Parameterizing the Notion of Automatability in Proof Complexity

MSc Thesis *(Afstudeerscriptie)*

written by

Hannah Van Santvliet

under the supervision of **Dr Ronald de Haan**, and submitted to the Examinations Board in partial fulfillment of the requirements for the degree of

MSc of Logic

at the *Universiteit van Amsterdam.*

Date of the public defense:	**Members of the Thesis Committee:**
August 29, 2024	Prof Balder ten Cate
	Dr Ronald de Haan (Supervisor)
	Dr Ekaterina Shutova (Chair)
	Dr Florian Speelman

Institute for Logic, Language and Computation

Abstract

In the thesis, we extend the proof-theoretic notion of automatability to a parameterized version called fpt-automatability. Automatability gives an indication whether for a proof system automated theorem search is feasible. We develop parameterized automatability results that enrich the theory of automatability with a collection of graph parameters for the proof systems Extended Frege and resolution. We use methods such as interpolation and polynomial simulation for bounded arithmetic formulas, and we exploit properties for intersection graphs. Also, with fpt-dominance, we show relationships between the parameters.

Contents

1 Introduction

Proof complexity is a subfield of computational complexity. If computational complexity is seen as the study of computational resources needed to solve computational problems, then, proof complexity concerns itself with the computational resources needed to prove or refute statements. This includes questions such as the existence of proofs for a given tautology, the resources to automate proof search or bounds for the lengths of the proofs. A motivation for research on lower bounds of the lengths of proofs is the conjecture that $\mathsf{NP} \neq \mathsf{coNP}$ is equivalent to the statement that no propositional proof system has polynomial-size proofs for every tautology [CR23].

A motivation for research on automatability is the exploration of proof search and of what is possible and feasible even before software automate proving statements in large scales. If a proof system is automatable, we can find for every tautology of the proof system a proof in time polynomial in the shortest proof for the tautology [BPR00]. Thus, the theory of automatability indicates the feasibility with regard to time needed for proof search.

This motivation leads to the aproach of applying parameterized complexity to proof complexity. Parameterized complexity grants a setting to enrich a theory by introducing parameters that are significant to the problem. The thesis presents parameters that are significant for automatability and results bounding the proof length of a tautology. For this, we introduce the notion of fpt-automatability and fpt-boundedness whereby the prefix 'fpt' stands for fixed-parameter tractable and expresses feasibility for small values of the respective parameters.

The parameters that we consider are graph parameters. This is due to the fact that the representation of the proof as a proof graph or the representation of the tautology as a graph are natural parameters and therefore, good data structures for algorithms that automate proof systems.

We prove results for two proof systems: An extension of Frege proof systems, called Extended Frege and resolution. Frege proof systems are propositional proof systems that imitate Hilbert style calculus. We focused on Extended Frege since it is relevant to proof complexity and widely discussed in the literature (see for example [AB04], [Bus12], [BPR00] or [Kra95]).

Resolution is a proof system that is connected to the Boolean satisfiability problem in computational complexity. Many of the concrete algorithms that have been developed in the past decades to solve the Boolean satisfiability problem are based on a search procedure that can be seen as a form of resolution (see eg Theorem 4.2.1 in [Kra95], [AFT11] or [BN21] for a general survey).

We present the collection of fpt-automatability results separately, first, for Extended Frege, then, for resolution. Within the same proof system, we set the parameters in context via a term from parameterized complexity called fpt-dominance. The dominance of one parameter to another structures the thesis and forms a lattice on the order of the parameters.

In the following, we sketch our results. The overview map should guide through the thesis and structure its content. The results on non-fpt-automatability are conditional, thus, rely on hardness assumptions and whenever we mention non-fpt-automatability, we implicitly mean that they are conditional.

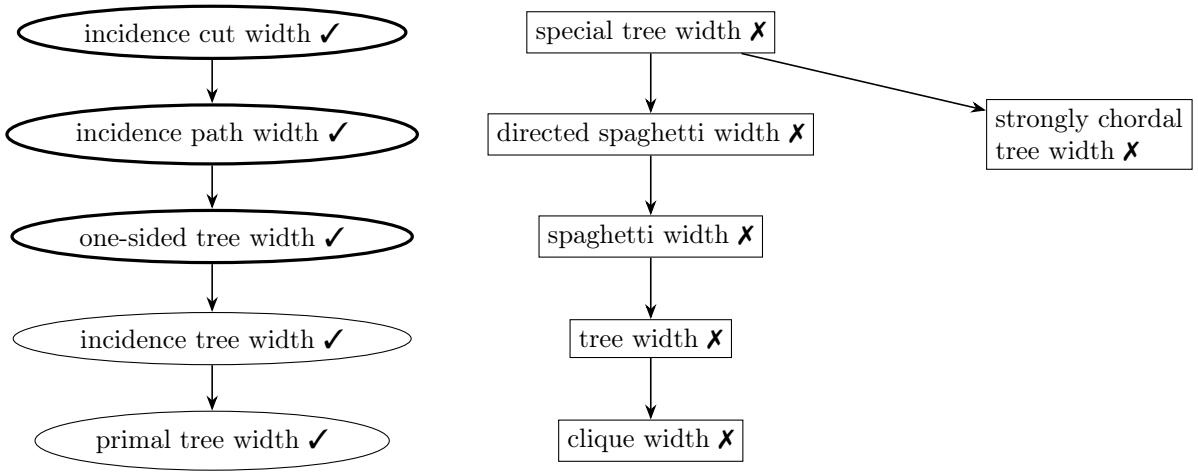


Figure 1: Overview over the parameters

Each node stands for an fpt-automatability (denoted by ✓) or (conditional) non-fpt-automatability result (denoted by ✗). We connect a parameter k with parameter k' by an edge from parameter k' to parameter k if k' dominates k. Furthermore, we classify the type of result by the style of the nodes. A thin, rectangular node stands for non-fpt-automatability for Extended Frege, a thin, circle-shaped node stands for fpt-automatability for resolution and a thick, circle-shaped node stands for fpt-automatability and fpt-boundedness for resolution.

1.1 Contributions

In the thesis, we introduced the new terms fpt-automatability and fpt-boundedness. Built on these terms we developed a collection of fpt-automatability and fpt-boundedness results illustrating the applicability of the terms.

We use different methods to prove our results. In particular, we use interpolation to show a building stone result on the parameter tree width since interpolation is closely related to automatability [BPR00]. Feasible interpolation refers to being able to tell apart two sets with help of a polynomial size circuit. An interpolation that is not feasible can tell apart the two sets but might need bigger circuits to do so.

In sequence, we generalize the result on tree width and motivate parameters such as spaghetti tree width via graph theory. Here, we stress the relationships between the parameters, visible in the lattice structure of our results.

In addition, the section on resolution that comes with many automating results describes algorithms how to constructs refutations for a given unsatisfiable formula. For this, we make use of the concept of dynamic programming and try to illustrate fpt-automation algorithmically.

The collection of results should show the use of the parameterized setting for automatability: Due to parameters that come along with an intuition/application, we can improve our estimation of what are feasible automatability problems in proof complexity.

2 Preliminaries

We split the preliminaries into the proof complexity part, parameterized complexity part and a part that merges the two previous ones to parameterized proof complexity. Parameterized proof complexity is a relatively new area and therefore, in sequence, we add only a short review of the literature of concepts that relate to proof complexity and parameterized complexity.

2.1 Proof complexity

The very base of proof complexity is the definition of a proof system. Proof systems are logical systems in which proofs can be deduced. For formalizing proving a statement, let us have a look at the definition of a propositional proof system given by Krajicek [Kra95].

Definition 2.1. *A* propositional proof system *is a surjective, polynomial time function P whose range is the set of tautologies and that maps proofs to tautologies.*

Surjectivity ensures that every tautology has a proof, thus, that the respective proof system is *sound*. Note that here we allow all kinds of functions to be proof systems. Nonetheless, there are a few established proof systems with proofs based on defined rules and axioms. These rules and axioms vary over the different proof systems. Some rely on a single rule but a variety of axioms such as the resolution proof system, some focus on a small number of axioms but have more rules such as Gentzen style sequent calculus. Another proof system is the *Frege* proof system.

Definition 2.2. *A* Frege rule *is a k+1-tuple of formulas $A_0, \ldots, A_k$ in atoms $p_1, \ldots, p_n$ written as*

$$\frac{A_1, \ldots, A_k}{A_0}$$

such that any assignment $\alpha : \{p_1, \ldots, p_n\} \to \{0, 1\}$ satisfying all formulas $A_1, \ldots, A_k$ also satisfies formula A_0 (soundness of the rule).

Intuitively, we can see a Frege proof as a machine that takes an initial set of axioms, and sequentially applies Frege rules to obtain new formulas until it reaches the formula that is the statement that was to prove. Formally, we define the *Frege proof system* similarly to Krajicek in [Kra95]:

Definition 2.3. *Let F be a finite collection of Frege rules.*

1. *A* Frege proof *of a formula η_0 from formulas $\{\eta_1, \ldots, \eta_u\}$ is a finite sequence $\theta_1, \ldots, \theta_k$ of formulas such that every θ_i is either an element of $\{\eta_1, \ldots, \eta_u\}$ of inferred from some earlier θ_j with $j < i$ by a rule from F and such that the last formula is η_0.*
2. *F is implicationally complete if and only if for any set $\{\eta_1, \ldots, \eta_u\}$ such that every assignment satisfying all formulas in this set also satisfies η_0, the formula η_0 is F-provable from $\{\eta_1, \ldots, \eta_u\}$.*
3. *F is a* Frege proof system *if and only if it is implicationally complete.*

There is not just one single Frege proof system but several ones because there is a plenitude of Frege rules. Since proof complexity aims to quantify computational resources needed for proving a statement in a certain proof system, it makes sense to compare the computational resources needed to prove a fixed statement in different proof systems. For this, we cite the notion of a *polynomial simulation* by Krajicek in [Kra95]. Thus, if we have a proof given in a proof system, simulations formalize the translation process of the proof to another proof system. With that, we can build a hierarchy of proof systems depending on whether two proof systems polynomially simulate each other.

Definition 2.4. *A* polynomial simulation *from Q to P (in short, p-simulation or $Q \leq_p P$) is a polynomial time function $f(w, \tau)$ such that for all w and τ*

$$Q(w) = \tau \to P(f(w, \tau)) = \tau.$$

We say that the stronger, more expressive proof system p-simulates the weaker system. So, for any proof in the weaker proof system, we can find a corresponding proof in the stronger proof system. Frege systems are on the same level of this hierarchy since every two Frege systems p-simulate each other (see Theorem 4.4.13 in [Kra95]). For extensions of Frege system such as Extended Frege this is not necessarily the case: Extended Frege polynomial simulates Frege proof systems but it is an open problem whether Frege systems can simulate Extended Frege [BBP95].

Definition 2.5. *Let F be a Frege system. An* extended Frege proof *is a sequence of formulas $\theta_1, \ldots, \theta_k$ such that every θ_i either is obtained from some previous θ_j by a Frege rule or has the form*

$$q \equiv \psi$$

where:

1. *the atom q appears in neither ψ nor θ_j for some $j < i$*
2. *the atom q does not appear in θ_k*
3. *$\alpha \equiv \beta$ abbreviates $(\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta)$.*

A formula of this form is called an extension axiom *and the atom q is called an* extension atom.

An extended Frege system *EF is the proof system whose proofs are the extended Frege proofs.*

Proofs in propositional proof systems such as Frege can be associated with a graph. The graph induced by a Frege proof has vertices for all axioms needed during the proof and formulas inferred by rules and connects two vertices if and only if one is inferred by the other by a rule. We call the graph associated with a Frege proof a *proof graph*. Depending on the proof system, we are going to build on the notion of a proof graph.

As a consequence, graph theory and graph measures such as tree width, path width or cut width can be applied to proofs. Aligned to that, we define the property of a proof being *treelike*.

Definition 2.6. *A Frege proof $\theta_1, \ldots, \theta_k$ is called* treelike *if and only if every step θ_i is a hypothesis of at most one inference in the proof.*

This is equivalent of saying that the proof graph of a proof should be a tree. For all other proof systems, we are going to be introduce later on, we also call a proof treelike if the graph associated with the proof is a tree. We stress again that for other proof systems the associated graph to proof is obtained differently than for a Frege proof.

To be able to differentiate between proofs of different proof systems, we stick with the notion of a P-proof, denoting that the proof is obtained via the proof system P. Furthermore, consider that the size of a formula φ we denote by $|\varphi|$ and is defined by the number of symbols in φ. The size of a proof π we denote by $|\pi|$ and is defined by the number of symbols in π.

This notion of size is needed for the definition of an *automatable* proof system. As the word indicates, we want to *automate* finding a proof and give an indication if the time needed to do so is feasible. Similar to problems in computational complexity, we cannot gain results to indicate the exact number of computation steps for any proof system as we cannot give the exact run time for an arbitrary computational problem. Instead, we have to relate the hardness of the problem to a sensible measure. For automatability as introduced by Maria Luisa Bonet, Toniann Pitassi and Ran Raz in [BPR00], we consider the measure of the shortest proof. Clearly, we cannot find a proof faster than the shortest proof there is.

Definition 2.7. *A proof system P is* automatable *if for any tautology φ we can find a P-proof π in polynomial time in the size of shortest proof of φ and we say that a proof system P is* non-automatable *otherwise.*

A practical motivation for results showing that a proof system is non-automatable is the indication of the difficulty of proof search for this proof system. If a proof system is not automatable it might be difficult to design a good algorithm for proof search in this proof system.

Aligned with the latter motivation, we want to show non-automatability results that indicate the complexity of finding a proof. One framework building upon computational complexity is parameterized complexity.

2.2 Parameterized complexity

Parameterized complexity introduces parameters for the respective computational problem that are significant to it. Subsequently, it measures the complexity of the problem in addition to the input size of the problem in the respective parameter. We call a problem *fpt-tractable* if it has effective running time with regard to the parameter k and the input size n. For small parameter values and effectively computable functions f, this should yield fast algorithms in practice.

Definition 2.8. *A problem is* fpt-tractable *with respect to a parameter k if there is a computable function f and the problem can be solved in time $f(k) \cdot p(n)$.*

Take for example the NP-complete problem VERTEX COVER [LGJ83]. Even though the problem is NP-complete, it is also fpt-tractable with regard to the parameter size of the potential vertex cover k. For small parameter values, this yields good algorithm $\mathcal{O}(2^k n)$, see chapter 4, page 31 in [Nie06]. This example should illustrate that knowing about the fpt-tractability might give some additional practical insights.

Similar to the hierarchy of complexity classes in the conventional setting, there is a hierarchy of parameterized complexity classes where it is so far unknown if the inclusion is strict.

Two prominent parameterized complexity classes are FPT and W[P]. While FPT is the class of decision problems that are fpt-tractable, the definition introduced in [SFG07] of the class W[P] is the following:

Definition 2.9. W[P] *is the class of problems that can be decided by a nondeterministic $h(k)|x|^{\mathcal{O}(1)}$-time Turing machine that makes at most $\mathcal{O}(f(k)\log(n))$ nondeterministic choices in the computation on (x, k).*

The work focuses on the notion of fpt-tractability and attempts to translate this to proof complexity.

2.3 Parameterized proof complexity

In the following, we want to apply the parameterized setting to proof theoretic definitions. In the literature, different paramerized setting were introduced for proof theory. The work of Stefan Dantchev, Barnaby Martin and Stefan Szeider introduces a framework for parameterized resolution with the parameter Hamming weight of the clauses in the resolution [DMS07]. Alternatively, the work of Jörg Flum and Moritz Müller introduces notions of fpt-simulations between proof systems employing parameters such as the number of applications of certain rules within the proof [FM12]. We extend the definition of *p-bounded proof system* of Jörg Flum and Moritz Müller to an *fpt-bounded proof system*:

Definition 2.10. *A proof system P is* fpt-bounded *with respect to parameter k if there is a computable function f such that for every tautology φ a P-proof π has size at most $f(k) \cdot p(n)$ for $|\varphi| = n$.*

Similarly to the definition of fpt-bounded, we take the definition of an automatable proof system and modify it. While the term fpt-bounded talks about the size of a proof for a given tautology with respect to the size of the tautology, the term *fpt-automatable* refers to the time to find a proof of a tautology with respect to the proof length. In the conventional setting, we take the shortest proof as a reference to the time to find a proof. In the parameterized setting, we have a different reference, namely the *minimal candidate value*, to estimate the time.

This notion is needed to make the technicalities of the definition work since the minimal candidate value is dependant on our parameterization. When we parameterize a problem, we look for a significant parameter and rephrase the problem with respect to the parameter. The parameter can be a fixed value k or the value of a parameter function κ. We see that parameter functions like tree width allow several interpretations of the value of κ. The parameter k could be for example $k = \min_{G_\pi} treewidth(G_\pi)$ over all graphs G corresponding to a proof π. On the other hand, we could take the tree width of the shortest proof $k = treewidth(G_{\hat{\pi}})$ for $\hat{\pi} = \min_\pi |\pi|$. Since these variants lead to valid parameters, we fix in our definition the way how the value k is obtained by κ. This definition naturally implies fpt-bounded and fpt-automatability with regard to a fixed value if we consider κ as a constant function with value k.

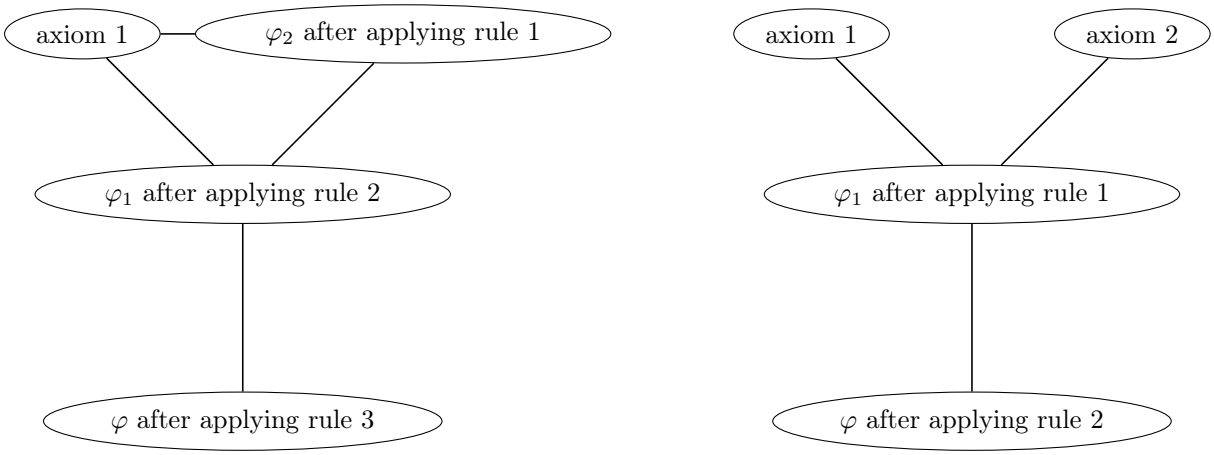
Definition 2.11. *Let the* minimal candidate value $v_{\varphi,f,\kappa,P,p}$ *with regard to tautology φ, function f, parameter function κ with $k = \kappa(\varphi, \pi)$, proof system P and polynomial p be the value over the minimum over all potential proofs π of the statement φ in the proof system P of the term $f(k)\cdot p(n)$ for the parameter function κ with $|\pi| = n$. In short, $v_{\varphi,f,\kappa,P,p} := \min_\pi f(k)p(n)$.*

With the definition of the minimal candidate value, we can define an fpt-automatable proof system as follows:

Definition 2.12. *A proof system P is* fpt-automatable *with regard to parameter $k = \kappa(\varphi, \pi)$ if we can find a computable function f such that for any tautology φ, we have a P-proof π in time polynomial q in the minimal candidate value $v_{\varphi,f,\kappa,P,p}$ for a polynomial p and we say that a proof system P is* non-fpt-automatable *with regard to a parameter k otherwise.*

The function κ takes as input a propositional formula φ and a P-proof π that computes a value $\kappa(\varphi, \pi)$. This value is our parameter. In the following, we give an example on how to determine the minimal candidate value for Frege proofs.

Example 2.13. *Let $\varphi = f$ be the function $f(x) = 2^x$ and P a proof system. Take as a parameter function κ the minimal size of the biggest cycle over all potential proofs in the proof system. Then, there are several potential Frege proofs π of the statement φ. Consider two imaginary ones that yield the following proof graphs.*



(a) Graph G_1 of proof π_1 (b) Graph G_2 of proof π_2

Figure 2: Example of determining parameter k

We see that the biggest cycle of G_1 has size 3, whereas the biggest cycle of G_2 is 2. We conclude that $k = \min\{2^2, 2^3\} = 4$.

Since the definition relies on the parameter k, there are fpt-automatable proof systems with regard to the one parameter k that are non-fpt-automatable with regard to an other parameter k'. For example, Extended Frege is not fpt-automatable with regard to tree width (see Theorem 3.27) but is fpt-automatable with regard to path width (see Corollary 4.4). Just as well, it is possible to gain automatability results for proof systems that do not hold for other proof systems. Nonetheless, we see that there is a connection between the results for different proof systems if one polynomially simulates the other.

Lemma 2.14. *If a proof system P_1 polynomially simulates proof system P_2 ($P_2 \leq_q P_1$), P_2 is fpt-automatable with regard to a parameter k and $v_{\varphi,f,\kappa,P_1,p} \geq v_{\varphi,f,\kappa,P_2,p}$, we have that the proof system P_1 is fpt-automatable with regard to parameter k.*

Proof. Since P_1 polynomially simulates proof system P_2, we can translate any proof in P_2 in polynomial time q to a proof in P_1. Since P_2 was fpt-automatable, we can find a proof in time $g(v_{\varphi,f,\kappa,P_2,p})$ for a polynomial g for any tautology φ. Subsequently, we find a proof in time $q(g(v_{\varphi,f,\kappa,P_2,p}))$ for any tautology φ and since $q \circ g$ is a polynomial and $v_{\varphi,f,\kappa,P_1,p} \geq v_{\varphi,f,\kappa,P_2,p}$, the assertion follows. □

Here, we also note that there is a relation between fpt-automatability and automatability with regard to the same proof system.

Lemma 2.15. *If a proof system P is is automatable, it is fpt-automatable with regard to any parameter k.*

Proof. A proof system P is fpt-automatable with regard to parameter k if for any tautology φ, we can find a computable function f and a P-proof π in time of polynomial in the minimal candidate value $v_{\varphi,f,\kappa,P,p} = \min_\pi f(k)p(n)$ with $|\pi| = n$. Since

$$p(\min_\pi |\pi|) \leq \min_\pi f(k)q(|\pi|)$$

for a polynomials p and q, we have that a proof system P is fpt-automatable with regard to any parameter k, if it is automatable. □

Intuitively, this makes sense since with the introduction of a parameter, we aim to obtain more information and enrich the theory of automatability. If there were proof systems that are only fpt-automatable with regard to a certain parameter but not with regard to the shortest proof, we would have a contradiction between the parameterized setting and conventional automatability results.

