FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Structure in #SAT and QBF

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

DI Friedrich Slivovsky
Registration Number 0202583

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Prof. Dr. Stefan Szeider

The dissertation has been reviewed by:

Stefan Szeider	Hubie Chen

Vienna, 24th March, 2015

Friedrich Slivovsky

Erklärung zur Verfassung der Arbeit

DI Friedrich Slivovsky
Sobieskigasse 18/5, 1090 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 24. März 2015

Friedrich Slivovsky

To Dad,
whose passion for the APL programming language
didn't stop me from studying computer science,
and Mom,
who uses me as her web browser.

Acknowledgements

This thesis took some time to finish, and I am indebted to several people who supported me along the way. First and foremost, I would like to thank my advisor Stefan Szeider for guiding me through the treacherous waters of complexity towards islands of tractable research. Collaborating with Stefan has been a pleasure, and his work ethic and mathematical rigor are a continuing source of inspiration. I would also like to thank my co-advisor Stefan Woltran for proofreading and providing valuable suggestions for improving this thesis.

I had the honor of being among the chosen few enrolled in the doctoral program "mathematical logic in computer science". I am grateful to Agata Ciabattoni, who pointed out this program to me and encouraged me to apply. I also thank Chris Fermüller for his kind support during the beginning of my graduate studies.

Cheers to Daniel Paulusma, whose scientific expertise is matched only by his encyclopedic knowledge of independent music, for hosting me during two pleasant research visits at Durham University.

Special thanks go to my friends from the department, in particular Harald, Simone, Robert, Ronald, and the latest additions to our roster, Eduard and Neha, for enabling my coffee addiction and keeping me company in the ivory tower.

Finally, I would like to thank the following institutions for financial support during my PhD studies: the European Research Council, the FWF Austrian Science Fund, the Vienna Science and Technology Fund, and the Vienna University of Technology.

Kurzfassung

In der Komplexitätstheorie geht man davon aus, dass für zahlreiche zentrale Probleme keine effizienten Algorithmen existieren. Einige dieser Probleme lassen sich in der Praxis dennoch lösen, was üblicherweise damit begründet wird, dass praxisrelevante Instanzen „Struktur" aufweisen, die von Lösungsverfahren ausgenutzt werden kann. Die vorliegende Arbeit untersucht konkrete Ausprägungen dieses Strukturbegriffs für zwei äußerst schwierige Probleme: das Erfüllbarkeitsproblem quantifizierter boolescher Formeln (QSAT) und das Abzählproblem von Modellen aussagenlogischer Formeln (#SAT).

QSAT. Die Alternierung von Existenz- und Allquantoren im Präfix quantifizierter boolescher Formeln erzeugt Abhängigkeiten unter Variablen, die von Lösungsverfahren für QSAT berücksichtigt werden müssen. Gängige Verfahren gehen davon aus, dass alle prinzipiell möglichen Abhängigkeiten tatsächlich bestehen. Oft ist jedoch nur ein Bruchteil dieser Abhängigkeiten triftig, während die übrigen, „falschen" Abhängigkeiten lediglich zu unnötigen Einschränkungen führen. Wir untersuchen Dependency Schemes als Mittel zur Identifikation solcher falscher Abhängigkeiten, mit folgenden Resultaten.

- Wir zeigen, dass das Resolution-Path Dependency Scheme in Polynomialzeit berechnet werden kann. Unter den derzeit bekannten Dependency Schemes erkennt das Resolution-Path Dependency Scheme eine maximale Menge falscher Abhängigkeiten.
- Wir definieren notwendige und hinreichende Bedingungen für den Einsatz von Dependency Schemes in suchbasierten Algorithmen für QSAT und zeigen, dass diese Bedingungen von den in DepQBF implementierten Dependency Schemes sowie einer Variante des Resolution-Path Dependency Schemes erfüllt werden.
- Dependency Schemes waren ursprünglich zum Verschieben von Quantoren im Präfix quantifizierter boolescher Formeln gedacht. Wir zeigen, dass gängige Dependency Schemes eine allgemeinere Operation zur Manipulation des Präfixes erlauben, und demonstrieren, wie diese Operation zur Minimierung der Alternierungstiefe von Formeln verwendet werden kann.

#SAT. Das Zählen von Modellen aussagenlogischer Formeln ist nicht nur im Allgemeinen schwer, sondern selbst für Formelklassen, für die das zugehörige Entscheidungsproblem

in Polynomialzeit gelöst werden kann, beispielsweise für Horn- oder 2CNF-Formeln. Wir untersuchen den Effekt von strukturellen (über Graphenparameter definierten) Einschränkungen auf die Komplexität von #SAT und bestimmen neue Formelklassen, die das Zählen von Modellen in Polynomialzeit erlauben.

- Das Kontrahieren von Modulen in Graphen ist eine gängige Technik zur Vereinfachung kombinatorischer Optimierungsprobleme. Wir definieren die modulare Baumweite eines Graphen als seine Baumweite nach dem Kontrahieren von Modulen und zeigen, dass #SAT für Formeln, deren Inzidenzgraphen beschränkte modulare Baumweite haben, in Polynomialzeit gelöst werden kann.

- Die symmetrische Cliquenweite ist ein Parameter, der sowohl Baumweite als auch modulare Baumweite verallgemeinert. Wir zeigen, dass #SAT für Formelklassen, deren Inzidenzgraphen beschränkte symmetrische Cliquenweite aufweisen, in Polynomialzeit lösbar ist.

Abstract

Computational problems that are intractable in general can often be efficiently resolved in practice due to latent structure in real-world instances. This thesis considers structural properties that can be used in the design of more efficient algorithms for two highly intractable problems: the satisfiability problem of quantified Boolean formulas (QSAT) and propositional model counting (#SAT).

QSAT. The nesting of existential and universal quantifiers in quantified Boolean formulas (QBFs) generates dependencies among variables that have to be respected by QSAT solvers. In standard decision algorithms, it is assumed that all possible variable dependencies exist. But often, only a fraction of these dependencies is realized, while the remaining, "spurious" dependencies lead to unnecessary restrictions that inhibit solver performance. We study dependency schemes as a means to identifying spurious dependencies and establish the following results.

- Among dependency schemes considered in the literature, the resolution-path dependency scheme identifies a maximal set of spurious dependencies. We prove that the resolution-path dependency scheme can be computed in polynomial time.

- We state sufficient conditions for the sound deployment of dependency schemes in search-based QSAT solvers and prove that these conditions are met by several dependency schemes, including those implemented in the solver DepQBF and a variant of the resolution-path dependency scheme.

- We show that known dependency schemes support a reordering operation that is more powerful than quantifier shifting, and present an application to the reduction of quantifier alternations of a QBF.

#SAT. The model counting problem (#SAT) asks for the number of satisfying assignments of a propositional formula in conjunctive normal form. This problem is hard even for classes that admit satisfiability testing in polynomial time, such as Horn or 2CNF formulas. We prove the following results on the complexity of #SAT with respect to structural parameters based on graph width measures, identifying new classes of formulas amenable to efficient model counting.

- Contraction of modules in a graph is a commonly used preprocessing step in combinatorial optimization. We define the modular treewidth of a graph as its treewidth after contraction of modules, and prove that #SAT is polynomial-time tractable for classes of formulas with incidence graphs of bounded modular treewidth.
- Symmetric clique-width is a graph parameter that generalizes treewidth as well as modular treewidth. We show that #SAT is polynomial-time tractable for classes of formulas with incidence graphs of bounded symmetric clique-width.

Contents

Preface

The last decade has seen an impressive increase in the performance of propositional satisfiability (SAT) solvers. Undeterred by NP-completeness, researchers have engineered decision procedures to a point where they can solve real-world instances with millions of variables. One can construct small formulas that bring these solvers to their knees, but this hardly matters in practice, where SAT solvers are used as part of state-of-the-art systems for formal verification and planning [13, 17, 84, 77], and "reduction to SAT" is becoming a viable strategy for dealing with NP-complete problems. Theorists are still struggling to explain this unexpected triumph of SAT solvers over intractability [121].

This success story warrants some optimism about the efficient resolution of even more challenging problems. We consider two such problems: propositional model counting (#SAT) and the satisfiability problem of quantified Boolean formulas (QSAT). Both are generalizations of SAT, and both have offered more resistance in practice, where the largest instances state-of-the-art procedures can deal with are significantly smaller than those decided by SAT solvers. This is in line with results in complexity theory: every problem in the polynomial hierarchy can be solved by a polynomial-time algorithm with access to a #SAT oracle [116], and QSAT is complete for the class PSPACE which contains the entire polynomial hierarchy [113]. By comparison, SAT is located at the first level of the polynomial hierarchy. Efficient algorithms for #SAT and QSAT have many potential applications that are beyond SAT, such as probabilistic reasoning, conditional planning, and unbounded model checking [6, 105, 95, 75].

SAT solvers that perform well on real-world instances are typically slow on randomly generated ones, indicating that the former exhibit latent structure that SAT solvers can exploit [54]. This thesis studies structural properties of #SAT and QSAT instances that can be used in the design of more efficient algorithms for these problems.

Part I: Dependency Schemes for QBF. Quantified Boolean formulas (QBFs) extend propositional formulas with existential and universal quantification over truth values. Nested quantifiers give rise to dependencies among variables that must be respected by decision procedures for QSAT, and standard algorithms deal with this by implicitly making the most conservative assumption about variable dependencies: if x is in the scope of y, then x depends on y. Part I explores *dependency schemes* as a means to relaxing this assumption. Formally, a dependency scheme is a mapping that associates a QBF (in prenex normal form) with a binary relation on its variables that indicates

potential variable dependencies. This relation, which we call a *dependency relation*, is an overapproximation that may contain spurious dependencies.

Dependency schemes can be used in practice only if they can be computed efficiently, so a tradeoff has to be made between the time it takes to compute a dependency relation and the number of spurious dependencies it contains. In Chapter 4, we prove that the so-called *resolution-path dependency scheme* can be computed in polynomial time. Among those introduced in the literature so far, this dependency scheme computes an inclusion-minimal dependency relation.

Dependency schemes are best known for their integration in the QBF solver DepQBF [15, 82]. Unfortunately, the resolution-path dependency scheme cannot be used in DepQBF without modifications, due to a mismatch between the original definition of dependency schemes and their intended use in QSAT solvers. We deal with this mismatch in Chapter 5, which forms the core of Part I. We demonstrate that the original definition of dependency schemes is not restrictive enough to warrant their use in solvers, and identify sufficient conditions based on proof systems that generalize quantified resolution. We then proceed to show that several dependency schemes do in fact satisfy these conditions, including those implemented in DepQBF and a variant of the resolution-path dependency scheme.

Chapter 6 revisits the original definition of dependency schemes, which requires soundness of certain reorderings of a formula's quantifier prefix [102]. We show that known dependency schemes support a more general (and arguably, natural) reordering operation and present an application in the (heuristic) reduction of quantifier alternations of a formula, a measure closely tied to the complexity of QSAT.

Part II: Propositional Model Counting. #SAT asks for the number of satisfying assignments of a propositional formula. This problem is hard even for classes of formulas for which SAT is well-known to be easy, such as Horn and 2CNF formulas. Part II presents new tractability results for #SAT with respect to structural parameters. These go beyond known results on the tractability of #SAT parameterized by the incidence treewidth (the treewidth of the incidence graph) or the signed incidence clique-width (the directed clique-width of the signed incidence graph)[1] and identify new classes amenable to efficient model counting.

Specifically, we show that #SAT is tractable for formulas with incidence graphs of bounded modular treewidth or bounded symmetric clique-width. The treewidth of a graph measures how close it is to being a tree – a graph has treewidth 1 if it is a tree (or forest), and the larger its treewidth the less it looks like a tree [18]. A module of a graph is a set S of vertices which have the same neighborhood outside S. Contracting modules, that is, replacing each module by a single vertex, is a common preprocessing step in combinatorial optimization. Defining the *modular treewidth* of a graph as its treewidth

[1]The incidence graph has variables and clauses of a formula as its vertices and contains an edge between a variable x and a clause C if x occurs in C. The signed incidence graph is a directed version of the incidence graph where the orientation of an edge indicates whether a variable occurs negated or unnegated.

after contraction of modules, Chapter 11 proves that #SAT is polynomial-time tractable for formulas whose modular incidence treewidth (the modular treewidth of the incidence graph) is bounded by a constant. The modular incidence treewidth of a formula is never larger than its incidence treewidth, and the difference can grow arbitrarily large, so this result yields tractability in cases that are out of reach of algorithms exploiting small incidence treewidth.

Chapter 12 goes even further and shows that #SAT is polynomial-time tractable for formulas of bounded symmetric incidence clique-width (the symmetric clique-width of the incidence graph). *Symmetric clique-width* is a generalization of treewidth that is equivalent to several other graph parameters such as clique-width or rank-width [69], and our result can be equivalently stated in terms of these parameters. The symmetric incidence clique-width of a class of formulas is bounded whenever its modular incidence treewidth or signed incidence clique-width is bounded, but there are classes of unbounded modular incidence treewidth or signed incidence clique-width but bounded symmetric clique-width. As far as polynomial-time tractability of #SAT is concerned, our result is currently the most general tractability result for a structural parameter based on a graph width measure.

Assuming that readers will want to focus on the topic they are more interested in, each part has been written so as to be self-contained. In particular, we chose to give each part its own chapter on preliminaries, at the cost of introducing some redundancy.

Publications. This thesis is based on the following publications:

1. Friedrich Slivovsky and Stefan Szeider. Computing Resolution-Path Dependencies in Linear Time. *Theory and Applications of Satisfiability Testing - SAT 2012.* Lecture Notes in Computer Science, vol. 7317, pp. 58-71, Springer 2012.

2. Daniel Paulusma, Friedrich Slivovsky, and Stefan Szeider. Model Counting for CNF Formulas of Bounded Modular Treewidth. *Symposium on Theoretical Aspects of Computer Science - STACS 2013.* Leibniz International Proceedings in Informatics, vol. 20, pp 55-66, 2013.

3. Friedrich Slivovsky and Stefan Szeider. Model Counting for Formulas of Bounded Clique-Width. *International Symposium on Algorithms and Computation - ISAAC 2013.* Lecture Notes in Computer Science, vol. 8283, pp. 677-687, Springer 2013.

4. Friedrich Slivovsky and Stefan Szeider. Variable Dependencies and Q-Resolution. *Theory and Applications of Satisfiability Testing - SAT 2014.* Lecture Notes in Computer Science, vol. 8561, pp. 269-284, Springer 2014.

Part I

Dependency Schemes for QBF

CHAPTER 1

Introduction

Quantified Boolean Formulas (QBFs) enrich propositional formulas with quantification over truth values. The satisfiability problem of QBFs (QSAT) offers natural and compact encodings of problems that cannot be succinctly expressed in SAT, such as unbounded model checking and conformant planning [50]. This increase in expressivity comes at the cost of higher complexity: QSAT is PSPACE-complete [113]. The problem also appears to be much harder to solve in practice than SAT, and—significant progress notwithstanding—QSAT solvers have not yet reached the level of maturity of modern SAT solvers.

