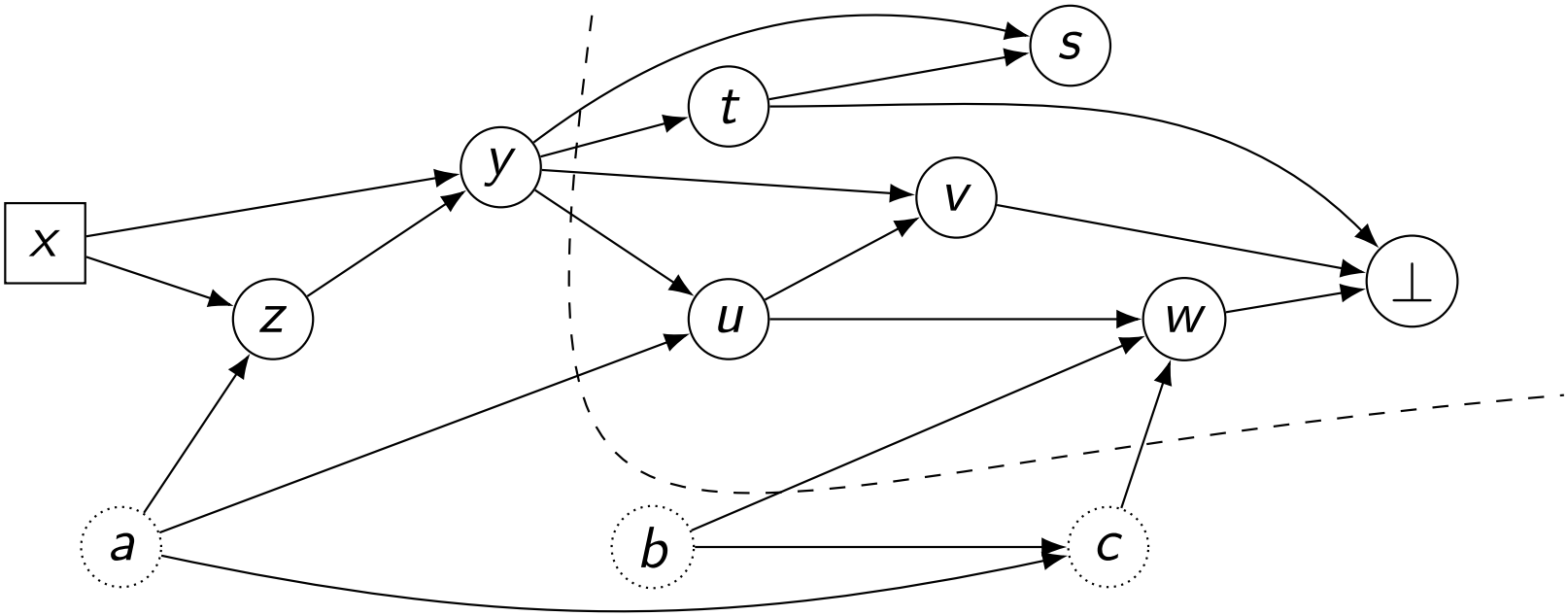
$$\Gamma \restriction \sigma \text{ implies } \bot \text{ by unit propagation.}$$

"C is inferred by *reverse unit propagation* (RUP)" or "C is an *asymmetric tautology* (AT)".

- C is **asserting**: All literals are false in the current partial assignment ρ (before backtracking). One literal is set at some level ℓ; remaining are set at lower levels, namely at $\ell' < \ell$ and lower.

Backtracking must return to level $L' < \ell \leq L$.
Common choice is to backtrack to level $L' = \ell'$.
Called **"fast backtracking"** if $L' < L-1$.

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.