2.4 Relation to other work

We introduced proof systems as mappings from proofs to tautologies as used in literature in [Bus12], [DMS07], [PS10] and [FM12]. But there is another canonical definition of a proof system P that is used by Krajicek in [Kra95]. Here, we switch the domain and the range of the function in the definition of a proof system. To differentiate between the two definitions of proof systems, we denote the alternative version of proof system that maps tautologies to proofs with P'.

Definition 2.16. *A propositional proof system is a polynomial time function P' whose domain is the set of tautologies and that maps no two different tautologies to the same proof.*

Thus, on the contrary to Definition 2.1, we map tautologies to proofs. The condition that the proof system maps no two different tautologies to the same proof implies injectivity of P'. We assume P' to be total, thus, for $f : X \to Y$ for every $x \in X$, we have that $f(x) \in Y$. While totality is implicit for proof systems P', for proof systems P, we demand surjectivity in Definition 2.1. As a consequence, we can consider a surjective proof system that maps proofs to tautologies as the right inverse[1] of a proof system that maps tautologies to proofs. Similarly, we consider P' as a left inverse to P.

The relationship between proof systems P and P' makes it possible to transfer results that rely on different proof system definitions to each other. For this, we define when a proof system P' is fpt-bounded analogously to when a proof system P is fpt-bounded.

Definition 2.17. *A proof system P is* fpt-bounded *with respect to parameter k if for every tautology φ there is a computable function f and a P-proof π with size at most $f(k) \cdot p(n)$ for $|\varphi| = n$.*

And with this definition, we can easily prove the following:

Proposition 2.18. *A proof system P' that maps tautologies to proofs is fpt-bounded if and only if the respective surjective proof system P that maps proofs to tautologies is fpt-bounded.*

Proof. Assume that a proof system P' that maps tautologies to proofs is fpt-bounded. Then, the proof system P' maps each tautology φ to a proof of size at most $f(k)p(n)$ for a computable function f and a parameter k. Thus, the image of P' are proofs that of size at most $f(k)p(n)$ for a computable function f and a parameter k. But the image of P' is the domain of the left inverse P: A surjective left inverse exists vacuously if the domain and the range are finite. Since P is surjective, each tautology φ in the range of P has a proof π in the domain with $P(\pi) = \varphi$. But the proofs in the domain were all bounded in size. It follows that every tautology has a proof of size at most $f(k)p(n)$ and therefore, P is fpt-bounded.

Assume that a proof system P' that maps proofs to tautologies is fpt-bounded. Then, any tautology φ has a P'-proof of size $f(k)n^{\mathcal{O}(1)}$ for a computable function f and parameter k. But a proof system P' that maps no two different tautologies to the same proof is a right inverse to P: The injective right inverse exists vacuously for a surjective P' if domain and range are finite. But if every tautology has a proof of size at most $f(k)p(n)$ as preimage, then, P' maps every tautology to a proof of size at most $f(k)p(n)$. Thus, P' is fpt-bounded. □

The equivalence of fpt-boundedness leads us to the natural question whether a proof system P is if and only if automatable if P' is automatable. But here, we fail to give a good definition for an automatable proof system P' due to the following reason: On the contrary to fpt-boundedness, automatability relies on the fact that there are several proofs for a single tautology. There needs to be at least one proof due to the completeness of the respective proof system. The definition of automatability relies on measuring the time to find a proof with respect to a reference proof (namely, the shortest one). If there is only one proof since the image of an element cannot contain more than one element for a conventional function, the definition of automatability becomes nonesensical.

This also serves as a motivation to stick with our definition of a proof system P.

In the following, we start a survery of fpt-automatability with regard to different parameters and proof systems P.

[1] As common in analysis, we define a right inverse of a function $f : X \to Y$ as the function $r : Y \to X$ such that $f \circ r = id_Y$. Similarly, we define the left inverse of a function $f : X \to Y$ as the function $l : Y \to X$ such that $l \circ f = id_X$.

3 Parameters for Frege proof systems

In sequence, we want to classify fpt-automatability results with respect to different parameters. We separate the parameters with regard to the proof system and whether they yield fpt-automatability. For each parameter, we introduce the respective tools to show fpt-automatability. First, we dive into the theory of bounded arithmetic.

3.1 Bounded arithmetic and Gentzen proofs

Bounded arithmetic requires the range of quantifiers to be bounded. With that, bounded arithmetic is a weaker theory than first-order arithmetic since there, we allow ranges over unbounded and infinite domains. Since we require quantifier ranges to be bounded, those formulas can be rewritten as propositional formulas. But first, let us formally introduce the concept of a bounded formula as in [Kra95].

Definition 3.1. *We define the function $|\cdot|$ that takes as input a term and outputs the length of the binary representation of the term. The class $\Sigma_0^b = \Pi_0^b$ of* sharply bounded *formulas consists of formulas in which all quantifiers have have form*

$$\exists x \leq |t| \text{ or } \forall x \leq |t|$$

for a term t that may not depend on the variable it is bounding. We call a formula bounded if all quantifiers have the form

$$\exists x \leq t \text{ or } \forall x \leq t.$$

To illustrate the translation of a bounded formula to a propositional formula consider the following example.

Example 3.2. *Let $\varphi = \forall x \leq 10 \exists x \leq y. x = 10 + y$. Then, φ is a bounded formula. We can rewrite φ to*

$$\bigwedge_{x=0}^{10} \bigvee_{y=0}^{x} x = 10 + y$$

and this formula is propositional since both x and y take eleven different values.

The following formula φ' is a sharply bounded formula since the binary representation of the number 10 is 1010_2 and has length four and the length of the binary representation of y is $\lceil \log_2(y+1) \rceil$. With that, we can write φ as

$$\varphi' := \forall x \leq |10| \exists x \leq |y|. x = 10 + y$$

or also

$$\varphi' := \forall x \leq 4 \exists x \leq \lceil \log_2(y+1) \rceil . x = 10 + y.$$

In particular, we see that a sharply bounded formula is also a bounded formula.

An extension of sharply bounded formulas are the classes of formulas Σ_1^b and Π_1^b.

Definition 3.3. *The classes Σ_1^b and Π_1^b are the smallest classes satisfying*

- $\Sigma_0^b \subseteq \Sigma_1^b \cap \Pi_1^b$
- *Σ_1^b and Π_1^b are close under sharply bounded quantification, disjunction and conjunction*
- *Σ_1^b is closed under bounded existential quantification*
- *Π_1^b is closed under bounded universal quantification*
- *the negation of a Π_1^b-formula is a Σ_1^b-formula and the negation of a Σ_1^b-formula is a Π_1^b-formula*

We see that the example before is a Σ_1^b-formula as well as a Π_1^b-formula due to the first condition. We modify our example to obtain a formula that is part of Σ_1^b but that is not a Σ_0^b-formula:

Example 3.4. *Let* $\varphi = \exists w \leq 8 \forall z < 9 \forall x \leq 10 \exists x \leq y. x = 10 + y + w + z \vee x = w - z$. *Then,* φ *is a* Σ_1^b*-formula. Nonetheless,* φ *is not part of the class* Π_0^b *since* Π_0^b *is not closed under bounded existential quantification.*

We can still rewrite φ *to*

$$\bigwedge_{x=0}^{8} \bigvee_{y=0}^{8} \bigwedge_{y=0}^{10} \bigvee_{w=0}^{x} x = 10 + y \vee x = w - z$$

and this formula is propositional since x, y, z *and* w *take finitely many different values.*

Building on the definition of bounded formulae, we want to introduce another two proof systems: S_2^1 and G_1. For proofs that can be derived in the proof system S_2^1, we have a proof translation to the G_1 proof system that is bounded in size.

The proof system S_2^1 builds upon the $BASIC$ proofs system that entails 32 axioms and the symbols $\{0, 1, +, -, *, <, =, \lfloor x \rfloor, |x|, x\#y\}$ for variables x and y. For the full list of the axioms, see Definition 5.1 in the appendix.

Definition 3.5. *The proof system* S_2^1 *is a language extending BASIC by the polynomial induction axiom PIND*

$$\phi(0) \wedge \forall x.(\phi(\lfloor \frac{x}{2} \rfloor) \to \phi(x)) \to \forall x \phi(x)$$

for all Σ_1^b*-formulas* $\phi(a)$*. The formula* $\phi(a)$ *may have other free variables than* a*.*

We cited this definition from Krajicek (see Definition 5.2.3 [Kra95]) as well as the definition of a Gentzen proof (see Definition 4.3.2 and 4.6.2 [Kra95]). The Gentzen proof system is also called sequent calculus since its main objects are *sequents*.

Definition 3.6. *A* sequent *is an ordered pair of two finite sequences (possibly empty) of formulas written as*

$$\phi_1, \ldots \phi_n \longrightarrow \psi_1, \ldots, \psi_n.$$

A proof in the Gentzen proof system is a sequence of sequents that are obtained using several rules. In comparison to Frege systems, the focus lies more on the rules than on the axioms.

Definition 3.7. *A* G-proof *is a sequence of sequents in which every sequent is either a sequent of the form*

$$A \longrightarrow A, 0 \longrightarrow \text{ or } \longrightarrow 1$$

with A *an atom or is derived from previous sequents by one of the following rules:*

1. *weakening rules* $\textbf{left}\ \dfrac{\Gamma \longrightarrow \Delta}{A, \Gamma \longrightarrow \Delta}$ *and* $\textbf{right}\ \dfrac{\Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \Delta, A}$
2. *exchange rules* $\textbf{left}\ \dfrac{\Gamma_1, A, B, \Gamma_2 \longrightarrow \Delta}{\Gamma_1, B, A, \Gamma_2 \longrightarrow \Delta}$ *and* $\textbf{right}\ \dfrac{\Gamma \longrightarrow \Delta_1, A, B, \Delta_2}{\Gamma \longrightarrow \Delta_1, B, A, \Delta_2}$
3. *contraction rules* $\textbf{left}\ \dfrac{\Gamma_1, A, A, \Gamma_2 \longrightarrow \Delta}{\Gamma_1, A, \Gamma_2 \longrightarrow \Delta}$ *and* $\textbf{right}\ \dfrac{\Gamma \longrightarrow \Delta_1, A, A, \Delta_2}{\Gamma \longrightarrow \Delta_1, A, \Delta_2}$
4. $\neg$ *introduction rules* $\textbf{left}\ \dfrac{\Gamma \longrightarrow \Delta, A}{\neg A, \Gamma \longrightarrow \Delta}$ *and* $\textbf{right}\ \dfrac{A, \Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \Delta, \neg A}$
5. $\wedge$ *introduction rules* $\textbf{left}\ \dfrac{A, \Gamma \longrightarrow \Delta}{A \wedge B, \Gamma \longrightarrow \Delta}$
   and $\textbf{left}\ \dfrac{A, \Gamma \longrightarrow \Delta}{B \wedge A, \Gamma \longrightarrow \Delta}$ *and* $\textbf{right}\ \dfrac{\Gamma \longrightarrow \Delta, A \ \Gamma \longrightarrow \Delta, B}{\Gamma \longrightarrow \Delta, A \wedge B}$
6. $\vee$ *introduction rules* $\textbf{left}\ \dfrac{A, \Gamma \longrightarrow \Delta \ B, \Gamma \longrightarrow \Delta}{A \vee B, \Gamma \longrightarrow \Delta}$
   and $\textbf{right}\ \dfrac{\Gamma \longrightarrow \Delta, A}{\Gamma \longrightarrow \Delta, A \vee B}$ *and* $\textbf{right}\ \dfrac{\Gamma \longrightarrow \Delta, A}{\Gamma \longrightarrow \Delta, B \vee A}$
7. *cut-rule* $\dfrac{\Gamma \longrightarrow \Delta, A \ A, \Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \Delta}$
8. $\forall$ *introduction rules* $\textbf{left}\ \dfrac{A(B), \Gamma \longrightarrow \Delta}{\forall x A(x), \Gamma \longrightarrow \Delta}$ *and* $\textbf{right}\ \dfrac{\Gamma \longrightarrow \Delta, A(p)}{\forall x A(x), \Gamma \longrightarrow \Delta, \forall x A(x)}$

9. $\exists$ *introduction rules* $\textbf{left}\ \dfrac{A(p), \Gamma \longrightarrow \Delta}{\exists x A(x), \Gamma \longrightarrow \Delta}$ *and* $\textbf{right}\ \dfrac{\Gamma \longrightarrow \Delta, A(B)}{\forall x A(x), \Gamma \longrightarrow \Delta, \exists x A(x)}$

We call a proof G_1-proof if it contains only Σ_1^b-formulas. We call a proof G_1^*-proof if it allows only Σ_1^b-formulas and is treelike.

With the definition of the proof system S_2^1 and G_1^*, we have the vocabulary to cite a theorem needed to prove the non-fpt-automatability result with regard to parameter proof tree width.

Theorem 3.8 (Propositional translation)**.** *Let $A(a)$ be a Σ_1^b-formula. Assume that*

$$S_2^1 \vdash A(a).$$

Then there is a bounding polynomial $q(x)$ for A such that for all m the formulas $\|A\|_{q(m)}^m$ have size $m^{\mathcal{O}(1)}$ G_1^*-proofs. (See Theorem 9.2.6 in Chapter 9 in the book in [Kra95].)

Note that this theorem holds for all choices of m and that $\|A\|_{q(m)}^m$ refers to a specific translation. See Section 5 in the appendix for more details on how to construct $\|A\|_{q(m)}^m$. Intuitively, the theorem tells us that there is a good translation to the Gentzen proof system such that the translated proofs are bounded in size and treelike.

The second tool needed to prove the fpt-automatability result with regard to parameter proof tree width is a polynomial simulation between G_1^* and Extended Frege. The proof is rather technical and the first direction splits into a polynomial simulation from G_1^* to SF and SF to G_1^* which together gives the polynomial simulation from G_1^* to Extended Frege. The other direction is proven on induction. Let us define the intermediate proof system as described by Krajicek in Definition 4.5.1 in [Kra95].

Definition 3.9. *The* substitution rule *allows simultaneous substitution of formulas for atoms in one inference step*

$$\frac{\theta(p_1, \ldots, p_n)}{\theta(\phi_1, \ldots, \phi_n)}$$

A Frege system with the additional substitution rule will be denoted by SF.

Thus, the proof system SF is similar to Extended Frege just an extension of a common Frege proof system.

Lemma 3.10. *The proof system G_1^* and Extended Frege polynomially simulate each other.*

Proof. We take the polynomial simulations described in Lemma 4.6.3, Lemma 4.4.8, Lemma 4.5.4, 4.4.10 and 4.4.9 in [Kra95].

First, we want to show that G_1^* polynomially simulates SF.

Since G_1^* simulates any Frege system [CR23] and SF is a Frege system with the additional substitution rule, it suffices to show that G_1^* simulates an instance of the substitution rule:

$$\frac{\theta(p_1, \ldots, p_n)}{\theta(\phi_1, \ldots, \phi_n)}$$

For this, we want to construct a treelike G_1-proof by assuming $\longrightarrow \theta(p_1, \ldots, p_n)$ and concluding $\longrightarrow \theta(\phi_1, \ldots, \phi_n)$.

To the sequent $\longrightarrow \theta(p_1, \ldots, p_n)$, we apply n-times $\forall$ **right** to obtain

$$\longrightarrow \forall x_1 \ldots \forall x_n \theta(x_1, \ldots, x_n).$$

By assuming the axiom of the atomic sequent $A \longrightarrow A$, we can deduce by a constant length G_1^*-proof that

$$\theta(\phi_1, \ldots, \phi_n) \longrightarrow \theta(\phi_1, \ldots, \phi_n)$$

by the introduction rules. By n application of $\forall$ **left**, it follows from $\theta(\phi_1, \ldots, \phi_n) \longrightarrow \theta(\phi_1, \ldots, \phi_n)$ that

$$\forall x_1 \ldots \forall x_n \theta(x_1, \ldots, x_n) \longrightarrow \theta(\phi_1, \ldots, \phi_n).$$

With the cut rule, we infer from

$$\longrightarrow \forall x_1 \ldots \forall x_n \theta(x_1, \ldots, x_n)$$

and

$$\forall x_1 \dots \forall x_n \theta(x_1, \dots, x_n) \longrightarrow \theta(\phi_1, \dots, \phi_n)$$

that

$$\longrightarrow \theta(\phi_1, \dots, \phi_n).$$

Second, we want to show that if G_1^* polynomially simulates treelike SF, G_1^* also polynomially simulates EF.

Let $q_1 \equiv \psi_1, \dots, q_r \equiv \psi_r$ be all the extension axioms introduced in an EF proof $\theta_1, \dots, \theta_k$ of an given statement τ of size m. We can reorganise the EF proof such that $q_1 \equiv \psi_1, \dots, q_r \equiv \psi_r$ are the first steps of the proof. By definition of the extension rule, all q_i do not occur in τ but may appear in subsequent proof steps.

For the following implication

$$q_r \equiv \psi_r \to (q_{r-1} \equiv \psi_{r-1} \to (\dots (q_1 \equiv \psi_1)) \dots) \to \tau$$

of size $\mathcal{O}(m)$, we want to construct a Frege proof of size $\mathcal{O}(m^2)$. This reduces to the problem of finding a Frege proof for one implication $\psi \to q$ of size $\mathcal{O}(m)$ and from that, we successively build the full implication of implications. Assume that q has a proof $\eta_1, \dots, \eta_k$.

We have that

$$\frac{A_{j_1}, \dots, A_{j_r}}{A_k}$$

is an Frege rule and therefore, the following formula is a tautology for a fixed z:

$$((z \to A_{j_1}) \to (\dots \to (z \to A_k) \dots)$$

Take η_i inferred by $\eta_{j_1}, \dots, \eta_{j_l}$ and use the substitution rule, to obtain:

$$((\psi \to \eta_{j_1}) \to (\dots \to (\psi \to \eta_{j_l}) \to (\psi \to \eta_i) \dots)$$

From this formula, together with $\psi \to \eta_{j_s}$ for $s \leq l$, it follows with modus ponens that $\psi \to \eta_i$.

Since $\eta_1, \dots, \eta_k$ was a proof of q, we can successively build a proof for $\psi \to q$. We iterate this over the nested implication and obtain a Frege proof of

$$q_r \equiv \psi_r \to (q_{r-1} \equiv \psi_{r-1} \to (\dots (q_1 \equiv \psi_1)) \dots) \to \tau.$$

We apply the substitution rule and substitute ψ_r for q_r

$$\psi_r \equiv \psi_r \to (q_{r-1} \equiv \psi_{r-1} \to (\dots (q_1 \equiv \psi_1)) \dots) \to \tau.$$

We can derive $\psi_r \equiv \psi_r$ by a proof of size $\mathcal{O}(|\psi_r|)$ since we can substitute the ψ_i for atoms p_i.

By repeated application of modus ponens to the implication and $\psi_i \equiv \psi_i$, we get a proof of τ in size $\mathcal{O}(m^2)$.

In sum, we have that SF polynomially simulates EF and G_1^* polynomially simulates SF and therefore, G_1^* polynomially simulates EF.

For showing that EF polynomially simulates G_1^*, we refer to literature. Initially, the Lemma appears as Lemma 4.6.3 in [Kra95]. Krajicek has written a second book on the matter that was published 26 years later. There, we can find the lemma as Theorem 4.1.3 in [Kra19]. For more detail on how the p-simulation is constructed, we refer to Theorem VII.4.16 in [CN10]. Cook and Nguyen prove the statement on induction on the number of sequents in a proof in G_1^* and sequents of the form

$$\begin{aligned} &\alpha_1(\boldsymbol{p}), \dots, \alpha_i(\boldsymbol{p}), \dots, \exists \boldsymbol{x}_1 \beta_1(\boldsymbol{p}, \boldsymbol{x}_1), \dots, \exists \boldsymbol{x}_j \beta_j(\boldsymbol{p}, \boldsymbol{x}_j), \dots \\ &\to \gamma_1(\boldsymbol{p}), \dots, \gamma_s(\boldsymbol{p}), \dots, \exists \boldsymbol{y}_1 \delta_1(\boldsymbol{p}, \boldsymbol{y}_1), \dots, \exists \boldsymbol{y}_t \delta_t(\boldsymbol{p}, \boldsymbol{y}_t), \dots \end{aligned}$$

with variable vectors $\boldsymbol{p}, \boldsymbol{x}_i$ and $\boldsymbol{y}_t$ and quantifier-free formulas $\alpha_i(\boldsymbol{p})$ and $\gamma_s(\boldsymbol{p})$.

For the induction step, they make a case distinction depending on the Gentzen rule and construct a Extended Frege proof for each of the cases. □

Let us summarize what we have got so far: The notion of certain types of bounded formulas and the result that returns a polynomial size Gentzen proof for formulas in Σ_1^b that are derived in S_1^2. Furthermore, we can build upon the theorem since there is a polynomial simulation from Gentzen proofs to Extended Frege proofs. From that, we want to obtain a treelike Extended Frege proof. Recall that the proof graph of a Frege proof is has formulas as vertices and edges if a formula is inferred by the another formula. Similarly, we define the proof graph of an Extended Frege proof (as it is not yet defined by Krajicek). Here, we can have vertices that contain extension axioms and edges that stand for inferences based on extension atoms.

The definition leads to the following result:

Lemma 3.11. *Treelike Extended Frege p-simulates Extended Frege.*

Proof. For this, we want to show that we can transform an EF proof π of a tautology φ in polynomial time to a treelike EF proof π^* of φ.

Consider π. First, we reorganise the EF proof such that $q_1 \equiv \psi_1, \ldots, q_r \equiv \psi_r$ are the first steps of the proof. Call the (conjunction of) the lines based on extension axioms Ext.

The proof π_0 can be seen as a Frege proof for the entailment $\mathsf{Ext} \models \varphi$ where the lines of Ext are additional assumptions, resulting in the formula φ at the end of the proof.

We can transform π_0 in polynomial time to an treelike Frege proof π_0^* by using the result of Krajicek about treelike Frege, i.e., Thm 2.2.1 in the new book or Lem 4.4.8 in the old book. This is still a proof for the entailment $\mathsf{Ext} \models \varphi$.

Since Ext contains only extension axioms of π, we know that π_0^* is a treelike EF proof for φ. We let $\pi^* = \pi_0^*$ and conclude that treelike Extended Frege p-simulates Extended Frege. □

It is relevant that the property of being treelike is preserved since we want to show non-fpt-automatability with regard to tree width. Before we formally introduce the notion of tree width, we give motivation why the theorem that provides a polynomial size Gentzen proof is crucial to the fpt-automatability result. We aim to show that if the size of a certain proof is polynomial, we can transform the proof into a circuit that interpolates a cryptographic problem.

3.2 Cryptographic problems and feasible interpolation

If a circuit interpolates the problem of differentiating two sets, loosely speaking, the circuit outputs 1 if an element is in one of the sets and 0 if it is in the other set. With that, it is clear that the sets are not the same. Consider the following RSA encryption scheme from cryptography and the problem whether an attacker can break the encryption scheme.