To lift techniques from SAT to QSAT one must take into account dependencies among variables generated by nested quantifiers. Consider the QBF

$$\Phi = \forall x \exists y.(x \vee \neg y) \wedge (\neg x \vee y).$$

This formula is true: if x is set to *true*, the first clause is satisfied and the second clause can be satisfied by setting y to *true*; if x is set to *false*, the second clause is satisfied and the first clause can be satisfied by setting y to *false*. However, there is no single truth assignment to variable y that is guaranteed to satisfy both clauses – the satisfying assignment *depends* on the value assigned to x. Intuitively, this is what we mean by a variable dependency.

Common decision algorithms and preprocessing techniques for QSAT implicitly make the most conservative assumption about variable dependencies: if variable x is in the scope of variable y, then x depends on y. In many cases, this is needlessly restrictive. For instance, take the formula

$$\Psi = \forall x \forall u \exists y \exists e.(x \vee \neg y) \wedge (\neg x \vee y) \wedge (u \vee \neg e) \wedge (\neg u \vee e).$$

This QBF consists of two variable-disjoint copies of Φ, with quantifiers moved to the front so as to comply with the common Prenex Conjunctive Normal Form, or PCNF (a formula in PCNF consists of a leading quantifier prefix, followed by a purely propositional

formula, called the matrix, in conjunctive normal form). Again, it is not difficult to see that Ψ is satisfiable: for any choice of truth values for x and u we simply assign y the same value as x and e the same value as u. Such a strategy for choosing assignments in a way that satisfies the matrix is called a *model* of Ψ. In this particular model, the value assigned to y does not depend on the value of u, and the value assigned to e does not depend on the value of x.

If we know that a formula has models with limited dependencies, we can apply more aggressive techniques in pruning the search space of the corresponding QSAT instance. This is of little practical use if, as in the above case, we already have a model and know that the formula is true. Instead, we would like to determine in advance that, if a formula has models at all, it has models whose dependencies satisfy certain restrictions. Unfortunately, deciding questions of this kind is typically (at least) PSPACE-hard [102], so obtaining information on variable dependencies seems to presuppose the existence of the very solvers whose performance that information is intended to improve.

Dependency schemes offer a way out of this vicious circle by trading precision for efficiency. Formally, a dependency scheme is a mapping that associates a PCNF formula with a binary relation (called a dependency relation) on its variables. This dependency relation is an *overapproximation* of variable dependencies. That is, a pair (x, y) in this relation indicates that y *may* depend on x, but this dependency can be spurious. Conversely, if the dependency relation does not contain a pair (x, y), then y definitely *does not* depend on x. While dependency relations do not always provide the most accurate representation of variable dependencies, they can be computed very efficiently. For the formula Ψ presented above, the *standard dependency scheme* will detect that y does not depend on u and that e does not depend on x based on a linear-time analysis of Ψ. In the following chapters, we will survey dependency schemes and show how they can be applied in QSAT solving.

Other things being equal, we prefer dependency schemes that compute smaller dependency relations and induce fewer constraints in applications. Among the dependency schemes considered in the literature, the *resolution-path dependency scheme* computes a minimal (with respect to set inclusion) dependency relation. We prove that this dependency scheme is still tractable, meaning that there is a polynomial-time algorithm for computing its dependency relation for a given PCNF formula. The question of whether such an algorithm exists was posed as an open problem in the original paper on resolution-path dependencies [120].

Tractability of a dependency scheme is a minimal requirement for its application in QSAT solvers such as DepQBF [15, 82]. DepQBF is a search-based solver that implements an algorithm known under the acronym QDPLL [25]. QDPLL is a generalization of the Davis-Putnam-Logemann-Loveland (DPLL) procedure for SAT, a clever brute-force algorithm that branches on the assignments to propositional variables and prunes the search space based on falsified clauses [36]. Perhaps somewhat surprisingly, state-of-the-art SAT solvers are still largely based on DPLL [54]. One of the obstacles to QDPLL matching their performance lies in constraints introduced to account for variable dependencies. In DPLL, every unassigned variable can be used for branching, and sophisticated heuristics

for choosing the right variable are among the key features of modern SAT solvers [106]. In QDPLL, on the other hand, a variable can only be assigned if it appears in the leftmost quantifier block (we will assume that formulas are in PCNF by default) that has unassigned variables, a constraint that renders branching heuristics much less effective.

DepQBF uses dependency schemes to relax this constraint: a variable can be assigned if it does not depend, according to the dependency relation computed by the *refined standard dependency scheme* (see Chapter 3) on unassigned variables. This use of dependency schemes is arguably the most distinctive feature of DepQBF and certainly the most important application of dependency schemes, but it has not shown to be sound. To be more precise: the original argument for soundness relies on an assumption that does not hold in general, viz., that it is sufficient for a dependency scheme to allow for quantifier reordering for its use in DepQBF to be sound (see Section 5.1 for a counterexample).

We present a proof-theoretic argument for soundness of using selected dependency schemes in DepQBF. Our point of departure is the correspondence between traces of QDPLL solvers and proofs, or *certificates*, in a calculus known as *Q-resolution* [24, 51, 86]. More specifically, traces can be used to generate *Q-resolution refutations* of false formulas and *Q-term resolution proofs* of true formulas. As a consequence of the way it uses dependency schemes, DepQBF produces certificates that—in general—do not correspond to Q-resolution proofs. To reason about these certificates, we introduce *Q(D)-resolution* and *Q(D)-term resolution*, two families of proof systems that take a dependency scheme D as a parameter. These systems differ from Q-resolution and Q-term resolution in their use of more general versions of $\forall$-reduction and $\exists$-reduction, respectively. Although DepQBF explicitly applies these generalized reduction rules in constraint learning [15, p.7], and Q-resolution certificates are essentially read off from the sequence of learned constraints [86], the fact that this leads to a departure from Q-resolution remained implicit in the original works on DepQBF [15, 82]. Introducing Q(D)-resolution and Q(D)-term resolution allows us to formally state (and answer) the question of whether DepQBF is sound as a question about the soundness of these proof systems.

Dependency schemes were originally introduced for quantifier reordering [102]. In this setting, a dependency relation is interpreted as a set of constraints on the relative order of pairs of variables in the quantifier prefix – if quantifiers (and variables) are rearranged in a way that satisfies these constraints, the resulting formula will have the same truth value as the original formula. We prove that known dependency schemes have this property. As an application of reordering, we combine (tractable) dependency schemes of this kind with an algorithm that computes a quantifier prefix with the fewest quantifier alternations among reorderings compatible with a given dependency relation. This leads to a polynomial time preprocessing routine that may reduce the number of quantifier alternations of a PCNF formula. As mentioned earlier, soundness of reordering is not a sufficient condition for soundness of Q(D)-resolution or Q(D)-term resolution. Finally, we show that Q(D)-resolution can simulate reordering plus Q-resolution and that Q(D)-term resolution can simulate reordering plus Q-term resolution. It follows that every dependency scheme for which Q(D)-resolution and Q(D)-term resolution are sound

can be used for reordering.

Organization of Part I. After reviewing prerequisite notation and definitions in Chapter 2, we introduce common dependency schemes and compare their dependency relations in Chapter 3. In Chapter 4, we prove that there is a polynomial-time algorithm for computing the resolution-path dependency relation for a given PCNF formula. Chapter 5 is concerned with the application of dependency schemes in search-based QSAT solvers. There, we introduce Q(D)-resolution and Q(D)-term resolution and prove that these systems are sound for particular dependency schemes. In Chapter 6, we show that known dependency schemes can be used for quantifier reordering and present an application of reordering to the problem of minimizing the number of quantifier alternations of PCNF formulas. We conclude and give a brief overview of related work in Chapter 7.

Several results proved in the subsequent chapters have been presented at workshops or published, in preliminary form, as part of conference proceedings:

- Tractability of the resolution-path dependency scheme (Chapter 4, Theorem 2) is the main result of a paper published in the proceedings of SAT 2012 [107]. In the same work, we show that the resolution-path dependency scheme is a cumulative dependency scheme – this is entailed by our result stating that the resolution-path dependency scheme is a permutation dependency scheme (Chapter 6, Theorem 9).
- We introduced Q(D)-resolution and showed that the system is sound for the standard dependency scheme in a paper that was presented at the QBF 2013 workshop [109].
- Soundness of Q(D)-resolution for the reflexive resolution-path dependency scheme (Chapter 5, Theorem 5) was originally proved in a paper published at SAT 2014 [111], which also contains the simplified proof of tractability for the resolution-path dependency scheme presented in Chapter 4.
- The proof of soundness of Q(D)-term resolution for the resolution-path dependency scheme (Chapter 5, Theorem 6) is from a paper that was presented at the QBF 2014 workshop [110].

CHAPTER 2

Preliminaries

We will denote the set of natural numbers (that is, positive integers) by $\mathbb{N}^+$. If $n \in \mathbb{N}^+$, we write $[n]$ for the set $\{1, \ldots, n\}$.

Sequences. We write ε for the empty sequence. If $s = s_1, \ldots, s_k$ and $r = r_1, \ldots, r_l$ are sequences then sr is the sequence $s_1, \ldots, s_k, r_1, \ldots, r_l$. If s and r are sequences and there is a sequence t such that $r = st$ then s is a *prefix* of r. If $t \neq \varepsilon$ then s is a *proper prefix* of r. The *length* of a sequence $s_1, \ldots, s_k$ is k.

Permutations. Let $s = s_1, \ldots, s_n$ be a sequence. By a *permutation* of s we mean a sequence $s_{f(1)}, \ldots, s_{f(n)}$, where $f : [n] \to [n]$ is a bijection. A *transposition* of s is a permutation $s_{f(1)}, \ldots, s_{f(n)}$ of s such that $f(i) = f(i+1)$ and $f(i+1) = i$ for some $1 \leq i < n$, and $f(j) = j$ for every $j \neq i$.

Relations. Let R be a binary relation on a set S. For $s \in S$ we write $R(s) = \{\, t \in S : (s,t) \in R \,\}$; for $S' \subseteq S$ we let $R(S') = \cup_{s \in S'} R(s)$. By R^* we denote the reflexive transitive closure of R. That is R^* denotes the smallest (with respect to set inclusion) relation $\mathcal{S}$ such that $(s,s) \in \mathcal{S}$ for every $s \in S$, and such that $(s,u) \in \mathcal{S}$ whenever $(s,t) \in \mathcal{S}$ and $(t,u) \in \mathcal{S}$. Finally we let $\overline{R} = \{\, (y,x) : (x,y) \in R \,\}$ denote the inverse of R.

Graphs. A *directed graph*, or *digraph*, for short, is a pair $G = (V, E)$, where V is a finite set and $E \subseteq V \times V$ is an irreflexive binary relation. The elements of V and E are called the *vertices* and *edges* of G, respectively. An edge (v,w) of G is an *incoming edge* of w and an *outgoing edge* of v. Let $W \subseteq V$. The *subgraph of G induced by W* is $G[W] = (W, E \cap (W \times W))$. A *walk (from v_1 to v_n)* in G is a sequence $w = v_1, \ldots, v_n$ of vertices such that $(v_i, v_{i+1}) \in E$ for each $i \in [n-1]$. The *length* of w is $n-1$. If the v_i are pairwise distinct then w is a *path*. A *longest (shortest) path (from v to u)* in G is a path (from v to u) in G of maximum (minimum) length. A walk $v_1, \ldots, v_n$ is *closed* if $v_1 = v_n$.

A digraph is *acyclic* if it does not contain a closed walk. The sets of *out-neighbors* and *in-neighbors* of a vertex v of G are as defined as $N_G^+(v) = \{ w \in V : (v,w) \in E \}$ and $N_G^-(v) = \{ w \in V : (w,v) \in E \}$, respectively. If E is symmetric, that is, if $E = \overline{E}$, then G is a *graph*. In this case, we may write vw to denote an edge $(v,w) \in E$. If $G = (V,E)$ is a graph and there is a path in G from $v \in V$ to $w \in W$ then v and w are *connected* in G. A *tree* is a graph $T = (V,E)$ such that T is acyclic and any two vertices $v, w \in V$ are connected in T.

Quantified Boolean formulas. A *literal* is a negated or unnegated variable. If x is a variable, we write $\overline{x} = \neg x$ and $\overline{\neg x} = x$, and let $\mathrm{var}(x) = \mathrm{var}(\neg x) = x$. If X is a set of literals, we write $\overline{X}$ for the set $\{ \overline{x} : x \in X \}$. A *clause* is a finite disjunction of literals, and a *term* is a finite conjunction of literals. A *CNF formula* is a finite conjunction of clauses. Whenever convenient, we treat clauses and terms as sets of literals, and a CNF formula as a set of sets of literals. If S is a clause or term, we let $\mathrm{var}(S)$ be the set of variables occurring (negated or unnegated) in S. For a CNF formula φ we let $\mathrm{var}(\varphi) = \bigcup_{C \in \varphi} \mathrm{var}(C)$. A *(quantifier) prefix* is a (possibly empty) sequence $\mathcal{Q} = Q_1 x_1 \dots Q_n x_n$, where $Q_i \in \{\forall, \exists\}$, and the x_i are pairwise distinct variables. The set of variables occurring in the prefix $\mathcal{Q}$ is $\mathrm{var}(\mathcal{Q}) = \{x_1, \dots, x_n\}$. We let $q_{\mathcal{Q}}(x_i) = Q_i$ and define the relation $R_{\mathcal{Q}}$ as $R_{\mathcal{Q}} = \{ (x_i, x_j) : i < j \}$. We extend $R_{\mathcal{Q}}$ to a relation on literals in the obvious way, and drop the subscript from $R_{\mathcal{Q}}$ and $q_{\mathcal{Q}}$ whenever the prefix is clear from the context. A *quantifier block* (of $\mathcal{Q}$) is a maximal (sub)sequence $Q_i x_i \dots Q_k x_k$ such that $1 \leq i < k \leq n$ and $Q_i = Q_j$ for each j with $i \leq j \leq k$. Relative to prefix $\mathcal{Q}$, variable x_i is called *existential* (*universal*) if $Q_i = \exists$ ($Q_i = \forall$). The set of existential (universal) variables occurring in a the prefix $\mathcal{Q}$ is denoted $\mathrm{var}_\exists(\mathcal{Q})$ ($\mathrm{var}_\forall(\mathcal{Q})$). A literal ℓ is existential (universal) relative to a prefix $\mathcal{Q}$ if $\mathrm{var}(\ell)$ is existential (universal) relative to $\mathcal{Q}$. A *PCNF formula* is a pair $\Phi = \mathcal{Q}.\varphi$ consisting of a prefix $\mathcal{Q}$ and a CNF formula φ (called the *matrix* of Φ) such that $\mathrm{var}(\varphi) = \mathrm{var}(\mathcal{Q})$. If $\Phi = \mathcal{Q}.\varphi$ is a PCNF formula, we let $\mathrm{var}(\Phi) = \mathrm{var}(\mathcal{Q})$, $\mathrm{var}_\exists(\Phi) = \mathrm{var}_\exists(\mathcal{Q})$, and $\mathrm{var}_\forall(\Phi) = \mathrm{var}_\forall(\mathcal{Q})$. Moreover, we let $\mathit{lit}(\Phi) = \mathrm{var}(\Phi) \cup \overline{\mathrm{var}(\Phi)}$ and $R_\Phi = R_{\mathcal{Q}}$. We call a clause *tautological* (and a term *contradictory*) if it contains the same variable negated as well as unnegated. We assume that the matrix of a PCNF formula contains only non-tautological clauses. The *size* of a PCNF formula $\Phi = \mathcal{Q}.\varphi$ is defined $|\Phi| = \sum_{C \in \varphi} |C|$.

