Innovations in Computer Science 2010

Effectively Polynomial Simulations

Toniann Pitassi[1] Rahul Santhanam[2]
[1]University of Toronto, Toronto, Canada
[2]University of Edinburgh, Edinburgh, UK
toni@cs.toronto.edu rsanthan@inf.ed.ac.uk

Abstract: We introduce a more general notion of efficient simulation between proof systems, which we call *effectively-p simulation*. We argue that this notion is more natural from a complexity-theoretic point of view, and by revisiting standard concepts in this light we obtain some surprising new results. First, we give several examples where effectively-p simulations are possible between different propositional proof systems, but where p-simulations are impossible (sometimes under complexity assumptions). Secondly, we prove that the rather weak proof system G_0 for quantified propositional logic (QBF) can effectively-p simulate *any* proof system for QBF. Thus our definition sheds new light on the comparative power of proof systems. We also give some evidence that with respect to Frege and Extended Frege systems, an effectively-p simulation may not be possible. Lastly, we prove new relationships between effectively-p simulations, automatizability, and the P versus NP question.

Keywords: Effectively polynomial simulation; proof complexity; automatizability

1 Introduction

It is well known that NP=coNP if and only if there exists a polynomially bounded propositional proof system. A large research program over the last thirty years has been to classify proof systems according to their relative strength, and to develop lower bound methods for proving that these increasingly powerful proof systems are not polynomially bounded. Conventionally, "relative strength" has been measured using the notion of polynomial simulation (p-simulation). A proof system A p-simulates a proof system B if every tautology has proofs in A of size at most polynomially larger than in B. Intuitively, if A p-simulates B, then it is at least a strong a proof system as B, because it can prove tautologies at least as efficiently as B.

The notion of p-simulation is very natural from the *proof theory* perspective, where a primary goal is reverse mathematics. In reverse mathematics, the goal is to understand the minimal set of axioms required to prove specific mathematical statements. For example, Extended Frege systems correspond to a system of bounded arithmetic allowing induction on polynomial-time computable predicates, and it is known that a significant amount of number theory can be carried out within this axiomatic system. A fundamental open question in reverse mathematics is to determine the weakest propositional proof system that can resolve the P versus NP question. The notion of p-simulation is a very useful tool in this overall endeavor, as it allows us to show that any mathematical statement that can be proven efficiently in one system B can also be proven in another system A.

However, the notion of p-simulation has some limitations as a measure of relative proof system strength from the *algorithmic perspective*. From an algorithmic point of view, proof systems can be viewed as non-deterministic algorithms for unsatisfiability. In fact, all SAT solvers to date can be viewed as deterministic implementations of a particular propositional proof system. For example, the Davis-Putnam algorithm (DPLL) algorithm for SAT, as well as the highly successful clause learning methods are highly optimized deterministic implementations of Resolution. Similarly, the "lift and project" methods used in combinatorial optimization can be viewed as deterministic implementations of the proof system LS+. Thus proof complexity is not just an approach to the NP versus $coNP$ question, but to the P versus NP question as well. This is because proving superpolynomial proof size lower bounds for a particular proof system unconditionally rule out a large class of methods for solving SAT in polynomial time. When comparing such methods to one another, it seems a needless restriction to insist on comparing them on the same input as would be required by the p-simulation notion. If the notion of feasibility is polynomial time, then arguably some polynomial-time preprocessing of the input should be allowed for free when comparing two different methods.

Also, when proving lower bounds against proof systems, the lower bounds are typically shown for some

specific class of tautologies. In the analogous situation in computational complexity, when a lower bound is shown, eg., Parity not in AC^0, the lower bound also holds for all languages that Parity is reducible to under $DLOGTIME$ reductions, since AC^0 is closed under $DLOGTIME$ reductions. In a similar way, in proof complexity, one would like to rule out not just lower bounds for a certain sequence of tautologies, but also for any sequence of tautologies that is at least as "hard" as the sequence for which the lower bound is proved. There seems no straightforward way to do this using p-simulation without insisting that the lower bound should go through for *all* sequences of tautologies that don't have small proofs.

In this paper, we explore a more general notion of p-simulation called "effectively polynomial simulation" which we feel is very natural from a complexity-theoretic point of view. Though closely related concepts have been studied (see Related Work subsection), we are the first to explicitly define this new notion and demonstrate its wide-ranging applicability. Essentially, given proof systems A and B for tautologies, B is said to effectively p-simulate A if there is an efficient truth-preserving reduction mapping tautologies with small proofs in A to tautologies with small proofs in B. A p-simulation is simply the special case of an effectively-p simulation where the reduction is the identity.

While the original notion of p-simulation is fundamental, there are several reasons to study this more general notion of simulation as well. First, we feel that the effective simulation notion addresses the weaknesses of the notion of p-simulation mentioned earlier. Where we're chiefly interested in feasibility, it is natural to think of polynomial time as "given for free", and for the notion of simulation we use to be closed under polynomial-time reductions. From the point of view of practice, this captures the possibility that when a proof system is used as a SAT solver, polynomial-time preprocessing applied to the input formula could make the algorithm more effective. In fact, encoding problems (such as planning and inference) into SAT has become a huge subarea within artificial intelligence.

It could be the case that effectively-p simulation is an interesting notion, but that there are no interesting examples of it which are not p-simulations. We show that this is far from the truth. We give a number of examples (some new, some implicit in earlier work) of effectively-p simulations between proof systems where either no p-simulation is known or a p-simulation has been proven not to exist. This gives a much cleaner picture of proof complexity. For example, there are many variants of Resolution - Linear Resolution, Clause learning, k-Res - which are either not equivalent to Resolution with respect to p-simulations or there the equivalence is not known, however they are *all* equivalent to Resolution with respect to effectively-p simulations. Moreover, effectively-p simulation can be used to compare proof systems of different kinds. For instance, proof systems with different vocabularies can be compared by this notion, or a quantified proof system can be compared in power with a propositional one. Indeed, one of our main results in this paper shows that a certain quantified proof system is universal in that it effectively-p simulates every propositional or quantified system.

The study of effective simulations is helpful in understanding the concept of automatizability of proof systems, which has seen a lot of interest in recent years. An effectively-p simulation of proof system A by proof system B implies a reduction from automatizability of A to automatizability of B. In the case that proof system A is already known not to be automatizable under a certain assumption, this gives non-automatizability of B under the same assumption.

Effectively-p simulations raise the possibility of "lifting" a lower bound from a weaker proof system to a stronger one, in a similar way to a recent paper of Beame, Huynh-Ngoc and Pitassi [6]. If we have an effectively p-simulation of a stronger proof system (in the sense of p-simulation) by a weaker one (a phenomenon of which we give many examples in this paper), and supposing that we have a lower bound method against the weaker proof system which works for tautologies in the range of the reduction, we automatically get lower bounds against the stronger proof system as well.

Finally, the study of effectively-p simulations gives rise to interesting new questions, eg. does Resolution effectively-p simulate Extended Frege, answering which seems to require strengthening and extending known proof techniques and developing new connections between computational complexity and proof complexity.