Definition 3.12. *Take two sufficiently large primes p and q and compute the product $n := p \cdot q$. An encoding key, that is public, is a pair (n, e), where $1 < e < n$. An $x < n$ is encoded by $y := x^e \mod n$. The secret decoding key is a number d such that $e \cdot d \equiv 1 \mod \varphi(n)$, where $\varphi(n)$ is the Euler function. To decode x from y compute $x := y^d \mod n$.*

In cryptography, we want to encode messages from a sender to a recipient such that no attacker can easily decipher the message. The RSA (Rivest Shamir Adleman) protocol was introduced 1978 [RSA78] and is still used in practice. The notion of breaking a cryptographic protocol means that the attacker can easily decipher the message. More rigorously, we define:

Definition 3.13. *We call a cryptographic protocol* secure *if the attacker cannot deterministically and fully recover the message from the encrypted message with circuits of polynomial size in the input.*

We aim to interpolate the RSA problem and phrase it as a differentiation problem between the two sets:

$$A_0 := \{(n, e, y) \mid \exists x, d, r < n . x \equiv 0 \mod 2 \wedge x^e \equiv y \mod n \wedge y^e \equiv x \mod n \wedge y^r \equiv 1 \mod n \wedge gcd(e, r) = 1\}$$
$$A_1 := \{(n, e, y) \mid \exists x, d, r < n . x \equiv 1 \mod 2 \wedge x^e \equiv y \mod n \wedge y^e \equiv x \mod n \wedge y^r \equiv 1 \mod n \wedge gcd(e, r) = 1\}$$

The idea is that we can only differentiate the sets A_0 and A_1 if we find a polynomial size circuit that breaks RSA. The fact that RSA is still used in practice together with number theoretic results on the complexity of integer factorization, make it unlikely that it is easy to break RSA. Thus, it should be unlikely that we can easily find an interpolating circuit that differentiates between A_0 and A_1. Formally, we define an interpolant likewise as in [KP98] as a formula that is a tautology if combined with the formulas $\alpha(\boldsymbol{x})$ or $\beta(\boldsymbol{y})$ and stress that $\boldsymbol{x}, \boldsymbol{y}$ and $\boldsymbol{z}$ are vectors by writing them in bold font.

Definition 3.14. *An* interpolant *of a tautological formula $\alpha(\boldsymbol{x}, \boldsymbol{y}) \to \beta(\boldsymbol{x}, \boldsymbol{z})$ is a formula $\gamma(\boldsymbol{x})$ such that $\alpha(\boldsymbol{x}, \boldsymbol{y}) \to \gamma(\boldsymbol{x})$ and $\gamma(\boldsymbol{x}) \to \beta(\boldsymbol{x}, \boldsymbol{z})$ are tautologies. The* size *of an interpolant is the number of gates needed to represent the interpolant as a circuit.*

Note that here, the interpolant is any kind of formula. The representation of a formula as a circuit and vice versa is possible since we can transform bounded formulas to propositional formulas (see [Kra95]) and for each connector in a propositional formula, there is a corresponding gate. In addition to that, we can represent an interpolant as set.

Therefore, we also call the set I an interpolant of $\alpha(\boldsymbol{x}, \boldsymbol{y}) \to \beta(\boldsymbol{x}, \mathbf{z})$ if it separates the sets A and B induced by $\alpha(\boldsymbol{x}, \boldsymbol{y})$ and $\beta(\boldsymbol{x}, \boldsymbol{z})$.

Lemma 3.15. *The formula $\gamma(\boldsymbol{x})$ is an interpolant of $\alpha(\boldsymbol{x}, \boldsymbol{y}) \to \beta(\boldsymbol{x}, \boldsymbol{z})$ if and only if there is a set I with $I \subseteq A$ and $I \cap B = \emptyset$ where*

$$A := \{\boldsymbol{v} \in \{0,1\}^n \mid \exists \boldsymbol{y} \in \{0,1\}^k \alpha(\boldsymbol{v}, \boldsymbol{y}) = 1\}$$
$$B := \{\boldsymbol{v} \in \{0,1\}^n \mid \exists \boldsymbol{z} \in \{0,1\}^l \neg\beta(\boldsymbol{v}, \boldsymbol{z}) = 1\}.$$

Proof. Assume we have an interpolant γ given for the tautology $\alpha(\boldsymbol{x}, \boldsymbol{y}) \to \beta(\boldsymbol{x}, \boldsymbol{z})$. We want to construct a set I such that $I \subseteq A$ and $I \cap B = \emptyset$. Define the set I as follows

$$I := \{\boldsymbol{v} \in \{0,1\}^n \mid \gamma(\boldsymbol{v}) = 1 \wedge \exists \boldsymbol{y} \in \{0,1\}^k \alpha(\boldsymbol{v}, \boldsymbol{y}) = 1\}.$$

It vacuously holds that $I \subseteq A$.

For showing that $I \cap B = \emptyset$, assume that there is an $\boldsymbol{x} \in I$ and $\boldsymbol{x} \in B$. Then, we have that $\gamma(\boldsymbol{x}) = 1$ and that there exists a $\boldsymbol{z} \in \{0,1\}^l$ such that $\neg\beta(\boldsymbol{x}, \boldsymbol{z}) = 1$. With that, we have $\beta(\boldsymbol{x}, \boldsymbol{z}) = 0$ but this contradict the fact that $\gamma(\boldsymbol{x}) \to \beta(\boldsymbol{x}, \boldsymbol{z})$ is a tautology.

For the other direction, we see that the following set is the biggest set such that $I \subseteq A$ and $I \cap B = \emptyset$. As a result, every other I' needs to be contained in I.

$$I := \{\boldsymbol{v} \in \{0,1\}^n \mid \exists \boldsymbol{y} \in \{0,1\}^k \alpha(\boldsymbol{v}, \boldsymbol{y}) = 1 \vee \forall \boldsymbol{z} \in \{0,1\}^l \neg\beta(\boldsymbol{v}, \boldsymbol{z}) = 0\}.$$

We want to show that there is a γ such that $\alpha(\boldsymbol{x}, \boldsymbol{y}) \to \gamma(\boldsymbol{x})$ and $\gamma(\boldsymbol{x}) \to \beta(\boldsymbol{x}, \boldsymbol{z})$ are tautologies. Let $\gamma(\boldsymbol{x}) = \exists \boldsymbol{y} \in \{0,1\}^k \alpha(\boldsymbol{x}, \boldsymbol{y}) = 1 \vee \forall \boldsymbol{z} \in \{0,1\}^l \neg\beta(\boldsymbol{x}, \boldsymbol{z}) = 0$.

Assume for contradiction that there are $\boldsymbol{x}, \boldsymbol{y}$ with $\gamma(\boldsymbol{x}) = 0$ and $\alpha(\boldsymbol{x}, \boldsymbol{y}) = 1$. But this is a contradiction if $\gamma(\boldsymbol{x}) = 0$, we have $\forall \boldsymbol{y} \in \{0,1\}^k \alpha(\boldsymbol{v}, \boldsymbol{y}) = 0$ which contradicts $\alpha(\boldsymbol{x}, \boldsymbol{y}) = 1$.

Assume for contradiction that there are $\boldsymbol{x}, \boldsymbol{z}$ with $\gamma(\boldsymbol{x}) = 1$ and $\beta(\boldsymbol{x}, \boldsymbol{z}) = 0$. If we have a $\boldsymbol{y} \in \{0,1\}^k \alpha(\boldsymbol{v}, \boldsymbol{y}) = 1$, this is a contradiction because $\alpha(\boldsymbol{x}, \boldsymbol{y}) \to \beta(\boldsymbol{x}, \boldsymbol{z})$ is tautology.

But if $\forall \boldsymbol{z} \in \{0,1\}^l \neg\beta(\boldsymbol{v}, \boldsymbol{z}) = 0$ this is also contradiction since we cannot have $\forall \boldsymbol{z} \in \{0,1\}^l \neg\beta(\boldsymbol{x}, \boldsymbol{z}) = 0$ and $\beta(\boldsymbol{x}, \boldsymbol{z}) = 0$ at the same time. □

Note that there is another definition of interpolant used by Maria L. Bonet, Tonnian Pitassi and Ran Raz in [BPR00]. For completeness, we prove the equivalence of this definition to the two preceding ones.

Lemma 3.16. *The formula $\gamma(\boldsymbol{x})$ is an interpolant of tautology $\alpha(\boldsymbol{x}, \boldsymbol{y}) \to \beta(\boldsymbol{x}, \boldsymbol{z})$ if and only if for the unsatisfiable formula $\alpha(\boldsymbol{x}, \boldsymbol{y}) \wedge \neg\beta(\boldsymbol{x}, \boldsymbol{z})$, there is a function i for the formula $\alpha(\boldsymbol{x}, \boldsymbol{y}) \wedge \neg\beta(\boldsymbol{x}, \boldsymbol{z})$ that takes a truth assignment t of $\boldsymbol{x}$ as input and outputs 1 only if $\alpha(\boldsymbol{x}, \boldsymbol{y})$ is unsatisfiable, 0 only if $\beta(\boldsymbol{x}, \boldsymbol{z})$ is unsatisfiable and an arbitrary value if both $\alpha(\boldsymbol{x}, \boldsymbol{y})$ and $\neg\beta(\boldsymbol{x}, \boldsymbol{z})$ are unsatisfiable.*

Proof. Assume we an interpolant function i given for the unsatisfiable formula $\alpha(\boldsymbol{x}, \boldsymbol{y}) \wedge \neg\beta(\boldsymbol{x}, \boldsymbol{z})$. Since $\alpha(\boldsymbol{x}, \boldsymbol{y}) \wedge \neg\beta(\boldsymbol{x}, \boldsymbol{z})$ is unsatisfiable, we have that $\alpha(\boldsymbol{x}, \boldsymbol{y}) \to \beta(\boldsymbol{x}, \boldsymbol{z})$ is tautological. Let $\gamma(\boldsymbol{x}) = i(t, \boldsymbol{x})$ for a truth assignment t and variables $\boldsymbol{x}$.

If $\alpha(\boldsymbol{x}, \boldsymbol{y}) = 1$ for any x, y, then $\gamma(\boldsymbol{x}) = 1$ for any $\boldsymbol{x}$. But if so, we have that $\alpha(\boldsymbol{x}, \boldsymbol{y}) \to \gamma(\boldsymbol{x})$ is a tautology. In addition, we have that $\gamma(\boldsymbol{x}) \to \beta(\boldsymbol{x}, \boldsymbol{z})$ is a tautology.

If $\alpha(\boldsymbol{x}, \boldsymbol{y}) = 0$ for some $\boldsymbol{x}, \boldsymbol{y}$, we have that $\beta(\boldsymbol{x}, \boldsymbol{z}) = 1$ for any $\boldsymbol{x}, \boldsymbol{y}$ since $\alpha(\boldsymbol{x}, \boldsymbol{y}) \to \beta(\boldsymbol{x}, \boldsymbol{z})$ was tautological. But then, we have that $\alpha(\boldsymbol{x}, \boldsymbol{y}) \to \gamma(\boldsymbol{x})$ are $\gamma(\boldsymbol{x}) \to \beta(\boldsymbol{x}, \boldsymbol{z})$ is a tautologies.

Assume there is a γ such that $\alpha(\boldsymbol{x}, \boldsymbol{y}) \to \gamma(\boldsymbol{x})$ and $\gamma(\boldsymbol{x}) \to \beta(\boldsymbol{x}, \boldsymbol{z})$ are tautologies and let $i(t, \boldsymbol{x}, \boldsymbol{y}) = \neg\gamma(\boldsymbol{x})$ If $\gamma(\boldsymbol{x}) = 0$, we have because $\alpha(\boldsymbol{x}, \boldsymbol{y}) \to \gamma(\boldsymbol{x})$ that $\alpha(\boldsymbol{x}, \boldsymbol{y}) = 0$ for all $\boldsymbol{x}, \boldsymbol{y}$, thus α is unsatisfiable. Similarly, if $\gamma(\boldsymbol{x}) = 1$, we have with $\gamma(\boldsymbol{x}) \to \beta(\boldsymbol{x}, \boldsymbol{z})$ that $\beta(\boldsymbol{x}, \boldsymbol{z}) = 1$ for all $\boldsymbol{x}, \boldsymbol{z}$, so $\neg\beta$ is unsatisfiable. □

From now on, we switch between different representations of an interpolant: The interpolant appears as a formula, as a circuit or as a set. The notion of differentiating two sets in polynomial time was also called 'The canonical NP/coNP-pair of α is polynomially separable' in [AB04], Definition 6, page 187. We see that the choice of A_0 and A_1 are in the case of RSA canonical and in the definition by Atserias and Bonet, they constitute the NP/coNP-pair.

In sequence, we build on the definition of interpolation of a problem by specifying the size of the interpolating circuit. Intuitively, it makes sense that a problem that proves to be difficult to solve such as RSA, is not interpolated by small size circuits.

Definition 3.17. *We say that a problem has a* feasible interpolation *if there is a polynomial size circuit called the interpolating circuit C such that for every tautology φ of the form $\alpha(\boldsymbol{x}, \boldsymbol{y}) \to \beta(\boldsymbol{x}, \boldsymbol{z})$, we have that there is a truth assignment t on the variables $\boldsymbol{x}$ such that*

$$C_\varphi(t) = \begin{cases} 0 \text{ if and only if } \alpha(t(\boldsymbol{x}), \boldsymbol{y}) \text{is a tautology} \\ 1 \text{ if and only if } \neg\beta(t(\boldsymbol{x}), \boldsymbol{z}) \text{is a tautology.} \end{cases}$$

Equivalently, we also say that the circuit C is a feasible interpolation of the problem of separating the sets if C is polynomial in the input size and if

$$A := \{\boldsymbol{t} \in \{0,1\}^n \mid \exists \boldsymbol{y} \in \{0,1\}^k \alpha(\boldsymbol{t}, \boldsymbol{y}) \text{ is true}\}$$

and

$$B := \{\boldsymbol{t} \in \{0,1\}^n \mid \exists \boldsymbol{y} \in \{0,1\}^l \neg\beta(\boldsymbol{t}, \boldsymbol{z}) \text{ is true}\}.$$

To show non-fpt-automatability with regard to tree width, we are constructing a contradiction on the feasible interpolation of RSA. RSA is a widely used encryption scheme, thus, is used to encode bits that are believed to be safe from decryption in polynomial size circuits by an attacker who does not know about the message or the secret key [Ale+88]. As a result, RSA is likely not to admit an feasible interpolation.

There are many characterizations of interpolation and therefore, interpolation works well in describing a problem of separating two sets. The RSA problem was our example for an application of a problem where we need to separate two sets and it is intuitive that other problems can be as well formulated as separating two sets.

The following definition leads to a theorem that is a second motivation why it makes sense to consider the term interpolation.

Definition 3.18. *We say P is* closed under restrictions *if, whenever $\varphi(x)$ has an P-proof of size m, any restriction of $\varphi(x)$ by a partial truth assignment on x has an P-proof of size at most polynomial in m.*

Here, we can think of closed under restriction as a certain kind of monotonicity: Even if we restrict the truth assignment function, thus, only map some elements of the domain to the image, we can find a corresponding proof.

As one might have noted, we switched from Theorem 3.8 that provides a polynomial proof search algorithm to an attacker of an encryption scheme that has polynomial size circuits as resources. We can do so, due to a connection between interpolation and automatability for some proof systems.

Lemma 3.19. *If $EF \vdash \alpha \to \beta$ for a formula $\alpha \to \beta$ and we can find a proof of $\alpha \to \beta$ in time polynomial in $|\alpha \to \beta|$, then there is an interpolating function i for $\alpha \to \beta$ that is polynomial in its input.*

Proof. This proof is a modification of Theorem 1.1 in [BPR00].

With $EF \vdash \alpha \to \beta$, we can compute an Extended Frege proof π of $\alpha \to \beta$ in time $p(n)$ for a polynomial p and $|\alpha \to \beta| =: n$ with an algorithm A. Since EF is closed under restrictions, we can take the algorithm A and check if it returns a proof of $\neg\alpha$ in time $p(|\neg\alpha|)$. If so, define that i returns 0. Otherwise, let i return 1. This is an interpolant since if A cannot find a proof of $\neg\alpha$, there is no proof of $\neg\alpha$ since A could find a proof of $\alpha \to \beta$ in time $p(n)$. But if there is no proof of $\neg\alpha$, then α is a tautology. Similarly, we have that there is a proof of $\neg\alpha$ in time $p(|\neg\alpha|)$, we see that α cannot be a tautology and since $\alpha \to \beta$ is a tautology, we have that β is a tautology.

We have that i is polynomial in its input size because we run A twice polynomial long in at most n. □

More general, we can cite the following connection between automatability and interpolation. Here, we denote with *weak automatability* of a proof system P that there exists proof system Q and an algorithm that, given a tautology, produces a Q-proof of it in time polynomial in its smallest P-proof (introduced by [AB04]).

Theorem 3.20. *For a proof system proving its own soundness and that is closed under restrictions, feasible interpolation is equivalent to weak automatability.*

For the direction from feasible interpolation to automatability, see Theorem 1.1 in [BPR00], for the other direction see Proposition 3.6 in [Pud03].

Since Extended Frege proves its own soundness and is closed under restrictions [Kra95], we obtain the following corollary. Note that the proof system resolution does not prove its own soundness [AB04] and thus, the corollary would not be true for resolution.

Corollary 3.1. *In Extended Frege, weak automatability is equivalent to feasible interpolation.*

With the definition of interpolation, we can put together the tools we have gathered so far, to prove the a non-fpt-automatability result with regard to proof tree width.

3.3 Tree width

The tree width of a graph is a measure to estimate how close the graph resembles a tree. Throughout the work, we call a tree an acyclic, finite graph. If the tree width of a graph equals one, the graph is a tree, if the tree width is a very high integer, say 1234, the graph needs a lot of modification before it can be turned into a tree. More formally, we define *tree width* as in Definition 10.1 in [Nie06].

Definition 3.21. *Let $G = (V, E)$ be a graph. A* tree decomposition *of G is a pair $\langle \{X_i \mid i \in I\}, T\rangle$ where each X_i is a subset of V , called a* bag*, and T is a tree with the elements of I as nodes. The following three properties must hold:*

1. $\bigcup_{i \in I} X_i = V$
2. *for every edge $\{u, v\} \in E$, there is an $i \in I$ such that $\{u, v\} \subseteq X_i$*
3. *for all $i, j, l \in I$, if j lies on the path between i and l in T then $X_i \cap X_l \subseteq X_j$.*

The width *of $\langle \{X_i \mid i \in I\}, T\rangle$ is defined as $\max\{|X_i| \mid i \in I\} - 1$. The* tree width *of G ($tw(G)$) is the minimum k such that G has a tree decomposition of width k.*

Since the tree width is a measure to determine how close a graph is a tree, it seems reasonable that tree have tree width one.

Lemma 3.22. *Any tree $T = (V, E)$ has tree width one.*

Proof. For any edge in T, we introduce a bag containing this edge. If two bags have a nonempty intersection, we introduce an edge in the underlying graph of the tree decomposition.

With that we have that property one of the definition is satisfied since $\cup_{e \in E} \{e\} = V$. Furthermore, we have property two since every edges equals an X_i. In addition, we have property three, since for any path (i, j, l) of three different vertices i, j, l, we have that $X_i \cap X_l = \emptyset$. Finally, we have that the underlying graph of the tree decomposition is actually a tree since, if we had a cycle, then, there is a cycle in G. Since G was a tree, we can conclude that any tree $T = (V, E)$ has tree width one. □

In the following, we look at an example that shows a graph that is not a tree and the corresponding tree decomposition.

Example 3.23. *Take the graph $G = (V, E)$ depicted as follows.*

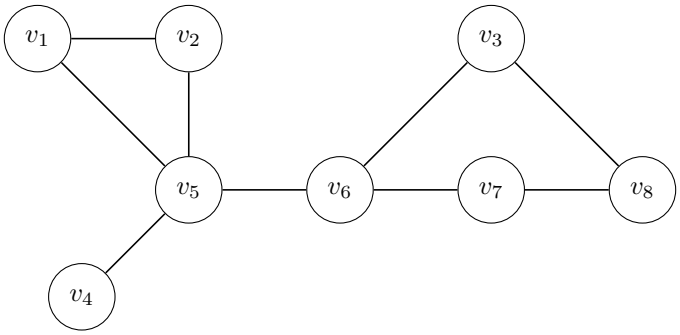


Figure 3: Graph G

The following figure shows a tree composition of graph G. Formally, the tree decomposition can be described as $\langle\{v_1, v_2, v_4, v_5\}, \{v_4, v_5\}, \{v_5, v_6\}, \{v_3, v_6, v_7, v_8\}, T\rangle$ and T consists of three vertices and two edges $(X_1, X_2), (X_2, X_3), (X_3, X_4)$ and (X_4, X_5). The structure $\langle\{X_i \mid i \in \{1, 2, 3, 4\}, T\rangle$ fits the definition of a tree composition since we see that T forms a tree and every vertex is in one of the three dashed bags and each edge is contained in at least in of the bags. Furthermore, we have that $X_1 \cap X_4 = \emptyset \subseteq X_3 = \{v_5, v_6\}$, $X_2 \cap X_4 = \emptyset \subseteq X_3 = \{v_5, v_6\}$ and $X_1 \cap X_2 = \{v_5\} \subseteq X_3 = \{v_5, v_6\}$.

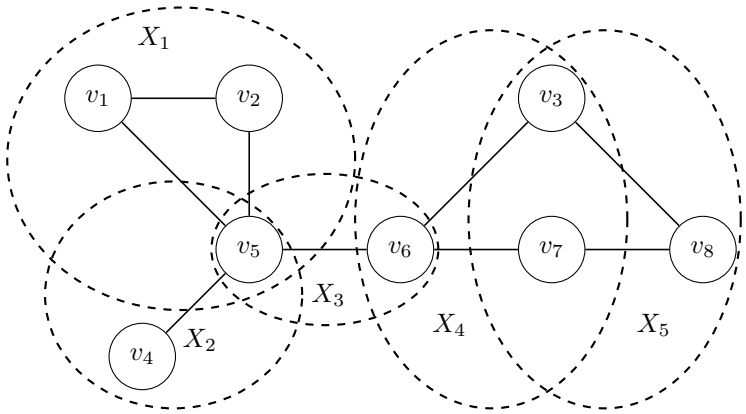
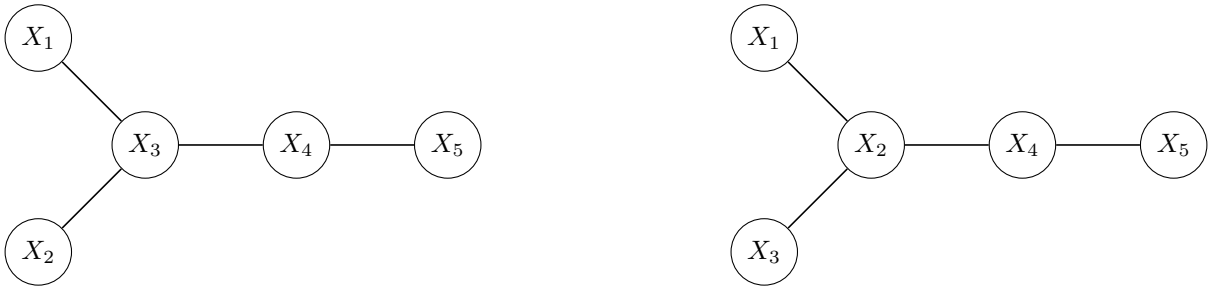


Figure 4: The bags of a tree decomposition of G

We see that this sketch does not yet uniquely define the tree decomposition since several choices for T are possible. In the following, we sketch two possible T.