For a set X of variables, a *truth assignment* (or simply *assignment*) is a mapping $\tau : X \to \{0,1\}$. We extend τ to literals by letting $\tau(\neg x) = 1 - \tau(x)$. Let $\tau : X \to \{0,1\}$ be a truth assignment. We define the application $C[\tau]$ of τ to a clause C as follows. If there is a literal $\ell \in C \cap (X \cup \overline{X})$ such that $\tau(\ell) = 1$ then $C[\tau] = 1$. Otherwise, if $\mathrm{var}(C) \subseteq X$ then $C[\tau] = 0$. Dually, the application $T[\tau]$ of τ to a term T is defined $T[\tau] = 0$ if there is a literal $\ell \in T$ such that $\tau(\ell) = 0$, and $T[\tau] = 1$ if there is no such literal and $\mathrm{var}(T) \subseteq X$. If $\tau : X \to \{0,1\}$ is a truth assignment and $Y \subseteq X$, then the *restriction of τ to Y*, in symbols $\tau|_Y$, is the truth assignment $\tau' : Y \to \{0,1\}$ such that $\tau'(x) = \tau(x)$ for each $x \in Y$. If $\tau : X \to \{0,1\}$ and $\sigma : Y \to \{0,1\}$ are truth assignments and $X \cap Y = \emptyset$, then the *union* of τ and σ is the truth assignment $\tau \cup \sigma : X \cup Y \to \{0,1\}$ such that $\tau \cup \sigma(x) = \tau(x)$ if $x \in X$, and $\tau \cup \sigma(y) = \sigma(y)$ if $y \in Y$.

Models and countermodels. Let $\Phi = \mathcal{Q}.\varphi$ be a PCNF formula. For a variable $x \in \mathrm{var}(\Phi)$, let $L_x^\Phi = \{\, y \in \mathrm{var}(\Phi) : (y,x) \in R_\Phi \text{ and } q(x) \neq q(y) \,\}$. We call a partial function $f_x : 2^{L_x^\Phi} \to \{0,1\}$ a *model function for* x. Given an indexed family $\mathbf{f} = \{f_x\}_{x \in X}$ of model functions for variables in $X \subseteq \mathrm{var}(\Phi)$ and a truth assignment $\tau : Y \to \{0,1\}$ to $Y \subseteq \mathrm{var}(\Phi) \setminus X$, we define the truth assignment $\mathbf{f}(\tau)$ as follows. Whenever $f_x(\tau|_{L_x^\Phi})$ is defined we let $\mathbf{f}(\tau)(x) = f_x(\tau|_{L_x^\Phi})$, and otherwise leave $\mathbf{f}(\tau)(x)$ undefined. An indexed family $\mathbf{f} = \{f_x\}_{x \in \mathrm{var}_\exists(\Phi)}$ of model functions for the existential variables in Φ is called a *model* of Φ if, for every assignment $\tau : \mathrm{var}_\forall(\Phi) \to \{0,1\}$ and every clause $C \in \varphi$, $C[\tau \cup \mathbf{f}(\tau)] = 1$. If f_x is a total function for each $x \in \mathrm{var}_\exists(\Phi)$ then $\mathbf{f}$ is a *complete model* of Φ. An indexed family $\mathbf{g} = \{g_x\}_{x \in \mathrm{var}_\forall(\Phi)}$ of model functions for the universal variables in Φ is a *countermodel* of Φ if, for every assignment $\tau : \mathrm{var}_\exists(\Phi) \to \{0,1\}$, there exists a clause $C \in \varphi$ such that $C[\tau \cup \mathbf{g}(\tau)] = 0$. If g_x is a total function for each $x \in \mathrm{var}_\forall(\Phi)$ then $\mathbf{g}$ is a *complete countermodel* of Φ. Complete models are sometimes referred to as *Skolem-function models*, while complete countermodels are also known as *Herbrand-function countermodels* [8]. A PCNF formula Φ is *true* if it has a model, and *false* if it has a countermodel. Two PCNF formulas Φ and Ψ are *equivalent*, in symbols $\Phi \equiv \Psi$, if they are both true or both false.

Labeled trees. We will represent resolution derivations as labeled, rooted trees. A *labeled, rooted tree* is a triple $\mathcal{T} = (T, r, \lambda)$, where $T = (V, E)$ is a tree, $r \in V$ is a designated vertex, called the *root* of $\mathcal{T}$, and λ is a mapping that associates each vertex or edge $x \in V \cup E$ with a *label* $\lambda(x)$ that is either a literal or a set of literals. Let $\approx$ denote the equivalence relation on labeled, rooted trees defined in the obvious way. We now define operations for manipulating labeled, rooted trees whose result is unique up to $\approx$.

- If C is a set of literals then $\triangle(C)$ is a labeled, rooted tree $\mathcal{T} = (T, r, \lambda)$ such that $T = (\{r\}, \emptyset)$ and $\lambda(r) = C$.

- Let $\mathcal{T}_1 = (T_1, r_1, \lambda_1)$ and $\mathcal{T}_2 = (T_1, r_2, \lambda_2)$ be labeled, rooted trees, and let $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$. We assume without loss of generality that V_1 and V_2 are disjoint. Given a literal ℓ, we define $\mathcal{T}_1 \odot_\ell \mathcal{T}_2$ to be the labeled, rooted tree $\mathcal{T} = (T, r, \lambda)$, where r is a fresh vertex not in $V_1 \cup V_2$,

  $$T = (V_1 \cup V_2 \cup \{r\}, E_1 \cup E_2 \cup \{r_1 r, r_2 r\}),$$

  and the labeling λ is defined as follows. If $v \in V_1 \cup V_2 \cup \{r\}$, then

  $$\lambda(v) = \begin{cases} \lambda_1(v), \text{ if } v \in V_1, \\ \lambda_2(v), \text{ if } v \in V_2, \\ (\lambda(r_1) \setminus \{\ell\}) \cup (\lambda(r_2) \setminus \{\overline{\ell}\}), \text{ otherwise.} \end{cases}$$

If $e \in E_1 \cup E_2 \cup \{r_1 r, r_2 r\}$, then

$$\lambda(e) = \begin{cases} \lambda_1(e), \text{ if } e \in E_1, \\ \lambda_2(e), \text{ if } e \in E_2, \\ \ell, \text{ if } e = r_1 r, \\ \overline{\ell}, \text{ otherwise.} \end{cases}$$

- Let $\mathcal{T} = (T, r, \lambda)$ be a labeled, rooted tree with $T = (V, E)$, and let ℓ be a literal. We define $\mathcal{T}\|_\ell$ to be the labeled, rooted tree $\mathcal{T}' = (T', r', \lambda')$, where $r' \notin V$ is a fresh vertex, $T = (V \cup \{r'\}, E \cup \{rr'\})$, and the labeling λ' is defined as follows. If $v \in V \cup \{r'\}$, then $\lambda'(v) = \lambda(v)$ if $v \in V$ and $\lambda'(v) = \lambda(r) \setminus \{\ell\}$ otherwise. If $e \in E \cup \{rr'\}$ then $\lambda'(e) = \lambda(e)$ if $e \in E$ and $\lambda(e) = \ell$ otherwise.

In order to make statements about the structure of labeled trees without introducing unwanted bound variables, we allow the use of the wild card $\star$ in expressions involving the $\approx$ relation. Below, let $\mathcal{T}$ and $\mathcal{T}'$ be labeled, rooted trees and let ℓ be a literal.

- $\mathcal{T} \approx \mathcal{T}' \odot_\ell \star$ is true if there is a labeled, rooted tree $\mathcal{T}''$ and $\mathcal{T} \approx \mathcal{T}' \odot_\ell \mathcal{T}''$. Symmetrically, $\mathcal{T} \approx \star \odot_\ell \mathcal{T}'$ is true if $\mathcal{T}' \odot_{\overline{\ell}} \star$ is true.
- $\mathcal{T} \approx \star\|_\ell$ is true if there is a labeled, rooted tree $\mathcal{T}''$ and $\mathcal{T} \approx \mathcal{T}''\|_\ell$.

CHAPTER 3

Taxonomy of Dependency Schemes

In this chapter, we provide definitions of dependency schemes used in later chapters and determine which inclusions hold among their dependency relations. Syntactically, dependency schemes are mappings that refine the *trivial dependency scheme*. The trivial dependency scheme essentially computes the linear order induced by a quantifier prefix but ignores "dependencies" between pairs of variables of the same type (i.e., both existential or both universal).[1]

Definition 1. The *trivial dependency scheme* is the mapping D^{trv} that associates each PCNF formula Φ with the relation $D^{\mathrm{trv}}_{\Phi} = \{\, (x, y) \in R_{\Phi} : q(x) \neq q(y) \,\}$ called the *trivial dependency relation* of Φ.

The definition of the trivial dependency scheme can be motivated by the following two (related) facts:

1. The relative order of variables within a quantifier block can be changed without this affecting the truth value of a PCNF formula.

2. A model function for an existential variable takes an assignment to universal variables on its left as an argument, and a model function for a universal variable takes an assignment to existential variables on its left as an argument.

For instance, if Φ is a formula with quantifier prefix $\exists x \forall y \exists z$ then its trivial dependency relation is $D^{\mathrm{trv}}_{\Phi} = \{(x, y), (y, z)\}$. Prima facie, every refinement of the trivial dependency

[1] In the original definition of the trivial dependency scheme, only dependencies between variables in the same *quantifier block* are ignored [102]. The definition given here is due to Biere and Lonsing [15]. Either definition can be used for our purposes, but we believe the latter is a better fit for the intended interpretation(s) of dependency relations.

scheme is a potential dependency scheme. We refer to such mappings as *proto-dependency schemes*.

Definition 2. A *proto-dependency scheme* is a mapping D that associates each PCNF formula Φ with a relation $D_\Phi \subseteq D^{\text{trv}}_\Phi$ called the *dependency relation* of Φ with respect to D. A proto-dependency scheme D is *tractable* if D_Φ can be computed in polynomial time for every PCNF formula Φ.

The trivial dependency relation is entirely determined by the quantifier prefix. A simple syntactic analysis of formulas can often lead to additional information on (potential) variable dependencies. As a somewhat contrived example, consider a formula obtained by prenexing the conjunction of two variable-disjoint PCNF formulas Φ and Ψ. Intuitively, in the resulting formula there is no semantic relation between two variables $x \in \text{var}(\Phi)$ and $y \in \text{var}(\Psi)$. The dependency schemes presented below make this intuition formal by including a pair of variables in the dependency relation only if they are suitably *connected*.

3.1 Standard Dependencies

We begin by giving the definition of the *standard dependency scheme* [102].

Definition 3 (X-path)**.** Let $\Phi = \mathcal{Q}.\varphi$ be a PCNF formula. An *X-path*, $X \subseteq \text{var}(\Phi)$, between two clauses $C, C' \in \varphi$, is a sequence $C_1, \ldots, C_k$ of clauses in φ with $C = C_1$ and $C' = C_k$ such that $\text{var}(C_i) \cap \text{var}(C_{i+1}) \cap X \neq \emptyset$ for all $1 \leq i < n$. Two clauses $C, C' \in \varphi$ are *connected* with respect to $X \subseteq \text{var}(\Phi)$ if there is an X-path between them.

Definition 4 (Dependency pair)**.** Let $\Phi = \mathcal{Q}.\varphi$ be a PCNF formula and let $x, y \in \text{var}(\Phi)$ such that $q(x) \neq q(y)$. An *(x, y)-dependency pair* with respect to $X \subseteq \text{var}(\Phi)$ is a pair $(C_1, C_2) \in \varphi \times \varphi$ of clauses such that (1) C_1 and C_2 are connected with respect to X and (2) $x \in \text{var}(C_1)$ as well as $y \in \text{var}(C_2)$.

Definition 5 (Standard dependency scheme)**.** The *standard dependency scheme* is the mapping D^{std} that associates each PCNF formula Φ with the relation $D^{\text{std}}_\Phi = \{\, (x, y) \in D^{\text{trv}}_\Phi : \Phi \text{ contains an } (x, y)\text{-dependency pair with respect to } R_\Phi(x) \setminus \text{var}_\forall(\Phi) \,\}$.

The standard dependency scheme is most well known for its implementation in the PCNF solver DepQBF [15, 82]. Strictly speaking, DepQBF uses a refinement of the standard dependency scheme defined as follows [82, p.49]. If Φ is a PCNF formula and $x \in \text{var}(\Phi)$, we let $R^\diamond_\Phi(x)$ denote the relation

$$R^\diamond_\Phi(x) = \{\, y : \exists z \in R_\Phi(x) \text{ such that } q_\Phi(z) \neq q_\Phi(x) \text{ and } y \in R_\Phi(z) \text{ or } y = z \,\}.$$

That is, $R^\diamond_\Phi(x)$ is the set of variables to the right of x in the prefix of Φ but not in the same quantifier block as x.

Definition 6 (Refined standard dependency scheme)**.** The *refined standard dependency scheme* is the mapping D^{rst} that associates each PCNF formula Φ with the relation $D^{\mathrm{rst}}_\Phi = \{\, (x,y) \in D^{\mathrm{trv}}_\Phi : \Phi \text{ contains an } (x,y)\text{-dependency pair with respect to } R^{\diamond}_\Phi(x) \setminus \mathrm{var}_\forall(\Phi) \,\}$.

Observe that the refined standard dependency relation coincides with the standard dependency relation for pairs (x,y) where x is universal and y is existential. This fact will be useful in Chapter 5, so we formally state it below.

Lemma 1. *Let Φ be a PCNF formula and let $(x,y) \in \mathrm{var}_\forall(\Phi) \times \mathrm{var}_\exists(\Phi)$. Then $(x,y) \in D^{\mathrm{std}}_\Phi$ if and only if $(x,y) \in D^{\mathrm{rst}}_\Phi$.*

Proof. If $(x,y) \notin R_\Phi$ then $(x,y) \notin D^{\mathrm{std}}_\Phi$ and $(x,y) \notin D^{\mathrm{rst}}_\Phi$. Otherwise, $R^{\diamond}_\Phi(x) \setminus \mathrm{var}_\forall(\Phi) = R_\Phi(x) \setminus \mathrm{var}_\forall(\Phi)$ and the lemma follows. □

The relation D^{rst}_Φ can be computed and represented very efficiently [82, Ch.4]. Here, we only show that both D^{std}_Φ and D^{rst}_Φ can be computed in polynomial time using the *primal graph* of a PCNF formula Φ.