1.1 Related Work

We observe that the concept of effectively-p simulation has been around implicitly for a long time. In fact in the original paper by Cook and Reckhow [14] defining propositional proof systems, and p-simulations, they compare several different nondeterministic algorithms (proof systems), not all of which are for the same coNP complete language. For instance, Resolution works exclusively with unsatisfiable CNF formulas, whereas different Frege systems work with tautological formulas over varying bases. In order to

compare these proof systems directly, they consider natural reductions between the different languages. A more extreme example exists when comparing the strength of the Hajos calculus to the strength of various propositional proof systems. Here we are trying to compare nondeterministic algorithms for very different co-NP complete languages, and once again the comparison can only be accomplished by introducing a mapping (reduction) that allows one to convert strings in one language to strings in the other.

Razborov [23] and Pudlak [22] defined a notion of reduction between disjoint NP-pairs, with application to proof complexity in mind. Their notion corresponds to our notion of simulation when applied to the canonical pair of a propositional proof system. However, our notion is more general in that it applies, for example, to quantified propositional proof systems as well.

It was already observed in Pudlak's work, as well in subsequent work of Atserias and Bonet [1], that reductions between disjoint NP-pairs help to understand automatizability. Though Atserias and Bonet never define effective simulations explicitly, one of their main results can be interpreted as an effective simulation of k-Res by Resolution.

The fact that reductions are implicitly used even in practice was pointed out very elegantly in a paper by Hertel, Hertel and Urquhart [17]. In their paper, they argue that sometimes reductions, even very natural and seemingly harmless ones, can be quite dangerous (meaning that they can drastically alter the difficulty of the problem if one is not extremely careful). To support this claim, they present several examples of reductions between proof systems where the blowup under one natural setting is polynomial, but the blowup under another natural setting is exponential.

A summary of our contributions and outline of the rest of the paper is as follows. In Section 2 we define effectively-p simulations and present some basic facts concerning the definitions. As mentioned earlier, although this concept was around before, we are the first to present a general definition and to study the concept in its own right. In Section 3, we present some positive results, giving several examples where effectively-p simulations are possible, even in cases where p-simulations are provably impossible, or are unlikely. In Section 4, we present some negative results, giving some examples where effectively-p simulations are unlikely. A big question is whether or not a sufficiently strong system can effectively-p simulate a seemingly stronger one. When we began this work, we conjectured (or hoped) that Frege would effectively-p simulate Extended Frege systems. However, in Section 4, we present some conditional results indicating that this is probably unlikely. In Section 5 we present some new results concerning the connection between effectively-p simulation, automatizability, and the P versus NP question. We conclude with open problems in Section 6.

2 Definitions

We first recall the usual notion of polynomial simulations given in the literature. In what follows, we will be working with boolean formulas over the standard basis: AND, OR and NOT. We assume some standard encoding of propositional formulas. For a formula f, let $[f]$ denote the encoding of f. Let $TAUT$ denote the set of valid propositional formulas. For the sake of convenience, we do not distinguish between formulas and their encodings here.

DEFINITION 2.1 *A propositional proof system is a polynomial-time onto function A from $\{0,1\}^*$ to $TAUT$.*

In the above definition, we think of a domain element as an encoding of a proof, and A maps the encoding of a proof to the (encoding of) the formula that is being proved. The onto condition ensures that the proof system is complete.

Note that the above definition does not require the proof system to be propositional in the usual intuitive sense (where the objects being manipulated are restricted to being propositional formulas.) For example, first order systems of arithmetic, such as PA (Peano arithmetic) fit the definition of a propositional proof system. Even systems which do not explicitly talk about boolean formulas (such as standard systems of set theory, ZFC) can also be viewed as fitting the definition, where the function interprets certain formulas in the underlying language as representing the corresponding boolean formula.

Now let $QTAUT$ denote the set of all encodings of valid quantified propositional formulas, where the quantification is over all the variables of the formula. We define quantified propositional proof systems analogously to propositional proof systems.

DEFINITION 2.2 *A quantified propositional proof system is a polynomial-time onto function A from $\{0,1\}^*$ to $QTAUT$*

DEFINITION 2.3 *Let A and B be two proof systems. Then A p-simulates B if for all formulas f, the shortest A-proof for f is at most polynomially longer than the shortest B proof of f.*

A strongly p-simulates B if A p-simulates B and moreover, there is a polynomial-time computable function f that transforms B-proofs of f into A-proofs of f.

Remark 1. Note that our definition is implicitly with respect to some class of formulas. For propositional systems A and B, the p-simulation is (by default) with respect to propositional formulas and for quantified propositional proof systems, the p-simulation is (by default) with respect to quantified boolean formulas. More generally for any class of formulas, and any two proof systems that can prove formulas of this type, we can define a p-simulation with respect to this class of formulas. When talking about p-simulations, or effectively-p simulations (defined next), we will explicitly mention the formulas only when it is not the default.

As defined above, for tautology f, if one proof system A always contains a proof of f that is within a polynomial factor of the size of the smallest B-proof of f, then A is said to p-simulate B. This relationship maps B-proofs of f to similarly-sized A-proofs of f. We can relax this definition to produce another kind of simulation (an effectively-p simulation, defined below) in which we map B proofs of f to similarly sized A-proofs of f', where f' is some formula which is a tautology if and only if f is a tautology, and moreover, f' can be produced from f efficiently.

We use the following definition of a truth-preserving transformation both for propositional and quantified tautologies.

DEFINITION 2.4 *We say that R is a polynomial-time in m truth-preserving transformation from boolean formulas to boolean formulas if, for all boolean formulas f, f is in TAUT (respectively QTAUT) if and only if $R(f,m)$ is in TAUT (respectively QTAUT), and $R(f,m)$ runs in time polynomial in $|f|+m$, where $|f|$ is the number of connectives in f and m is an auxiliary parameter.*

DEFINITION 2.5 *Let A and B be two proof systems. Then A effectively-p simulates B if there is a polynomial-time in m truth-preserving transformation from (encodings of) boolean formulas to (encodings of) boolean formulas, $R(f,m)$ such that when m is at least the size of the shortest B-proof of f, $R(f,m)$ has an A proof of size polynomial in $|f|+m$. If there also exists a polynomial-time function (again polytime in $|f|+m$) that maps B-proofs of f to A-proofs of $R(f,m)$, then we say that A strongly effectively-p simulates B.*

Remark 2. The role of the parameter m in the definition might not be clear at first sight. We could define our notion omitting m completely by stipulating that $R(f)$ is computable in time polynomial in $|f|$ and that $R(f)$ has small A-proofs if f has small B-proofs. The point is that our definition is more relaxed - it allows the reduction to operate in time polynomial in the size of the smallest A-proof for f rather than in the size of f. As we show later, this relaxed notion still gives a reduction from automatizability of B to automatizability of A. Since one of our main motivations for exploring this notion is the connection to automatizability, it makes sense to work with the weakest notion of simulation for which this connection holds. We note, though, that of the several positive results about effectively p-simulations in this paper, all but Proposition 3.1 and Theorem 3.3 go through even if the stronger notion where the reduction can only take time polynomial in $|f|$ is used.

