Proof Complexity

Olaf Beyersdorff

School of Computing
University of Leeds, UK

Outline of this tutorial

Tour of proof systems

- Resolution
- Frege and beyond
- Cutting Planes
- ...

Relations to other areas

- Separation of complexity classes
- Analysis of SAT algorithms
- Proof search – Automatizability
- First-Order Logic – Bounded Arithmetic
- Further topics

A Tour of Proof Systems

Proof Systems

Definition (Cook, Reckhow 79)

A proof system for a language L is a function f with $rng(f) = L$.
If $f(w) = x$, then w is called an f-proof of $x \in L$.

- correctness: $rng(f) \subseteq L$
- completeness: $L \subseteq rng(f)$
- efficiency: proofs should be easy to check,
  i.e. f should be easy to compute.

- Most research in proof complexity has studied propositional proof systems where $L = TAUT$.

A First Example: Truth Tables

A proof system for TAUT

$$TT(\alpha, \varphi) = \begin{cases} \varphi & \text{if } \alpha \text{ is a truth table for } \varphi \text{ with all entries 1} \\ p \vee \neg p & \text{otherwise.} \end{cases}$$

Why is this not a good proof system?

- Most proofs are exponentially long in the size of the formula.

A First Example: Truth Tables

A proof system for TAUT

$$TT(\alpha, \varphi) = \begin{cases} \varphi & \text{if } \alpha \text{ is a truth table for } \varphi \text{ with all entries 1} \\ p \vee \neg p & \text{otherwise.} \end{cases}$$

Why is this not a good proof system?

- Most proofs are exponentially long in the size of the formula.
- We look for proof systems with shorter proofs.

The Most Studied Proof System: Resolution

- Introduced by Blake 1937, Davis & Putnam 1960, and Robinson 1965
- Resolution proofs operate with clauses.
- Refutation system
- only one rule

$$\frac{C \vee p \qquad D \vee \neg p}{C \vee D}$$

- many subsystems studied: tree-like, regular ...

Complexity of Resolution

First historical lower bound:

- Pigeonhole principle: $n+1$ pigeons cannot sit in n holes
- CNF formulation PHP_n^{n+1}

$$\bigvee_{j\in[n]} x_{i,j} \qquad \text{for all pigeons } i \in [n+1]$$

$$\neg x_{i_1,j} \vee \neg x_{i_2,j} \qquad \text{for all distinct } i_1, i_2 \in [n+1] \text{ and } j \in [n]$$

- PHP_n^{n+1} requires Resolution refutations of size $2^{\Omega(n)}$. [Haken 85]

Many strong lower bounds

- Combinatorial principles: ordering principle, ...
- Graph-theoretic principles: Tseitin formulas, pebbling ...
- Random 3-CNF's are hard for Resolution. [Beame et al. 98]

A Strong System: Frege

Axioms

$$p_1 \to (p_2 \to p_1)$$
$$(p_1 \to p_2) \to (p_1 \to (p_2 \to p_3)) \to (p_1 \to p_3)$$
$$p_1 \to p_1 \vee p_2$$
$$p_2 \to p_1 \vee p_2$$
$$(p_1 \to p_3) \to (p_2 \to p_3) \to (p_1 \vee p_2 \to p_3)$$
$$(p_1 \to p_2) \to (p_1 \to \neg p_2) \to \neg p_1$$
$$\neg\neg p_1 \to p_1$$
$$p_1 \wedge p_2 \to p_1$$
$$p_1 \wedge p_2 \to p_2$$
$$p_1 \to p_2 \to p_1 \wedge p_2$$

Modus Ponens

$$\frac{p_1 \qquad p_1 \to p_2}{p_2}$$

Frege Proofs

A Frege proof of a formula φ is a sequence

$$(\varphi_1, \ldots, \varphi_n = \varphi)$$

of propositional formulas such that for $i = 1, \ldots, n$:

- φ_i is a substitution instance of an axiom, or
- φ_i was derived by modus ponens from φ_j, φ_k with $j, k < i$.

Frege Proofs

A Frege proof of a formula φ is a sequence

$$(\varphi_1, \ldots, \varphi_n = \varphi)$$

of propositional formulas such that for $i = 1, \ldots, n$:

- φ_i is a substitution instance of an axiom, or
- φ_i was derived by modus ponens from φ_j, φ_k with $j, k < i$.

Major open problem

Show non-trivial lower bounds on the size of Frege proofs.

Restrictions and Extensions of Frege Systems

Bounded-depth Frege

Allow only formulas of logical depth d in the proof for a given constant d.

Extended Frege *EF*

Abbreviations for complex formulas: $p \equiv \varphi$,
where p is a new propositional variable.

Frege systems with substitution *SF*

Substitution rule: $\dfrac{\varphi}{\sigma(\varphi)}$

for arbitrary substitutions σ

Extensions of *EF*

Let Φ be a polynomial-time computable set of tautologies.
$EF + \Phi$: Φ as axiom schemes

Reductions between Proof Systems

Definition (Cook, Reckhow 79, Krajíček, Pudlák 89)

Let f and g be proof systems for L.

- f simulates g, if for any g-proof w there is an f-proof w' of length $|w'| = |w|^{O(1)}$ s.t. $f(w') = g(w)$.
- If w' is computable from w in polynomial time, then f p-simulates g.
- f and g are (p-)equivalent if they (p-)simulate each other.

Reductions between Proof Systems

Definition (Cook, Reckhow 79, Krajíček, Pudlák 89)

Let f and g be proof systems for L.

- f simulates g, if for any g-proof w there is an f-proof w' of length $|w'| = |w|^{O(1)}$ s.t. $f(w') = g(w)$.
- If w' is computable from w in polynomial time, then f p-simulates g.
- f and g are (p-)equivalent if they (p-)simulate each other.

Definition (Krajíček, Pudlák 89)

A proof system f for L is (p)-optimal if f (p-)simulates every proof system for L.

Simulations Between Proof Systems

Theorem (Cook, Reckhow 79)

All Frege systems are polynomially equivalent.

