Automatization and Non-Automatizability

Tobias Lieber

July 11, 2009

In this abstract of my talk I present results on the automatizability of proof systems. Besides the knowledge of the existence of long proofs in proof systems one is interested if there are algorithms which can calculate such proofs in a fast way. In the first part I present a result which excludes such fast algorithms unless a complexity theoretic assumption holds. In the second part a connection between the Resolution and Res(k)-proof systems and its automatiziablity is shown.

1 Introduction

In propositional proof complexity, many results are results on lower bounds for the size of proofs in proof systems. In automatizability of proof systems one is interested if a proof of a propositional formula can be computed efficiently. It is clear that the size of proofs in a proof system is not a good measure since there are exponential lower bounds for the size of proofs for many important proof systems. This implies existence of formulae for which every algorithm needs at least exponential time to write the proof.

Therefore a better characterization for the efficiency of automatizability algorithms is, the time an algorithm actually needs to find a proof for a tautology in depenece of the shortest proof:

Definition 1.1. *A proof system P is automatizable if there is a deterministic algorithm which returns in polynomial time of the shortest P-proof of a tautology τ its P-proof.*

Definition 1.2. *A proof system P is weakly automatizable if there is a proof system S that p-simulates P and is automatizable.*

2 Non-Automatizability of Resolution

To proof non-automatizability of Resolution (under a complexity theoretic assumption) we need some tools to deal with $\mathcal{NP}$-complete problems. One tool is fixed parameter tractability which tries to decide hard problems exactly (and thus not polynomial time unless $\mathcal{P} \neq \mathcal{NP}$). We focus on this later.

2.1 Preliminaries

Another common tool to deal with $\mathcal{NP}$-completeness, is to find polynomial time algorithms which approximate the exact solution and thus possibly gives false positive answers. To measure those failures the approximation ratio is a common tool. It compares the worst failure an approximation algorithm can do to the optimal solution.

Definition 2.1. *The approximation ratio ρ of an algorithm for an optimization problem is defined by*

$$\rho := \max\left\{\frac{OPT(A)}{OPT}, \frac{OPT}{OPT(A)}\right\}.$$

There are many types of apprximability e.g., constant factor approximations, which yield approximation results which are at most a constant factor worse than the optimal solution (VertexCover). But there are also problems like Clique known which hardly can be approximated unless $\mathcal{P} = \mathcal{NP}$. On the other hand there are problems which can be approximated arbitrary good if we give the algorithms just enough (polynomial) time:

Definition 2.2. *An optimization problem has a polynomial time approximation scheme (PTAS), if there is an algorithm, which for every $\epsilon > 0$ computes, in time of at most $n^{O(\frac{1}{\epsilon})}$, an $(1+\epsilon)$-approximation.*

Definition 2.3. *An optimization problem has an efficient polynomial time approximation scheme (EPTAS), if there is an algorithm, which for every $\epsilon > 0$ computes, in time of at most $f(\frac{1}{\epsilon})p(n)$, an $(1+\epsilon)$-approximation (p a polynomial, f computable).*

Those classes of problems are related to fixed parameter tractability as we see later:

2.2 Parametrized Complexity

In parametrized complexity the restriction to polynomial time algorithms is relaxed but an exact solution of a decision problem is required. A problem instance in parametrized complexity consists of a classical input as it is known from classical complexity theory and an integer which restricts the elements contained in a language L (with still infinite elements in L).