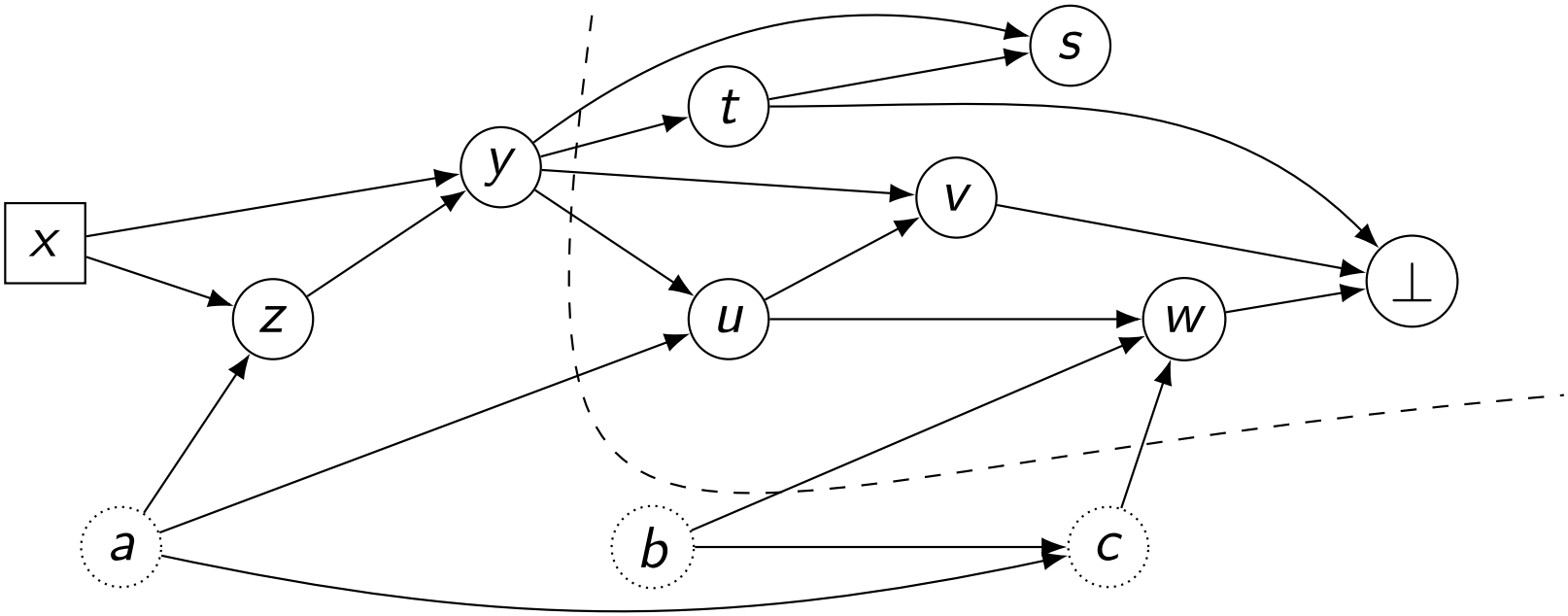
a, b, c were set at lower decision levels.

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

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.

This can force "fast backtracking" to a level $L' < L-1$.

```
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
        Learn an asserting clause (and possibly other clauses); add them to Γ;
        Choose a backjumping level L′ < L compatible with the asserting clause;
        Unassign all literals set at levels > L′;
        L ← L′;
    else
        Pick some unset literal x (the decision literal);
        L ← L + 1;
        Extend ρ to set x true;
    end
    continue (with the next iteration of the loop);
end loop
```

Restarts

- A **restart** backtracks the CDCL proof search back to level zero ($L' = 0$), 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

Resolution can p-simulate CDCL+restarts.

Proof idea: All learned clauses are inferred by Reverse Unit Propagation, hence can be derived by resolution. □

The converse holds too...

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.
Indeed, if $\mathrm{P} \neq \mathrm{NP}$, then resolution is not automatizable. [Atserias-Müller'19].

Proof on next slides ...

Proof (sketch). Suppose $\mathcal{R}$ is a resolution refutation of Γ.
$\mathcal{R}$ contains the clauses $C_1, C_2, C_3, \ldots, C_m = \bot$.
We want to build a CDCL+restarts refutation of Γ.

- The idea is to let the CDCL algorithm learn successively each clause C_i.
- However, we do not know how to do this.
- Instead we arrange that the CDCL algorithm learns clauses enlarging Γ, so that, successively for each C_i, we have

$$\text{first, } \Gamma \vdash_1 C_i \quad \text{and second, } \Gamma \textit{ absorbs } C_i.$$

Def'n: Γ **absorbs** $C = x_1 \vee x_2 \vee \cdots \vee x_k$ provided:
For each x_i, setting all x_j, $j \neq i$, to false yields a unit propagation derivation (from Γ) of either x_i or $\bot$.

Lemma 1:

- If $C \in \Gamma$, then $\Gamma \vdash_1 C$ and Γ absorbs C.
- If Γ absorbs both $x \vee C$ and $\overline{x} \vee D$, then $\Gamma \vdash_1 C \vee D$.

Pf: These are easy to check. $\square$

We cannot conclude however that $C \vee D$ is absorbed (since C and D may have a common literal). But instead:

Lemma 2: If $\Gamma \vdash_1 C$, then CDCL can learn $\Gamma^* \supset \Gamma$ such that Γ^* absorbs C.

Proof on next slide ...

Let $C = x_1 \vee x_2 \vee \cdots \vee x_k$. If C is not absorbed w.r.t. x_k, choose the decision literals (always!) in the order $\overline{x}_1, \ldots, \overline{x}_k$ and run CDCL — learning (and enlarging Γ) and backtracking as usual. (But skip any decision literal $\overline{x}_j$ that is already set to that polarity.)