Theorem (Krajíček, Pudlák 89)

Every proof system is simulated by a proof system of the form $EF + \Phi$.

Problem (Krajíček, Pudlák 89)

Do optimal proof systems exist?

The Propositional Sequent Calculus

- Historically one of the first and best analyzed proof systems [Gentzen 35]
- basic objects: sequents $\varphi_1, \ldots, \varphi_m \vdash \psi_1, \ldots, \psi_k$.
- Sequents of the form

$$A \vdash A, \quad 0 \vdash, \quad \vdash 1$$

  are called initial sequents.
- An *LK*-proof of a propositional formula φ is a derivation of the sequent

$$\vdash \varphi$$

  from initial sequents by the following rules.

Rules of *LK*

$$\frac{\Gamma \vdash \Delta}{A, \Gamma \vdash \Delta} \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A} \quad \text{(weakening)}$$

$$\frac{\Gamma_1, A, B, \Gamma_2 \vdash \Delta}{\Gamma_1, B, A, \Gamma_2 \vdash \Delta} \qquad \frac{\Gamma \vdash \Delta_1, A, B, \Delta_2}{\Gamma \vdash \Delta_1, B, A, \Delta_2} \quad \text{(exchange)}$$

$$\frac{\Gamma_1, A, A, \Gamma_2 \vdash \Delta}{\Gamma_1, A, \Gamma_2 \vdash \Delta} \qquad \frac{\Gamma \vdash \Delta_1, A, A, \Delta_2}{\Gamma \vdash \Delta_1, A, \Delta_2} \quad \text{(contradiction)}$$

$$\frac{\Gamma \vdash \Delta, A}{\neg A, \Gamma \vdash \Delta} \qquad \frac{A, \Gamma \vdash \Delta}{\Gamma \vdash \Delta, \neg A} \quad (\neg \text{ introduction})$$

$$\frac{A, \Gamma \vdash \Delta}{A \wedge B, \Gamma \vdash \Delta} \qquad \frac{A, \Gamma \vdash \Delta}{B \wedge A, \Gamma \vdash \Delta} \qquad \frac{\Gamma \vdash \Delta, A \qquad \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \wedge B} \quad (\wedge \text{ rules})$$

$$\frac{A, \Gamma \vdash \Delta \qquad B, \Gamma \vdash \Delta}{A \vee B, \Gamma \vdash \Delta} \qquad \frac{\Gamma \vdash \Delta, A}{\Gamma \vdash \Delta, A \vee B} \qquad \frac{\Gamma \vdash \Delta, A}{\Gamma \vdash \Delta, B \vee A} \quad (\vee \text{ rules})$$

$$\frac{\Gamma \vdash \Delta, A \qquad A, \Gamma \vdash \Delta}{\Gamma \vdash \Delta} \quad \text{(cut rule)}$$

A robust proof system: Frege/LK

Proposition (Cook, Reckhow 79)

Frege systems and the propositional sequent calculus LK are polynomially equivalent.

Polynomially Bounded Proof Systems

Polynomial Bounds on Proofs

A proof system f for L is polynomially bounded if there exists a polynomial p such that every $x \in L$ has an f-proof of size $\leq p(|x|)$.

Polynomially Bounded Proof Systems

Polynomial Bounds on Proofs

A proof system f for L is polynomially bounded if there exists a polynomial p such that every $x \in L$ has an f-proof of size $\leq p(|x|)$.

Examples

- The standard proof system for SAT is polynomially bounded:

$$sat(\alpha, \varphi) = \begin{cases} \varphi & \text{if } \alpha \text{ is a satisfying assignment for } \varphi \\ p & \text{otherwise.} \end{cases}$$

- The truth-table system is not a polynomially bounded proof system for TAUT.

The Cook-Reckhow Theorem

Question

Is there a polynomially bounded proof system for TAUT?

Theorem (Cook, Reckhow 79)

A language L has a polynomially bounded proof system if and only if $L \in$ NP*.*

For propositional proof systems

TAUT has a polynomially bounded proof system
if and only if NP $=$ coNP.

Cook's Programme

Separate NP from coNP (and hence P and NP) by showing super-polynomial lower bounds to the size of proofs in all propositional proof systems.

Cook's Programme

Separate NP from coNP (and hence P and NP) by showing super-polynomial lower bounds to the size of proofs in all propositional proof systems.

Showing lower bounds for a system P means

finding an infinite family θ_n of propositional tautologies s.t.

- $|\theta_n| = n^{O(1)}$;
- θ_n requires super-polynomial size proofs in P.

Cook's Programme

Separate NP from coNP (and hence P and NP) by showing super-polynomial lower bounds to the size of proofs in all propositional proof systems.

Showing lower bounds for a system P means

finding an infinite family θ_n of propositional tautologies s.t.

- $|\theta_n| = n^{O(1)}$;
- θ_n requires super-polynomial size proofs in P.
  - Better: ... exponential size proofs.

Cook's Programme

Separate NP from coNP (and hence P and NP) by showing super-polynomial lower bounds to the size of proofs in all propositional proof systems.

Showing lower bounds for a system P means

finding an infinite family θ_n of propositional tautologies s.t.

- $|\theta_n| = n^{O(1)}$;
- θ_n requires super-polynomial size proofs in P.
  - Better: ... exponential size proofs.

Even better

- Find a sequence of polynomially constructible formulas which require long proofs.
- This is usually the case: take θ_n as the propositional formalization of some combinatorial principle.
- Find a large set of formulas (e.g. random 3-CNF) which require long proofs.

Cook's Programme

Separate NP from coNP (and hence P and NP) by showing super-polynomial lower bounds to the size of proofs in all propositional proof systems.

Cook's Programme

Separate NP from coNP (and hence P and NP) by showing super-polynomial lower bounds to the size of proofs in all propositional proof systems.