The width of this tree composition is 2 because the biggest bag X_1 contains 3 elements. We see that this is the smallest width for a tree decomposition since for T to be a tree, we have to include cliques in a bag, otherwise, we violate condition 3. Furthermore, we cannot use several bags to cover a clique due to the following Lemma.

Lemma 3.24. *If a graph G contains a clique C, for a tree decomposition the clique must be contained in a bag.*

Proof. Assume there was a tree decomposition $\langle\{X_i\}_i \mid T\rangle$ and C is contained in two bags X_1 and X_2 but for neither of them, we have $C \subseteq X_1$ or $X \subseteq X_2$. Subsequently, we have a $v \in X_1 \setminus X_2$ and a

$w \in X_2 \setminus X_1$. But since C is a clique, there is an edge between v and w and because of $v \in X_1 \setminus X_2$ and a $w \in X_2 \setminus X_1$, we have that the edge is neither contained in X_1 nor in X_2. If there is no additional bag containing $\{v, w\}$, we have a contradiction with the second property of the definition.

But if we have that additional bag there is no way to align the three bags to in a path such that property thre is true.

As a result, all cliques must be contained in a bag! □

Due to the central notion of parameter based on proof graphs, we abbreviate fpt-automatability with regard to the minimal candidate value.

Definition 3.25. *For any proof system P, and P-proof π, computable function f and proof graph G_π (for Frege systems) or associated graph with π of a tautology φ and polynomial p, we abbreviate the parameter function κ of G with regard to the minimal candidate $v_{\varphi,f,\kappa,P,p}$ as* proof parameter function *of G. Equally, we denote the proof tree width of G as $\kappa(G)$.*

With the definition of proof tree width and the assumption that RSA is secure, we have the necessary definitions to prove non-fpt-autmatability with regard to treewidth. For this, we are going to take a formula that expresses the irreconcilability of two sets A_0 and A_1. The two sets are constructed such that they encode the problem RSA. The RSA encryption scheme can be deciphered by the inverse logarithm, nonetheless, computing the inverse logarithm is in practice not feasible [McC90].

As a result, we want to contrast that RSA can be deciphered if we assume fpt-automatability and more formally, we want to show that we can prove in S_2^1 that we can distinguish the sets A_0 and A_1, nonetheless, this distinction should not lead to an effective algorithm to break RSA.

Lemma 3.26. *There is a proof in S_2^1 for the formula*

$$\exists x, d, r.\alpha_0(n, e, y, x, d, r) \to \neg\exists x, d, r.\alpha_1(n, e, y, x, d, r).$$

Proof. See also proof of Theorem 1 in [KP98].

We want to show that in S_1^2 for every y there is at most one $x < n$ which satisfies the defining formula of either A_0 or A_1 which are for $i \in \{0, 1\}$ defined as follows:

$$A_i := \{(n, e, y) \mid \exists x, d, r < n.x \equiv i \mod 2 \wedge x^e \equiv y \mod n \wedge y^e \equiv x \mod n \wedge y^r \equiv 1 \mod n \wedge gcd(e, r) = 1\}$$

Suppose that there are two $x_0, x_1 < n$ and r_0, d_0, d_1 with $y^{r_0} \equiv 1 \mod n, gcd(e, r_0) = 1$ that satisfy

$$x_i^e \equiv y \mod n \wedge y^{d_i} \equiv x_i \mod n$$

for $i \in \{0, 1\}$. From this, it follows that

$$x_i^{r_0} \equiv (y^{d_i})^{r_0} \equiv (y^{r_0})^{d_i} \equiv 1 \mod n$$

for $i \in \{0, 1\}$. Due to the Extended Euclidean algorithm and $gcd(e, r_0) = 1$, we have that there exists an inverse d' to $e \mod r_0$, thus, for an integer k we have $d'e = 1 + kr_0$. With $x_i^{r_0} = 1 \mod n$, for $i \in \{0, 1\}$ we have

$$y^{d'} \equiv x_i^{ed'} \equiv x_i^{1+kr_0} \equiv x_i \mod n,$$

whence $x_0 = x_1$ since $y^{d'}$ independent of i. Subsequently, for any y there is at most one x satisfying the given formulas of either A_0 or A_1. □

Together with the Lemma and the assumption fpt-automatability, in the following theorem, we want to use the tools introduced in the previous sections and construct a contradiction to RSA security.

Theorem 3.27. *If the RSA protocol is secure, Extended Frege is non-fpt-automatable with regard to proof tree width.*

Proof. This proof builds on Corollary 10 in [KP98].

For $i \in \{0, 1\}$, consider the following set:

$$A_i := \{(n, e, y) \mid \exists x, d, r < n.x \equiv i \mod 2 \wedge x^e \equiv y \mod n \wedge y^e \equiv x \mod n \wedge y^r \equiv 1 \mod n \wedge gcd(e, r) = 1\}$$

Let $\exists x, d, r.\alpha_i(n, e, y, x, d, r)$ be the Σ_1^b-formula that defines A_i. We see that this is actually a Σ_1^b-formula since the range of x, d and r is bounded and only one existential quantifier appears at the start of the formula. Subsequently, the following formula is a Π_1^b-formula:

$$\psi := \exists x, d, r.\alpha_0(n, e, y, x, d, r) \to \neg\exists x, d, r.\alpha_1(n, e, y, x, d, r)$$

We have that

$$S_2^1 \vdash \psi$$

due to Lemma 3.26.

Because ψ is a Π_1^b-formula, we get via Theorem 3.8 a proof translation to G_1^*.

Due to Lemma 3.10 and Lemma 3.11, we can polynomially simulate this proof to Extended Frege and obtain a treelike Extended Frege proof π that has polynomial size in the length n of the proposition ψ.

Assume that Extended Frege is fpt-automatable with regard to proof tree width. Then, every tautology in S_2^1 has a computable function f and a P-proof π in polynomial time in the minimal candidate value $v_{\psi,f,EF,p''}$. As a result, we can find a computable function f such that the size of π is upper bounded by $f(k)p''(n)$ with k the tree width of the associated graph of π. Since π is treelike with Lemma 3.22, we can upper bound the size of π by $f(1)p''(n)$. Since Extended Frege is sound, we follow with $S_2^1 \vdash \psi$ that ψ is a tautology. Therefore, we can find a proof of the proposition ψ for a polynomial p' in time $p'(v_{\psi,f,EF,p''})$. Furthermore, we upper bound p' by a polynomial p that is monotone increasing which is possible since we can polynomially upper bound any monotone decreasing part.

By definition of the minimal candidate value with regard to the tautology ψ, the function f and the proof system EF, we have $v_{\psi,f,EF,p''} \leq f(1)p''(n)$ and since p is monotone increasing, it follows that $p(v_{\psi,f,EF,p''}) \leq p(f(1)p''(n))$. With that, we can upper bound the time needed to find a proof of ψ by $p(f(1)p''(n))$, thus, by a polynomial in n.

By Lemma 3.19, there is a circuit with size polynomial in n that interpolates ψ. But we have assumed that RSA is secure, thus, that no attacker can differentiate A_0 and A_1 with circuits in polynomial size in n. Contradiction! It follows that Extended Frege is not fpt-automatable with regard to tree width. □

3.4 Fpt-dominance

With the help of fpt-dominance, we aim to link different parameters to each other and transfer fpt-automatability results from one parameter to the other. The preceding result on non-fpt-automatability with regard to tree width will be our base for this.

Definition 3.28. *Call parameter k dominated by parameter k' if there exists a computable and surjective function g over non-negative integers such that $k \leq g(k')$.*

In the following, we use fpt-dominance to deduce results on fpt-automatability.

Lemma 3.29. *If k is dominated by k' and a proof system P is fpt-automatable with regard to parameter k, then the proof system is also fpt-automatable with regard to parameter k'.*

Conversely, if k is dominated by k' and a proof system P is non-fpt-automatable with regard to parameter k', then the proof system is also non-fpt-automatable with regard to parameter k.

Proof. Assume the proof system P is fpt-automatable with regard to the parameter k. With that, we can find a proof for any tautology φ in polynomial time p in the minimal candidate value $v_{\varphi,f,\kappa,P,p}$ for a computable function f. Without loss of generality, we assume f to be monotone increasing. Let the time needed to find φ be captured by the function $p'(f(k)p(n))$ for a monotone polynomial p' with $n := |\varphi|$.

But if k is dominated by k', we have that the proof search algorithm with regard to parameter k, should output a proof in time $p'(f(g(k'))p(n))$ for the computable composition $f \circ g$. We conclude that P is fpt-automatable with regard to k'.

Assume that the proof system P is non-fpt-automatable with regard to the parameter k'. With that, we cannot find a proof search algorithm for a tautology φ in polynomial time p in the minimal candidate value $v_{\varphi,f,\kappa',P,p}$ for a computable function f. Without loss of generality, we assume f to be monotone increasing. Thus, we can assume that the best algorithm exceeds the time $p'(f(k)p(n))$ for a monotone polynomial p' with $n := |\varphi|$.

But if k is dominated by k', we have that the proof search algorithm with regard to parameter k, cannot output a proof in even less time $p'(f(g^{-1}(k))p(n))$ where g^{-1} denotes the left-inverse that exists since g was surjective. We conclude that P is non-fpt-automatable with regard to k. □

Similarly, we can prove a relationship for fpt-boundedness.

Lemma 3.30. *If k is dominated by k' and a proof system P is fpt-bounded with regard to parameter k, then the proof system is also fpt-bounded with regard to parameter k'. Conversely, if k is dominated by k' and a proof system P is not fpt-bounded with regard to parameter k', then the proof system is also not fpt-bounded with regard to parameter k.*

Proof. Assume that a proof system P is fpt-bounded with regard to parameter k. With that, any proof of a formula has size at most $f(k)n^{\mathcal{O}(1)}$. Without loss of generality, we assume f to be monotone increasing. Due to fpt-dominance, any proof of a formula has size at most $f(g(k'))n^{\mathcal{O}(1)}$ and with that P is fpt-bounded with regard to parameter k'.

Assume that a proof system P is not fpt-bounded with regard to parameter k'. With that, there is a formula and a proof that exceeds $f(k')n^{\mathcal{O}(1)}$. Without loss of generality, we assume f to be monotone increasing. But if k is dominated by k', we have that the proof needs to exceeds $p(f(g^{-1}(k))n^{\mathcal{O}(1)})$ where g^{-1} denotes the left-inverse that exists since g was surjective. We conclude that P is not fpt-bounded with regard to k. □

By applying these lemmas, an other parameter comes along with a natural non-fpt-automatability result.

3.5 Clique width

Clique width is a parameter function taking a graph as input and returning a natural number. Here, a general motivation is to quantify how close a graph is to containing no clique. On the contrary to tree width or path width, it is not defined via a decomposition of a graph G but via steps needed to build G. Each such building step refers to an algebraic expression and therefore, a chain of algebraic expressions translates to an instruction on how to build G.

Formally, we are going to define *clique width* as in [Fel+09]. For this, we take an initial graph G defined as follows:

Definition 3.31. *A k-graph is a graph whose vertices are labeled by integers from $\{1, ..., k\}$. We call the k-graph consisting of exactly one vertex v (say, labeled by $i \in \{1, ..., k\}$) an initial k-graph and denote it by $i(v)$.*

From this class of graphs, we build the given graph G with three different operations. Depending on how many different elements the initial graph must contain, we obtain the *clique width.*

Definition 3.32. *The* clique width *$clw(G)$ of a graph G is the smallest integer k such that G can be constructed from initial-graphs through repeated application of the following three operations.*

- *Disjoint union (denoted by $\oplus$)*
- *Relabeling: changing all labels i to j (denoted by $\rho_{i\to j}$)*
- *Edge insertion: connecting all vertices labeled by i with all vertices labeled by j, $i \neq j$ (denoted by $\eta_{i,j}$ or $\eta_{j,i}$)*

Let us consider two examples of graphs G, the corresponding algebraic term describing G and its clique width.

Example 3.33. *Consider the graph G (also known as K_3).*

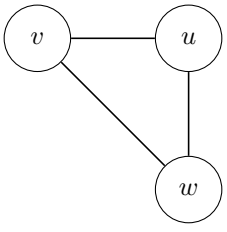


Figure 6: Graph G

The following expression builds G from the initial 2-graphs:

$$\eta_{1\to 2}(\rho_{1,2}(\eta_{1\to 2}(2(u) \oplus 1(v))) \oplus 2(w))$$

The operation $\rho_{1,2}(\eta_{1\to 2}(2(u) \oplus 1(v))$ inserts an edge between vertex u and v. Then, we relabel, so that both u and v have the label 1. With the edge insertion, we add the edges $\{v, w\}$ and $\{u, w\}$.

We have that we can build this graph from the initial graphs consisting of the vertices $2(u), 1(v)$ and $2(w)$. As a result, G is built from initial 2-graphs since the vertices u, v and w are labeled with only two different labels 1 and 2. Therefore, G has clique width two.

The second example is more complicated but shows the expressivity of the three operators.

Example 3.34. *Take the following graph G.*

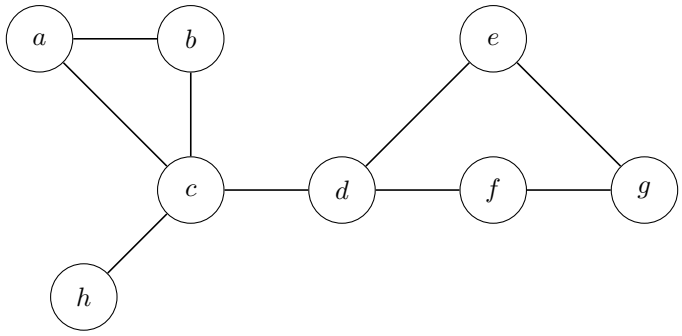


Figure 7: Graph G

We see that the first example is contained in the second example and we modify it, so that we obtain the graph induced by the vertices a, b, c, and h with the term

$$\eta_{1\to 2}(\rho_{1,2}(\eta_{1\to 2}(2(u) \oplus 1(v)) \oplus 2(w)) \oplus 1(h)).$$

Now, we want to build the graph induced by the vertices d, e, f and g without interference to what we have built so far. As a result, the vertices a, b, c and h must have a different label, so that we do not build unwanted edges while building the graph induced by d, e, f and g. Here, we see that we need d to have a different label than a, b, c and h and with that, the clique width is to be at least 3.

We see that

$$\eta_{1,3}(\underbrace{\rho_{1\to 2}(\eta_{2,3}(\eta_{1\to 2}(\rho_{1,2}(\eta_{1\to 2}(2(u) \oplus 1(v)) \oplus 2(w)) \oplus 1(h) \oplus 3(d))))}) \oplus 1(e) \oplus 1(f) \oplus 3(g))$$

builds the remaining graph. The bracket denotes the graph induced by a, b, c, h and d with label 2 for a, b, c and h and label 3 for vertex d. We conclude that G has clique width three.

In Example 3.23, we have seen that the same graph as in the example above has tree width two. This is no coincidence since in general, we have due to [CO00] that

Theorem 3.35. *For any graph G, $clw(G) \leq 3 \cdot 2^{tw(G)-1}$.*

But with that, we have that tree width dominates clique width and therefore, we can apply fpt-dominance with Lemma 3.29.

Corollary 3.2. *Extended Frege is non-fpt-automatable with regard to proof clique width if RSA secure.*

In sum, we can see that fpt-dominance is a useful method to deduce automatability results. We see that we could artificially construct many more parameters that are dominated by tree width by relieving the constraints on the tree decomposition or the algebraic operators. For example, we obtain a modified clique width parameter if we only allow the operators *disjoint union* and *edge insertion*. Nonetheless, this parameter is very limited in its expression and does not go along with the general intuition of a clique width parameter.

As a result, we focus on the other direction. Namely, we focus on parameters that dominate tree width. With that, we cannot apply fpt-dominance, nor show fpt-automatability or non-fpt-automatability.

The general motivation in looking for parameters "above" tree width is, first, to find a fpt-automatability result and a very close non-fpt-automatability result, so that we can reason about properties that are responsible for the jump from non-fpt-automatability to automatability. Note, that we would have have a jump here, since the relation of fpt-dominance is transitive and therefore, if we draw all parameters as a lattice structure, we can fit a line such that above, all parameters are fpt-automatable and below, all parameters are non-fpt-automatable.

A second motivation are the parameters themselves. If they bear a certain structure, we come closer to what is relevant to automation. Parameters such as tree width and clique width come with the natural intuition of how much we need to modify a graph to obtain either a tree or clique-free graph. Other parameters might give information via the structure that they inherit.

3.6 Special tree width and intersection graphs

Intersection graphs are graphs that are obtained by defining a graph from a class of graphs. The theory of intersection graphs comes from graph theory and has many mathematical applications (see [MM99]).

Definition 3.36. *The* intersection graph *of a family of graphs F is the graph G_F whose vertices are the members of the family such that two distinct vertices f, f' of G_F are adjacent, if and only if the corresponding graphs have a common vertex.*

We can also build a family of graphs from a single graph and then, build the intersection graph from the graph family. For example, we can split a graph into all its paths. These paths are graphs themselves, thus, form a graph family. From the collection of paths, we can build an intersection graph. This is captured in the following definition.

Definition 3.37. *A* path graph *of a graph G is the intersection graph built from the set of paths between any two vertices in G.*

Consider the following example.

Example 3.38. *Take the following graph and the set $\{e, d, f\}, \{e, g\}, \{e, h\}, \{d, f\}$ of undirected paths.*

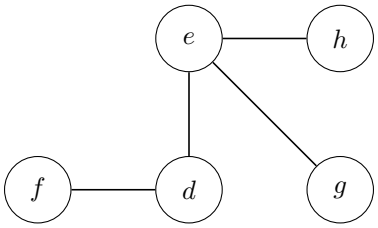


Figure 8: Tree T rooted in e

Then, we obtain the following path graph with the given paths as vertices and edges whenever there is a nonempty intersection between sets associated with the vertices.

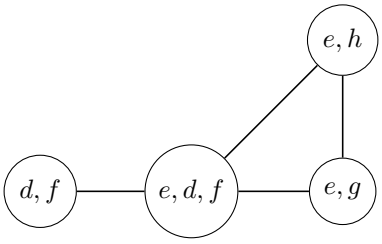


Figure 9: Path graph on T

As a result of this, we can further section classes of path graphs as it was introduced by Monma and Wei [MW86].

Definition 3.39. *A graph is an* undirected vertex path graph *(UV graph) if it is the intersection graph of a set of paths in a tree. Similarly, call a graph a* directed vertex path graph *(DV graph) if it is the intersection graph of a set of directed paths in a directed tree and call a graph* rooted directed vertex path graph *(RDV graph) if it is the intersection graph of directed paths in a rooted tree.*

In the following, consider a modification of Example 3.38 for a graph that is an DV graph but doesn't qualify as an RDV graph since the underlying graph is not rooted.

Example 3.40. *Take the directed paths $(e, d, f), (e, g), (e, h), (d, f), (i, f)$. Then, we can construct from the tree on the left the DV graph on the right. We see that the graph on the left is not rooted since both vertex i and vertex e have only incoming edges.*

Figure 10: Path graph on T

With the new definitions in mind, we can define some new graph parameters. These graph parameters relate to proof complexity since we obtain a graph by representing a proof in for example Frege calculus and on this graph, we can apply graph parameters. Intersection graphs give additional information on parameters. This is best seen by a characterization result on tree width by Bodlaender [Bod98].

Theorem 3.41. *A graph has tree width at most k if and only if it is a subgraph of a chordal graph with clique number at most $k+1$.*

The property of having a certain tree width is translated to containment in a class. We define chordal graphs as follows:

Definition 3.42. *A* chord *in a cycle of a graph, is a pair of adjacent vertices on the cycle that are not consecutive on the cycle. A graph is* chordal*, if each cycle with length at least four has a chord.*

Let us test the theorem by checking that an example with a certain tree width has the guaranteed clique number.

Example 3.43. *Take the following graph G from Example 3.23 that has tree width two.*

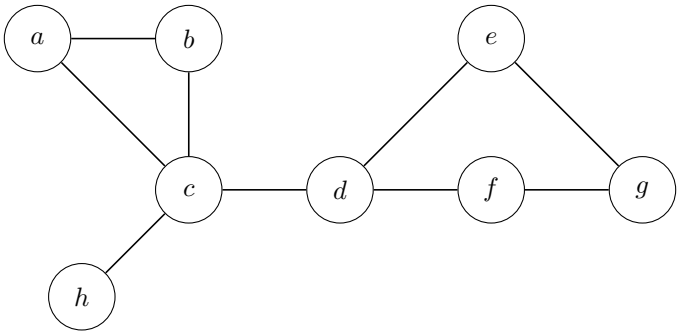


Figure 11: Graph G with tree width two

We want to check that the graph has clique number at most three and this is the case since the biggest clique has size 3: The vertices a, b and c form a clique whereas the vertices e, d, f and g form only a cycle.

Furthermore, we see that the graph is a subgraph of a chordal graph since there is only one cycle with length at least four and if we add an edge between e and f, we have that the vertices e and f are vertices on the cycle that are not consecutive and therefore, e and f form a chord.

Theorem 3.41 also shows that we could define tree width via containment in the class of chordal graphs. This poses the question of defining parameters via containment in other classes. In sequence, we define parameters via containment in the classes of UV graphs, DV graphs and RDV graphs as it is done in [Bod+17]. The parameter *special tree width* was first introduced by Courcelle in [Cou12].

Definition 3.44. *A graph has* special tree width *at most k if and only if it is a subgraph of an RDV graph with clique number at most $k+1$. A graph has* directed spaghetti tree width *at most k if and only if it is a subgraph of a DV graph with clique number at most $k+1$. A graph has* spaghetti tree width *at most k if and only if it is a subgraph of a UV graph with clique number at most $k+1$.*

The naming *width* already indicates that there is an equivalent definition for the parameters that depend on tree decompositions. Let us define the following decompositions:

Definition 3.45. *A* special tree decomposition *of a graph G is a tree decomposition $\langle \{X_i\}_i \mid T\}\rangle$ where T is a rooted tree and for every vertex v in G, the bags containing v induce a directed path in T. Similarly, a* directed spaghetti tree decomposition *of a graph G is a tree decomposition $\langle \{X_i\}_i \mid T\}\rangle$ where T is a directed tree (not necessarily rooted) and for every vertex v in G, the bags containing v induce a directed path in T and a* spaghetti tree decomposition *of a graph G is a tree decomposition $\langle \{X_i\}_i \mid T\}\rangle$ where for every vertex v in G, the bags containing v induce a path in T .*

These tree decompositions lead to an alternative definition of the parameters spaghetti tree width, directed spaghetti tree width and special tree width.

Theorem 3.46. *A graph G has special tree width at most k if and only if the minimum width $sctw(G)$ of all special tree decompositions of G is at most k.*

We show both directions by giving a concrete algorithm how to make a special tree decomposition from an RDV graph and the other way around. For this, we first walk through an example.