Definition 7 (Primal graph)**.** Let $\Phi = \mathcal{Q}.\varphi$ be a PCNF formula. The *primal graph* of Φ is the graph $G = (V,E)$, where $V = \mathrm{var}(\Phi)$ and $E = \{\, xy : \text{ there is a clause } C \in \varphi \text{ such that } x,y \in \mathrm{var}(C) \,\}$.

Lemma 2. *Let Φ be a PCNF formula with primal graph G, let $x,y \in \mathrm{var}(\Phi)$, and let $X \subseteq \mathrm{var}(\Phi)$. The following two statements are equivalent:*

1. *There are clauses $C, C' \in \varphi$ such that $x \in \mathrm{var}(C)$, $y \in \mathrm{var}(C')$, and such that C and C' are connected with respect to X.*
2. *There is a path connecting x and y in $G[X \cup \{x,y\}]$.*

Proof. $(1 \Rightarrow 2)$ Let $C_1, \ldots, C_k$ be an X-path such that $x \in \mathrm{var}(C_1)$ and $y \in \mathrm{var}(C_k)$. We prove that there is a walk $x_1, \ldots, x_k$ in $G[X \cup \{x,y\}]$ such that $x_1 = x$ and $x_k = y$ by induction on k. If $k = 1$ then $x,y \in \mathrm{var}(C_1)$ and $xy \in E(G)$ by definition of the primal graph, so x,y is a walk in $G[X \cup \{x,y\}]$. Let $k > 1$ and suppose the statement holds for X-paths of length strictly less than k. Let $z \in \mathrm{var}(C_1) \cap \mathrm{var}(C_2) \cap X$. The sequence $C_2, \ldots, C_k$ is an X-path such that $z \in \mathrm{var}(C_2)$ and $y \in \mathrm{var}(C_k)$. By induction hypothesis, there is a walk $x_2, \ldots, x_k$ in $G[X \cup \{z,y\}] = G[X \cup \{y\}]$ such that $x_2 = z$ and $x_k = y$. Because $x, z \in \mathrm{var}(C_1)$ there is an edge $xz \in E(G)$, so the sequence $x_1, \ldots, x_k$, where $x_1 = x$, is a walk in $G[X \cup \{x,y\}]$. Clearly, there is a walk from x to y in $G[X \cup \{x,y\}]$ only if there is a path connecting x and y.

$(2 \Rightarrow 1)$ Let $x_1, \ldots, x_k$ be a path in $G[X \cup \{x,y\}]$ such that $x_1 = x$ and $x_k = y$. We prove by induction on k that there are clauses $C, C' \in \varphi$ such that $x \in \mathrm{var}(C)$, $y \in \mathrm{var}(C')$, and such that C and C' are connected with respect to X. If $k = 1$ then there must be a clause $C \in \varphi$ such that $x,y \in \mathrm{var}(C)$ and we simply let $C' = C$. Let $k > 1$ and suppose the statement holds for paths of length strictly less than k. Because x_1x_2 is an edge of G there must be a clause $C \in \varphi$ such that $x_1, x_2 \in \mathrm{var}(C)$. If $x_2 = y$ we again set $C' = C$. Otherwise, let $x_2 \neq y$. Since $x_1, \ldots, x_k$ is a path in $G[X \cup \{x,y\}]$

we must have $x_1 \notin \{x_2, \ldots, x_k\}$ and the sequence $x_2, \ldots, x_k$ is a path in $G[X \cup \{y\}]$. In particular, $x_2, \ldots, x_k$ is a path in $G[X \cup \{x_2, y\}]$, so we can apply the induction hypothesis to conclude that there are clauses $C', C'' \in \varphi$ such that $x_2 \in \mathrm{var}(C'')$, $y \in \mathrm{var}(C')$, and such that C'' and C' are connected with respect to X. Let $C_1, \ldots, C_l$ be an X-path such that $C_1 = C''$ and $C_l = C'$. Since $\{x_2, \ldots, x_k\} \subseteq X \cup \{y\}$ and $x_2 \neq y$ we must have $x_2 \in X \cap \mathrm{var}(C) \cap \mathrm{var}(C'')$, so the sequence $C, C_1, \ldots, C_l$ is an X-path between C and C'. □

From this, tractability follows easily.

Theorem 1. *The (refined) standard dependency scheme D^{std} (D^{rst}) is tractable: for each PCNF formula Φ the relation D^{std}_Φ (D^{rst}_Φ) can be computed in polynomial time.*

Proof. We describe a polynomial time algorithm for computing D^{std} (or D^{rst}). Given a PCNF formula Φ, the algorithm first constructs the primal graph G of Φ. Then it determines for each pair $(x, y) \in D^{\mathrm{trv}}_\Phi$ whether there is a path from x to y in $G[X \cup \{x, y\}]$, where $X = R_\Phi(x) \setminus \mathrm{var}_\forall(\Phi)$ for D^{std} and $X = R^\diamond_\Phi(x) \setminus \mathrm{var}_\forall(\Phi)$ for D^{rst}. By Lemma 2, this amounts to checking whether Φ contains an (x, y)-dependency pair with respect to X, and thus to deciding whether $(x, y) \in D^{\mathrm{std}}_\Phi$ (respectively $(x, y) \in D^{\mathrm{rst}}_\Phi$). □

3.2 Resolution-Path Dependencies

Next, we define the *resolution-path dependency scheme* [120, 107], as well as the *reflexive-resolution path dependency scheme* [111]. These dependency schemes rely on a notion of connectivity through so-called *resolution paths*, defined as follows.

Definition 8 (Resolution Path)**.** Let $\Phi = \mathcal{Q}.\varphi$ be a PCNF formula and let $X \subseteq \mathrm{var}_\exists(\Phi)$. A *resolution path* (from ℓ_1 to ℓ_{2k}) via X (in Φ) is a sequence $\ell_1, \ldots, \ell_{2k}$ of literals satisfying the following properties:

A) For all $i \in [k]$, there is a $C_i \in \varphi$ such that $\ell_{2i-1}, \ell_{2i} \in C_i$.

B) For all $i \in [k]$, $\mathrm{var}(\ell_{2i-1}) \neq \mathrm{var}(\ell_{2i})$.

C) For all $i \in [k-1]$, $\{\ell_{2i}, \ell_{2i+1}\} \subseteq \mathit{lit}(X)$.

D) For all $i \in [k-1]$, $\overline{\ell_{2i}} = \ell_{2i+1}$.

If $p = \ell_1, \ldots, \ell_{2k}$ is a resolution path in Φ via X, we say that ℓ_1 and ℓ_{2k} are *connected in* Φ (with respect to X). For every $i \in \{1, \ldots, k\}$ we say that p *goes through* $\mathrm{var}(\ell_{2i})$.

Resolution paths are intimately related to resolution (see Section 5.1): whenever a clause containing two literals can be derived from a formula by resolution, there has to be a resolution path connecting these literals. For example, consider the PCNF formula

$$\Phi = \forall x \exists y \exists z \exists e.(x \vee y) \wedge (\neg y \vee z) \wedge (\neg z \vee e).$$

By resolving the first two clauses (on y), we obtain the clause $(x \vee z)$, which in turn can be resolved with the third clause (on z) to derive the clause $(x \vee e)$. The sequence $x, y, \neg y, z, \neg z, e$ is the corresponding resolution path.

Alternatively, resolution paths can be defined so as to include the intermediate clauses C_i mentioned in Condition A (cf. [107]), but we find the above definition more convenient to work with. Connectivity through resolution paths is a stronger notion than connectivity through X-paths, as witnessed by the following result.

Lemma 3. *Let $\Phi = \mathcal{Q}.\varphi$ be a PCNF formula, let $\ell, \ell' \in lit(\Phi)$, and let $X \subseteq \mathrm{var}(\Phi)$. If there is a resolution path from ℓ to ℓ' in Φ via X then there are clauses $C, C' \in \varphi$ such that $\ell \in C$, $\ell' \in C'$, and such that C and C' are connected in Φ with respect to X.*

Proof. Let $\ell_1, \ldots, \ell_{2k}$ be a resolution path from ℓ to ℓ' via X. By Condition A of Definition 8, there is a sequence $C_1, \ldots, C_k$ of clauses such that $\ell_{2i-1}, \ell_{2i} \in C_i$, for each $i \in [k]$. In particular, $\ell = \ell_1 \in C_1$ and $\ell' = \ell_{2k} \in C_k$. Definition 8 also requires that $\mathrm{var}(\ell_{2i}) = \mathrm{var}(\ell_{2i+1}) \in X$, for $i \in [k-1]$ (Conditions C and D). Setting $x_i = \mathrm{var}(\ell_{2i})$, we get $x_i \in \mathrm{var}(C_i) \cap \mathrm{var}(C_{i+1}) \cap X$, for $i \in [k-1]$. That is, the sequence $C_1, \ldots, C_k$ is an X-path of Φ. Choosing $C = C_1$ and $C' = C_k$, we obtain the desired clauses. □

We will sometimes implicitly use the fact that a resolution path taken in inverse order is a resolution path as well, which is easy to verify. Moreover, two resolution paths can be concatenated if their first and last literals, respectively, are different polarities of the same variable, as stated in the following lemma.

Lemma 4. *Let Φ be a PCNF formula. Let $p = \ell_1, \ldots, \ell_{2i}, \ell_{2i+1}, \ldots, \ell_{2k}$ be a sequence of literals, let $p' = \ell_1, \ldots, \ell_{2i}$, and let $p'' = \ell_{2i+1}, \ldots, \ell_{2k}$. The following statements are equivalent:*

1. *p is a resolution path in Φ.*
2. *p' and p'' are resolution paths in Φ and $\overline{\ell_{2i}} = \ell_{2i+1}$.*

Resolution-path dependency pairs are induced by a pair of resolution paths that connects two literals and their negations, as follows.

Definition 9 (Resolution-path dependency pair)**.** Let Φ be a PCNF formula and $x, y \in \mathrm{var}(\Phi)$. We say $\{x, y\}$ is a *resolution-path dependency pair* of Φ with respect to $X \subseteq \mathrm{var}(\Phi)$ if at least one of the following conditions holds:

- x and y, as well as $\neg x$ and $\neg y$, are connected in Φ with respect to X.
- x and $\neg y$, as well as $\neg x$ and y, are connected in Φ with respect to X.

The pair $\{x, y\}$ is an *irreflexive resolution-path dependency pair* of Φ with respect to $X \subseteq \mathrm{var}_\exists(\Phi)$ if $\{x, y\}$ is a resolution-path dependency pair with respect to $X \setminus \{x, y\}$.

Definition 10 (Resolution-path dependency scheme)**.** The *resolution-path dependency scheme* is the mapping D^{res} that assigns to each PCNF formula the relation D^{res}_Φ defined as $D^{\mathrm{res}}_\Phi = \{\, (x, y) \in D^{\mathrm{trv}}_\Phi : \{x, y\}$ is an irreflexive resolution-path dependency pair of Φ with respect to $R_\Phi(x) \setminus \mathrm{var}_\forall(\Phi) \,\}$.

As we will see in Chapter 5, the resolution-path dependency scheme is not suited for use in QDPLL solvers, in the sense that a proof system named *$Q(D)$-resolution* is unsound when parameterized by D^{res}. This proof system turns out to be sound when parameterized by the following, "reflexive" version, however [111].

Definition 11 (Reflexive resolution-path dependency scheme)**.** The *reflexive resolution-path dependency scheme* is the mapping D^{rrs} that assigns to each PCNF formula the relation D^{rrs}_Φ defined as $D^{\mathrm{rrs}}_\Phi = \{\, (x, y) \in D^{\mathrm{trv}}_\Phi : \{x, y\}$ is a resolution-path dependency pair of Φ with respect to $R_\Phi(x) \setminus \mathrm{var}_\forall(\Phi) \,\}$.

The use of irreflexive resolution-path dependency pairs in Definition 10 has the effect that resolution paths inducing a dependency of an existential variable y on a universal variable x cannot go through y. As a consequence, the resolution-path dependency scheme does not include some of the dependencies detected by its reflexive variant. Can we introduce a variant of the standard dependency scheme in a similar manner? The following claim shows that an analogue distinction between irreflexive and ordinary dependency pairs collapses for connectivity defined in terms of X-paths.

Claim 1. *Let Φ be a PCNF formula, let $x, y \in \mathrm{var}(\Phi)$, and let $X \subseteq \mathrm{var}(\Phi)$. The following statements are equivalent:*

1. *Φ contains an (x, y)-dependency pair with respect to X.*

2. *Φ contains an (x, y)-dependency pair with respect to $X \setminus \{x, y\}$.*

Proof. The second statement trivially entails the first, so it suffices to show that the first statement entails the second. Let (C, C') be an (x, y)-dependency pair of Φ with respect to X, and let $C_1, \ldots, C_k$ be an X-path such that $C_1 = C$ and $C_k = C'$. Let i, j be indices such that $1 \leq i \leq j \leq k$, $x \in \mathrm{var}(C_i)$, $y \in \mathrm{var}(C_j)$, and such that the difference $j - i$ is minimized. Such indices must exist since $x \in \mathrm{var}(C) = \mathrm{var}(C_1)$ and $y \in \mathrm{var}(C') = \mathrm{var}(C_k)$. Then $C_i, \ldots, C_j$ is an $X \setminus \{x, y\}$-path, and (C_i, C_j) is an (x, y)-dependency pair of Φ with respect to $X \setminus \{x, y\}$. □

The question of whether resolution paths can be computed efficiently was stated as an open problem by Van Gelder [120]. In Chapter 4 we provide an answer to this question by showing that both D^{res} and D^{rrs} are tractable.

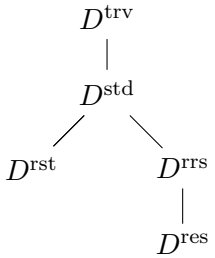

Figure 3.1: Hasse diagram for proto-dependency schemes with respect to the partial order $\subsetneq$ of Definition 12.

3.3 Inclusions among Dependency Relations

For the applications of proto-dependency schemes discussed in Chapters 5 and 6, the presence a pair (x, y) in a dependency relation represents a constraint – other things being equal, we prefer dependency schemes that compute smaller dependency relations. With this in mind, we define a pointwise partial order $\subsetneq$ that allows us to compare proto-dependency schemes.

Definition 12. We define relations $\subseteq$ and $\subsetneq$ on proto-dependency schemes as follows. Let D^1 and D^2 be proto-dependency schemes. Then $D^1 \subseteq D^2$ holds if $D^1_\Phi \subseteq D^2_\Phi$ for every PCNF formula Φ. If $D^1 \subseteq D^2$ we say D^1 *is at least as general as* D^2. We define $D^1 \subsetneq D^2$ to be true if (a) $D^1 \subseteq D^2$ and (b) there is a PCNF formula Φ such that $D^1_\Phi \subsetneq D^2_\Phi$. If that is the case we say that D^1 *is more general than* D^2. If neither $D^1 \subseteq D^2$ nor $D^2 \subseteq D^1$ then D^1 and D^2 are *incomparable*.

