Lectures on Proof Theory

Samuel Buss
University of California San Diego

Some day

Technical Report Some number
School of Computer Science
McGill University

Foreword

January 11th 1996

The 1995 McGill-Montréal Invitational Workshop on Complexity Theory was held at McGill University's Bellairs Research Institute in Barbados from March 6 to March 11, organized by Pierre McKenzie (U. de Montréal) and Denis Thérien (McGill U.).

The core of this year's workshop was a series of lectures given by Sam Buss from the University of California at San Diego, on the theme of logic (mostly proof theory) and complexity. The participants thus had the priviledge of being given a solid coverage of many beautiful recent results.

This year again, it was decided to keep a permanent record of the workshop and to produce a technical report containing the material presented by Sam. The presentation has been kept rather informal and no attempt has been made to overpolish the material.

Warm thanks go to the note-takers: K. Regan (Buffalo), J. Toran (Barcelona), A. Maciel (McGill), E. Allender (Rutgers), P. McKenzie (Montréal), P. Clote (Boston College), M. Bonet (U. of Pennsylvania), R. Raz (Weizmann Institute), C. Zamora (McGill), C. Lautemann (Mainz) and T. Pitassi (Pittsburgh). Above all, we are very much indebted to Sam for having put so much energy in these lectures: they were very much enjoyed by everyone.

Denis Thérien

Contents

Proof Theory and Complexity

by Kenneth W. Regan and Jacobo Torán

Monday Morning, March 6

1 Preliminars

At the first meeting on Monday morning, March 6th, Sam was welcomed by organizers Denis Therién and Pierre McKenzie. Sam mentioned that it was of course the custom to thank the organizers; "here, it's a sheer delight." He said he'd be flexible on what he could do, having brought material on the following:

1. Introduction to Propositional Proof Lengths
2. Interpolation and "Natural Proofs." (A theorem by Razborov that if certain pseudorandom number generators (PSRGs) exist, then certain Bounded Arithmetics (BAs) cannot prove $P \neq NP$.)
3. Introduction to Bounded Arithmetic.
4. Gödel's Theorem on Lengths of Proofs
5. First-Order Proof Theory (cut-elimination, basic results)
6. Cutting-Planes Proofs
7. The Boolean Formula Value Problem is in ALOGTIME—newer proof.

In the end, we decided to hear virtually all of these topics, in the above order.

Scribe's Note (KWR): I have added some of my own exposition to what Sam had on his slides. Some of these are just "glosses" that fill in a few sentences of explanation, and I have left them unmarked. Others, however, introduce new content or raise points that were not mentioned during the lectures, and here I've marked them "added by KWR." I also took down several questions that people asked during the lectures, and I felt it would add "color" to include them in the notes. *Let the Notes Begin!*

2 Introduction to Lengths of Propositional Proofs

First, an outline:

- Frege systems
- Resolution
- Abstract proof systems
- Extended Frege ($e\mathcal{F}$) and substitution Frege ($s\mathcal{F}$) systems
- Tautologies based on the Pigeon-Hole Principle (PHP)
- Best-known lower bounds, including a survey of the state of the art for bounds on proof lengths in various *restricted fragments* of Frege systems

For all of this work, we will use *propositional formulas*, which are built up out of the following:

- Variables: $p_1, p_2, p_3, \ldots$; or informally, $p, q, r, \ldots$
- Logical connectives: $\neg$, $\wedge$, $\vee$, $\to$, $\leftrightarrow$, $\oplus$, ... We will talk about systems with subsets of these connectives, and more abstractly, of systems with finite sets of connectives of any arities.
- Parentheses! (,).
- Propositional formulas: Typified by: p_1, $(\neg p_1)$, $(p_1 \to (p_2 \vee p_3))$. This uses "fully-parenthesized" syntax. We will often omit parentheses and rely on the usual rules of precedence: $\neg$ binds tighter than $\{ \wedge, \vee \}$, which bind tighter than $\to$, which binds tighter than $\leftrightarrow$. Exclusive-or $\oplus$ is not assigned a precedence.

- The constants $\top$ (or TRUE) for "true" and $\bot$ (or FALSE) for "false." These are often identified with '1' for $\top$ and '0' for $\bot$, or vice-versa, or '–1' for $\top$ and '1' for $\bot$, or etc. Or we may dispense with these constants altogether and define $\top \equiv (p_1 \vee \neg p_1)$, $\bot \equiv (p_1 \wedge \neg p_1)$. The differences will not be important, and we shall be free to choose whatever looks best at a given time.

Definition 2.1. Let ϕ be a propositional formula, with variables $p_1, \ldots, p_n$. A *truth assignment* $\vec{a} \in \{0,1\}^n$ assigns a true/false value to each p_i, and induces a value $\phi(\vec{a})$. If $\phi(\vec{a}) = \top$ then we say that $\vec{a}$ *satisfies* ϕ, and we sometimes also write $\vec{a} \models \phi$.

A formula ϕ is a *tautology* (or: is *valid*) if it is true under all truth assignments; i.e., if $(\forall \vec{a} \in \{0,1\}^n)\, \vec{a} \models \phi$. Sometimes one writes simply $\models \phi$ to say that ϕ is valid. Two formulas ϕ and ψ are *equivalent* if $\phi \leftrightarrow \psi$ is a tautology, which is the same as saying that ϕ and ψ have the same set of satisfying truth assignments.

Let V stand for the language of tautologies, under a straightforward encoding scheme. *Cook's Theorem* [Coo71] shows that P = NP iff $V \in$ P; i.e., iff there is a deterministic polynomial-time algorithm for recognizing tautologies. V is complete for coNP under polynomial-time many-one reducibility $\leq_m^p$; the NP complete language SAT is essentially the same as the complement of V, going from ϕ to $\neg\phi$.

It's worth asking at this point: what methods do we use to test the validity of a formula? In school one learns (1) the method of "Truth Tables," which takes time on the order of 2^n. No "shortcut" method is known that does any better than time-2^n in the worst case. But (2) the normal way to tell that something is valid is to *prove* it! Whether proofs with polynomial-size lengths exist, and whether they can be efficiently found, are the proof-theory sides of the question of whether SAT has small circuits or belongs to P. This starts us on the road of examining specific *proof systems* for propositional formulas.

2.1 Frege systems

A *Frege proof system* $\mathcal{F}$ has a finite set of *schematic axioms* and *rules of inference*. The meaning of these terms is best conveyed by a concrete example:[1]

Rules of inference: Only one, called *modus ponens* (MP):

$$\cdot \; \frac{P \qquad P \to Q}{Q}\,\mathrm{MP}$$

Axioms:

1. $(P \wedge Q) \to P$
2. $(P \wedge Q) \to Q$
3. $P \to (P \vee Q)$
4. $Q \to (P \vee Q)$
5. $(P \to Q) \to ((P \to \neg Q) \to \neg P)$
6. $(\neg\neg P) \to P$
7. $P \to (Q \to P \wedge Q)$
8. $(P \to R) \to ((Q \to R) \to (P \vee Q \to R))$
9. $P \to (Q \to P)$
10. $(P \to Q) \to (P \to (Q \to R)) \to (P \to R)$.

Here it is important to note that P, Q, and R are not single formulas, but meta-symbols that can stand for *any* propositional formula. Commonly one would call the above "one rule and ten axioms," but formally each item stands for an infinite set of *instances* of the rule or axiom. Each instance is obtained by substituting some propositional formula for P, Q, and/or R. (This distinction between axioms and axiom schemas becomes much more important in predicate logic, where one has quantifiers, and especially in

[1]We consulted with Sam on which of many possible axiom sets to illustrate. Although they all linearly simulate each other, some are "nicer" for basic theorems when you teach this stuff in detail. Much of the other text in this subsection is added by KWR.

formal systems of arithmetic. For instance, Peano Arithmetic (PA) can be given six single axioms plus the *axiom schema* of induction, but there is no finite axiom set whose arithmetical consequences are exactly those of PA.)

Since every propositional variable is itself a formula, one can form the "smallest instances" of each of the above by inserting the variables p, q, and r in place of P, Q, and R. Then every other instance of the rule or axiom is obtainable by a *substitution* σ of formulas for those variables. Generally, one can define *substitutions* of formulas for the variables in any given propositional formula ϕ; this notion will be prominent later on.

It is now clear how to abstract the definition of a Frege system $\mathcal{F}$ to say that it consists of

(1) a domain T of well-formed formulas over some propositional *language* $\mathcal{L}$,

(2) a finite set $\mathcal{A}$ of schematic axioms, and

(3) a finite set $\mathcal{R}$ of schematic rules.

Alternatively, one can define (3) as a finite set of partial functions on T. Each rule $R \in \mathcal{R}$ has some arity $k \geq 1$. In the propositional case, we can identify the *language* $\mathcal{L}$ with the set of connectives allowed in the formulas in T. For instance, the above system has language $\{\neg, \wedge, \vee, \rightarrow\}$. (In predicate logic and arithmetic, the *language* may also include special distinguished predicate and function symbols, for instance $=$ (equality), $<$, S (for the successor function), $+$, and $\cdot$; then the style of proof system we are building is commonly called a *Hilbert system*.) In speaking of $\mathcal{F}$ as a Frege system, it is taken for granted that $\mathcal{L}$ is *propositionally complete*, meaning that every formula ϕ over the "standard basis" $\{\neg, \wedge, \vee\}$ has an equivalent formula ϕ' over $\mathcal{L}$.

A *Frege proof* Π in a system $\mathcal{F} = (\mathcal{L}, \mathcal{A}, \mathcal{R})$ is a sequence $(\psi_1, \psi_2, \ldots, \psi_m)$ such that for all i, either ψ_i is an (instance of an) axiom, or there exist $j_1, \ldots, j_k < i$ and a k-ary rule $R \in \mathcal{R}$ such that $\psi_i = R(\psi_{j_1}, \ldots, \psi_{j_k})$. Then Π is a proof *of* the *theorem* $\psi = \psi_m$, and we may variously write $\vdash \psi$, $\vdash_\Pi \psi$, $\vdash_\mathcal{F} \psi$, or $\mathcal{F} \vdash \psi$.

The system $\mathcal{F}$ is *sound* if every theorem is valid, and *complete* if every valid formula ψ has a proof. These properties are usually built into the term "Frege system."

Theorem 2.1 *There exist (many) sound and complete Frege proof systems, including the above.*

(At this point, a few questions were raised. Someone brought up the notion of *implicational completeness*: Write $\phi \models \psi$ if for every truth assignment $\vec{a}$ to variables occurring in ϕ and/or ψ, $\vec{a} \models \phi \Longrightarrow \vec{a} \models \psi$. Write $\phi \vdash \psi$ if adding ϕ as an axiom would allow one to construct a proof of ψ. (Note that neither ϕ or ψ need be valid by themselves.) Then $\mathcal{F}$ is *implicationally complete* if whenever $\phi \models \psi$, also $\phi \vdash_{\mathcal{F}} \psi$. When $\mathcal{F}$ has *modus ponens* among its rules of inference, this is easily seen to be equivalent to the simple notions of completeness, but it is possible to craft "pathological" Frege systems without MP that are complete but not implicationally complete. Similarly define $\mathcal{F}$ to be *implicationally sound* if whenever $\phi \vdash_{\mathcal{F}} \psi$, then $\phi \models \psi$. When we generalize to *substitution Frege systems*, we will lose implicational soundness, because the substitution rule e.g. allows ψ to be ϕ with its variables renamed, but such ϕ will generally not be a consequence of ϕ.

Many of the notions defined in this and the next subsection extend naturally to systems of predicate logic, first-order arithmetics, and even to higher-order logics. They were used freely in these other contexts by Sam in later lectures, and we were expected to carry them over. Adding four "quantifier axiom schemas" to 1.–10. above yields a "Frege-style" system (rather, a Hilbert system) for first-order predicate logic that is sound and complete. However, when we move to arithmetics with $+$ and $\cdot$, and with a computable set of axioms and inference rules, at least one of soundness or completeness goes out the window—of course this is *Gödel's First Incompleteness Theorem.*)

2.2 Complexity of Proofs

Now we can highlight the three principal complexity notions for proofs:

Definition 2.2. (a) The *number of lines* or *steps* in a proof $\Pi = (\psi_1, \ldots, \psi_m)$ equals m.

(b) The *symbol-length* of the proof is $n = |\Pi| = \sum_{i=1}^{m} |\psi_i|$.

(c) The *depth* d of the proof is the maximum AND/OR depth of a formula ψ_i occurring in the proof.

By the *length* or *size* of a proof we usually mean the symbol-length. We write $\mathcal{F} \vdash^n A$ to mean that A has a proof in $\mathcal{F}$ of at most n symbols.

The AND/OR depth is one notion of the complexity of an individual formula ψ: Write ψ over the basis $\{\wedge, \vee, \neg\}$, and use DeMorgan's Laws to bring the $\neg$s on the variables only. Then count the maximum number of alternations between $\wedge$ and $\vee$ in a path from the top operand to a variable in the formula. (Alternatively, by careful padding we can rewrite ψ in a "leveled" form such that all paths have the same sequence of $\wedge$ and $\vee$, and we can count the number of alternations in that.)

(Added by KWR: A fourth measure is the number of distinct propositional variables used in the proof; this is relevant to Sam's result on "variable renaming" being as powerful as general substitution (Theorem 6.3 below). This is like a measure of *space*; see Immerman [Imm91] for a halfway-related result. A fifth notion can be stated as follows: Draw a graph G whose nodes are the formulas $\psi_1, \ldots, \psi_n$ in the proof. Instances of axioms are sinks, while ψ_n is a (not necessarily lone) source. Now suppose ψ_i is derived by a k-ary rule of inference R, so that there are $j_1, \ldots, j_k < i$ such that $R(\psi_{j_1}, \ldots, \psi_{j_k}) = \psi_i$. Then draw arcs from ψ_i to each of $\psi_{j_1}, \ldots, \psi_{j_k}$. If there is more than one way that ψ_i could be derived, make some unique choice. Then G is a DAG that represents the "flow of logic" in the proof. The maximum length of a path from ψ_n to a sink is called the *height* of the proof, to avoid confusion with the above notion of depth. If G is a tree, except for multiple edges into axioms, then the proof is *tree-like*. Related "proof-structure" graphs and measures come into play in others of Sam's papers, particularly for *Gentzen systems* rather than Frege systems.)

Theorem 2.1 yields an inductive procedure that cranks out a proof of any given tautology $\psi \in V$. However, the induction causes exponential blowup in both symbol-length and the number of lines as a function of the length of ψ (or more specifically, as a function of the number of logical connectives in ψ). This proof is no better or worse than that obtainable by slogging through the truth table. The \$64,000 question (if you want \$64,000 Canadian, you'll need to collect about \$100,000 Barbadian) is:

Open Problem 1. Do the tautologies have polynomial-size Frege proofs? I.e., is there a polynomial p such that for all $\psi \in V$, there exists a proof Π of ψ of length at most $p(|\psi|)$?

If so, then NP = coNP! This is because a nondeterministic TM on input ψ can guess Π, and then in deterministic polynomial time verify that Π is correct—this only requires checking that each concrete axiom or inference in Π belongs to one of finitely many schemas. This would place the coNP-complete set V into NP.

2.3 Robustness of proof systems

The following "robustness theorems" of Cook and Reckhow [CR79, Rec76] show that the particular choice of a Frege system does not matter for our present purposes. First we consider Frege systems over the same language, such as $\{\wedge, \vee, \neg, \rightarrow\}$.

Theorem 2.2 ([CR79, Rec76]) *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be Frege systems over the same language. Then there is a constant $c > 0$ such that for all ϕ and n, if $\mathcal{F}_1 \vdash^n \phi$, then $\mathcal{F}_2 \vdash^{\leq cn} \phi$.*

Proof. (Inserted by KWR, following p40 of [CR79].) For every schematic axiom A of $\mathcal{F}_1$, let η_A be the "smallest instance" of A as defined above. By completeness, there is an $\mathcal{F}_2$-proof π_A of η_A. Likewise, for every schematic rule R of $\mathcal{F}_1$, take the "smallest instance" $R(\eta_1, \ldots, \eta_k) = \eta_0$. Then there is an $\mathcal{F}_2$-proof π_R of η_0, in which $\eta_1, \ldots, \eta_k$ appear as hypotheses. The neat point (Lemma 2.5 in [CR79], proved simply by induction on formula structure) is that for every substitution σ, the formula $\sigma(\eta_A)$ has the $\mathcal{F}_2$-proof $\sigma(\pi_A)$ defined by applying σ to all formulas in π_A. (Here one can arrange that π_A has no variables other than those in η_A, or one can define σ to be the identity on other variables.) Likewise, $\sigma(\pi_R)$ is an $\mathcal{F}_2$ proof of $\sigma(\eta_0)$ from $\sigma(\eta_1), \ldots, \sigma(\eta_k)$.

Now let π_1 be a proof of ϕ in $\mathcal{F}_1$. For every instance ψ of a schematic axiom A in π_1, let σ be the (effectively unique) substitution such that $\sigma(\eta_A) = \psi$ and let the $\mathcal{F}_2$ proof π_2 have the sequence $\sigma(\pi_A)$ in place of the occurrence of ψ. For every application $R(\psi_1, \ldots, \psi_k) = \psi$ of a rule, there is a single substitution σ such that $\sigma(\eta_1) = \psi_1, \ldots, \sigma(\eta_k) = \psi_k$ and $\sigma(\eta_0) = \psi$. By induction we may suppose that $\psi_1, \ldots, \psi_k$ have already been proved in the $\mathcal{F}_2$-proof we are building. Hence they fill the roles of the hypotheses in the $\mathcal{F}_2$-proof sequence π_R, and so we need only splice the remainder of π_R into the segment of π_2 corresponding to the occurrence of ψ in π_1.

For the size analysis, let the constant K be the maximum of $|\pi_A|$ or $|\pi_R|$ over the finitely many schematic axioms and rules of $\mathcal{F}_1$. The key is that for every substitution σ,

$$|\sigma(\pi_A)| \leq |\pi_A| \cdot |\sigma(\eta_A)| \leq K \cdot |\sigma(\eta_A)|.$$

A similar inequality holds for instances of rules R and proof segments π_R. This says that for every occurrence of a formula ψ in π_1, the symbol-length of the segment of the $\mathcal{F}_2$-proof π_2 corresponding to that occurrence is linear in ψ. $\square$

In order to talk about simulations between Frege systems over *different* languages, we must first fix a translation from formulas ϕ of one system to "equivalent" formulas ϕ' of the other. There are two problems to overcome here:

(1) First, something more than formal equivalence of ϕ and ϕ' has to be meant, because all tautologies are formally equivalent. One needs a notion that the translated formula ϕ' has the same "meaning" or "structure" as ϕ.

(2) A "direct translation," by which one means substituting an equivalent composition of functions in $\mathcal{L}_2$ for each $\mathcal{L}_1$-connective in the formula ϕ, may not be polynomial-time computable. Consider the languages $\mathcal{L}_1 = \{\wedge, \oplus, \neg\}$ and $\mathcal{L}_2 = \{\wedge, \vee, \neg\}$, and ϕ over $\mathcal{L}_1$ of the form $\phi = \phi_1 \oplus \phi_2$. One can define the translation

$$\phi' = (\phi_1 \wedge \neg\phi_2) \vee (\neg\phi_1 \wedge \phi_2),$$

but doing this slavishly recursively leads to exponential blowup if ϕ has linear nesting depth.

Reckhow solved these problems in his thesis [Rec76]. His translation scheme distinguishes systems over binary connectives and distinguishes $\{\leftrightarrow, \oplus\}$ from the other binary connectives. Reckhow showed that among the binary connectives apart from $\{\leftrightarrow, \oplus\}$, "direct translations" have polynomial-size overhead, and that translating *from* a formula over the standard basis $\{\wedge, \vee, \neg\}$ poses no special difficulty. The key is how to translate from formulas ϕ over arbitrary bases into the standard basis. He presented a uniform way to do this via the method of Spira [Spi71] (see also [BCE91, BB94]), which implicitly "re-balances" ϕ during the recursion. (Sam remarked that [Spi71] was a conference in Hawaii, thus spiritual kin to the present gathering.) Reckhow called this an "indirect translation." The above all combines to define a unique translation from any given language $\mathcal{L}_1$ to a given language $\mathcal{L}_2$, and we can combine Reckhow's terms "direct" and "indirect" and call this the *natural translation* from $\mathcal{L}_1$ to $\mathcal{L}_2$.[2]

Theorem 2.3 ([Rec76]) *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be any two (sound and complete) Frege systems, and let $\phi \mapsto \phi'$ be the natural translation from $\mathcal{F}_1$ to $\mathcal{F}_2$. Then there is a polynomial p such that for every $\mathcal{F}_1$-proof π of a tautology ϕ, there is an $\mathcal{F}_2$-proof π' of ϕ' such that $|\pi'| \leq p(|\pi|)$. Moreover, π' is computable in polynomial time given π.*

Proof Sketch. The main task is to verify that the Spira-based translation into the standard basis has only polynomial blowup. Then one can adapt the proof of the last theorem. Consider first the case where $\mathcal{L}_1$ has binary connectives only, including $\oplus$ and/or $\leftrightarrow$, so that ϕ is a binary tree. The basic lemma, used earlier by Hartmanis and Stearns [HS65] to put CFLs in DSPACE[$\log^2 n$], is that every binary tree T has a subtree S satisfying $\frac{1}{3}|T| \leq |S| \leq \frac{2}{3}|T|$.

To apply this lemma, make a new tree by adding a new root node labeled $\vee$, with two new $\wedge$ nodes as children. One $\wedge$ has S as one child, and the other child is what you get from $T \setminus S$ by substituting a '1' for the edge from the parent of S to $T \setminus S$. The other $\wedge$ node has for children the negation of S and the result of a '0' substitution on that edge into $T \setminus S$. This process is continued recursively on the four subtrees. At the bottom, we will be down to positively and negatively signed literals—all the $\oplus$ and $\leftrightarrow$ magically go away in the steps that simplify $T \setminus S$. The resulting tree has at most $2d$ levels of alternating $\vee$ and $\wedge$, where $d = \log_{3/2}|T| = O(\log|T|)$. The size is at most $O(|T|^2)$.

In the case of, say, a 6-ary connective in $\mathcal{L}_1$, one would use a "$\frac{1}{7}$, $\frac{6}{7}$" tree-splitting lemma to get a similar log-depth, poly-size translation, though

[2](Added by KWR: In their paper [CR79], Cook and Reckhow skipped these details, saying that the p-simulation results for the "more natural" extended Frege systems (where the above growth problem can be circumvented) superseded them in importance.)

with a larger polynomial blowup in size.

Once this translation is done, the rest is similar to the proof of Theorem 2.2. $\square$

This prompted some discussion and related points. The same rebalancing idea shows that polynomial-size formulas equal non-uniform NC^1. An interesting question is whether the quadratic blowup above in the binary $\oplus$ case above can be improved to (nearly) linear. A recent IPL paper by Bonet and Buss [BB94] shows that with a more-involved scheme for chopping up the tree T, one can get a sub-quadratic simulation. (Is this related to Ben-Or and Cleve's sub-quadratic simulation of Barrington's branching-program theorem [BC88]? See also [CL94, Bar89].) Here I (KWR) feel it may be good to study a recent paper by Kosaraju and Delcher [KD90, KD92], which is based on a notably different idea of "fracturing" a tree of size t into many little pieces, viz. $\sqrt{t}$-many subtrees, each of size roughly $\sqrt{t}$.

2.4 Resolution Proof Systems

A resolution *clause* is a finite set of literals, and stands for the disjunction of the literals. For convenience, we use * as a function to negate literals, so that $p_i^* = (\neg p_i)$ and $(\neg p_i)^* = p_i$.

Definition 2.3. A clause α_3 is inferred from two clauses α_1, α_2 by the resolution rule

$$\cdot \frac{\alpha_1 \qquad \alpha_2}{\alpha_3} \text{Res}$$

precisely when

(a) there is a unique literal x such that $x \in \alpha_1$ and $x^* \in \alpha_2$, and

(b) $\alpha_3 = (\alpha_1 \cup \alpha_2) \setminus \{ x, x^* \}$.

The resolution rule is a variant of modus ponens. In the case $\alpha_1 = \{ x \}$ and $\alpha_2 = \{ x^* \} \cup B$, where B is arbitrary, α_2 is equivalent to $\alpha_1 \rightarrow B$, and the resolution rule yields B just as MP does. The same kind of thing holds for bigger clauses α_1. The restriction that x be unique in (a) loses no generality—consider for instance $\alpha_1 = (x \vee y \vee w)$ and $\alpha_2 = (x^* \vee y^* \vee z)$. Resolving on x alone would yield the useless tautological clause $(y \vee y^* \vee w \vee z)$, while eliminating x and y together to get $(w \vee z)$ is *unsound*.

Resolution is used to prove formulas ϕ that are in DNF, by obtaining a *resolution refutation* of the CNF formula $(\neg\phi)$, namely a sequence of resolution inferences that produces the empty clause. This refutation is also called a *resolution proof* of ϕ. The so-called *Davis-Putnam procedure* is an exponential-time deterministic algorithm that always produces a resolution proof of a given DNF tautology ϕ. This shows that resolution is *complete* for DNF formulas, and it is also clear that the resolution rule is *sound*.

Note that if the Davis-Putnam procedure ran in polynomial time, then P would equal NP. The fact that it runs in polynomial time for ϕ of clause size two is the proof that 2-*SAT* belongs to P. The Davis-Putnam procedure and various refinements of it work in polynomial time for other classes of formulas, and are used all the time as heuristics in practical applications. (Someone remarked that two main reasons for its popularity are that it's easy to program, and that "it's the kind of 'blind dumb search' on strings of symbols that computers are good at." Not to be too pejorative about this: there are many well-developed heuristics to guide this search, and they extend to general first-order resolution as well. Resolution is very important in practice.) It appears that Tseitin [Tse68] was the first to study the lengths of individual resolution proofs and of shortest proofs for a given ϕ—note that this is a different matter from the running time of a deterministic algorithm that generates a resolution proof of ϕ.

Allender brought up the limitation of resolution to DNF formulas. Every Boolean formula ψ has a DNF equivalent with the same variables, but with an exponential blowup in size. One can instead add new variables to obtain a polynomial-sized ψ' that is a tautology iff ψ is. (This is basically the same idea as the standard reduction from general Boolean formula satisfiability to CNF-SAT as presented in, say, [HU79].) These extra "extension variables," however, create more work for resolution itself.

One cause of theoretical interest in resolution is that in quite a few cases, both upper bounds and *matching* lower bounds of exponential proof size have been proved. More will be said about the complexity of resolution proofs later. But first, let us consider what seems to be the most abstract sensible idea of a proof system.

3 Abstract Proof Systems

The following definition was put forward by Cook et al. in much the same breath of research as the fundamental NP-completeness results.

Definition 3.1. An *abstract propositional proof system* is a polynomial-time computable function f such that $Ran(f) = V$; i.e., the range of f is the set of all Boolean tautologies. An *f-proof* of a formula A is a string w such that $f(w) = A$.

Note that f need not be *polynomially honest*; i,e, there need not be a polynomial p such that for all w, $p(|f(w)|) > |w|$. If f is honest, then all proofs have polynomial size. The following example makes this point clear:

Example 3.1. A given Frege proof system $\mathcal{F}$ yields the function $f_{\mathcal{F}}(w) = A$ if w is a valid $\mathcal{F}$-proof of A, and $f_{\mathcal{F}}(w) = (p_1 \ \vee \ \neg p_1)$ otherwise.

One point of the general definition is that strong theories $\mathcal{F}$ such as PA or ZF can also be used as propositional proof systems; the only condition is that the theory is encoded in such a manner that validity of proofs can be checked in polynomial time. This can be done by *fiat* by defining f so that w is not a ZF proof itself, but the computation of a Turing machine that checks validity of ZF proofs. So one can actually use this idea for any recursively axiomatizable theory. But by and large all proof systems that have been studied already have polynomial-time proof-checking under their natural encodings. Two other examples are *cutting-planes proof systems*, originated by W. Cook, C. Coullard, and G. Turán in [CCT87], and "quantified propositional logic" (see [Dow85, KP90]).

Definition 3.2. A proof system f is *super* if every propositional tautology has a polynomial-size f-proof.

A super proof system can be modified to an equivalent proof system f' in which f' is polynomially honest after all.

Theorem 3.1 ([CR79]) *There exists a super proof system iff* NP = coNP.

Proof. A super proof system would place the tautologies into NP, but V is coNP-complete under polynomial-time many-one reductions. For the converse, suppose NP = coNP, and let N be a polynomial-time NTM that recognizes V. Then define $f(w) = A$ if $w = \langle A, c\rangle$, where c is an accepting computation of N on input A, and $f(w) = (p_1 \vee \neg p_1)$ otherwise. This f is super. $\square$

This theorem is one of the prime motivations for the study of propositional proof length—it justifies the study of upper and lower bounds on proof length for a variety of concrete propositional proof systems. Another motivation from the practical side comes from building efficient automated deduction systems. (Added by KWR: The statement I've heard of "Cook's program for proving NP $\neq$ coNP" runs: Start with concrete proof systems $\mathcal{F}_1, \mathcal{F}_2, \ldots$, prove that $\mathcal{F}_1$ is not super, prove that $\mathcal{F}_2$ is not super, until one gains enough knowledge to understand why *no* proof system can be super.)

Here it would be nice to have a theorem of the form "System $\mathcal{F}$ is super iff a super proof system exists." Is there in some useful sense a system $\mathcal{F}$ for which one can demonstrate this concretely? By "concretely" we have in mind giving a reducibility relation $\leq$ that preserves upward the property "is super," and showing that $\mathcal{F}$ is "complete" under $\leq$.

Definition 3.3 ([CR79]). Let $\mathcal{F}$ and $\mathcal{F}'$ be proof systems with the same propositional language. Then say $\mathcal{F}'$ *simulates* $\mathcal{F}$ if there is a polynomial p such that for every tautology ϕ and $\mathcal{F}$-proof π of ϕ, there exists an $\mathcal{F}'$-proof π' of ϕ such that

$$|\pi'| \leq p(|\pi|).$$

Also say that $\mathcal{F}'$ *p-simulates* $\mathcal{F}$ if the mapping from π to π' is polynomial-time computable.

Similar definitions can be made when $\mathcal{F}$ and $\mathcal{F}'$ have different languages, provided a "natural translation" from $\mathcal{F}$ to $\mathcal{F}'$ is given.

Definition 3.4. A proof system is *optimal* if it p-simulates every other proof system.

Open Problem 2. (1) Does there exist an optimal proof system?

(2) Are Frege systems optimal?