Example 3.47. *We take the same graph as in Example 3.38 and note that the underlying graph is a rooted, directed graph, therefore, we can construct a RDV graph on paths $(e,d,f),(e,g),(e,h),(d,f)$ from it.*

We note that the RDV graph has clique number three.

Therefore, we want to construct a special tree decomposition with the size of the biggest bag at most three.

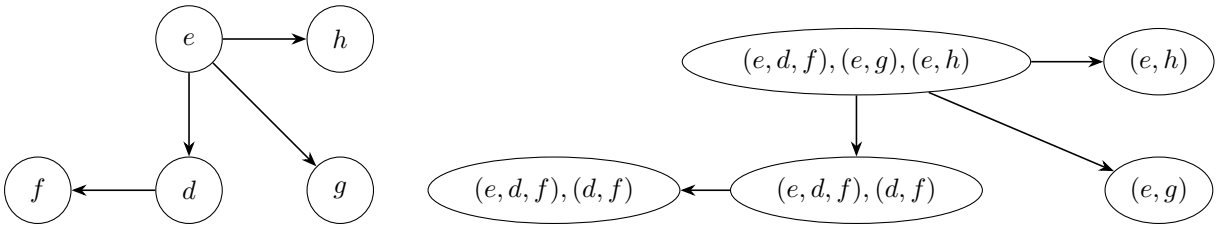


Figure 12: Constructing a special tree decomposition (right) from the underlying graph T (left) of an RDV graph

We construct the special tree decomposition as follows: For each vertex v in T, we introduce a bag and write all paths in the bag that contain v. We introduce an edge in the special tree decomposition for every edge in T.

We see that the special tree decomposition is rooted, directed and induces a path for each vertex in the RDV graph. Furthermore, it has the biggest bag size of three. It might not be the special tree decomposition with the minimal width but the minimal width can be only lower than three.

The other direction of the proof takes a special tree decomposition and constructs the underlying graph for an RDV graph.

For this, we replace each of the bags by a fresh vertex and keep the edges. With that, we obtain the underlying graph on the left.

Keeping the example in mind, we want to prove the theorem.

Proof. Assume that a graph G has special tree width at most k. We want to construct a special tree decomposition whose biggest bag size is at most $k+1$.

Since G is a subgraph of an RDV graph for paths P with clique number at most $k+1$, we construct a graph as before in the example: For each vertex v in T, we introduce a bag and write all paths in the bag that contain v. We introduce an edge in the tree decomposition for every edge in T.

We see that the tree decomposition is directed and rooted since T must be directed and rooted. To show that any vertex v in G, the bags containing v induce a directed path in the tree decomposition, take such a vertex v in G. Since G is a path graph, the vertex is a directed path $v = (p_1,\dots,p_n)$ in P. Per definition, we have that the bag contains all paths that visit p_1 naturally contains $(p_1,\dots,p_n)$. Similarly, we have that the bag contains all paths that visit p_2 naturally contains $(p_1,\dots,p_n)$ and since $(p_1,\dots,p_n)$ is a path in T, there is an edge between those two vertices. As a consequence, we have a directed path that is induced by v. We have that no other bag contains v since the other bags b' are defined by containment of other v' and $b' \notin (p_1,\dots,p_n)$.

Furthermore, we have that $scrw(G) \leq k$ since the special tree decomposition we have constructed has a biggest bag size $k+1$ since the RDV graph has clique of size $k+1$ if and only if intersection of $k+1$ paths is nonempty.

For the other direction, we assume that we have a special tree decomposition with $sctw(G) \leq k$. Let the underlying graph T of the tree decomposition be the underlying graph of an RDV graph. Since the tree decomposition is special, we have that T is rooted and directed. Let P be the set of all paths in T that are induced by a vertex in a bag of the special tree decomposition. By definition of a special tree decomposition, these paths are directed. We have that the RDV graph on T and P has clique number at most $k+1$ since the biggest bag size is at most $k+1$, thus, any $k+2$ elements of P must have an empty intersection. □

Since DV and UV graphs are defined similarly to RDV, the following corollary follows.

Corollary 3.3. *A graph G has directed spaghetti tree width at most k if and only if the minimum width $dspghtw(G)$ of all directed spaghetti tree decompositions of G is at most k and has spaghetti tree width at most k if and only if the minimum width $spghtw(G)$ of all spaghetti tree decompositions of G is at most k.*

Proof. Since the proof is very similar for RDV, DV and UV graph, we prove the corollary for the UV case.

Assume that the graph G has spaghetti tree width at most k. Thus, G is a subgraph of a UV graph, allowing the underlying tree of the tree decomposition to contain more than one root. We use the same construction as in the theorem for RDV graphs. Per construction and definition of a path graph, we have that any vertex $v \in G$ induces an undirected path in the spaghetti decomposition. Furthermore, we have that the width of the tree decomposition is at most $k+1$ since we have the clique number is at most $k+1$, so that the intersection of more than $k+1$ elements in P is empty. But if it is empty, we know that for a vertex v, we cannot add more $p \in P$ to a bag, therefore, the biggest bag size is bounded by $k+1$.

For the other direction, assume $sptw(G) \leq k$ and we also follow the construction of the theorem. We have that the RDV graph on T and P has clique number at most $k+1$ since the biggest bag size is at most $k+1$, thus, any $k+2$ elements of the set of undirected paths P induced by the decomposition must have an empty intersection. □

The equivalent formulation of the parameters spaghetti tree width, directed spaghetti tree width and special tree width makes it visible that they dominate each other.

Lemma 3.48. *For any graph G, we have $tw(G) \leq spghtw(G) \leq dspghtw(G) \leq spctw(G)$.*

Proof. Assume we have a directed tree decomposition $\langle \{X_i \mid i \in I\}, T\rangle$ with width k and that $tw(G) = k$. If for every vertex v in G, the bags containing v induce a path in T, it follows that $spghtw(G) = k$. If there is a vertex v' in G such that the bags containing v' do not induce a path, this is not a spaghetti tree decomposition. We have to increase the bag size and with that $k < spghtw(G)$. In sum, $tw(G) \leq spghtw(G)$.

Assume we have a spaghetti tree decomposition $\langle \{X_i \mid i \in I\}, T\rangle$ with width k and that $spghtw(G) = k$. If for every vertex v in G, the bags containing v induce a directed path in T, it follows that $dspghtw(G) = k$. If there is a vertex v' in G such that the bags containing v' do not induce a directed path, this is not a directed spaghetti tree decomposition. But since it is a spaghetti tree decomposition any v in G induced a path that is not necessarily directed. If any spaghetti tree decomposition $\langle \{X_i \mid i \in I\}, T\rangle$ with width k and that $spghtw(G) = k$ contains v' in G such that the bags containing v' do not induce a directed path, we have to increase the bag size and with that $k < dspghtw(G)$. In sum, $spghtw(G) \leq dspghtw(G)$.

Assume we have a directed spaghetti tree decomposition $\langle \{X_i \mid i \in I\}, T\rangle$ with width k and that $spghtw(G) = k$. If T is rooted, we have that $sptw(G) = k$. If T and any other spaghetti tree decomposition $\langle \{X_i \mid i \in I\}, T\rangle$ is not rooted, we have to increase the bag size and with that $k < spctw(G)$. In sum, $dspghtw(G) \leq sptw(G)$. □

As already mentioned, we cannot use fpt-dominance to conclude non-fpt-automatability with regard to spaghetti tree width, directed spaghetti tree width or special tree width. Rather, the fpt-dominance shows that even if we modify tree width and add restrictions to the shape of the tree decomposition, later we can show that Extended Frege is still not fpt-automatable with regard to the modified tree width parameters.

To show that Extended Frege is not fpt-automatable with regard to spaghetti tree width, directed spaghetti tree width or special tree width, we cite the following result by [Bod+17].

Proposition 3.49. *Let G be a graph. The following are equivalent.*

- *G has tree width at most one.*
- *G has spaghetti treewidth at most one.*
- *G has directed spaghetti treewidth at most one.*
- *G has special treewidth at most one.*

Thus, if we have that a tree T, we have that the tree width of T is one. As a consequence, it follows that T has spaghetti width one, directed spaghetti width one and special width one.

Theorem 3.50. *If the RSA protocol is secure, Extended Free is non-fpt-automatable with regard to proof spaghetti tree width, proof directed spaghetti tree width and proof special tree width.*

Proof. Since we have already proven this similarly for tree width, we only sketch a proof.

First, we define two sets A_0 and A_1 that only be separated if RSA is not secure. Then, we construct a treelike proof in Extended Frege that proves an implication separating the sets A_0 and A_1. Since the proof is treelike, the graph representation of the proof has spaghetti tree width, directed spaghetti tree width and special tree width at most one. With that, we can upper bound the size of the proof with respect to the constant value one of the parameters. Assuming, automatability with respect of the parameters, we obtain an effective algorithm to separate A_0 and A_1 which leads to a contradiction with the assumption of RSA security. It follows that Extended Frege is not fpt-automatable with regard to proof spaghetti tree width, proof directed spaghetti tree width and proof special tree width. □

Let us summarize what we reasoned so far: There are natural parameters generalizing tree width, namely spaghetti tree width, directed spaghetti tree width and special tree width. These parameters dominate tree width. But even though we cannot use fpt-dominance, we obtain that Extended Frege is not fpt-automatable with regard to these parameter since we can generalize the proof that shows that Extended Frege is not fpt-automatable with regard to proof tree width.

With the additional non-fpt-automatability results, we can expand our lattice of parameters since Farber showed that every RDV graph is strongly chordal [Far81].

But if every RDV graph is strongly chordal, we have that special tree width dominates *strongly chordal tree width.*

Definition 3.51. *The* strongly chordal tree width *of a graph G, denoted by sctw(G), is the minimum k such that G is a subgraph of a strongly chordal graph with clique number $k+1$.*

Again, equivalently, we could define strongly chordal tree width via containment in the class of strongly chordal graphs.

Due to Farber and Lemma 3.29, we conclude the following corollary.

Corollary 3.4. *If RSA is secure, Extended Frege is non-fpt-automatable with regard to strongly chordal proof tree width.*

After we have seen result on non-fpt-automatability, we want to provide examples of parameters for which a proof system is fpt-automatable. Practically, this means that we provide an algorithm that constructs a proof for us and show that running the algorithm is fpt-tractable.

In the following, we look at a second proof system: resolution. Quite some notions are analogue to Frege systems. Nonetheless, we can see that here, that there is more variety in the type of representation that differs from the proof graphs for Frege systems. Furthermore, on the contrary to Extended Frege, resolution seems easier to automate.

4 Parameters for the proof system resolution

Resolution is a refutational proof system. Thus, we prove claims indirectly. Furthermore, resolution is known due to its close connection to the satisfiability problem (see Theorem 4.2.1 in [Kra95]).

4.1 Resolution and graphs corresponding to a refutation

We define the proof system called *resolution* similar to [Ima17]. Resolution operates on propositional conjunctive normal formulas and is therefore rather restricted.

Definition 4.1. *The following rule, is a resolution rule for clauses C, D*

$$\frac{C \vee x, D \vee \overline{x}}{C \vee D}$$

The clause $C \vee D$ is called resolvent*.*

A resolution refutation of a formula φ *is a sequence of clauses $C_1, C_2, \ldots, C_k$ such that*

- *for each i the clause C_i is a clause occurring in φ or in a resolvent of two previous clauses*
- *the last clause C_k is an empty clause*

Note that deducing a tautologies such as $x_i \vee \overline{x_i}$ is not a valid resolution proof since we think of resolution proofs as refutations and only deducing the empty set leads to a valid resolution refutation.

For resolution, there are different ways how to define the corresponding proof graph of a refutation. Recall, that for Frege proofs, the proof graph had an edge between two vertices marked with a formula if and only if only one formula was inferred by the other one via a Frege rule.

For resolution, we introduce two different corresponding graphs for a fixed refutation: the primal graph and the incidence graph.

Definition 4.2. *The* primal graph *of a refutation $C_1, \ldots, C_n$ has a vertex for each variable occurring in $C_1, \ldots, C_n$ and an edge between two vertices if and only if the variables are both contained in a clause.*

For both ways to construct a graph from a conjunctive normal form formula (CNF), we exemplify the graph for the same formula.

Example 4.3. *Take the formula $F = (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee x_3 \vee \overline{x_4}) \wedge (x_2 \vee x_3 \vee x_5)$. The primal graph of this formula is*

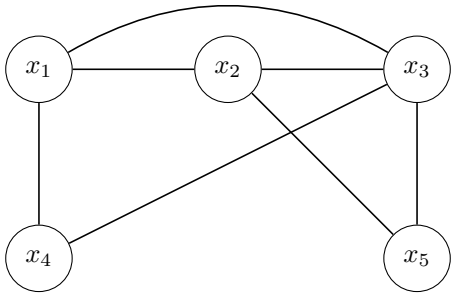


Figure 13: Primal graph of F

The second graph representation of a formula is the *incidence graph*. The incidence graph carries more information than the primal graph since there are different types of vertices in the incidence graph so that we can deduce which variables belong to which clauses. This is not always possible given a primal graph.

Definition 4.4. *The* incidence graph *of a refutation $C_1, \ldots, C_n$ has a vertex for each clause C_i in $C_1, \ldots, C_n$ and for each variable occurring in $C_1, \ldots, C_n$ and an edge between a clause and a variable if and only if the variable is contained in the clause.*

For illustration of an incidence graph, we take the same formula as from the example before and then, the incidence graph is as follows.

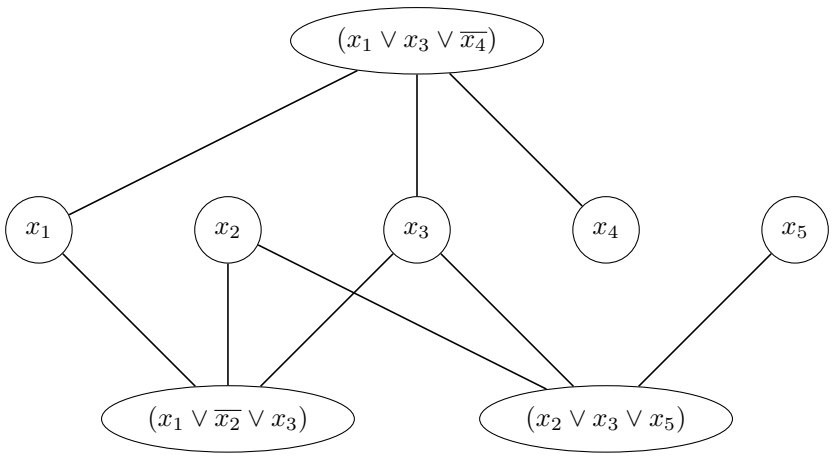


Figure 14: Incidence graph of F

As already stated, with the proof system resolution, we have more options on how to represent a proof as a graph. Before, we have considered graph parameters for arbitrary graphs but if we take resolution as a proof system, our parameter can depend on the choice of the graph representation.

As a result, we add the name of the graph representation to the parameter. In place of tree width, we are going to talk about *primal tree width* (in short *ptw*) or *incidence tree width* (in short *itw*).

4.2 Treelike resolution and primal tree width

We want to show that resolution is fpt-automatable with regard to primal tree width. We do so by splitting this into two proofs. First, we show that a weaker version of resolution, namely, treelike resolution is fpt-automatable with regard to primal tree width and provide some examples. Second, we show that resolution is also fpt-automatable with regard to primal tree width. Here, we needed a multiple of the time as in treelike resolution and examples explode in the number of clauses. Subsequently, we stay more high-level and count on the intuition gained by reading through the proof and the examples for treelike resolution.

But first, let us define treelike resolution.

Definition 4.5. *A* treelike *resolution refutation of a formula φ is a sequence of clauses $C_1, C_2, \ldots, C_k$ such that every clause that is inferred by a resolution rule can be used only at most once.*

This restricts the shape of resolution proofs and therefore, treelike resolution is a weaker proof system than resolution. In the following, consider an example of a refutation that is treelike and one that is not.

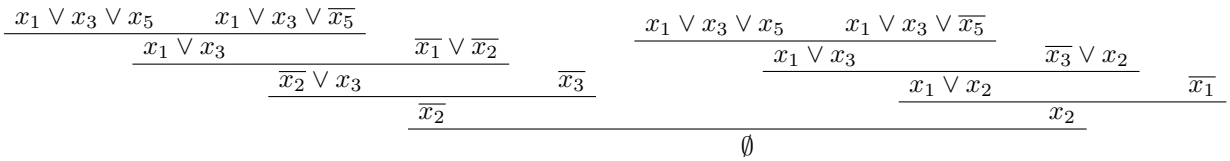
Example 4.6. *Take the formula $\varphi = (x_1 \vee x_3 \vee x_5) \wedge (x_1 \vee x_3 \vee \overline{x_5}) \wedge (\overline{x_2} \vee \overline{x_1}) \wedge (\overline{x_3} \vee x_4) \wedge (\overline{x_3}) \wedge (\overline{x_1})$ and the following refutation.*

$$\dfrac{\dfrac{\dfrac{x_1 \vee x_3 \vee x_5 \quad x_1 \vee x_3 \vee \overline{x_5}}{x_1 \vee x_3} \quad \overline{x_2} \vee \overline{x_1}}{\overline{x_2} \vee x_3} \quad \overline{x_3} \qquad \dfrac{\overline{x_3} \vee x_4}{x_1 \vee x_2} \quad \overline{x_1}}{\dfrac{\overline{x_2}}{\emptyset}} \qquad \dfrac{x_2}{\emptyset}$$

This refutation is not treelike since the inferred clause $x_1 \vee x_3$ is part of two different inferences.

But the following example is treelike.

Example 4.7. *Take the same formula $\varphi = (x_1 \vee x_3 \vee x_5) \wedge (x_1 \vee x_3 \vee \overline{x_5}) \wedge (\overline{x_2} \vee \overline{x_1}) \wedge (\overline{x_3} \vee x_4) \wedge (\overline{x_3}) \wedge (\overline{x_1})$ and the following refutation.*



Here, we needed to copy the subprooftree of $(x_1 \vee x_3)$ which occurs twice and conclude that generally, treelike resolution refutation tend to have more clauses and inferences.

For showing that treelike resolution is fpt-automatable with regard primal tree width, the idea is that we use the tree decomposition as a framework for a dynamic programming approach: Starting from the leaves from the tree decomposition, we iteratively build a resolution refutation that is complete when we arrive at the root of the tree decomposition.

The difficulty lays in defining how to modify a resolution proof from one bag to the other. We facilitate this difficulty by building our algorithm on a slightly modified tree decomposition, namely, *nice tree decompositions.*

Definition 4.8. *A* nice tree decomposition *is a tree decomposition such that for every bag X_i, we have one of the following:*

- *(leaf) i is a leaf in T and $|X_i| = 1$*
- *(introduce) $\exists v \in V.X_i = X_{i-1} \cup \{v\}$*
- *(forget) $\exists v \in V.X_i = X_{i-1} \setminus \{v\}$*
- *(join) i has two children i_1 and i_2 in T and $X_i = X_{i_1} = X_{i_2}$*

Note that nice tree decompositions are rooted since a bag/child cannot be obtained by two parents. Nice tree decompositions facilitate the algorithmic aspect and can be easily computed from a regular tree decomposition but might have more bags then a conventional tree decomposition. This has been proven by Kloks (see Lemma 13.1.3 in [Klo94]):

Theorem 4.9. *Given a tree decomposition of a graph G with n bags of width k, one can find a rooted nice tree decomposition of G of width k and with at most $4n$ bags in $\mathcal{O}(n)$ time.*

We are going to use this theorem as a subroutine for building a nice tree decomposition.

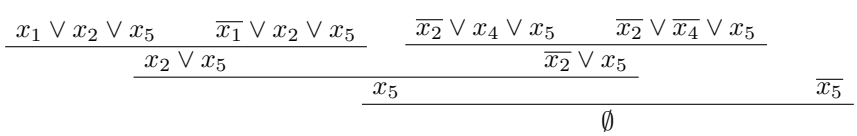
The tree decomposition is the framework for the algorithm that computes a refutation for a given unsatisfiable formula. Since we can have more than one refutation for a formula and we want to prove automatability, we look for the shortest refutation possible for a given formula. Therefore, we introduce a measure for the length of a refutation.

Definition 4.10. *The* resolution proof depth *of a formula φ is the number of resolution rules needed to deduce φ from a set of given formulas. If there is no resolution proof that refutes φ, the resolution proof depth is infinity.*

The following example should illustrate how to compute the resolution proof depth for a formula.

Example 4.11. *Take the following set of clauses $\{x_1 \vee x_2 \vee x_5, \overline{x_1} \vee x_2 \vee x_5, \overline{x_2} \vee x_4 \vee x_5, \overline{x_1} \vee \overline{x_4} \vee x_5, \overline{x_5}\}$.*

We can build the formula $\varphi = (x_1 \vee x_2 \vee x_5) \wedge (\overline{x_1} \vee x_2 \vee x_5) \wedge (\overline{x_2} \vee x_4 \vee x_5) \wedge (\overline{x_1} \vee \overline{x_4} \vee x_5) \wedge (\overline{x_5})$ from this set of clauses and find the following refutation for φ:



From the proof tree, we can deduce that the clauses $x_1 \vee x_2 \vee x_5, \overline{x_1} \vee x_2 \vee x_5, \overline{x_2} \vee x_4 \vee x_5, \overline{x_1} \vee \overline{x_4} \vee x_5$ and $\overline{x_5}$ have depth zero since they are part of the clause set. The clauses $x_2 \vee x_5$ and $\overline{x_2} \vee x_5$ have depth one, the clause x_5 has depth three and the empty set has depth four.

We see that for a treelike refutation, we obtain the depth of a child if we add the depth of all parents and add one. Thus, we can compute the depth recursively.

We note that this is not true for refutations that are not treelike. There, we could obtain a smaller resolution proof depth if a clause is part of several inference rules.

The resolution proof depth will be our measure to find the shortest refutation for a formula. We do so, by finding the shortest refutation in fpt-time with regard to the parameter primal tree width. Recall, that a proof system is automatable if we can find a proof in time polynomial in the shortest proof and the shortest proof with regard to the resolution proof depth is shorter than the shortest proof with regard to the primal tree width. This is captured in Lemma 2.15 but we adapt it in the following since due to the proof system resolution, we take the shortest proof with regard to resolution proof depth and not the number of symbols in a proof.

Lemma 4.12. *The polynomial of the resolution proof depth of the proof of φ with the smallest primal tree width has at least a polynomial of the resolution proof depth of the proof of φ with the least resolution proof depth. Furthermore, we have*

$$p(n_{\hat{\pi}}) \leq \min_{\pi} f(k)q(n_{\hat{\pi}})$$

for a function f, polynomial p and q and the proof $\hat{\pi}$ with the shortest resolution proof depth $n_{\hat{\pi}}$.

Proof. Take a proof with primal tree width k. Therefore, there must be a tree decomposition where the biggest bag has size $k+1$. We follow that the $k+1$ variables form a clique in the primal graph. By definition of the primal graph, we have for each two variables in the clique there is a clause that contains both variables and for each clique K in the graph, we can form the respective set of clauses C_K. All these sets C_k together form the formula, otherwise there would be an edge missing in the primal graph.