Definition 2.4. *$\mathcal{FPT}$ consists of all languages $L \subseteq \Sigma^* \times \mathbb{N}$ for which an algorithm Φ exists, a constant c and a recursive function $f : \mathbb{N} \to \mathbb{N}$ such that:*

- *the running time of $\Phi(x,k)$ is at most $f(k)|x|^c$*
- *$(x,k) \in L$ iff $\Phi(x,k) = 1$*

The class $\mathcal{FPT}$ can be seen as an equivalent to $\mathcal{P}$ in parametrized complexity. Thus a natural question is if there is an analogue to $\mathcal{NP}$ in parametrized complexity. This analogon is defined through a complete problem via a special reduction preserving the properties of parametrized complexity classes. Since we do not deal further with this type of reduction we omit the definition of the reduction and proceed with the definition of the $\mathcal{NP}$-analogue in parametrized complexity:

Definition 2.5. *The class $\mathcal{W}[\mathcal{P}]$ contains all the problems which can be parametrized reduced to weighted circuit satisfiability:*
Input: A circuit C and an positive integer k.
Question: Is there a satisfying assignment of C with k ones?

One central lemma we use connects approximation algorithms with the the class $\mathcal{FPT}$:

Lemma 2.6. *If a problem A has an $EPTAS$ then A is in $\mathcal{FPT}$.*

In this paper we especially consider a problem which is complete in $\mathcal{W}[\mathcal{P}]$ (via parametrized reduction). We use this problem in its optimization version to construct a PTAS. The problem is defined on monotone circuits i.e., circuit containing only "and" or "or" gates.

Definition 2.7. *The problem monotone minimum circuit satisfying assignment (MMCSA) is an optimization problem with a circuit C with n variables as input.*
Objective function: $\sigma(a)$ which returns the number of ones in an assignment $a \in \{0,1\}$ such that $C(a) = 1$.

The optimal solution of MMCSA is definied as:

Definition 2.8.

$$\sigma(C) = \min_{a \text{ is solution of } MMCSA} \sigma(a)$$

Up to now there is no direct method to convert the constructed PTAS to an EPTAS. Thus we use a randomized construction. Therefore we define an equivalent version of $\mathcal{R}$ in parametrized complexity:

Definition 2.9. *The class $\mathcal{FPR}$ of parametrized problems consists of all languages $L \subseteq \Sigma^* \times \mathbb{N}$ for which there is a probabilistic algorithm Φ, a constant c and a recursive function $f : \mathbb{N} \to \mathbb{N}$ such that:*

- *$\Phi(x,k)$ runs in at most $f(k)|x|^c$*
- *if $(x,k) \in L$ then $Pr[\Phi(x,k) = 1] \geq \frac{1}{2}$*
- *if $(x,k) \notin L$ then $Pr[\Phi(x,k) = 1] = 0$*

As in classical complexity theory it holds

$$\mathcal{FPT} \subseteq \mathcal{FPR}.$$

Since $\mathcal{W}[\mathcal{P}]$ is the $\mathcal{NP}$ analogue it follows intuitively:

$$\mathcal{FPT} \subseteq \mathcal{W}[\mathcal{P}].$$

Another important tool which we use for enlarging the gap between an optimal solution and the worst case guaranteed solution by an approximation algorithm is given by the Selfimprovement Lemma:

Lemma 2.10 (Selfimprovement)**.** *For every fixed integer $d \geq 1$ there exists a polynomial time computable function π which maps monotone circuits into monotone circuits with $\sigma(\pi(C)) = \sigma(C)^d$ for all C and n^d input nodes if C has n input nodes.*

Proof. Simply consider a tree of circuits of depth, which connects each output of C to one input of a C in a higher level. The Lemma follows immediately. □

2.3 Results

The central goal is the following theorem:

Theorem 1. *If Resolution or tree-like Resolution is automatizable then $\mathcal{W}[\mathcal{P}] \subseteq$ co-$\mathcal{FPR}$.*

Similar to classical complexity theory it is believed that $\mathcal{W}[\mathcal{P}] \subseteq$ co-$\mathcal{FPR}$ does not hold. Thus Theorem 1 implies that if $\mathcal{W}[\mathcal{P}] \not\subseteq$ co-$\mathcal{FPR}$ Resolution is not automatizable.