Let E stand for DTIME$[2^{O(n)}]$, and NE for the corresponding nondeterministic time class. Krajíček and Pudlák [KP89] showed that NE $=$ E is a sufficient condition for (1). They also show that if NE $=$ co-NE, then there is a propositional proof system that is optimal in the weaker non-uniform sense of simulating every other proof system.

If (2) holds, then this would be a striking "complexity-theoretic conservation' result," saying that ordinary propositional logic is as efficient as anything powerful axioms like those in ZF set theory can do on propositional tautologies. Barrington remarked that refuting (2) would be "like" showing that NP $\neq$ ALOGTIME, though no clean-cut connection here is known. Haken [Hak85] proved that certain families of DNF tautologies require exponential length for resolution proofs, so resolution proof systems cannot be optimal even for DNF tautologies.

(Added by KWR: Here's a stab at an optimal system that shows some of the issues. Let $N_1, N_2, \ldots$ be a standard recursive enumeration of polynomial-time NTMs, and define a given N_i to be *sound* if $L(N_i) \subseteq V$. Now define

$$\begin{aligned} f(w) = \phi \quad \text{iff} \quad & (\exists i, c, v)[w = \langle \phi, \pi, c \rangle \ \wedge \\ & c \text{ is an accepting computation of } N_i \text{ on input } \phi \ \wedge \\ & \pi \text{ is a proof in ZF that } N_i \text{ is sound.} \end{aligned}$$

However, this is just p-simulation equivalent to ZF: Given a ZF proof π of ϕ, let then N_i be an (N)TM such that $L(N_i)$ transparently equals $\{\,\phi\,\}$; then π is essentially a proof that N_i is sound, and the rest of w is clear. Conversely, given a proof w in this system, the component π of w is already a proof in ZF, and the c component maps to a proof in ZF that N_i accepts ϕ of roughly the same size. Hence this system is super iff ZF is super.

Well, I was trying to do something interesting with universal NTMs and/or consistency or arithmetical soundness statements about ZF, but this isn't it. Perhaps some known "conservation results" for ZF over arithmetical theories can be applied? The problem of whether there are "super-complete" or optimal proof systems seems to me rather different from analogous complexity questions, such as whether NP $\cap$ coNP has complete sets. Sam remarked in Barbados that we don't have any complexity-theoretic *consequences* of the existence of optimal proof systems.)

The next section describes a system that is suspected to be properly higher than the Frege systems under p-simulation.

4 Extended Frege Systems

Given an ordinary Frege system $\mathcal{F}$, an *extended Frege* proof, $e\mathcal{F}$ proof for short, is a sequence of formulas $A_1, A_2, A_3, \ldots$ such that for all i, either A_i follows from earlier formulas by a rule of $\mathcal{F}$, or A_i is an axiom instance of $\mathcal{F}$, or else A_i is an *extension formula* of the form

$$p_i \leftrightarrow \phi$$

where ϕ is any formula and p_i is a fresh "extension variable"; i.e., p_i occurs neither in ϕ nor in any of $A_1, \ldots, A_{i-1}$. We also speak of '$p_i \leftrightarrow \phi$' as inferred by the *extension rule*. The final formula A_n in the proof is not allowed to have any extension variables.

The idea is that p_i can now be used as an abbreviation for ϕ at all subsequent steps of the proof. This can reduce the proof size (number of symbols) greatly, though the number of steps is not reduced. Proof using these rules can still be checked in polynomial time, so this is also a polynomial abstract proof system. This notion was studied by Tseitin [Tse68], Statman [Sta77], and by Cook and Reckhow [CR79, Rec76]. With the same provision about "natural translations" in case the underlying Frege systems are over different languages:

Theorem 4.1 ([CR79]) *Any two extended Frege proof systems p-simulate each other.*

Hence we can regard "$e\mathcal{F}$" as a single proof system, just as "$\mathcal{F}$" itself is a single Frege system, up to p-simulation equivalence. Since $e\mathcal{F}$ is a polynomial abstract proof system, it follows that if all tautologies have polynomial-size $e\mathcal{F}$ proofs, then NP = coNP. Statman proved that the size measure of $e\mathcal{F}$ proofs is essentially the same as the *step* measure of ordinary Frege proofs.

Theorem 4.2 ([Sta77]) *For any Frege proof of step-length n of a formula A, A has an $e\mathcal{F}$-proof of size $O(n + |A|)$.*

The proof idea is that the $e\mathcal{F}$-proof introduces abbreviations for every "active" subformula in the Frege proof of A.

Open Problem 3. (1) Can Frege proof systems (p)-simulate $e\mathcal{F}$ systems?

(2) Are $e\mathcal{F}$ proof systems optimal?

A major testing ground for problem (1) is the subject of the next section.

5 The Propositional Pigeonhole Principle

For each $n \geq 0$, we define a propositional formula PHP_n that expresses the *pigeonhole principle* for $n+1$ "pigeons" and n "holes." For each "pigeon" i and "hole" k, we allocate a propositional variable p_{ik}, with the intent that setting this variable *true* means that pigeon i has been placed into hole k. If we think of a function f from $[n{+}1] = \{0, \ldots, n\}$ into $[n] = \{0, \ldots, n{-}1\}$, then truth of p_{ik} signifies $f(i) = k$. The condition that every pigeon goes to some hole is encoded by the antecedent $\bigwedge_{i=0}^{n} \bigvee_{k=0}^{n-1} p_{ik}$. The "Pigeonhole Principle" states that then some hole must have more than one pigeon, expressed by $\bigvee_{k=0}^{n-1} \bigvee_{0 \leq i < j \leq n} (p_{ik} \wedge p_{jk})$. In full, for each $n \geq 1$,

$$PHP_n = \bigwedge_{i=0}^{n} \bigvee_{k=0}^{n-1} p_{ik} \quad \rightarrow \quad \bigvee_{k=0}^{n-1} \bigvee_{0 \leq i < j \leq n} (p_{ik} \wedge p_{jk}). \tag{1}$$

For example, with $r = n - 1$, $PHP_0 =$ "false $\rightarrow$ false," and:

$$\begin{aligned} PHP_1 &= p_{00} \wedge p_{10} \rightarrow p_{00} \wedge p_{10}, \\ PHP_2 &= (p_{00} \vee p_{01}) \wedge (p_{10} \vee p_{11}) \wedge (p_{20} \vee p_{21}) \rightarrow \\ & \quad (p_{00} \wedge p_{10}) \vee (p_{01} \wedge p_{11}) \vee (p_{00} \wedge p_{20}) \vee \\ & \quad (p_{01} \wedge p_{21}) \vee (p_{10} \wedge p_{20}) \vee (p_{11} \wedge p_{21}). \end{aligned}$$

(Added by KWR: First, a historical note: Every Oxford and Cambridge college has a "Porter's Lodge" with racks of pass-through mail slots for Fellows and students. These are called "pigeonholes" because of the resemblance to a bird cote; indeed, the twice-daily college mail system is called the "pigeon post." Among various accounts one hears for the origin of the term "Pigeonhole Principle," the one I give most credence is that a century-or-two ago, a don at Trinity College, Cambridge, mused that since the College had n Fellows, if $n + 1$ letters arrived in the post, at least one lucky Fellow was bound to get at least two letters. However, Sam, Steve Maurer when I took a course from him, and most others I've heard speak about $n+1$ pigeons going into n holes, which sounds more like how Andy Capp than an Oxbridge don would say it. Another form mentioned by Sam is the "Dirichlet Box Principle."

Second, note that the left-hand side of (1) does not actually define a function from $[n+1]$ into $[n]$—it figuratively allows a pigeon to be assigned to more than one hole! To encode faithfully the statement that "every function from $[n+1]$ into $[n]$ is non-injective," we must conjoin to the left-hand side the clause

$$\bigwedge_{i=0}^{n} \bigwedge_{0 \le k < \ell \le n-1} (\neg p_{ik} \ \vee \ \neg p_{i\ell}).$$

Call the resulting statement PHP'_n. Note that this is intuitively a weaker assertion than PHP_n. A relevant technical counterpart to the idea of "weaker" is that from hypothesis PHP_n one can derive PHP'_n in just a few more proof lines and symbols; but the converse direction is not so clear. Sam interchanged PHP_n and PHP'_n rather freely in his lectures, and (later communication from him) this can be justified by defining

$$q_{ij} \equiv \text{`}g(i) = j\text{'} \leftrightarrow p_{ij} \ \wedge \ (\neg p_{i0} \ \wedge \ \neg p_{i1} \ldots \ \wedge \ \neg p_{i,j-1}).$$

This translation "factors through" to adapt the following proofs of PHP'_n into proofs of PHP_n. The adaptation also preserves constant-depth proofs.

Ajtai [Ajt94] worked with a still-weaker third form that restricts attention to functions f that are *onto* $[n]$, obtained by conjoining also

$$\bigwedge_{k=0}^{n-1} \bigvee_{i=0}^{n} p_{ik}$$

onto the left-hand side of (1). The neatly symmetrical form given by Ajtai for this, which we may call PHP''_n, is:

$$\neg \quad \left[\left(\bigwedge_{i=0}^{n} \bigvee_{k=0}^{n-1} p_{ik} \right) \quad \wedge \quad \left(\bigwedge_{k=0}^{n-1} \bigvee_{i=0}^{n} p_{ik} \right) \ \wedge \right.$$
$$\left. \left(\bigwedge_{i=0}^{n} \bigwedge_{0 \le k < \ell \le n-1} (\neg p_{ik} \ \vee \ \neg p_{i\ell}) \right) \quad \wedge \quad \left(\bigwedge_{k=0}^{n-1} \bigwedge_{0 \le i < j \le n} (\neg p_{ik} \ \vee \ \neg p_{jk}) \right) \right]. \tag{2}$$

The point offered by Ajtai is that for *negative* results such as those in his paper, using the weakest form (PHP''_n) makes the results apply to the others, while for *positive* results such as those given by Sam, one should use the strongest form (PHP_n). Sam (later communication) remarks that PHP''_n versus $\{ PHP_n, PHP'_n \}$ "sometimes makes a difference.")

The form (2) makes it clear that for all n, PHP_n and the two other forms are (transparently equivalent to) DNF formulas. Hence they are "in-bounds" for resolution, as well as for Frege proof systems. Kreisel was apparently the first to discuss the propositional forms of the pigeonhole principle, but the first in-depth analysis from the complexity point of view owes to the same paper by Cook and Reckhow that we have been discussing all along. Note that for each n, PHP_n has size proportional to n^3, so up to "polynomial scaling," it is OK to express complexity bounds in terms of n rather than the true size of PHP_n.

Theorem 5.1 ([CR79]) *The tautologies PHP_n have polynomial-size extended Frege proofs.*

The sketch of their proof is a good example of the effect of the extension rule for cutting down proof size. The conceptual pattern is a proof by reductio-ad-absurdum from the counterfactual hypothesis that f is 1-1. (Note by KWR: Sam spoke in these functional terms, and Cook and Reckhow actually used PHP'_n. However, the Cook-Reckhow proof works immediately for the most general case (i.e., for PHP_n), and I've re-worked what was on Sam's slides to make this plain. Intuitively speaking, we're allowing f to be a "multi-valued" function with domain $\{0, \dots, n\}$ and values in $\{0, \dots, n\text{–}1\}$, and the right-hand side of (1) is read as saying that f must have a "collision." I've written F in place of f for the possibly-multivalued case.)

Proof. Write $f : [n] \overset{\text{1-1}}{\to} [n-1]$, where $[n]$ is short for $\{0, \dots, n\}$. Define $f_m : [m] \to [m-1]$ inductively for $m = n$ down to $m = 1$ by: $f_n = f$, and for $m \leq n-1$, $i \in [m]$:

$$f_m(i) = \text{if } f_{m+1}(i) < m \text{ then } f_{m+1}(i) \text{ else } f_{m+1}(m).$$

The idea is to prove the implication "f_m is 1-1 $\to$ f_{m-1} is 1-1" for each m, reducing things down to the assertion that $f_1 : \{0, 1\} \to \{0\}$ is 1-1, which is clearly *absurdum*. Frege proofs can do reductio-ad-absurdum.

More generally, let F stand for a possibly-multivalued function with domain $[n]$ and values in $[n-1]$. Then define $F_n = F$ and inductively for $m < n$:

$$F_m(i) \mapsto k \quad \text{if } k < m \text{ and } [F_{m+1}(i) \mapsto k \vee (F_{m+1}(i) \mapsto m \wedge F_{m+1}(m) \mapsto k)].$$

If one lets B_m stand for the formula asserting that F_m has domain $[m]$, then $B_{m+1} \rightarrow B_m$ is immediate, since

$$F_{m+1}(i) \mapsto m \ \vee \ (\bigvee k < m)\ F_{m+1}(i) \mapsto k$$

follows from B_{m+1}. Now consider what happens if F_m has a collision; i.e., if there exist $i < j \in [m]$ and $k \in [m-1]$ such that $F_m(i) \mapsto k$ and $F_m(j) \mapsto k$. If both values can arise from the first disjunct, then $F_{m+1}(i) \mapsto k$ and $F_{m+1}(j) \mapsto k$, contradicting the assertion that F_{m+1} has no collisions. If both can arise from the second disjunct, then $F_{m+1}(i) \mapsto m$ and $F_{m+1}(j) \mapsto m$, ditto. If one value, say $F_m(j) \mapsto k$, comes from the first disjunct and the other comes from the latter, we have $F_{m+1}(j) \mapsto k$ and $F_{m+1}(i) \mapsto m$ and $F_{m+1}(m) \mapsto k$. This is also a contradiction since $i, j \leq m-1$. Thus we also get a reductio-ad-absurdum, starting from F_n having no collisions and ending with F_1 having no collisions as a multivalued function from $\{0, 1\}$ to $\{0\}$, which contradicts the assertion B_1 that F_1 is total.

Now for the extended Frege proof, we introduce new "extension variables" q^m_{ik}, each intended to stand for "$f_m(i) = k$," or in the multi-valued case, "$F_m(i) \mapsto k$." Recall that we start with variables p_{ik} expressing "$f(i) = k$" or "$F(i) \mapsto k$," The extension rules that introduce the new variables are

$$\begin{aligned} q^n_{ik} &\leftrightarrow p_{ik} \text{ and for } m < n, && (3)\\ q^m_{ik} &\leftrightarrow q^{m+1}_{ik} \vee (q^{m+1}_{im} \wedge q^{m+1}_{mk}). && (4)\end{aligned}$$

The *whole point* is that by using these extension variables at each stage, the formal proof of each stage in the "reductio ad absurdum" *has the same symbol-length*. Indeed, mimicking the prose details given above for the multi-valued case, the symbols used in each stage are the same except for the superscripts m. Since each individual stage has polynomial symbol-length, the whole thing gets multiplied only by a factor of n, and is still polynomial. $\square$

However, if the above $e\mathcal{F}$ proof is converted into a Frege proof by removing the extension inferences and replacing the extension variables q^m_{ij} by the formulas they abbreviate, then the Frege proof would "bush out" by a factor of 3 for the right-hand side of (4) at each stage, giving roughly 3^n symbols.

That shows the difference between $\mathcal{F}$ and $e\mathcal{F}$ proofs. Note also here that the number of *lines* of the Frege proof is basically unaffected by the presence or absence of (4), in keeping with Statman's theorem.

For some time, the PHP_n formulas were considered a prime candidate for an exponential separation between $\mathcal{F}$ and $e\mathcal{F}$ proofs, which if established would be a major plank in Cook's program. But there is an alternative strategy that produces polynomial-sized Frege proofs of PHP_n. Intuitively speaking, it replaces the above use of induction by a clever stratagem for encoding *counting* into propositional formulas, and establishing some basic facts about counting via polynomial-sized Frege proofs. The stratagem involves the log-depth circuits for vector addition and counting from Ofman [Ofm63] and Wallace [Wal64].

Theorem 5.2 ([Bus87b]) *The formulas PHP_n do have polynomial-sized Frege proofs.*

(KWR: Sam's proof is for PHP'_n, but it can be adapted for PHP_n.)

Proof. For each ℓ, $0 \leq \ell \leq n-1$, define M_ℓ to be the number of i in the domain such that $f(i) \leq \ell$. Formally,

$$M_\ell := \|\{\, i \in [n] : \bigvee_{0 \leq k \leq \ell} p_{ik} \,\}\|.$$

Again we take the hypothesis that f is 1-1 (or "has no collisions"). Thus $M_0 \leq 1$, since otherwise there would be a collision at $k = 0$. The objective is to prove the successive inequalities:

$$\begin{array}{rcl} M_0 & \leq & 1 \\ M_1 & \leq & M_0 + 1 \\ M_2 & \leq & M_1 + 1 \\ & \vdots & \\ M_{n-1} & \leq & M_{n-2} + 1. \end{array}$$

From this it follows that $M_{n-1} \leq n$. However, given that $f : [n] \to [n-1]$ is total on $[n]$, it follows that $M_{n-1} = n + 1$. This contradiction finishes the outline of the Frege proof. It remains first to give a polynomial-size Frege proof of each step, and then to show that the deduction $M_{n-1} \leq n$ also can be done in polynomial size.

First we need an encoding scheme in propositional logic for the numerical quantities M_ℓ. We write M_ℓ in standard binary notation as an $(a+1)$-bit number, where $a = \lfloor \log_2(n+1) \rfloor$, using leading zeroes if necessary. To these bits we will associate a sequence of formulas

$$\hat{m}^\ell = m_a^\ell, m_{a-1}^\ell, \ldots, m_0^\ell,$$

where each m_i^ℓ expresses whether the corresponding ith bit is 1 or 0.

To do this, it suffices to define formulas $Count_i$, $0 \leq i \leq a$, such that

(1) Each $Count_i$ has exactly $n+1$ free variables $x_0, \ldots, x_n$.

(2) $Count_i$ gives bit i (i.e., the place for 2^i) of the binary representation of the number of x_j that are true.

(3) The formulas $Count_i$ behave correctly with regard to basic "intensional" properties of counting in binary, and these properties have polynomial-size Frege proofs.

(4) The size of $Count_i$ is polynomially bounded in n.

The idea is that for the $x_0, \ldots, x_n$ we are going to substitute formulas for "$f(0) \leq \ell, \ldots, f(n) \leq \ell$." Then the resulting formulas $Count_i^\ell$ define the bits m_i^ℓ.

Now properties (1), (2), and (4) are immediate to arrange because each $Count_i$ is a symmetric function of $n+1$ inputs, and every such function belongs to (non-uniform) NC^1. It remains to establish (3), and for this we have to say more concretely *how* $Count_i$ is defined.

First idea: Define the sum of two a-bit numbers n_y and n_z, represented by sequences of formulas

$$\begin{aligned} \hat{y} &= y_a, y_{a-1}, \ldots, y_0 \\ \hat{z} &= z_a, z_{a-1}, \ldots, z_0, \end{aligned}$$

by $Add_0(\hat{y}, \hat{z}) := y_0 \oplus z_0$, and for bits $i > 0$,

$$Add_i(\hat{y}, \hat{z}) := y_i \oplus z_i \oplus Carry_i,$$

where

$$Carry_i = \bigvee_{j<i} \left[y_j \ \wedge \ z_j \ \wedge \ \bigwedge_{j<k<i} (y_k \ \oplus \ z_k) \right].$$

The *problem* with this is that the "Carry" formulas have log depth in the big ANDs and ORs. To see the knock-on effect of this, let us follow through with our intended use of the formulas Add_i. This is a kind of "divide and conquer": For all i, j define the number

$$A^{ij} := \|\{ x_k : x_k = \top \ \wedge \ j{\cdot}2^i \leq k < (j+1){\cdot}2^i \}\|.$$

In other words, this is the number of x_k that are true in the jth segment of size 2^i in $0 \ldots n$. We can count up all these segments by the "d & c" recursion with basis

$$A^{0j} = \text{if } x_j \text{ then } 1 \text{ else } 0,$$

and induction

$$A^{i+1,j} = A^{i,2j} + A^{i,2j+1}.$$

To express A^{ij} by propositional formulas, we use the same encoding scheme as above, seeking formulas a_b^{ij} to represent the bits of A^{ij} in binary by the sequence

$$\hat{a}^{ij} = a_i^{ij}, \ldots, a_b^{ij}, \ldots, a_0^{ij}.$$

For $i = 0$, we have that a_0^{0j} is just x_j itself, and for bits $b > 0$, $a_b^{0j} = \perp$ (where we might encode $\perp$ by $(p_1 \ \wedge \ \neg p_1)$, say). Then the induction case is represented by

$$a_b^{i+1,j} = Add_b(\hat{a}^{i,2j}, \hat{a}^{i,2j+1}). \tag{5}$$

Then $Count_i$ is given by a_i^{a0}, i.e., by $a_i^{\log_2 n, 0}$.

Alas, when we analyze the size blowup of the recursion (5), we get the following: Each formula a_b^{ij} appears $\Theta(\log^2 n)$ times in $a_{b'}^{i+1,j'}$, for all $b' \geq b$, where $j' = \lfloor j/2 \rfloor$. Since we need to recurse up to $i = \log n$, we have in all $(\Theta(\log^2 n))^{\log n})$ occurrences of the base variables x_j. However, this equals $n^{\Theta(loglogn)}$, which is super-polynomial. Hence this scheme using the straightforward log-depth definition of Add_b is too big.

What we need to do to achieve polynomial size is to get the number of occurrences of variables "like" a_b^{ij} in something like (5) down from $O(\log^2 n)$ to *constant*; even $O(\log n)$ is not good enough. If our addition formula were constant-depth, we'd be OK. (At this point, Barrington and I chimed in that one could use a "redundant" or "signed bit" notation instead of standard binary; cf. the results on constant-depth addition in these notations by Borodin, Cook, and Pippenger [BCP83]. But such a notation scheme would cause other headaches. Sam added that since the numbers concerned have size $O(\log n)$ bits, he could also have used a "ripple carry" circuit.)

The idea used in Sam's proof is to achieve the same effect via *carry-save addition* (CSA), a trick that goes back to the 1960s. This converts three numbers n_0, n_1, n_2 into two numbers m_0, m_1 such that $n_0+n_1+n_2 = m_0+m_1$. Each bit b of m_0 is the bitwise sum mod 2 of the bits b of n_0, n_1, and n_2, while bit b of m_1 is 1 if the sum of these bits is 2 or 3 (which would be the carry propagated into the next column in the standard way of summing $n_0+n_1+n_2$), and 0 otherwise. Put another way, $m_{1b}m_{0b}$ equals $n_{0b}+n_{1b}+n_{2b}$ as a binary number between 0 and 3. We may also define "$m_{1,-1}$" to be 0.

Now define the following "double-barreled" recursion:

$$C^{0,j} = 0, \qquad S^{0,j} = \text{if } x_j \text{ then 1 else 0.}$$

and for $1 \leq i \leq a-1$,

$$(C^{i+1,j}, S^{i+1,j}) = \mathit{CSA}(\mathit{CSA}(C^{i,2j}, S^{i,2j}, C^{i,2j+1}), S^{i,2j+1}). \tag{6}$$

The effect of the two applications of CSA is to give a constant-depth adder from four numbers into two. Finally, $C^{ij} + S^{ij}$ gives the desired quantity A^{ij}.

Now do the propositional translations $\hat{s}^{ij}$ and $\hat{c}^{ij}$ of C^{ij} and S^{ij} as before. The payoff is that in the propositional translation of (6), each s_b^{ij} and c_b^{ij} appears some constant k number of times in $s_b^{i+1,j'}$ and $c_{b+1}^{i+1,j'}$, where $j' = \lfloor j/2 \rfloor$ as before. Hence the base variables x_j appear at most $k^{\log n} = n^k$ times in $s_b^{\log n,0}$ and $c_b^{\log n,0}$. Finally, the propositional translation $\hat{a}^{ij}$ of A^{ij} is obtained by one application of the original $\mathit{Add}_b(\cdot,\cdot)$ formulas. Then from $\hat{a}^{ij}$ we obtain propositional representations of Count_i as before, and these have polynomial size.

It remains to verify that the formulas Count_i thus obtained are equivalent to the earlier definition in terms of the quantities A^{ij}, and that the nice properties listed above carry through. Then polynomial-sized Frege proofs of the sequence of inequalities for the M_ℓ can be put together. The details are long but straightforward and not so interesting, "hence best omitted." $\square$

5.1 A note on circuit complexity and proof-system simulations

There is a natural connection between extended Frege systems and Boolean circuits, since both models allow the introduction of "abbreviations." The formulas in any line of an $e\mathcal{F}$-proof can be transformed into circuits, with the idea that the symbols that come from extension rules correspond to gates with fan-out greater than 1. Conversely a circuit can be transformed into a formula in an extended Frege proof by introducing new variables for each internal node of the circuit. Since the Circuit Value Problem is complete for the class P [Lad75], one would expect some relationship between $e\mathcal{F}$-proofs and polynomial time computations. This intuition however has not been formalized yet (as the reader can judge from the vagueness of this paragraph—KWR).

Similarly Frege proofs can be connected to Boolean formulas, which are the same as Boolean circuits of fan-out 1 at each gate. Since the Boolean Formula Value Problem is in NC^1 [Bus87a, BCGR92, Bus93], one tends to associate Frege-proofs with NC^1 computations. It would be very interesting to formalize these connections, showing for example that a polynomial simulation of $e\mathcal{F}$ proofs by $\mathcal{F}$ proofs is possible if and only if the complexity classes P and NC^1 coincide (in the nonuniform case).

6 More Propositional Proof Systems and Simulations

In this section we consider variations of the Frege proof systems that arise when allowing different substitution rules. As we will see, all these systems are polynomially equivalent to the $e\mathcal{F}$ proof systems.

Definition 6.1. A substitution Frege ($s\mathcal{F}$) proof system is a Frege system augmented with the substitution rule

$$\frac{\varphi(p)}{\varphi(\psi)}$$

that indicates that every occurence of the variable p in the formula φ, is replaced by the formula ψ.

As the next result shows, Frege systems with substitution are as powerful as extended Frege systems.

Theorem 6.1 *Given a $s\mathcal{F}$ and an $e\mathcal{F}$ system with the same language, then*

(a) [CR79] *the $s\mathcal{F}$ system p-simulates the $e\mathcal{F}$ system.*

(b) [Dow85, KP89] *the $e\mathcal{F}$ system p-simulates the $s\mathcal{F}$ system.*

Proof. (a) Let A be a formula and P be an $e\mathcal{F}$ proof of A using the extension rules $p_1 \leftrightarrow B_1, \ldots p_k \leftrightarrow B_k$. Let $P = A_1, \ldots, A_n$ with $A_n = A$. It is not hard to see that for every A_j, $1 \leq j \leq n$, there is a polynomial size Frege proof of the formula

$$\left(\bigwedge_{i=1}^{k} p_i \leftrightarrow B_i \right) \rightarrow A_j.$$

In particular there is a Frege proof Q of size $O(|P|^3)$ of

$$\left(\bigwedge_{i=1}^{k} p_i \leftrightarrow B_i \right) \rightarrow A.$$

We can suppose that the $p_i \leftrightarrow B_i$ are numbered in reverse order of how they appear in P so that p_i does not appear in A_j for $j > i$. The last line of Q can be written as

$$(p_1 \leftrightarrow B_1) \ \wedge \ \left(\bigwedge_{i=2}^{k} p_i \leftrightarrow B_i \right) \rightarrow A,$$

substituting B_1 for p_1 and removing $B_1 \leftrightarrow B_1$ one gets

$$\left(\bigwedge_{i=2}^{k} p_i \leftrightarrow B_i \right) \rightarrow A.$$

This process can be repeated k times until A is obtained, and therefore this formula has a substitution Frege proof of size $O(|P|^3)$.

(b) Let $\varphi_1, \varphi_2, \ldots, \varphi_r$ be an $s\mathcal{F}$-proof of φ_r with propositional variables $p_1, \ldots, p_m$. We need to construct an $e\mathcal{F}$ proof of φ_r. The $e\mathcal{F}$-proof will have the set of extension variables $q_1^{i,k}, \dots, q_m^{i,k}$ for $1 \le i \le k \le r$, and will prove successively the formula

$$\neg\varphi_r \to \bigvee_{i=1}^{k} \neg\varphi_i(\vec{q^{i,k}})$$

for $k = r, r-1, \dots, 2, 1$. This will suffice since when $k = 1$ we have an $e\mathcal{F}$-proof ending with

$$\neg\varphi_r \to \neg\varphi_1(\vec{q^{1,1}}),$$

and since $\varphi_1(\vec{q^{1,1}})$ is an axiom instance, φ_r follows in a few more steps.

We define next the extension rules for the variables $q_j^{i,k}$. This is done successively for $k = r, r-1, \dots, 1$, and whenever we define the extension rule for $q_j^{i,k}$ we suppose that we have a short $e\mathcal{F}$-proof for

$$\neg\varphi_r \to \bigvee_{i=1}^{k+1} \neg\varphi_i(\vec{q^{i,k+1}}).$$

When $k = r$, $q^{i,k}$ is defined by the rule

$$q_j^{r,r} \leftrightarrow p_j$$

$$q_j^{i,r} \leftrightarrow \bot \quad \text{for } i < r.$$

Obviously

$$\neg\varphi_r \to \neg\varphi_r(\vec{q^{r,r}}).$$

For $k < r$ we have three cases depending on how φ_{k+1} has been obtained.