Remark 3. It is clear that if A can p-simulate B, then A can also effectively-p simulate B. For example, EF can effectively-p simulate Frege. In the opposite direction, it may seem at first that by using extension variables, many reverse effective simulations are easily possible. Using our same example, we could try to effectively simulate EF by Frege by adding a polynomial-sized set of extension axioms for predicates that are complete for $P/poly$, thereby allowing Frege to simulate each EF step by using an instance of the newly defined predicate. As far as we can see, this is not possible, since one seems to need the exact predicates that are required in the EF proof, even in the presence of the substitution axiom. Thus intuitively, obtaining an effective simulation of EF by Frege seems to require either (i) that the reduction, given f, finds an EF proof of f and then defines the needed predicates via extension axioms (an impossiblility under complexity assumptions), or (ii) arguing that there exists a small (polynomial in n) "core" of predicates that would suffice to simulate EF proofs for *all* formulas of size n.

We next define automatizability. Like p-simulation and effectively-p simulation, automatizability comes in two flavors: strong and weak.

DEFINITION 2.6 *A (propositional or quantified) proof system A is strongly automatizable if there is an algorithm Q such that if ϕ is a valid formula whose smallest A-proof is of size s, then $Q(\phi)$ runs in time $poly(s+|\phi|)$ and produces an A-proof of ϕ. If Q produces not an A-proof but a proof in some other proof system B, then A is said to be weakly automatizable.*

When we say "automatizable" in future, we mean "weakly automatizable" by default. effectively-p simulation of proof system A by proof system B implies

that if B is weakly automatizable then so is A. In other words, effective simulation gives a reduction between the automatizability properties of proof systems. This was observed in essence already by [22] and [1]; it is even easier to see with our definitions.

Proposition 2.7 *Let A and B be proof systems. If B effectively-p simulates A and B is weakly automatizable, then A is weakly automatizable.*

We also consider at times in this paper the "quasi"-analogues of the polynomial-time notions defined above. For instance, a proof system is quasi-automatizable if there is a proof-finding procedure that operates in time quasipolynomial in the size of the smallest proof, and a quasi-effective simulation is one that operates in time quasipolynomial in the parameter m. Analogues of our propositions for the polynomial-time versions of simulation and automatizability also hold for the quasi-analogues.

We now describe some specific proof systems.

2.1 Propositional Proof Systems

The resolution principle says that if C and D are two clauses and x is a variable, then any assignment that satisfies both $(C \vee x)$ and $(D \vee \neg x)$ also satisfies $C \vee D$. The clause $C \vee D$ is said to be the *resolvent* of the clauses $C \vee x$ and $D \vee \neg x$ and derived by *resolving on* the variable x. A *resolution refutation* of a clause C from a CNF formula f consists of a sequence of clauses in which each clause is either a clause of F or is a resolvent of two previous clauses, and C is the last clause in the sequence. It is a *refutation* of f if C is the empty clause. The *size* of a resolution refutation is the number of resolvents in it.

A *linear resolution* refutation of f is a resolution refutation with the additional restriction that the underlying graph structure must be linear. That is, the proof consists of a sequence of clauses $C_1, \ldots, C_m$ such that C_m is the empty clause, and for every $1 \leqslant i \leqslant m$, either C_i is an initial clause or C_i is derived from C_{i-1} and C_j for some $j < i - 1$.

We briefly review the definition of Frege and Extended Frege systems. More detailed definitions can be found in [8, 26, 28]. The sequent calculus is a very elegant proof system that can be used as a framework for capturing many natural and well-studied proof systems. A propositional *sequent* is a line of the form $\Gamma \to \Delta$, where Γ and Δ are finite sets of propositional formulas. The intended meaning of the sequent is that the conjunction of the formulas in Γ implies the disjunction of the formulas in Δ. A PK proof (a propositional sequent calculus proof) of a sequent $\Gamma \to \Delta$ is a sequence of sequents, where: (i) each sequent is either an instance of a PK axiom, or follows from one or two previous formulas by an instance of a PK rule and (ii) the final sequent is $\Gamma \to \Delta$. The PK rules are very natural. They include some structural rules, as well as two rules for each connective, one for introducing the connective on the left side of the arrow, and one for introducing the connective on the right side of the arrow. The most important rule of PK is the *cut-rule*, which allows one to infer $\Gamma \to \Delta$ from $\Gamma, A \to \Delta$ and $\Gamma \to A, \Delta$. A PK proof of a formula f is a proof of the sequent $\to f$.

With no restrictions on the cut-rule, PK is polynomially equivalent to Frege systems. By restricting the cut rule, we can elegantly obtain many commonly studied subsystems of Frege systems. For example, if the cut rule is restricted to formulas A which are just literals, then we have a system which is equivalent to resolution. By restricting the cut rule to bounded-depth formulas (AC_0), we obtain bounded-depth, or AC_0-Frege systems, and so on. An Extended Frege proof of a formula f is a proof of $E \to f$, where E is a sequence of extension axioms. An extension axiom is an axiom of the form $(A \iff l_1 \vee \ldots \vee l_k)$, where l_i are literals and A is a new variable. Extension axioms allow efficient reasoning about predicates computable by polynomial-size circuits, by introducing new variables to represent the various subcomputations of the circuit.

2.2 Quantified Propositional Systems and Beyond

First we recall the usual inductive definitions of quantified boolean formulas. $\Sigma_0^q = \Pi_0^q$ is the class of quantifier free propositional formulas. Both Σ_i^q and Π_i^q are closed under the boolean operations $\wedge$, $\vee$ and $\neg$, and the negation of a Σ_i^b formula is a Π_i^q formula, and vice versa. Σ_{i+1}^q contains both Σ_i^q and Π_i^q and formulas of the form $\exists x_1 \ldots \exists x_k A$, where A is a Π_i^q formula. Similarly, Π_{i+1}^q contains both Σ_i^b and Π_i^q and formulas of the form formulas are formulas of the form $\forall x_1 \ldots \forall x_k A$, where A is a Σ_i^q formula. Thus, Σ_i^q (Π_i^q) formulas are quantified boolean formulas with i blocks of alternating quantifiers, beginning with $\exists$ ($\forall$).

The system G is a proof system for QBF formulas that extends PK [18]. Lines in the proof are still sequents, $\Gamma \to \Delta$ but now Γ, Δ are finite sets of QBF formulas. The rules of G include all propositional rules of PK, and additionally include rules for introducing each quantifier (on both the left side, and the right side of the arrow). The system G_0 is a proof system for QBF where the cut rule is restricted to propositional formulas only. Similarly, G_i is a subsystem of G obtained by restricting the cut rule to Σ_i^q QBF formulas. Note that the G systems can be used

to prove any QBF formula.

Beyond QBF proof systems, we can view any standard axiomatic system as being a proof system for propositional reasoning. As mentioned earlier, Peano Arithmetic, and even ZFC (Zermelo-Fraenkel Set Theory) can be studied with respect to their ability to prove propositional formulas (with a suitable encoding of propositional formulas).

3 Effectively-p Simulations: Positive Results

As mentioned earlier, anytime we have a p-simulation between two proof systems, we also have an effectively-p simulation. Thus, for example, the usual hierarchy of p-simulations continues to hold under effectively-p simulation.