- If after setting $\overline{x}_1, \ldots, \overline{x}_i$ where $i < k$, UP infers $\bot$ or x_k, or x_j for $j < k$, we are done. C has been absorbed w.r.t. x_k.
- Otherwise, since $\Gamma \vdash_1 C$, a conflict occurs. Some literal z is asserted in the learned clause, with z not an x_j or $\overline{x}_j$ for $j < k$. z is a UP-consequence of $\overline{x}_1, \ldots \overline{x}_{k-1}$.
- The newly asserted literals must be distinct, hence the last case can happen only once per variable in Γ.

The above procedure is repeated for each literal in C until C is absorbed w.r.t. each x_i. $\square$
Q.E.D.

Without restarts

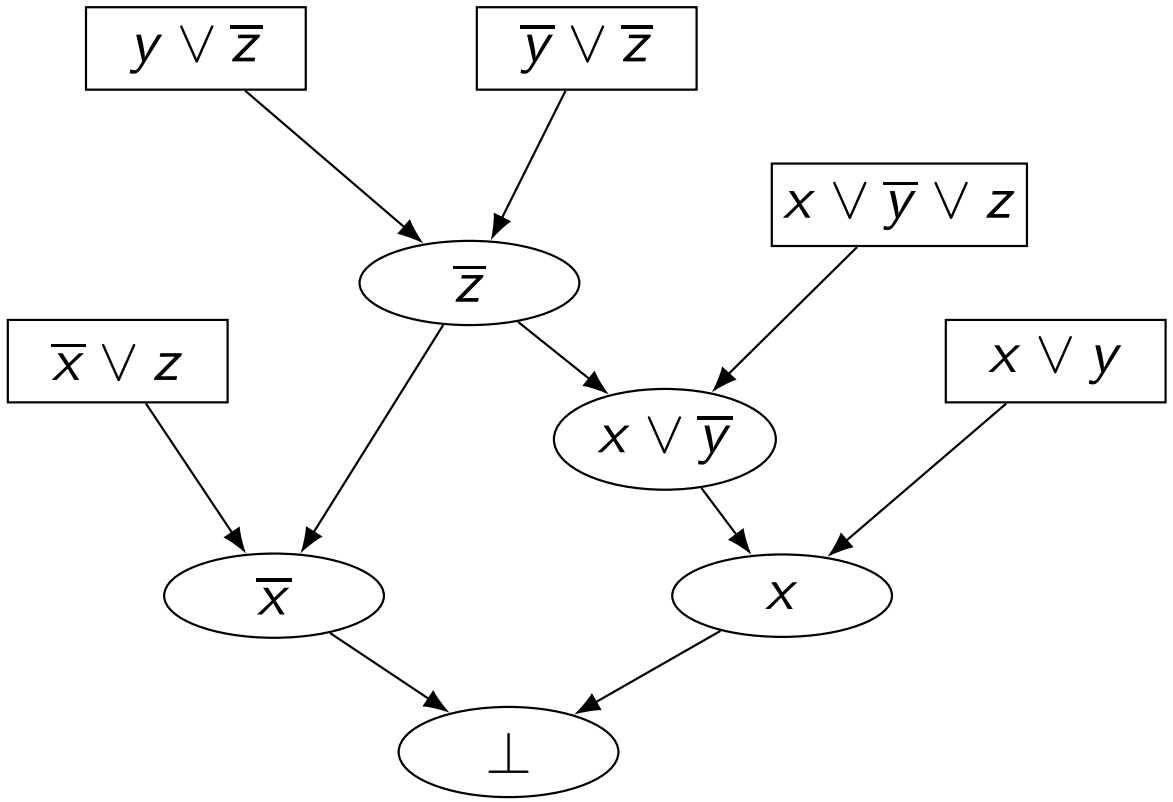
Open: Can the CDCL proof search without retarts p-simulate resolution?

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

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

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

Pool resolution refutation — example



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

However, it has a regular depth-first traversal, hence it is a pool resolution refutation. The point is that the first time $\overline{z}$ is traversed, it becomes learned and does not need to be re-traversed.

However: Modern CDCL solvers infer clauses that do not fit within either Pool resolution or RegWRTI.

- A ***self-subsumption*** inference is a resolution inference where the conclusion is a subclause of one of the hypotheses.
- Self-subsumption inferences are often done during clause learning.
- As implemented in CDCL solvers, self-subsumption is performed on clauses in which all literals have been assigned value.

Question: Does CDCL without restarts, but with self-subsuming resolution on clauses which participated in unit propagation, polynomially simulate resolution?
What if "self-subsuming" is dropped?

CDCL w/o restarts effectively p-simulates resolution

For an "effective p-simulation", it is allowed to transform a formula F; in our case, by disjoining it with a satisfiable formula over different variables.