As a central lemma we use the following lemma without a proof, which maps every possible input of the MMCSA problem to an unsatisfiable CNF formula such that a contradiction can be found within an upper bound and a minimal size.

Lemma 2.11. *[[AR01]] There exists a polynomial time computable function τ which maps any pair $(C, 1^m)$, with a monotone circuit C and an integer m, to an unsatisfiable CNF $\tau(C, m)$ such that:*

$$S_T(\tau(C, m)) \leq |C| m^{O(\min\{\sigma(C), \log m\})}$$

and

$$S(\tau(C, m)) \geq m^{\Omega(\min\{\sigma(C), \log m\})}.$$

As mentioned above the proof consists of two parts:

1. Creating a PTAS for MMCSA under the assumption that (tree-like) Resolution is automatizable.
2. Create an EPTAS of the PTAS by randomization.

For the construction of the PTAS for MMCSA we give an algorithm which has approximation ratio h.

Lemma 2.12. *If Resolution or tree-like Resolution is automatizable then a constant $h > 1$ exists, as well as an algorithm Φ working on pairs (C, k), where C is a monotone circuit and k is an integer such that:*

- *the running time of $\Phi(C, k)$ is at most $\exp(O(k^2))|C|^{O(1)}$*
- *if $\sigma(C) \leq k$ then $\Phi(C, k) = 1$*
- *if $\sigma(C) \geq hk$ then $\Phi(C, k) = 0$.*

Proof. We consider the algorithm Φ which constructs a CNF from C and an integer r (which is yet not defined). Afterwards it Resolution is simulated on the obtained CNF and stops this simulation after $(|C|r^k)^{h_0}$ steps. The algorithm outputs 1 if the steps the simulation took ($:= S(C,r)$) is at most $(|C|r^k)^{h_1}$ and 0 otherwise.

By Lemma 2.11 we know that $S(C,r)$ has an upper and a lower bound:

$$r^{\epsilon \min\{\sigma(C),\log r\}} \leq S(C,r) \leq \left(|C|r^{\min\{\sigma(C),\log r\}}\right)^{h_1} \tag{1}$$

with constants $\epsilon, h_0, h_1 > 0$. The constant h from the lemma is chosen such that

$$\frac{h_1}{\epsilon}(h+1) < h^2 \tag{2}$$

holds. We define the last parameter:

$$r := 2^{h \max\{k, \frac{\log |C|}{k}\}} \tag{3}$$

By the selection of the parameters and by the fact that we simulate at most $(|C|r^k)^{h_0}$ steps of the refutation of the constructed CNF, it is clear that the runtime is roughly bound by $r^k \leq \exp(O(k^2))|C|^{O(1)}$.

Thus we have to check the other two requirements: In general it is clear that $\log m \geq hk > k$ holds. The first case is $\sigma(C) \leq k$:

The restriction to the number of simulated steps is less stringent than the restriction to the maximal accepted value of $S(C,r)$. Therefore it follows immediately that the algorithm outputs 1 if $\sigma(C) \leq k$ and the second requirement of the Lemma is fulfilled.

The last case to check is $\sigma(C) \geq hk$. Since it still holds $\log r \geq hk$, $S(C,m) \geq r^{\epsilon hk}$ can be easily deduced. Therefore it is left to show that $r^{\epsilon hk} > (|C|r^k)^{h_1}$. We distinguish the two cases of (3).