Progress in this programme

- Haken (1985): exponential lower bound to the proof size in Resolution for the pigeonhole principle
- Ajtai (1988): Super-polynomial lower bound for bounded-depth Frege systems (Improved by Beame, Impagliazzo, Krajíček, Pitassi, Pudlák, Woods)
- Lower bounds for algebraic and geometric proof systems:
  - Cutting Planes [Pudlák 97]
  - Polynomial Calculus [Razborov 98, . . .]
  - Nullstellensatz [Buss et al. 97] [Grigoriev 98]
  - OBDD proof systems [Krajíček 08] [Segerlind 08]

Techniques and Barriers

Techniques for lower bounds

- feasible interpolation [Krajíček 97]
- size-width relation [Ben-Sasson & Wigderson 01]
- game-theoretic techniques [Pudlák, Buss, Impagliazzo,. . .]
- proof complexity generators [Krajíček, Alekhnovich et al.]

The current barrier

Show lower bounds for Frege systems

Hard Formulas for Frege Systems?

Theorem (Buss 87)

The pigeonhole principle has polynomial-size proofs in Frege systems.

The search for hard formulas

- A number of combinatorial principles have been suggested, but most have poly-size Frege proofs.
- E.g., matrix multiplication: $AB = I \implies BA = I$ [Hrubeš & Tzameret 12]
- A good candidate from logic: reflection principles
- Problem: hard to analyze
- A promising approach: formulas from pseudo-random generators (Krajíček, Razborov)

Cutting Planes

- Cutting Planes uses the idea of linear programming.
- As in Resolution, CP is a refutation system that works with clauses.
- Clauses are translated into linear inequalities.

The Translation

- Clauses are translated into linear inequalities

$$a_1 p_1 + \ldots + a_n p_n \geq b \tag{1}$$

  with integer coefficients $a_1, \ldots, a_n$ and b.
- Propositional variables p are identically represented by integer variables p.
- $\neg p$ is translated to $1 - p$.
- A clause

$$C = \{l_1, \ldots, l_n\}$$

  with literals $l_i = p_i$ or $l_i = \neg p_i$ is translated into

$$f_1 + \ldots + f_n \geq 1$$

  with

$$f_i = \begin{cases} p_i & \text{if } l_i = p_i \\ 1 - p_i & \text{if } l_i = \neg p_i \end{cases}$$

  for $i = 1, \ldots, n$.
- To get an inequality of the form (1), constants are moved to the right hand side.

Axioms of CP

1. Let $\Gamma = \{C_1, \ldots, C_k\}$ be a set of clauses in variables $p_1, \ldots, p_n$.
2. As axioms in CP we use the translations of clauses $C_1, \ldots, C_k$ together with

$$p_i \geq 0, \quad -p_i \geq -1 \qquad i = 1, \ldots, n \ .$$

Rules of CP

1. **Addition:**

$$\frac{a_1 p_1 + \ldots + a_n p_n \geq b \qquad a'_1 p_1 + \ldots + a'_n p_n \geq b'}{(a_1 + a'_1) p_1 + \ldots + (a_n + a'_n) p_n \geq b + b'}$$

2. **Multiplication:**

$$\frac{a_1 p_1 + \ldots + a_n p_n \geq b}{c a_1 p_1 + \ldots + c a_n p_n \geq c b}$$

   with an arbitrary integer $c > 0$.

3. **Division:**

$$\frac{c a_1 p_1 + \ldots + c a_n p_n \geq b}{a_1 p_1 + \ldots + a_n p_n \geq \left\lceil \frac{b}{c} \right\rceil}$$

   with an arbitrary integer $c > 0$.