We see that the proof with the shortest resolution proof depth has a smaller/equal resolution proof depth than the addition of all shortest resolution proof depth of C_K over all K since the proof of the whole φ can make use of common proof parts.

We see that the shortest resolution proof depth of a C_K is at most $|C_K| \leq k+1$. Subsequently, $\mathcal{O}(\sum_K |C_K|) = \mathcal{O}(k+1)$ and with that we can upper bound the shortest refutation proof depth.

We follow that the polynomial of the resolution proof depth of the proof of φ with the smallest primal tree width has at least a polynomial of the resolution proof depth of the proof of φ with the least resolution proof depth.

The second part

$$p(n_{\hat{\pi}}) \leq \min_{\pi} f(k)q(n_{\hat{\pi}})$$

follows by the nature of the minimum since the right term minimizes over π and q is a polynomial. This is similar to Lemma 2.15. □

Thus, if we can show that we can find the shortest refutation in fpt-time, we have shown a weaker property than automatability and a stronger property than fpt-automatability. Subsequently, we can conclude that fpt-automatability with regard to primal tree width. Note that this result does not depend on the resolution proof being treelike.

The following algorithm aims to find the shortest refutation in fpt-time with regard to primal tree width, given the nice tree decomposition $\langle \{X_i\}_i \mid T \rangle$ of the primal graph of a formula φ.

Algorithm 1 Computing a refutation with parameter primal tree width

Require: Nice tree decomposition $\langle \{X_i\}_i \mid T \rangle$ of the primal graph of a formula φ
Ensure: Refutation of φ

$B \leftarrow$ all bags enumerated from leaves to root, branches from left to right
for $b \in B$ **do**
    Make a three column table $T^b = (T^b_{j,1}, T^b_{j,2}, R^b_{j,3})_j$
    Write all possible clauses c_j over the variables in b in the left column, row j in $T^b_{j,1}$
    if b is a leaf **then**
        for $c_j \in T^b$ **do**
            if $c \in \varphi$ **then**
                $T^b_{j,2} \leftarrow 0$
            else
                $T^b_{j,2} \leftarrow \infty$
            end if
        end for
    else if b is a forget bag and h is the child of b **then**
        $T^b \leftarrow$ `copyChildTable(h,b)`
        if b is the root **then**
            return refutation $R^b_{j,2}$
        end if
    else if b is a introduce bag and h is the child of b **then**
        $T^b \leftarrow$ `copyChildTable(h,b)`
        $T^b \leftarrow T^b$ append `addMissingClauses(b)`
        $T^b \leftarrow$ `checkForShorterRefutations(`T^b`)`
        $T^b \leftarrow$ `searchForNewClauses(`T^b`)`
    else if b is a join bag and h and k are children of b **then**
        $T^b \leftarrow$ `mergeAndCopyChildrenTables(h,k,b)`
        $T^b \leftarrow$ `checkForShorterRefutations(`T^b`)`
        $T^b \leftarrow T^b$ `searchForNewClauses(`T^b`)`
    end if
end for

In words, the algorithm computes a matrix T^b for every bag $b \in B$. The ordering ensures that the matrix of the children of a bag is computed first which is possible since the tree decomposition forms a tree.

For each T^b, we keep track of the refutation that is build iteratively. If one clause is inferred by two other clauses, we save the two clauses in the third column R^b in row j for the resolvent c_j. At the end of the algorithm, we can trace back the refutation by jumping from the root to clause to clause until the clause is contained in φ.

For simplicity, we denote with R^b_j the whole refutation that is associated with clause c_j for bag b and not only the the last two clauses c_j is inferred by.

In sequence, we fill the table T^b depending on the type of the bag and the values in the the matrix of the child of the bag. Due to the properties of a nice tree decomposition this case distinction is well-defined: Either a bag has no children (leaf), exactly one child (forget bag and introduction bag) or exactly two children (join bag).

If the bag b is a leaf bag, we initialize the matrix T^b with possible clauses over the variable set b and zero if the clause occurs in φ and ∞ if the clause does not occur in φ.

If the bag b is a forget bag, we call the method `copyChildTable(h,b)`. This method copies all clauses in T^h that have a finite resolution proof depth over the variables b to the table T^b. Since b is a forget bag, we have $b \subseteq h$, therefore, we do not lose information over the clauses. Furthermore, we can ignore clauses with an infinite resolution proof depth because we aim to find a refutation at the end of the algorithm.

If the bag b is a introduce bag, we call again the method `copyChildTable(h,b)` that copies all clauses in T^h that have a finite resolution proof depth over the variables b to the table T^b. Since b is an introduce bag, we have that the bag b contains one more variable than h. The method `addMissingClauses(`T^b`)` adds all possible clauses that contain $x_i = b \setminus h$ to the first column of T^b and initialize the respective resolution proof depth with zero if the clause is contained in φ and else ∞.

But the added clauses are not necessarily associated with the minimal resolution proof depth. It could be that a shorter refutation is possible with the added clauses.

Therefore, we check for all pairs of clauses in T^b if shorter resolution proof can be obtained and overwrite the resolution proof depth if that's the case. Here, we make use from the fact that we assume that the associated resolution proof depth copied from the child is actually a minimal one for the child, so that it suffices to check if two clauses yield a shorter proof. Since the refutation is assumed to be treelike, we can compute the resolution proof depth with the formula $T^b_{k,2} + T^b_{l,2} + 1$. With that, we have assigned to all clauses in T^b the current minimal resolution proof depth.

In case of overwriting the resolution proof depth, we also keep track of a different resolution proof. Therefore, we write c_k and c_l into the cell R^b_j.

Algorithm 2 Helper method for Algorithm 1: `checkForShorterRefutations`

Require: The current table T^b
Ensure: Table T^b
for $c_k, c_l \in T^b$ **do**
  if $c_k \# c_l = c_j \in T^b_{j,1}$ and $T^b_{j,1} > T^b_{k,2} + T^b_{l,2} + 1$ **then** ▷ # denotes the resolution rule
    $T^b_{j,2} \leftarrow T^b_{k,2} + T^b_{l,2} + 1$
    Update R^b_j
  end if
end for
return T^b

In addition to all possible clauses over b, there could be clauses that can be inferred by the added clauses and the clauses from the child. As a result, we call the following method.

Algorithm 3 Helper method for Algorithm 1: `searchForNewClauses`

Require: The current table T^b
Ensure: Table T^b
Add all pairs clause pairs c_k, c_l in T^b to the set W^b
while $c_k, c_l \in W^b$ **do**
  if $c_k \# c_l \notin T^b_{j,1}$ **then**
    $c_w := c_k \# c_l$
    Add c_w to T^b
    $T^b_{w,2} \leftarrow T^b_{k,2} + T^b_{l,2} + 1$
    Update R^b_j
    Add all clause pairs c_k, c_w to W^b
    $W^b \leftarrow W^b \setminus \{c_k, c_l\}$
  end if
end while
return T^b

The method `searchForNewClauses` checks whether from the clauses in T^b new clauses can be obtained by using the resolution rule for any pair of clauses in T^b. If the clause is new, we can update R^b_j and add the resolution proof depth with the formula $T^b_{k,2} + T^b_{l,2} + 1$ since the refutation is assumed to be treelike. The new clause could be object to a new derivation, therefore, we check for pairs containing the new clause if another new clause can be generated etc. We have that the while loop terminates since the set of variables that were added that contained the variable x_i has a fixed size and therefore, only a finitely many new clauses can be found (see Lemma 4.15).

If the bag b is a join bag, we call the method `mergeAndCopyChildrenTables(h,k,b)` which is similar to the method `copyChildTable(h,b)`. The method `mergeAndCopyChildrenTables(h,k,b)` copies clauses and the respective resolution proof depth from both children h and k. Whenever there is a clause that occurs both in h and k, we copy the clause with the lower resolution proof depth.

Since the children can contain different clauses, we need to call the method `checkForShorterRefutations(`T^b`)` to check if new combinations yield a shorter refutation. When we have ensured that all clauses have their shortest resolution proof depth, we call the method `searchForNewClauses(`T^b`)` and check if we can obtain new clauses.

We see that the algorithm only stops when we reach the root that is labeled with the empty set. Since it is labeled with the empty set, it is necessarily a forget bag and due to the structure of a nice tree

decomposition, we have that the whole algorithm terminates after considering all the bags below the root.

We start with an easy example to illustrate how the algorithm works.

Example 4.13. *Consider the formula $\varphi = (x_1 \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_1})$. This formula is unsatisfiable since with clause $(\overline{x_1})$ we need to set x_1 to* `false` *but then either clause $(x_1 \vee x_2)$ or $(x_1 \vee \overline{x_2})$ will be unsatisfied.*

We can find the following refutation of φ and note that this is the shortest refutation of φ.

$$\dfrac{\dfrac{x_1 \vee x_2 \qquad x_1 \vee \overline{x_2}}{x_1} \qquad \overline{x_1}}{\emptyset}$$

The goal of the algorithm is to find this refutation (or an other refutation of the same size) and it does so, by iteratively compose it through refutations of subformulas. The framework for the iterative build-up is the nice tree decomposition of φ.

We have that the primal graph G of φ is the following:

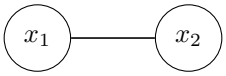


Figure 15: The primal graph G of φ

And from that, we can make a nice tree decomposition $\langle \{X_i\}_i \mid T \rangle$ with $X_4 = \emptyset, X_3 = \{x_1\}, X_2 = \{x_1, x_2\}$ and $X_1 = \{x_1\}$.

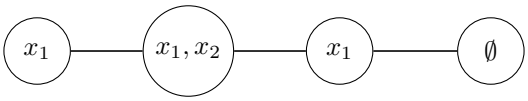


Figure 16: A nice tree decomposition of G

Iteratively, we fill in the following table, starting with X_1. For each X_i, we list all possible clauses over the variables X_i and write them in the table. Some clauses occur in φ, those have resolution proof depth zero and else ∞. Then, we try to deduce the other clauses in the list of X_i via resolution. We can deduce them from clauses from the list of X_i that are already deduced or contained in φ.

In the algorithm, we have an additional column that saves the current resolution proof for each clause. At the end of the algorithm, we trace back the refutation that is associated with the root. For simplicity, in the examples, we only mark the steps that are relevant to the outcome of the refutation of the root and do mark anything when the column is that short so that it clear. When a resolution rule is part of the final refutation, we mark two clause and the resolvent of them with either a circle or a square and additionally, mark the resolvent in bold.

X_1		X_2		X_3		X_4	
$\emptyset$	∞	$\emptyset$	∞	$\emptyset$	∞	$\emptyset$	2
(x_1)	∞	(x_1)	⓵	(x_1)	1		
$(\overline{x_1})$	0	$(\overline{x_1})$	0	$(\overline{x_1})$	0		
		(x_2)	1				
		$(\overline{x_2})$	1				
		$(x_1 \vee x_2)$	⓪				
		$(\overline{x_1} \vee x_2)$	0				
		$(x_1 \vee \overline{x_2})$	⓪				
		$(\overline{x_1} \vee \overline{x_2})$	∞				

Table 1: Table to compute minimal size resolution proof

Both the clauses $(x_1 \vee \overline{x_2})$ and $(x_1 \vee x_2)$ occur in the formula φ and are therefore equipped with a zero. If we apply the resolution rule to the clauses, we obtain (x_1) and together with the clause that occurs in the formula $(\overline{x_1})$, we can deduce with a second application of the resolution rule the empty set which yields the refutation, we wanted to construct.

Here, we see that different choices of tree decomposition lead to different refutations. In other words, the output of the algorithm depends on the choice of tree decomposition. As a result, we want to show that the algorithm is well-defined and outputs the shortest refutation independent of the choice of tree decomposition.

But since in Example 4.13, we haven't used the branch in the algorithm for join bags, let us first go through a second, more complicated example.

Example 4.14. *Consider the formula* $\psi = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee x_3 \vee x_4) \wedge (\overline{x_1}) \wedge (\overline{x_4})$. *This formula is unsatisfiable since with clause* $(\overline{x_1})$ *and* $(\overline{x_4})$ *we need to set* x_1 *and* x_4 *to* `false` *but then with clause* $(x_1 \vee x_3 \vee x_4)$, *we need to set* x_3 *to* `true` *and then, either clause* $(x_1 \vee x_2 \vee \overline{x_3})$ *or* $(x_1 \vee \overline{x_2} \vee \overline{x_3})$ *will be unsatisfied.*

We have that the primal graph G *of* ψ *is the following:*

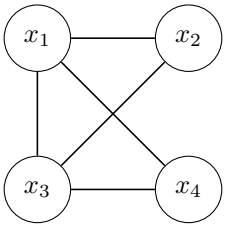


Figure 17: The primal graph G of ψ

Since G *contains a 3-clique, we have that the tree width of the graph is at least two. In this case, it is exactly two.*

We use the algorithm in [Klo94], to first build a normal tree decomposition and from that, a nice tree decomposition.

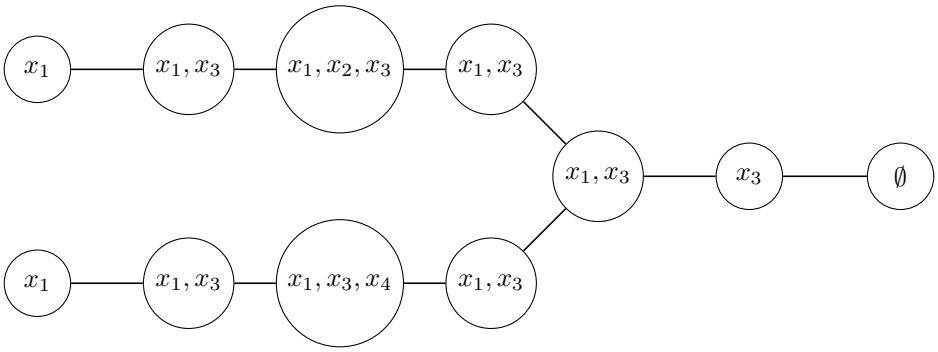


Figure 18: A nice tree decomposition of G

For convenience, we enumerate the bags starting from the upper left to the bag containing x_1, x_3, *then starting from the lower left to the lower bag containing* x_1, x_3 *and then, starting from the join bag containing* x_1, x_3 *until the root. We aim to simplify the table and split it into one table for the upper branch, one for the lower branch and one for joining the upper and the lower branch. As it was done in the previous example, we do not show the third column that saves the current resolution proof and just illustrate the output refutation by circles and squares wrapped around the resolution proof depth of the respective clauses. This yields the following tables.*

x_1		x_1, x_3		x_1, x_2, x_3		x_1, x_3	
$(\overline{x_1})$	0	$(\overline{x_1})$	0	$(\overline{x_1})$	0	$(\overline{x_1})$	⓪
				$(x_1 \vee x_2 \vee \overline{x_3})$	⓪	$(x_1 \vee \overline{x_3})$	①
				$(x_1 \vee \overline{x_2} \vee \overline{x_3})$	⓪	$(\overline{x_3})$	②
				$(x_1 \vee \overline{x_3})$	①		
				$(x_2 \vee \overline{x_3})$	1		
				$(\overline{x_2} \vee \overline{x_3})$	1		
				$(\overline{x_3})$	2		

Table 2: Table for the upper branch

We omit the clauses that have an infinite resolution proof depth because they do not contribute to the final resolution proof output.

x_1		x_1, x_3		x_1, x_3, x_4		x_1, x_3	
$(\overline{x_1})$	0	$(\overline{x_1})$	0	$(\overline{x_1})$	0	$(\overline{x_1})$	⓪
				$(\overline{x_4})$	⓪	$(x_1 \vee x_3)$	①
				$(x_1 \vee x_3 \vee x_4)$	⓪	(x_3)	②
				$(x_1 \vee x_3)$	①		
				$(x_3 \vee x_4)$	1		
				(x_3)	2		

Table 3: Table for the lower branch

The table that is the continuation of the table for the upper branch and the lower branch merges the two branches. Even though the join bag contains the same variables as the last branch of the lower and the upper branch, we obtain resolution proofs for more clauses.

x_1, x_3		x_3		$\emptyset$	
(x_1)	3	(x_3)	2	$\emptyset$	5
$(\overline{x_1})$	2	$(\overline{x_3})$	2		
(x_3)	2				
$(\overline{x_3})$	2				
$(x_1 \vee \overline{x_3})$	1				
$(x_1 \vee x_3)$	1				

Table 4: Table that joins the upper and the lower branch

The algorithm ends by filling in a finite number to empty clause. In this case, we obtain the number five, thus, we have constructed a resolution proof of ψ of size five.

We can deduce the proof by tracking down the sub proofs that led to the number five.

First, in the table of the upper branch in column six, row four and seven, we deduced the clauses $(x_1 \vee \overline{x_3})$ and then the clause $(\overline{x_3})$. Similarly, in the table of the lower branch, we deduced $(x_1 \vee x_3)$ and (x_3) in column six and eight. Together, we complete the refutation in Table 4 and obtain the empty clause.

The proof tree for the constructed refutation of ψ looks like in the following:

$$\dfrac{\dfrac{\dfrac{\overline{x_4} \qquad x_1 \vee x_3 \vee x_4}{x_1 \vee x_3} \qquad \overline{x_1}}{x_3} \qquad \dfrac{\overline{x_1} \qquad \dfrac{x_1 \vee x_2 \vee \overline{x_3} \qquad x_1 \vee \overline{x_2} \vee \overline{x_3}}{x_1 \vee \overline{x_3}}}{\overline{x_3}}}{\emptyset}$$

We see that this is the shortest refutation by trying out all other combinations.

With the two examples in mind, we want to prove the correctness of the algorithm.

Lemma 4.15. *Algorithm 1 outputs the shortest refutation for an unsatisfiable formula independent of the choice of the tree decomposition.*

Proof. Since the tree decomposition is rooted, we can identify each bag with a level. The root has the level of the longest path from root to leaf in the tree. The level of any other bag is the longest path from root to leaf in the tree minus the length of the shortest path from the respective bag to the root.

We want to prove by induction on the levels that Algorithm 1 outputs the shortest refutation for an unsatisfiable formula.

Base case: Take any bag b on level 0. Then, the bag b is a leaf. Therefore, we have $|b| = 1$ and the algorithm trivially outputs the shortest refutation if it exists.

Induction hypothesis: For all bags of level n every clause is equipped with the correct resolution depth and the corresponding resolution proof and the resolution proof of a clause in a bag h assumes only clauses in T^h.

Induction step: Take any bag b of level $n + 1$. We want to show that all clauses in T^b are equipped with the correct resolution depth and the corresponding resolution proof that assumes only clauses in T^b.

Let us make a case distinction on the type of bag of b. We see that b can be a leaf since the tree decomposition does not need to have all leaves on the same level. If b is a leaf, the assertion follows trivially.

Case 1: If b is a forget bag and h is the child of b, all clauses over b are clauses over h since $b \subset h$. Since we copy T^h to T^b, the assertion follows with the induction hypothesis.

Case 2: If b is an introduce/join bag and h is the child of b, we have that only the set of clauses whose resolution proof depth number has been updated might not have a correct resolution proof depth due to the induction hypothesis. But similarly, since we have a tree decomposition, thus, for all $i, j, l \in I$, if j lies on the path between i and l in T then $X_i \cap X_l \subseteq X_j$, we exhaust all possibilities of a shorter resolution proof in the algorithm.

In sum, it follows that Algorithm 1 outputs the shortest refutation for an unsatisfiable formula.

For the second part, we have that Algorithm 1 outputs the shortest refutation independent of the choice of the tree decomposition by definition of a nice tree decomposition: Each clause induces three edges e in the primal graph. The variables of e need to be contained in at least a bag and when we reach the root all edges and therefore, all clauses in φ were covered. Since we update for each bag if there are shorter proof the order which the clauses appear in the tree decomposition does not matter. As a result, the choice of tree decomposition does not matter. □

For showing fpt-automatability the running time is important. We want to show that we can split it into a part that is only dependent on the parameter primal tree width and a part that is only dependent on the input length.

Lemma 4.16. *Algorithm 1 has running time $f(k)p(n)$ for a function f, polynomial p, parameter $k = ptw(\pi)$ and input length n for an unsatisfiable formula φ and refutation π of φ with the least resolution proof depth.*

Proof. Since the tree decomposition is part of the input, we have that the number of bags is smaller or equal to n. We fix a bag b and want to show that the algorithm runs in time $f(k)$ whereas f is not dependent on n.

For a leaf, initializing the resolution proof depth value of each clause over b takes linear time in the number of rows of T^b. Since each bag has size at most $k+1$, there are at most 2^{k+1} many clauses in a table T^b. With that, we can upper bound the number of rows in T^b by 2^{k+1}.

For a forget bag that is not the root, we have also a polynomial running time in 2^{k+1}.

Since both introduce bag and join bag call similar methods, we focus on the running time of the join bag. The method `mergeAndCopyChildrenTables` takes only polynomial time in the size of the tables T^h and T^k and both T^h and T^k are bounded by 2^{k+1}.

The method `checkForShorterRefutations` runs over all subsets of size two of the clauses contained in T^b. We upper bound this by $\binom{|T_1^b|}{2} \leq |T_1^b|^2 \leq 2^{k+2}$. The checks in line 2 to 4 of `checkForShorterRefutations` can be done in constant time, which results in a total running time of `checkForShorterRefutations` of $\mathcal{O}(2^{k+2})$.

For the running time of the method `searchForNewClauses`, we see that the set W^b can contain at most $\binom{|T_1^b|}{2} \leq |T_1^b|^2 \leq 2^{k+2}$ since the bag size of b is bounded by $k+1$. The worst case in the following while loop is if every pair in W^b yields a new clause. But then, we need to add $2^{k+2} + 2^{k+2} = 2^{k+3}$ since there are at most 2^{k+2} such pairs.

For the join bag, we end up with a total running time of $\mathcal{O}(2^{k+3})$.

We note that the trace back of line 16, takes at most as long as the longest path from root to a leaf times the number of resolution rules used for the refutation within the table for a fixed bag. We can upper bound this by $n \cdot \max_b |T^b| = n2^{k+1}$. We do this trace back only once at the very end of the algorithm and therefore, we have a total running time of the algorithm of $n\mathcal{O}(2^{k+3}) + n\mathcal{O}(2^{k+1}) = n\mathcal{O}(2^{k+1})$. □

With that, the theorem on the fpt-automatability follows easily.

Theorem 4.17. *Treelike resolution is fpt-automatable with regard to primal tree width.*

Proof. Take an unsatisfiable conjunctive normal form formula φ, then, there is a resolution proof refuting φ (see Theorem 4.2.1 in [Kra95]).

Compute the primal graph for this formula and the corresponding nice tree decomposition. This can be done in linear time [Klo94].

Due to Lemma 4.15 and Lemma 4.16, we know that the algorithm outputs the shortest refutation in fpt time of the shortest refutation with regard to primal tree width. With Lemma 4.12, we follow fpt-automatability with regard to primal tree width. □

4.3 Resolution and primal tree width

In the following, we want to generalize the result on fpt-automatability with regard to primal tree width from treelike resolution to resolution. As we have seen, resolution is the stronger proof system, therefore, we need a variation of the algorithm that computes a refutation also in the case of a refutation where an inferred clause is used in several rules at the same time.