In the first case we assume $r = 2^{hk}$:

$$r^{\epsilon hk} = 2^{\epsilon h^2 k^2} \overset{(2)}{>} 2^{h_1(h+1)k^2} = (2^{k^2 h} \cdot 2^{k^2})^{h_1} = (r^k \cdot 2^{k^2})^{h_1} \tag{4}$$

Since $r = 2^{hk}$ we know $k^2 \geq \log |C|$ and it holds:

$$(r^k \cdot 2^{k^2})^{h_1} \geq (r^k \cdot 2^{\log |C|})^{h_1} = (r^k \cdot |C|)^{h_1} \tag{5}$$

In the second case we have to assume $r = 2^{\frac{h \log |C|}{k}} = |C|^{\frac{h}{k}}$:

$$r^{\epsilon hk} = 2^{\frac{\epsilon h^2 k \log |C|}{k}} = |C|^{\epsilon h^2} \overset{(2)}{>} |C|^{h_1(h+1)} = (|C| \cdot |C|^{h\frac{k}{k}})^{h+1} = (|C| \cdot r^k)^{h+1} \tag{6}$$

Thus by definition of Φ it outputs 0 in the third case and the proof of Lemma 2.12. □

Theorem 2. *If Resolution or tree-like Resolution is automatizable then for any fixed $\epsilon > 0$ there exists an algorithm Φ receiving as input a monotone circuit C which runs in time $\exp(\sigma(C)^{O(1)})|C|^{O(1)}$ and approximates $\sigma(C)$ within a factor $1+\epsilon$.*

Proof. From the last lemma we can construct an approximation algorithm with approximation ratio h:
Compute $\Phi(C,1)\dots\Phi(C,l)$ as long as $\Phi(C,l)\neq 0$ and return l if $\Phi(C,l)=0$.

With the Selfimprovement Lemma we can transform this h-approximation easily into an $1+\epsilon$-approximation algorithm by choosing $d=\lceil\frac{1}{\epsilon}\ln h\rceil$. □

So far we have an algorithm which is an PTAS. The main problem is, that for a more exact solution we have to create a bigger circuit. To proof Theorem 1 we therefore replace the construction of a bigger circuit via usage of randomness by a circuit of polynomial size in n and k.

Proof of Theorem 1. For the following construction we need that for a circuit in n variables and the parameter k,

$$10 \leq k \leq \epsilon(\left(\frac{\log n}{\log\log n}\right)^2 \tag{7}$$

is fulfilled. This is not a real restriction. If $k \leq 10$ we simply use the brute force algorithm which is allowed since this concerns only finitely many k. The restriction $k \leq \epsilon\frac{\log n}{\log\log n}^2$ is equivalent to $n^{(2^{\sqrt{k}})} \leq 2^n$. This holds for for every fixed k for initially many n. Especially there is an m such that for all $n \in \mathbb{N} : n > m$ the equation holds. We can fix those circuits which do not fulfill (7) by adding m dummy inputs which are all connected by an AND, which is ORed with the ouput of the original circuit.

To proof the theorem we construct a (randomized) circuit $\beta(C,k)$ and $\alpha(k)$ in polynomial time depending only on C and k such that the following holds:

$$\begin{aligned}\sigma(C) \leq k \Rightarrow Pr[\sigma(\beta(C,k)) \leq \alpha(k)] &= 1\\ \sigma(C) \geq k+1 \Rightarrow Pr[\sigma(\beta(C,k)) \geq 2\alpha(k)] &\geq \tfrac{1}{2}\end{aligned}$$

Since the gap between the accepting and the rejecting condition is very small, we use the Selfimprovement Lemma with $d=2$ to obtain a larger gap. With $s := k^2$ we search a construction of $\beta(C,k)$ and $\alpha(k)$ such that

$$\sigma(C) \leq s \Rightarrow Pr[\sigma(\beta(C,s)) \leq \alpha(s)] = 1 \tag{8}$$
$$\sigma(C) \geq s+2\sqrt{s} \Rightarrow Pr[\sigma(\beta(C,s)) \geq 2\alpha(s)] \geq \tfrac{1}{2} \tag{9}$$

holds.

The construction of the randomized circuit $\beta(C,s)$ is easy. It consists of $d := \sqrt{s}$ layers with each $N := n^3$ copies of the circuit C. Therefore each layer has n^4 inputs. Every input of layer $i+1$ is randomly connected to a output of layer i. The output of $\beta(C,s)$ is the output of a random chosen C in the $\sqrt{s}$-th layer. Additionally we define $\alpha(s) := s^{\sqrt{s}}$.