Construction:

- Given a CNF formula $F(\vec{x})$ in n variables $\vec{x}$, with a resolution refutation of size k.
- Find a suitable satisfiable CNF formula $G_n(\vec{y})$. The formula $G_n(\vec{y})$ depends only on n, not F or k.
- Show that $F(\vec{x}) \vee G_n(\vec{y})$ has a CDCL refutation without restarts, of size $k^{O(1)}$.

This construction works for CDCL solvers that use the usual methods of asserting learning and backtracking.

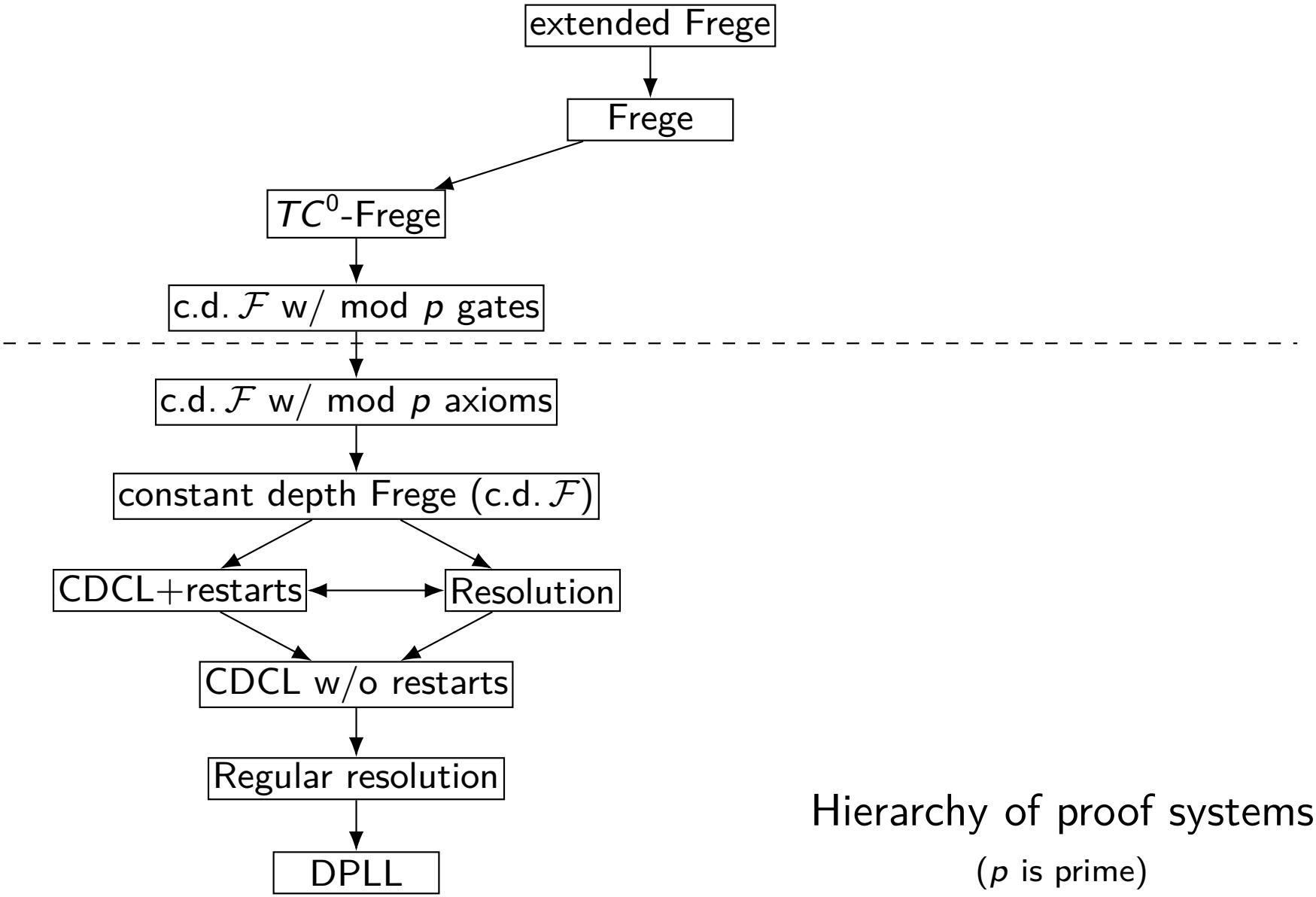
[Beame-Sabharwal'14] building on
[Hertel-Bachus-Pitassi-van Gelder'08; B-Hoffmann-Johannsen'08]

Construction for the effective p-simulation:

- A CDCL+restarts solver $\mathcal{S}_1$ refutes $F(\vec{x})$ using an execution $\mathcal{E}_1$ with at most k^c restarts.
- W.l.o.g. $k \leq 2^{2n}$. Choose $G_n(\vec{y})$ to be satisfiable such that there is an execution $\mathcal{E}_2$ of the CDCL solver $\mathcal{S}_2$ finding G_n to be satisfiable that fast-backtracks $> 2^{2cn}$ many times.
  C_2 does not use restarts.
- Then a CDCL refutation without restarts of $F \vee G_n$ repeats the following loop:
  - Continue running the execution $\mathcal{E}_2$ of $\mathcal{S}_2$ on G_n until just before a fast-backtrack.
  - Continue running the execution $\mathcal{E}_1$ of $\mathcal{S}_1$ on F until it needs to restart.
  - Run one more step of $\mathcal{E}_2$ on G_n causing a fast-backtrack, which effectively backtracks the execution $\mathcal{E}_1$ of $\mathcal{S}_1$ back to its decision level 0.

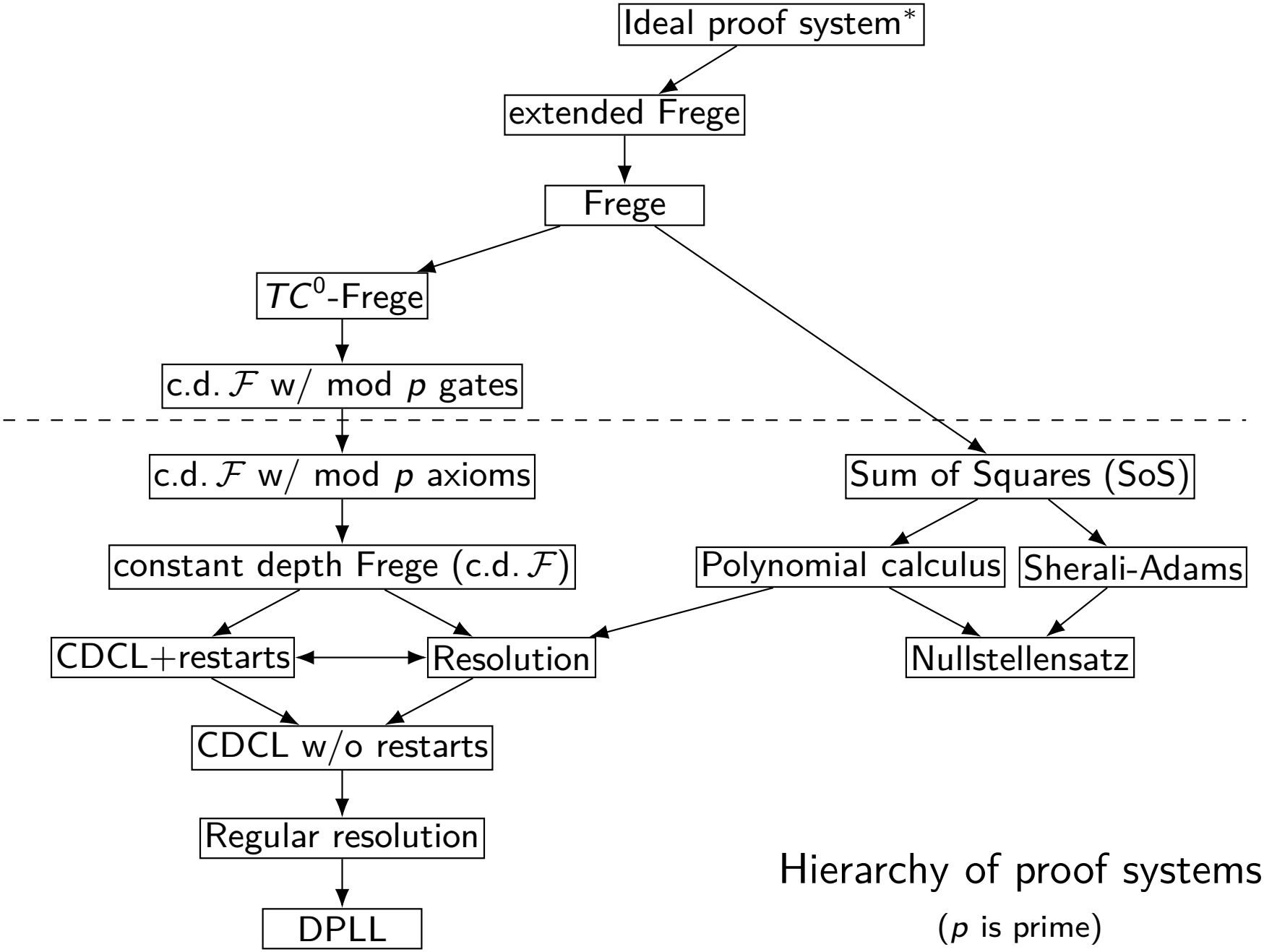
Moral: Fast-backtracking has a lot in common with restarts.