We adapt Algorithm 1, so that it can also output refutations that are not treelike.

Algorithm 4 Computing a potentially not treelike refutation with parameter primal tree width

Require: Nice tree decomposition $\langle \{X_i\}_i \mid T \rangle$ of the primal graph of a formula φ
Ensure: Refutation of φ

$B \leftarrow$ all bags enumerated from leaves to root, branches from left to right
for $b \in B$ **do**
  Make a two column table $S^b = (S^b_{i,1}, R^b_{i,2})_i$
  Write possible subsets $S_{i,1}$ of possible clauses over b in S^b_1
  Initialize $R^b_{i,2}$ with ∞
  We are going to write for each $S_{i,1}$ the shortest resolution proof proving all elements $S_{i,1}$
  if b is a leaf **then**
    for $S \in S^b_1$ **do**
      if $\forall c \in S : c \in \varphi$ **then**
        $R^b_{i,2} \leftarrow \langle c \mid c \in S \rangle$
      else
        $R^b_{i,2} \leftarrow \infty$
      end if
    end for
  else if b is a forget bag and h is the child of b **then**
    $S^b \leftarrow$ `copyChildSubsetTable(h,b)`
    if b is the root **then**
      return refutation $R^b_{i,2}$
    end if
  else if b is a introduce bag and h is the child of b **then**
    $S^b \leftarrow$ `copyChildSubsetTable(h,b)`
    $S^b \leftarrow$ `addMissingSubsetClauses(b)`
    $S^b \leftarrow$ `checkForShorterSubsetRefutations(`S^b`)`
  else if b is a join bag and h and k are children of b **then**
    $S^b \leftarrow$ `mergeAndCopyChildrenSubsetTables(h,k,b)`
    $S^b \leftarrow$ `checkForShorterSubsetRefutations(`S^b`)`
  end if
end for

The above algorithm computes a matrix S^b for every bag $b \in B$ and for each S^b, we keep track of the refutation that is build iteratively. If one clause is inferred by two other clauses, we save them as children of the resolvent in a tree in the second R^b. At the end of the algorithm, we output the refutation saved in $R^b_{i,2}$ for the root bag. In sequence, we fill the table S^b depending on the type of the bag and the values in the the matrix of the child of the bag. With S_1 we denote the first column of S^b.

If the bag b is a leaf bag, we initialize the matrix S^b with the set of clauses if all the clause occur in φ and ∞ if at least one clause does not occur in φ.

If the bag b is a forget bag, we call the method `copyChildSubsetTable(h,b)`. This method copies the content of S^h to the table S^b. Since b is a forget bag, we have $b \subseteq h$, therefore, we do not lose information over the clauses.

If the bag b is a introduce bag, we call again the method `copyChildSubsetTable(h,b)`. Since b is an introduce bag, we have that the bag b contains one more variable than h.

The method `addMissingSubsetClauses(`S^b`)` adds all possible subsets over all possible clauses that contain $x_i = b \setminus h$ and initializes them like the leaf bags. Subsequently, there could be shorter proofs that make use of a subset whose clauses are contained in φ. Therefore, we call `checkForShorterSubsetRefutations(`S^b`)` to update the current resolution proofs with help of the already existing proofs.

Algorithm 5 Helper method for Algorithm 4: `checkForShorterSubsetRefutations`

Require: The current table S^b
Ensure: Table S^b
for $(S^b_{i,1}, R^h_{i,2}), (S^b_{j,1}, R^b_{j,2}) \in S^b$ **do**
  for clause $c \in S^b_{i,1}$ **do** ▷ $R^b_{j,2}|_c$ denotes the subgraph of $R^b_{j,2}$ rooted in c
    if $R^b_{j,2}|_c \neq \emptyset$ and $R^b_{j,2}|_c$ has a lower resolution proof depth than $R^b_{i,2}|_c$ **then**
      ▷ $R^b_{i,2} \leftarrow^c R^b_{j,2}$ denotes the refutation where $R^b_{i,2}|_c$ is replaced by $R^b_{j,2}|_c$
      $S^b_i \leftarrow (S^b_{i,1}, R^b_{i,2} \leftarrow^c R^b_{j,2})$
    end if
  end for
end for
return S^b

Compared to the algorithm for treelike resolution, the method `checkForShorterSubsetRefutations(`S^b`)` combines `checkForShorterRefutations(`T^b`)` and `searchForNewClauses(`T^b`)`. This due to the clause subsets. Since we already consider subsets, we cannot miss potential clauses over the variables in b.

If the bag b is a join bag, we call the following method to pick the best resolution proof for each subset between from the pool of resolution proof from the children.

Algorithm 6 Helper method for Algorithm 1: `mergeAndCopyChildrenSubsetTables`

Require: The current table S^h and S^k
Ensure: Table S^b
for $(S^h_1, R^h_2) \in S^h$ and $(S^k_1, R^{S_k}_2) \in S^k$ **do**
  for $i \in \{1, \ldots, |S^h_1|\}$ **do**
    for $j \in \{1, \ldots, |S^k_1|\}$ **do**
      if $S^h_{i,1} = S^k_{j,1}$ and $R^h_{i,2}$ has a lower or equal resolution proof depth than $R^k_{i,2}$ **then**
        $S^b_{i,1} \leftarrow (S^h_{i,1}, R^h_{i,2})$
      else if $S^h_{i,1} = S^k_{j,1}$ and $R^k_{i,2}$ has a lower resolution proof depth than $R^h_{i,2}$ **then**
        $S^b_{i,1} \leftarrow (S^k_{i,1}, R^k_{i,2})$
      end if
    end for
    $S^b_{i,1} \leftarrow (S^h_{i,1}, R^h_{i,2})$
  end for
end for
return S^b

After we collected the pairwise best proof, we check for a shorter refutation for pairs within S^b with a call of the method `checkForShorterSubsetRefutations`. This has not been checked before and the new mix of clause sets from both children can yield shorter refutations

We see that the algorithm only stops when we reach the root that is labeled with the empty set. Since it is labeled with the empty set, it is necessarily a forget bag and due to the structure of a nice tree decomposition, we have that the whole algorithm terminates after considering all the bags below the root.

Since even for Example 4.13, we have that one column has size up to $2^8 = 256$ clause subsets, we do not provide examples. Since the algorithms are very similar, we see that the same induction proof as in Lemma 4.15 works to show correctness for resolution.

Lemma 4.18. *Algorithm 4 outputs the shortest (possibly not treelike) refutation for an unsatisfiable formula independent of the choice of the tree decomposition.*

Compared to treelike resolution, we have a higher running time: The size of the set of all clause subsets increases the computational resources needed to construct the shortest refutation.

Lemma 4.19. *Algorithm 1 has running time $f(k)p(n)$ for a function f, polynomial p, parameter $k = ptw(\pi)$ and input length n for an unsatisfiable formula φ and refutation π of φ with the least resolution proof depth.*

Proof. Since the tree decomposition is part of the input, we have that the number of bags is smaller or equal to n. We fix a bag b. Due to similar reasoning as in Lemma 4.16, we focus on the running time for introduce bags. We see that the running time of the method `checkForShorterRefutation` exceeds the running time for `copyChildSubsetTable` and `addMissingSubsetClauses` and therefore, we restrict our attention `checkForShorterRefutation`.

The first for loop iterates over all pairs of rows in S^b and since the size of S^b is bounded by the number of all subsets of the biggest bag, we can upper bound the running time of the for loop by $2^{2^{2k}}$.

The second for loop in line 2 iterates over all clauses in $S^b_{i,1}$ that are subset of variables in b and since there are at most 2^k such subsets, we upper bound the running time with 2^k.

For the if clause, we perform a depth first search through both current resolution proof graphs. With that, we can check if there is an entry at $R^b_{j,2}|_c$ and obtain the resolution proof depth via a counter, counting the edges above c. The running time of the depth first search can be upper bounded by the size of a table S^b and the length of the tree decomposition since the algorithm only potentially extends the current resolution proof per tuple $(S^b_{i,1}, R^h_{i,2})$. Subsequently, lines 3 to 5 take at most $2^{2^k} \cdot n$ time.

In sum, for a fixed b, we have running time $2^{2^{2k}} \cdot 2^k \cdot 2^{2^k} n$ and over all b, we have a total running time of $2^{2^{2k}} \cdot 2^k \cdot 2^{2^k} n^2$. □

With that, the theorem on the fpt-automatability follows easily.

Theorem 4.20. *Treelike resolution is fpt-automatable with regard to primal tree width.*

Proof. Compute the primal graph for this formula and the corresponding nice tree decomposition. This can be done in linear time [Klo94].

Due to Lemma 4.18 and Lemma 4.19, we know that the algorithm outputs the shortest refutation in fpt time of the shortest refutation with regard to primal tree width. With Lemma 4.12, we follow fpt-automatability with regard to primal tree width. □

Naturally, it poses the question if this is also true for the parameter incidence tree width.

4.4 Incidence tree width and one-sided tree width

We want show that resolution is fpt-automatable with regard to incidence tree width with help of fpt-dominance.

Lemma 4.21. *Incidence tree width fpt-dominates primal tree width. In particular we have for any graph G that $ptw(G) \geq itw(G) - 1$.*

Proof. We show this by constructing an incidence graph from a primal graph G and show that if G has primal tree width k, the translated incidence graph of G has incidence tree width at most $ptw(G)$ plus one.

Take a tree decomposition of a primal graph of φ. We see that the primal graph consists of cliques for the respective clauses of φ. For each clique C, we add a vertex labelled with C and connect it with all the vertices in the clique. Since in a primal graph there is an edge between two vertices if and only if the two vertices are both contained in a clause, we can conclude that the clause variables have an edge to vertices labelled with all variables contained in the clause.

As a result, we have constructed an incidence tree decomposition of width $ptw(G) + 1$. □

Proposition 4.22. *Resolution is fpt-automatable with regard to incidence tree width.*

Proof. Since incidence tree width fpt-dominates primal tree width (Lemma 4.21) and resolution is fpt-automatable with regard to primal tree width (Theorem 4.20), we conclude with Lemma 3.29 that resolution is fpt-automatable with regard to incidence tree width. □

Here, we see that the representation of the formula as a graph does not matter: If the depict the formula as primal graph or as incidence graph, in both cases resolution is fpt-automtable with regard to the respective parameters. Indeed, resolution is even fpt-automatable if we restrict the incidence graph with regard to different types of vertices: those that originate from clauses and those that originate from variables.

For this, let us introduce the parameter *width of one-sided tree-decomposition* as first defined in Definition 1 in [CR19].

Definition 4.23. *Let $\langle \{X_i\}_i \mid T \rangle$ be a rooted incidence tree decomposition of φ. For each clause C of φ, let V_C denote the set of bags in T whose bags contain C. Then $\langle \{X_i\}_i \mid T \rangle$ is* one-sided *if for each $C \in \varphi$ we have that $T[V_C]$ is a directed path. The* one-sided (incidence) treewidth *of φ is the smallest width of an one-sided incidence tree decomposition of φ.*

For illustration, let us take a look at the following example:

Example 4.24. *Take the following incidence graph from the formula from Example 4.3.*

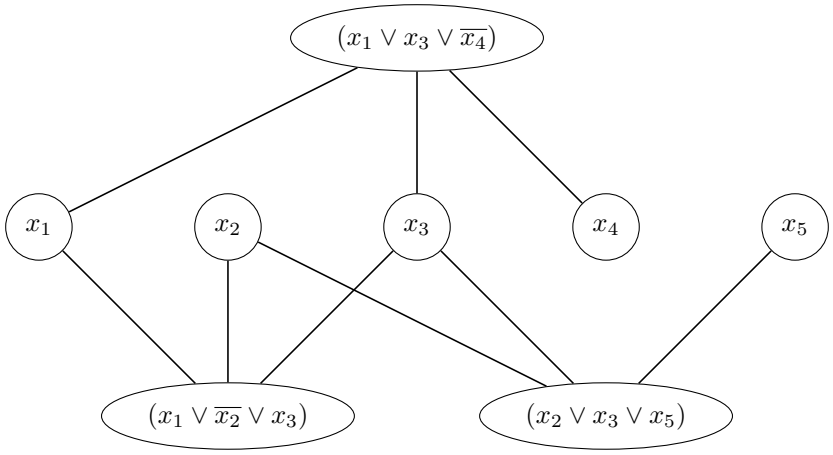


Figure 19: Incidence graph G

We want to build a rooted incidence tree decomposition based on this graph.

The following is a tree decomposition of G with $X_1 = \{x_4, (x_1 \vee x_3 \vee \overline{x_4})\}$, $X_2 = \{x_1, x_3, (x_1 \vee x_3 \vee \overline{x_4})\}$,$X_3 = \{x_1, x_3, (x_1 \vee \overline{x_2} \vee x_3)\}$, $X_4 = \{x_2, (x_1 \vee \overline{x_2} \vee x_3), (x_1 \vee x_3 \vee x_5)\}$, $X_5 = \{x_3, (x_1 \vee \overline{x_2} \vee x_3), (x_1 \vee x_3 \vee x_5)\}$ and $X_6 = \{x_5, \vee x_3), (x_1 \vee x_3 \vee x_5)\}$.

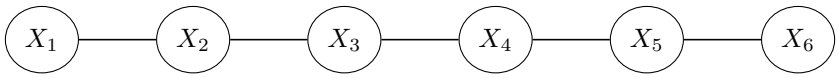


Figure 20: A one-sided tree decomposition of G

We see that the tree decomposition is rooted. We have the path for $C = (x_1 \vee x_3 \vee \overline{x_4})$, the path from X_1 to X_2 for $(x_1 \vee \overline{x_2} \vee x_3)$ and the path from X_3 to X_4 to X_5 for $(x_2 \vee x_3 \vee x_5)$ the path from X_4 to X_5 to X_6. Subsequently, we have a rooted one sided incidence tree decomposition.

The parameter one-sided tree width takes motivation also from the fact that it is a parameter that is not fpt-automatable but fpt-bounded. To show fpt-bounded, we cite the following theorem in [CR19]. Note that the introduction of the set of long clauses called **LONG** makes this theorem even stronger. If we choose the empty set for **LONG**, the result grants only the existence of a resolution with a higher size.

Theorem 4.25. *Let φ be a CNF, $\boldsymbol{LONG} \subseteq \varphi$ to which we refer as long clauses and $k > 0$ be an integer. Assume that $\varphi \setminus \boldsymbol{LONG}$ has a one-sided tree decomposition (T, B) of the incidence graph of path width at most k. Then there is a resolution of φ of size $(n + |\boldsymbol{LONG}|^{|\boldsymbol{LONG}|}) \cdot n \cdot 2^{\mathcal{O}(k^2 + k \cdot |\boldsymbol{LONG}|)}$.*

Since the function $(n+|\mathbf{LONG}|^{|\mathbf{LONG}|}) \cdot n \cdot 2^{\mathcal{O}(k^2+k\cdot|\mathbf{LONG}|)}$ can be written in the form $p(n) \cdot f(k)$ for a polynomial p, we can find for every tautology in for the proof system resolution, a fpt-bounded proof. Since a proof system that contains resolutions, simulates resolution, it follows that any proof system containing resolution is fpt-bounded with regard to the parameter one-sided tree width.

Corollary 4.1. *Resolution is fpt-bounded with regard to the parameter one-sided tree width.*

In this case, we have also fpt-automatability. Intuitively, this expresses that there are short, thus, potentially easy proofs and it's easy to find them.

We show fpt-automatability by fpt-dominance.

Lemma 4.26. *The parameter one-sided tree width is dominates by primal tree width.*

Proof. One-sidedness is a restriction on the shape of the graph whereas special tree width has no restriction on the shape of the graph. □

Similar to the fpt-dominance results we have seen before, we can deduce fpt-automatability. Since the result on special tree width relies on the assumption that RSA is secure, we need to carry over this assumption.

Corollary 4.2. *Resolution is fpt-automatable with regard to the parameter proof one-sided tree width.*

Proof. Since resolution is fpt-automatable with regard to one-sided tree width and one-sided tree width dominates incidence tree width, it follows with Lemma 3.29 that resolution is fpt-automatable with regard to the parameter one-sided tree. □

Similar to the hierarchy for Extended Frege, we consider the parameter path width.

4.5 Incidence path width

The parameter function path width is defined analogously to tree width. Here, we decompose a graph into a path decomposition. The tree decomposition was a modification of a graph G to turn G into a graph. Similarly, the path decomposition is a modification of a graph G that assembles/thickens paths of G so that the path decomposition forms a single path.

Definition 4.27. *Let $G = (V, E)$ be a graph. A* path decomposition *of G is a pair $\langle \{X_i \mid i \in I\}, P\rangle$ where each X_i is a subset of V , called a* bag*, and P is a path with the elements of I as nodes. The following three properties must hold:*

1. *for every edge $\{u, v\} \in E$, there is an $i \in I$ such that $\{u, v\} \subseteq X_i$*
2. *for all $i, j, l \in I$, if j lies on the path between i and l in P then $X_i \cap X_l \subseteq X_j$.*

The width *of $\langle \{X_i \mid i \in I\}, P\rangle$ is defined as $\max\{|X_i| \mid i \in I\} - 1$. The* path width *of G is the minimum k such that G has a path decomposition of width k.*

Consider the following example.

Example 4.28. *Take the graph $G = (V, E)$ depicted as follows.*

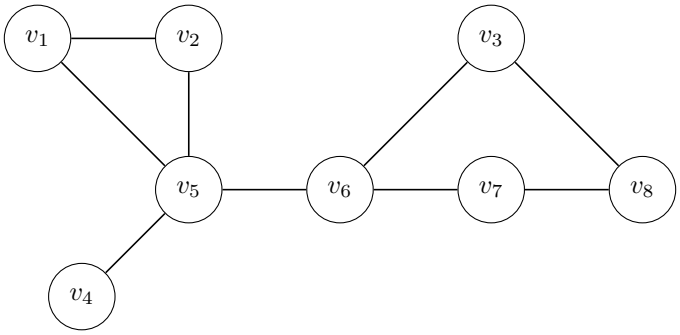


Figure 21: Graph G

The following figure shows a tree composition of graph G. Formally, the path decomposition can be described as $\langle \{v_1, v_2, v_4, v_5\}, \{v_4, v_5, v_6\}, \{v_3, v_6, v_7\}, \{v_3, v_7, v_8\} P\rangle$ and P consists of the four vertices that form a path . The structure $\langle \{X_i \mid i \in \{1, 2, 3, 4\}, P\rangle$ fits the definition of a path composition since we see that P forms a path since each edge is contained in at least in of the bags and we have that $X_1 \cap X_3 = \emptyset \subseteq X_2 = \{v_5, v_6\}$ and $X_2 \cap X_4 = \emptyset \subseteq X_3 = \{v_3, v_6, v_7\}$.

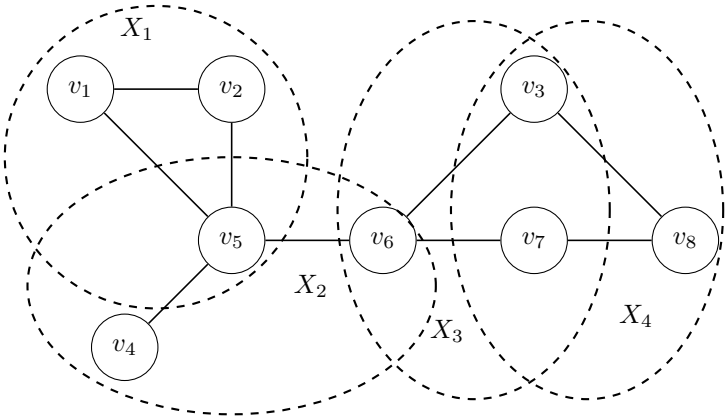


Figure 22: A path decomposition of G

The width of this tree composition is 2 *because the biggest bag* X_1 *contains* 3 *elements. We see that this is the smallest width for a path decomposition since we cannot make the bag* $\{v_1, v_2, v_5\}$ *smaller without violating the property that the path decomposition should be a path. Furthermore, there is only one path* P *that we can associate with the choice of bags due to the second property of the definition. The path* P *starts in* X_1 *and ends in* X_4*.*

The parameter incidence path width is a version of the path width for Extended Frege on incidence graphs. Consider the following Theorem 2 in [Ima17]:

Theorem 4.29. *There exists an algorithm that takes a CNF formula* φ *and the path decomposition* (T, X) *with one of the following properties for each edge vertex* X_i

- *(introduce)* $\exists v \in V.X_i = X_{i-1} \cup \{v\}$
- *(forget)* $\exists v \in V.X_i = X_{i-1} \setminus \{v\}$

of its incidence graph with path width $pw(\varphi)$ *as inputs, runs in* $2^{\mathcal{O}(pw(\varphi))}|\varphi|^{\mathcal{O}(1)}$ *time, and*

- *produces a satisfying assignment of* φ *if* φ *is satisfiable.*
- *constructs a resolution refutation of size* $\mathcal{O}(3^{pw(\varphi)})$ *if* φ *is unsatisfiable.*

Here, we have hidden the definition of a nice path decomposition within the theorem. We note that the property of a *nice* path decomposition is captured with the *introduce* and *forget* bags. For a nice tree decomposition as defined in Definition 4.8, we have the additional *join* bag which naturally is not part of a path decomposition.

We conclude that the proof system resolution produces proofs that are bounded in size with regard to the parameter path width.

Corollary 4.3. *Resolution is fpt-bounded with regard to the parameter (incidence) path width.*

Although fpt-boundedness does not necessarily imply fpt-automatability, in this case we have both. Since we can find such a proof in time $2^{\mathcal{O}(pw(\varphi))}|\varphi|^{\mathcal{O}(1)}$, we have:

Corollary 4.4. *Resolution is fpt-automatable with regard to the parameter (incidence) path width.*

Note that here, in comparison to tree width, we do not minimize over all φ, the running time $2^{\mathcal{O}(pw(\varphi))}|\varphi|^{\mathcal{O}(1)}$ depends on the parameter value $pw(\varphi)$. Similar to primal tree width and incidence tree width, we decided to call the parameter path width instead of proof path width. Nonetheless, one could think of a parameter proof path width that is defined analogously to proof tree width and that minimized over all φ. Since the fpt-automatability result holds for all choices of φ, in particular, it is true for the minimum and therefore, we have that resolution is fpt-automatable with regard to proof path width.

The attentive reader might have noticed that there is a second way how we could show fpt-automatability.

Corollary 4.5. *Resolution is fpt-automatable with regard to the parameter (incidence) path width.*

Proof. We have that incidence path width fpt-dominates one-sided tree width because every path decomposition is also one-side tree decomposition: Both decomposition build on incidence graphs and the property that for the set of bags induced by any clause vertex should form a path is weaker than if all bags need to form a path. Since resolution is fpt-automatable with regard to primal tree width (Corollary 4.2), we conclude with Lemma 3.29 that resolution is fpt-automatable with regard to incidence tree width. □

The parameter path width is particularly interesting since we can also embed in into the Frege hierarchy.