Thus we have to check the two requirements. We begin with requirement (8). It is easy to see that $\sigma(\beta(C,s))$ at level i is $\leq s^{d+1-i}$. Therefore in the first level it is $\leq \alpha(s)$ and (8) is fulfilled.

To proof (9) takes more effort. We use the fact that a random bipartite graph between the layers is a very good expander. This means

$$P[\text{A set of } b \text{ circuits in a layer has } \leq bn - a \text{ input circuits}] \leq N^s \left(\frac{4b^2n^2}{N}\right)^a \qquad (10)$$

which is proven in detail in [AR01].

We proof (9) by two steps. We show that the probability, that the circuit obtained by the randomized function which do not directly imply $\sigma(\beta(C,s)) \geq 2\alpha(s)$ occur rarely. In a second step we show that the other circuits imply $\sigma(\beta(C,s)) \geq 2\alpha(s)$ if $\sigma(C) \geq s+2\sqrt{s}$ holds.

We define $s_i := (s+\sqrt{s})^{d-i}$, which is a measure of how many outputs in layer d have to be satisfied. In the first step we show that the number of $\beta(C,s)$ which have a layer $i+1$ which contains a set of s_{i+1} original circuits which have a small number of sources ($\leq s_{i+1}(n-\sqrt{s})$) in layer i is small enough. We call a circuit having this property is called bad.

To proof this we use the expander property (10) of the circuit. Thus the sum over all bad graphs i.e., the sum over probabilities of layers which have a small number of sources is:

$$\begin{aligned} P[\beta(C,s) \text{ is bad}] &\leq \sum_{i=1}^{d-1} N^{s_{i+1}} \left(\frac{4s_{i+1}^2 n^2}{N}\right)^{s_{i+1}\sqrt{s}} \\ &= \sum_{i=1}^{d-1} \left(\frac{4s_{i+1}^2}{n^{1-3/\sqrt{s}}}\right)^{s_{i+1}\sqrt{s}} \overset{(7)}{\leq} \sum_{i=1}^{d-1} \left(\frac{1}{3}\right)^{s_{i+1}\sqrt{s}} \leq \frac{1}{2} \end{aligned}$$

Thus we know that the fraction of $\beta(C,s)$ which are good is at least $\frac{1}{2}$. Now we check that these graphs imply $\sigma(\beta(C,s)) \geq 2\alpha(s)$ if $\sigma(C) \geq s+2\sqrt{s}$. For the last layer it is clear that there is $1 = s_d$ output which we want to be true, the output which was randomly selected. Therefore at layer $i+1$ we need at least $(s+2\sqrt{s})s_{i+1}$ inputs at layer $i+1$ satisfied. Since our graph is good, we know that there are at most $\sqrt{s} \cdot s_{i+1}$ connections between layer i and layer $i+1$ which have the same sources in layer i. Thus, to satisfy $(s+2\sqrt{s})s_{i+1}$ inputs at layer $i+1$ we need to satisfy at least $(s+2\sqrt{s})s_{i+1} - s_{i+1}\sqrt{s} = s_i$ outputs of layer i. This means in layer 1 there have to be $(s+\sqrt{s})^{d-1}$ outputs satisfied, which implies at least $(s+2\sqrt{s})(s+\sqrt{s})^{d-1} > 2\alpha(s)$ inputs.

Thus (9) is proven and therefore the proof of Theorem 1 is complete. □

There is just on final note: In 2008 it was shown that the same result holds for polynomial calculus, too [GL09].