Case 1: If φ_{k+1} is an axiom then let

$$q_j^{i,k} \leftrightarrow q_j^{i,k+1} \quad \text{for all } i, j.$$

$\varphi_{k+1}(\vec{q^{k+1,k+1}})$ has an $e\mathcal{F}$-proof of one line, and the hypothesis

$$\neg\varphi_r \to \bigvee_{i=1}^{k+1} \neg\varphi_i(\vec{q^{i,k+1}})$$

immediately implies

$$\neg\varphi_r \to \bigvee_{i=1}^{k} \neg\varphi_i(\vec{q^{i,k}}).$$

Case 2: If φ_{k+1} is inferred by substitution

$$\frac{\varphi_l(p_1,\ldots,p_m)}{\varphi_{k+1}(\psi_1,\ldots,\psi_m)}$$

then define $q_j^{i,k}$ as follows: For $i \neq l$

$$q_j^{i,k} \leftrightarrow q_j^{i,k+1}$$

and for $i = l$,

$$q_j^{i,k} \leftrightarrow (\neg\varphi_l(\vec{q^{l,k+1}}) \ \wedge \ q_j^{l,k+1}) \ \vee \ (\varphi_l(\vec{q^{l,k+1}}) \ \wedge \ \psi_j(\vec{q^{k+1,k+1}})).$$

Put another way, the definition of $q_j^{i,k}$ in the case $i = l$ is:

$$q_j^{i,k} \leftrightarrow \begin{cases} q_j^{l,k+1} & \text{if } \neg\varphi_l(\vec{q^{l,k+1}}) \\ \\ \psi_j(\vec{q^{k+1,k+1}}) & \text{otherwise.} \end{cases}$$

By the way extensions are defined, from the formula $\varphi_l(\vec{q^{l,k}})$ we could infer $\varphi_l(\vec{q^{l,k+1}})$ and $\varphi_l(\psi_j(\vec{q^{k+1,k+1}}))$, but this last formula equals $\varphi_{k+1}(\vec{q^{k+1,k+1}})$. This establishes the implication

$$\neg\varphi_{k+1}(\vec{q^{k+1,k+1}}) \rightarrow \neg\varphi_l(\vec{q^{l,k}}).$$

Since $l \leq k$, this combines with the induction hypothesis to yield a short $e\mathcal{F}$ proof of

$$\neg\varphi_r \rightarrow \bigvee_{i=1}^{k} \neg\varphi_i(\vec{q^{i,k}}).$$

Case 3: If φ_{k+1} is inferred by modus ponens

$$\frac{\varphi_l \quad \varphi_{l'}}{\varphi_{k+1}}$$

then for $i \neq l$ and $i \neq l'$ define

$$q_j^{i,k} \leftrightarrow q_j^{i,k+1},$$

and in case either $i = l$ or $i = l'$, define

$$q_j^{i,k} \leftrightarrow \begin{cases} q_j^{i,k+1} & \text{if } \neg\varphi_i(q^{i,\vec{k}+1}) \\ q_j^{k+1,k+1} & \text{otherwise.} \end{cases}$$

By the way extensions are defined, from $\varphi_l(q^{\vec{l},k}) \;\wedge\; \varphi_{l'}(q^{\vec{l'},k})$, we get $\varphi_l(q^{k+\vec{1},k+1}) \;\wedge\; \varphi_{l'}(q^{k+\vec{1},k+1})$, which entails $\varphi_{k+1}(q^{k+\vec{1},k+1})$. Thus we have

$$\neg\varphi_{k+1}(q^{k+\vec{1},k+1}) \rightarrow \neg\varphi_l(q^{\vec{l},k}) \;\vee\; \neg\varphi_{l'}(q^{\vec{l'},k}),$$

and since l and l' are smaller than $k+1$, this combines with the induction hypothesis to yield an $e\mathcal{F}$-proof of

$$\neg\varphi_r \rightarrow \bigvee_{i=1}^{k} \neg\varphi_i(q^{\vec{i},k}).$$

□

The above theorem shows that the substitution rule is rather powerful. The next two results show that the whole power of this rule can be achieved using two weak forms of substitution, True-False substitution and variable substitution.

Definition 6.2. The True-False substitution rules,

$$\frac{\varphi(p)}{\varphi(\top)} \quad \text{and} \quad \frac{\varphi(p)}{\varphi(\bot)}$$

allow the substitution of the nullary constants $\top$ and $\bot$ (true or false) for the variable p in the formula φ.

Theorem 6.2 ([Bus95]) *A Frege system augmented with T-F-substitution can p-simulate extended Frege systems.*

Proof. The idea is that for any formula φ, and any variable p in φ, T-F-substitution can simulate in a polynomial number of steps any substitution of a formula ψ for p. This can be done obtaining the formulas $\varphi(\top)$ and $\varphi(\bot)$

(using two T-F-substitutions), and deriving the valid formulas $\psi \wedge \varphi(\top) \rightarrow \varphi(\psi)$ and $\neg\psi \wedge \varphi(\bot) \rightarrow \varphi(\psi)$. These can be derived using a number of lines that is linearly bounded in the number of connectives in $\varphi(p)$. From these formulas $\varphi(\psi)$ can be easily inferred in a constant number of lines. $\square$

Another type of substitution that seems weak but is as powerful as the general substitution is the renaming rule. This rule allows to rename and identify different variables.

Definition 6.3. The renaming rule is

$$\frac{\varphi(p)}{\varphi(q)}$$

(p and q are propositional variables).

Theorem 6.3 ([Bus95]) *A Frege system augmented with the renaming rule can p-simulate extended Frege systems.*

Proof. By the previous theorem it suffices to show that a Frege system with renaming can simulate Frege proofs with T-F-substitutions. Let A be a formula and P be a T-F-substitution Frege proof of A, and let $p_1, \ldots, p_k$ be the variables that appear in P. One can first prove without renaming the formulas $(p_1 \wedge \ldots \wedge p_k) \rightarrow A(\vec{p})$ and $(\neg p_1 \wedge \ldots \wedge \neg p_k) \rightarrow A(\vec{p})$. This can be done proving first the formula $A(\top, \ldots, \top)$ (for the first case) that does not have variables and therefore a proof only needs to prove its true subformulas and disprove the false ones. Then one can prove the formula $(p_1 \wedge \ldots \wedge p_k) \wedge A(\top, \ldots, \top) \rightarrow A(\vec{p})$. From these two, $(p_1 \wedge \ldots \wedge p_k) \rightarrow A(\vec{p})$ can be inferred in a constant number of steps.

Let D be the formula

$$\neg(p_1 \wedge \ldots \wedge p_k) \wedge (p_1 \vee \ldots \vee p_k).$$

We show that there is a Frege proof with renaming of the formula $D \rightarrow A$, (from this and the proofs of the formulas considered before, we can conclude that there is a proof for A). To do this we construct a new proof P' to simulate the proof P by proving $D \rightarrow B$ for each formula B in P. If B is inferred in P by a Frege inference, then $D \rightarrow B$ can easily be inferred from previous lines in P'. The case left is when B comes from a T-F-substitution like

$$\frac{B(p_i)}{B(\top)}.$$

By hypothesis we have in P' $D \to B(p_i)$ and we have to infer $D \to B(\top)$. Doing $k-1$ renaming inferences we can get $D(p_i/p_j) \to B(p_j)$ for all $j \neq i$. ($D(p_i/p_j)$ represents the replacement of p_i with p_j in D). Then one can get proofs for the formulas $p_j \wedge B(p_j) \to B(\top)$. Combining these one can infer $D_i \to B(\top)$, where D_i is the formula

$$\neg(p_1 \wedge \ldots \wedge p_{i-1} \wedge p_{i+1} \wedge \ldots \wedge p_k) \wedge (p_1 \vee \ldots \vee p_{i-1} \vee p_{i+1} \vee \ldots \vee p_k).$$

(Observe that $D(p_i/p_j)$ is equivalent to D_i). Now one can easily infer

$$D \wedge \neg D_i \to B(T)$$

since the hypothesis of this formula holds only for two possible values of the variables. With this and $D_i \to B(\top)$, $D \to B(\top)$ can be inferred in a constant number of lines. $\square$

6.1 Tree-like versus non-tree-like proofs

A tree-like proof is one in which intermediate results (formulas) are used as hypotheses for an inference only once. As the next theorem shows, in the case of Frege systems, non-tree-like proofs can be transformed into tree-like without changing the size very much.

Theorem 6.4 ([Kra94]) *If A has a (non-tree-like) $\mathcal{F}$-proof on size n then A has a tree-like $\mathcal{F}$-proof of size $p(n)$ for some polynomial p.*

Proof Sketch. If the $\mathcal{F}$-proof has lines $B_1, \ldots, B_k = A$, the idea is to form a tree-like $\mathcal{F}$-proof which proves B_1, then $B_1 \wedge B_2$, then $B_1 \wedge B_2 \wedge B_3, \ldots$, until $B_1 \wedge B_2 \wedge \ldots \wedge B_k$ is proved.

Bonet in [Bon91] improved Krajíček's construction by giving better bounds on the size of the tree-like proof.

6.2 Best known lower bounds for (extended) Frege proofs

Theorem 6.5 *(a) Let $\mathcal{F}$ be an arbitrary Frege system. There are tautologies that require quadratic size $\mathcal{F}$ proofs.*

(b) Same holds for $e\mathcal{F}$ proof systems.

These results follow from the following theorem.

Theorem 6.6 ([Bus95]) *Let A have n connectives and let m be the sum of the sizes of the subformulas of A without repetition, and suppose also that A is not an instance of a shorter tautology. Then*

(a) Any extended Frege proof of A has $\Omega(m)$ symbols.

(b) Any Frege proof has $\Omega(n)$ lines.

Proof Sketch. Consider the formula A

$$\bot \vee (\bot \vee (\bot \vee (\ldots \vee (\bot \vee \top))\ldots))$$

with n disjunctions. Each subformula of A must be "active" in any proof, where "active" means its principal connective is used by some inference. Otherwise, the subformula could be replaced everywhere by $\bot$, and we would still have a valid proof, but of a non-tautology. Each inference makes only a constant number of subformulas active, so proof size must be

$$\Omega(\sum \|\{\, B : B \text{ is a subformula of } A \,\}\|) = \Omega(m).$$

$\square$

Although the formula A in the above proof is an easy one, in fact these are the best known lower bounds on Frege proof size. These results constrast with the exponential lower bounds that are known for other (weaker) proof systems, like resolution.

Theorem 6.7 ([Hak85]) *There is some constant $c > 0$ such that any resolution proof of the pigeonhole principle PHP_n requires at least 2^{cn} lines.*

Since as we have seen, PHP_n has polynomial size Frege proofs we get

Corollary 6.8 *Resolution does not p-simulate Frege proof systems.*

Exponential lower bound are known for Frege proof systems of constant depth. In a language $\{ \wedge , \vee , \neg \}$; the depth of a formula is the number of alternations of $\wedge$'s and $\vee$ s (after $\neg$ has been pushed to the literals). A constant depth Frege proof is one in which the formula depth is bounded by a constant. The following was independently obtained by Krajíček, Pudlák, and Woods, and by Pitassi, Beame, and Impagliazzo.

Theorem 6.9 ([KPW91, PBI93]) *Depth d Frege proofs of PHP_n require $\Omega(2^{n^{\frac{1}{6^d}}})$ symbols.*

The next result shows that a Frege proof for a formula can be transformed into a new one where the symbol-size is related to line-size and the depth of the original proof.

Theorem 6.10 ([Bus95]) *A depth d Frege proof with m lines can be transformed into a depth d Frege proof with $O(m^d)$ symbols.*

Corollary 6.11 *PHP_n requires depth d Frege proofs of $\Omega(2^{n^{\frac{1}{6^d}}})$ lines.*

References

[Ajt94] M. Ajtai. The complexity of the pigeonhole principle. *Combinatorica*, 14:417–433, 1994.

[Bar89] D. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comp. Sys. Sci.*, 38:150–164, 1989.

[BB94] M.L. Bonet and S. Buss. Size-depth tradeoffs for Boolean formulae. *Inf. Proc. Lett.*, 29:151–155, 1994.

[BC88] M. Ben-Or and R. Cleve. Computing algebraic formulas using a constant number of registers. In *Proc. 20th STOC*, pages 254–257, 1988.

[BCE91] N. Bshouty, R. Cleve, and W. Eberly. Size-depth tradeoffs for algebraic formulae. In *Proc. 32nd FOCS*, pages 334–341, 1991.

[BCGR92] S. Buss, S. Cook, A. Gupta, and V. Ramachandran. An optimal parallel algorithm for formula evaluation. *SIAM J. Comput.*, 21:755–780, 1992.

[BCP83] A. Borodin, S. Cook, and N. Pippenger. Parallel computation for well-endowed rings, and space-bounded probabilistic machines. *Inform. and Control*, 58:113–136, 1983.

[Bon91] M. Bonet. *The lengths of propositional proofs and the deduction rule*. PhD thesis, Univ. of California at Berkeley, 1991.

[Bus87a] S. Buss. The boolean formula value problem is in ALOGTIME. In *Proc. 19th STOC*, pages 123–131, 1987.

[Bus87b] S. Buss. Polynomial-size proofs of the propositional pigeonhole principle. *J. Symb. Logic*, 52:916–927, 1987.

[Bus93] S. Buss. Algorithms for Boolean formula evaluation and for tree contraction. In P. Clote and J. Krajíček, editors, *Arithmetic, Proof Theory, and Computational Complexity*, pages 96–115. Oxford University Press, 1993.

[Bus95] S. Buss. Some remarks on lengths of propositional proofs, 1995. Submitted to *Archiv für Mathematische Logik*.

[CCT87] W. Cook, C. Coullard, and G. Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18:25–38, 1987.

[CL94] J.-Y. Cai and R. Lipton. Subquadratic simulations of balanced formulae by branching programs. *SIAM J. Comput.*, 23:563–572, 1994.

[Coo71] S. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd STOC*, pages 151–158, 1971.

[CR79] S. Cook and R. Reckhow. The relative efficiency of propositional proof systems. *J. Symb. Logic*, 44:36–50, 1979.

[Dow85] M. Dowd. Model-theoretic aspects of P $\neq$ NP. Typewritten manuscript, 1985.

[Hak85] A. Haken. The intractability of resolution. *Theor. Comp. Sci.*, 39:297–308, 1985.

[HS65] J. Hartmanis and R. Stearns. On the computational complexity of algorithms. *Transactions of the AMS*, 117:285–306, 1965.

[HU79] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison–Wesley, Reading, MA, 1979.

[Imm91] N. Immerman. DSPACE$[n^k]$ = VAR$[k+1]$. In *Proc. 6th Structures*, pages 334–340, 1991.

[KD90] S.R. Kosaraju and A. Delcher. A tree-partitioning technique with applications to expression evaluation and term matching. In *Proc. 31st FOCS*, pages 163–172, 1990.

[KD92] S.R. Kosaraju and A. Delcher. A tree-partitioning technique with applications to expression evaluation and term matching, November 1992.

[KP89] J. Krajíček and P. Pudlák. Propositional proof systems, the consistency of first-order theories, and the complexity of computations. *J. Symb. Logic*, 54:1063–1079, 1989.

[KP90] J. Krajíček and P. Pudlák. Quantified propositional logic and fragments of bounded arithmetic. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 36:29–46, 1990.

[KPW91] J. Krajíček, P. Pudlák, and A. Wood. Exponential lower bound to the size of bounded depth Frege proofs of the pigeonhole principle, 1991. Typeset manuscript, to appear in the journal *Random Structures and Algorithms*.

[Kra94] J. Krajíček. Lower bounds to the size of constant-depth Frege proofs. *J. Symb. Logic*, 59:587–598, 1994.

[Lad75] R. Ladner. The circuit value problem is log-space complete for P. *SIGACT News*, 7:18–20, 1975.

[Ofm63] Y. Ofman. On the algorithmic complexity of discrete functions. *Soviet Physics—Doklady*, 7(7):589–591, 1963. English translation.

[PBI93] T. Pitassi, P. Beame, and R. Impagliazzo. Exponential lower bounds for the pigeonhole principle. *Computational Complexity*, 3:97–140, 1993.

[Rec76] R. Reckhow. *On the lengths of proofs in the propositional calculus.* PhD thesis, University of Toronto, 1976. Computer Science TR #87.

[Spi71] M. Spira. On time-hardware complexity tradeoffs for Boolean functions. In *Proceedings of the Fourth International Symposium on Systems Sciences*, pages 525–527, 1971.

[Sta77] R. Statman. Complexity of derivations from quantifier-free Horn formulae, mechanical introduction of explicit definitions, and refinement of completeness theorems. In *Proc. Logic Colloquium '76*, pages 505–517. North-Holland, 1977.

[Tse68] G. Tseitin. On the complexity of derivations in propositional calculus. In A. Slisenko, editor, *Studies in Mathematics and Mathematical Logic, Part II*, pages 115–125. 1968. English translation.

[Wal64] C. Wallace. A suggestion for a fast multiplier, 1964.

Interpolation Theorems for Propositional Logic

Notes by Eric Allender and Alexis Maciel

Monday Evening, March 6

1 Introduction

Let $\vec{p}$, $\vec{q}$ and $\vec{r}$ be vectors of variables and let $A(\vec{p},\vec{q})$ and $B(\vec{p},\vec{r})$ be two propositional formulas involving only the indicated variables. Suppose that $A(\vec{p},\vec{q})$ implies $B(\vec{p},\vec{r})$. Since B does not involve any variable from $\vec{q}$, whatever A says about $\vec{p}$ should be sufficient to imply B. This intuition is formalized by the Interpolation Theorem.

Theorem 1.1 *Let $A(\vec{p},\vec{q})$ and $B(\vec{p},\vec{r})$ be propositional formulas involving only the indicated variables. Also, suppose that $A(\vec{p},\vec{q}) \to B(\vec{p},\vec{r})$ is a tautology. Then there is a propositional formula $C(\vec{p})$ not involving q's and r's such that*

$$A(\vec{p},\vec{q}) \to C(\vec{p}) \quad \text{and} \quad C(\vec{p}) \to B(\vec{p},\vec{r})$$

are tautologies.

Proof Let $\tau_1,\dots,\tau_n$ be the truth assignments to $p_1,\dots,p_k$ for which it is possible to make $A(\vec{p},\vec{q})$ true by further assignment of truth values to $\vec{q}$. Let $C(\vec{p})$ say that one of $\tau_1,\dots,\tau_n$ holds for $\vec{p}$, i.e.,

$$C(\vec{p}) = \bigvee_{i=1}^{n} (p_1^{(i)} \wedge p_2^{(i)} \wedge \cdots \wedge p_k^{(i)})$$

where

$$p_j^{(i)} = \begin{cases} p_j & \text{if } \tau_i(p_j) = \text{True} \\ \neg p_j & \text{if } \tau_i(p_j) = \text{False} \end{cases}$$

Then, clearly, $A(\vec{p}, \vec{q}) \models C(\vec{p})$.

On the other hand, a truth assignment to $\vec{p}$ that satisfies $C(\vec{p})$ can be extended to a truth assignment to $\vec{p}, \vec{q}$ that satisfies $A(\vec{p}, \vec{q})$. Since $A(\vec{p}, \vec{q}) \models B(\vec{p}, \vec{r})$, every extension of this truth assignment to $\vec{p}, \vec{q}, \vec{r}$ must satisfy $B(\vec{p}, \vec{r})$. Therefore, $C(\vec{p}) \models B(\vec{p}, \vec{r})$. $\square$

A stronger version of the interpolation theorem was proved by Craig for first-order logic [6]. Also, note that the above proof allows $C(\vec{p})$ to be exponentially big, as a function of the sizes of $A(\vec{p}, \vec{q})$ and $B(\vec{p}, \vec{r})$.

The following example exhibits a relationship between the size of interpolants and complexity theory.

Example 1.2 Consider the language FACT of all pairs (X, U) where X is the binary representation of a number and U is the prefix of some prime V that divides X. Let $p_1, \ldots, p_k$ code a pair (X, U). Let $A(\vec{p}, \vec{q})$ be a formula that is satisfiable if and only if $(X, U) \in$ FACT. For example, $A(\vec{p}, \vec{q})$ says that (i) $\vec{q}$ codes numbers $V_1, \ldots, V_m$ and Pratt primality witnesses for each V_i, and that (ii) $V_1 \cdots V_m = X$ and U is the prefix of some V_i. Let $B(\vec{p}, \vec{r})$ be a formula that is satisfiable if and only if $(X, U) \notin$ FACT. For example, $B(\vec{p}, \vec{r})$ says that (i) $\vec{r}$ codes numbers $V_1, \ldots, V_m$ and Pratt primality witnesses for each V_i, and that (ii) $V_1 \cdots V_m = X$ and U is not the prefix of any of the V_i. Then, $(X, U) \in \text{FACT} \Leftrightarrow \exists \vec{q} A(\vec{p}, \vec{q}) \Leftrightarrow \neg \exists \vec{r} B(\vec{p}, \vec{r})$, so that

$$A(\vec{p}, \vec{q}) \rightarrow \neg B(\vec{p}, \vec{r})$$

is a tautology. Therefore, an interpolant $C(\vec{p})$ must express $(X, U) \in$ FACT, since $(X, U) \in \text{FACT} \Rightarrow \exists \vec{q} A(\vec{p}, \vec{q}) \Rightarrow C(\vec{p})$ and $C(\vec{p}) \Rightarrow \forall \vec{r} \neg B(\vec{p}, \vec{r}) \Rightarrow (X, U) \in \text{FACT}$. $\square$

As a consequence, a polynomial upper bound on the size of interpolants for propositional logic would immediately translate into a polynomial upper bound on the size of formulas or circuits for the language FACT, depending on how the size of an interpolant is defined. Note that FACT $\in$ NP $\cap$ coNP; this is easily seen by using the formulas $A(\vec{p}, \vec{q})$ and $B(\vec{p}, \vec{r})$ above. On the other hand, it is not known if FACT has polynomial-size formulas or circuits. In fact, if this language has polynomial-size circuits, then so does factoring. The conjecture would therefore be that FACT $\notin$ P/poly. Generalizing this example gives the following:

Theorem 1.3 ([9]) *If there is a polynomial upper bound on the circuit size of interpolants in propositional logic, then*

$$\mathrm{NP/poly} \cap \mathrm{coNP/poly} = \mathrm{P/poly}.$$

Proof Let $\exists\vec{q}A(\vec{p},\vec{q})$ express an NP/poly property $R(\vec{p})$ and let $\forall\vec{r}B(\vec{p},\vec{r})$ express the same property in coNP/poly form. Then

$$\exists\vec{q}A(\vec{p},\vec{q}) \Rightarrow \forall\vec{r}B(\vec{p},\vec{r}),$$

which is equivalent to

$$A(\vec{p},\vec{q}) \rightarrow B(\vec{p},\vec{r})$$

being a tautology. Let $C(\vec{p})$ be a polynomial circuit size interpolant such that

$$A(\vec{p},\vec{q}) \rightarrow C(\vec{p}) \ \text{ and } \ C(\vec{p}) \rightarrow B(\vec{p},\vec{r})$$

are tautologies. Thus

$$\exists\vec{q}A(\vec{p},\vec{q}) \Rightarrow C(\vec{p}) \ \text{ and } \ C(\vec{p}) \Rightarrow \forall\vec{r}B(\vec{p},\vec{r}),$$

i.e., $R(\vec{p}) \Leftrightarrow C(\vec{p})$. Therefore, $R(\vec{p})$ has a polynomial-size circuit and thus is in P/poly. □

In the remainder of this lecture, we prove upper bounds on the size of interpolants in two restricted cases: when $A(\vec{p},\vec{q}) \rightarrow B(\vec{p},\vec{r})$ has a short cut-free proof in the propositional sequent calculus, and when $A(\vec{p},\vec{q}) \rightarrow B(\vec{p},\vec{r})$ has a short resolution refutation. We also prove a monotone version of the latter and one for resolution with limited extension.

2 The propositional sequent calculus

Let propositional formulas be built with the connectives $\top$, $\neg$, $\wedge$ and $\vee$ for true, negation, conjunction and disjunction. A *sequent* is of the form

$$A_1, A_2, \ldots, A_k \rightarrow B_1, B_2, \ldots, B_l$$

where $k \geq 0$, $l \geq 0$ and the A_i's and B_j's are formulas. The intended meaning of the sequent is

$$(A_1 \wedge A_2 \wedge \cdots \wedge A_k) \rightarrow (B_1 \vee B_2 \vee \cdots \vee B_l).$$

(An empty conjunction has value true; an empty disjunction has value false.)

The propositional sequent calculus PK is a proof system in which each line in a proof is a sequent. The *axioms* or *initial sequents* are

(1) $p \to p$, p any propositional variable

(2) $\to \top$

The rules of inference of PK are as follows. Γ and Δ represent formulas separated by commas (*cedents*).

(1) *Weak structural rules:*

$$\frac{\Gamma \to \Delta}{\Gamma' \to \Delta'}$$

provided every formula in Γ and Δ appears also in Γ' and Δ', respectively.

(2)

$$\wedge\text{:left}\ \frac{A, B, \Gamma \to \Delta}{A \wedge B, \Gamma \to \Delta} \qquad \wedge\ \text{:right}\ \frac{\Gamma \to \Delta, A \qquad \Gamma \to \Delta, B}{\Gamma \to \Delta, A \wedge B}$$

(3)

$$\vee\text{:left}\ \frac{A, \Gamma \to \Delta \qquad B, \Gamma \to \Delta}{A \vee B, \Gamma \to \Delta} \qquad \vee\ \text{:right}\ \frac{\Gamma \to \Delta, A, B}{\Gamma \to \Delta, A \vee B}$$

(4)

$$\neg\text{:left}\ \frac{\Gamma \to \Delta, A}{\neg A, \Gamma \to \Delta} \qquad \neg\text{:right}\ \frac{A, \Gamma \to \Delta}{\Gamma \to \Delta, \neg A}$$

(5) *Cut rule:*

$$\frac{\Gamma \to \Delta, A \qquad A, \Gamma \to \Delta}{\Gamma \to \Delta}$$

PK is sound: if PK $\vdash \Gamma \to \Delta$, then $\Gamma \to \Delta$ is valid. PK is also complete: if $\Gamma \to \Delta$ is valid, then there is a PK proof of $\Gamma \to \Delta$. In fact, PK is cut-free complete: if $\Gamma \to \Delta$ is valid, then there is a cut-free PK proof of $\Gamma \to \Delta$. Obviously, cut-free completeness implies completeness. The cut-free completeness of PK can be proved by induction on the number of propositional connectives occurring in $\Gamma \to \Delta$. (See [4].)

Note that $\wedge$:left and $\vee$:right are dual. The same is true of $\vee$:left and $\wedge$:right. In $\Gamma \to \Delta$, Γ is called the *antecedent*, and Δ is called the *succedent*. Let $V(A)$ denote the set of variables occurring in A. Let $|A|$ denote the number of symbols in formula A, and let $|A|_{\mathrm{dag}}$ denote that number of symbols in the circuit form of A. Let $|P|$ and $|P|_{\mathrm{dag}}$ denote the number of sequents in the tree form and in the circuit form of a proof P.

3 Interpolation for cut-free proofs

Theorem 3.1 (Buss) *Let P be a cut-free PK proof of $A \to B$ where $V(A) \subseteq \{\vec{p}, \vec{q}\}$ and $V(B) \subseteq \{\vec{p}, \vec{r}\}$. Then, there is an interpolant C such that*

(i) $A \to C$ *and* $C \to B$ *are valid,*

(ii) $V(C) \subseteq \{\vec{p}\}$,

(iii) $|C| \leq 2|P|$ *and* $|C|_{\mathrm{dag}} \leq 2|P|_{\mathrm{dag}}$.

Therefore, tree-like cut-free proofs have interpolants of polynomial formula size and general cut-free proofs have interpolants of polynomial circuit size.

Note that the proof below will not only show that $A \to C$ and $C \to B$ are valid, but that $A \to C$ and $C \to B$ have short cut-free PK proofs. In addition, the theorem also holds for proofs P that have cuts only on formulas D such that $V(D) \subseteq \{\vec{p}, \vec{q}\}$ or $V(D) \subseteq \{\vec{p}, \vec{r}\}$. On the other hand, it is not known if similar bounds hold on the size of interpolants for general PK proofs.

Proof We will prove a slightly more general statement:

If P is a cut-free PK proof of $\Gamma \to \Delta$, if Γ is $\Gamma_1 \cup \Gamma_2$ and Δ is $\Delta_1 \cup \Delta_2$ (possibly reordered), if $V(\Gamma_1, \Delta_1) \subseteq \{\vec{p}, \vec{q}\}$ and $V(\Gamma_2, \Delta_2) \subseteq \{\vec{p}, \vec{r}\}$, then there is an interpolant C such that *(i)* $\Gamma_1 \to \Delta_1, C$ *and* $C, \Gamma_2 \to \Delta_2$ *are valid,*

(ii) $V(C) \subseteq \{\vec{p}\}$,

(iii) $|C| \leq 2|P|$ *and* $|C|_{\text{dag}} \leq 2|P|_{\text{dag}}$.

The proof will be by induction on the number of inferences in P.

Base case: No inferences. If the initial sequent is $q_i \to q_i$, then take C to be $(\neg\top)$, since $q_i \to q_i, \neg\top$ and $\neg\top \to$ are valid. If the initial sequent is $r_i \to r_i$, then C can be $\top$. If the initial sequent is $p_i \to p_i$, then C will be $\top$, $(\neg\top)$, p_i, or $(\neg p_i)$ depending on how the two instances of p_i are split into $\Gamma_1, \Gamma_2, \Delta_1, \Delta_2$. For example, if Γ_1 and Δ_2 are p_i and Δ_1 and Γ_2 are empty, then C can be p_i. If the initial sequent is $\to \top$, then C will be $\top$ or $\neg\top$.

Induction step: Consider the last inference.

Case (1): The last inference is

$$\vee\text{:right}\ \frac{\Gamma \to \Delta, A, B}{\Gamma \to \Delta, A \vee B}$$

In this case, the interpolant for the upper sequent will still work for the lower sequent. By hypothesis, $A \vee B$ is either contained in Δ_1 or it is contained in Δ_2. If $A \vee B \in \Delta_1$, then an interpolant for the upper sequent is such that

$$\Gamma_1 \to \Delta_1^-, A, B, C \quad \text{and} \quad C, \Gamma_2 \to \Delta_2$$

are valid, where $\Delta_1^- = \Delta_1 - \{A \vee B\}$. If $A \vee B \in \Delta_2$, then an interpolant for the upper sequent is such that

$$\Gamma_1 \to \Delta_1, C \quad \text{and} \quad C, \Gamma_2 \to \Delta_2^-, A, B$$

are valid, where $\Delta_2^- = \Delta_2 - \{A \vee B\}$.

Case (2): The last inference is

$$\wedge\text{:right}\ \frac{\Gamma \to \Delta, A \qquad \Gamma \to \Delta, B}{\Gamma \to \Delta, A \wedge B}$$

If $A \wedge B \in \Delta_1$, apply the induction hypothesis twice to get interpolants C_A and C_B such that

$$\Gamma_1 \to \Delta_1^-, A, C_A \qquad\qquad C_A, \Gamma_2 \to \Delta_2$$

$$\Gamma_1 \to \Delta_1^-, B, C_B \qquad\qquad C_B, \Gamma_2 \to \Delta_2$$

are valid. Now,

$$\frac{\Gamma_1 \to \Delta_1^-, A, C_A \vee C_B \qquad \Gamma_1 \to \Delta_1^-, B, C_A \vee C_B}{\Gamma_1 \to \Delta_1^-, A \wedge B, C_A \vee C_B}$$

and

$$\frac{C_A, \Gamma_2 \to \Delta_2 \qquad C_B, \Gamma_2 \to \Delta_2}{C_A \vee C_B, \Gamma_2 \to \Delta_2}$$

Therefore, $(C_A \vee C_B)$ is an interpolant.