Lemma 4.30. *Path width dominates special tree width.*

Proof. Take a a path decomposition. We see that this is also a special tree decomposition since the property that for every vertex in G, the bags induce a directed path is weaker than that the all bags together form a path. The path of a path decomposition is rooted and can be therefore transformed to a directed graph. □

Therefore, it seems natural to ask whether Frege is fpt-automatable with regard to path width. If this was the case, parameters that dominate path width are also fpt-automatable.

Conjecture 4.31. *Treelike, bounded depth Frege is fpt-automatable with regard to path width.*

Since we associate a graph with a Frege proof, we can restrict the depth, thus the longest path from leaf to root, of the proof graph. This allows a smaller class of possible proofs and is therefore, more likely to be fpt-automatable.

In the thesis, we have not focused on results fpt-boundedness results, even though they are a natural continuation of applying parameterized complexity to questions in proof complexity. One reason why we have shown fpt-automatability for incidence path with help of Theorem 4.29 is that it additionally grant fpt-boundedness.

Nonetheless, we see again that fpt-dominance is a very strong tool. Furthermore, we are going to use it to deduce an other parameter. This is possible since the graph measures path width and cut width are related.

4.6 Incidence cut width

Cut width is another parameter function such as path width and tree width. Furthermore, cut width is closely related to them. Formally, we define cut width as follows.

Definition 4.32. *Given a graph $G = (V, E), |V| = n$, a numbering of G is a one-to-one mapping $L : V \to \{1, \ldots, n\}$. The* cut width of an order L *is* $\max_{1 \leq p \leq n} |\{(u, v) \in E : L(u) \leq p < L(v)\}|$. *The* cut width $c(G)$ of G *is the minimum cut width over all orders.*

Consider the following example. Note that the graph in the example is the same one as in the example of tree width Example 3.23 and path width Example 4.28.

Example 4.33. *Take the graph $G = (V, E)$ depicted as follows.*

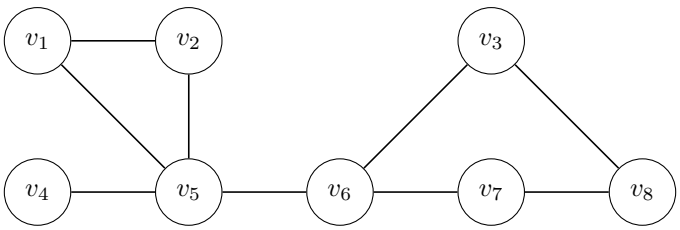


Figure 23: Graph G

We order the vertices from in a row such that $L(v_1) = 1$ for the left-most vertex v_1 and the ordering increases going to the right. For the the right-most vertex v_3, we have $L(v_3) = 8$.

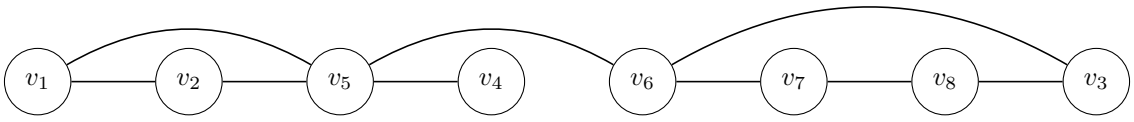


Figure 24: An ordering of graph G

We see that any vertical cut crosses at most two edges. Furthermore, we can modify the ordering to decrease this number since v_1, v_2, v_3 form a cycle. Subsequently, we have $cw(G) = 2$.

Let $cw(G)$ denote the cut width of a graph G and let $pw(G)$ denote the path width of a graph G.

Lemma 4.34. *For every graph G, $pw(G) \leq cw(G)$.*

Proof. See [KS93], Lemma 3. □

Proposition 4.35. *Resolution is fpt-automatable with regard to the parameter cut width.*

Proof. Due to the previous Lemma 4.34, we have that cut width is dominated by path width. Since with resolution is fpt-automatble with regard to path width (see Corollary 4.4), we conclude with Lemma 3.29 that resolution is fpt-automatable with regard to cut width. □

In exactly the same way, we get a fpt-bounded result.

Proposition 4.36. *Resolution is fpt-automatable with regard to the parameter cut width.*

Proof. Due to the previous Lemma 4.34, we have that cut width is dominated by path width. Since with resolution is fpt-automatble with regard to path width (see Corollary 4.3), we conclude with Lemma 3.30 that resolution is fpt-automatable with regard to cut width. □

The fpt-results for treelike resolution and resolution have their correspondents within the theory of automatability.

4.7 Literature on automatability results for resolution

Similar to the proof for non-fpt-automatability with regard to tree width, the following theorems show that if resolution is automatable then something unlikely is true. In case of non-fpt-automatability with regard to tree width, the unlikely event was to break the encryption scheme RSA. For the following result, the unlikely event is that parts of the parameterized hierarchy collapses.

Definition 4.37. *The class* FPR *(fixed-parameter randomized) of parameterized problems consists of all languages $L \subseteq \Sigma^* \times \mathbb{N}$ for which there exists a probabilistic algorithm Φ, 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 $f(k)|x|^c$*
- *if $(x, k) \in L$, then $P[\Phi(x, k) = 1] \geq \frac{1}{2}$*
- *if $(x, k) \notin L$, then $P[\Phi(x, k) = 1] \geq 0$.*

The class coFPR *is the set of all languages whose complement is in* FPR*. Thus,* coFPR $:= \{L \mid \Sigma^* \setminus L \in$ FPR$\}$.

Since *co*FPR is an expressive complexity class it could be that case that W[P] $\subseteq$ *co*FPR. We have that FPR = FPT with the derandomization of Eickmeyer-Grohe-Gruber [EGG08], so if FPT $\neq$ W[P], the following theorem by Michael Alekhnovich and Alexander Razborov (Theorem 2.7 in [AR08]) can be interpreted as an automatability result.

Theorem 4.38. *If either resolution or tree-like resolution is automatable, then* W[P] $\subseteq$ *co*FPR.

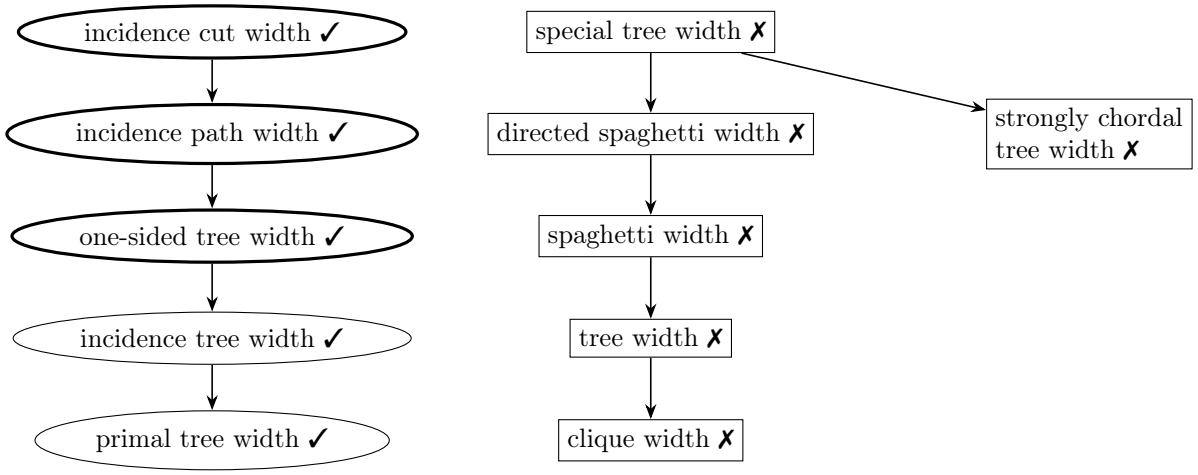
Another result by Albert Atserias and Moritz Müller shows that resolution is not automatable unless P = NP (see in [AM20]).

Theorem 4.39. *Resolution is not automatable in polynomial time unless* NP $\subseteq$ P.

Even though automatability and fpt-automatability are related (Lemma 2.15), we see that the automatability result do not contradict the fpt-automatability result. Indeed, there even add to the motivation to research on fpt-automatability since the result show that with regard to some parameters difficult problems such as automating resolution become feasible.

5 Conclusion

In the thesis, we have developed the notion of fpt-automatability and fpt-boundedness and used them to show a sequence of results. Recall the following sketch overviewing them.



We can see that the sketch splits into disjoint parts on resolution and Extended Frege. While all the results on resolution show fpt-automatability, Extended Frege seems more likely to be not fpt-automatable. Intuitively, this makes sense since Extended Frege is a more expressive proof system.

To show the results, we used very different methods. We started with the theory of interpolation and showed that Extended Frege is not fpt-automatable with regard to tree width. For this, we carved out results from bounded arithmetic and proof translations and subsequently, constructed a contradiction with the security of the encryption scheme RSA.

With the graph-theoretic notion of intersection graphs, we motivated the parameters special tree width, strongly chordal tree width, directed spaghetti width and spaghetti width. As a consequence, we generalized the result on tree width, taking advantage of a characterization result of these parameters. From these results on, with fpt-dominance, we concluded the non-fpt-automatability of clique width.

For showing fpt-automatability for resolution, we provided an algorithm for the parameter primal tree width that computes a refutation and presented this algorithm in a version for treelike resolution and conventional resolution. We have that this parameter is particularly interesting due to the jump from the fact that Extended Frege is non-fpt-automatable with regard to tree width whereas resolution is fpt-automatable with regard to incidence tree width.

For the parameter incidence path width, we cited a similar refutation algorithm and by fpt-dominance, we concluded the fpt-automatability of one-sided tree width and incidence cut width, noting the additional property fpt-boundedess.

The thesis focused on automatability but an analogue research question to this paper would be a collection of parameters for fpt-boundedness. Some results such as one-sided tree width, incidence path and incidence cut width have this property. If we were to make a hierarchy, those parameters would be at the very top since they are fpt-automatable and at the same time fpt-bounded.

In addition, one could add other proof systems to the overview map. It poses the question where we would locate the difficulty of automation/boundedness. A starting point for this, could be our conjecture that bounded depth Frege is automatable with regard to path width. From this, we could generalize the proof to Extended Frege and hence, have an example for fpt-automatability for Extended Frege.

Similarly, further research could try to loosen the assumptions for the non-fpt-automatability results for Extended Frege. So far, we assumed that RSA is secure building on a paper by Krajicek and Pudlak [KP98]. A paper by Bonet, Pitassi and Raz [BPR97] logically follows the paper by Krajicek and Pudlak, generalizing the results. They show that a weaker Frege proof system called TC_0-Frege is connected to the security of the encryption protocol Diffie-Hellman. Further research could work on fpt-automatability results analogue to our results based on the security of Diffie-Hellman.

References

[AB04] Albert Atserias and Maria L. Bonet. "On the automatizability of resolution and related propositional proof systems". In: *Information and Computation* 189.2 (2004), pp. 182–201.

[AFT11] Albert Atserias, Johannes K. Fichte, and Marc Thurley. "Clause-learning algorithms with many restarts and bounded-width resolution". In: *Journal of Artificial Intelligence Research* 40 (2011), pp. 353–373.

[Ale+88] Werner Alexi et al. "RSA and Rabin functions: Certain parts are as hard as the whole". In: *SIAM Journal on Computing* 17.2 (1988), pp. 194–209.

[AM20] Albert Atserias and Moritz Müller. "Automating resolution is NP-hard". In: *Journal of the ACM (JACM)* 67.5 (2020), pp. 1–17.

[AR08] Michael Alekhnovich and Alexander A. Razborov. "Resolution is not automatizable unless W[P] is tractable". In: *SIAM Journal on Computing* 38.4 (2008), pp. 1347–1363.

[BBP95] Maria L. Bonet, Samuel R Buss, and Toniann Pitassi. "Are there hard examples for Frege systems?" In: *Feasible Mathematics II* (1995), pp. 30–56.

[BN21] Sam Buss and Jakob Nordström. "Proof complexity and SAT solving". In: *Handbook of Satisfiability*. IOS Press, 2021, pp. 233–350.

[Bod+17] Hans L. Bodlaender et al. "Characterizing width two for variants of treewidth". In: *Discrete Applied Mathematics* 216 (2017), pp. 29–46.

[Bod98] Hans L. Bodlaender. "A partial k-arboretum of graphs with bounded treewidth". In: *Theoretical computer science* 209.1-2 (1998), pp. 1–45.

[BPR00] Maria L. Bonet, Toniann Pitassi, and Ran Raz. "On interpolation and automatization for Frege systems". In: *SIAM Journal on Computing* 29.6 (2000), pp. 1939–1967.

[BPR97] Maria L. Bonet, Toniann Pitassi, and Ran Raz. "No feasible interpolation for TC_0-Frege proofs". In: *Proceedings 38th Annual Symposium on Foundations of Computer Science*. 1997, pp. 254–263.

[Bus12] Samuel R. Buss. "Towards NP-P via proof complexity and search". In: *Annals of Pure and Applied Logic* 163.7 (2012). The Symposium on Logical Foundations of Computer Science 2009, pp. 906–917.

[CN10] Stephen Cook and Phuong Nguyen. *Logical foundations of proof complexity*. Vol. 11. Cambridge University Press Cambridge, 2010.

[CO00] Bruno Courcelle and Stephan Olariu. "Upper bounds to the clique width of graphs". In: *Discrete Applied Mathematics* 101.1-3 (2000), pp. 77–114.

[Cou12] Bruno Courcelle. "On the model-checking of monadic second-order formulas with edge set quantifications". In: *Discrete Applied Mathematics* 160.6 (2012), pp. 866–887.

[CR19] Andrea Cali and Igor Razgon. "Regular resolution for CNFs with almost bounded one-sided treewidth". In: *arXiv preprint arXiv:1905.10867* (2019).

[CR23] Stephen A. Cook and Robert A. Reckhow. "The relative efficiency of propositional proof systems". In: *Logic, Automata, and Computational Complexity: The Works of Stephen A. Cook*. 2023, pp. 173–192.

[DMS07] Stefan Dantchev, Barnaby Martin, and Stefan Szeider. "Parameterized proof complexity: a complexity gap for parameterized tree-like resolution". In: *Electronic Colloquium on Computational Complexity (ECCC), Technical Report TR07-001*. Citeseer. 2007.

[EGG08] Kord Eickmeyer, Martin Grohe, and Magdalena Gräber. "Approximation of Natural W[P]-Complete Minimisation Problems Is Hard". In: *2008 23rd Annual IEEE Conference on Computational Complexity*. 2008, pp. 8–18. DOI: 10.1109/CCC.2008.24.

[Far81] Martin Farber. *Applications of linear programming duality to problems involving independence and domination*. Simon Fraser University, Computing Science, 1981.

[Fel+09] Michael R Fellows et al. "Clique-width is NP-complete". In: *SIAM Journal on Discrete Mathematics* 23.2 (2009), pp. 909–939.

[FM12] Jörg Flum and Moritz Müller. "Some definitorial suggestions for parameterized proof complexity". In: *International Symposium on Parameterized and Exact Computation*. Springer. 2012, pp. 73–84.

[Ima17] Kensuke Imanishi. "An upper bound for resolution size: Characterization of tractable sat instances". In: *International Workshop on Algorithms and Computation*. Springer. 2017, pp. 359–369.

[Klo94] Ton Kloks. *Treewidth: computations and approximations*. Springer, 1994.

[KP98] Jan Krajicek and Pavel Pudlak. "Some consequences of cryptographical conjectures for S_2^1 and EF". In: *Information and Computation* 140.1 (1998), pp. 82–94.

[Kra19] Jan Krajicek. *Proof complexity*. Vol. 170. Cambridge University Press, 2019.

[Kra95] Jan Krajicek. *Bounded arithmetic, propositional logic and complexity theory*. Vol. 60. Cambridge University Press, 1995.

[KS93] Ephraim Korach and Nir Solel. "Tree-width, path-width, and cutwidth". In: *Discrete Applied Mathematics* 43.1 (1993), pp. 97–101.

[LGJ83] Harry R. Lewis, Michael R. Garey, and David S. Johnson. "Computers and intractability. A guide to the theory of NP-completeness. WH Freeman and Company, San Francisco1979, 338 pp." In: *The Journal of Symbolic Logic* 48.2 (1983), pp. 498–500.

[McC90] Kevin S. McCurley. "The discrete logarithm problem". In: *Proc. of Symp. in Applied Math.* Vol. 42. USA. 1990, pp. 49–74.

[MM99] Terry A. McKee and Fred R. McMorris. *Topics in intersection graph theory*. SIAM, 1999.

[MW86] Clyde L. Monma and Victor K. Wei. "Intersection graphs of paths in a tree". In: *Journal of Combinatorial Theory, Series B* 41.2 (1986), pp. 141–181.

[Nie06] Rolf Niedermeier. *Invitation to fixed-parameter algorithms*. Vol. 31. OUP Oxford, 2006.

[PS10] Toniann Pitassi and Rahul Santhanam. "Effectively Polynomial Simulations." In: *ICS*. 2010, pp. 370–382.

[Pud03] Pavel Pudlak. "On reducibility and symmetry of disjoint NP pairs". In: *Theoretical Computer Science* 295.1-3 (2003), pp. 323–339.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In: *Communications of the ACM* 21.2 (1978), pp. 120–126.

[SFG07] Thomas Schwentick, J. Flum, and M. Grohe. "Parameterized complexity theory. Texts in Theoretical Computer Science." In: *Bulletin of Symbolic Logic* 13.2 (2007), pp. 246–248.

Appendix

Definition 5.1. *The theory BASIC consists of the following 32 axiom in the language L:*

1. $a \leq b \to a \leq b + 1$
2. $a \neq a + 1$
3. $0 \leq a$
4. $(a \leq b \wedge a \neq b) \to a + 1 \leq b$
5. $a \neq 0 \to 2a \neq 0$
6. $a \leq b \vee b \leq a$
7. $(a \leq b \wedge b \leq a) \to a = b$
8. $(a \leq b \wedge b \leq c) \to a \leq c$
9. $|0| = 0$
10. $a \neq 0 \to (|2a| = |a| + 1 \wedge |2a + 1| = |a| + 1)$
11. $|1| = 1$
12. $a \leq b \to |a| \leq |b|$
13. $|a\#b| = |a \cdot b| + 1$
14. $|0\#a| = 1$
15. $a \neq 0 \to (1\#(2a) = 2(1\#a)) \wedge (1\#(2a + 1) = 2(1\#a))$
16. $a\#b = b\#a$
17. $|a| = |b| \to a\#b = a\#c$
18. $|a| = |b| + |c| \to a\#d = (b\#d) \cdot (c\#d)$
19. $|a| \leq |a| + |b|$
20. $(a \leq b \wedge a \neq b) \to (2a + 1 \leq 2b \wedge 2a + 1 \neq 2b)$
21. $a + b = b + a$
22. $a + 0 = a$
23. $a + (b + 1) = (a + b) + 1$
24. $a + (b + c) = (a + b) + c$
25. $a + b \leq a + c \to b \leq c$
26. $a \cdot 0 = 0$
27. $a \cdot (b + 1) = a \cdot b + a$
28. $a \cdot b = b \cdot a$
29. $a \cdot (b + c) = a \cdot b + a \cdot c$
30. $1 \leq a \to ((a \cdot b \leq a \cdot c) \equiv (b \leq c))$
31. $a \neq 0 \to |a| = |\lfloor(\frac{a}{2})\rfloor| + 1$
32. $a = \lfloor(\frac{b}{2})\rfloor \equiv (2a = b \vee 2a + 1 = b)$

Definition 5.2. *Let $A(a_1, \ldots, a_k)$ be a bounded formula in Σ_1^b. Let $q(x)$ be a bounding polynomial for the formula A. For every m, we construct a bounded quantified formula $\|A\|^m_{q(m)}$ with atoms $\overline{p_i}$ for $i \in \{1, \ldots, k\}$ and $\overline{p_i} = (p_i^0, \ldots, p_i^{q(m)})$. We proceed by induction on the formula A:*

- *For an atomic formula $t(\overline{a}) = s(\overline{a})$ define*

$$\|A\|^m_{q(m)} := \exists x_0 \ldots x_{q(m)}, y_0 \ldots y_{q(m)} B_t(\overline{p_1}, \ldots, \overline{p_k}, q_j/x_j) \wedge B_s(\overline{p_1}, \ldots, \overline{p_k}, q_j/y_j) \wedge \bigwedge_{i \leq q(m)} x_i \equiv y_i$$

  where we define the tuples $\overline{p_1}, \ldots, \overline{p_k}, \overline{q}$ in B have $q(m)+1$ bits, with p_i^j for $j > m$ not occurring and with not all q^j necessarily occurring.

- *For an atomic formula $t(\overline{a}) \leq s(\overline{a})$ define*

$$\|A\|^m_{q(m)} := \exists x_0 \ldots x_{q(m)}, y_0 \ldots y_{q(m)} B_t(\overline{p_1}, \ldots, \overline{p_k}, q_j/x_j) \wedge B_s(\overline{p_1}, \ldots, \overline{p_k}, q_j/y_j)$$

$$\wedge \bigwedge_{0 \leq i \leq q(m)} ((\bigwedge_{i+1 \leq j \leq q(m)} x_i \equiv y_i) \wedge x_i \to y_i)$$

  (the last conjunct defines the lexicographic order on $\overline{x}, \overline{y}$).

- *For $A = \neg A_1$ define*

$$\|A\|^m_{q(m)} = \neg \|A_1\|^m_{q(m)}$$

- *For $A = A_1 \circ A_2$ where $\circ$ is either $\wedge$ or $\vee$ define*

$$\|A\|^m_{q(m)} = \|A_1\|^m_{q(m)} \circ \|A_2\|^m_{q(m)}$$

- *For $A(a) = \exists x \leq |t|.A_1(a, x)$ define*

$$\|A\|^m_{q(m)} = \bigvee_{\overline{\varepsilon}} \|b \leq |t| \wedge A(a, b)\|^m_{q(m)}(\overline{q}/\overline{\varepsilon})$$

  and for $A(a) = \forall x \leq |t|.A_1(a, x)$ define

$$\|A\|^m_{q(m)} = \bigwedge_{\overline{\varepsilon}} \|b \leq |t| \to A(a, b)\|^m_{q(m)}(\overline{q}/\overline{\varepsilon})$$

  where $\overline{\varepsilon}$ ranges over $q(m)+1$ tuples of $0, 1$ such that $\varepsilon_i = 0$ for $i > |q(m)|$.

- *For $A(a) = \exists x \leq t.A_1(a, x)$ with t not of the form $|s|$ define*

$$\|A\|^m_{q(m)} = \bigvee_{\overline{\varepsilon}} \|b \leq t \wedge A(a, b)\|^m_{q(m)}(\overline{q}/\overline{x})$$

  and for $A(a) = \forall x \leq t.A_1(a, x)$ with t not of the form $|s|$ define

$$\|A\|^m_{q(m)} = \bigwedge_{\overline{\varepsilon}} \|b \leq t \to A(a, b)\|^m_{q(m)}(\overline{q}/\overline{x})$$

  where $\overline{q}$ is the tuple associated with b.

Acknowledgements

We very much thank Noel Arteche for his inspiration and help.