3 Resolution and Res(k)

We have just shown that Resolution is probably not automatizable. Now we want to connect this result to interpolation, which can be used to show lower bounds for a proof system. The Res(k)-proof system works with k-disjunctions. Each k-disjunction is a disjunction of arbitrary many k-terms. Where each k-term is a conjunction of up to k literals. Res(k) has three inference rules: Weakening $\frac{A}{A \vee B}$, $\wedge$-Introduction $\frac{A \vee l_1 \quad B \vee (l_2 \wedge ... \wedge l_s)}{A \vee B \vee (l_1 \wedge ... \wedge l_s)}$ and Cut: $\frac{A \vee (l_1 \wedge ... \wedge l_s) \quad B \vee \neg l_1 \vee ... \vee \neg l_s}{A \vee B}$.

3.1 Preliminaries

To show the result I introduce some basics which connect Resolution to Res(2). The first definitions simplify the transformation of Res(k) formulas to Resolution formulae.

Definition 3.1. *The variable $z_{l_1,...,l_s}$ of variables $l_1, \ldots, l_2$ is constituted by its defining clauses:*

$$\neg z_{l_1,...,l_s} \vee l_i \quad \forall i \in [s]$$
$$z_{l_1,...,l_s} \vee \neg l_1 \vee \cdots \wedge \neg l_s$$

It can be interpreted as $l_1 \wedge \cdots \wedge l_s$.

Definition 3.2. *The set C_k of a set of clauses C is the union of C with all the defining clauses for the variables $z_{l_1,...,l_s}$.*

Definition 3.3. *The set $REF(S)$ is the set of pairs (C, m) with an CNF formula C that has an S-refutation with size m.*

The set SAT^* contains the pairs (C, m) such that C is a satisfiable CNF formula.

$(REF(S), SAT^*)$ is called the canonical pair of S.

A canonical pair is separable if there is an algorithm running in polynomial time and returns $false$ on every input from $REF(S)$ and $true$ if (C, m) is in SAT^*.

Definition 3.4 (Reflection Principle)**.** *A CNF formula which is true iff*

- *z encodes a truth assignment of a CNF x*
- *x is of size r and uses n variables*

is called $SAT_n^r(x, z)$.

Let us call a CNF $REF_{r,m}^n(x, y)$ if it evaluates to true iff

- *y encodes an S-refutation of a CNF x*
- *the size of the refutation is m*
- *x is of size r and uses n variables*

The collection of the CNFs $REF^n_{r,m}(y,z) \wedge SAT^n_r(x,z)$ is the Reflection Principle of S.

Definition 3.5. *A proof system S has the interpolation property in time $T = T(m)$ if there is an algorithm which runs in time T and decides for an contradictory CNF $B := A_0(x,y_0) \wedge A_1(x,y_1)$ (x, y_0, y_1 are disjoint sets) if $A_0(x,y_0)$ or $A_1(x,y_1)$ is contradictory where m is the minimal size of an refutation of B.*

If $T(m)$ is polynomial in m then S has feasible interpolation.

Theorem 3 (Pudlak)**.** *If the reflection principle of S has polynomial sized refutations in a proof system that has feasible interpolation, then the canonical pair for S is separable in polynomial time.*

3.2 Results

Now we can show that, if there is a refutation for Res(k) of size S, then there is a refutation in Resolution on the modified clause set C_k which is not too large.

Lemma 3.6. *If the set of clauses C has a Res(k) refutation of size S, then C_k has a Resolution refutation of size $O(kS)$. If the Res(k) refutation is tree-like, then the Resolution refutation is also tree-like.*

Proof. To obtain from a Res(k)-refutation for the set of clauses C a refutation for Resolution in C_k we transform every k-disjunction into a clause. We simply replace each k-term $l_1 \wedge ... \wedge l_s$ with $s \leq k$ by its defining clause $z_{l_1 ... l_s}$. To complete the proof we have to check that every inference rule of Res(k) can be represented tree-like in Resolution and is of size O(k).