We also observe that effective simulations can establish equivalences between two proof systems, where the equivalence with respect to p-simulation hinges on finding short proofs for a particular statement. In our view, these are examples pointing out that sometimes p-simulation is the better concept, since effectively-p simulation does not provide a fine enough granularity between systems for applications in reverse mathematics. For example, it is known that the monotone sequent calculus (monotone PK) can quasipolynomially simulate PK with respect to monotone sequents, and it is open whether or not a p-simulation is possible [2]. On the other hand, it is not hard to show that monotone PK can effectively-p simulate PK with respect to monotone sequents. Similarly, it is not hard to see, using the results of Soltys and Cook [27], that Frege can quasipolynomially effectively-p simulate the system LAP (capturing linear algebra reasoning).

Another simple observation allows us to obtain effective p-simulations between two proof systems whenever the stronger of the two is automatizable:

Proposition 3.1 *Let P and P' be two proof systems that are both automatizable. Then each effectively-p simulates the other.*

Proof. We show that P' effectively simulates P; the other direction follows by symmetry. Given an input formula ϕ for P and the parameter m, we define an efficient simulation as follows. We run the automatization procedure for P on ϕ. If it halts with a proof within $poly(m)$ steps, we output a trivial tautology which has polynomial-size proofs in P'. If not, then we output ϕ. This transformation is truth-preserving, since the output of the reduction is a tautology iff the input is. Also, if the input formula has proofs of size $\leqslant poly(m)$ in P, then the output formula has small proofs too, since it is a trivial tautology.

As a consequence, we get that the following pairs of proof systems effectively (quasi)simulate each other: Nullstellensatz and Polynomial Calculus, Tree Resolution and Polynomial Calculus, small rank LS and small rank LS+, tree-LS and small rank LS, small rank LS+ and tree-LS. On the other hand, it is known that between many of these systems there are no p-simulations. For example, it is known that Nullstellensatz does not (quasi)p-simulate PC [12]; low rank LS does not p-simulate low rank LS^*; and Tree resolution does not (quasi)p-simulate any of the other systems.

In this section we present some other examples where an effectively-p simulation is possible, but a p-simulation is not possible, or is conjectured to be not possible.

3.1 Linear Resolution

Our first example is the theorem whose proof has been known for some time, showing that linear resolution can effectively-p simulate all of Resolution.

Theorem 3.2 *[11] Linear resolution effectively-p simulates Resolution.*

We sketch the proof here, both for completeness, and to give the reader an idea of how such a simulation can be proven. Let f be a CNF formula over $x_1, \ldots x_n$ and let g be the following set of $2n^2$ clauses:

$$\{x_i \vee \neg x_i \vee x_j^a \mid 1 \leqslant i, j \leqslant n, a \in \{0,1\}\}.$$

Suppose that f is an unsatisfiable CNF formula that has a resolution refutation of size S. Then it can be shown inductively that there is a linear resolution refutation of $f \wedge g$ of size polynomial in S, as follows. Let $\pi = C_1, \ldots C_S$ be the resolution refutation of f. Since $C_1 \in f$, we can clearly derive C_1 in linear resolution. Now assume we have a linear resolution derivation L that ends with C_i and includes $C_1, \ldots C_{i-1}$ in order along the line. We show how to extend L to derive C_{i+1}.

There are two cases. The first is where C_{i+1} is derived from two earlier clauses C_j, C_k in π by resolving on x, $1 \leqslant j < k \leqslant i$. If $i = k$ then we can simply add C_{i+1} to the end of L. Otherwise let $l_1, \ldots, l_w$ be the literals in C_i. Resolve C_i with the following (initial) clauses of g: $(x \vee \neg x \vee \neg l_1), \ldots, (x \vee \neg x \vee \neg l_w)$ until the last clause in L is $(x \vee \neg x)$. Now resolve this last clause on x with C_j and then C_k so the last clause becomes C_{i+1}. The other case is when C_{i+1} is an axiom containing the literal x^a. In this case, derive the clause $(x \vee \neg x)$ as above from C_i and then resolve the

axiom C_{i+1} with it to obtain C_{i+1} again at the end of the line.

It is still unknown whether or not linear resolution can p-simulate resolution, but it is conjectured to be false.

3.2 Clause Learning

Our second example is a very recent result proving that Clause Learning effectively-p simulates Resolution. Clause Learning is a particular refinement of Resolution that is very important. Most state-of-the-art complete algorithms for SAT make use of highly optimized Resolution SAT solvers and all are based on the idea of Clause Learning. Informally, clause learning is an implementation of DPLL whereby intermediate clauses that are generated are "learned" or "cached" along the way. Then in later states of the DPLL algorithm, the cache is checked to see if the current subproblem to be solved has already been solved earlier. This gives a way of pruning the DPLL tree and it has been shown to be highly effective, not only for SAT, but also for important generalizations of SAT such as QBF solvers and Bayesian inference. (See for example [5, 15, 21, 25].) [7] and [29] formalize Clause Learning and the former shows that that it is superpolynomially more efficient than other common resolution refinements (such as regular and tree resolution.) Whether or not Clause Learning p-simulates Resolution is an important open problem. However, the following somewhat surprising theorem was recently proven.

Theorem 3.3 *[4] Clause learning effectively-p simulates Resolution.*

On the one hand, this proves formally that Clause Learning is as powerful as all of resolution with respect to solving SAT. But on the other hand, it unfortunately shows that finding clause learning proofs (in a worst-case sense) is as hard as finding general resolution proofs.

3.3 Effectively-p Simulations for Local Extensions

We make a simple observation that allows us to see several examples where p-simulations do not hold, but effectively-p simulations do hold.

DEFINITION 3.4 *Let f be a boolean function on k variables, $y_1, \ldots, y_k$. We assume without loss of generality that f is a CNF formula. The formula f^D is a CNF formula defining f. The variables of f^D are $y_1, \ldots, y_k$ plus variables x_C, for each clause C of f. The clauses of f^D are as follows. For each clause C of f, we have clauses that express the fact that C is equivalent to x_C.*

DEFINITION 3.5 *Let $x_1, \ldots, x_n$ be a vector of n Boolean variables. The set of all k-local boolean functions over $\vec{x}$ consists of all functions f such that f is a boolean function defined on a subset of k variables of $\vec{x}$. The formula $EXT(k, n)$ consists of the conjunction of the formulas f^D, where f ranges over all k-local boolean functions over $\vec{x}$.*

DEFINITION 3.6 *(k-local extensions of proof systems) Let P be a rule-based propositional proof system. Define $P(k)$ to be a propositional proof system containing all rules and axioms of P plus the additional axioms f^D for all k-local functions f.*

Examples of well-studied k-local extensions of standard proof systems include: $Res(k)$, $CP(k)$, $LS(k)$ and $LS^+(k)$. Indeed, Atserias and Bonet [1] implicitly show that Resolution effectively simulates $Res(k)$ for each constant k, and Pudlak [22] implicitly shows that CP effectively simulates $CP(2)$.

We generalize the above observations. Each of the above proof systems is obtained from the base system by introducing extension axioms for all conjunctions of up to k-literals. Note that our k-local extension is more general than these since we allow extension variables for *every* function on k variables and not just the AND function. The following lemma shows that as long as we obtain P' from P by adding extension variables for some local functions, then P can effectively-p simulate P'.