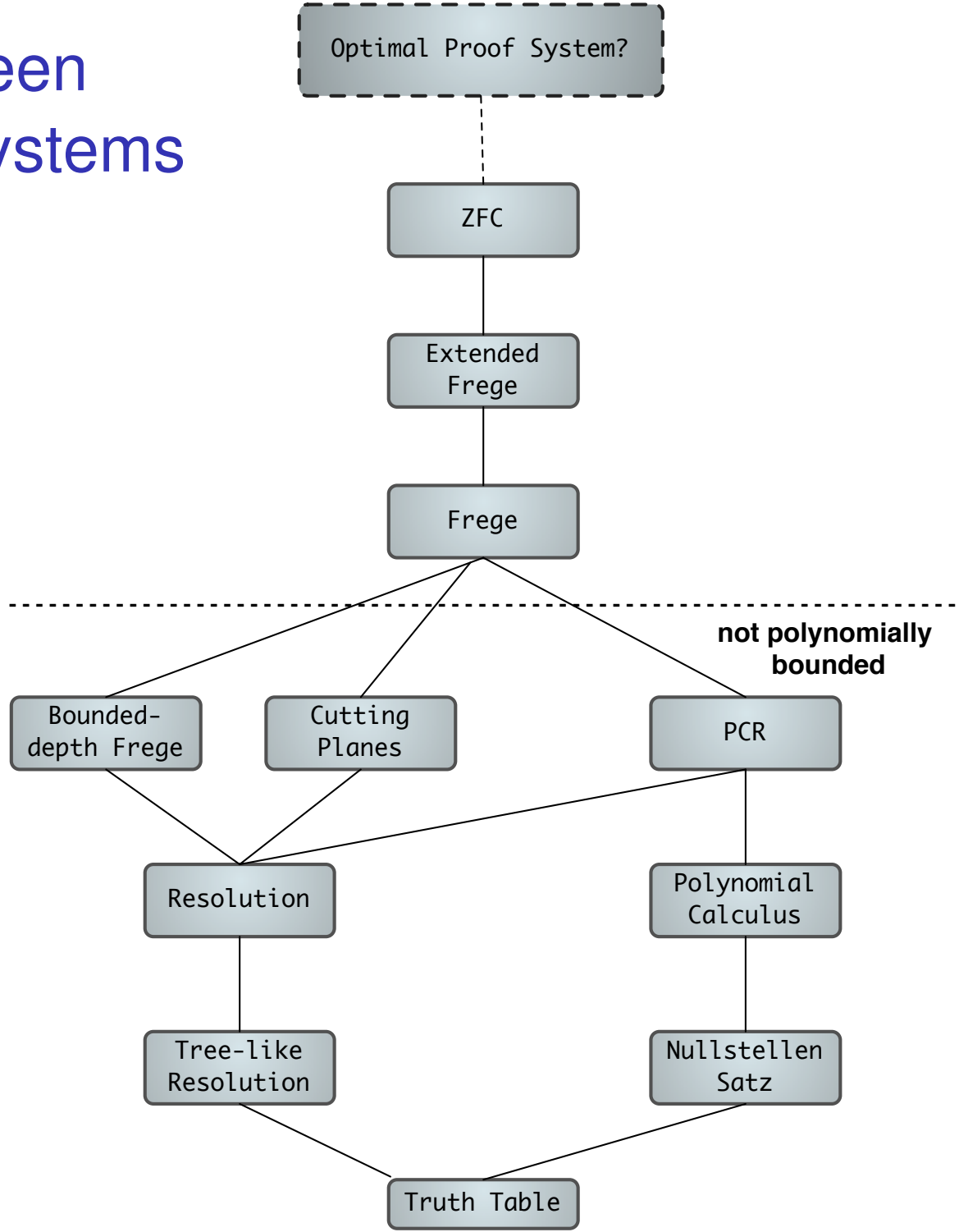
CP Refutations

- A CP refutation of a set of clauses Γ is a CP derivation of

$$0 \geq 1$$

  from the axioms corresponding to Γ.
- Easy to see: CP p-simulates Resolution.
- The converse is false.
- Frege systems p-simulate CP [Goerdt 91].

Simulations between important proof systems



Summary and Outline

Proof Complexity

- is at the intersection of logic and complexity.
- uses concepts and intuition from algebra, geometry, . . .

Main Objective

study lengths of proofs

Connections to other areas

- Separation of complexity classes
- Analysis of SAT algorithms
- Proof search – Automatizability
- First-Order Logic – Bounded Arithmetic
- Proving lower bounds is hard!

Proof Complexity

and

Analysis of SAT algorithms

Complexity of SAT

Propositional satisfiability

- Input: a propositional formula F
- Question: Is F satisfiable?

SAT is hard

- Cook 71: SAT is NP complete.
- No efficient algorithm unless P=NP.
- Exponential-time hypothesis: SAT has worst-case running time $2^{\Omega(n)}$.

SAT is efficient

SAT algorithms

- Intensively investigated and improved since the 90's (SAT competition, SAT conference)
- Routinely solve industrial instances with $\geq$ 100.000's of variables.
- Many problems can be efficiently encoded in SAT.
- Applied in many areas: model checking, data bases, bioinformatics ...

The success of SAT solvers

- Why are SAT algorithms so successful?
- What are their limitations?

DPLL Procedures

Davis, Putnam, Logemann & Loveland 60, 62 (DPLL)

- choose a variable x
- set x to 0/1
- simplify the formula
- recursively iterate until satisfying assignment is found or search fails

SAT solvers

- based on DPLL
- improved by unit propagation, clause learning, restarts ...
- implementation tuning

DPLL and Resolution

Well known

On unsatisfiable formulas DPLL produces tree-like Resolution refutations.

Tree-like Resolution

- operates with clauses
- only one rule

$$\frac{C \vee x \qquad D \vee \neg x}{C \cup D}$$

- proofs are trees

$$\dfrac{\dfrac{\{x_1, x_2\} \quad \{\neg x_1, x_2\}}{\{x_2\}} \qquad \dfrac{\{\neg x_1, \neg x_2\} \quad \{x_1, \neg x_2\}}{\{\neg x_2\}}}{\square}$$

An Equivalent Model

Boolean Decision Trees

- Binary tree
- Inner nodes are labeled with variables from F.
- Leafs are labeled with clauses from F.
- Each path in the tree corresponds to a partial assignment where a variable x gets value 0 or 1 according to whether the path branches left or right at the node labeled with x.
- In the tree, each path α must lead to a clause which is falsified by the assignment corresponding to α.

Boolean Decision Trees and the Search Problem

- A boolean decision tree solves the search problem for F:
  - given an assignment α,
  - find a clause from F falsified by α.
- Each tree-like Resolution refutation of F yields a boolean decision tree for F and vice versa, where the size of the Resolution proof equals the number of nodes in the decision tree.

Tree-like Resolution and SAT Solvers

- On unsatisfiable formulas, the DPLL algorithm produces a Boolean decision tree (e.g. a tree-like Resolution refutation) of the formula.
- To analyse the complexity of classical DPLL we need to understand the complexity of tree-like Resolution.

A game for tree-like Resolution lower bounds

- Let F be unsatisfiable.
- Delayer claims she knows a satisfying assignment.
- Prover wants to find a contradiction.

In each round

- Prover asks a variable x.
- Delayer "answers" 0/1 and gets some points for his choice.
- Prover wins if the partial assignment falsifies a clause from F.

Question

How many points can Delayer earn?

Details

If Prover asks variable x

- Delayer assigns two weights p_0 and p_1 which satisfy:

$$p_0 \geq 0 \quad p_1 \geq 0 \qquad p_0 + p_1 = 1.$$

- Prover chooses value b.
- Delayer gets $\log \frac{1}{p_b}$ points.
- If Prover chooses b with $p_b = 0$, Delayer gets ∞ points.

Intuition

In each round Delayer's points relate to the amount of information provided by Prover for his 0/1 choice on x.

A characterisation of tree-like Resolution size

Theorem (B., Galesi & Lauria 12)

For any unsatisfiable CNF F, the maximum score achievable in an Asymmetric Prover-Delayer game by a Delayer is exactly $\log \left\lceil \frac{S_T(F)}{2} \right\rceil$ *where $S_T(F)$ is the size of the shortest tree-like Resolution refutation of F.*

Corollary

If we can exhibit a Delayer strategy which gives p points to the Delayer for F (against any Prover), then each tree-like Resolution refutation of F must have size $2^{\Omega(p)}$.