Cut rule:

$$\frac{A \vee (l_1 \wedge .. \wedge l_s) \quad B \vee \neg l_1 \vee ... \vee \neg l_s}{A \vee B}$$

The corresponding clauses are $A \vee B$,$A \vee z_{l_1 ... l_s}$ and $B \vee \neg l_1 \vee ... \vee \neg l_s$. Latter and $\neg z_{l_1,...,l_s} \vee l_i$ from C_k can be resolved tree-like to $B \vee \neg z_{l_1,...,l_s}$. Which resolves with $A \vee z_{l_1,...,l_s}$ to $A \vee B$. In total, the size of the new tree-like refutation is in $O(k)$.

$\wedge$-Introduction: Notice that there is a tree-like proof of $\neg l_1 \vee \neg z_{l2,...,l_s} \vee z_{l_1,...,l_s}$ in C_k. Thus via the translated clauses $A \vee l_1$ and $B \vee (l_2 \wedge ... \wedge l_s)$ it can be easily resolved that $A \vee B \vee (l_1 \wedge ... \wedge l_s)$ holds.

Weakening is directly a weakening rule for Resolution which can be eliminated easily. □

Theorem 4. *The Reflection Principle for Resolution $SAT^n_r(x,z) \wedge REF^n_{r,m}(x,y)$ has Res(2) refutations of size $(nr + nm)^{O(1)}$.*

We omit the proof in this abstract since it is purely technical and refer to [AB02].

Lemma 3.7. *If Res(2) has feasible interpolation, then Resolution is weakly automatizable.*

Proof. Theorem 4 says that the Reflection Principle for Resolution has Res(2) refutation of polynomial size. By assumption we know that Res(2) has feasible interpolation. With Theorem 3 of Pudlak we deduce that the canonical pair for Resolution can be separated in polynomial time. This implies directly that Resolution is weakly automatizable via Res(2). □

Another lemma of Pudlak says [Pud09]:

Lemma 3.8 (Pudlak)**.** *The canonical pair of a proof system S is separable in polynomial time iff S is weakly automatizable.*

Theorem 5. *If Resolution is weakly automatizable, then Res*(2) *has feasible interpolation.*

Proof. Because Resolution is weakly automatizable, with Lemma 3.8 we obtain that the canonical pair of Resolution is polynomially separable. The canonical pair of Res(2) is polynomially separable: For an input of Res(2) by Lemma 3.6 we obtain a refutation for Resolution whose size is polynomial in the size of the Res(2) refutation. By the polynomial separability of Resolution we obtain the result for Res(2). For an input of interpolation $A_0(x, y) \wedge A_1(x, z)$ in Res(2) we check by the polynoimal separability of the canonical pair of Res(2) if $A_0(x, y)$ is satisfiable. Thus we can check in polynomial time if A_0 or A_1 is contradictory. □

The results can be generalized as follows: Resolution is weakly automatizable iff Res(k) has feasible interpolation [AB04].

References

[AB02] Albert Atserias and Maria L. Bonet. On the automatizability of resolution and related propositional proof systems. In *CSL '02: Proceedings of the 16th International Workshop and 11th Annual Conference of the EACSL on Computer Science Logic*, pages 569–583, London, UK, September 2002. Springer.

[AB04] Albert Atserias and Maria L. Bonet. On the automatizability of resolution and related propositional proof systems. *Information and Computation*, 189(2):182–201, 2004.

[AR01] Michael Alekhnovich and Alexander A. Razborov. Resolution is not automatizable unless W[P] is tractable. In *Proceedings of 42nd IEEE Symposium on Foundations of Computer Science*, pages 210–219, Washington DC, USA, October 2001. IEEE Computer Society.

[GL09] Nicola Galesi and Massimo Lauria. On the automatizability of polynomial calculus. *Theory of Computing Systems*, pages ?–?, 2009.

[Pud09] Pavel Pudlák. On reducibility and symmetry of disjoint NP pairs. *Theory of Computing Systems*, 295(1–3):323–339, 2009.