II. Algebra and Semialgebraic Proof Systems



Hierarchy of proof systems

(p is prime)



Hierarchy of proof systems

(p is prime)

Definitions of the algebraic proof systems (NS and PC)

Work over a field (optionally finite).
Variables $x_1, x_2, \ldots$ are 0/1 valued.
Identify 0 with *True* and 1 with *False*.

A polynomial f is identified with the assertion $f = 0$.

Example: A clause $x \vee \overline{y} \vee z$ can be expressed with the polynomial

$$(1 - x) \cdot y \cdot (1 - z)$$

An ***algebraic*** refutation of a set of polynomials f_j shows that the f_j's cannot be simultaneously given value zero by finding a polynomial identity

$$\sum_j f_j \cdot g_j + \sum_i (x_i^2 - x_i) \cdot h_i = 1.$$

The use of $(x_i^2 - x_i)$ is justified since only Boolean (0/1) values are allowed for variables.

A **Nullstellensatz** refutation is given by explicitly writing out the polynomials g_j and h_i such that

$$\sum_j f_j \cdot g_j + \sum_i (x_i^2 - x_i) \cdot h_i = 1.$$

A **Polynomial Calculus** refutation uses inference steps: initial formulas are the f_j's and $(x_i^2 - x_i)$'s; the inference rules are addition and multiplication:

$$\frac{f \quad g}{f + g} \qquad\qquad \frac{f}{f \cdot g}$$

The final line of a polynomial calculus refutation is the polynomial 1.

Nullstellensatz is called a "static" proof system. The polynomial calculus is called a "dynamic" proof system.

The polynomial calculus and the nullstellensatz systems can refute the same sets of polynomials; but a polynomial calculus can be substantially shorter, or have lower degree, due to cancellation of monomials in intermediate steps.

The semialgebraic proof systems (S.A. and SoS)

Work in an ordered field.
Variables $x_1, x_2, \ldots$ are 0/1 valued.
Identify 0 with *True* and 1 with *False*.
We now work with polynomial *inequalities* $f \geq 0$.
Example: A clause $x \vee \overline{y} \vee z$ can be expressed with the inequality

$$x - y + z \geq 0.$$

A set Γ of clauses expresses a set of polynomial inequalities of the forms $f_j \geq 0$ and $x_i^2 - x_i \geq 0$ and $x_i - x_i^2 \geq 0$.
A **semialgebraic** refutation of Γ gives (or proves there exists) a polynomial identity

$$\sum_k p_k \cdot r_k \;=\; -1,$$

where the polynomials p_k range over the f_j's, the $(x_i^2 - x_i)$'s and the $(x_i - x_i^2)$'s, and where the r_k's are polynomials which are known to be nonnegative for Boolean inputs.

The **Sherali-Adams** refutation system (static version) gives a polynomial identity

$$\sum_k p_k \cdot r_k \;=\; -1,$$

where the r_k's which are equal to products of terms of the form x_i and $(1 - x_i)$.
For example, r_k could be $(1 - x_1)x_2(1 - x_3)$.
(This is sometimes called a "non-negative junta".)

The **Sum of Squares** refutation gives the same kind of polynomial identity, but with the r_k's equal to squares, that is, $r_k = s_k^2$ for some polynomial s_k.

Both systems can also be stated in dynamic form as well.

Ideal Proof System (IPS)

The ideal proof system is an algebraic proof system.
Again, work in a field and identify 0 with *True* and 1 with *False*.