Lemma 3.7 *Let P be a rule based proof system. Suppose that P' is another proof system such that $P(k)$ p-simulates P', and P' p-simulates P. Then P effectively-p simulates P'. In particular, P effectively-p simulates $P(k)$.*

Proof. The proof is straightforward. Let P, P', $P(k)$ be defined as above, and let f be a formula over n variables, $\vec{x}$. We map f to $f' = f \wedge EXT(k, n)$. It is clear that the mapping is polynomial-time, and that it preserves satisfiability. We claim that if f has a short P' proof, then $f' = f \wedge EXT(k, n)$ has a short P-proof. By the p-simulation of P' by $P(k)$, f has a short $P(k)$ proof, and thus f' has a short P proof.

It follows from the above lemma that Res effectively-p simulates $Res(k)$ [1] and similarly for $CP/CP(k)$, $LS/LS(k)$, and $LS^+/LS^+(k)$. In all of these cases, it is known that p-simulations are not possible. (See [26].)

3.4 G_0 Can Effectively-p Simulate any Proof System

In this section, we will prove that G_0 can effectively-p simulate any quantified propositional proof system, including Peano Arithmetic, and Zermelo-Frankl Set Theory (ZFC). Sadowski [24] showed that if there is an optimal quantified propositional proof system, i.e., a quantified propositional proof system that p-simulates all others, then $NP \cap coNP$ has complete languages, which is considered unlikely. Our result shows that in contrast, there is a proof system which is *effectively optimal*.

Theorem 3.8 *For any i, G_0 can effectively-p simulate any proof system for Σ_i^q quantified boolean formulas.*

Proof. (sketch) Let S be any quantified proof system for Σ_i^q-QBF formulas. We want to show that G_0 can effectively-p simulate S. The high level idea is as follows. We define a reduction from Σ_i^q quantified propositional formulas to Σ_{i+1}^q quantified propositional formulas as follows. Given a Σ_i^q QBF formula f, and a number m, we map f to f', where f' is the formula: $Refl_m^S \rightarrow f$. $Refl_m^S$ is the reflection principle for S and it will be a fixed $\forall\Sigma_i^q$ formula depending only on S and m that asserts that for any Σ_i^q formula A, and for any α, if α is an S proof of A, Then A is satisfied by all assignments. We now proceed with the details, and begin by defining f'.

By definition, S is a polynomial-time algorithm that maps strings (encodings of S-proofs) to strings (encodings of Σ_i^q-QBF formulas). We will assume without loss of generality that all proof systems S map strings of length m to strings of length m: we can always pad the output with leading zeroes if this is not the case.

Now fix m and consider S on inputs of length m. Since S is polynomial-time computable, there is a fixed circuit, C_m, of size polynomial in m with inputs $\vec{x} = x_1, \ldots, x_m$ that computes $S(\alpha)$ for each $\alpha \in \{0,1\}^m$. Using extension variables to represent each intermediate gate of C_m, we can define a formula $Proof_m^S(\vec{x}, \vec{y})$ such that $Proof_m^S(\alpha, \beta)$ is true if and only if C_m on input α outputs β. (Note that the variables of the formula are $\vec{x}$, $\vec{y}$, plus the extension variables used to define each intermediate gate of C_m.)

Fix some standard encoding of Σ_i^q-QBF formulas. Then we can define a propositional formula $Formula_i(\vec{y})$ that is true if and only if y encodes a $(\Sigma_i^q \cup \Pi_i^q)$-QBF formula. Similarly we can define a Σ_i^q formula $SAT_{i,m}(\vec{y}, \vec{z})$ that is true if and only if $Formula_i(\vec{y})$ is true, and $\vec{z}$ satisfies the Σ_i^q formula encoded by $\vec{y}$. (Here m is the length of the vectors $\vec{x}, \vec{y}, \vec{z}$.) $SAT_{i,m}$ is defined inductively. For example, the following equalities hold:

(1) $SAT_{i,m}([\exists x A(x)], \tau) = \exists x SAT_{i,m}([A(x)], \tau)$,
(2) $SAT_{i,m}([\forall x A(x)], \tau) = \forall x SAT_{i,m}[A(x)], \tau)$,
(3) $SAT_{i,m}([\neg A], \tau) = \neg SAT_{i,m}([A], \tau)$, and
(4) $SAT_{i+1,m}([A], \tau) = SAT_{i,m}([A], \tau)$ whenever $A \in \Pi_i^q \cup \Sigma_i^q$.

Note that $SAT_{i,m}$ will be a Σ_i^q formula. (Of course, we will need to introduce polynomial in m many extension variables in order to be able to encode and decode QBF formulas, and in order to manipulate them.)

Finally, we define $Refl_m^S$ to be the following formula: $\forall\vec{x}\forall\vec{y}\forall\vec{z}(\neg Proof_m^S(\vec{x}, \vec{y}) \vee SAT_{i,m}(\vec{y}, \vec{z})$. This formula states that for every $\vec{x}, \vec{y}, \vec{z}$ of length m, if $\vec{x}$ codes an S-proof of some formula, f encoded by $\vec{x}$, then f is satisfied by every assignment $\vec{z}$ to its free variables. The formula $Refl_m^S$ is a $\forall\Sigma_i^q$ formula.

Our reduction, given f and m, will map f to $f' = Refl_m^S \rightarrow f$. The reduction is clearly polynomial-time and truth preserving. It is left to argue that if f is a Σ_i^q-QBF formula with a short S-proof, then f' has a short G_0 proof.

Let $[f]$ be the encoding of f, and suppose that f has an S-proof, α, of size m. We will first argue that G_0 can efficiently prove $\exists\vec{x} Proof_m^S(\vec{x}, [f])$. By definition, the circuit C_m on input α, outputs $[f]$. Therefore it is not hard to see that G_0 has a polynomial-size proof of $Proof_m^S(\alpha, [f])$. This is just a matter of verifying in G_0 that the circuit C_m on input α, outputs $[f]$. Now using the rule for $\exists$, G_0 can derive $\exists\vec{x} Proof_m^S(\vec{x}, [f])$ from $Proof_m^S(\alpha, [f])$, as claimed.

Secondly, we claim that G_0 can prove that $\neg f \rightarrow \exists\vec{z}\neg SAT_{i,m}([f], \vec{z})$. (See [18] for example.) Now combining the above two arguments, it follows that G_0 can efficiently prove $\neg f \rightarrow \neg Refl_m^S$, as desired.

Could it be the case that there is a propositional proof system which effectively simulates all propositional proof systems? This is a possibility, but the construction of such a system would imply the existence of a complete disjoint NP-pair, which is a longstanding open problem [16]. However, perhaps the more interesting question is whether a "natural", well-studied propositional proof system like EF effectively simulates all other propositional proof systems that are "natural" in some sense. We have no evidence in support of or against this possibility.

4 Effectively-p Simulation: Negative Results

In this section we discuss several situations where effectively-p simulations do not seem to be possible.

Our first observation in this direction is as follows.