The Optimal Bound for PHP

The pigeonhole principle

- PHP_n^m uses variables $x_{i,j}$ with $i \in [m]$ and $j \in [n]$,
- $x_{i,j}$ indicates that pigeon i goes into hole j.
- PHP_n^m consists of the clauses

$$\bigvee_{j \in [n]} x_{i,j} \quad \text{for all pigeons } i \in [m]$$

and

$$\neg x_{i_1,j} \vee \neg x_{i_2,j}$$

for all choices of distinct pigeons $i_1, i_2 \in [m]$ and $j \in [n]$.

Theorem (Iwama & Miyazaki 99, Dantchev & Riis 01)
PHP_n^m has tree-like Resolution refutation of size $2^{\theta(n \log n)}$.

The Proof by Asymmetric PD-games

- Let α be a partial assignment to the variables $\{ x_{i,j} \mid i \in [m], j \in [n] \}$.
- Let

  $$E_i(\alpha) = |\{ j \in [n] \mid \alpha(x_{i,j}) = 0 \text{ and } \alpha(x_{i',j}) \neq 1 \text{ for all } i' \in [m] \}| \ .$$

- Intuitively, $E_i(\alpha)$ corresponds to the number of holes which are still free but are explicitly excluded for pigeon i by α.

Delayer's Strategy

If Prover asks $x_{i,j}$ in game configuration α, Delayer chooses

$$\begin{array}{ll} p_0 = 1,\ p_1 = 0 & \text{if there exists } i' \in [m] \setminus \{i\} \text{ s.t. } \alpha(x_{i',j}) = 1 \text{ or} \\ & \text{if there exists } j' \in [n] \setminus \{j\} \text{ s.t. } \alpha(x_{i,j'}) = 1; \\ p_0 = 0,\ p_1 = 1 & \text{if } E_i(\alpha) \geq \frac{n}{2} \end{array}$$

and otherwise

$$p_0 = \frac{\frac{n}{2} - E_i(\alpha)}{\frac{n}{2} + 1 - E_i(\alpha)} \quad \text{and} \quad p_1 = \frac{1}{\frac{n}{2} + 1 - E_i(\alpha)} \ .$$

Intuition

Delayer leaves the choice to Prover as long as pigeon i does not already sit in a hole, hole j is still free, and there are at most $\frac{n}{2}$ excluded free holes for pigeon i.

Intuition on the Strategy

$$p_0 = \frac{\frac{n}{2} - E_i(\alpha)}{\frac{n}{2} + 1 - E_i(\alpha)} \quad \text{and} \quad p_1 = \frac{1}{\frac{n}{2} + 1 - E_i(\alpha)} \ .$$

- First observation: Delayer always earns more when Prover is setting a variable $x_{i,j}$ to 1 instead of setting it to 0.
- This is intuitively correct: the amount of freedom for Delayer to continue the game is by far more diminished by sending pigeon i to some hole than by just excluding a hole for pigeon i.
- In fact, our choice of scores can be completely explained by the following information-theoretic interpretation.

Information-theoretic Interpretation

- When Prover sends a pigeon to a hole, Delayer should always get about $\log n$ points on that pigeon.
- When we play the game, in each round Delayer should get some number of points proportional to the progress Prover made towards fixing pigeon i to a hole.

Information-theoretic Interpretation

- When Prover sends a pigeon to a hole, Delayer should always get about $\log n$ points on that pigeon.
- When we play the game, in each round Delayer should get some number of points proportional to the progress Prover made towards fixing pigeon i to a hole.

Example 1

- Prover fixes i to a hole in the very beginning by answering 1 to $x_{i,j}$.
- Then Delayer should get the $\log n$ points immediately.

$$\log \frac{1}{p_1} = \log \left(\frac{n}{2} + 1 - E_i(\alpha) \right) = \log \left(\frac{n}{2} + 1 \right)$$

Information-theoretic Interpretation

When we play the game, in each round Delayer should get some number of points proportional to the progress Prover made towards fixing pigeon i to a hole.

Example 2

- Prover has already excluded $\frac{n}{2} - 1$ holes for pigeon i.
- Then it does not matter whether Prover sets $x_{i,j}$ to 0 or 1 because after both answers pigeon i will be forced to a hole.
- Consequently, Delayer gets just 1 point regardless of whether Prover answers 0 or 1.

$$p_0 = \frac{\frac{n}{2} - E_i(\alpha)}{\frac{n}{2} + 1 - E_i(\alpha)} \quad \text{and} \quad p_1 = \frac{1}{\frac{n}{2} + 1 - E_i(\alpha)} \ .$$

This is exactly what our score function provides.

Analysis

- If Delayer uses this strategy, then the small clauses $\neg x_{i_1,j} \vee \neg x_{i_2,j}$ from PHP_n^m will not be violated in the game.
- Therefore, a contradiction will always be reached on one of the big clauses $\bigvee_{j\in[n]} x_{i,j}$.
- Let us assume now that the game ends by violating $\bigvee_{j\in[n]} x_{i,j}$, i. e., for pigeon i all variables $x_{i,j}$ with $j \in [n]$ have been set to 0.
- As soon as the number $E_i(\alpha)$ of excluded free holes for pigeon i reaches the threshold $\frac{n}{2}$, Delayer will not leave the choice to Prover.
- Instead, Delayer will try to place pigeon i into some hole.
- If Delayer still answers 0 to $x_{i,j}$ even after $E_i(\alpha) > \frac{n}{2}$, it must be the case that some other pigeon already sits in hole j, i. e., for some $i' \neq i$, $\alpha(x_{i',j}) = 1$.
- Therefore, at the end of the game at least $\frac{n}{2}$ variables have been set to 1.

Analysis

We know

- At the end of the game at least $\frac{n}{2}$ variables have been set to 1.
- We assume that these are the variables x_{i,j_i} for $i = 1, \ldots, \frac{n}{2}$.

How many points does Delayer earn?

- We calculate the points separately for each pigeon $i = 1, \ldots, \frac{n}{2}$.
- Distinguish two cases: whether x_{i,j_i} was set to 1 by Delayer or Prover.