If D^1 is more general than D^2 while retaining the properties required for a particular application, then the pairs in $D^2_\Phi \setminus D^1_\Phi$ for a given formula Φ can be thought of as *spurious dependencies*. With respect to the partial order $\subsetneq$, the proto-dependency schemes introduced above are ordered as follows.

Proposition 1.

1. *The standard dependency scheme* D^{std} *is more general than the trivial dependency scheme* D^{trv}.

2. *The refined standard dependency scheme* D^{rst} *is more general than the standard dependency scheme* D^{std}.

3. *The reflexive resolution-path dependency scheme* D^{rrs} *is more general than the standard dependency scheme* D^{std}.

4. *The resolution-path dependency scheme* D^{res} *is more general than the reflexive resolution-path dependency scheme* D^{rrs}.

5. *The refined standard dependency scheme D^{rst} and the (reflexive) resolution-path dependency scheme D^{res} (D^{rrs}) are incomparable.*

Proof. We prove each statement separately:

1. The inclusion $D^{\mathrm{std}}_\Phi \subseteq D^{\mathrm{trv}}_\Phi$ holds trivially, so $D^{\mathrm{std}} \subseteq D^{\mathrm{trv}}$. Consider the formula $\Phi = \forall x \exists y.(x) \wedge (y)$. Then $(x, y) \in D^{\mathrm{trv}}_\Phi$ but $(x, y) \notin D^{\mathrm{std}}_\Phi$.

2. For every formula Φ, variables $x, y \in \mathrm{var}(\Phi)$, and sets $X \subseteq X' \subseteq \mathrm{var}(\Phi)$, an (x, y)-dependency pair of Φ with respect to X is an (x, y)-dependency pair of Φ with respect to X'. It follows that $D^{\mathrm{rst}} \subseteq D^{\mathrm{std}}$. Let $\Phi = \exists x \exists y \forall z.(x \vee y) \wedge (y \vee z)$. Then $(x, z) \in D^{\mathrm{std}}_\Phi$ but $(x, z) \notin D^{\mathrm{rst}}_\Phi$.

3. Let Φ be a PCNF formula Φ and let $(x, y) \in D^{\mathrm{rrs}}_\Phi$. Assume without loss of generality that x and y, as well as $\neg x$ and $\neg y$, are connected in Φ with respect to $R_\Phi(x) \setminus \mathrm{var}_\forall(\Phi)$. By Lemma 3, the formula Φ has an (x, y)-dependency pair with respect to $R_\Phi(x) \setminus \mathrm{var}_\forall(\Phi)$ and $(x, y) \in D^{\mathrm{std}}_\Phi$. We conclude that $D^{\mathrm{rrs}} \subseteq D^{\mathrm{std}}$. Let $\Phi = \forall x \exists y.(x \vee y)$. Then $(x, y) \in D^{\mathrm{std}}_\Phi$ but $(x, y) \notin D^{\mathrm{rrs}}_\Phi$.

4. An irreflexive resolution-path dependency pair is a resolution-path dependency pair with respect to the same set of variables, so $D^{\mathrm{res}} \subseteq D^{\mathrm{rrs}}$. Consider the PCNF formula
$$\Phi = \forall x \exists y \exists z.(x \vee y) \wedge (\neg x \vee y) \wedge (\neg y \vee z) \wedge (\neg z \vee \neg y).$$
Verify that x and y, as well as $\neg x$ and $\neg y$, are connected via $R_\Phi(x) \setminus \mathrm{var}_\forall(\Phi)$, so that $(x, y) \in D^{\mathrm{rrs}}_\Phi$. On the other hand, every resolution path from x or $\neg x$ to $\neg y$ goes through y, so $\{x, y\}$ is not a resolution-path dependency pair with respect to $R_\Phi(x) \setminus (\mathrm{var}_\forall(\Phi) \cup \{x, y\})$ and thus $(x, y) \notin D^{\mathrm{res}}_\Phi$.

5. Consider again the formula $\Phi = \forall x \exists y.(x \vee y)$. We have $(x, y) \in D^{\mathrm{rst}}_\Phi$ but $(x, y) \notin D^{\mathrm{rrs}}_\Phi \supseteq D^{\mathrm{res}}_\Phi$, so $D^{\mathrm{rst}} \not\subseteq D^{\mathrm{res}}$ and $D^{\mathrm{rst}} \not\subseteq D^{\mathrm{rrs}}$. On the other hand, take the formula
$$\Psi = \exists x \exists y \forall z.(x \vee y) \wedge (\neg y \vee z) \wedge (\neg x \vee \neg y) \wedge (y \vee \neg z).$$
It is straightforward to verify that $(x, y) \in D^{\mathrm{res}}_\Phi \subseteq D^{\mathrm{rrs}}_\Phi$ but $(x, y) \notin D^{\mathrm{rst}}_\Phi$, so $D^{\mathrm{res}} \not\subseteq D^{\mathrm{rst}}$ and $D^{\mathrm{rrs}} \not\subseteq D^{\mathrm{rst}}$.

□

The Hasse diagram for the dependency schemes introduced in the previous section, partially ordered by the relation $\subsetneq$, is shown in Figure 3.1.

3.4 Further Dependency Schemes

This section covers additional dependency schemes studied in the literature that are intermediate in generality between the standard and resolution-path dependency schemes. Like the resolution-path dependency scheme, they test for connections between different

polarities of variables, while relying on the notion of connectivity through X-paths, rather than resolution paths.

The first of these dependency schemes is the *triangle dependency scheme*, which is based on a generalization of dependency pairs to *dependency triples* [102].

Definition 13 (Dependency triple)**.** Let $\Phi = \mathcal{Q}.\varphi$ be a PCNF formula and let $x, y \in \mathrm{var}(\Phi)$ such that $q(x) \neq q(y)$. An (x, y)*-dependency triple* with respect to $X \subseteq \mathrm{var}(\Phi)$ is a triple $(C_1, C_2, C_3) \in \varphi \times \varphi \times \varphi$ such that

1. If $q(x) = \forall$ and $q(y) = \exists$, then
   - C_1 and C_2 as well as C_1 and C_3 are connected with respect to $X \cup \{x\}$
   - $x \in \mathrm{var}(C_1)$, $y \in C_2$, and $\neg y \in C_3$
2. If $q(x) = \exists$ and $q(y) = \forall$, then
   - C_1 and C_2 as well as C_1 and C_3 are connected with respect to $X \cup \{y\}$
   - $y \in \mathrm{var}(C_1)$, $x \in C_2$, and $\neg x \in C_3$

Definition 14 (Triangle dependency scheme)**.** The *triangle dependency scheme* is the mapping $D^{\triangle}$ that assigns to each PCNF formula Φ the relation $D^{\triangle}_{\Phi} = \{\, (x, y) \in R_{\Phi} : \Phi$ contains an (x, y)-dependency triple with respect to $R_{\Phi}(x) \setminus (\mathrm{var}_{\forall}(\Phi) \cup \{y\}) \,\}$.

Using a generalization of Lemma 2, it is fairly straightforward to show that the triangle dependency scheme is tractable. By means of a slightly more involved argument, one can even prove the following result [102, Proposition 4].

Proposition 2. *The triangle dependency scheme is tractable. Given a PCNF formula* Φ *of size* n *and* $x \in \mathrm{var}(\Phi)$*, we can compute* $D^{\triangle}_{\Phi}(x)$ *in time* $O(n)$*.*

Note the asymmetry between existential and universal variables in the definition of dependency triples: the existential variable has to appear both positively and negatively, while this is not required for the universal variable. By contrast, resolution-path dependency pairs are symmetric with respect to existential and universal variables. *Strict standard dependencies* relax resolution-path dependencies in a dual way, by demanding triples of clauses in which the universal variable occurs both positively and negatively, while the existential variable may appear in only one polarity. Finally, *quadrangle dependencies* relax resolution-path dependencies by only demanding connectivity through X-paths, rather than resolution paths. Both strict standard and quadrangle dependencies were introduced along with resolution-path dependencies [120]. Though they naturally lead to dependency *schemes*, they were not introduced as such since *dependency schemes* require certain reordering operations on a formula's quantifier prefix to be truth-value preserving (see Chapter 6), and reordering was not the intended application of dependency relations in [120].

Resolution-path dependencies, as well as quadrangle and strict standard dependencies, were originally defined in terms of so-called *depth-limited clause-literal graphs* [120, Definition 5.1].

Definition 15 (Quantifier depth)**.** Let $\Phi = Q_1 x_1 \dots Q_n x_n.\varphi$ be a PCNF formula. The *quantifier depth* of a variable x_i with respect to Φ, in symbols $qdepth_\Phi(x_i)$ where $i \in [n]$, is inductively defined as follows:

$$qdepth_\Phi(x_i) = \begin{cases} 1 & \text{if i = 1;} \\ qdepth_\Phi(x_{i-1}) & \text{if } i > 1 \text{ and } Q_{i-1} = Q_i, \\ qdepth_\Phi(x_{i-1}) + 1 & \text{if } i > 1 \text{ and } Q_{i-1} \neq Q_i. \end{cases}$$

As usual, we drop the subscript from $qdepth_\Phi(x)$ and write $qdepth(x)$ if the formula Φ is clear from the context.

Definition 16 (Depth-limited clause-literal graph)**.** Let $\Phi = \mathcal{Q}.\varphi$ be a PCNF formula and let $k = \max\{\, qdepth(x) : x \in \mathrm{var}(\Phi) \,\}$. For each $i \in [k-1]$, the *depth-limited clause-literal graph* of Φ at depth i is the graph $G = (U \cup V \cup W, E)$, where

$$\begin{aligned} U &= \{\, C \in \varphi : \exists x \in \mathrm{var}(C) \text{ such that } qdepth(x) > i \text{ and } q(x) = \exists \,\}, \\ V &= \{\, x \in \mathrm{var}_\exists(\Phi) : qdepth(x) > i \text{ and there is a } C \in U \text{ such that } x \in C \,\}, \\ W &= \{\, \neg x \in \overline{\mathrm{var}_\exists(\Phi)} : qdepth(x) > i \text{ and there is a } C \in U \text{ such that } \neg x \in C \,\}, \\ E &= \{\, \neg x x : x \in V, \neg x \in W \,\} \cup \{\, \ell C : \ell \in C, \ell \in V \cup W, C \in U \,\}. \end{aligned}$$

Strict standard, quadrangle, and resolution-path dependencies are then defined as follows [120, Definition 5.2].

Definition 17. Let Φ be a PCNF formula, let $x \in \mathrm{var}_\forall(\Phi)$, and let $y \in \mathrm{var}_\exists(\Phi)$, such that $qdepth(x) = i$ and $qdepth(y) = i + 1$. Let G denote the depth-limited clause-literal graph of Φ at depth i.

1. The pair (x, y) is a *strict standard dependency* of Φ if there are literals ℓ_x, ℓ_y, and ℓ'_y with $\mathrm{var}(\ell_x) = x$ and $\mathrm{var}(\ell_y) = \mathrm{var}(\ell'_y) = y$, as well as clauses $C_1, C_2, C_3, C_4 \in V(G)$ with $\ell_x \in C_1, \overline{\ell_x} \in C_2, \ell_y \in C_3, \ell'_y \in C_4$, such that there are walks in G from C_1 to C_3 and from C_2 to C_4.

2. The pair (x, y) is a *quadrangle dependency* of Φ if there are literals ℓ_x, ℓ_y with $\mathrm{var}(\ell_x) = x$ and $\mathrm{var}(\ell_y) = y$, as well as clauses $C_1, C_2, C_3, C_4 \in V(G)$ with $\ell_x \in C_1, \overline{\ell_x} \in C_2, \ell_y \in C_3, \overline{\ell_y} \in C_4$, such that there is walk in G from C_1 to C_3 that does not go through $\overline{\ell_y}$, as well as a walk from C_2 to C_4 that does not go through ℓ_y.

3. The pair (x, y) is a *resolution-path dependency* of Φ if there are literals ℓ_x, ℓ_y with $\mathrm{var}(\ell_x) = x$ and $\mathrm{var}(\ell_y) = y$, as well as clauses $C_1, C_2, C_3, C_4 \in V(G)$ with $\ell_x \in C_1, \overline{\ell_x} \in C_2, \ell_y \in C_3, \overline{\ell_y} \in C_4$, such that there is a walk in G from C_1 to C_3 that does not go through $\overline{\ell_y}$, as well as a walk from C_2 to C_4 that does not go through ℓ_y, subject to the following additional conditions:

   a) If the walk arrives at a literal ℓ from a clause, the next vertex on the walk must be $\overline{\ell}$.

b) If the walk arrives at a literal $\overline{\ell}$ from ℓ, the next vertex on the walk must be a clause.

c) If the walk arrives at a clause C from ℓ, the next vertex on the walk (if it exists) must be a literal different from ℓ.[2]

It is immediate from this definition that these three notions are in increasing order of generality, as stated in the following claim.

Claim 2. *Let Φ be a PCNF formula and let $x, y \in \mathrm{var}(\Phi)$. Then the implications $3. \Rightarrow 2. \Rightarrow 1.$ hold among the following statements.*

1. *The pair (x, y) is a strict standard dependency of Φ.*
2. *The pair (x, y) is a quadrangle dependency of Φ.*
3. *The pair (x, y) is resolution-path dependency of Φ.*

We now prove that the definition of the resolution-path dependency *scheme* is conservative with respect to resolution-path dependencies, in the following sense.