Claim 4.1 *Let A be a propositional proof system that is automatizable, and let B be another propositional proof system that is not automatizable (under assumptions), then under the same assumptions, A cannot effectively-p simulate B.*

From the above claim, it follows that Tree-Resolution, Nullstellensatz, PC, and low rank LS, LS^+ cannot effectively p-simulate Frege or Extended Frege, under assumptions about hardness of factoring [9, 10, 19].

As a further example, we show that Tree Resolution is unlikely to effectively simulate G_0.

Theorem 4.2 *If $NP \not\subseteq QP$, then Tree Resolution does not effectively-p simulate G_0.*

Proof. Theorem 5.4 in the next section shows that if $NP \not\subseteq P$, then G_0 is not automatizable. The same proof scales to show that if $NP \not\subseteq QP$, then G_0 is not quasi-automatizable. If Tree Resolution effectively-p simulated G_0, then G_0 would be quasi-automatizable, since Tree Resolution is. Thus, under the assumption that $NP \cap coNP \not\subseteq QP$, Tree Resolution cannot effectively-p simulate G_0.

How about if both two proof systems are not automatizable (under reasonable complexity assumptions)? This is the typical case for strong enough proof systems, say bounded-depth Frege or stronger. We can still show a negative result in this case, however one of the proof systems involved is rather "unnatural".

Theorem 4.3 *There is a proposition proof system P such that if Factoring is not in polynomial time infinitely often, then*

1. *EF(Extended Frege) is not automatizable*
2. *P is not automatizable*
3. *P does not effectively-p simulate EF*

Proof Sketch. The basic idea is to define P to be a "sparsified" version of EF in some sense. P will retain enough of the nature of EF that automatizability of P would have unlikely consequences, and yet an effective simulation of EF by P would imply that EF is automatizable infinitely often, which again would have an unlikely complexity consequence. This proof idea is analogous to Ladner's construction [20] of a set in NP that is neither in P nor NP-complete, assuming $NP \neq P$.

We need to define what "sparsified" means. On infinitely many tautology lengths, P will be exactly like EF, however there will be a triply exponential separation between each two consecutive input lengths. On all remaining tautology lengths, P will be exactly like the truth-table proof system, with each tautology having only exponential-size proofs.

Bonet, Pitassi and Raz [10] showed that if EF is automatizable, then Factoring is easy. Their proof also shows that if EF is automatizable on infinitely many tautology lengths, then Factoring is easy infinitely often. Thus, if EF is automatizable or P is automatizable, then Factoring is easy infinitely often.

It remains to be shown that the same conclusion follows if P effectively simulates EF. We focus on tautology lengths n for which P looks like the truth-table proof system for all input lengths between $log(n)$ to 2^n - by definition of P, there are infinitely many of these. Assume, for the sake of contradiction, that there is an effectively polynomial simulation R of EF by P, and let c be a constant such that if f has an EF-proof of size m, then $R(f, m)$ has a P-proof of size m^c. Let f be any tautology of length n. We define a procedure $Q(f, m)$ running in polynomial time such that if f is a tautology of size n with an EF- proof of size at most m, then Q outputs a proof of f (in a different proof system). This implies that EF is automatizable.

$Q(f, m)$ runs $R(f, m)$. If $R(f, m)$ outputs a formula with more than $c \log(m)$ variables, then Q outputs something arbitrary. The point is that in such a case, f *cannot* be a tautology with EF-proofs of size at most m, since the output formula does not have P-proofs of size at most m^c (P looks like the truth-table proof system in this range of lengths), so it does not matter what Q does. On the other hand, suppose that $R(f, m)$ outputs a formula with at most $c \log(m)$ variables. By exhaustive search, Q determines if $R(f, m)$ is a tautology or not. If it is, then Q outputs $R(f, m)$ together with the truth-table proof that $R(f, m)$ is a tautology, otherwise it does something arbitrary.

Since R is tautology-preserving, $R(f, m)$ together with its truth-table proof act as a proof of f in some propositional proof system. It's clear that Q operates in polynomial time and outputs a proof of f whenever f is a tautology of size at most m.

The argument given above works for all f of size n, where n is in the "sparse" range of P, and there are infinitely many n, as we observed. Thus under the assumption that P effectively-p simulates EF, EF is automatizable infinitely often, which means that Factoring is easy infinitely often by the result of Bonet, Pitassi and Raz [10].

4.1 No Effectively-p Simulations under Restricted Reductions

We don't know how to say anything in general about the non-existence of effectively-p simulations between two natural proof systems neither of which

is believed to be automatizable. However, we can say something if we constrain the form of the reduction.

Claim 4.4 *Let P and P' be two propositional proof systems for refuting unsatisfiable CNF formulas, and such that P effectively-p simulates P'. Let A be the polynomial time algorithm that transforms f to f'. Then we can assume without loss of generality that A maps f to $f' = (f \wedge g)$, for some g that depends on f and m.*

Since the reduction is truth preserving, we can always take the conjunction of whatever formula A returns with f. This formula will still preserve satisfiability, and moreover the size of the P-refutation for this new formula will be the same as before.

If $NP = P$, any two proof systems effectively-p simulate each other. Hence we need to put some assumptions on A in order to get negative results without proving that P is different from NP. Next we define a natural restrictions on A. We assume without loss of generality that f is a 3CNF formula in n variables. We slightly abuse notation and say that such an f has size n.

DEFINITION 4.5 *(Oblivious reductions) Let A be a polynomial-time truth-preserving reduction from f, m to $f \wedge g$. A is an oblivious reduction if for all n there exists a unique g such that for all f of size n, $A(f)$ maps to $f \wedge g$. That is, A is oblivious to everything about f except for its size.*

This type of reduction is natural and have been defined and studied in many contexts similar to ours. The intuition behind this restricted definition is that it is hard to determine whether or not f is satisfiable, and that basically no useful information can be obtained about an arbitrary f in polytime, just by looking at f.

Now assume that A is an oblivious reduction mapping f to $f \wedge g$. We can assume without loss of generality that g is also a CNF formula. g is a formula involving the original variables of f, call them $\vec{x}$, plus new variables $\vec{y}$. Furthermore, it must be the case that for every assignment α to the variables of f, there exists an assignment β to the new variables of g such that $g(\alpha, \beta)$ is true. This is because the reduction is oblivious. Assume for sake of contradiction that there is an assignment α to the x variables such that for all β, $g(\alpha, \beta)$ is false. Fix some f of size n such that $f(\alpha)$ is true. Then A is not truth preserving on input f. Thus g has the property that for every α, there exists a β such that $g(\alpha, \beta)$ is true. Note that this implies that each clause of g must involve at least one new variable.

Other reasonable assumptions are as follows.

DEFINITION 4.6 *Let A be a polynomial-time truth-preserving reduction from f, m, to $f \wedge g$. Let $\vec{x}$ be the original set of variables underlying f, and let g be a CNF over the $\vec{x}$ variables, plus new variables, $\vec{y}$. A is symmetric if for all permutations π of x, there is a permutation π' to y such that $g(\vec{x}, \vec{y}) = g(\pi(\vec{x}), \pi'(\vec{y}))$. A is extensional if for each assignment to $\vec{x}$, there is exactly one assignment for $\vec{y}$ such that $g(\vec{x}, \vec{y})$ is true.*

All of our positive results for effectively-p simulations excepting those based on automatizability are oblivious, symmetric and extensional.