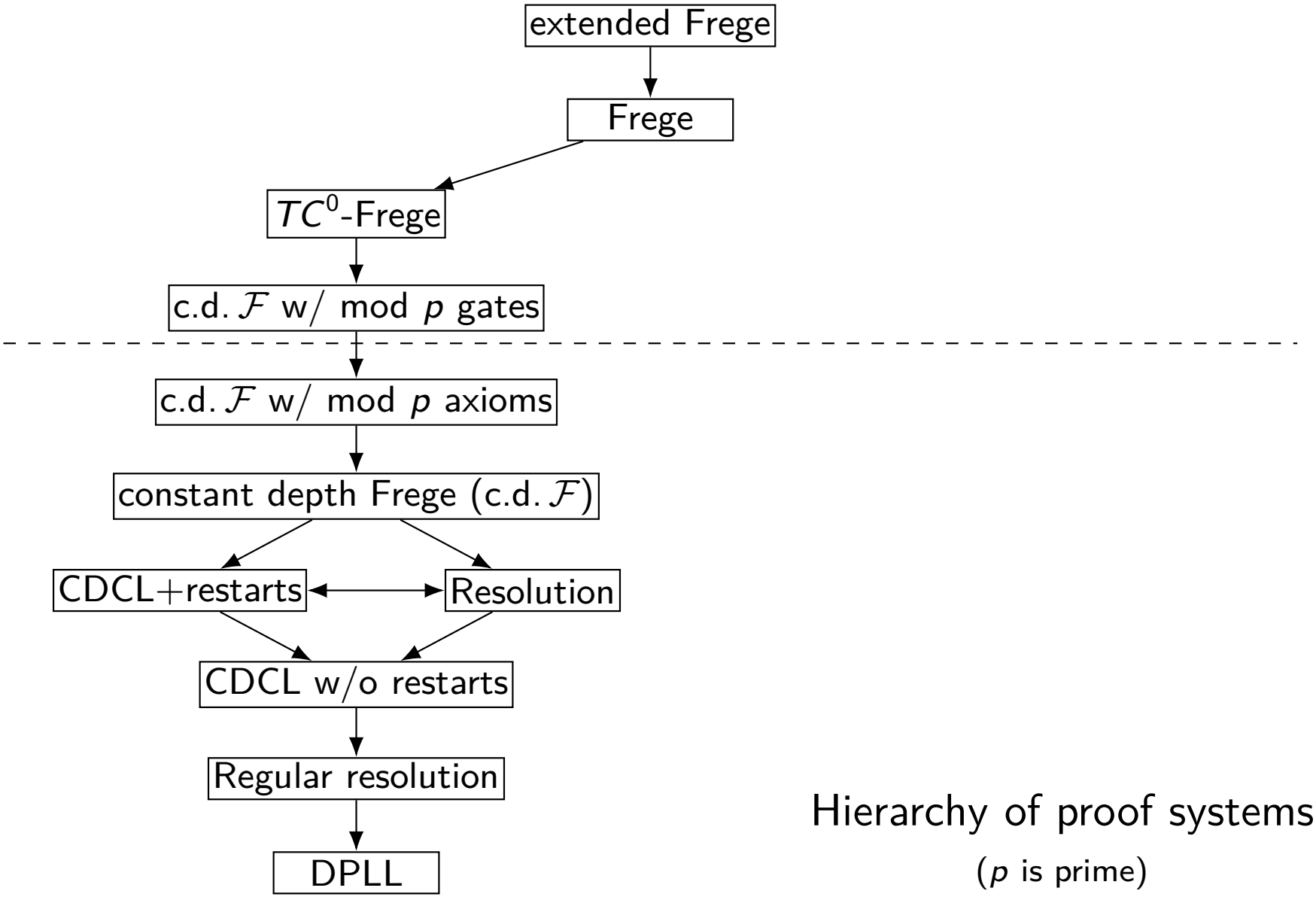
Let f_i be a set of polynomials representing an unsatisfiable set Γ of clauses. Here $f_i = 0$ expresses that the i-th clause is true.
Let the polynomials p_k range over the polynomials f_i and the polynomials $(x_i^2 - x_i)$.

A **Ideal Proof System (IPS)** refutation of Γ is an algebraic circuit $C(\vec{u}, \vec{v})$ so that the two identities hold:

- $C(\vec{u}, \vec{v}) = 0$, and
- $C(\vec{u}, \vec{p}) = 1$.