If $A \wedge B \in \Delta_2$, apply the induction hypothesis twice to get interpolants C_A and C_B such that

$$\Gamma_1 \to \Delta_1, C_A \qquad\qquad C_A, \Gamma_2 \to \Delta_2^-, A$$

$$\Gamma_1 \to \Delta_1, C_B \qquad\qquad C_B, \Gamma_2 \to \Delta_2^-, B$$

are valid. Similarly to before,

$$\frac{C_A \wedge C_B, \Gamma_2 \to \Delta_2^-, A \qquad C_A \wedge C_B, \Gamma_2 \to \Delta_2^-, B}{C_A \wedge C_B, \Gamma_2 \to \Delta_2^-, A \wedge B}$$

and

$$\frac{\Gamma_1 \to \Delta_1, C_A \qquad \Gamma_1 \to \Delta_1, C_B}{\Gamma_1 \to \Delta_1, C_A \wedge C_B}$$

Therefore, $(C_A \wedge C_B)$ is an interpolant.

All other cases are handled similarly. In addition, the size bounds on C are easily verified. □

4 Resolution

In this section we review the basic notions and definitions concerning resolution-based theorem proving.

Definition 4.1. A *literal* is either a propositional variable p or a negated variable $\neg p$ (which will also be denoted $\overline{p}$).

Definition 4.2. A formula in *CNF - Conjunctive Normal Form* is expressed as a set of *clauses* $\{C_1, \ldots, C_k\}$, where each C_i is a set of literals $\{l_1, \ldots, l_j\}$ interpreted as the disjunction of those literals, $l_1 \vee \ldots \vee l_j$ and the formula is interpreted as the conjunction of the clauses. We will assume throughout that no clause contains both p and $\overline{p}$.

Definition 4.3. A *resolution derivation* from $C_1, \ldots, C_k$ is a sequence $D_1, \ldots, D_l$ such that each D_i is either

1. one of the C_i's or

2. of the form $D_i = D_{j_1} \cup D_{j_2} \setminus \{x_r, \overline{x_r}\}$ where D_{j_1} and D_{j_2} have $x_r, \overline{x_r}$ as their only pair of complementary literals, and both j_1 and j_2 are less than i. (D_i is said to be the result of *resolving* D_{j_1} and D_{j_2}.)

Definition 4.4. A *refutation* is a derivation whose last element is $\emptyset$ (denoted by the symbol $\Box$).
This empty clause has the meaning of False; the only way it can be obtained in a derivation is for $C_{j_1} = \{x\}, C_{j_2} = \{\overline{x}\}$ for some literal x and clearly $x \wedge \overline{x}$ cannot be True.

If φ is a formula in DNF, then we may abuse notation slightly and speak of $\neg\varphi$ as being in CNF (whereas it would be more precise to consider the formula in CNF that results by applying DeMorgan's laws to $\neg\varphi$).

Although this form of proof is limited, it is in fact widely used and is therefore of interest. The following well-known result shows that, at least for proving DNF tautologies, resolution is a complete proof system.

Theorem 4.1 *Let φ be in DNF. Then $\models \varphi$ (i.e. φ is a tautology) iff there is a resolution refutation of $\neg\varphi$.*

Note that every resolution refutation has an associated graph, with nodes labeled by clauses, and edges from C to D if clause D is the result of resolving C with another clause. A resolution refutation is said to be *tree-like* if this graph is a tree. Since any refutation can be made tree-like by merely repeating parts of the derivation if need be, tree-like resolution is also a complete proof system for proving tautologies in DNF.

5 Interpolation for resolution

In this section, we show that a theorem analogous to Theorem 3.1 holds also for resolution refutations. If we look at Theorem 1.1 and consider the special case where $A(\vec{p},\vec{q})$ is in CNF, and $B(\vec{p},\vec{r})$ is in DNF, then saying that $A(\vec{p},\vec{q}) \to B(\vec{p},\vec{r})$ is a tautology is equivalent to saying that a set of clauses of the form $\{A_i(\vec{p},\vec{q})\} \cup \{B_j(\vec{p},\vec{r})\}$ is unsatisfiable. Thus the interpolant guaranteed by Theorem 1.1 has the form mentioned in the following theorem.

Theorem 5.1 *[10, 8] Let $\Gamma = \{A_i(\vec{p},\vec{q})\} \cup \{B_j(\vec{p},\vec{r})\}$ have a resolution refutation of length n. Then there is an interpolant $C(\vec{p})$ such that*

$$\bigwedge_i A_i(\vec{p},\vec{q}) \Rightarrow C(\vec{p})$$

and

$$C(\vec{p}) \Rightarrow \neg \bigwedge_j (B_j(\vec{p},\vec{r}))$$

and $C(\vec{p})$ has circuits of size $\leq 3n$.

If the refutation is tree-like, then $C(\vec{p})$ will have formulae of size $O(n)$.

Proof

For each expression E in our resolution refutation, we will have a gate C_E in our circuit. (The circuit $C_\Box$ for the final clause in the refutation will be the circuit $C(\vec{p})$.)

1. If $E = A_i(\vec{p},\vec{q})$, then C_E is the constant 0.

2. If $E = B_j(\vec{p},\vec{r})$, then C_E is the constant 1.

3. If $E = F \cup G$ from $(F \cup \{p_i\}, G \cup \{\overline{p_i}\})$, then

$$C_E ::= (\overline{p_i} \wedge C_{F\cup\{p_i\}}) \vee (p_i \wedge C_{G\cup\{\overline{p_i}\}})$$

4. If $E = F \cup G$ from $(F \cup \{q_i\}, G \cup \{\overline{q_i}\})$, then

$$C_E ::= C_{F\cup\{q_i\}} \vee C_{G\cup\{\overline{q_i}\}}$$

5. If $E = F \cup G$ from $(F \cup \{r_i\}, G \cup \{\overline{r_i}\})$, then

$$C_E ::= C_{F\cup\{r_i\}} \wedge C_{G\cup\{\overline{r_i}\}}$$

(Note, circuit $C_\Box$ has inputs only for 0, 1, p_i, $\overline{p_i}$, as required.)

In order to finish the proof of the theorem, we must prove the following lemma. (This is clearly sufficient to prove the theorem, since the output gate of this circuit of $C_\Box$, and $\tau(\Box) = \bot$ for *every* truth assignment τ. Note that a truth assignment τ is the same as an input assignment to the circuit $C_\Box$.)

Lemma 5.2 *If τ is a truth assignment such that $\tau(E) = \bot$, then*

$$\tau(C_E) = \bot \Longrightarrow \exists i\ \tau(A_i) = \bot$$

$$\tau(C_E) = \top \Longrightarrow \exists i\ \tau(B_i) = \bot$$

(Note C_E does *not* compute expression E in general.)

Proof (of the lemma)

1. If $E = A_i(\vec{p}, \vec{q})$, then $\tau(C_E) = \bot$. The hypothesis of the lemma is that $\tau(A_i(\vec{p}, \vec{q}) = \bot$, so the claim holds trivially in this case.
2. $E = B_j(\vec{p}, \vec{r})$, then $\tau(C_E) = \top$ and $\tau(B_j(\vec{p}, \vec{r})) = \bot$, similar to case 1.
3. $E = F \cup G$ from $(F \cup \{p_i\}, G \cup \{\overline{p_i}\})$

$$C_E ::= (\overline{p_i} \wedge C_{F\cup\{p_i\}}) \vee (p_i \wedge C_{G\cup\{\overline{p_i}\}})$$

Since $\tau(E) = \bot$, we have that $\tau(F) = \tau(G) = \bot$

If $\tau(C_E) = \bot$, then

case a: $\tau(p_i) = \top$, then $\tau(C_{G\cup\{\overline{p_i}\}}) = \bot$ and $\tau(G \cup \{\overline{p_i}\}) = \bot$. By induction hypothesis, $\exists i\ \tau(A_i) = \bot$.

case b: $\tau(\overline{p_i}) = \top$, then $\tau(C_{F\cup\{p_i\}}) = \bot$ and $\tau(F \cup \{p_i\}) = \bot$. By induction hypothesis, $\exists i\ \tau(A_i) = \bot$.

If $\tau(C_E) = \top$, then

case a: $\tau(p_i) = \top$, then $\tau(C_{G\cup\{\overline{p_i}\}}) = \top$ and $\tau(G \cup \{\overline{p_i}\}) = \bot$. By induction hypothesis, $\exists i\ \tau(B_i) = \bot$.

case b: $\tau(\overline{p_i}) = \top$, then $\tau(C_{F\cup\{p_i\}}) = \top$ and $\tau(F \cup \{p_i\}) = \bot$. By induction hypothesis, $\exists i\ \tau(B_i) = \bot$.

4. $E = F \cup G$ from $(F \cup \{q_i\}, G \cup \{\overline{q_i}\})$

$$C_E ::= C_{F\cup\{q_i\}} \vee C_{G\cup\{\overline{q_i}\}}$$

Since $\tau(E) = \perp$, we know that $\tau(F) = \tau(G) = \perp$.

If $\tau(C_E) = \perp$, then $\tau(C_{F\cup\{q_i\}}) = \tau(C_{G\cup\{\overline{q_i}\}}) = \perp$.

case a: $\tau(q_i) = \top$, then $\tau(C_{G\cup\{\overline{q_i}\}}) = \perp$ and $\tau(G \cup \{\overline{q_i}\}) = \perp$. By induction hypothesis, $\exists i\ \tau(A_i) = \perp$.

case b: $\tau(\overline{q_i}) = \top$, then $\tau(C_{F\cup\{q_i\}}) = \perp$ and $\tau(F \cup \{q_i\}) = \perp$. By induction hypothesis, $\exists i\ \tau(A_i) = \perp$.

If $\tau(C_E) = \top$, then at least one of $\tau(C_{F\cup\{q_i\}})$ and $\tau(C_{G\cup\{\overline{q_i}\}})$ is true. Also, at least one of $\tau(F \cup \{q_i\})$ and $\tau(G \cup \{\overline{q_i}\})$ is true. Note that:

If $\tau(F \cup \{q_i\}) = \perp$ and $\tau(C_{F\cup\{q_i\}}) = \top$, then by inductive hypothesis, $\exists j\ B_j(\vec{p}, \vec{r}) = \perp$.

If $\tau(G \cup \{\overline{q_i}\}) = \perp$ and $\tau(C_{G\cup\{\overline{q_i}\}}) = \top$, then by inductive hypothesis, $\exists j\ B_j(\vec{p}, \vec{r}) = \perp$.

Thus we need only worry about the cases where

case a:

$$\tau(F \cup \{q_i\}) = \perp,\ \tau(C_{F\cup\{q_i\}}) = \perp;\ \tau(G \cup \{\overline{q_i}\}) = \top, \tau(C_{G\cup\{\overline{q_i}\}}) = \top$$

case b:

$$\tau(F \cup \{q_i\}) = \top,\ \tau(C_{F\cup\{q_i\}}) = \top;\ \tau(G \cup \{\overline{q_i}\}) = \perp, \tau(C_{G\cup\{\overline{q_i}\}}) = \perp$$

But note that $C_{F\cup\{q_i\}}, C_{G\cup\{\overline{q_i}\}}$, and $B_j(\vec{p}, \vec{r})$ don't have q_i as a variable.

Thus if $\tau'(q_i) = \begin{cases} \perp & \text{if } \tau(q_i) = \top \\ \top & \text{otherwise} \end{cases}$ and $\tau = \tau'$ on all other variables,

then we have that

case a: $\tau'(C_{F\cup\{q_i\}}) = \perp$ (since $\tau(F) = \tau'(F) = \perp$ and $\tau(F \cup \{q_i\}) = \top$ and $\tau(F \cup \{q_i\}) = \top$. By induction hypothesis, $\exists j\ \tau'(B_j) = \perp$, and $\tau'(B_j) = \tau(B_j)$.

case b: Similar to case a.

5. $E = F \cup G$ from $(F \cup \{r_i\}, G \cup \{\overline{r_i}\})$

$$C_E ::= C_{F\cup\{r_i\}} \wedge C_{G\cup\{\overline{r_i}\}}$$

This is similar to case 4.

This completes the proof of the lemma and of the theorem.

$\square$

6 Monotone circuits from resolution refutations

A modification of the proof of the preceding section yields *monotone* circuits for the interpolant, for a particular class of clauses being refuted.

Theorem 6.1 *[10, 8]*
Let $\Gamma = \{A_i(\vec{p}, \vec{q})\} \cup \{B_j(\vec{p}, \vec{r})\}$ have a refutation of length n, where **either** *the $\vec{p}$ variables occur only positively in the A_i's* **or** *they occur only negatively in the B_j's. Then there is a monotone circuit $C(\vec{p})$ of size $O(n)$ such that for every τ,*

$$\tau(C(\vec{p})) = \bot \Longrightarrow \exists i \ \tau(A_i(\vec{p}, \vec{q})) = \bot$$
$$\tau(C(\vec{p})) = \top \Longrightarrow \exists j \ \tau(B_j(\vec{p}, \vec{r})) = \bot$$

Note: in order to get a monotone circuit, we need to assume that the p_l's either appear only positively in A_i or only negatively in B_j.

Proof We present the proof only in the case when the p_l's occur only negatively in the B_j's; the other case is similar.

We will build a circuit C_E for each expression E in the resolution refutation.

1. $E = A_i(\vec{p}, \vec{q})$, then C_E is the constant 0.

2. $E = B_j(\vec{p}, \vec{r})$, then C_E is the constant 1.

3. $E = F \cup G$ from $(F \cup \{p_i\}, G \cup \{\overline{p_i}\}$[1]

$$C_E ::= C_{F\cup\{p_i\}} \vee (p_i \wedge C_{G\cup\{\overline{p_i}\}})$$

4. $E = F \cup G$ from $(F \cup \{q_i\}, G \cup \{\overline{q_i}\})$

$$C_E ::= C_{F\cup\{q_i\}} \vee C_{G\cup\{\overline{q_i}\}}$$

5. $E = F \cup G$ from $(F \cup \{r_i\}, G \cup \{\overline{r_i}\})$

$$C_E ::= C_{F\cup\{r_i\}} \wedge C_{G\cup\{\overline{r_i}\}}$$

Clearly, $C_\Box$ is a monotone circuit of size $O(n)$. As in the proof of the preceding theorem, we base our proof of correctness on a lemma that we prove by induction. The statement of the lemma for this monotone construction is more complicated than the statement of the corresponding lemma in the preceding result.

For any clause E appearing in the refutation, define two "sub-clauses" E^A and E^B as follows. E^A is the disjunction of all the literals occurring in E that involve q-variables and p-variables; that is, all of the literals involving r-variables are "erased". E^B is the disjunction of all of the literals occurring in E that involve r-variables and the *negative* p-literals; that is, all q-variables and all non-negated p-variables are "erased". Note that E^A and E^B are not necessarily disjoint.[2]

Note that the following lemma is clearly sufficient to prove the theorem, since $\Box^A = \Box^B = \Box$, and the output gate of the circuit is $C_\Box$.

Lemma 6.2

$$\tau(E^A) = \bot \textit{ and } \tau(C_E) = \bot \Longrightarrow \exists i\ \tau(A_i) = \bot$$

$$\tau(E^B) = \bot \textit{ and } \tau(C_E) = \top \Longrightarrow \exists i\ \tau(B_i) = \bot$$

[1]To prove the theorem when the p_i's occur only positively in the A_j's, define C_E to be $(p_i \vee C_{F\cup\{p_i\}}) \wedge C_{G\cup\{\overline{p_i}\}}$ in this case.

[2]To prove the theorem when the p_i's occur only positively in the A_j's, define E^A to be the result of erasing the r-literals and the *positive* p-literals from E, and define E^B to the result of erasing the q-literals from E. The rest of the argument is similar.

Proof (of the lemma)

By induction on the structure of, where E appears in the resolution refutation.

1. If $E = A_i(\vec{p}, \vec{q})$, then for all $\tau, \tau(C_E) = \perp$, so only the first implication in the Lemma needs to be considered. Note also that $E^A = E$, and thus if the hypothesis for the first implication holds, then trivially $\tau(A_i) = \perp$.

2. $E = B_j(\vec{p}, \vec{r})$, then $\tau(C_E) = \top$ and $E = E^B$. Thus this is similar to the previous case.

3. $E = F \cup G$ from $(F \cup \{p_i\}, G \cup \{\overline{p_i}\})$

$$C_E ::= (C_{F\cup\{p_i\}}) \vee (p_i \wedge C_{G\cup\{\overline{p_i}\}})$$

Case a: $\tau(E^A) = \perp$ and $\tau(C_E) = \perp$. We consider two cases, depending on $\tau(p_i)$.

If $\tau(p_i) = \top$, then $\tau((G \cup \{\overline{p_i}\})^A) = \tau(G^A \cup \{\overline{p_i}\}) = \perp$. Also, $\tau(C_{G\cup\{\overline{p_i}\}}) = \perp$. Thus, by induction, there is some j such that $\tau(A_j) = \perp$.

If $\tau(p_i) = \perp$, then $\tau((F \cup \{p_i\})^A) = \tau(F^A \cup \{p_i\}) = \perp$. Also, $\tau(C_{F\cup\{p_i\}}) = \perp$. Again, the claim follows by induction.

Case b: $\tau(E^B) = \perp$ and $\tau(C_E) = \top$.

In this case, note that $(F \cup \{p_i\})^B = F^B$ and thus $\tau(F \cup \{p_i\})^B = \perp$. Since $C_E = \top$, there are the following two cases.

If $\tau(C_{F\cup\{p_i\}}) = \top$, then the induction hypothesis implies that for some $j, \tau(B_j) = \perp$.

Otherwise, $\tau(p_i \wedge C_{G\cup\{\overline{p_i}\}}) = \top$. Thus $\tau((G \cup \{\overline{p_i}\})^B) = \perp$. Again, the induction hypothesis yields the desired result.

4. $E = F \cup G$ from $(F \cup \{q_i\}, G \cup \{\overline{q_i}\})$

$$C_E ::= C_{F\cup\{q_i\}} \vee C_{G\cup\{\overline{q_i}\}}$$

Case a: $\tau(E^A) = \perp$ and $\tau(C_E) = \perp$.

Note that $\tau(C_{F\cup\{q_i\}}) = \tau(C_{G\cup\{\overline{q_i}\}}) = \perp$. Also, either $\tau(F \cup \{p_i\}) = \perp$ or $\tau(G \cup \{\overline{p_i}\}) = \perp$. In either case, the induction hypothesis yields that for some j, $\tau(A_j) = \perp$.

Case b: $\tau(E^B) = \bot$ and $\tau(C_E) = \top$.

Note that $E^B = (F \cup \{q_i\})^B \cup (G \cup \{\overline{q_i}\})^B$, and thus $\tau((F \cup \{q_i\})^B) = \tau((G \cup \{\overline{q_i}\})^B) = \bot$. Also, since $\tau(C_E) = \top$, either $\tau(C_{F\cup\{q_i\}}) = \top$ or $\tau(C_{G\cup\{\overline{q_i}\}}) = \top$. In either case the induction hypothesis yields that for some j, $\tau(B_j) = \bot$.

5. $E = F \cup G$ from $(F \cup \{r_i\}, G \cup \{\overline{r_i}\})$

$$C_E ::= C_{F\cup\{r_i\}} \wedge C_{G\cup\{\overline{r_i}\}}$$

This is similar to case 4.

$\square$

7 Lower bounds on proof length via monotone circuits

In this section we will use the results of the preceding section to show that known lower bounds on monotone circuit size provide lower bounds on the lengths of resolution refutations.

Here is how to build a set of clauses that encode the clique and coloring problems. Consider the following clauses:

$$\begin{array}{ll} \{q_{i,1}, q_{i,2}, \ldots, q_{i,n}\} & \text{for } 1 \le i \le k \\ \{\neg q_{i,m}, \neg q_{j,m}\} & \text{for } 1 \le m \le n \text{ and } 1 \le i < j \le k \\ \{\neg q_{i,m}, \neg q_{j,l}, p_{m,l}\} & \text{for } 1 \le m < l \le n \text{ and } 1 \le i, j \le k \end{array}$$

The above clauses encode a graph that contains a k-clique as follows:

- The q's encode a one-to-one function from $\{1, \ldots, k\} \to \{1, \ldots, n\}$ if we set $q_{i,j} = 1 \Leftrightarrow q(i) = j$. Thus the clause $\{q_{i,1}, \ldots, q_{i,n}\}$ which means $[(q(i) = 1 \vee \ldots \vee (q(i) = n)]$ says that $q(i)$ is defined (we could also add information saying that there is no more than *one* j such that $q(i) = j$, but that isn't needed for our purposes) and the clause $\{\neg q_{i,m}, \neg q_{j,m}\}$ (which is equivalent to $[q(i) = m \Rightarrow q(j) \neq m]$) ensures that the function is one-to-one.

- The p's encode a graph if we take $p_{m,l} = 1$ to mean that there's an edge between m and l. With this intuition, the last set of clauses $\{\neg q_{i,m}, \neg q_{j,l}, p_{m,l}\}$ is equivalent to $q(i) = m, q(j) = l \Rightarrow$ there is an edge between m and l.

Thus,$\{p_{m,l} : 1 \le m < l \le n\}$ encodes a graph containing a k-clique iff there exist assignments to the q variables making these clauses true.

Next, consider the sets of clauses that encode the property of being an l-partite graph:

$$\begin{array}{ll} \{r_{i,1}, \ldots, r_{i,l}\} & \text{for } 1 \le i \le n \\ \{\neg r_{i,a}, \neg r_{i,b}\} & \text{for } 1 \le i \le n, 1 \le a < b \le l \\ \{\neg r_{i,a}, \neg r_{j,a}, \neg p_{i,j}\} & \text{for } 1 \le a \le l \text{ and } 1 \le i < j \le n \end{array}$$

Here the explanation is as follows:

- The first set of clauses encode the coloring function: $r_{i,c} = 1$ means vertex i has color c, $(r(i) = c)$ and the second set of clauses means that each vertex has at most one color.

- Finally, we make this a proper coloring with the last set of clauses: If $r(i) = a$ and $r(j) = a$ then $p_{i,j} = 0$, (i.e. there is no edge between vertices i and j).

Claim: If $k = l + 1$ then these clauses (i.e.both sets together) are unsatisfiable.
Proof: Every assignment to the p's gives a graph. If all clauses are satisfiable, there is some assignment to the q's encoding a k-clique in the graph and at the same time an assignment to the r's that gives a proper $k-1$-coloring of the graph. This is of course impossible.

Theorem 7.1 *Any resolution refutation of these clauses requires length $2^{\Omega(\sqrt{k})}$, if $k \le \sqrt[4]{n}$.*

Proof It is known that any monotone circuit that evaluates to 1 on all of the k-cliques and evaluates to 0 on all of the $k-1$-partite graphs must have size at least $2^{\Omega(\sqrt{k})}$, for k in this range. (A nice proof is found in [2]; a slightly stronger lower bound appears in [1]. All of these use the proof technique developed in [11].) Since these clauses satisfy the restrictions in the hypothesis of Theorem 6.1, a lower bound on the length of a refutation follows immediately. □

It should be noted that strong lower bounds on the length of resolution refutations have been known since the work of [7]. For further discussion of the history of such results, see [5]. However, the proof presented above seems to be the first that explicitly makes use of circuit lower bounds. Further progress in this direction for the stronger "cutting planes" proof system also make use of these circuit lower bounds. These results are reported in [3, 10].

8 Interpolation for resolution with limited extension

It is natural to wonder if the results of the preceding sections can be extended to proof systems that are more powerful than resolution. One improvement in this direction involves the work on cutting planes [3] mentioned in the preceding paragraph. Another improvement is actually an immediate consequence of Theorem 5.1, and will be presented in this section.

The term "extension" refers to the process of taking some existing proof system, and extending it by allowing the introduction of new variables (say, σ_A) that can be used to represent the truth value of a propositional formula A. This allows short representations of long formulae, and implicitly allows a system such as resolution to deal with formulae that are not in CNF. The following paragraphs make this notion more precise, for the specific case of extending resolution in this way.

For every formula A, we will have a variable σ_A. In the case where A consists of a single propositional variable p, σ_A is just the formula p.

Now we will define, for each formula A, a set of clauses $LE(A)$.

Definition 8.1. "Limited Extension." $LE(A)$ is defined inductively. If $A = p$, then $LE(A) = \emptyset$.

- $LE(\neg A) ::= \{ \ \{\sigma_{\neg A}, \sigma_A\} \ \{\overline{\sigma_{\neg A}}, \overline{\sigma_A}\} \ \} \cup LE(A)$.
- $LE(A \wedge B) ::= \{ \ \{\overline{\sigma_{A\wedge B}}, \sigma_A\}, \{\overline{\sigma_{A\wedge B}}, \sigma_B\}, \ \ \{\sigma_{A\wedge B}, \overline{\sigma_A}, \overline{\sigma_B}\} \ \} \cup LE(A) \cup LE(B)$.
- $LE(A \vee B) ::= \{ \ \{\overline{\sigma_A}, \sigma_{A\vee B}\}, \{\overline{\sigma_B}, \sigma_{A\vee B}\}, \ \ \{\sigma_A, \sigma_B, \overline{\sigma_{A\vee B}}, \} \ \} \cup LE(A) \cup LE(B)$.

Note that these clauses ensure that any truth assignment satisfying the clauses has σ_A equal to the negation of $\sigma_{\neg A}$, $\sigma_{A\wedge B}$ is equal to the logical and of σ_A and σ_B, etc.

Definition 8.2. Let $\mathcal{A}$ be any set of formulae. Then define

$$LE(\mathcal{A}) ::= \bigcup_{\{A\in\mathcal{A}\}} LE(A).$$

Note that it is clear that $\mathcal{A}$ has a resolution refutation if and only if $\mathcal{A} \cup LE(\mathcal{A})$ has a resolution refutation.

Theorem 8.1 (Buss) *Let* $\Gamma = \mathcal{A} \cup \mathcal{B}$, *where* $\mathcal{A} = \{A_i(\vec{p}, \vec{q})\}$ *and* $\mathcal{B} = \{B_j(\vec{p}, \vec{r})\}$. *Let* $\Gamma \cup LE(\Gamma)$ *have a resolution refutation of length* n.

Then there is an interpolant $C(\vec{p})$ *such that*

$$\bigwedge_i A_i(\vec{p}, \vec{q}) \Rightarrow C(\vec{p})$$

and

$$C(\vec{p}) \Rightarrow \neg \bigwedge_j (B_j(\vec{p}, \vec{r}))$$

and $C(\vec{p})$ *has circuits of size* $\leq 3n$.

Proof Note that $\Gamma \cup LE(\Gamma) = (\mathcal{A} \cup LE(\mathcal{A})) \cup (\mathcal{B} \cup \mathcal{LE}(\mathcal{B}))$. Note that the only variables that are shared between $(\mathcal{A} \cup LE(\mathcal{A}))$ and $(\mathcal{B} \cup LE(\mathcal{B}))$ are the variables in $\vec{p}$. Thus Theorem 5.1 gives us an interpolant with circuit size $O(n)$ for $(\mathcal{A} \cup LE(\mathcal{A}))$ and $(\mathcal{B} \cup LE(\mathcal{B}))$. That is,

$$(\bigwedge_i A_i(\vec{p}, \vec{q})) \wedge LE(\mathcal{A}) \Rightarrow C(\vec{p})$$

and

$$C(\vec{p}) \Rightarrow \neg(LE(\mathcal{B}) \wedge \bigwedge_j (B_j(\vec{p},\vec{r})))$$

It suffices to observe now that this same $C(\vec{p})$ is also an interpolant for $\mathcal{A}$ and $\mathcal{B}$. But this is obvious, because of the following observations.

Let τ be any truth assignment with domain $\{\vec{p},\vec{q}\}$ that satisfies $\mathcal{A}$. Then there is a unique extension of τ that satisfies $(\mathcal{A} \cup LE(\mathcal{A}))$. Thus if $\tau(C(\vec{p})) = \bot$, must be the case that there is some i such that $\tau(A_i(\vec{p},\vec{q})) = \bot$.

Similarly, if $\tau(C(\vec{p})) = \top$, there must be some j such that $\tau(A_i(\vec{p},\vec{q})) = \bot$ (since otherwise this τ could be extended to satisfy $LE(\mathcal{B})$, in contradiction to C being an interpolant for $(\mathcal{A} \cup LE(\mathcal{A}))$ and $(\mathcal{B} \cup LE(\mathcal{B}))$). $\square$

Acknowledgments Thanks are due to Malka Rosenthal and Jing Wu for allowing us to incorporate here some material that they prepared as class notes for a class taught by Eric Allender, covering some of this material.

References

[1] Alon, B., and R. Boppana, The monotone circuit complexity of Boolean functions, *Combinatorica* **7** (1987) 1–22.

[2] Boppana, R., and M. Sipser, The complexity of finite functions, in *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, J. van Leeuwen, ed., MIT Press/Elsevier, 1990, pp. 757–804.

[3] Bonet, M., T. Pitassi, and R. Raz, Lower bounds for cutting planes proofs with small coefficients, *STOC* (1995).

[4] Buss, S.R., An Introduction to Proof Theory. Manuscript, 1995. To appear in *Handbook of Proof Theory.*

[5] Chvátal, V., and E. Szemerédi, Many hard examples for resolution, *J. ACM* **35** (1988) 759–768.

[6] Craig, W., Linear reasoning. A new form of the Herbrand-Gentzen theorem. *Journal of Symbolic Logic* **22** (1957) 250–268

[7] Haken, A., The intractability of resolution, *Theoretical Computer Science* **39** (1985) 297–308.

[8] Krajíček, J., Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic, preprint.

[9] Mundici, D., Tautologies with a unique Craig interpolant, uniform vs. nonuniform complexity, *Annals of Pure and Applied Logic* **27** (1984) 265–273.

[10] Pudlák, P., Lower bounds for resolution and cutting planes proofs and monotone computations, draft, 1995.

[11] Razborov, A., Lower bounds on the monotone complexity of some Boolean functions, *Dokl. Akad. Nauk. SSSR* **281** (1985) 798–801 (in Russian); English translation in: *Soviet Math. Dokl.* **31** (1985) 354–357.

Natural proofs and split versions of bounded arithmetic

Notes by Peter Clote and Pierre McKenzie[1]

Tuesday Morning, March 7th

In this lecture we will consider the concept of natural proofs [RR94] which we will then apply to bounded arithmetic [Raz94].

The notion of a *natural* proof was introduced by Razborov and Rudich [RR94], as an abstraction of the known non-monotone complexity lower bound proofs for explicit Boolean functions. We define *quasipolynomial-time natural* proofs here (the proofs discussed in [RR94] were polynomial-time natural).