Analysis

Result

In total, Delayer gets at least

$$\frac{n}{2}\log\left(\frac{n}{2}+1\right)$$

points in the game.

Corollary

We obtain $2^{\frac{n}{2}\log\left(\frac{n}{2}+1\right)}$ as a lower bound to the size of each tree-like Resolution refutation of PHP_n^m.

Tree-like vs. DAG-like Proof Systems

A general question

Are dag-like proof systems more powerful than tree-like systems?
Is the dag-like proof system simulated by the corresponding tree-like proof system?

The answer depends on the proof system.

- For Resolution: Dag-like systems are more powerful (exponential separation).
- For Frege systems: dag-like and tree-like versions are equivalent. [Krajíček 95]

Tree-like vs. DAG-like Proof Systems

Theorem (Krajíček 95)

Tree-like Frege systems p-simulate (dag-like) Frege.

Proof.

- Let $A_1, \ldots, A_m$ be a proof in (dag-like) Frege.
- Let

$$B_i = A_1 \wedge \ldots \wedge A_i$$

  for $i = 1, \ldots, m$.
- We get linear-size tree-like Frege proofs of

$$B_i \rightarrow B_{i+1}$$

  for $i = 1, \ldots, m-1$.
- $m-1$ applications of Modus Ponens give A_m.
- The proof is tree-like.

□

Tree-like vs. DAG-like Resolution

The result

There is a family of unsatisfiable CNF that have polynomial-size dag-like Resolution refutations, but require exponential-size tree-like Resolution refutations.

History

- Goerdt 92: first separation: example with poly-size dag-like refutations, but only quasi-polynomial tree-like refutations (modification of PHP).
- Bonet, Galesi, Esteban, Johannsen 98: first exponential separation
- Ben-Sasson, Impagliazzo, Wigderson 03: simplified and improved separation by using games

Pebbling Games

- pebbling games are played on DAGs
- source nodes: in-degree 0
- target nodes: out-degree 0
- game: place pebbles on nodes according to rules
- aim: place a pebble at some target node

Rules

1. Source nodes can be pebbled freely.
2. All other nodes can be pebbled if all their parents are pebbled.
3. Pebbles can be removed at any time.

Pebbling Number

Complexity measure

Maximal number of pebbles placed simultaneously on the graph.

Pebbling number of a strategy to pebble a graph

- Let S be a strategy to pebble the dag G.
- $P(G, S) =$ max # of pebbles placed simultaneously on G while following strategy S

Pebbling number of G

$P(G) = \min\{\, P(G, S) \mid S \text{ is a strategy to pebble } G \,\}$

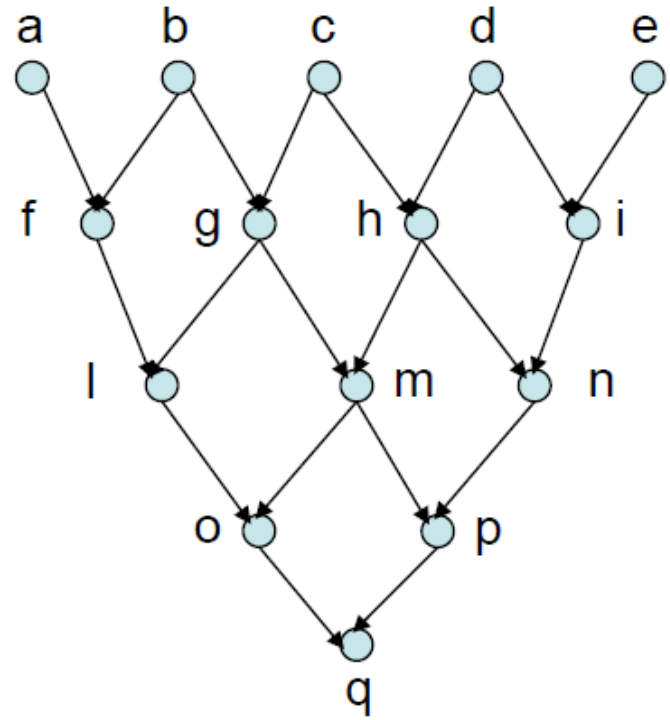
Graphs with High Pebbling Numbers

Theorem (Celoni, Paul, Tarjan 77)

There exist graphs G with n vertices such that

$$P(G) = \Omega\left(\frac{n}{\log n}\right).$$

- The proof is constructive.
- Example: pyramidal graphs



Pebbling Formulas

DAG $G = (V, E)$

Propositional variables

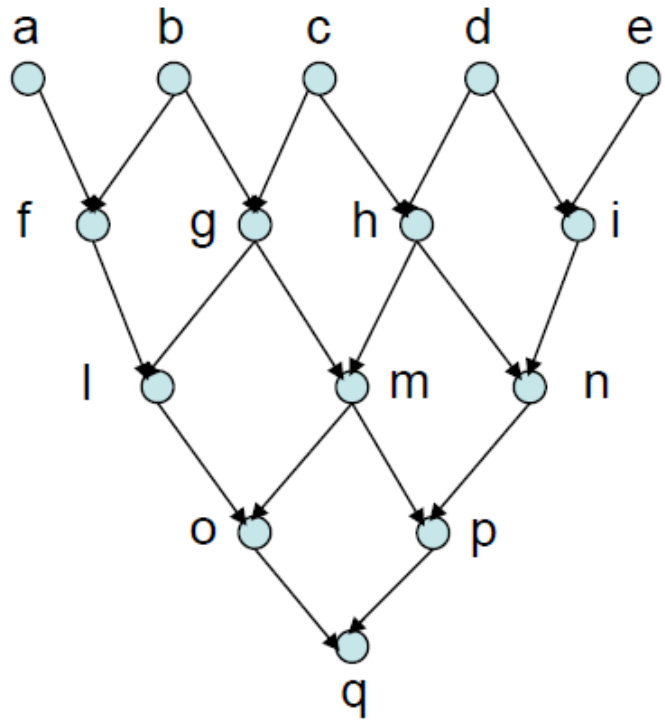
- x_v for all $v \in V$
- Meaning: $x_v = 1$ if v has been pebbled

Clauses in $Peb^0(G)$

x_v	for any source node v
$(\bigwedge_{u \in N^-(v)} x_u) \rightarrow x_v$	for all nodes v where $N^-(v)$ are the parents of v
$\neg x_v$	for any target node v

Complexity of Peb^0

- $Peb^0(G)$ is unsatisfiable.
- But: They have polynomial-size tree-like Resolution refutations.
- Idea: Start from the bottom and explore the graph in a breadth-first fashion.