Our next results use only the symmetric and extensional restrictions. We will need the following amazing theorem of Clote and Krannakis [13], later generalized in [3].

Theorem 4.7 *(Clote, Kranakis) Let $f = \{f_n \mid n = 1, 2, \ldots\}$ be a boolean function, where f_n denote the function in ouputs of length n. For each n, we define an equivalence relation on the set of all permutations of $\vec{x}$ as follows. Let π_1 and π_2 be two permutations of $\vec{x}$ Then $\pi_1 \equiv \pi_2$ if and only if $f(\pi_1(x))$ is isomorphic to $f(\pi_2(x))$. We will say that the function f_n is k-symmetric if the number of equivalence classes for f_n is k. So if f_n is a truly symmetric function, then it is 1-symmetric. We say that f is poly-symmetric if there exists a constant k such that for all sufficiently large n, the number of equivalence classes is at most n^k. If f is poly-symmetric, then f is an NC_1 function.*

Theorem 4.8 *Assume that our reduction is symmetric and extensional. Then Frege effectively-p simulates Extended Frege if and only if Frege p-simulates Extended Frege.*

Proof. Let A be a symmetric, extensional reduction, mapping f to $f \wedge g$. Since A is extensional g defines a set of boolean functions $H = \{h_1, \ldots, h_l\}$, using using extension variables. For each such function, we must have all symmetric versions of it defined in g. Since g is polynomial size, this implies that each h is poly-symmetric. Now by the above theorem, this implies that each $h_i \in H$ is an NC_1 function. But this implies that Frege can efficiently prove $f \wedge g$ if and only if Frege can efficiently prove f. But this implies that Frege (by itself, with no advice "g") can p-simulate EF.

Finally, we can prove that if the reduction is extensional and has low communication complexity, then neither tree-like Cutting Planes nor sublinear width Resolution can effectively p-simulate Frege. Note that

the restriction on communication complexity is essential. Since we are not insisting in this result that reductions are efficiently computable, if there is no restriction on the communication complexity, Resolution *can* simulate Frege by extensional reductions, using Remark 3 in Section 2.

Theorem 4.9 *Suppose that our reduction A is an extensional reduction, mapping f to $f \wedge g$, and such that all functions defined by g have communication complexity at most n^ϵ for some $\epsilon < 1$. Then such a reduction will not give an effectively-p simulation for Frege systems by tree-like Cutting Planes, or small width Resolution*

Proof. (sketch) We follow the proof of [10]. Let f be the clique-coclique interpolant statement as in that paper, over n variables in total. The formula has the form $Clique(x,y) \wedge coClique(x,z)$, where $Clique(x,y)$ states that y is a subset of k vertices in the graph x (on n vertices) that forms a clique, and $Coclique(x,z)$ states that z is a partition of the n vertices of x into $k+1$ sets such that no edges exist between the sets. These statements have polynomial-size Frege proofs. Now suppose that A maps f to $f \wedge g$, where g defines a set of new functions of low communication complexity. Assume for sake of contradiction that A works. Since f has short Frege proofs, $f \wedge g$ should have short tree-like CP (Resolution) proofs. On the other hand, since g defines functions that have small communication complexity, we can still apply the feasible interpolation argument using the proof from [10]. That is, we can build a monotone circuit of small size takes as input an assignment to the x variables (a graph) and that says "1" if the graph contains a k-clique, and says "0" if the graph contains a $k+1$-cocliques, violating known monotone circuit lower bounds. Thus we reach a contradiction from the existence of such a reduction A.

5 Effectively Polynomial Simulations and Automatizability

In this section, we use what we know about effective simulations to draw conclusions about automatizability.

First, we use some of our observations earlier to give evidence that automatizing Linear Resolution might be hard.

Proposition 5.1 *If Res(k) is not automatizable for some k, then Linear Resolution is not automatizable.*

Proof. By Theorem 3.2, Linear Resolution effectively simulates Resolution. By Lemma 3.7, Resolution effectively simulates Res(k) for any constant k. By transitivity of effective simulations, Linear Resolution effectively simulates Res(k). By the connection between automatizability and effective simulations, we get the statement in the proposition.

Alekhnovich and Razborov showed that Resolution is not strongly automatizable unless the parameterized class W[P] is tractable. From the fact that Theorem 3.2 actually gives a strong effective simulation, we derive the following corollary to their result.

Corollary 5.2 *Linear Resolution is not strongly automatizable unless W[P] is tractable.*

Next, we try to say something more general about how automatizability of proof systems relates to the NP vs P question.

Lemma 5.3 *If $NP! = P$, then there is a propositional proof system that is not automatizable.*

Proof. Consider the propositional proof system A from $\{0,1\}^*$ to $T\hat{A}UT$ defined as follows:
$A(\langle \phi, 0w \rangle) = \phi \vee (\neg\phi)$ if w is a satisfying assignment to ϕ,
$A(\langle \phi, 1^{2^{|\phi|}} \rangle) = \phi$ if ϕ is a tautology,
$A(z) = 1$ for all other z.

First we show that A is indeed a propositional proof system. A is polynomial-time computable since we can check in time exponential in the length of a formula whether the formula is a tautology or not. A is onto since every tautology ϕ has pre-image $\langle \phi, 1^{2^\phi} \rangle$.

Next, we prove that if there is an automatization procedure F for A, then SAT can be solved in polynomial time. Assume that $F(z)$ runs in time N^k, where N is the size of the smallest proof for z in proof system A. Our algorithm to solve SAT is simple: Given input ϕ, run F on $\phi \vee (\neg\phi)$ for $(2|\phi|)^k$ steps. If F halts within that time, then output "yes", otherwise output "no".

The correctness of this algorithm follows from the fact that $\langle \phi \vee (\neg\phi) \rangle$ has proofs of size at most $2|\phi|$ according to A iff ϕ is satisfiable.

We show how to use the results of previous sections to show that in some sense, G_0 is "universal" in terms of automatizability, i.e., if G_0 is automatizable, so are all quantified proof systems. Moreover, the automatizability of G_0 is equivalent to $NP = P$.

Theorem 5.4 *The following four statements are equivalent:*

1. *G_0 is automatizable*
2. *All propositional proof systems are automatizable*
3. *$NP = P$*

4. All quantified proof systems are automatizable

Proof.

We show (1) implies (2) implies (3) implies (4) implies (1).

(1) implies (2): This follows from the fact that every propositional proof system is effectively simulated by G_0, using the connection between effective simulations and automatizability.

(2) implies (3): This follows from Lemma 5.3.