Definition 1 Represent an n-ary Boolean function $f_n : \{0,1\}^n \to \{0,1\}$ by its truth table (this has size $N = 2^n$). A combinatorial property $C = \{C_n\}_{n \geq 1}$ is *quasipolynomial-time natural* against *P/poly* provided:

(0) For each n, C_n is a set of n-ary Boolean functions,

(1) Constructivity: $\{f_n : \{0,1\}^n \to \{0,1\} | f_n \in C_n\} \in \mathrm{TIME}(2^{(\log N)^{O(1)}})/poly$ (i.e. $\mathrm{TIME}(2^{n^r})/poly$, for some constant r, as opposed to $\mathrm{TIME}((2^n)^r)/poly$ in [RR94]),

(2) Largeness: $|C_n| \geq 2^{-cn} \cdot 2^{2^n}$ for some constant c (i.e. the probability that a randomly chosen Boolean function on n variables belongs to C_n is at least $\frac{1}{2^{cn}}$),

(3) Usefulness: If $f_n \in C_n$ for all n, then the family $\{f_n\}_{n \geq 1} \notin$ *P/poly*.

[1]Thanks to S.R. Buss for the use of macros `bussproofs.sty` in order to typeset Gentzen sequent calculus proofs.

In summary, C is quasipolynomial time natural against $P/poly$ if the sequence $\{C_n\}_{n\geq 1}$ belongs to $QP/poly$ but no "traversing sequence" $\{f_n\}_{n\geq 1}$ belongs to $P/poly$. The intuition behind the constructivity condition is that constructive proofs of $NP \not\subseteq P/poly$ ought to give a (quasi)polynomial time property which is natural against $P/poly$. The largeness and the usefulness conditions are observed properties of known lower bound proofs.

The main result of Razborov and Rudich is the non-existence, under the strong pseudo-random number generator (SPRNG) conjecture, of certain natural proofs.

Definition 2 (see [BlMi84]) Let $G_n : \{0,1\}^n \rightarrow \{0,1\}^{2n}$ be a pseudo-random number generator. The *hardness* of G_n, $H(G_n)$, is the least $S > 0$ such that, for some circuit C of size S,[2]

$$\left| \Pr_{\bar{x}\in\{0,1\}^n}[C(G_n(\bar{x})) = 1] \; - \Pr_{\bar{y}\in\{0,1\}^{2n}}[C(\bar{y}) = 1] \right| \;\geq\; \frac{1}{S}.$$

(By G_n we of course mean a sequence $\{G_n\}_{n\geq 0}$ so that S is a function of n.)

For example, the very poor generator $G_n(x) = xx$ has constant hardness, as seen by using the circuit C which, on input w, outputs 1 iff bit n of w equals bit $2n$ of w. On the other hand, if $G_n(x)$ were close to truly random, then only a large circuit C could presumably pick up the subtle difference between $\Pr[C(G_n(\bar{x})) = 1]$ and $\Pr[C(\bar{y}) = 1]$ (an extreme case for such a C would output 1 on input w iff w is in the range of G_n; the size of C is then an upper bound on $H(G_n)$; however such a large C is an overkill because it drives the probability difference very close to 1).

Conjecture 4. The SPRNG Conjecture states that for some $\epsilon > 0$, there exists a pseudo-random number generator $\{G_n\}_{n\geq 0}$ having polynomial size circuits, with $H(G_n) \geq 2^{n^\epsilon}$.

Theorem 3 *[Razborov-Rudich [RR94]] If the SPRNG conjecture is true, then there are no properties which are quasipolynomial-time natural against P/poly.*

[2]Circuits have arbitrary fan-in, and the size is the number of internal gates (which for polynomial size circuits is within a polynomial factor of the number of wires).

Proof [Following Razborov-Rudich]. Assume that natural proofs C_n exist. By constructivity and the usual simulation of Turing machines by circuits (see [Bor77]), $\{C_n\}_{n\geq 1}$ has circuit size 2^{n^r} for r a constant.

Let $k > 0$ vary, and pick $G^{(k)}$ a purported pseudo-random number generator, that is, $G^{(k)}$ maps $\{0,1\}^k$ to $\{0,1\}^{2k}$, $\{G^{(k)}\}_{k\geq 1}$ has circuits of size polynomial in k, and $H(G^{(k)}) \geq 2^{k^\epsilon}$ for a fixed ϵ. We must show for a contradiction that

$$H(G^{(k)}) < 2^{k^\epsilon}.$$

Let $n = \lfloor k^{\epsilon/2r} \rfloor$ and

$$F_n = \{f : \{0,1\}^n \to \{0,1\}\} \cong \{0,1\}^{2^n}.$$

We iterate $G^{(k)}$ to form a function $f^{(k)} : \{0,1\}^k \to F_n$ as follows. For $\bar{x} \in \{0,1\}^k$, define

$$\begin{aligned} G_0^{(k)}(\bar{x}) &= \text{low order } k \text{ bits of } G^{(k)}(\bar{x}), \\ G_1^{(k)}(\bar{x}) &= \text{high order } k \text{ bits of } G^{(k)}(\bar{x}). \end{aligned}$$

Then for $y \in \{0,1\}^n$, $y = y_1 y_2 \ldots y_n$, define (dropping the ${}^{(k)}$ superscript on the subscripted G's)

$$G_y = G_{y_n} \circ G_{y_{n-1}} \circ \cdots \circ G_{y_2} \circ G_{y_1} : \{0,1\}^k \to \{0,1\}^k$$

and for $\bar{x} \in \{0,1\}^k$ define

$$(f^{(k)}(\bar{x}))(y) = \text{low order bit of } G_y(\bar{x}). \tag{7}$$

In this way indeed $(f^{(k)}(\bar{x})) \in F_n$, for each $\bar{x} \in \{0,1\}^k$. Moreover $(f^{(k)}(\bar{x}))$ is computable with circuits of size polynomial in n (and polynomial in k, since k and n are polynomially related); this is because $G^{(k)}$, hence G_0 and G_1, have circuits of size polynomial in k, and $(f^{(k)}(\bar{x}))$ is computable using a cascade of n subcircuits for G_0 and G_1.

By the usefulness property of the natural proof C_n, for all n_0 there exists $k > n_0$ such that $(f^{(k)}(\bar{x})) \notin C_n$ for any $\bar{x} \in \{0,1\}^k$. (Indeed suppose to the contrary that only finitely many k had the property: $(f^{(k)}(\bar{x})) \notin C_n$ for all $\bar{x} \in \{0,1\}^k$. Then for some n_0 and each $k > n_0$ one could find $\bar{x}(k)$ such that $(f^{(k)}(\bar{x}(k))) \in C_n$. This could be extended into a "traversing sequence" of C_n in *P/poly*.) Hence, for infinitely many k, the circuit for C_n (which we also denoted C_n) outputs 0 on input $f^{(k)}(\bar{x})$:

$$\Pr_{x\in\{0,1\}^k}[C_n(f^{(k)}(x)) = 1] = 0.$$

On the other hand, by the largeness property of C_n,

$$\Pr_{f\in F_n}[C_n(f) = 1] \geq 2^{-cn},$$

for some constant $c > 0$.

Now we use a trick from [GGM86]. Let T be a complete binary tree of height n. Enumerate the internal nodes of T as $v_1, v_2, \ldots, v^{2^n-1}$ so that a node is enumerated only after its children have been enumerated. Denote by T_0 the set of leaves of T and by T_i, $0 < i \leq n$, the subforest of T induced by $T_0 \cup \{v_1, v_2, \ldots v_i\}$.

Now for each node v in T, let x_v be a variable ranging over $\{0,1\}^k$. For each i, $0 \leq i \leq n$, and for y a leaf and v the root of the (unique) tree in T_i subtending y, define $G_{i,y} \in \{0,1\}^k$ as follows:

$$G_{i,y} = \begin{cases} x_v & \text{if } y = v \text{ (} y \text{ a tree by itself)}, \\ G_{y_n} \circ G_{y_{n-1}} \circ \cdots \circ G_{y_{n-\mathrm{height}(v)+1}}(x_v) & \text{otherwise.} \end{cases}$$

Finally, define

$$f_i(y) = \text{low order bit of } G_{i,y}.$$

Note that the variables $x_v \in \{0,1\}^k$ on which f_i depends are those such that v is a root of one of the trees in T_i. For fixed x_v's, $f_i \in F_n$, that is, f_i is a function on n Boolean variables.

Now when $i = 0$, $f_i(y)$ and $f_i(y')$ for $y \neq y'$ are independent, because, for each y, $G_{0,y}$ is the independent variable x_y. Hence

$$\Pr_{\text{all } x_v\text{s}}[C_n(f_0) = 1] \;=\; \Pr_{f\in F_n}[C_n(f) = 1)] \;\geq\; 2^{-cn}.$$

(Note that, here and further on, whether we take probabilities over all x_v's, or only over those x_v's which are known to be relevant, makes no difference.) Yet when $i = 2^n - 1$, f_i only depends on $x_{v_{2^n-1}}$ and is exactly the function $f^{(k)}(x_{v_{2^n-1}})$ defined by equation (7), so that

$$\Pr_{x_{v_{2^n-1}}\in\{0,1\}^k}[C_n(f_{2^n-1}) = 1] \;=\; \Pr_{x\in\{0,1\}^k}[C_n(f^{(k)}(x)) = 1)] \;=\; 0.$$

Therefore, there is some $i \in \{0, 1, \dots, 2^n - 2\}$ such that

$$\Pr_{\text{all } x_v\text{s}}[C_n(f_i) = 1] \; - \; \Pr_{\text{all } x_v\text{s}}[C_n(f_{i+1}) = 1] \; \geq \; 2^{-cn}/2^n \; = \; 2^{-(c+1)n}. \tag{8}$$

Now how does f_i differ from f_{i+1}? Well, f_i depends on all variables x_v for v a root in T_i, and T_{i+1} differs from T_i by the addition of a node w having as immediate children two roots in T_i, say w' and w''. Hence f_i depends on a set of variables $V \cup \{x_{w'}, x_{w''}\}$, while f_{i+1} depends on $V \cup \{x_w\}$.

Among the possible settings for the set of variables $x_v \in \{0,1\}^k$ for $v \notin \{w, w', w''\}$ (the set of such x_v's includes V and all variables on which neither f_i nor f_{i+1} depends), choose a setting for which

$$\Pr_{x_{w'}, x_{w''}}[C_n(f_i) = 1] \; - \; \Pr_{x_w}[C_n(f_{i+1}) = 1] \; \geq \; 2^{-(c+1)n}. \tag{9}$$

(This is possible because (8) asserts that the average of the probability differences over all possible settings exceeds $2^{-(c+1)n}$.)

Now define $C^*(x_{w'}, x_{w''})$ as a circuit which calculates $C_n(f_i)$, with all other x_v's fixed as per the setting chosen above. This circuit has $2k$ inputs. The circuit first computes the size-2^n truth table of f_i by iterating over each $y \in \{0,1\}^n$ and by following the rules for constructing $G_{i,y}$, repeatedly using the polynomial size circuits for G_0 and G_1. The circuit then feeds the truth table for f_i into the size-2^{n^r} circuit for C_n. The total size of C^* is thus in $O(2^{n^r})$.

Finally,

$$\Pr_{x_w \in \{0,1\}^k}[C^*(G^{(k)}(x_w)) = 1] \;\; = \Pr_{x_w \in \{0,1\}^k}[C_n(f_{i+1}) = 1]$$

because computing f_{i+1} as a function of x_w exactly amounts to computing f_i as a function of $x_{w'} = G_0^{(k)}(x_w)$ and $x_{w''} = G_1^{(k)}(x_w)$. Moreover,

$$\Pr_{y \in \{0,1\}^{2k}}[C^*(y) = 1] \; = \Pr_{x_{w'}, x_{w''}}[C_n(f_i) = 1]$$

by definition.

Therefore, substituting into (9),

$$\left| \Pr_{y \in \{0,1\}^{2k}}[C^*(y) = 1] \; - \Pr_{x \in \{0,1\}^k}[C^*(G^{(k)}(x)) = 1] \right| \; \geq \; 2^{-(c+1)n}.$$

Since C^* has size at most $d2^{n^r}$ for some constant d, and $2^{-(c+1)n} \geq 1/(d2^{n^r})$ for sufficiently large n, the hardness of $G^{(k)}$ verifies

$$H(G^{(k)}) \leq d2^{n^r} = d2^{(k^{\epsilon/2r})^r} = d2^{k^{\epsilon/2}} < 2^{k^\epsilon},$$

for sufficiently large k.

This concludes the proof of the Razborov-Rudich theorem. ∎

The early oracle results of Baker, Gill and Solovay indicate that if $P \neq NP$ then only a non-relativizable proof technique can establish this. The Razborov-Rudich result just proved indicates that if $P \neq NP$ then (assuming one believes the SPRNG conjecture) a different approach is necessary than the random restriction and algebraic boolean circuit lower bound techniques so far introduced. In [Raz94] A.A. Razborov applied Theorem 3 to show (essentially) that $P \neq NP$ cannot be proved in a certain theory of bounded arithmetic, provided that the SPRNG conjecture holds. Specifically, Razborov's theorem below states that superpolynomial lower bounds for languages in PH cannot be proved in a certain theory $S_2^2(\alpha)$ of bounded arithmetic, unless the SPRNG conjecture fails. Recall that if $P = NP$ then NP (and hence PH) has polynomial size circuits. As well, if NP has polynomial size circuits then PH collapses (Karp-Lipton). Thus if $S_2^2(\alpha)$ proves $P \neq NP$ then (presumably[3]) $S_2^2(\alpha)$ proves a superpolynomial lower bound for a set in PH, contradicting the SPRNG conjecture.

Bounded arithmetic was introduced by R. Parikh and more fully developed by J. Paris, A. Wilkie [PW83, WP87] and S.R. Buss [Bus86] (see as well the monograph [Kra95] by J. Krajíček).

Definition 4 (S.R. Buss [Bus86]) The language of *bounded arithmetic* consists of the binary relation symbol $\leq$, the 0-ary constant symbol 0, the unary successor function symbol $S(x) = x + 1$, the binary addition and multiplication symbols $+$, $\cdot$, the unary function symbols $\lfloor \frac{x}{2} \rfloor$, $|x| = \lceil \log_2(x+1) \rceil$, and the binary function symbol $x \# y = 2^{|x| \cdot |y|}$ pronounced *smash*. *Bounded quantifiers* [resp. *sharply bounded quantifiers*] are of the form $\exists x < t$, $\forall x < t$, [resp. $\exists x < |t|$, $\forall x < |t|$], where t is a term.

(1) $\Sigma_0^b = \Pi_0^b = \Delta_0^b$ is the collection of formulas in the language of bounded arithmetic, all of whose quantifiers are sharply bounded;

[3]This depends on whether the contrapositive of the Karp-Lipton theorem can be proved in $S_2^2(\alpha)$. We haven't checked whether the usual proof of Karp-Lipton formalizes in $S_2^2(\alpha)$.

(2) Σ^b_{n+1} is the smallest collection of formulas containing Π^b_n, and closed under $\wedge$, $\vee$, sharply bounded quantifiers, and existential bounded quantifiers $\exists x < t$;

(3) Π^b_{n+1} is the smallest collection of formulas containing Σ^b_n, and closed under $\wedge$, $\vee$, sharply bounded quantifiers, and universal bounded quantifiers $\forall x < t$;

(4) $\Sigma^b_\infty = \cup_n \Sigma^b_n$.

Theories of bounded arithmetic are formulated using Gentzen sequent calculus. The collection $BASIC$ consists of a finite number of axioms concerning the function and relation symbols of bounded arithmetic ([Bus86], p. 30). For each axiom $A \in BASIC$, there is an *initial sequent* $\vdash A$. The first order theory S^n_2 of bounded arithmetic includes the Σ^b_n-PIND induction rule of inference:

$$\frac{\Gamma, \Phi(\lfloor a/2 \rfloor) \rightarrow \Phi(a), \Delta}{\Gamma, \Phi(0) \rightarrow \Phi(t), \Delta}$$

where the *eigenvariable* a does not appear in the lower sequent, t is a term of bounded arithmetic, and Φ is a Σ^b_n formula. The theory T^n_2 includes the Σ^b_n-IND induction inference:

$$\frac{\Gamma, \Phi(a) \rightarrow \Phi(a+1), \Delta}{\Gamma, \Phi(0) \rightarrow \Phi(t), \Delta}$$

where the *eigenvariable* a does not appear in the lower sequent, t is a term of bounded arithmetic, and Φ is a Σ^b_n formula.

It is known that the Σ^b_n-PIND rule is equivalent to the Π^b_n-PIND rule, and similarly the Σ^b_n-IND rule is equivalent to the Π^b_n-IND rule [Bus86]. The following theorem, proved by cut elimination (together with a function algebra characterization of the collection of polynomial time computable functions) indicates that the theory S^1_2 corresponds exactly to polynomial time.

Theorem 5 (S.R. Buss [Bus86] p. 86) *A function $f(\vec{x})$ is polynomial time computable if and only if there is a Σ^b_1 representation $\Phi(\vec{x}, y)$ of the graph "$f(\vec{x}) = y$" of f for which S^1_2 proves the sequent*

$$\vdash (\forall \vec{x})(\exists y)\Phi(\vec{x}, y) \wedge (\forall \vec{x}, y, z)(\Phi(\vec{x}, y) \wedge \Phi(\vec{x}, z) \supset y = z).$$

Moreover, a language L is in the polynomial time hierarchy PH if and only if L is definable by a Σ^b_∞ formula [Bus86].

Definition 6 (A.A. Razborov [Raz94]) Let α and β be new unary predicate symbols. The second order theories $S^n_2(\alpha,\beta)$, $T^n_2(\alpha,\beta)$ are defined by allowing α, β to appear in the previous axioms and rules of inference for S^n_2, T^n_2. Let $\Sigma^b_\infty(\alpha)$ denote all bounded formulas in language of $\{0,S,+,\cdot,\lfloor\frac{x}{2}\rfloor,|x|,\#\}\cup\{\alpha\}$. Let $\Sigma^b_1(X)$ [resp. $\Pi^b_1(X)$] be the closure of X under $\wedge,\vee$, sharply bounded quantification and existential [resp. universal] bounded quantification,[4] and let $\Sigma^b_{i+1}(X)$ be $\Sigma^b_1(\Pi^b_i(X))$, $\Pi^b_{i+1}(X)$ be $\Pi^b_i(\Sigma^b_i(X))$. Define $\mathbf{S}\Sigma^b_i$ to be $\Sigma^b_i(\Sigma^b_\infty(\alpha),\Sigma^b_\infty(\beta))$.

Definition 7 The theory of Split S^i_2, denoted by $\mathbf{S}S^i_2$ has initial sequents $\vdash A$ for $A \in BASIC$ and, along with the other rules of sequent calculus, has the $\mathbf{S}\Sigma^b_i$-PIND rule of inference. Split T^i_2, denoted by $\mathbf{S}T^i_2$, is similarly determined by the $\mathbf{S}\Sigma^b_i$-IND rule of inference.

Let $N \geq 0$ and $n = |N| \simeq \log(N)$, and suppose $t(n) = n^{\omega(1)}$, a superpolynomial function. If $S(N,x)$ is a Σ^b_∞-formula then let $LB(t,S,\alpha)$ be the statement

$\neg[\alpha$ codes a circuit of size $\leq t(n)$ such that
$\forall x \in \{0,1\}^n((\alpha(x)$ outputs $1) \leftrightarrow S(N,x))]$

Note that

a. The free variables of $LB(t,S,\alpha)$ are N and α, and that LB is a $\Sigma^b_1(\alpha,S)$ formula, since the quantification $\forall x \in \{0,1\}^n$ is of the form

$$(\forall x < 2N)[|x| = |N| \to \cdots]$$

b. By "α codes a circuit", we mean that α encodes the gate types and gate connections in some straightforward manner, (encoding for instance the *direct connection language* or *extended connection language* of [Ruz81]) plus, optionally, α may encode the full truth-table description of the functions computed by every gate in the bounded fan-in circuit.

[4] Note that in the case at hand X is closed under complementation.

The formula $LB(t, S, \alpha)$ states that the property defined by S is not computable by α, an undetermined unary relation symbol ranging over all circuits of superpolynomial size t. The remainder of this lecture is dedicated to showing Razborov's [Raz94] result that superpolynomial lower bounds are not provable in $S_2^2(\alpha)$, unless the SPRNG conjecture fails.

Theorem 8 *Let $SLB(t, S, \alpha, \beta)$ be the formula*

> $\neg[\alpha$ *codes a circuit of size* $< (\frac{t}{2} - 1)$ *and* β *codes a circuit of size* $< (\frac{t}{2} - 1)$ *and* $\forall x \in \{0,1\}^n[(\alpha(x) = 1) \oplus (\beta(x) = 1) \leftrightarrow S(N, x)]]$.

If $S_2^2(\alpha) \vdash LB(t, S, \alpha)$ *then* $\mathbf{S}S_2^2(\alpha, \beta) \vdash SLB(t, S, \alpha, \beta)$.

Proof If $\mathbf{S}S_2^2 \not\vdash SLB(t, S, \alpha, \beta)$, then by the completeness theorem of logic, there is a model

$$(M, \alpha^M, \beta^M) \models \mathbf{S}S_2^2 \wedge \neg SLB(t, S, \alpha, \beta).$$

Define $\delta^M \subseteq M$ to encode the circuit $\alpha^M \oplus \beta^M$. Then

$$(M, \delta^M) \models S_2^2(\alpha) \wedge \neg LB(t, S, \alpha)$$

where the unary predicate symbol α is interpreted by δ^M. But by the completeness theorem, the hypothesis implies that all models of $S_2^2(\alpha)$ satisfy $LT(t, S, \alpha)$. Contradiction. ∎

Noting that $\gamma \oplus (S \oplus \gamma) \equiv S$, we can rephrase $SLB(t, S, \alpha, \beta)$ by introducing a new predicate symbol γ. Let the formula $CC(t, T(x), \alpha)$ state

- if $\mathbf{S}S_2^2 \vdash SLB(t, S, \alpha, \beta)$ then

  $$\mathbf{S}S_2^2 \vdash \neg CC(\frac{t}{2} - 1, \gamma, \alpha) \vee \neg CC(\frac{t}{2} - 1, S \oplus \gamma, \beta)$$

  where $CC(t, T(x), \alpha)$ states that

  > [α codes a circuit of size $\leq t$ and for all $x \in \{0,1\}^n$
  > $(\alpha(x) = 1 \leftrightarrow T(x))$]

Here, CC stands for *codes a circuit*. Equivalently, in sequent form, $\mathbf{S}S_2^2$ proves

$$CC(\tfrac{t}{2}-1,\gamma,\alpha), CC(\tfrac{t}{2}-1, S\oplus\gamma,\beta)\rightarrow$$

By [Bus90], S^2_2 is $\forall\Sigma^b_2$-conservative over T^1_2; in other words, every formula of the form $\forall\Sigma^b_2$ which is provable in S^2_2 is provable in T^1_2, and conversely. The same proof yields that $\mathbf{S}S^2_2$ is $\forall\Sigma^b_2$-conservative over $\mathbf{S}T^1_2$. Since CC is a Π^b_1-formula, we know that this sequent is also provable in $\mathbf{S}T^1_2$.

Theorem 9 (A.A. Razborov [Raz94]) *If $\mathbf{S}S^2_2\vdash SLB(t,S,\alpha,\beta)$ for some $t=n^{\omega(1)}$ and $S\in\Sigma^b_\infty$, then the SPRNG conjecture is false.*

Corollary 10 *If the SPRNG conjecture holds, then $S^2_2(\alpha)$ does not prove superpolynomial lower bounds on circuit size for any bounded formula (i.e., for any polynomial time hierarchy predicate).*

Proof of Theorem [This version of the proof is due to Krajicek]. We shall prove that, if $\mathbf{S}T^1_2$ proves

$$\vdash CC(t,\gamma,\alpha), CC(t,S\oplus\alpha,\beta)\rightarrow$$

then there are quasipolynomial size circuits which are natural against $P/poly$.

First Step:
Convert the $\mathbf{S}T^1_2$ proof and the sequent into statements in propositional logic, using a technique first given by Paris-Wilkie [PW83] (see p. 293 of [Kra94] for details of translation).

Expand the language to include a β-function which encodes sequences (as in [Bus86]). As earlier mentioned in the case of T^b_1, $\mathbf{S}T^b_1$ can be shown to be equivalent to the system with the $\mathbf{S}\Pi^b_1$-IND rule in place of $\mathbf{S}\Sigma^b_1$-IND. Using free-cut elimination [Bus86], without loss of generality, we may assume that every formula in the $\mathbf{S}T^1_2$-proof is of the form

$$(\forall y\le r)(\exists z\le |r'|)\Phi \tag{10}$$

where Φ is a boolean combination of $\Sigma^b_\infty(\alpha)$ and $\Sigma^b_\infty(\beta)$ formulas and of formulas $\gamma(\ldots)$.

To convert to propositional logic:
Use variables $\vec{q}$ for the values of $\alpha(x)$, i.e. q_i denotes $\alpha(i)$. Use variables $\vec{r}$ for the values of $\beta(x)$, and variables $\vec{p}$ for the values of $\gamma(x)$. As follows, define the translation $\langle A(\vec{x})\rangle_n$ for the sequence $\vec{n}$ of natural numbers of the same length as sequence $\vec{x}$ of variables.

1. If $A(\vec{n})$ is a true [resp. false] atomic sentence (such as $m_1 + m_2 = m_3$, etc.), then $\langle A(\vec{x})\rangle_{\vec{n}}$ is 1 [resp. 0].
2. If $t(\vec{x})$ is a term, $A(\vec{x})$ is the atomic formula $\alpha(t(\vec{x}))$ and $m = t(\vec{n})$, then $\langle A(\vec{x})\rangle_{\vec{n}}$ is q_m.
3. If $t(\vec{x})$ is a term, $A(\vec{x})$ is the atomic formula $\beta(t(\vec{x}))$ and $m = t(\vec{n})$, then $\langle A(\vec{x})\rangle_{\vec{n}}$ is r_m.
4. If $t(\vec{x})$ is a term, $A(\vec{x})$ is the atomic formula $\gamma(t(\vec{x}))$ and $m = t(\vec{n})$, then $\langle A(\vec{x})\rangle_{\vec{n}}$ is p_m.
5. If $A(\vec{x})$ is $B(\vec{x}) \wedge C(\vec{x})$ [resp. $B(\vec{x}) \vee C(\vec{x})$, resp. $\neg B(\vec{x})$] then $\langle A(\vec{x})\rangle_{\vec{n}}$ is $\langle B(\vec{x})\rangle_{\vec{n}} \wedge \langle C(\vec{x})\rangle_{\vec{n}}$ [resp. $\langle B(\vec{x})\rangle_{\vec{n}} \vee \langle C(\vec{x})\rangle_{\vec{n}}$, resp. $\neg\langle B(\vec{x})\rangle_{\vec{n}}$].
6. If $t(\vec{x})$ is a term, $m = t(\vec{n})$, and $A(\vec{x})$ is the formula $(\exists y \leq t(\vec{x}))B(\vec{x}, y)$ [resp. $(\forall y \leq t(\vec{x}))B(\vec{x}, y)$], then $\langle A(\vec{x})\rangle_{\vec{n}}$ is $\bigvee_{i=0}^{m}\langle B(\vec{x}, y)\rangle_{\vec{n},i}$ [resp. $\bigwedge_{i=0}^{m}\langle B(\vec{x}, y)\rangle_{\vec{n},i}$].

The formula (10) is converted into propositional form

$$\bigwedge_{i=0}^{2^{n^{O(1)}}} \bigvee_{j=0}^{n^{O(1)}} E_{i,j} \tag{11}$$

where after simplifying formulas involving 1 (TRUE) and 0 (FALSE), each $E_{i,j}$ is either

1. P_i or $\neg P_i$
2. involves only $\vec{p}$ and $\vec{q}$, or
3. involves only $\vec{p}$ and $\vec{r}$.

Fixing N and letting $f(x)$ be the characteristic function of the formula $S(N, x)$, we obtain by this process a propositional sequent calculus proof of

$$\textstyle\bigwedge_i A_i(\vec{p}, \vec{q}), \bigwedge_j B_j(\vec{p}, \vec{r}) \rightarrow$$

where

1. $\{A_i(\vec{p}, \vec{q})\}_i$ is a set of clauses stating that $\vec{q}$ codes a circuit of size t computing the function γ with graph given by $\vec{p}$.

2. $\{B_j(\vec{p},\vec{r})\}_j$ is a set of clauses stating that $\vec{r}$ codes a circuit of size t computing $\gamma \oplus f$.

3. f does not have a circuit of size $2t + O(1)$.

4. Each formula in the proof is a conjunction of disjunctions of formulas involving only $\vec{p},\vec{q}$ or only $\vec{p},\vec{r}$ – see (11).

5. Each sequent has at most c many formulas for c a constant (independent of N).

6. The proof has only $2^{n^{O(1)}}$ many symbols.

Second Step: Remove the $\bigwedge$'s from the proof as follows. Given a sequent

$$\bigwedge_i^{p_1} E_{1,i}, \dots, \bigwedge_i^{p_{c'}} E_{c',i} \rightarrow \bigwedge_i^{q_1} F_{1,i}, \dots, \bigwedge_i^{q_{c''}} F_{c'',i}$$

replace it with the $q_1 \cdot q_2 \cdot \ldots \cdot q_{c''}$ many sequents

$$E_{1,1}, E_{1,2}, \dots E_{1,p_1}, E_{2,1}, \dots, E_{c',1}, \dots, E_{c',p_{c'}} \rightarrow F_{1,i_1}, F_{2,i_2}, \dots, F_{c'',i_{c''}}$$

Since each $q_i = 2^{n^{O(1)}}$ and $c'' = O(1)$, this is still only $2^{n^{O(1)}}$ many sequents.

In the proof, a cut on $\bigwedge_i^P F_i$ occurring in the cut rule

$$\frac{\Gamma \rightarrow \Delta, \bigwedge_{i=1}^p F_i \qquad \bigwedge_{i=1}^p F_i, \Gamma \rightarrow \Delta}{\Gamma \rightarrow \Delta}$$

is replaced by p many cuts; i.e.

$$\dfrac{\Gamma^* \rightarrow \Delta^*, F_p \qquad \dfrac{\vdots}{F_p, \Gamma^* \rightarrow \Delta^*}}{\Gamma^* \rightarrow \Delta^*}$$

$$\dfrac{\Gamma^* \rightarrow \Delta^*, F_3 \qquad \dfrac{\Gamma^* \rightarrow \Delta^*, F_2 \qquad \dfrac{\Gamma^* \rightarrow \Delta^*, F_1 \qquad F_1, F_2, \dots, F_p, \Gamma^* \rightarrow \Delta^*}{F_2, F_3, \dots, F_p, \Gamma^* \rightarrow \Delta^*}}{F_3, F_4, \dots, F_p, \Gamma^* \rightarrow \Delta^*}}{F_4, \dots, F_p, \Gamma^* \rightarrow \Delta^*}$$

We thus obtain a treelike sequent calculus proof of

$$A_1(\vec{p},\vec{q}), \dots, A_k(\vec{p},\vec{q}), B_1(\vec{p},\vec{r}), \dots, B_l(\vec{p},\vec{q}), \rightarrow$$

such that every formula in the proof is a disjunction of formulas which either involve just $\vec{p}$ and $\vec{q}$ or involve just $\vec{p}$ and $\vec{r}$.