Adding Complexity to $Peb^0(G)$

Idea

- Use pebbles of two different colors: black and white
- Consider a node pebbled if it has a black or white pebble on it

The new principle

- Source nodes can always be pebbled black or white.
- For an internal node v, if all its parents are pebbled black or white, then v can be pebbled either black or white.
- No target node is pebbled black or white.

The New Pebbling Formulas

DAG $G = (V, E)$ with in-degree ≤ 2

Propositional variables

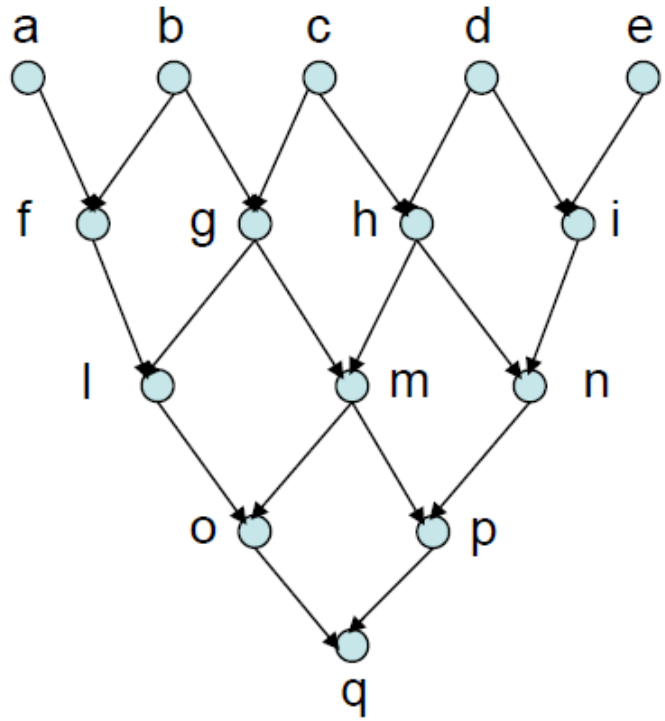
- $x_{v,c}$ for all $v \in V$ and $c \in \{B, W\}$
- Meaning: $x_{v,B} = 1$ if v has been pebbled black
  $x_{v,W} = 1$ if v has been pebbled white

Clauses in $Peb(G)$

$x_{v,B} \vee x_{v,W}$	for any source node v
$x_{u,a} \wedge x_{w,b} \rightarrow x_{v,B} \vee x_{v,W}$	for all nodes $v \in V$, $a, b \in \{B, W\}$ where u and w are the parents of v
$\neg x_{v,B}, \neg x_{v,W}$	for any target node v

Complexity of $Peb(G)$

- $Peb(G)$ is unsatisfiable.
- Proof strategy as for Peb^0 does not work anymore.
- But: They have polynomial-size dag-like Resolution refutations.
- Aim: Show a lower bound for tree-like Resolution



The Separation

Theorem

There exists an infinite family of explicitly constructible formulas θ_n *s.t.*

1. $|\theta_n| = O(n)$;
2. θ_n *require tree-like Resolution refutations of size* $2^{\Omega\left(\frac{n}{\log n}\right)}$;
3. θ_n *have Resolution refutations of size* $O(n)$.

Beyond DPLL / Tree-like Resolution

- Each run of a SAT algorithm on an unsatisfiable formula yields a proof of unsatisfiability.
- Therefore: each SAT algorithm defines a proof system for all unsatisfiable formulas.

Central task

For practical SAT solvers: understand and analyse the corresponding proof system.

As a consequence

this leads to lower bounds on the running time of SAT algorithms on unsatisfiable formulas.

Clause Learning

DPPL+Clause Learning / CDCL solvers

- enhances DPLL by learning new clauses from conflicts
- major algorithmic improvement upon DPLL

Corresponding proof systems

- Pool Resolution [Van Gelder 04]
- Resolution trees with lemmas [Buss, Hoffmann & Johannsen 08]
- ...

What is the proof-theoretic strength of these algorithms?

- All these proof systems are simulated by Resolution.
- But: general Resolution is simulated by some of these systems (DPLL + clause learning + restarts) [Beame, Kautz & Sabharwal 04], [Pipatsrisawat & Darwiche 11]

Other complexity measures

Proof size

- corresponds to running time of algorithms
- most studied measure

Proof space

- maximal size of blackboard to carry out proof
- corresponds to memory requirements for SAT algorithms

Strong results

- Lower bounds on space
  [Torán 99, Alekhnovich et al. 00]
- Size-space trade-offs
  [Ben-Sasson & Nordström 11,
  Beame, Beck & Imagliazzo 12]

Proof Search – Automatizability

Automatizability of proof systems

From the practical perspective

- So far we concentrated on estimating the length of a proof for a formula.
- But how complicated is it to actually find a proof?

Automatizability of proof systems

From the practical perspective

- So far we concentrated on estimating the length of a proof for a formula.
- But how complicated is it to actually find a proof?

Definition

P is automatizable if there exists a deterministic algorithm with

input:	a formula φ
output:	a P-proof of φ (if it exists)
time:	polynomial in the length of the shortest P-proof of φ

Which Proof Systems are Automatizable?

A trivial positive example

The truth-table system is automatizable.

What about interesting systems?

Theorem (Krajíček & Pudlák 98)

Extended Frege systems are not automatizable unless RSA is insecure.