Proposition 3. *Let $\Phi = \mathcal{Q}.\varphi$ be PCNF formula and let $x \in \mathrm{var}_\forall(\Phi)$ and $y \in \mathrm{var}_\exists(\Phi)$ such that $\mathrm{qdepth}(x) = j$ and $\mathrm{qdepth}(y) = j+1$. Then (x, y) is a resolution-path dependency of Φ if and only if $(x, y) \in D^{\mathrm{res}}_\Phi$.*

Proof. Let G denote the depth-limited clause-literal graph of Φ at depth j. Suppose (x, y) is a resolution-path dependency of Φ. We will show that $\{x, y\}$ is an irreflexive resolution-path dependency pair of Φ with respect to $R_\Phi(x) \setminus \mathrm{var}_\forall(\Phi)$. Let ℓ_x, ℓ_y be literals and let $C_1, C_2, C_3, C_4 \in V(G)$ be clauses satisfying the conditions specified in Definition 17, and let p and p' be the corresponding paths in G between, respectively, C_1 and C_3, and C_2 and C_4. Then

$$p = C'_1, \ell_1, \overline{\ell_1}, C'_2, \ell_2, \overline{\ell_2}, \ldots, C'_{k-1}, \ell_{k-1}, \overline{\ell_{k-1}}, C'_k,$$

where $C'_i \in \varphi$ for each $i \in [k]$, and $\ell_i \in \mathit{lit}(\Phi)$ for each $i \in [k-1]$. We claim that the sequence $s = \ell_x, \ell_1, \overline{\ell_1}, \ell_2, \overline{\ell_2}, \ldots, \ell_{k-1}, \overline{\ell_{k-1}}, \ell_y$ is a resolution-path in Φ via $R_\Phi(x) \setminus (\mathrm{var}_\forall(\Phi) \cup \{y\})$. Since $\ell_x \in C'_1 = C_1$ and $\ell_y \in C'_k = C_3$, and G contains an edge ℓC only if $\ell \in C$, Condition A of Definition 8 is satisfied. It follows from Condition 3.c of Definition 17 that Condition B of Definition 8 is satisfied as well. By definition of the clause-literal graph at depth j, each literal ℓ_i for $i \in [k-1]$ satisfies $\mathrm{var}(\ell_i) \in R(x) \setminus \mathrm{var}_\forall(\Phi)$. If ℓ_y would occur in p then $\overline{\ell_y}$ would have to occur as well by Condition 3.a of Definition 17, but this literal is avoided by the path p. So $\mathrm{var}(\ell_i) \in R(x) \setminus (\mathrm{var}_\forall(\Phi) \cup \{y\})$ and Condition C of Definition 8 (for $X = R(x) \setminus (\mathrm{var}_\forall(\Phi) \cup \{y\})$) is met. Finally, Condition D is satisfied by choice of p. So s is indeed a resolution path in Φ via $R(x) \setminus (\mathrm{var}_\forall(\Phi) \cup \{y\})$. A parallel argument shows that $\overline{\ell_x}$ and $\overline{\ell_y}$ are connected in Φ with respect to $R(x) \setminus (\mathrm{var}_\forall(\Phi) \cup \{y\})$, so

[2] This restriction is not mentioned in Definition 5.2 of [120], but it appears in Definition 4.1 of the same paper.

that $\{x, y\}$ is a resolution-path dependency pair of Φ with respect to $R(x) \setminus (\mathrm{var}_\forall(\Phi) \cup \{y\})$. Since $x \notin R(x)$, the pair $\{x, y\}$ is even an irreflexive resolution-path dependency pair of Φ with respect to $R(x) \setminus \mathrm{var}_\forall(\Phi)$.

For the converse, let $\{x, y\}$ be an irreflexive resolution-path dependency pair of Φ with respect to $R(x) \setminus \mathrm{var}_\forall(\Phi)$. We will show that (x, y) is a resolution-path dependency of Φ. Assume, without loss of generality, that x and y, as well as $\neg x$ and $\neg y$, are connected in Φ with respect to $R(x) \setminus (\mathrm{var}_\forall(\Phi) \cup \{x, y\})$. Let $p = \ell_1, \ldots, \ell_{2k}$ be a resolution path from x to y via $R(x) \setminus (\mathrm{var}_\forall(\Phi) \cup \{x, y\})$. Let $C_1, \ldots, C_k$ be a sequence of clauses such that $\ell_{2i-1}, \ell_{2i} \in C_i$, for each $i \in [k]$ – such a sequence must exist by Condition A of Definition 8. We claim that the sequence

$$s = C_1, \ell_2, \ell_3, C_2, \ell_4, \ell_5, \ldots, C_{i-1}, \ell_{2k-2}, \ell_{2k-1}, C_k$$

is a walk satisfying the conditions stated in Definition 17. We first verify that every literal or clause appearing in s is a vertex of G. Since $\ell_{2k} = y$, for each $i \in [k]$ the clause C_i contains an existential literal ℓ_{2i} such that $\mathrm{var}(\ell_{2i}) \in R(x)$. Because by assumption x is a universal variable, the corresponding existential variable $z_i = \mathrm{var}(\ell_{2i})$ satisfies $qdepth(z_i) > j$, so each clause C_i is a vertex of G. For $i \in [k-1]$, the literals ℓ_{2i} and ℓ_{2i+1} are in $lit(X)$ for $X = R(x) \setminus (\mathrm{var}_\forall(\Phi) \cup \{x, y\})$ by Condition C of Definition 8, so they are existential literals whose associated variables have quantifier depth at least $j + 1$, and they are contained in the clause C_i. It follows that ℓ_{2i} and ℓ_{2i+1} are vertices of G for each $i \in [k-1]$. It also follows from the literals ℓ_{2i} and ℓ_{2i+1} being in $lit(X)$ for $X = R(x) \setminus (\mathrm{var}_\forall(\Phi) \cup \{x, y\})$ that s avoids the literal $\neg y$. It remains to prove that Conditions 3.a-3.c of Definition 17 are satisfied. Condition 3.a holds since $\overline{\ell_{2i}} = \ell_{2i+1}$ for each $i \in [k-1]$, by the Condition D of Definition 8. Condition 3.b is satisfied by construction. Finally, Condition 3.c holds since $\mathrm{var}(\ell_{2i-1} \neq \mathrm{var}(\ell_{2i})$ for all $i \in [k]$, by Condition B of Definition 8. A parallel argument proves that there are clauses C and C' in $V(G)$ such that $\neg x \in C$, $\neg y \in C'$, and such that C and C' are connected by a path avoiding y and satisfying the conditions specified in Definition 17. It follows that (x, y) is a resolution-path dependency of Φ, as claimed. □

The notions of strict standard and quadrangle dependencies can be generalized and used to define dependency schemes by simply dropping the requirement that "dependent" pairs of variables occur in neighboring quantifier blocks.

Definition 18. For every PCNF formula Φ, we define relations D^{sst}_Φ and D^{qdr}_Φ as follows. Let $(x, y) \in D^{\mathrm{trv}}_\Phi$, let $qdepth(x) = i$ and let G denote the depth-limited clause-literal graph of Φ at depth i.

- We let $(x, y) \in D^{\mathrm{sst}}_\Phi$ if there are literals ℓ_x, ℓ_y, and ℓ'_y with $\mathrm{var}(\ell_x) = x$ and $\mathrm{var}(\ell_y) = \mathrm{var}(\ell'_y) = y$, as well as clauses $C_1, C_2, C_3, C_4 \in V(G)$ with $\ell_x \in C_1, \overline{\ell_x} \in C_2, \ell_y \in C_3, \ell'_y \in C_4$, such that there are walks in G from C_1 to C_3 and from C_2 to C_4.
- We let $(x, y) \in D^{\mathrm{qdr}}_\Phi$ if there are literals ℓ_x, ℓ_y with $\mathrm{var}(\ell_x) = x$ and $\mathrm{var}(\ell_y) = y$, as well as clauses $C_1, C_2, C_3, C_4 \in V(G)$ with $\ell_x \in C_1, \overline{\ell_x} \in C_2, \ell_y \in C_3, \overline{\ell_y} \in C_4$, such

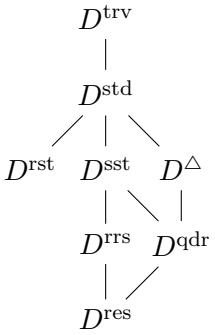

Figure 3.2: Extended Hasse diagram for proto-dependency schemes with respect to the partial order $\subseteq$ of Definition 12.

that there is walk in G from C_1 to C_3 that does not go through $\overline{\ell_y}$, as well as a walk from C_2 to C_4 that does not go through ℓ_y.

The *strict standard dependency scheme* is the mapping D^{sst} that assigns to each PCNF formula Φ the relation D^{sst}_Φ, and the *quadrangle dependency scheme* is the mapping D^{qdr} that assigns to each PCNF formula Φ the relation D^{qdr}_Φ.

Including the strict standard and quadrangle dependency schemes in the Hasse diagram of Figure 3.1 leads to the diagram shown in Figure 3.2. Given a PCNF formula Φ and a pair $(x, y) \in D^{\mathrm{trv}}_\Phi$, determining whether $(x, y) \in D^{\mathrm{sst}}_\Phi$ or $(x, y) \in D^{\mathrm{qdr}}_\Phi$ comes down to to deciding a constant number of simple reachability problems in induced subgraphs of a depth-limited clause-literal graph, so we obtain the following result.

Proposition 4 (Van Gelder [120])**.** *The strict standard dependency scheme and the quadrangle dependency scheme are tractable.*

The quadrangle dependency scheme affords an efficient implementation in terms of *biconnected components* of clause-literal graphs [120]. The original motivation for introducing quadrangle dependencies was a lack of polynomial-time algorithms for computing the more general resolution-path dependency scheme. In the next chapter, we turn to the problem of whether resolution paths can be found efficiently.

Summary

We introduced the notion of a proto-dependency scheme as a mapping that associates each PCNF formula with a binary dependency relation on its variables. A dependency relation is a subset of the so-called trivial dependency relation, which encodes the linear order of variables in the quantifier prefix of a PCNF formula. We defined several proto-dependency schemes, more specifically

- the standard dependency scheme D^{std} and the refined standard dependency scheme D^{rst}, as well as
- the resolution-path dependency scheme D^{res} and the reflexive resolution-path dependency scheme D^{rrs}.

For the applications discussed in later chapters, we prefer proto-dependency schemes that compute smaller dependency relations. We proved that, among the proto-dependency schemes considered, the refined standard dependency scheme D^{rst} and the resolution-path dependency scheme D^{res} compute minimal dependency relations (Proposition 1). Finally, we briefly covered proto-dependency schemes that are intermediate in generality between the standard and resolution-path dependency schemes: the triangle dependency scheme, the strict standard dependency scheme, and the quadrangle dependency scheme.

CHAPTER 4

Finding Resolution Paths

In this chapter, we prove tractability of both the resolution-path dependency scheme and the reflexive resolution-path dependency scheme, answering a question posed by Van Gelder [120]. Recall that a proto-dependency scheme D is *tractable* if there is a polynomial-time algorithm that computes the dependency relation D_Φ for each PCNF formula Φ.

The proof presented below reduces the task of finding a resolution path to a simple connectivity problem in a suitably defined digraph [111] and substantially simplifies our original proof of tractability [107].

Definition 19 (Implication graph[1])**.** Let $\Phi = \mathcal{Q}.\varphi$ be a PCNF formula. The *implication graph* of Φ is the digraph $G = (V, E)$ with vertex set $V = \mathit{lit}(\Phi)$ and edge set $E = \{\, (\overline{\ell}, \ell') : \text{there is a } C \in \varphi \text{ such that } \ell, \ell' \in C \text{ and } \ell \neq \ell' \,\}$.

Lemma 5. *Let Φ be a PCNF formula and let $\ell, \ell' \in \mathit{lit}(\Phi)$ be distinct literals. Let $X \subseteq \mathrm{var}(\Phi)$ and let G denote the implication graph of Φ. The following statements are equivalent.*

1. *The literals ℓ and ℓ' are connected in Φ with respect to X.*
2. *There is a walk from $\overline{\ell}$ to ℓ' in $G[\mathit{lit}(X) \cup \{\overline{\ell}, \ell'\}]$.*

Proof. Let $s = \ell_1, \ldots, \ell_{2k}$ be a resolution path from ℓ to ℓ' via X. Let $\ell_{j_1}, \ldots, \ell_{j_k}$ be the sequence obtained from s by taking every other element, so that $j_i = 2i$ for each $i \in [k]$. We claim that the sequence $w = \overline{\ell_1}, \ell_{j_1}, \ldots, \ell_{j_k}$ is a walk in G. By Definition 8, for each $i \in [k]$ there has to be a clause C_i such that $\ell_{2i-1}, \ell_{2i} \in C_i$. In particular we have $\ell_1, \ell_2 \in C_1$, so $(\overline{\ell_1}, \ell_2)$ is an edge of G. Moreover, $\overline{\ell_{2i}} = \ell_{2i+1}$ for $i \in [k-1]$ and thus $\overline{\ell_{2i}}, \ell_{2(i+1)} \in C_i$ for each $i \in [k-1]$. It follows that $(\ell_{2i}, \ell_{2(i+1)}) = (\ell_{j_i}, \ell_{j_{i+1}})$ is an edge

[1]The implication graph was used to prove tractability of QSAT for PCNF formulas with 2CNF matrices [3]. It also goes under the name of *associated graph* [78, p.75].

of G for each $i \in [k-1]$. This proves the claim. Since $\mathrm{var}(\ell_{2i}) \in X$ for $i \in [k-1]$, the walk w is even a walk in $G[\mathit{lit}(X) \cup \{\overline{\ell}, \ell'\}]$.

For the converse, let $p = \ell_1, \ldots, \ell_k$ be a path in $G[\mathit{lit}(X) \cup \{\overline{\ell}, \ell'\}]$ such that $\ell_1 = \overline{\ell}$ and $\ell_k = \ell'$. We claim that the sequence

$$s = \overline{\ell_1}, \ell_2, \overline{\ell_2}, \ldots, \ell_{k-1}, \overline{\ell_{k-1}}, \ell_k$$

is a resolution path via X. We have to verify that the conditions of Definition 8 are satisfied. Because (ℓ_i, ℓ_{i+1}) is an edge of G, there must be a clause $C_i \in \varphi$ such that $\overline{\ell_i}, \ell_{i+1} \in C_i$ for each $i \in [k-1]$; moreover, $\overline{\ell_i}$ and ℓ_{i+1} must be distinct by definition of the implication graph, so $\mathrm{var}(\overline{\ell_i}) \neq \mathrm{var}(\ell_{i+1})$ as C_i is non-tautological. This establishes the first and second condition of Definition 8. Since p is a path the literals ℓ_i must be pairwise distinct. In particular we have $\overline{\ell}, \ell' \notin \{\ell_2, \ldots, \ell_{k-1}\}$, so $\{\ell_2, \ldots, \ell_{k-1}\} \cup \overline{\{\ell_2, \ldots, \ell_{k-1}\}} \subseteq \mathit{lit}(X)$ and the third condition holds. Finally, the fourth condition is satisfied by construction. □

The implication graph of a formula can be constructed in time quadratic in the size of Φ and directed reachability can be decided in linear time. In combination with Lemma 5, these observations yield tractability of the resolution-path dependency scheme and the reflexive resolution-path dependency scheme.

Theorem 2. *Both the resolution-path dependency scheme and the reflexive resolution-path dependency scheme are tractable: given a PCNF formula Φ, each of the dependency relations D^{res}_Φ and D^{rrs}_Φ can be computed in polynomial time.*

If the size of input clauses is bounded by a constant the implication graph can even be constructed in linear time, since every clause contributes a constant number of edges. If not, we can convert the input formula into an equivalent formula with bounded clause size by splitting clauses [78, p.40]. The following lemma states that this transformation preserves connectivity of literals.