(3) implies (4): Let A be a quantified proof system. Using the assumption that $NP = P$, we define a procedure F that outputs A-proofs for valid formulae in time polynomial in the size of the smallest A-proof. Let ϕ be a valid formula given as input to F, and let $n = |\phi|$. We define F as a polynomial-time procedure with an NP oracle L, but from the assumption that $NP = P$, it follows that F itself can be implemented in polynomial time. The NP-oracle L is defined as follows: $\langle \phi, 1^m, w \rangle \in L$ iff there is an A-proof of ϕ of size at most m with prefix w. F first sets an internal parameter m to be equal to n. It queries its NP oracle with $\langle \phi, 1^m, \epsilon \rangle$. If the query answers yes, then it uses self-reducibility to find the lexicographically smallest proof of size at most m, using the NP-oracle L to search for the proof. If the query answers no, it sets $m < -2m$, and repeats the process. Since every tautology has a proof in A, this process will eventually terminate. By definition of L, the procedure actually halts and outputs an A-proof for ϕ in time that's polynomial in the smallest A-proof for ϕ.

(4) implies (1): This is immediate since G_0 is a quantified proof system.

6 Discussion

There are many research directions worthy of exploration. In this paper, we have given several examples of effectively polynomial simulations. It would be interesting to generalize these results and provide a high-level framework which would facilitate the discovery of further examples. More challenging is to find new lower bound techniques to rule out the possibility of effectively-p simulations in specific cases. We highlight several problems below.

- Resolve (unconditionally) whether or not Resolution can obliviously effectively-p simulate a stronger proof system such as Frege or Extended Frege (or even AC^0-Frege) A positive result would be quite surprising, and as mentioned in the introduction, could allow us to prove lower bounds for stronger proof systems by proving Resolution lower bounds for specific unsatisfiable formulas. On the other hand, a negative result seems to require extending lower bound techniques for Resolution. In either case, a new and very interesting lower bound would be established.
- We proved that if one proof system A is automatizable, and another proof system B is not (under assumptions), then A does not effectively-p simulate B (under the same assumptions). We would like to know if the same implication holds for the weaker notion of feasible interpolation. That is, prove (or disprove) the following conjecture: If A has feasible interpolation, and B does not (under assumptions) then B does not effectively-p simulate A (under same assumptions). A proof would show, under complexity assumptions, that Resolution cannot effectively-p simulate AC^0-Frege, Frege, or Extended Frege.
- Resolve whether or not Frege can effectively-p simulate Extended Frege. We conjecture that such a simulation is not possible. Note that a negative answer will require some assumption(s) since an effectively-p simulation would exist if $NP = P$. Resolving the question even for extensional reductions would also be very interesting.

Acknowledgement

We thank Albert Atserias, Jakob Nordstrom, Phuong Nguyen, Iddo Tzammeret and Avi Wigderson for useful discussions.

References

[1] A. Atserias and M. L. Bonet. On the automatization of resolution and related propositional proof systems, 2004.

[2] A. Atserias, N. Galesi, and P. P. Monotone simulations of nonmonotone proofs. *Journal of Computer and System Sciences*, 65(4):626–638, 2002.

[3] L. Babai, R. Beals, and P. Takacsi-Nagy. Symmetry and compexity. In *Proceedings 24th STOC*, pages 438–449, 1992.

[4] F. Bacchus, P. Hertel, T. Pitassi, and A. Van Gelder. Clause learning can effectively psimulate resolution. In *Proceedings from AAAI 2008*.

[5] R. J. Bayardo and J. D. Pehoushek. Counting Models using Connected Components. In *Proceedings of the AAAI National Conference (AAAI)*, pages 157–162, 2000.

[6] P. Beame, D.-T. Huynh-Ngoc, and T. Pitassi. Hardness amplification in proof complexity. Manuscript,2009.

[7] P. Beame, H. Kautz, and A. Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, 2004.

[8] P. Beame and T. Pitassi. Propositional Proof Complexity: Past, Present, and Future. In G. Paun, G. Rozenberg, and A. Salomaa, editors, *Current Trends in Theoretical Computer Science: Entering the 21st Century*, pages 42–70. World Scientific Publishing, 2001.

[9] M. Bonet, C. Domingo, R. Gavalda, A. Maciel, and T. Pitassi. Non-automatizability of bounded-depth Frege proofs. In *ProceedingsFourteenth Annual IEEE Conference on Computational Complexity (formerly: Structure in Complexity Theory Conference)*, pages 15–23, Atlanta, GA, May 1999.

[10] M. Bonet, T. Pitassi, and R. Raz. On interpolation and automatization for frege systems. *SIAM Journal of Computing*, 29(6):1939–1967, 2000.

[11] J. Buresh-Oppenheim and T. Pitassi. The complexity of resolution refinements. *Journal of Symbolic Logic*, 72(4):1336–1352, 2007.

[12] M. Clegg, J. Edmonds, and R. Impagliazzo. Using the Gröbner basis algorithm to find proofs of unsatisfiability. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pages 174–183, Philadelphia, PA, May 1996.

[13] P. Clote and E. Kranakis. Boolean functions, invariance groups, and parallel complexity. *SIAM Journal of Computing*, 20:553–590, 1991.

[14] S. A. Cook and R. A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, 1977.

[15] J. Davies and F. Bacchus. Using more reasoning to improve #SAT solving. In *Proceedings of the AAAI National Conference (AAAI)*, pages 185–190, 2007.

[16] C. Glasser, A. Selman, and L. Zhang. Survey of disjoint NP-pairs and relations to propositional proof systems. In *Theoretical Computer Science, Essays in Memory of Shimon Even.*, pages 241–253. Springer, 2006.

[17] A. Hertel, P. Hertel, and A. Urquhart. Formalizing dangerous sat encodings. In *Proceedings from SAT 2007*, pages 159–172.

[18] J. Krajíček and P. Pudlák. Quantified propositional calculi and fragments of bounded arithmetic. 36(1), 1990.

[19] K. Krajíček and P. Pudlák. Some consequences of cryptographic conjectures for S_2^1 and EF. In D. Leivant, editor, *Logic and Computational Complexity: international workshop, LCC '94*, volume 960 of *Lecture Notes in Computer Science*, pages 210–220. Springer-Verlag, 1995.

[20] R. E. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, 22(1):155–171, Jan. 1975.

[21] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. In *Proc. of the Design Automation Conference (DAC)*, 2001.

[22] P. Pudlák. On reducibility and symmetry of disjoint np pairs. *Theoretical Computer Science*, 295:323–339, 2003.

[23] A. A. Razborov. On provably disjoint NP-pairs. Technical Report RS-94-36, BRICS, 1994.

[24] Z. Sadowski. On an optimal quantified propositional proof system and a complete language for NP cap co-NP. In *Proceedings of the Eleventh International Symposium on Fundamentals of Computation Theory*, pages 423–428, 1997.

[25] T. Sang, F. Bacchus, P. Beame, H. A. Kautz, and T. Pitassi. Combining component caching and clause learning for effective model counting. In *Theory and Applications of Satisfiability Testing (SAT)*, 2004.

[26] N. Segerlind. The complexity of propositional proofs. *Bulletin of Symbolic Logic*, 13(4):482–537, 2007.

[27] M. Soltys and S. Cook. The proof complexity of linear algebra. *Annals of Pure and Applied Logic*, 130:277–323, 2004.

[28] A. Urquhart. The complexity of propositional proofs. *Bulletin of Symbolic Logic*, 1(4):425–467, Dec. 1995.

[29] A. Van Gelder. Pool resolution and its relationship to regular resolution and dpll with clause learning. In *In Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, pages 580–594, 2005.