Third Step: Convert to a refutation in resolution with limited extension. Each sequent in the proof obtained in the second step has the form:

$$\bigvee_i^{p_1} E_{1,i}, \bigvee_i^{p_2} E_{2,i}, \ldots, \bigvee_i^{p_u} E_{u,i}, \rightarrow \bigvee_i^{q_1} F_{1,i}, \ldots, \bigvee_i^{q_v} F_{v,i} \tag{12}$$

where each $E_{a,i}, F_{a,i}$ involves only $\{\vec{p}, \vec{q}\}$ or $\{\vec{p}, \vec{r}\}$. Associate with the sequent (12), the following set of clauses

$$\begin{gathered} \{\{E_{1,i} : i = 1, \ldots, p_1\}, \ldots, \{E_{u,i} : i = 1, \ldots, p_u\}\} \\ \cup \\ \{\overline{F_{1,1}}\}, \{\overline{F_{1,2}}\}, \ldots, \{\overline{F_{1,q_1}}\}, \{\overline{F_{2,1}}\}, \ldots, \{\overline{F_{v,q_v}}\} \end{gathered} \tag{13}$$

Now (13) is not a proper set of clauses, since clauses are supposed to contain literals, so instead we really use the extension variables, to form the following set (14) of clauses

$$\begin{gathered} \{\{\sigma_{E_{1,i}}\}_{i=1,\ldots,p_1}, \ldots, \{\sigma_{E_{u,i}}\}_{i=1,\ldots,p_u}\}\} \\ \cup \\ \{\{\sigma_{\neg F_{1,1}}\}, \{\sigma_{\neg F_{1,2}}\}, \ldots, \{\sigma_{\neg F_{1,q_1}}\}, \ldots, \{\sigma_{\neg F_{v,q_v}}\}\}. \end{gathered} \tag{14}$$

If the sequent (12) is $\Gamma \rightarrow \Delta$, then the set (14) of clauses is denoted $(\Gamma \rightarrow \Delta)^{LE}$.

Lemma 11 *If $\Gamma \rightarrow \Delta$ is derived in m lines of the sequent calculus proof above, then*

$$(\Gamma \rightarrow \Delta)^{LE} \cup LE(\vec{p}, \vec{q}) \cup LE(\vec{p}, \vec{r})$$

has a resolution refutation (not necessarily tree-like) of $O(m^2)$ resolution inferences.

Proof By induction on m. The proof splits into cases depending on the last inference.

Case(1) $\Gamma \to \Delta$ is $A \to A$.

(a) $\{\{\sigma_{\neg A}\}, \{\sigma_A\}\} \cup LE(\neg A)$ has a resolution refutation.

(b) If $A = \bigvee_{i=1}^{n} A_i$, then

$$\{\{\sigma_{A_1}, \ldots, \sigma_{A_u}\}, \{\sigma_{\neg A_1}\}, \ldots, \{\sigma_{\neg A_1}\}\} \cup LE(\neg A_1, \ldots, \neg A_u)$$

has a resolution refutation of $O(n)$ inferences.

Case(2) Suppose $A = \bigvee_{i=1}^{n} A_i$ involves only $\vec{p}, \vec{q}$. Then $\{\sigma_A\}$ and $\{\sigma_{A_1}, \ldots, \sigma_{A_u}\}$ can be derived from each other in the presence of $LE(A)$. So it is not important how we express formulas as disjunctions when we have a choice.

Case(3) $\wedge$:left and $\wedge$:right inferences involve only formulas that use just $\vec{p}$, $\vec{q}$ or just $\vec{p}$, $\vec{r}$; these are therefore straightforward (the $\wedge$:right case is harder than the $\wedge$:left case).

Case(4) An $\vee$:left inference is of the form

$$\frac{\bigvee_i E_i, \Gamma \to \Delta \qquad \bigvee_j F_j, \Gamma \to \Delta}{\bigvee \{E_i, F_j\}_{i,j}, \Gamma \to \Delta}$$

The induction hypotheses gives refutations R_1 and R_2

$$\left.\begin{array}{l} (\Gamma \to \Delta)^{LE} \\ \{\sigma_{E_i}\}_i \\ LE(\vec{p}, \vec{q}) \\ LE(\vec{p}, \vec{r}) \end{array}\right\} \stackrel{R_1}{\Longrightarrow} \emptyset$$

$$\left.\begin{array}{l} (\Gamma \to \Delta)^{LE} \\ \{\sigma_{F_j}\}_j \\ LE(\vec{p}, \vec{q}) \\ LE(\vec{p}, \vec{r}) \end{array}\right\} \stackrel{R_2}{\Longrightarrow} \emptyset$$

Combine these as follows.

$$\left.\left.\begin{array}{l}(\Gamma \to \Delta)^{LE}\\ \{\sigma_{E_i}\}_i \cup \{\sigma_{F_j}\}_j\\ LE(\vec{p},\vec{q})\\ LE(\vec{p},\vec{r})\end{array}\right\} \overset{R_1'}{\Longrightarrow} \begin{array}{l}\{\sigma_{F_j}\}\\ (\Gamma \to \Delta)^{LE}\\ LE(\vec{p},\vec{q})\\ LE(\vec{p},\vec{r})\end{array}\right\} \overset{R_2}{\Longrightarrow} \emptyset$$

Here, R_1' is like R_1 but uses $\{\sigma_{E_i}\}_i \cup \{\sigma_{F_j}\}_j$ in place of $\{\sigma_{E_i}\}_i$.

Remark Note that the refutation produced is not tree-like since $\{\sigma_{F_j}\}_j$ may be used multiple times in R_2.

<u>Case(5)</u> The last inference is the cut rule.

$$\frac{\Gamma \to \Delta, \bigvee_i A_i \qquad \bigvee_i A_i, \Gamma \to \Delta}{\Gamma \to \Delta}$$

The induction hypotheses gives refutations R_1 and R_2

$$\left.\begin{array}{l}(\Gamma \to \Delta)^{LE}\\ \{\sigma_{\neg A_1}\}, \ldots, \{\sigma_{\neg A_u}\}\\ LE(\vec{p},\vec{q})\\ LE(\vec{p},\vec{r})\end{array}\right\} \overset{R_1}{\Longrightarrow} \emptyset$$

and

$$\left.\begin{array}{l}(\Gamma \to \Delta)^{LE}\\ \{\sigma_{A_1}, \ldots, \sigma_{A_u}\}\\ LE(\vec{p},\vec{q})\\ LE(\vec{p},\vec{r})\end{array}\right\} \overset{R_2}{\Longrightarrow} \emptyset$$

Combine these as below, with R_1' minus any uses of $\{\sigma_{\neg A_i}\}$.

$$\left.\begin{array}{l}(\Gamma \to \Delta)^{LE}\\ LE(\vec{p},\vec{q})\\ LE(\vec{p},\vec{r})\end{array}\right\} \overset{R_1'}{\Longrightarrow} \ldots \left.\begin{array}{l}(\Gamma \to \Delta)^{LE}\\ \{\sigma_{A_1}, \ldots, \sigma_{A_u}\}\\ LE(\vec{p},\vec{q})\\ LE(\vec{p},\vec{r})\end{array}\right\} \overset{R_2}{\Longrightarrow} \emptyset$$

This completes the proof of the lemma. ∎

The following is a corollary to Lemma 11 and an earlier interpolation theorem for resolution.

Corollary 12 *There is a circuit $C(\vec{p})$ of size $2^{n^{O(1)}}$ such that,*

(1) *If $C(\vec{p}) = 0$, then $\{A_i(\vec{p}, \vec{q})\}_i$ is unsatisfiable.*

(2) *If $C(\vec{p}) = 1$, then $\{B_j(\vec{p}, \vec{r})\}$ is unsatisfiable.*

Note that the size of $C(\vec{p})$ is $2^{(\log N)^{O(1)}}$ which is quasipolynomial in $N = 2^n$. In case (1), when $C(\vec{p}) = 0$, the function $\gamma(x)$ does not have a circuit of size $t = n^{\omega(1)}$. In case(2), when $C(\vec{p}) = 1$, the function $(\gamma \oplus f)(x)$ does not have a circuit of size $t = n^{\omega(1)}$, where we recall that $f(x)$ does not have a circuit of size $2t + 1$.

Definition 13 Let $C^*(\vec{p})$ be defined by $(\neg C(\vec{p})) \vee C(\vec{p} \oplus f)$ where $\vec{p} \oplus f$ is

$$p_0 \oplus f(\bar{0}), \ldots, p_{2^n-1} \oplus f(\overline{2^n - 1})$$

and where of course each $f(\bar{i})$ is 0 or 1.

Theorem 14 *Under the above assumptions, $C^*(\vec{p})$ forms a quasipolynomial time/poly natural property against P/poly.*

Proof

(1) *Constructivity*
C^* has circuits of size $2^{(\log N)^{O(1)}}$ since C does.

(2) *Largeness*
For all γ, either $C^*(\gamma)$ or $C^*(\gamma \oplus f)$ holds. Therefore at least half of all Boolean functions satisfy C^*.

(3) *Usefulness*

(a) If $\neg C(\vec{p})$, i.e. $C(\vec{p}) = 0$, then $\gamma := \vec{p}$ does not have circuit of size t.

(b) If $C(\vec{p} \oplus f)$, i.e. $C(\vec{p} \oplus f) = 1$, then $(\vec{p} \oplus f) \oplus f = \vec{p} (= \gamma)$ does not have circuit of size t.

Therefore, if $C^*(\gamma)$, γ does not have a polynomial size circuit. This concludes the proof of Theorem 14 and Theorem 9. ∎

References

[BlMi84] M. Blum and S. Micali, How to generate cryptographically strong sequences of pseudo-random bits, *SIAM J. on Computing* **13**, 4 (1984), pp. 850-864.

[Bor77] A. Borodin, On relating time and space to size and depth, *SIAM J. on Computing* **6**, 4 (1977), pp. 733-744.

[Bus86] S. Buss. *Bounded Arithmetic*, volume 3 of *Studies in Proof Theory*. Bibliopolis, 1986. 221 pages.

[Bus90] S.R. Buss. Axiomatizations and conservation results for fragments of bounded arithmetic. In W. Sieg, editor, *Logic and Computation*, pages 57–84. American Mathematical Society, 1990. Contemporary Mathematics 106, Proceedings of a Workshop held at Carnegie Mellon University, June 30 - July 2, 1987.

[GGM86] O. Goldreich, S. Goldwasser and S. Micali, How to construct random functions, *J. of the Association for Computing Machinery* **33**, 4 (1986), pp. 792-807.

[Kra94] J. Krajíček. On Frege and extended Frege systems. In P. Clote and J. Remmel, editors, *Feasible Mathematics II*, pages 284—319. Birkhäuser, 1994.

[Kra95] J. Krajíček. *Bounded Arithmetic, Propositional Logic, and Complexity Theory*. Cambridge University Press, 1995.

[PW83] J. B. Paris and A. J. Wilkie. Counting problems in bounded arithmetic. In C. A. di Prisco, editor, *Methods in Mathematical Logic*, pages 317 – 340. Springer Verlag Lecture Notes in Mathematics, 1983. Proceedings of Logic Conference held in Caracas, 1983.

[Raz94] A.A. Razborov. Unprovability of lower bounds on circuit size in certain fragments of bounded arithmetic. *Izvestiya of the RAN*, 1994.

[RR94] A. Razborov and S. Rudich, Natural proofs, *Proc. of the 26th ACM Symp. on the Theory of Computing* (1994), pp. 204-312.

[Ruz81] W.L. Ruzzo. On uniform circuit complexity. *J. Comput. System Sci.*, 22:pp. 365–383, 1981.

[WP87] A.J. Wilkie and J.B. Paris. On the schema of induction for bounded arithmetic formulas. *Annals of Pure and Applied Logic*, 35:261 – 302, 1987.

Buss's Simplified ALOGTIME Algorithm for Boolean Sentence Evaluation

Notes by Toniann Pitassi

Tuesday Evening, March 7

1 Introduction

In this session, Sam presented a new, simpler $ALOGTIME$ algorithm for the Boolean sentence value problem (BSVP).

Definition 1.1. A boolean formula is defined inductively by: (1) A literal 1 (true), 0 (false) or $x_1, x_2, x_3, ...$ is a boolean formula; (2) If α and β are boolean formulas then so are $(\neg\alpha)$, $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$.

Definition 1.2. The boolean sentence value problem (BSVP) is the problem of, given a variable-free Boolean formula, determining if it evaluates to 1 (true). In other words, to recognize true Boolean sentences.

Definition 1.3. $|\alpha|$ is the number of symbols in the expression α.

Obviously, BSVP is in P, by any obvious algorithm for evaluating the formula. An interesting question, open for quite a long time, was to show that BSVP has polynomial-size formulas. Sam's motivation to solve this problem comes from a problem in logic. Cook showed that PV proves $CONS(EF)$, where $CONS(EF)$ is a formula expressing the partial consisency of Extended Frege– for all n, there is no n symbol Extended Frege proof of $0 = 1$. Cook also showed a relationship between PV proofs and polynomial-size Extended

Frege proofs. By this relationship, it follows that there exist polynomial-size Extended Frege proofs of $[[Con(EF)]]_n$, where $[[Con(EF)]]_n$ is a propositional formula, in n variables, expressing the fact that no n-symbol Extended Frege proof ends in 0 (false). A related question is whether or not there are polynomial-size Frege proofs of $[[Con(F)]]_n$ for all n. A natural way to show this is to show inductively that every formula in a Frege proof must be valid, and therefore, the final formula cannot be 0 (false). To carry this out formally, one needs a polynomial-size formula, $\phi(x_1, .., x_n)$ that defines the truth of formulas of at most n symbols, and this is exactly the boolean sentence value problem! Using Sam's algorithm for BSVP, he was able to show that $[[Con(F)]]_n$ does have polynomial-size Frege proofs. See [B2] for this proof, and consequences of this result.

In 1977, Nancy Lynch showed that BSVP is in $LOGSPACE$. Her algorithm used a depth-first evaluation, always evaluating longer subformulas first. This algorithm inspired Sam's first algorithm[B87], so we will describe it first.

Theorem 1.1 *(Lynch, 1977) BSVP* $\in$ $LOGSPACE$.

Proof To evaluate $(\alpha \wedge \beta)$ where $|\beta| > |\alpha|$, first evaluate β and save its value, and then evaluate α and combine the results. Then, if $|\alpha| = n$, the algorithm will never need to save more than $\log(n)$ intermediate results. To see this, note that if $\gamma_1, .., \gamma_k$ are subformulas for which intermediate results are saved, then

$$|\gamma_i| \geq \sum_{j=i+1}^{k} |\gamma_j|.$$

So, $\sum_{i=1}^{k} |\gamma_i| \geq 2^{k-1}$. Note that Lynch's algorithm works even when other connectives such as parity are present.

Next, we will present an algorithm due to Buss (1992) for the BSVP which works on formulas in which ANDs and ORs are alternating in layers and there are no negations. The algorithm scans the formula from left-to-right only and maintains two counters. If the formula has depth d, then only $2\lceil \log d \rceil + 1$ many bits of storage are needed. The algorithm is as follows.

Algorithm for BSVP with connectives $\wedge$ and $\vee$:

Input: a Boolean sentence ϕ.
Initialize: $\mathit{Ignore\_Flag} = 0$
$\mathit{Ignore\_Depth} = 0$
Input tape head at leftmost symbol of ϕ
Loop:
Read current symbol α from input tape
If $\mathit{Ignore\_Flag} = 1$
If $\alpha =$ "(", increment $\mathit{Ignore\_Depth}$ by 1
If $\alpha =$ ")"
decrement $\mathit{Ignore\_Depth}$ by 1
if $\mathit{Ignore\_Depth} = 0$, set $\mathit{Ignore\_Flag} = 0$
Endif
Endif
If $\mathit{Ignore\_Flag} = 0$
If $\alpha =$ "0" set $\mathit{Val} = 0$
If $\alpha =$ "1" set $\mathit{Val} = 1$
If $\alpha =$ "$\wedge$" and $\mathit{Val} = 0$, set $\mathit{Ignore\_Flag} = 1$
If $\alpha =$ "$\vee$" and $\mathit{Val} = 1$, set $\mathit{Ignore\_Flag} = 1$
Endif
Move input tape head right one square
Endloop
Output: Val

The above algorithm shows that the BSVP for formulas with connectives AND, OR and NOT can be evaluated in space $O(\log d)$ where d is the depth of the formula.

Definition 1.4. A formula is balanced if its depth is $O(\log n)$.

Theorem 1.2 *The balanced BSVP is in $SPACE(\log\log n)$.*

Later we'll discuss the following theorem.

Theorem 1.3 *The balanced BSVP is complete for $ALOGTIME$ under deterministic logtime reductions.*

2 Background on $ALOGTIME$

Definition 2.1. The class alternating logtime ($ALOGTIME$) is defined as follows. The basic underlying model is a multiptape Turing Machine, with two types of states, existential and universal states. The input tape is accessed randomly–that is, there is a special index tape onto which an address i can be written in binary, and each state of the Turing machine can read the i-th input symbol. With this convention, it is possible to access the entire input within $O(\log n)$ time. A language L is in $ALOGTIME$ if there exists a machine of the above type that runs for $O(\log n)$ steps on inputs of length n, and accepts exactly L. For a more formal description of $ALOGTIME$, see [B87], or [CKS].

Definition 2.2. Deterministic Logtime ($LOGTIME$) is defined to be the class of predicates computable in $O(\log n)$ time with no alternations, and $\Sigma_k - LOGTIME$ is the class of predicates computable in $O(\log n)$ time with k alternations of quantifiers, beginning with existential states. The Logtime hierarchy, LH is $\cup_k \Sigma_k - LOGTIME$.

Definition 2.3. Let R be a predicate on $\{0,1\}^*$. $R \in NC^k$ if and only if R has a family of polynomial-size, $O(\log^k n)$-depth, bounded fan-in circuits. $R \in AC^k$ if and only if R has a family of polynomial-size, $O(\log^k n)$-depth unbounded fan-in circuits.

There are several natural uniformity conditions that can be put on the complexity class NC^k. See [Ruzzo], and [Cook] for two standard definitions of uniform NC^k. Their definitions agree for $k > 1$, but appear to differ for $k = 1$. For Cook, uniform NC^1 equals $ALOGTIME$.

Thus, we have the following inclusions between the complexity classes that we have just defined:

$$LOGTIME \subset \Sigma_1 - LOGTIME \subset \Sigma_2 - LOGTIME \subset \cdots \subset$$

$$ALOGTIME = uniformNC^1 \subseteq LOGSPACE \subseteq NC^2 \subseteq \cdots \subseteq NC.$$

The fact that the logtime hierarchy is proper is due to Sipser.

2.1 Reductions

Definition 2.4. Let C be a complexity class. A function f is in C if and only if the predicate $A_f(c, i, x) =$"the ith symbol of $f(x)$ is c" is in C and f is of polynomial growth rate.

Definition 2.5. Let A and B be decision problems. A C-reduction of A to B is a function $f \in C$ such that for all x, $x \in A$ if and only if $f(x) \in B$.

If there is an $ALOGTIME$-reduction from A to B and if $B \in ALOGTIME$, the $A \in ALOGTIME$. Also, if there is a LH-reduction from A to B and if $B \in LH$, then $A \in LH$.

Definition 2.6. A is $ALOGTIME$-complete if and only if $A \in ALOGTIME$ and every decision problem in $ALOGTIME$ is LH-reducible to A.

Theorem 2.1 *(Main Theorem 1) $BSVP \in ALOGTIME$.*

Theorem 2.2 *(Main Theorem 2) For every $A \in ALOGTIME$, there is a deterministic logtime reduction of A to $BSVP$.*

Corollary 2.3 *$BSVP$ is $ALOGTIME$-complete.*

3 A first attempt to prove $BSVP \in ALOGTIME$

Theorem 3.1 *(Spira, 1971) Let A be a boolean formula of size $n = |A|$. There is an equivalent formula A^* of size $O(n^2)$ and depth $O(\log n)$.*

In 1974, Brent [Br] improved this to be A^* of size $O(n)$, and extended it to arithmetic formulas.

Proof We prove the theorem inductively by applying the following transformation. First, find a subformula B of A such that $1/3|A| \leq |B| \leq 2/3|A|$. Let C_0 be the formula A where B has been replaced by the constant 0 (false). Similarly, let C_1 be the formula A where B is replaced by 1 (true). Then $A^* = (B^* \wedge C_1^*) \vee (\neg B^* \wedge C_0^*)$, where B^*, C_0^* and C_1^* are obtained by recursively applying the same transformation.

The above "1/3 - 2/3 trick" gives a method of dividing a formula into two pieces of approximately the same size. This trick gives a basis for a two player game for determining the value of a boolean sentence, A. The two players are the Pebbler and the Challenger who claim that A has value 1 and 0, respectively.

The game is played as follows. In the first round, the pebbler places a pebble labelled 1 on the root node of the tree representation of the formula A. The Challenger then challenges this pebble value. Set $A_1 := A$. Upon entering the ith round ($i > 1$), there is a "scarred" subformula A_{i-1} such that: (1) the root of A_{i-1} has been pebbled with a truth value and that value has been challenged and (2) some subformulas of A_{i-1} have been removed (leaving scars), by being pebbled but not challenged. During round i, a (scarred) subformula B_{i-1} of A_{i-1} is chosen so that $1/3|A_{i-1}| \leq |B_{i-1}| \leq 2/3|A_{i-1}|$; the pebbler pebbles the root of B_{i-1} with a truth value. If this truth value is challenged by the Challenger, then set $A_i := B_{i-1}$. Otherwise A_i is set to A_{i-1} with subformula B_{i-1} removed (leaving a scar). The game ends when A_i has size at most 1. The winner of the game is the player who has not contradicted herself at the end of the game.

Theorem 3.2 *If A is true, the Pebbler has a winning strategy. Otherwise, the Challenger has a winning strategy.*

Proof If A is true, then the Pebbler always tells the truth. Obviously, the Pebbler will win because she never contradicts herself. Similarly, if A is false, then the Challenger always tells the truth, and thus wins. In otherwords, "honesty is the best policy."

The above game yields an alternating algorithm for the BSVP as follows. Step one: Simulate a playing of the game, where the Pebbler's moves are existentially chosen and the Challenger's moves are universally chosen. Step two: Determine who won the game and accept if and only if the Pebbler won. In step one, only two bits of choices are needed—one bit for the pebbled value and one bit indicating if the pebble was challenged. The game lasts only $\log_{3/2}|A|$ rounds, so step one is in $ALOGTIME$. But step two is harder to implement in $ALOGTIME$. The basic problem is that computing what A_i is and where the pebbles are after i rounds, as a function of the play of the game so far, is difficult to do.

Cook, Gupta [CG], and Ramachandran [R] have a variant of the above algorithm where they prove that BSVP is in $O(\log n \log\log n)$ alternating time.

4 $ALOGTIME$ algorithm for $BSVP$

In this section, a sketch of the ALOGTIME algorithm for the BSVP is given. Complete details can be found in [B93]

As above, the $ALOGTIME$ algorithm for $BSVP$ will be described as a pebbling game between the Pebbler and the Challenger. In this game, the Pebbler places pebbles on nodes of the boolean sentence: each pebble is labelled with either 0 or 1 (indicating that the Pebbler asserts this value for the subformula rooted at that node). After the Pebbler places some pebbles, the Challenger must respond by challenging one of the pebbled positions: this represents an assertion by the Challenger that the pebble value is incorrect. Furthermore, we will assume that if the Challenger challenges some position U, then all pebble positions below U are correctly labelled. The pebbling game is organized into rounds: during each round, the Pebbler pebbles a set of nodes and the Challenger either challenges a newly pebbled node, or rechallenges the previously challenged node. A player loses the game by asserting incompatible input and output values for a boolean gate. A player may also lose by violating the rules of the game. Below we will describe the exact rules for the pebbling game.

(1) Without loss of generality, a boolean sentence has leaves numbered from 1 to $2^{d+1}-1$, from left to right. (We can add dummy nodes if necessary to force this numbering.) The *rank* of a leaf numbered j is the maximum value i such that $2^i|j$.

(2) At round i, there are distinguished leaves L_i, C_i, R_i (left, center and right respectively) and distinguished nodes A_i and B_i (indicating above and below, respectively).

(3) The rank of C_i is at least $d-i$. The rank of L_i is equal to the rank of R_i which is equal to $d-i$.

(4) A_i and B_i are ancestors of C_i. B_i is the highest pebbled ancestor of C_i that has not yet been challenged. A_i is the lowest challenged ancestor of C_i. Informally, the players have agreed at B_i but disagree at A_i.

(5) Leaves below A_i but not below B_i are numbered in the ranges $(L_i - 2^{d-i}, L_i + 2^{d-i})$, $(R_i - 2^{d-i}, R_i + 2^{d-i})$. (See Figure 1 for a picture of nodes L_i, R_i, C_i, A_i, B_i.)

(6) At round 0, the Pebbler pebbles the root with 1, and the Challenger challenges the root node. In preparation for round 1, set A_1 equal to the root node, set $C_1 = B_1 =$ leaf of rank d (the leaf numbered 2^d); and set L_1 and R_1 to the leaves numbered 2^{d-1} and $2^d + 2^{d-1}$.

(7) During round $i \geq 1$, let U_i be the least common ancestor of L_i and C_i, and let V_i be the least common ancestor of C_i and R_i. Let $U_i^1, U_i^2, V_i^1, V_i^2$ be their children. (See Figure 2 for a picture of these nodes.) The pebbler must state whether U_i is an ancestor of V_i and must pebble U_i, U_i^1, U_i^2, V_i, V_i^1 and V_i^2. This requires seven bits. The Challenger must then challenge one of the pebbled nodes, $U_i, V_i, U_i^1, U_i^2, V_i^1, V_i^2, A_i$. (If U_i and V_i are not between A_i and B_i, the Challenger should rechallenge A_i.) It requires three bits for the Challenger to specify which of the pebbled nodes to challenge. For round $i+1$, the values of L_{i+1}, R_{i+1}, and C_{i+1} are set according to Tables 1 and 2. Note that A_{i+1} is the node challenged by the Challenger in round i, and B_{i+1} is the (unique) highest pebbled node below A_{i+1}.

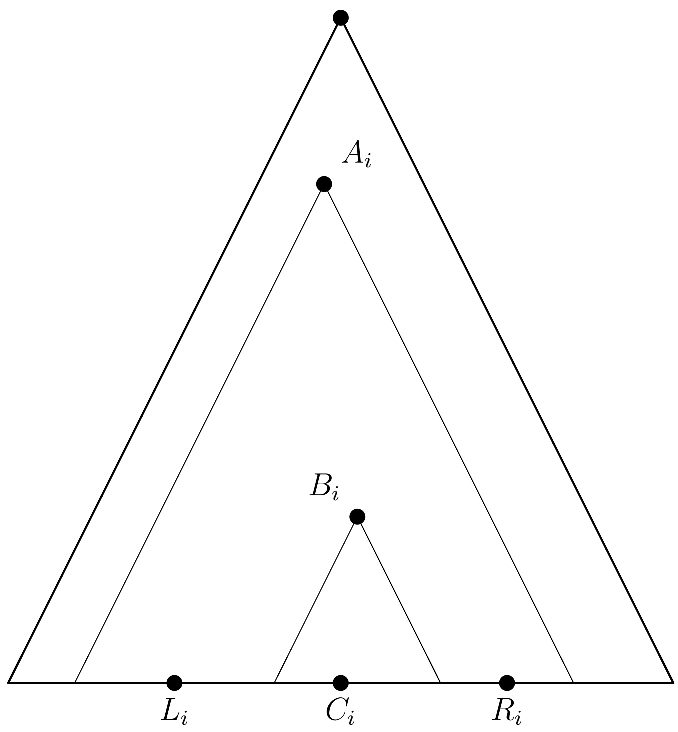

Figure 1

The triangles delineate subtrees. In general, it is not necessary for L_i and R_i to be descendents of A_i. On the other hand, it is also permissable for L_i and R_i to be descendents of B_i.

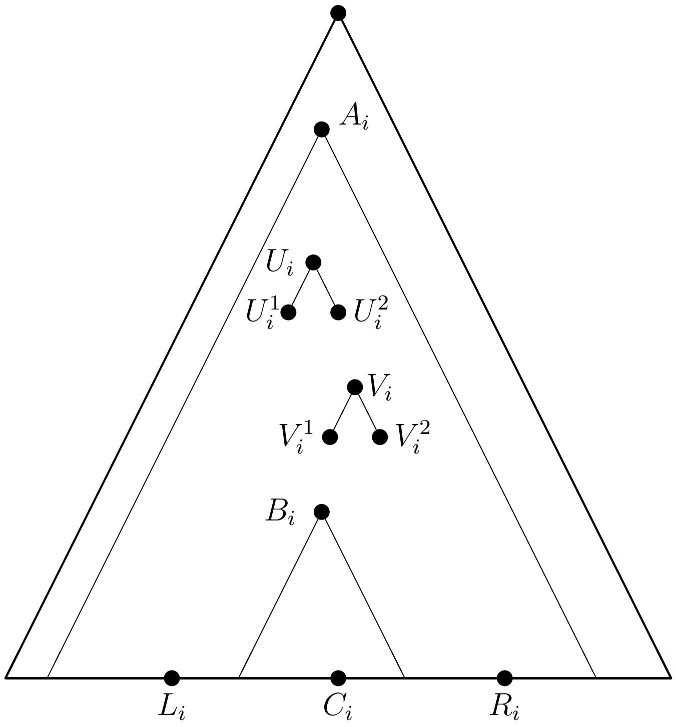

Figure 2

Figure 2 illustrates a possible configuration for the breakpoints chosen in round i. Handdrawn lines indicate paths in the tree.