Theorem (Bonet, Pitassi, Raz 00)

Frege systems are not automatizable unless Blum integers can be factored in polynomial time
(a Blum integer is the product of two primes which are both congruent 3 modulo 4).

Theorem (Bonet, Domingo, Gavaldà, Maciel, Pitassi 04)

Bounded-depth Frege systems are not automatizable under cryptographic assumptions.

Automatizability of Resolution

Theorem (Beame, Karp, Pitassi, Saks 02)

Tree-like Resolution is automatizable in quasi-polynomial time. (Quasi-polynomial time $= n^{O(\log n)}$*)*

Theorem (Alekhnovich & Razborov 01, Eickmeyer, Grohe & Grübner 08)

Resolution is not automatizable unless $FPT = W[P]$*.*

Open problem

Is Resolution weakly automatizable, i.e., is there an automatizable proof system which simulates Resolution?

Theorem (Beckmann, Pudlák & Thapen 13)

If Resolution is weakly automatizable then parity games can be decided in polynomial time.

Proof Complexity

and

First-order Logic

Bounded Arithmetic

- first-order arithmetic theories
- weak subsystems of Peano arithmetic
- axiomatized by
  - a number of basic axioms describing the interplay of $+, \cdot, \leq, 0, 1, \ldots$ and
  - some controlled amount of induction

Most important examples

- $I\Delta_0$ (induction for all bounded formulas)
- PV (formalizes poly-time computations) [Cook 75]
- $S_2^1 \subseteq T_2^1 \subseteq S_2^2 \subseteq T_2^2 \subseteq \ldots \subseteq S_2 = T_2$ [Buss 86]

Propositional Translations

Bounded formulas

- A bounded universal quantifier is of the form $(\forall x)(|x| \leq t \rightarrow \ldots)$ with some term t.
- Π_1^b-formulas only contain bounded universal quantifiers.
- Π_1^b-formulas describe coNP-sets.

From first-order to propositional formulas

A Π_1^b-formula $\varphi(x)$ can be translated into a sequence of propositional formulas $\|\varphi\|^n$ such that

- $\|\varphi\|^n$ has polynomial size in n;
- for each $a \in \mathbb{N}$, $\mathbb{N} \models \varphi(a)$ iff $\|\varphi\|^{|a|}(a) \in \mathit{TAUT}$.

Bounded Arithmetic and Propositional Proof Systems

The correspondence

An arithmetic theory T corresponds to a propositional proof system P if the following conditions are satisfied:

- For $\varphi \in \Pi^b_1$, if $T \vdash (\forall x)\varphi$, then there are poly-size P-proofs of $\|\varphi\|^n$.
- T proves the correctness of P, i.e. $T \vdash RFN(P)$.

Example

S^1_2 corresponds to extended Frege EF.

This correspondence can be applied to

- construct short P-proofs (upper bounds);
- show lower bounds to the proof size for P [Ajtai 94];
- show simulations between proof systems.

Uniform vs. Non-uniform Concepts

	Logic	Complexity
uniform	arithmetic theories Π_1^b formulas	P, NP, coNP, ... Turing machines
non-uniform	proof systems propositional formulas	AC^0, P/poly, NP/poly, ... Boolean circuits

Uniform vs. Non-uniform Concepts

	Logic	Complexity
uniform	arithmetic theories Π_1^b formulas	P, NP, coNP, ... Turing machines
non-uniform	proof systems propositional formulas	AC^0, P/poly, NP/poly, ... Boolean circuits

Our experience

Lower bounds in the non-uniform models are very hard.

Proof Complexity – Further Connections

Excursion 1: Parameterised Proof Complexity

Introduced by Dantchev, Martin & Szeider 11

Refined view on lengths of proofs

- Refined gap theorem for Resolution
  [Dantchev, Martin & Szeider 11]
- Many classically hard formulas become easy.
- Bounded-width CNFs have short proofs (fpt size).
  [B., Galesi, Lauria, Razborov 12]

Lower bounds for parameterised proofs are harder

- New lower bound techniques needed.
  [B., Galesi & Lauria 12]
- strong lower bound for pigeonhole principle
  [B., Galesi, Lauria, Razborov 12]

Excursion 2: Proof Complexity of Non-classical Logics

In the last decade

Intense research on complexity of proofs in non-classical logics

Why non-classical logics?

- Non-classical logics such as modal logics, tree logics, or non-monotonic logics have numerous applications, e. g. verification, model checking, expert systems, or modeling common sense reasoning.
- Yields better understanding of propositional proofs – we see new phenomena which do not appear in classical logic.

Strong results in non-classical logics

Separation of complexity classes

- Non-classical logics are often more expressive than propositional logic.
- Satisfiability of the modal logic K is PSPACE-complete [Ladner 77].
- Intuitively, lower bounds to the lengths of proofs in non-classical logic should be easier to obtain

Results

- In contrast to classical logic, we have exponential lower bounds for modal and intuitionistic Frege systems [Hrubeš 07, Jeřábek 09]
- new phenomena for extensions of Frege (EF vs. SF) [Jeřábek 09]
- other logics: default logic, autoepistemic logic [B., Meier, Müller, Thomas, Vollmer 11] [B. 13]

Conclusion

Proof Complexity

- Main objective: understand the complexity of theorem proving
- many strong results in the last decades
- many ingenious techniques developed

Proof Complexity – Interactions

- Computational Complexity: separation of complexity classes
- SAT solving: algorithmic limitations
- First-order logic: bounded arithmetic
- Proving lower bounds is hard

Proof Complexity – SAT – Interactions

From the Proof Complexity side

- understand current algorithmic techniques
- find simple compelling proof systems
- analyse their strength

From the SAT side

- use strong proof systems developed and analysed
- Cutting planes, Polynomial Calculus, ...

Proof Complexity – SAT – Interactions

From the Proof Complexity side

- understand current algorithmic techniques
- find simple compelling proof systems
- analyse their strength

From the SAT side

- use strong proof systems developed and analysed
- Cutting planes, Polynomial Calculus, ...

More Interaction!