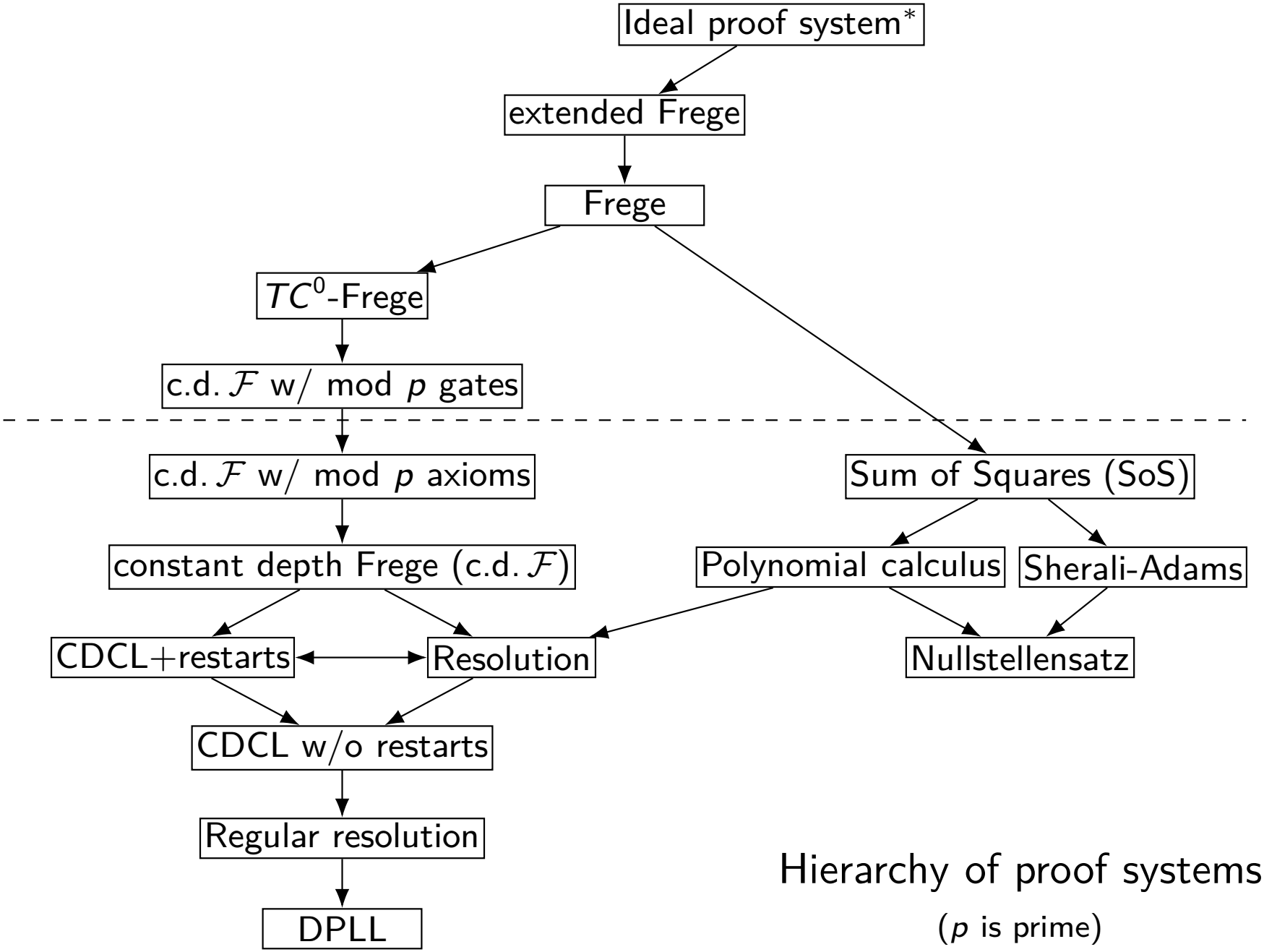
Caveat: IPS is not quite a proper proof system (as far as we know), as testing algebraic circuit identities is only known in to be randomized polynomial time (RP) instead of in deterministic polynomial time (P).

III. Two open problems at frontier of proof complexity



Hierarchy of proof systems

(p is prime)



Hierarchy of proof systems

(p is prime)

Constant depth Frege separations?

The ***depth*** of a propositional formula is the number of alternating blocks of $\vee$'s and $\wedge$'s (assume wlog that negations are pushed to the atoms).

A ***depth*** d Frege proof means a Frege proof in which all formulas have depth d.

[BIKPPW'92,PBI'93,KPW'95;Ajtai'88]: Exponential lower bounds for constant depth Frege proofs of the pigeonhole principle.

[BDGMP'04] Conditional non-automatizability of constant-depth Frege, based on hardness of Diffie-Hellmann.

Open problem: Give an exponential separation (or even a super-quasipolynomial separation) between depth d and depth $d+1$ Frege proofs refuting sets of clauses.

[Krajíček'19, Impagliazzo-Krajíček'02] gives a super-polynomial separation.

Constant depth Frege has exponential speedup over resolution for the weak pigeonhole principle.

Note that, from Håstad's switching lemma, depth $d+1$ formulas can give exponential speedup over depth d formulas, so depth $d+1$ Frege has more expressive power than depth d Frege

Mod-p Frege systems

Fix a prime p.
A $\oplus_p$ gate (unbounded fanin) outputs 1 iff the number of true inputs is 0 mod p.

Open problem: Give exponential lower bounds on depth d Frege proofs over the propositional language $\neg$, $\vee$, $\wedge$ and $\oplus_p$.

Notation: $AC^0(\oplus_p)$-Frege.

Suggestion: For a prime $q \neq p$, are there subexponential size $AC^0(\oplus_p)$-Frege proofs of the Mod-q Counting Principle?

By [Razborov'87, Smolensky'87], $AC^0(\oplus_p)$ circuits for counting modulo q require exponential size.

The **Mod-q Counting Principle** generalizes the Parity Principle for $q \neq 2$.

Defn: (Mod-q Counting Principle). For n not a multiple of q, there is no partition of $[n]$ into sets of size q.

- Variables x_S for each $S \subset [n]$ such that $|S| = q$.
- Clauses:
  - $\bigvee_{i \in S} x_S$, for each $i \in [n]$. And
  - $\overline{x}_S \vee \overline{x}_{S'}$, for each S, S' such that $0 < |S \cap S'| < q$.

Thank you!