Challenged Node	Pebbler says $U_i \rhd V_i$	Pebbler says $V_i \rhd U_i$
V_i^1	$C_{i+1} = C_i$ $R_{i+1} = R_i - 2^{\delta_i}$ $L_{i+1} = L_i + 2^{\delta_i}$	$C_{i+1} = C_i$ $R_{i+1} = R_i - 2^{\delta_i}$ $L_{i+1} = L_i - 2^{\delta_i}$
V_i^2	$C_{i+1} = R_i$ $R_{i+1} = R_i + 2^{\delta_i}$ $L_{i+1} = R_i - 2^{\delta_i}$	$C_{i+1} = R_i$ $R_{i+1} = R_i + 2^{\delta_i}$ $L_{i+1} = R_i - 2^{\delta_i}$
U_i^1	$C_{i+1} = L_i$ $R_{i+1} = L_i + 2^{\delta_i}$ $L_{i+1} = L_i - 2^{\delta_i}$	$C_{i+1} = L_i$ $R_{i+1} = L_i + 2^{\delta_i}$ $L_{i+1} = L_i - 2^{\delta_i}$
U_i^2	$C_{i+1} = C_i$ $R_{i+1} = R_i + 2^{\delta_i}$ $L_{i+1} = L_i + 2^{\delta_i}$	$C_{i+1} = C_i$ $R_{i+1} = R_i - 2^{\delta_i}$ $L_{i+1} = L_i + 2^{\delta_i}$
U_i or V_i	Game ends	Game ends

Challenged Node: A_i			
Pebbler says $A_i \rhd U_i, V_i$	Pebbler says $U_i, V_i \unrhd A_i$	Pebbler says $U_i \unrhd A_i \rhd V_i$	Pebbler says $V_i \unrhd A_i \rhd U_i$
$C_{i+1} = C_i$ $R_{i+1} = R_i + 2^{\delta_i}$ $L_{i+1} = L_i - 2^{\delta_i}$	$C_{i+1} = C_i$ $R_{i+1} = R_i - 2^{\delta_i}$ $L_{i+1} = L_i + 2^{\delta_i}$	$C_{i+1} = C_i$ $R_{i+1} = R_i + 2^{\delta_i}$ $L_{i+1} = L_i + 2^{\delta_i}$	$C_{i+1} = C_i$ $R_{i+1} = R_i - 2^{\delta_i}$ $L_{i+1} = L_i - 2^{\delta_i}$

Tables 1 and 2

Because there are at most 2^{d-i+1} leaves which are below A_i but not at or below B_i, it follows that one of the players will be forced into an "obvious mistake" before round d. We will now define who the winner of a play of the game is. L_i and R_i can be found in $ALOGTIME$. Similarly, C_i, A_i and B_i can be computed in $ALOGTIME$. (Least common ancestors can be found by counting parenthesis.) To determine the winner, we nondeterministically guess the first place in the game where an obvious mistake was made. This gives the loser of the game. The obvious mistakes that can cause a player to lose the game are as follows:

(a) Asserting incompatible values to inputs and outputs of a gate.

(b) Asserting an incorrect value of a leaf node.

(c) (Pebbler) Pebbling a node with both 0 and 1.

(d) (Pebbler) Being incorrect about whether U_i is above V_i, A_i is above U_i or A_i is above V_i.

(e) (Challenger) Challenging a node at or below a previously unchallenged pebble, or challenging a node above a previously challenged pebble.

The conversion of the pebbling game into an $ALOGTIME$ Turing machine is fairly straightforward. Given a boolean sentence ϕ as input, M computes $O(\log n)$ bits describing play of the game using existential guesses for the Pebbler's moves and universal choices for the Challenger's moves; M then accepts if and only if the Pebbler won this play of the game. Since the game lasts $O(\log n)$ rounds, and each round requires only a constant number of bits, it follows that M runs in time $O(\log n)$ time.

4.1 Some Extensions

(1) Parenthesis context-free languages are in $ALOGTIME$. [B87]

(2) Input driven languages are in $ALOGTIME$. [Dymond] Input driven languages are characterized by deterministic pushdown automata with pushing and popping determined by only the input symbol.

(3) There are log-depth arithmetic circuits for evaluating arithmetic formula over addition and multiplication, and over addition, multiplication and division. [BCGR]

5 Completeness of $BSVP$

In this section, we will sketch the proof of the previously stated theorem that the BSVP problem is complete for $ALOGTIME$, under deterministic logtime reductions.

The proof idea is as follows. Let M be an $ALOGTIME$ Turing machine. For a given input x, $M(x)$ is a tree, labelled as follows. Existential configurations will be labelled by $\vee$'s, universal configurations will be labelled by $\wedge$'s, accepting configurations will be labelled by 1's and rejecting configurations will be labelled by 0's. Thus, we will get a boolean formula with value 1 or 0 depending on whether $M(x)$ accepts or rejects. With some cleverness, it can be shown that this Boolean formula is computable in $ALOGTIME$ as a function of x (modulo assumptions on M).

6 Related Open Problems

One classic open problem is whether or not Division is in $ALOGTIME$. The best algorithm currently known is by Beame, Cook and Hoover [BCH].

Another open problem concerns Dyck languages. The Dyck language is the set of balanced formulas where the underlying symbols are two distinct types of parenthesis. Two-sided Dyck language consists words over the same alphabet, but now we can cancel in either direction. For example, the following word is in the two-sided Dyck language over $\{],[,(,)\}$: $])([][$.

It is an open problem to determine whether or not two-sided Dyck languages are in $ALOGTIME$. Another way to state this problem is whether or not the word problem for a k-generator free group with $k \geq 2$ is in $ALOGTIME$. Lipton and Zalcstein [LZ] showed that this problem is in $LOGSPACE$. In addition, this problem is known to be hard for $ALOGTIME$. However, it is not known to be in $ALOGTIME$, or complete for $LOGSPACE$. For related work and results in this area, see the UCSD thesis by Dave Robinson, where he studied the word problem for finitely generated infinite groups.

References

[B87] Buss, S. "The Boolean formula value problem is in ALOGTIME", in *Proceedings of the 19-th Annual ACM Symposium on Theory of*

Computing, May 1987, pp. 123-131.

[B93] Buss, S. "Algorithms for Boolean formula evaluation and for tree contraction," in *Arithmetic, Proof Theory and Computational Complexity*, Editors: P. Clote and J. Krajíček, Oxford University Press, 1993, pp. 96-115.

[Br] Brent, R. P., "The parallel evaluation of general arithmetic expressions," *Journal Assoc. Comput. Mach.*, v. 21, 1974, pp.201-206.

[BCGR] Buss, S., Cook, S.A., Gupta, A., and Ramachandran, V., "An optimal parallel algorithm for formula evaluation," *SIAM Journal on Computation* v. 21, 1992, pp.755-780.

[CKS] Chandra, A.K., Kozen, D.C., and Stockmeyer, L.J., "Alternation", *J. Assoc. Comput. Mach.*, v. 28, 1981, pp.114-133.

[LZ] Lipton, R.J., Zalcstein, Y., "Word problems solvable in logspace," *J. Assoc. Comput. Mach.*, v. 24, 1977, pp.522-526.

[Lynch] Lynch, N., "Log space recognition and translation of parenthesis languages," *J. Assoc. Comput. Mach.*, v. 1975, 1977, pp.583-590.

[Spira] Spira, P. M., "On time hardware complexity tradeoffs for Boolean functions," In *Proceedings of the Fourth Hawaii International Symposium on System Sciences*, 1971, pp.525-527.

Bounded Arithmetic

by Maria Luisa Bonet and Ran Raz

Wednesday Morning, March 8

The whole motivation of what follows is witnessing in the following sense: If $T \models \forall x \exists y (...)$, what can be said about the computational complexity of determining y, given x as an input?

First we will recall the definitions of the Polynomial Time Hierarchy.

1 The Polynomial Time Hierarchy

The following are the usual definitions of the polynomial time hierarchy.

P $= \triangle_1^P$ is the set of predicates on N which are recognized by a polynomial-time Turing machine.

$PF = \square_1^P$ is the set of functions on N which are computed by a polynomial-time Turing machine.

NP $= \Sigma_1^P$ is the set of predicates on N which are recognized by a non deterministic polynomial-time Turing machine.

$\Sigma_i^P =$ {predicates Q: for some $R \in \triangle_i^P$ and some polynomial q, $Q(\vec{x}) \iff (\exists y \leq 2^{q(|\vec{x}|)}) R(\vec{x}, y)$}, where $|x| = \lceil \log_2(x+1) \rceil$ is the length of the binary representation of x.

$\Pi_i^P =$ {predicates Q: for some $R \in \Sigma_i^P, Q(\vec{x}) \iff \neg R(\vec{x})$}

$\square_{i+1}^P$ is the polynomial time closure of Σ_i^P, i.e., the set of functions computable by a polynomial-time Turing machine with an oracle for a (complete) predicate in Σ_i^P.

$\triangle_{i+1}^P =$ {predicates Q: Q has characteristic function in $\square_{i+1}^P$}.

The boxes are the function analog of the triangle.

$$
\begin{array}{cccccc}
\vdots & & \vdots & & \vdots & \vdots \\
 & & \Delta_3^P & & & \Box_3^P \\
 & \subseteq & & \supseteq & & \\
\Pi_2^P & & & & \Sigma_2^P & \cup| \\
 & \supseteq & & \subseteq & & \\
 & & \Delta_2^P & & & \Box_2^P \\
 & \subseteq & & \supseteq & & \\
Co-NP=\Pi_1^P & & & & \Sigma_1^P=NP & \cup| \\
 & \supseteq & & \subseteq & & \\
 & & \Delta_1^P=P & & & \Box_2^P=PF
\end{array}
$$

The above classes comprise the Meyer-Stockmeyer polynomial time hierarchy. It is not known whether the inclusions are proper!

2 The Link to Mathematical Logic

We use the first order language of N with function symbols $0, S, +, \cdot, \sharp, |x|$ and $\lfloor \frac{1}{2}x \rfloor$, and predicate symbol $\leq$.

$$\lfloor \frac{1}{2}x \rfloor = \text{greatest integer} \leq \frac{x}{2}$$

$$x \sharp y = 2^{|x| \cdot |y|}$$

The $\sharp$ ("smash") function allows us to write terms $2^{q(|x|)}$ where q is any polynomial with nonnegative coefficients. Ed Nelson was the first one to define the smash function.

A bounded quantifier is a quantifier of the form $\forall x \leq t$ or $\exists x \leq t$, where t is a term. $\forall x \leq t$ is thought of as part of the syntax. A sharply bounded quantifier is a bounded quantifier of the form $\forall x \leq |t|$ or $\exists x \leq |t|$. Sharply bounded quantifiers are polynomial time feasible. You can check all values $\leq |t|$. $\forall x$ and $\exists x$ are unbounded quantifiers.

We define a hierarchy Σ_i^b and Π_i^b of bounded formulas by counting alternations of bounded quantifiers, ignoring sharply bounded quantifiers.

So $\Sigma_0^b = \Pi_0^b$ is the set of formulas with all quantifiers sharply bounded.

If $A \in \Sigma_i^b$ then $(\forall x \leq t)A$ is in Π_{i+1}^b, and $(\forall x \leq |t|)A$ and $(\exists x \leq t)A$ are in Σ_i^b.

Dually, if $A \in \Pi_i^b$ then $(\exists x \leq t)A$ is in Σ_{i+1}^b, and $(\exists x \leq |t|)A$ and $(\forall x \leq t)A$ are in Π_i^b.

$\wedge, \vee, \neg, \rightarrow$ are treated in the usual way.

Theorem 2.1 *Let $i \geq 1$. A predicate Q is in Σ_i^p iff there is a Σ_i^b- formula φ such that for all $\vec{n} \in N^k$,*

$$Q(\vec{n}) \Leftrightarrow N \models \varphi(\vec{n})$$

This is theorem is due essentially to Stockmeyer-Wrathall [Sto76, Wra76], and this exact form follows from a result of Kent-Hodgson [KH82].

3 Classical Theories of Bounded Arithmetic

The Σ_i^b-IND axioms are:

$$A(0) \wedge \forall x(A(x) \rightarrow A(Sx)) \rightarrow \forall x A(x)$$

where A is any Σ_i^b-formula.

Definition 3.1. T_2^i is the first order theory with language $0, S, +, \cdot, \sharp, \lfloor \frac{1}{2}x \rfloor, |x|, \leq$, and with axioms:

1. A finite set of open axioms defining simple properties of the function and relation symbols, called BASIC.
2. The Σ_i^b-IND axioms.

T_2 is $\bigcup_i T_2^i$.

The subscript 2 means that $\sharp$ is present.

Another approach to bounded arithmetic is theory $I\Delta_0$, introduced by R. Parikh, which has the symbols 0, S, $+$ and $\cdot$ only and has induction for all bounded formulas. This theory and an extension $I\Delta_0 + \Omega_1$ has been studied extensively by J. Paris, A. Wilkie and others. The additional axiom Ω_1 means essentially that the function $\sharp$ is total. Up to choice of language, we have that T_2 and $I\Delta_0 + \Omega_1$ are essentially the same theory.

The Σ_i^b-PIND axioms are:

$$A(0) \wedge \forall x(A(\lfloor \frac{1}{2}x \rfloor) \to A(x)) \to \forall x A(x)$$

(where A is any Σ_i^b-formula).

The Σ_i^b-LIND axioms are:

$$A(0) \wedge \forall x(A(x) \to A(Sx)) \to \forall x A(|x|)$$

(where A is any Σ_i^b-formula).

Both the PIND and LIND axioms are intuitively weaker than the regular induction axiom. The first has stronger assumptions for the same conclusion as IND. The second has the same assumptions as IND, but weaker conclusion. Since the length function is not total, LIND is not the same as IND.

Definition 3.2. S_2^i is the theory BASIC $+\Sigma_i^b$-PIND.

Theorem 3.1 ([Bus86]) *: $S_2^{i+1} \vdash \Sigma_i^b$-IND, for all $i \geq 0$.*

Theorem 3.2 ([Bus86], [BI95]) *: $S_2^i \equiv BASIC + \Sigma_i^b - LIND$, for all $i \geq 1$.*

Corollary 3.3 ([Bus86]) *:For $i \geq 1$, $T_2^i \vdash S_2^i$ and $S_2^{i+1} \vdash T_2^i$.*

Corollary 3.4 *$S_2 \equiv T_2$. (Recall these are essentially the same as $I\Delta_0 + \Omega_1$.)*

<u>Open</u>: Is the hierarchy below proper?

$$T_2^0 \neq S_2^1 \subseteq T_2^1 \subseteq S_2^2 \subseteq T_2^2 \subseteq \cdots$$

Remark: If the polynomial hierarchy collapses and T_2 proves it, then the hierarchy of theories of bounded arithmetic collapses.

Proof of theorem 3.1 ($S_2^{i+1} \vdash \Sigma_i^b$-IND. Hence $T_2^i \subseteq S_2^{i+1}$):

Let $A(x) \in \Sigma_i^b$. From now on we work in S_2^{i+1}.

Suppose $A(0) \wedge \forall x(A(x) \to A(Sx))$. Let $B(y)$ be:

$$\forall x \leq c(A(x \dot{-} y) \to A(x)).$$

So $B \in \Pi_{i+1}^b$.

<u>**Claim**</u>**(S_2^{i+1})**: $\forall y(B(\lfloor \frac{1}{2}y \rfloor) \to B(y))$.

Proof of the claim: Assume $B(\lfloor \frac{1}{2} y \rfloor)$. We will prove $B(y)$. $\forall x \leq c$, we have to prove $A(x \dot{-} y) \rightarrow A(x)$.

$$A(x \dot{-} y) \Rightarrow A(x \dot{-} 2\lfloor \frac{1}{2} y \rfloor) \Rightarrow A(x \dot{-} \lfloor \frac{1}{2} y \rfloor) \Rightarrow A(x)$$

$\square$ (of claim)

Now, by Π^b_{i+1}-PIND on B, $B(0) \rightarrow B(c)$. But $B(0)$ is obvious. Also $B(c) \rightarrow (A(0) \rightarrow A(c))$. Hence $A(c)$ holds. Since c is arbitrary, $\forall x A(x)$ is true. $\square$

4 The Main Theorem for S^i_2

Theorem 4.1 ([Bus86]) *: $(i \geq 1)$. Let A be a Σ^b_i-formula. Suppose that $S^i_2 \vdash \forall x \exists y A(x,y)$. Then there is a function $f \in PTC(\Sigma^P_{i-1}) = \square^P_i$, a formula $B \in \Sigma^b_i$ and a term t so that:*

1. *$S^i_2 \vdash \forall x \forall y (B(x,y) \rightarrow A(x,y))$*
2. *$S^i_2 \vdash \forall x \exists ! y B(x,y)$*
3. *$S^i_2 \vdash \forall x (\exists y \leq t) B(x,y)$ [due to Parikh]*
4. *For all x, $N \models B(x, f(x))$.*

Conversely, If $f \in PTC(\Sigma^P_{i-1})$ then there is a formula $B \in \Sigma^b_i$ and a term t so that (2), (3) and (4) hold.

Special Case: when $i = 1$, f is a polynomial time function.

Corollary 4.2 *: The functions which are definable in S^i_2 by Σ^b_i-formulas are precisely the functions in $PTC(\Sigma^P_{i-1}) = \square^P_i$.*

We can restate the main theorem using predicates instead of functions:

Theorem 4.3 ([Bus86]) *:$(i \geq 1)$. Suppose $A(x) \in \Sigma^b_i$ and $B(x) \in \Pi^b_i$ and $S^i_2 \vdash A \leftrightarrow B$. Then there is a predicate $Q \in \Delta^P_i$ so that for all $n \in N$,*

$$Q(n) \Leftrightarrow N \models A(n) \Leftrightarrow N \models B(n).$$

Conversely, if $Q \in \Delta^P_i$ then there are A and B so that the above holds.

Special Case: $(i = 1)$.

If $A(x)$ is a formula such that S_2^1 proves that A is equivalent to a Σ_1^b and a Π_1^b-formula (in other words, S_2^1 proves $A \in \mathrm{NP} \cap \mathrm{coNP}$), then $A(x)$ represents a predicate in P.

5 Proof of the Witnessing Theorem for S_2^i

5.1 Introduction and First Step

Use Gentzen's sequent calculus to express S_2^i-proofs.

A sequent calculus (LK) proof contains sequents:

$$A_1, \ldots, A_k \to B_1, \ldots, B_l$$

which mean:

$$A_1 \wedge \cdots \wedge A_k \to B_1 \vee \cdots \vee B_l$$

Initial sequents contain only atomic formulas and can be:

Logical axioms: $A \to A$ for example

Equality axioms

BASIC axioms: $x \leq y \to |x| \leq |y|$for example

Quantifier rules include:

$$\frac{\Gamma \to \Delta, A(t)}{\Gamma \to \Delta, \exists x A(x)}$$

and

$$\frac{\Gamma \to \Delta, A(b)}{\Gamma \to \Delta, \forall x A(x)}$$

(where b is an eigenvariable that doesn't appear in the lower sequent).

Bounded quantifier rules include:

$\exists \leq$: *right*:

$$\frac{\Gamma \to \Delta, A(s)}{s \leq t, \Gamma \to \Delta, (\exists x \leq t) A(x)}$$

$\forall \leq$: *right*:

$$\frac{b \leq t, \Gamma \to \Delta, A(b)}{\Gamma \to \Delta, (\forall x \leq t) A(x)}$$

(where b is an eigenvariable that doesn't appear in the lower sequent), and similarly $(\exists \leq: left)$ and $(\forall \leq: left)$ rules.

Instead of induction axioms, the sequent calculus, LK, uses induction rules:
Σ_i^b-IND will be:

$$\frac{\Gamma, F(b) \to F(S(b)), \Delta}{\Gamma, F(0) \to F(t), \Delta}$$

and Σ_i^b-PIND will be:

$$\frac{\Gamma, F(\lfloor \frac{1}{2} b \rfloor) \to F(b), \Delta}{\Gamma, F(0) \to F(t), \Delta}$$

where b is an eigenvariable and does not occur in the lower sequent.

Theorem 5.1 ([Gen69], [Tak87]) (free cut elimination theorem) *: If $\Gamma \to \Delta$ is S_2^i (or T_2^i) provable, then it has an $S_2^i(T_2^i)$ proof in which every cut formula is a Σ_i^b-formula.*

In particular, if every formula in $\Gamma \to \Delta$ is in Σ_i^b, then $\Gamma \to \Delta$ has a proof of $S_2^i(T_2^i)$, such that every formula appearing in the proof is in Σ_i^b.

STEPS OF THE PROOF OF THE MAIN THEOREM FOR S_2^i

STEP 1: We start with an S_2^i-proof P of $\exists y A(x,y)$. By cut elimination there is an S_2^i-proof P^* of $(\exists y \leq t) A(x,y)$ such that every formula in P^* is a Σ_i^b or Π_i^b formula.

STEP 2: Given the proof P^*, we can immediately get an algorithm to compute a function $f(x)$ such that for all $x \in N$,

$$A(x, f(x))$$

In other words, P^* is a "program" for computing $f(x)$. The proof of step 2 follows...

5.2 Second Step

Fix $i \geq 1$.

Definition 5.1. Let $A(\vec{a})$ be a Σ_i^b-formula, $\vec{a}$ includes all free variables of A. $Witness_A^{i,\vec{a}}(w, \vec{a})$ is a formula defined inductively as follows:

1. If $A \in \Sigma^b_{i-1} \cup \Pi^b_{i-1}$ then

$$Witness^{i,\vec{a}}_A(w,\vec{a}) \Leftrightarrow A(\vec{a})$$

2. If $A = B \wedge C$, then

$$Witness^{i,\vec{a}}_A(w,\vec{a}) \Leftrightarrow Witness^{i,\vec{a}}_B(\beta(1,w),\vec{a}) \wedge Witness^{i,\vec{a}}_C(\beta(2,w),\vec{a})$$

3. If $A = B \vee C$, then

$$Witness^{i,\vec{a}}_A(w,\vec{a}) \Leftrightarrow Witness^{i,\vec{a}}_B(\beta(1,w),\vec{a}) \vee Witness^{i,\vec{a}}_C(\beta(2,w),\vec{a})$$

4. If $A = (\exists x \leq t(\vec{a}))B(\vec{a},x)$, then

$$Witness^{i,\vec{a}}_A(w,\vec{a}) \Leftrightarrow Witness^{i,\vec{a},b}_{B(\vec{a},b)}(\beta(2,w),\vec{a},\beta(1,w)) wedge \beta(1,w) \leq t(\vec{a})$$

5. If $A = (\forall x \leq |t(\vec{a})|)B(\vec{a},x)$, then

$$Witness^{i,\vec{a}}_A(w,\vec{a}) \Leftrightarrow (\forall x \leq |t(\vec{a})|) Witness^{i,\vec{a},b}_{B(\vec{a},b)}(\beta(x+1,w),\vec{a},x)$$

6. If $A = \neg B$, use prenex operations to put A in the above forms.

Proposition 5.2 *1. $S^i_2 \vdash A(\vec{a}) \leftrightarrow (\exists w \leq t_A) Witness^{i,\vec{a}}_A(w,\vec{a})$ for some term t_A*

2. $Witness^{i,\vec{a}}_A(w,\vec{a}) \in \Delta^P_i$ ($= P$ when $i = 1$)

3. $Witness^{i,\vec{a}}_A$ is provably equivalent to a Σ^b_i and a Π^b_i formula, by the theory S^i_2.

The Witnessing Theorem follows now from the following lemma.

Lemma 5.3 (main): *Suppose $S^i_2 \vdash \Gamma \rightarrow \Delta$ where Γ and Δ contain only Σ^b_i formulas. Let $\vec{c}$ be the free variables of Γ, Δ. Then there is a function f such that:*

1. f is Σ^b_i defined by S^i_2

2. $S_2^i \vdash Witness^{i,\vec{c}}_{\bigwedge\Gamma}(w,\vec{c}) \to Witness^{i,\vec{c}}_{\bigvee\Delta}(f(w,\vec{c}),\vec{c})$

3. $f \in PTC(\Sigma^P_{i-1}) = \square^P_i$ *(this is polynomial time when $i=1$)*

Proof : By induction on the number of inferences in a free-cut free S_2^i-proof of $\Gamma \to \Delta$.

The free cut free proof contains only Σ^b_i-formulas and it explicitly contains an algorithm for computing f. As an example of how to prove the main lemma, suppose that P is a free cut free proof and the last inference in P is:

$$\frac{A(\lfloor \frac{1}{2}a \rfloor) \to A(a)}{A(0) \to A(b)}$$

By the induction hypothesis, there is a function g such that

1. g is Σ^b_i-defined by S_2^i
2. g is in PTC(Σ^P_{i-1}) (this is P when $i=1$)
3. $S_2^i \vdash Witness^{i,\vec{a},\vec{c}}_{A(\lfloor 1/2a \rfloor)}(w,a,\vec{c}) \to Witness^{i,a,\vec{c}}_{A(a)}(g(w,a,\vec{c}),a,\vec{c})$
4. $S_2^i \vdash \forall a \forall \vec{c}\,[g(w,a,\vec{c}) \le t_A(a,\vec{c})]$

Define f to be the function such that

$$\begin{array}{rcl} f(w,0,\vec{c}) & = & w \\ f(w,b,\vec{c}) & = & g(f(w,\lfloor \frac{1}{2}b \rfloor,\vec{c}),b,\vec{c}) \quad \text{if } b \neq 0 \end{array}$$

<u>**Claim**</u>:

1. $f \in$ PTC(Σ^P_{i-1}) ($= P$ when $i=1$).
2. S_2^i can Σ^b_i-define f and prove that f satisfies the conditions of the lemma.
3. $S_2^i \vdash Witness^{i,\vec{c}}_{A(0)}(w,\vec{c}) \to Witness^{i,b,\vec{c}}_{A(b)}(f(w,b,\vec{c}),b,\vec{c})$

(1) is true since f is defined form g by a limited recursion operation. (3) is true since $Witness^{i,a,\vec{c}}_{A(a)}$ is a Σ^b_i-formula, and S_2^i can prove this by Σ^b_i-PIND directly from the induction hypothesis of (3). (2) can be proved in S_2^i by Σ^b_i-PIND. $\square$

5.3 Some Comments

Theorem 5.4 ([Bus90]) *: $(i \geq 1)$. S_2^{i+1} is $\forall\Sigma_{i+1}^b$ - conservative over T_2^i.*

Proof idea: First show that T_2^i can Σ_{i+1}^b-define all the $\square_{i+1}^P$-functions (same as S_2^{i+1}).

Second, reprove the S_2^{i+1}-witnessing lemma, but now f is Σ_{i+1}^b-defined by T_2^i, and T_2^i proves

$$Witness_{\bigwedge\Gamma}^{i+1,\vec{c}}(w,\vec{c}) \rightarrow Witness_{\bigvee\Delta}^{i+1,\vec{c}}(f(w,\vec{c}),\vec{c})$$

□

Theorem 5.5 ([Bus86]) *:($i = 0$ case). $S_2^1(PV)$ is $\forall\Sigma_i^b$-conservative over PV.*

Recall that PV is an equational theory of polynomial time functions, introduced by [Coo75].

6 Bounded Arithmetic Hierarchy

$$\begin{array}{rcl}
T_2 & = & S_2 \\
& \vdots & \\
& \supseteq & \\
& & S_2^4 \\
& \preceq & \\
T_2^3 & & \\
& \supseteq & \\
& & S_2^3 \\
& \preceq & \\
T_2^2 & & \\
& \supseteq & \\
& & S_2^2 \\
& \preceq & \\
T_2^1 & & \\
& \supseteq & \\
& & S_2^1 \\
& \preceq & \\
T_2^0(PV) = PV & &
\end{array}$$

In the figure above, $\preceq$ means: "conservatively extended".

7 Bounded Arithmetic and Constant Depth Frege Proofs

Paris-Wilkie [PW81] developed the following connection between provability in $I\Delta_0$ (or $I\Delta_0 + \Omega \equiv S_2$) and lengths of proofs of constant depth Frege proofs:

Allow extra predicate symbols α and function symbols f, which may be used in induction axioms. Translate closed first-order bounded formulas into propositional logic as follows:*

(a): $(f(i) = j)^{PW}$ is $p_{i,j}$ a propositonal variable.

(b): $\alpha(i)^{PW}$ is q_i a propositional variable.

(c): other atomic sentences become $\top$ or $\bot$, according to whether they are true or false.

(d): Boolean connectives are unchanged.

(e): $[(\forall x \leq t)A(x)]^{PW}$ is $\bigwedge_{i=0}^{value(t)}[A(i)]^{PW}$

(f): $[(\exists x \leq t)A(x)]^{PW}$ is $\bigvee_{i=0}^{value(t)}[A(i)]^{PW}$

Theorem 7.1 ([PW81]) *: If $I\Delta_0(f, \alpha) \vdash \forall x A(x, f, \alpha)$, with A a bounded formula, then the tautologies $[A(i, f, \alpha)]^{PW}$ have polynomial size, constant depth Frege proofs.*

Theorem 7.2 ([PW81]) *: Same holds for $I\Delta_0 + \Omega_1(f, \alpha)$, but with quasipolynomial ($= 2^{\log n^{O(1)}}$) size Frege proofs.*

Proof idea : Translate a bounded proof of $A(b, f, \alpha)$ into propositional logic, by applying the PW-transformation to its formulas. Induction inferences are "unwound" into a series of cut inferences.

*A "closed" formula is a formula with no free variables, i.e., a sentence.

Theorem 7.3 ([PWW88]) *: $I\Delta_0 + \Omega_1(f) \vdash (\forall x > 0)WPHP(x, f)$, where $WPHP(x, f)$ = "f is not a one-to-one map from $[2x]$ to $[x]$" = $(\exists y \leq 2x)(f(y) > x) \vee (\exists y, z \leq 2x)(y \neq z \wedge f(y) = f(z))$ (a Σ_1^b formula).*

Corollary 7.4 *The tautologies PHP_n^{2n} have quasi-polynomial size constant-depth Frege proofs.*

Theorem 7.5 ([Ajt88]) *The tautologies PHP_n do not have polynomial-size, constant-depth Frege proofs. Hence $I\Delta_0(f) \nvdash (\forall x)PHP(x, f)$.*

Theorem 7.6 ([BIK+92]) *The tautologies PHP_n do not have quasipolynomial-size, constant-depth Frege proofs. Hence $S_2(f) \nvdash (\forall x)PHP(x, f)$.*

8 Bounded Arithmetic and Extended Frege Proofs

Cook [Coo75] defined an equational theory of polynomial time computable functions (PV). Let f, g be polynomial time computable functions.

$\|f(x) = g(x)\|_{|x|=n}$ is a propositional formula with variables $x_{n-1}, \ldots, x_0$ representing the bits of x, and variables $\vec{y}$ representing the computations of $f(x)$ and $g(x)$.

$\|f(x) = g(x)\|_{|x|=n}$ says: "if $\vec{y}$ correctly represents computations of $f(x)$ and $g(x)$, then $f(x) = g(x)$".

$\|R(x)\|_{|x|=n}$ is defined similarly, for R a polynomial time predicate.

Theorem 8.1 ([Coo75]) *: If $PV \vdash \forall x(f(x) = g(x))$, then the tautologies $\|f(x) = g(x)\|_{|x|=n}$, $n = 0, 1, 2, \ldots$ have polynomial size extended Frege proofs.*

Application:
PV (and S_2^1) can prove: there is no integer w which codes a 1 to 1 map from $[n]$ to $[n-1]$. Thus the tautologies PHP_n have polynomial size extended-Frege proofs.