Lemma 6. *Let $\Phi = \mathcal{Q}.\varphi$ be a PCNF formula and let $C \in \varphi$ be a clause of Φ such that $C = C_1 \cup C_2$ where $C_1 \cap C_2 = \emptyset$. Let $\Psi = \mathcal{Q}'.\varphi'$ be a PCNF formula with $\mathcal{Q}' = \mathcal{Q}\,\exists x$ and $\varphi' = (\varphi \setminus \{C\}) \cup \{C_1 \cup \{x\}, C_2 \cup \{\neg x\}\}$ for a fresh variable $x \notin \mathrm{var}(\Phi)$. Two literals ℓ, ℓ' are connected in Φ with respect to $X \subseteq \mathrm{var}(\Phi)$ if and only if ℓ, ℓ' are connected in Ψ with respect to $X \cup \{x\}$.*

Proof. Let G and H denote the implication graphs of Φ and Ψ, respectively, and let $G' = G[\mathit{lit}(X) \cup \{\overline{\ell}, \ell'\}]$ and $H' = H[\mathit{lit}(X \cup \{x\}) \cup \{\overline{\ell}, \ell'\}]$. We claim that there is a walk from $\overline{\ell}$ to ℓ' in G' if and only if there is a walk from $\overline{\ell}$ to ℓ' in H'. The result then follows from Lemma 5.

Let $w = \ell_1, \ldots, \ell_k$ be a walk in G' from $\overline{\ell}$ to ℓ'. Suppose (ℓ_i, ℓ_{i+1}) is an edge of G' but not of H' for any $i \in [k-1]$. Then $\overline{\ell_i} \in C_1$ and $\ell_{i+1} \in C_2$, or $\overline{\ell_i} \in C_2$ and $\ell_{i+1} \in C_1$. Assume without loss of generality that the first case holds. Then (ℓ_i, x) and (x, ℓ_{i+1}) are edges of H'. By inserting x between every such pair ℓ_i, ℓ_{i+1} of subsequent literals in w we obtain a walk from $\overline{\ell}$ to ℓ' in H'. For the converse, consider a walk $w = \ell_1, \ldots, \ell_k$ in H' from $\overline{\ell}$ to ℓ'. Note that $x \notin \mathrm{var}(\{\ell, \ell'\})$ because $x \notin \mathrm{var}(\Phi)$. If $\mathrm{var}(\ell_i) = x$ for $1 < i < k$

then (ℓ_{i-1}, ℓ_{i+1}) is an edge of G', so we obtain a walk from $\bar{\ell}$ to ℓ' simply by omitting occurrences of x and $\neg x$ from w. □

Lemma 7. *Given a PCNF formula Φ and a pair (x, y) of variables, one can decide whether $(x, y) \in D^{\mathrm{rrs}}_{\Phi}$ (respectively, whether $(x, y) \in D^{\mathrm{res}}_{\Phi}$) in linear time.*

Proof. If $(x, y) \notin D^{\mathrm{trv}}_{\Phi}$ the algorithm immediately rejects. Otherwise, it uses Lemma 6 to split clauses of Φ and construct a PCNF formula Ψ such that each clause of Ψ contains at most three literals and such that $(x, y) \in D^{\mathrm{rrs}}_{\Phi}$ if and only $(x, y) \in D^{\mathrm{rrs}}_{\Psi}$. This takes time linear in the size of Φ. The algorithm then determines whether $\{x, y\}$ is a dependency pair with respect to $R_{\Psi}(x) \setminus \mathrm{var}_{\forall}(\Psi)$. By Lemma 5, this comes down to deciding at most four instances of directed reachability in induced subgraphs of Ψ's implication graph. Since these subgraphs can be constructed in linear time, the overall runtime is linear in the size of Φ. By checking whether $\{x, y\}$ is an irreflexive resolution-path dependency pair with respect to $R_{\Psi}(x) \setminus \mathrm{var}_{\forall}(\Psi)$, this algorithm can be also used to decide whether $(x, y) \in D^{\mathrm{res}}_{\Phi}$. □

It follows that, given a PCNF formula, the resolution-path dependency relation can be computed in time cubic in the size of the formula.

Proof of Theorem 2. Given a PCNF formula Φ, the algorithm computes D^{trv}_{Φ} and tests whether $(x, y) \in D^{\mathrm{res}}_{\Phi}$ ($(x, y) \in D^{\mathrm{rrs}}_{\Phi}$) for each pair $(x, y) \in D^{\mathrm{trv}}_{\Phi}$. Since $|D^{\mathrm{trv}}_{\Phi}| \leq |\mathrm{var}(\Phi)|^2$ and each test can be performed in linear time by Lemma 7, this takes $O(|\Phi|^3)$ time in total. □

Formulas encountered in applications frequently have encoding sizes that render the above algorithm ineffective. The main advantage of the refined standard dependency scheme over the resolution-path dependency scheme is that it can be computed in linear time [15, 82]. The characterization of resolution paths as directed paths in the implication graph not only substantially simplifies the proof of Theorem 2 compared to our original proof [107], but also points to relaxations of the resolution-path dependency scheme based on strongly connected components of the implication graph that can be computed in linear time. Whether such relaxations will be useful in practice is an open question.

Summary

In this short chapter we proved that both the resolution-path dependency scheme and the reflexive resolution-path dependency scheme are tractable – that is, there are polynomial-time algorithms that compute the corresponding dependency relations for a given input formula. We showed that deciding whether a pair (x, y) is contained in one of these relations can be reduced to checking whether there is a pair of directed paths connecting literals over x and y in the so-called implication graph of a formula. Although this graph can be quadratic in the size of the formula, we argued that the problem can still be decided in linear time by splitting clauses in a preprocessing step.

CHAPTER 5

Quantified Resolution

We now turn to the most important application of dependency schemes: their use in search-based PCNF solvers. The integration of dependency schemes is arguably the main feature of DepQBF [14, 15, 82], one of the best state-of-the-art QBF solvers. In its basic form, the QDPLL algorithm implemented by search-based solvers is constrained by the trivial dependency relation of the input formula [25]. DepQBF generalizes QDPLL by replacing the trivial dependency relation with the dependency relation computed by a proto-dependency scheme – QDPLL is just a special case of this algorithm running with the trivial dependency scheme [15].

While the resulting algorithm cannot be correct for every proto-dependency scheme, it has been suggested that it is correct for every *cumulative*[1] dependency scheme [15], which would include all dependency schemes introduced in Chapter 3. As we will see in this chapter, things are not quite as simple, and proving correctness even for the standard dependency scheme turns out to be non-trivial.

We will prove correctness of DepQBF (for specific dependency schemes) in the sense that certain proof systems, implicitly used by DepQBF to generate certificates, are sound. Applications typically use solvers not as mere QSAT oracles but require them to generate certificates. This is the case in planning, where certificates correspond to plans, or in model checking, where certificates encode proofs of—or counterexamples to—the correctness of a given specification.

The traces of QDPLL solvers for PCNF formulas can be used to generate *Q-resolution* certificates [24, 51, 86]. More specifically, traces yield *Q-resolution refutations* of false formulas and *Q-term resolution proofs* of true formulas. As a consequence of the way it uses dependency schemes (see Section 5.1.2), DepQBF produces certificates that—in general—do not correspond to Q-resolution proofs. To reason about these certificates, we introduce *Q(D)-resolution* and *Q(D)-term resolution*, two proof systems that are parameterized by a (tractable) proto-dependency scheme D. In these systems, the

[1] See Section 6.1 for the definition of cumulative dependency schemes.

dependency relation computed by D is used to define more powerful versions of the $\forall$-reduction and $\exists$-reduction rules of Q-resolution (respectively Q-term resolution) [15, p.7]. We prove the following results:

1. $Q(D^{\text{std}})$-resolution and $Q(D^{\text{rst}})$-resolution are sound (Theorem 4).
2. $Q(D^{\text{rrs}})$-resolution is sound (Theorem 5).
3. $Q(D^{\text{rst}})$-term resolution is sound (Theorem 6).
4. $Q(D^{\text{res}})$-term resolution is sound (Theorem 7).

To prove the first result (Section 5.2.1) we argue that D^{std}-resolution and D^{rst}-resolution preserve a subset of the models of every true formula, and then apply a known result about the existence of such models.

The second result (Section 5.2.2) is proved using an entirely different strategy that relies on proof rewriting. We present an algorithm that turns $Q(D^{\text{rrs}})$-resolution refutations into Q-resolution refutations, though perhaps at the cost of an exponential increase in size.

Our approach in proving the third (Section 5.3.1) and fourth (Section 5.3.2) result is essentially dual to the one used to establish the first result: we show that $Q(D^{\text{res}})$-term and $Q(D^{\text{rst}})$-term resolution each preserve a (different) subset of *counter*models and then show how to construct such countermodels from (ordered) Q-resolution refutations.

Given soundness of $Q(D^{\text{res}})$-term resolution, one may wonder why we "only" prove soundness of the less general $Q(D^{\text{rrs}})$-resolution system and not soundness of $Q(D^{\text{res}})$-resolution. The answer is given in the form of an example in Section 5.1 that demonstrates that $Q(D^{\text{res}})$-resolution in general is *unsound*.

5.1 Q-resolution and Q(D)-resolution

5.1.1 Q-resolution and term resolution

Q-resolution is a generalization of propositional resolution to PCNF formulas [24]. Its derivation rules are displayed in Figure 5.1. Throughout this chapter, we refer to clauses (or terms) above the inference line of a derivation rule as the *premises*, and to the clause (or term) below the inference line as the *conclusion* of the derivation rule.

The dual of Q-resolution, operating on terms instead of clauses, is known as *term resolution*. Term resolution yields a proof system for (true) prenex QBFs with matrices in disjunctive normal form (DNF). One obtains a proof system for PCNF formulas by adding the so-called *model generation* rule. This rule carries out the CNF to DNF conversion, one term at a time [51]. We call the resulting proof system *Q-term resolution*. Its derivation rules are shown in Figure 5.2.

Q-resolution and Q-term resolution are closely related to search-based QBF solvers that implement the QDPLL algorithm [25]. The trace of a QDPLL solver can be used to generate either a Q-resolution proof (if the input formula is false) or a Q-term resolution

$$\frac{}{C}\ \text{(input clause)} \qquad \frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2}\ \text{(resolution)}$$

An *input clause* $C \in \varphi$ can be used as an axiom. From two clauses $C_1 \vee x$ and $\neg x \vee C_1$, where x is an existential variable of Φ (called the *pivot variable*), the *resolution* rule can derive the clause $C_1 \vee C_2$, provided that this clause is non-tautological.

$$\frac{C \vee \ell}{C}\ (\forall\text{-reduction})$$

The $\forall$-*reduction* rule derives a clause C from a clause $C \vee \ell$ provided that ℓ is a universal literal and $(\mathrm{var}(\ell), e) \notin R_\Phi$ for every existential variable $e \in \mathrm{var}(C)$.

Figure 5.1: The rules of Q-resolution for a PCNF formula $\Phi = \mathcal{Q}.\varphi$.

$$\frac{}{T}\ \text{(model generation)} \qquad \frac{T_1 \wedge x \qquad \neg x \wedge T_2}{T_1 \wedge T_2}\ \text{(resolution)}$$

The *model generation* rule [51] can derive a non-contradictory term T such that $T \cap C \neq \emptyset$ for every $C \in \varphi$. The *resolution* rule for terms is dual to resolution for clauses. The pivot variable x has to be a universal variable of Φ and $T_1 \wedge T_2$ must be non-contradictory.

$$\frac{T \wedge \ell}{T}\ (\exists\text{-reduction})$$

The $\exists$-*reduction* rule can derive the term T from $T \wedge \ell$ if ℓ is an existential literal and $(\mathrm{var}(\ell), u) \notin R_\Phi$ for every universal variable $u \in \mathrm{var}(T)$.

Figure 5.2: The rules of Q-term resolution for a PCNF formula $\Phi = \mathcal{Q}.\varphi$.

proof (if the input formula is true) [51]. This correspondence is perhaps even more immediate in the presence of constraint learning: every clause (term) learned by the solver can be derived by Q-resolution (Q-term resolution) [51].

5.1.2 Dependency Schemes and DepQBF

QDPLL lifts the well-known DPLL procedure [36] from SAT to QSAT. In essence, DPLL is a recursive algorithm that picks a variable of its input formula and calls itself for both possible instantiations of that variable. Modern SAT solvers derived from the DPLL algorithm delegate the choice of which variable to branch on to sophisticated heuristics [106].

In QDPLL, these heuristics are constrained by the order of variables given by the quantifier prefix: a variable may be assigned only if it occurs in the leftmost quantifier block with unassigned variables. This is often much more restrictive than necessary. For instance, two variables appearing in different, variable disjoint subformulas can be assigned in any order. More generally, a variable can be assigned as long as it *does not depend* on any unassigned variable. This is the insight underlying the algorithm

$$\frac{C \vee \ell}{C}\ (\forall(D)\text{-reduction})$$

The $\forall(D)$-*reduction* rule derives the clause C from $C \vee \ell$, where ℓ is a universal literal such that $(\mathrm{var}(\ell), e) \notin D_\Phi$ for every existential variable $e \in \mathrm{var}(C)$.

$$\frac{T \wedge \ell}{T}\ (\exists(D)\text{-reduction})$$

The $\exists(D)$-*reduction* rule derives the term T from $T \wedge \ell$, where ℓ is an existential literal such that $(\mathrm{var}(\ell), u) \notin D_\Phi$ for every universal variable $u \in \mathrm{var}(T)$.

Figure 5.3: $\forall(D)$-reduction and $\exists(D)$-reduction for Φ and a proto-dependency scheme D.

implemented in the solver DepQBF [15, 82]. DepQBF uses the dependency relation given by a proto-dependency scheme D (in the current implementation, the refined standard dependency scheme) to determine variables that can be branched on: if, for a given variable y, there is no unassigned variable x such that (x, y) is in the dependency relation, then y is considered ready for assignment. DepQBF also appeals to the dependency relation to generalize the $\forall$-reduction and $\exists$-reduction rules used in constraint learning [15]. These derivation rules, which we refer to as $\forall(D)$-*reduction* and $\exists(D)$-*reduction*, are shown in Figure 5.3.

5.1.3 Q(D)-resolution and Q(D)-term resolution

To study proofs generated by DepQBF in combination with different proto-dependency schemes, we introduce two families of proof systems as follows. Let D be a tractable[2] proto-dependency scheme. We define $Q(D)$-*resolution* as the proof system consisting of resolution and $\forall(D)$-reduction, and $Q(D)$-*term resolution* as the proof system consisting of (term) resolution, $\exists(D)$-reduction, and the model generation rule.

Derivations in these proof systems consist of repeated applications of the derivation rules to derive a clause or term from an input formula. Derivations are customarily represented as sequences of clauses (or terms) or as directed acyclic graphs (DAGs). Given a derivation in the latter representation, the underlying DAG can be turned into a tree, usually at the cost of increasing the size of the derivation.

For our purposes, it is most convenient to represent $Q(D)$-resolution derivations as labeled trees (see Section 2); $Q(D)$-term resolution derivations, on the other hand, will be represented simply as sequences.