Theorem 8.2 ([Coo75]) *: If f is a propositional proof system, and f incorporates a Frege-system then if $PV \vdash Con(f)$ then extended-Frege p-simulates f.*

Theorem 8.3 ([Dow79]) *: $PV \vdash Con$(substitution-Frege). Hence extended-Frege p-simulates substitution-Frege.*

9 Bounded Arithmetic Relativized Separations

Theorem 9.1 ([BK94]) *: For $i \geq 1$, $S_2^i(\alpha) \neq T_2^i(\alpha)$.*

For T_2^i versus S_2^{i+1} even better results are known:

Theorem 9.2 ([KPT91]) *: For $i \geq 0$, if $T_2^i = S_2^{i+1}$ then the polynomial time hierarchy collapses to $\Sigma_{i+2}^P = \Pi_{i+2}^P = \triangle_{i+1}^P/poly$.*

Their proof relativizes, so also:

Theorem 9.3 ([KPT91]) *: For $i \geq 0$, $T_2^i(\alpha) \neq S_2^{i+1}(\alpha)$*

Recently, an improvement was given:

Theorem 9.4 ([Bus95], [Zam]) *: For $i \geq 0$, if $T_2^i = S_2^{i+1}$ then*

- $T_2^i = S_2$
- *T_2^i proves that the polynomial time hierarchy collapses to $\Sigma_{i+3}^P = \Pi_{i+3}^P = \Sigma_{i+1}^P/poly$*
- *There is a constant k such that T_2^i can prove every polynomial time hierarchy predicate that can be expressed as a Boolean combination of k Σ_{i+2}^P predicates.*

References

[Ajt88] M. Ajtai. The complexity of the pigeonhole principle. In *Proceedings of the 29-th Annual IEEE Symposium on Foundations of Computer Science*, pages 346–355, 1988.

[BI95] Samuel R. Buss and Aleksandar Ignjatović. Unprovability of consistency statements in fragments of bounded arithmetic. *Annals of Pure and Applied Logic*, 74:221-244, 1995.

[BIK$^+$92] P. Beame, R. Impagliazzo, J. Krajíček, T. Pitassi, P. Pudlák, and A. Woods. Exponential lower bounds for the pigeonhole principle. In *Proceedings of the 24-th Annual ACM Symposium on Theory of Computing*, pages 200–220, 1992.

[BK94] Samuel R. Buss and Jan Krajíček. An application of Boolean complexity to separation problems in bounded arithmetic. *Proc. London Math. Society*, 69:1–21, 1994.

[Bus86] Samuel R. Buss. *Bounded Arithmetic.* Bibliopolis, 1986. Revision of 1985 Princeton University Ph.D. thesis.

[Bus90] Samuel R. Buss. Axiomatizations and conservation results for fragments of bounded arithmetic. In *Logic and Computation, proceedings of a Workshop held Carnegie-Mellon University, 1987*, volume 106 of *Contemporary Mathematics*, pages 57–84. American Mathematical Society, 1990.

[Bus95] Samuel R. Buss. Relating the bounded arithmetic and polynomial-time hierarchies. *Annals of Pure and Applied Logic*, 75:67–77, 1995.

[Coo75] Stephen A. Cook. Feasibly constructive proofs and the propositional calculus. In *Proceedings of the 7-th Annual ACM Symposium on Theory of Computing*, pages 83–97, 1975.

[Dow79] Martin Dowd. *Propositional Representation of Arithmetic Proofs.* PhD thesis, University of Toronto, 1979.

[Gen69] Gerhard Gentzen. *Collected Papers of Gerhard Gentzen.* North-Holland, 1969. Editted by M. E. Szabo.

[KH82] Clarence F. Kent and Bernard R. Hodgson. An arithmetic characterization of NP. *Theoretical Computer Science*, 21:255–267, 1982.

[KPT91] Jan Krajíček, Pavel Pudlák, and Gaisi Takeuti. Bounded arithmetic and the polynomial hierarchy. *Annals of Pure and Applied Logic*, 52:143–153, 1991.

[PW81] J. B. Paris and A. J. Wilkie. Δ_0 sets and induction. In W. Guzicki, W. Marek, A. Pelc, and C. Rauszer, editors, *Open Days in Model Theory and Set Theory*, pages 237–248, 1981.

[PWW88] J. B. Paris, A. J. Wilkie, and A. R. Woods. Provability of the pigeonhole principle and the existence of infinitely many primes. *Journal of Symbolic Logic*, 53:1235–1244, 1988.

[Sto76] Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1976.

[Tak87] Gaisi Takeuti. *Proof Theory*. North-Holland, Amsterdam, 2nd edition, 1987.

[Wra76] Celia Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3:23–33, 1976.

[Zam] Domenico Zambella. Notes on polynomially bounded arithmetic. To appear in *J. Symb. Logic*.

Cutting Planes Proof Systems

Notes by Clemens Lautemann and Carlos Zamora-Cura

Wednesday Evening, March 8

1 Sam's talk about CP systems.

In this talk Sam Buss presented the basic ideas about *Cutting Plane Proof Systems*. The cutting plane refutation system *CP* is an extension of resolution, where unsatisfiable propositional logic formulas in conjuctive normal form are recognized by showing the non-existence of boolean solutions to associated families of linear inequalities. Sam spoke about the equivalence between CP and its subsystem CP_2; and its relation with Frege Systems.

1.1 Preliminaries

The cutting planes system CP is a Refutation System for propositional logic formulas in conjuctive normal form (CNF). In CP the truth values TRUE and FALSE are interpreted by 1 and 0, and propositional formulas are expressed by systems of linear inequalities. The basic idea is that a clause $\{x_1, \ldots, x_k\}$ can be rewritten as an integer inequality $i_1 + \cdots + i_k \geq 1$; where $i_j := x_j$ if x_j is a positive literal, and $i_j := 1 - x_j$ otherwise. For example take the following CNF:

$$(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$$

is represented by the family

$$\begin{array}{rcl} x + y & \geq & 1 \\ 1 - x + y & \geq & 1 \\ x + 1 - y & \geq & 1 \\ 1 - x + 1 - y & \geq & 1 \end{array}$$

of linear inequalities, one for each clause.

We can simplify the last three of these inequalities to

$$\begin{aligned} -x+y &\geq 0 \\ x-y &\geq 0 \\ -x-y &\geq -1 \end{aligned}$$

More generally, for a clause C define $I(C)$ to be the inequality

$$\sum_i \alpha_i p_i \geq 1-m,$$

where

$$\alpha_i = \begin{cases} 1 & if \quad x_i \in C \\ -1 & if \quad \neg x_i \in C \\ 0 & if \quad neither\ in\ C \end{cases}$$

and m is equal to the number of negated variables in C.

The idea of CP proofs rests on the following fact:

Fact 1 *A set* Γ *of clauses is satisfiable if and only if the set of integer inequalities* $\{I_C : C \in \Gamma\}$ *is satisfiable, over* $\{0,1\}$.

More formally, we state that a line in a cutting plane proof is of the form:

$$\alpha_1 p_1 + \alpha_2 p_2 + \cdots + \alpha_k p_k \geq m$$

where $\alpha_i, m \in \mathbf{Z}$, and $p_1, \ldots, p_k$ are variables valued in the set $\{0,1\}$. We will call $\alpha_1 p_1 + \alpha_2 p_2 + \cdots + \alpha_k p_k$ a *CP expression.*

The system CP has as axioms:

$$\begin{aligned} p &\geq 0 \\ -p &\geq -1; \end{aligned}$$

and as rules of inference:

- *Addition*:

$$\frac{E \geq a \quad , \quad F \geq b}{E + F \geq a + b,}$$

- *Multiplication* by $c \in \mathbf{N}$:

$$\frac{E \geq b}{c \cdot E \geq c \cdot b,}$$

- *Division* by $c \in \mathbf{N}$, $c > 0$, $b \in \mathbf{Z}$, if $c \mid E$ with quotient E', then

$$\frac{E \geq b}{E' \geq \left\lceil \frac{b}{c} \right\rceil,}$$

A formula B in conjuctive normal form has a *cutting plane refutation*, if there is a sequence $s_0, \ldots, s_m$ of linear inequalities, such that

- s_m is $0 \geq 1$,
- for all $i \leq m$, either s_i is a cutting plane axiom, or it is the translation of one of the clauses of B, or there exist $j, k < i$ such that s_i is obtained form s_j, s_k by one of the rules of inference.

A formula B is said to have a *cutting planes proof* if its negation has a cutting planes refutation.

1.2 Power of *CP*.

One of the central problems of propositional logic is to find useful methods for recognizing tautologies; since A is a tautology if and only if $\neg A$ is not satisfiable this is essentially the same as the problem of finding methods for recognizing satisfiable formulas. Three of the principal propositional proof systems are Frege Proof Systems, the Sequent Calculus System, and the Resolution Refutation Proof System. Therefore, in order to measure the power of CP it is important to compare how CP is related to some of these proof systems.

One sign of the importance and strength of cutting planes proofs is given by the following theorem proved by Cook, Coullard and Turan in [Cook-Coullard-Turan], which shows how CP can p-simulate resolution.

Theorem 1.1 *The cutting planes proof system can* p-simulate *resolution.*

Proof. We will only sketch the proof. Given a refutation in resolution of a set Γ of clauses, translate it into a cutting planes refutation of $\{I_C\}_{C\in\Gamma}$.

A resolution inference[1]

$$\frac{C_1\ ,\ C_2}{C},$$

solving on some variable x, is simulated by the following (we can assume without loss of generality that $C_1 \setminus \{x\} = C$ and $C_2 \setminus \{\neg x\} = C$):

$$\frac{\sum \alpha_i p_i + x \geq 1 - m\ ,\ \sum \alpha_i p_i - x \geq 1 - m - 1}{\sum 2\alpha_i p_i \geq 2 - 2m - 1}$$

$$\frac{\sum 2\alpha_i p_i \geq 2 - 2m - 1}{\sum \alpha_i p_i \geq 1 - m}$$

where we are using addition, and division rules.

So, each application of a resolution inference rule can be p-simulated, because the transformation function is polynomially time computable. □

It is perhaps not surprising that *CP* is more "efficient" than resolution, since addition of two inequalities may amount to a simultaneous resolution on many variables. To quantify the efficiency of *CP* over resolution, consider a combinatorial principle called the pigeonhole principle PHP_n; which states that there is no injection from $\{0,\ldots,n\}$ into $\{0,\ldots,n-1\}$:

$$\bigwedge_{0\leq i\leq n}\ \bigvee_{0\leq j<n} p_{ij} \supset \bigvee_{0\leq i<i'\leq n}\ \bigvee_{0\leq j<n} (p_{ij} \wedge p_{i'j})$$

The negation of PHP_n can be formulated by a propositional CNF formula denoted $\neg PHP_n$ of size $O(n^3)$. We will measure the *size* of a proof by the total number of logical connectives.

[1]Suppose that C and D are clauses and that $x \in C$ and $\neg x \in D$ are literals. The *resolution rule* applied to C and D is the inference

$$\frac{C\ ,\ D}{(C \setminus \{x\}) \cup (D \setminus \{\neg x\}).}$$

Theorem 1.2 *PHP_n has polysize cutting plane.*

Proof. The $\neg PHP_n$ clauses become the inequalities:

$$\begin{aligned} \alpha_i &: \; p_{i,0} + \cdots + p_{i,n-1} \geq 1 \quad , \quad i = 0, \ldots, n \\ \beta_{i,m,j} &: \; -p_{i,j} - p_{m,j} \geq -1 \quad , \quad 0 \leq i < m \leq n \quad \textit{and} \quad 0 \leq j < n \end{aligned}$$

Now, firstly we derive inductively on k for each fixed j:

$$-p_{0,j} - p_{1,j} - \cdots - p_{k,j} \geq -1 \quad , \, k = 1, \ldots, n.$$

Base case: $\beta_{0,1,j}$ is what we want. Inductive step:

$$\sum_{i=0}^{k-1} \beta_{i,k,j} = -p_{0,j} - \cdots - p_{k-1,j} - kp_{k,j} \geq -k$$

this plus $[-p_{0,j} - \cdots - p_{k-1,j} \geq -1](k-1)$ and combined with the previous step we have

$$-kp_{0,j} - kp_{1,j} - kp_{1,j} - \cdots - kp_{k-1,j} - kp_{k,j} \geq -2k + 1.$$

Dividing by k gives:

$$-p_{0,j} - \cdots - p_{k,j} \geq \left\lceil \frac{-2k+1}{k} \right\rceil = -1$$

Summing all these formulas for all n values on j gives (with $k = n$):

$$\sum_{i,j} -p_{i,j} \geq -n,$$

and summing α_i for all $n+1$ values of i gives

$$\sum_{i,j} p_{i,j} \geq n + 1.$$

Finally, summing the two very last inequalities we have

$$0 \geq 1.$$

□

Another theorem, which we will leave without proof, proved by Goerdt in [Goerdt] states a relation between Frege Proof Systems and CP.

Theorem 1.3 *Frege Proof Systems F can* p-*simulate the cutting planes proof systems.*

1.3 CP_k and CP.

For an integer $k \geq 2$, the proof system CP_k is obtained from CP by restricting the division rule to division by k. The system CP_2 is quite strong, and the following theorem will show that CP_2 is p-equivalent to CP.

Theorem 1.4 *For $k \geq 2$, CP_k* p-*simulates CP.*

Proof. Here, we only present the case $k = 2$, all other cases are similar. Suppose CP_2 is trying to simulate

$$\frac{m \cdot \alpha \geq n}{\alpha \geq \left\lceil \frac{n}{m} \right\rceil},$$

where α is a linear combination of variables. Let $2^{p-1} < m \leq 2^p$. Letting r_0 be equal to the sum of negative coefficients in α, and using addition of axioms, we get $\alpha \geq r_0$. Iterate:

- from $\alpha \geq r_i$ derive

  $$\frac{(2^p - m)\alpha \geq (2^p - m)r_i}{2^p\alpha \geq n + (2^p - m)r_i,}$$

  here we are using addition with $m \cdot \alpha \geq n$,

  $$\frac{2^p\alpha \geq n + (2^p - m)r_i}{\alpha \geq \left\lceil \frac{n+(2^p-m)r_i}{2^p} \right\rceil},$$

  here we are using division by 2, p times.

- set

  $$r_{i+1} = \left\lceil \frac{n + (2^p - m)r_i}{2^p} \right\rceil.$$

Note that

$$r_{i+1} \geq \frac{n}{m}\left(\frac{m}{2^p}\right) + r_i\left(\frac{2^p - m}{2^p}\right).$$

So, after polynomially many iterations we will have that

$$r_{i+1} > \frac{n}{m} - \frac{1}{m}.$$

Thus, $r_{i+1} = \left\lceil \frac{n}{m} \right\rceil$. □

The most interesting open question concerning CP is to exhibit a combinatorial family of tautologies requiring superpolynomial CP proof size.

2 Lower bound on the size of CP proofs with small coefficients.

In this talk, Ran Raz presented a new paper of his with Maria Bonet and Toni Pitassi, in which they proved an exponential lower bound for the length of CP proofs for the k–clique tautology, under the restriction that all coefficients in the proof be polynomially small in n. The proof is a reduction to monotone circuits that separate k–cliques from $(k-1)$–cocliques, which by a result of Alon and Boppana require exponential size.

2.1 The k-clique tautology

This tautology expresses the fact that a $(k-1)$-coclique (i.e., a complete $(k-1)$-partite graph) on n vertices cannot contain a k-clique as a subgraph. In order to represent this fact by a propositional formula, we will use propositional variables x_{ij}, $1 \leq i \leq k$, $1 \leq j \leq n$ and y_{ij}, $1 \leq i \leq k-1$, $1 \leq j \leq n$. We use truth assignments to the x_{ij} as encodings of k-cliques and truth assignments to the y_{ij} as encodings of $(k-1)$-cocliques: A clique on k vertices from $\{1, \ldots, n\}$ will be encoded by ordering its vertices in some way as $v_1, \ldots, v_k$ and assigning 1 to x_{ij} iff $v_i = j$. Similarly, the classes of a $(k-1)$-coclique are ordered, and we assign 1 to y_{ij} iff vertex j belongs to class i. Note that these representations are not unique, however, every k-clique and every $(k-1)$-coclique can be represented. The k-clique tautology now states "if x is a clique and y is a $(k-1)$-coclique then some class of y must contain two vertices of x." Its negation can be written as the conjunction of the following propositional formulas (which will also be given as CP-inequalities):

1. $\mathfrak{x}_{i1} \vee \cdots \vee \mathfrak{x}_{in}$ $\qquad \sum_{j=1}^{n} \mathfrak{x}_{ij} \geq 1$
   $1 \leq i \leq k$
2. $\neg(\mathfrak{x}_{ij} \wedge \mathfrak{x}_{ij'})$ $\qquad -\mathfrak{x}_{ij} - \mathfrak{x}_{ij'} \geq -1$
   $1 \leq i \leq k,\ 1 \leq j < j' \leq n$
3. $\neg(\mathfrak{x}_{ij} \wedge \mathfrak{x}_{i'j})$ $\qquad -\mathfrak{x}_{ij} - \mathfrak{x}_{i'j} \geq -1$
   $1 \leq i < i' \leq k,\ 1 \leq j \leq n$
4. $\mathfrak{y}_{1j} \vee \cdots \vee \mathfrak{y}_{(k-1)j}$ $\qquad \sum_{i=1}^{k-1} \mathfrak{y}_{ij} \geq 1$
   $1 \leq j \leq n$
5. $\neg(\mathfrak{y}_{ij} \wedge \mathfrak{y}_{i'j})$ $\qquad -\mathfrak{y}_{ij} - \mathfrak{y}_{i'j} \geq -1$
   $1 \leq i < i' \leq k-1,\ 1 \leq j \leq n$
6. $\mathfrak{x}_{ij} \wedge \mathfrak{x}_{i'j'} \rightarrow \neg(\mathfrak{y}_{lj} \wedge \mathfrak{y}_{lj'})$ $\qquad -\mathfrak{x}_{ij} - \mathfrak{x}_{i'j'} - \mathfrak{y}_{lj} - \mathfrak{y}_{lj'} \geq -3$
   $1 \leq i,\ i' \leq k,\ 1 \leq j < j' \leq n,\ 1 \leq l \leq k-1$

The formulas of type 1 - 3 and of type 4, 5 state that x is a k-clique and y is a $(k-1)$-coclique, respectively[2]. The conjunction of all formulas of type 6 stipulates that any two clique vertices belong to different classes of the coclique, i.e., that the clique is a subgraph of the coclique.

2.2 The construction of the circuit

Let a CP-proof for the clique tautology, i.e., a CP refutation of the formulas above, be given. Each line L of this proof is of the form

$$\sum_{ij} \alpha_{ij}^{L} x_{ij} + \sum_{ij} \beta_{ij}^{L} y_{ij} \geq \gamma^{L}$$

.

We will use $k \times n$ matrices over $\{0, 1\}$ to represent truth assignments to x and $(k-1) \times n$ matrices to represent truth assignments to y. We will say that a matrix x is a k-clique if it represents a truth assignment which encodes

[2] We write x for the $(k \times n)$-matrix (x_{ij}), y for the $((k-1) \times n)$-matrix (y_{ij}).

a k-clique. Similarly, a matrix y will be said to be a $(k-1)$-coclique if it represents a truth assignment which encodes a $(k-1)$-coclique. We will now construct, for every line L, some monotone circuits which separate all those k-cliques x from all those $(k-1)$-cocliques y which together falsify L. To be more precise, call a pair $<\alpha, \beta>$ of numbers *attainable for* L, if there is a k-clique x and a $(k-1)$-coclique y such that $\sum \alpha^L_{ij} x_{ij} = \alpha$ and $\sum \beta^L_{ij} y_{ij} = \beta$.

For every attainable pair $<\alpha, \beta>$ with $\alpha + \beta < \gamma^L$, we will construct a monotone circuit $C^L_{\alpha\beta}$ with inputs $e_{jj'}$, $1 \leq j < j' \leq n$, which takes as input (a $0-1$-string encoding[3]) an n-vertex graph G and outputs

(1) 1 if G is (represented by) a k-clique x such that $\sum \alpha^L_{ij} x_{ij} = \alpha$;

(2) 0 if G is (represented by) a $(k-1)$-coclique y such that $\sum \beta^L_{ij} y_{ij} = \beta$.

Before proceeding with the construction, let us assume that we have been successful for the last line $L \equiv 0 \geq 1$. The only attainable pair for L is $<0, 0>$, and as $0 + 0 < 1$ there is one circuit C^L_{00} which separates all k-cliques from all $(k-1)$-cocliques which falsify L. Since L is always false, C^L_{00} separates all k-cliques from all $(k-1)$-cocliques. Indeed, if G is a k-clique, then any encoding x of G satisfies $\sum \alpha^L_{ij} x_{ij} = 0$ and consequently $C^L_{00}(G) = 1$, by (1). Similarly, if G is a $(k-1)$-coclique then any encoding y of G satisfies $\sum \beta^L_{ij} y_{ij} = 0$, and by (2), $C^L_{00}(G) = 0$.

We now construct the circuits $C^L_{\alpha\beta}$ inductively.

- If L is a logical axiom, L cannot be falsified, so for no attainable pair $<\alpha, \beta>$ it can be true that $\alpha + \beta < \gamma^L$. Hence no construction is required.

- If L is any of the inequalities of form 1 - 3, then all attainable pairs are of the form $<\alpha, 0>$. But since all k-cliques satisfy L, we have that $\alpha \geq \gamma^L$, so again, no construction is required.

- Similarly, no construction is necessary if L is one of the inequalities of form 4 or 5.

[3]This encoding is such that for some ordering of the vertices $v_1, \ldots, v_n$, $e_{jj'}$ iff $(v_j, v_{j'})$ is an edge of G.

- If L is the inequality $-x_{ij}-x_{i'j'}-y_{lj}-y_{lj'} \geq -3$ then the only attainable pair $< \alpha, \beta >$ with $\alpha+\beta < -3$ is $< -2, -2 >$. We set $C^L_{-2-2} = e_{jj'}$.

  (1) If x encodes a k-clique G, and $-x_{ij} - x_{i'j'} = -2$ then both j and j' belong to the clique, so $e_{jj'}(G) = 1$.

  (2) If y encodes a $(k-1)$-coclique G, and $-y_{lj} - y_{lj'} = -2$ then j and j' belong to the same class, so $e_{jj'}(G) = 0$.

- If L is the result of a multiplication step then $L = d \times M$, for some previous line M. Obviously $< \alpha, \beta >$ is attainable for L and $\alpha+\beta < \gamma^L$ iff $< \alpha, \beta >$ is of the form $< d\alpha', d\beta' >$, where $< \alpha', \beta' >$ is attainable for M and $\alpha' + \beta' < \gamma^M$. L is falsified by precisely the same x, y as M, so $C^L_{d\alpha d\beta} := C^M_{\alpha\beta}$ has the desired properties.

- Similarly, if $L = \lceil \frac{M}{d} \rceil$, for some previous line M then a pair $\langle \alpha, \beta \rangle$ is attainable for L with $\alpha + \beta < \gamma^L$ iff $< \alpha d, \beta d >$ is attainable for M with $\alpha d + \beta d < \gamma^M$. We set $C^L_{\alpha\beta} := C^M_{d\alpha\, d\beta}$.

- Let $L = M + N$, and let $< \alpha, \beta >$ be attainable for L. Call a number δ *x-consistent with* α if there is a k-clique x such that $\sum_{ij} \alpha^L_{ij} x_{ij} = \alpha$ and $\sum_{ij} \alpha^M_{ij} x_{ij} = \delta$, similarly, δ is y-consistent with β if $\sum_{ij} \beta^L_{ij} y_{ij} = \beta$ and $\sum_{ij} \beta^M_{ij} y_{ij} = \delta$, for some $(k-1)$-coclique y. If $\alpha + \beta < \gamma^L$, α^M is x-consistent with α, and β^M is y-consistent with β, then we have that $< \alpha^M, \beta^M >$ is attainable for M, $\langle \alpha^N, \beta^N \rangle := \langle \alpha - \alpha^M, \beta - \beta^M \rangle$ is attainable for N, and $\alpha^M + \beta^M < \gamma^M$ or $\alpha^N + \beta^N < \gamma^N$ (since $\alpha^M + \alpha^N + \beta^M + \beta^N = \alpha + \beta < \gamma^L = \gamma^M + \gamma^N$). Therefore one of the circuits $C^M_{\alpha^M \beta^M}$, $C^N_{\alpha^N \beta^N}$ exists and can be used in the construction of $C^L_{\alpha\beta}$. We let $D^L_{\alpha^M \beta^M}$ be $C^M_{\alpha^M \beta^M}$, if $\alpha^M + \beta^M < \gamma^M$, and $C^N_{\alpha^N \beta^N}$ otherwise. Finally, we set $C^L_{\alpha\beta} := \bigvee_{\alpha^M} \bigwedge_{\beta^M} D^L_{\alpha^M \beta^M}$, where the disjunction ranges over all those α^M which are x-consistent with α and the conjunction ranges over all those β^M which are y-consistent with β.

  (1) Let x be an encoding of a k-clique G and let $\sum \alpha^L_{ij} x_{ij} = \alpha$. Then, for $\alpha^M := \sum \alpha^M_{ij} x_{ij}$, consider some y-consistent β^M. If $\alpha^M + \beta^M < \gamma^M$ then $D^L_{\alpha^M \beta^M} = C^M_{\alpha^M \beta^M}$ and, by induction hypothesis, $C^M_{\alpha^M \beta^M}(G) = 1$. If $\alpha^M + \beta^M \geq \gamma^M$ then $\alpha^N + \beta^N < \gamma^N$,

and $D^L_{\alpha^M\beta^M} = C^N_{\alpha^N\beta^N}$. Again by induction hypothesis, we have $C^N_{\alpha^N\beta^N}(G) = 1$. Thus $\bigwedge_{\beta^M} D^L_{\alpha^M\beta^M}(G) = 1$ hence $C^L_{\alpha\beta}(G) = 1$.

(2) Let G be a $(k-1)$-coclique and let y be an encoding of G with $\sum \beta^L_{ij} y_{ij} = \beta$. Let $\beta^M := \sum \beta^M_{ij} y_{ij}$, and consider some value α^M which is x-consistent with α. If $\alpha^M + \beta^M < \gamma^M$ then $D^L_{\alpha^M\beta^M}(G) = C^M_{\alpha^M\beta^M}(G) = 0$, otherwise, $D^L_{\alpha^M\beta^M}(G) = C^N_{\alpha^N\beta^N}(G) = 0$. Thus for each α^M some $D^L_{\alpha^M\beta^M}$ yields 0 for G, so $C^L_{\alpha\beta}(G) = 0$.

This completes the construction.

2.3 Complexity analysis

We can best express the size of the circuit in terms of p, the maximum number of attainable pairs for any line in the proof. Thus, if a_L is the number of different values of $\sum \alpha^L_{ij} x_{ij}$, over all k-cliques x, and b_L is the number of different values of $\sum \beta^L_{ij} y_{ij}$, over all $(k-1)$-cocliques y, then $p = \max_L \{a_L \cdot b_L\}$.

Proposition 1 *The circuit constructed in the last section has at most sp^2 many gates (not counting input gates), where s is the number of addition steps in the proof.*

Proof. Imagine the circuit organised in levels, where the i^{th} level corresponds to the i^{th} addition line L in the CP proof and contains the output gates of all subcircuits $C^L_{\alpha\beta}$. We show by induction on i that up to the i^{th} level there are most $ip(p-1)$ non-input gates.

$i = 0$: Without addition steps there are only input gates.

$i > 0$: Each of the circuits $D^L_{\alpha^M\beta^M}$ used in the construction of $D^L_{\alpha\beta}$ resides in some level $j \leq i-1$. For each α^M the disjunction $\bigvee_{\beta^M} D^L_{\alpha^M\beta^M}$ costs $b_M - 1$ extra gates, and there are a_M many such disjunctions, costing at most $a_M(b_M - 1)$. Their conjunction $\bigwedge_{\alpha^M}\left(\bigvee_{\beta^M} D^L_{\alpha^M\beta^M}\right)$ costs another $a_M - 1$ gates. Since there are $a_L b_L$ many circuits $C^L_{\alpha\beta}$ on level i, we need no more than $a_L b_L(a_M b_M - 1) \leq p(p-1)$ extra gates to go from level $i-1$ to level i. $\square$

Now we are in a position to apply Alon & Boppana's lower bound. They showed in [AB 87] that for $k = \frac{1}{4}\left(\frac{n}{\log n}\right)^{\frac{2}{3}}$, the separation of k-cliques and $(k-1)$-cocliques on n vertices requires monotone circuit size $2^{\Omega(n^{\varepsilon})}$, for every $\varepsilon < \frac{1}{3}$.

Theorem 2.1 *Let $k = \frac{1}{4}\left(\frac{n}{\log n}\right)^{\frac{2}{3}}$, and let, for each $n \in N$, Π_n be a CP proof for the k-clique tautology. Assume further that for some polynomial p each Π_n has only coefficients between $-p(n)$ and $p(n)$. Then, if $s(n)$ is the number of addition steps of Π_n, we have $s(n) = 2^{\Omega(n^{\varepsilon})}$, for every $\varepsilon < \frac{1}{3}$.*

Proof. This follows directly from the proposition, together with the lower bound from [AB 87], since with coefficients between $-p(n)$ and $p(n)$ there can be no more than n^l attainable pairs in any line of Π_n, for some l. $sn^l = 2^{\Omega(n^{\varepsilon'})} \Longrightarrow s = 2^{\Omega(n^{\varepsilon'} - l\log n)} = 2^{\Omega(n^{\varepsilon})}$, where $\varepsilon' < \frac{1}{3}$ is chosen such that $n^{\varepsilon'} - l\log n > n^{\varepsilon}$ for large enough n. $\square$

References

[AB 87] N.Alon, R.B.Boppana, *The monotone circuit complexity of Boolean functions*, Combinatorica 7(1), 1987, pp. 1–22.

[Buss-Clote] S.Buss, P.Clote, *Cutting Planes, connectivity, and threshold logic*, september 2, 1994.2

[Cook-Coullard-Turan] W. Cook, C.R.Coullard, and G.Turan, *On the Complexity of cutting plane proof systems.* Discrete Applied Mathematics, 18:25-38, 1987.

[Goerdt] A.Goerdt, *Cutting Plane versus Frege Proof Systems.* In Egon Börger, editor, *Computer Science Logic 1990,* volume552, pages 174-194, 1992. Springer Lecture Notes in Computer Science.