Definition 20 (Q(D)-resolution derivation)**.** Let D be a tractable proto-dependency scheme and let $\Phi = \mathcal{Q}.\varphi$ be a PCNF formula. The set of *(tree-like)* $Q(D)$-*resolution derivations* from Φ is inductively defined as follows.

1. If $C \in \varphi$ then $\triangle(C)$ is a $Q(D)$-resolution derivation from Φ.

[2]Tractability of D ensures that Q(D)-resolution proofs can be checked in polynomial time.

2. Let $\mathcal{T}_1 = (T_1, r_1, \lambda_1)$ and $\mathcal{T}_2 = (T_2, r_2, \lambda_2)$ be $Q(D)$-resolution derivations from Φ. If $\ell \in \mathit{lit}(\Phi)$ is an existential literal such that $\ell \in \lambda_1(r_1)$ and $\overline{\ell} \in \lambda_2(r_2)$, and $(\lambda_1(r_1) \setminus \{\ell\}) \cup (\lambda_2(r_2) \setminus \{\overline{\ell}\})$ is a non-tautological clause, then $\mathcal{T}_1 \odot_\ell \mathcal{T}_2$ is a $Q(D)$-resolution derivation from Φ.

3. Let $\mathcal{T} = (T, r, \lambda)$ be a $Q(D)$-resolution derivation from Φ. If $\ell \in \mathit{lit}(\Phi)$ is a universal literal such that $\ell \in \lambda(r)$ and there is no existential literal $\ell' \in \lambda(r)$ such that $(\mathrm{var}(\ell), \mathrm{var}(\ell')) \in D_\Phi$, then $\mathcal{T}\|_\ell$ is a $Q(D)$-resolution derivation from Φ.

If $\mathcal{T} = (T, r, \lambda)$ is a $Q(D)$-resolution derivation then $\lambda(r)$ is a clause called the *conclusion* of $\mathcal{T}$, and $\mathcal{T}$ is said to be a *derivation of* $\lambda(r)$. If $\mathcal{T}$ is a $Q(D)$-resolution derivation of the empty clause $\emptyset$, then $\mathcal{T}$ is a $Q(D)$-*resolution refutation* of Φ. The *size* of a $Q(D)$-resolution derivation $\mathcal{T} = (T, r, \lambda)$, in symbols $|\mathcal{T}|$, is the number of vertices in T.

Observe that these construction rules are in accord with the rules of $Q(D)$-resolution as displayed in Figure 5.1 and Figure 5.3. We will sometimes identify a derivation with its final derivation step and say that $\mathcal{T} \approx \star\|_\ell$ is a $\forall$-*reduction step* or that $\mathcal{T} \approx \star \odot_\ell \star$ is a *resolution step*.

We use sequences of literals, called *positions*, occurring as edge labels on paths from the root of a derivation to address parts of a derivation.

Definition 21 (Position)**.** Let D be a proto-dependency scheme, let Φ be a PCNF formula, and let $\mathcal{T}$ be a $Q(D)$-resolution derivation from Φ. The set of *positions* of $\mathcal{T}$ is inductively defined as follows.

1. The empty sequence ε is a position of $\mathcal{T}$.

2. If $\mathcal{T} \approx \mathcal{T}' \odot_\ell \star$ and π is a position of $\mathcal{T}'$, then $\ell\pi$ is a position of $\mathcal{T}$.

3. If $\mathcal{T} \approx \mathcal{T}'\|_\ell$ and π is a position of $\mathcal{T}'$, then $\ell\pi$ is a position of $\mathcal{T}$.

If π is a position of $\mathcal{T}$ and there is no position π' of $\mathcal{T}$ such that π is a proper prefix of π', then π is a *leaf position* of $\mathcal{T}$.

Definition 22 (Subderivation)**.** Let D be a proto-dependency scheme, let Φ be a PCNF formula, and let $\mathcal{T}$ be a $Q(D)$-derivation from Φ. If π is a position of $\mathcal{T}$, the *subderivation of* $\mathcal{T}$ *at position* π, in symbols $\mathcal{T}[\pi]$, is defined as follows.

$$\mathcal{T}[\pi] = \begin{cases} \mathcal{T} & \text{if } \pi = \varepsilon, \\ \mathcal{T}'[\pi'] & \text{if } \pi = \ell\pi' \text{ and } \mathcal{T} \approx \mathcal{T}' \odot_\ell \star, \\ \mathcal{T}'[\pi'] & \text{if } \pi = \ell\pi' \text{ and } \mathcal{T} \approx \mathcal{T}'\|_\ell; \end{cases}$$

Definition 23 (Ordered Derivation)**.** Let D be a tractable proto-dependency scheme and let Φ be a PCNF formula. A $Q(D)$-resolution derivation $\mathcal{T}$ from Φ is *ordered* if for every position $\pi = \ell_1, \ldots, \ell_k$ satisfies $(\ell_i, \ell_j) \in R_\Phi$ for each pair of indices $i, j \in [k]$ such that $1 \leq i < j \leq k$.

Definition 24 (Q(D)-term resolution derivation)**.** Let D be a tractable proto-dependency scheme and let Φ be a PCNF formula. A $Q(D)$*-term resolution derivation* of a term T from Φ is a sequence $\mathcal{S} = T_1, \ldots, T_k$ of terms such that $T = T_k$, and such that each term is either derived by the model generation rule, or derived from terms appearing earlier in the sequence by resolution or $\exists(D)$-reduction. The term T is called the *conclusion* of $\mathcal{S}$. If $\mathcal{S}$ is a $Q(D)$-term resolution derivation of the empty term then $\mathcal{S}$ is a $Q(D)$*-term resolution proof* of Φ.

Verify that Q-resolution and Q-term resolution correspond to $Q(D^{\text{trv}})$-resolution and $Q(D^{\text{trv}})$-term resolution, respectively. Accordingly, we define *Q-resolution derivations (refutations)* as $Q(D^{\text{trv}})$-resolution derivations (refutations). Similarly, we define *Q-term resolution derivations (proofs)* as $Q(D^{\text{trv}})$-term resolution derivations (proofs). The following theorem states that Q-resolution and Q-term resolution are sound and complete proof systems for false and true PCNF formulas, respectively.

Theorem 3 (Kleine Büning et. al. [24], Giunchiglia et. al. [51])**.** *For every PCNF formula* Φ*,*

1. Φ *is false if and only if there is a Q-resolution refutation of* Φ*, and*
2. Φ *is true if and only if there is a Q-term resolution proof of* Φ*.*

Every Q-resolution derivation is a $Q(D)$-resolution derivation, and every Q-term resolution derivation is a $Q(D)$-term resolution derivation (see Figure 5.1, Figure 5.2, and Figure 5.3), so completeness of these systems follows immediately from Theorem 3. On the other hand, it is easy to find proto-dependency schemes D for which $Q(D)$-resolution or $Q(D)$-term resolution are unsound.

In fact, the following example (taken from [100]) demonstrates that $Q(D^{\text{res}})$-resolution is unsound.[3] This is in spite of the fact that, as we will see in Chapter 6, D^{res} is a *permutation dependency scheme* that supports strong reordering operations on the quantifier prefixes of PCNF formulas.

Example 1. Let $\Phi = \mathcal{Q}.\varphi$, where

$$\mathcal{Q} = \forall x \exists z \forall u \exists y, \text{ and}$$
$$\varphi = (x \vee u \vee \neg y) \wedge (\neg x \vee \neg u \vee \neg y) \wedge (z \vee u \vee y) \wedge (\neg z \vee u \vee \neg y) \wedge (\neg z \vee \neg u \vee y) \wedge (z \vee \neg u \vee \neg y).$$

The formula Φ is true, but Figure 5.1.3 shows a $Q(D^{\text{res}})$-resolution refutation of Φ, so $Q(D^{\text{res}})$-resolution is unsound. The pair (x, y) is not in D^{res}_{Φ} since every resolution path from x or $\neg x$ to y goes through y. As a consequence, one can derive the clause $(u \vee \neg y)$ from $(x \vee u \vee \neg y)$, and the clause $(\neg u \vee \neg y)$ from $(\neg x \vee \neg u \vee \neg y)$ by $\forall(D^{\text{res}})$-reduction.

[3] Actually, the example shows that already $Q(D^{\triangle})$-resolution is unsound. See Section 3.4 for the definition of the triangle dependency scheme $D^{\triangle}$.

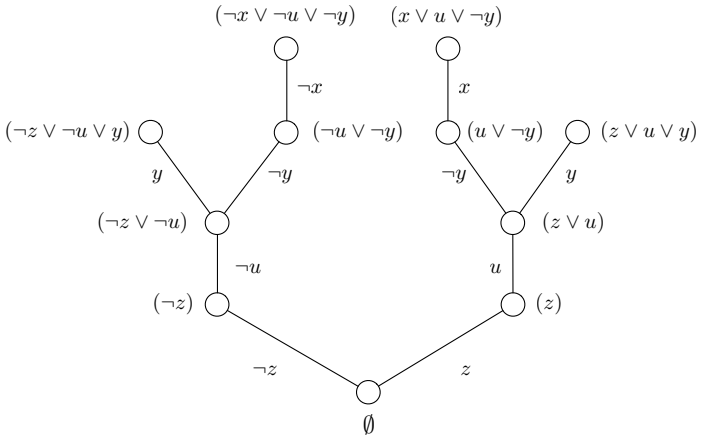

Figure 5.4: $Q(D^{\text{res}})$-refutation of Φ from Example 1.

The problem with D^{res} in Example 1 can be "fixed" if the resolutions paths $\neg x, \neg y$ and $x, \neg y, y, z, \neg z, y$ cause (x, y) to be in the dependency relation even though the second path goes through y. This observation motivated our definition of the reflexive resolution-path dependency scheme D^{rrs} [111]. Soundness of $Q(D^{\text{rrs}})$-resolution is the main result of the next Section.

5.2 Soundness of Q(D)-resolution

5.2.1 The (Refined) Standard Dependency Scheme

In this subsection, we prove that Q(D)-resolution is sound for the refined standard dependency scheme. Recall that by Lemma 1, the relations D^{std}_{Φ} and D^{rst}_{Φ} coincide for pairs $(u, e) \in \text{var}_{\forall}(\Phi) \times \text{var}_{\exists}(\Phi)$. It is an immediate consequence that $Q(D^{\text{std}})$-resolution and $Q(D^{\text{rst}})$-resolution coincide.

Lemma 8. *Let Φ be a PCNF formula. A labeled, rooted tree $\mathcal{T}$ is a $Q(D^{\text{std}})$-resolution derivation from Φ if and only if $\mathcal{T}$ is a $Q(D^{\text{rst}})$-resolution derivation from Φ.*

Soundness of Q-resolution can be proved by arguing that Q-resolution preserves every complete model of a PCNF formula. That is, if a formula Φ has a complete model $\mathbf{f}$ and σ is an assignment to the universal variables of Φ, then every clause that can be derived from Φ by Q-resolution is satisfied by the assignment $\sigma \cup \mathbf{f}(\sigma)$. Since the empty clause is trivially unsatisfiable, no true PCNF formula can have a Q-resolution refutation.

To prove that $Q(D^{\text{std}})$-resolution is sound, we generalize this argument and show that $Q(D^{\text{std}})$-resolution preserves a non-empty subset of models for every true PCNF formulas. More specifically, we will show that this is the case for models satisfying certain *independence* constraints.

Definition 25 (Independence)**.** Let $f : 2^X \to \{0,1\}$ be a function and let $x \in X$. We say *f is independent of x* if $f(\sigma) = f(\tau)$ for every pair of truth assignments $\sigma, \tau : X \to \{0,1\}$ such that $\sigma(y) = \tau(y)$ for all $y \in X \setminus \{x\}$.

Definition 26. Let D be a proto-dependency scheme and let Φ be a PCNF formula. A complete model $\mathbf{f}$ of Φ is a *D-model of Φ* if each function $f_e \in \mathbf{f}$ is independent of every universal variable u with $(u, e) \notin D_\Phi$.

Lemma 9. *Let D be a proto-dependency scheme and $\Phi = \mathcal{Q}.\varphi$ a PCNF formula. Let $\mathbf{f}$ be a D-model of Φ and let $\mathcal{T}$ be a $Q(D)$-resolution derivation of a clause C from Φ. Then $C[\sigma \cup \mathbf{f}(\sigma)] = 1$ for every assignment σ to the universal variables of Φ.*

Proof. We prove the lemma by induction on the structure of $\mathcal{T}$. Let $\sigma : \mathrm{var}_\forall(\Phi) \to \{0,1\}$ be an assignment.

1. If $\mathcal{T} \approx \triangle(C)$ then $C \in \varphi$ and $C[\sigma \cup \mathbf{f}(\sigma)] = 1$ since $\mathbf{f}$ is a model of Φ.

2. Let $\mathcal{T} \approx \mathcal{T}_1 \odot_\ell \mathcal{T}_2$ and suppose the lemma holds for the conclusions C_1 and C_2 of $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively. We distinguish two cases. If $C_1 \setminus \{\ell\}[\sigma \cup \mathbf{f}(\sigma)] = 1$ then in particular $C[\sigma \cup \mathbf{f}(\sigma)] = 1$ since $C_1 \setminus \{\ell\} \subseteq C$. Otherwise, we have $\mathbf{f}(\sigma)(\ell) = 1$ and $\mathbf{f}(\sigma)(\overline{\ell}) = 0$, so we must have $C_2 \setminus \{\overline{\ell}\}[\sigma \cup \mathbf{f}(\sigma)] = 1$. This again implies that $C[\sigma \cup \mathbf{f}(\sigma)] = 1$ as $C_2 \setminus \{\overline{\ell}\} \subseteq C$.

3. Let $\mathcal{T} \approx \mathcal{T}' \|_\ell$ and suppose the lemma holds for the conclusion C' of $\mathcal{T}$. Suppose towards a contradiction that $C[\sigma \cup \mathbf{f}(\sigma)] = 0$. Since $C = C' \setminus \{\ell\}$, and $C[\sigma \cup \mathbf{f}(\sigma)] = 1$ by assumption, this implies $\sigma(\ell) = 1$. Let $u = \mathrm{var}(\ell)$ and let $\sigma' : \mathrm{var}_\forall(\Phi) \to \{0,1\}$ be the assignment such that $\sigma'(\ell) = 0$ and $\sigma'(v) = \sigma(v)$ for every variable $v \in \mathrm{var}_\forall(\Phi) \setminus \{u\}$. By definition of the $\forall(D)$-reduction rule, there cannot be an existential variable $e \in \mathrm{var}(C')$ such that $(u, e) \in D_\Phi$. Because $\mathbf{f}$ is a D-model, it follows that for each existential variable $e \in \mathrm{var}(C')$, the model function f_e has to be independent of u. It follows that $f_e(\sigma'|_{L_e^\Phi}) = f_e(\sigma|_{L_e^\Phi})$ for each existential variable $e \in \mathrm{var}(C')$. From this, we get $C'[\sigma' \cup \mathbf{f}(\sigma')] = 0$, contradicting our initial assumption. We conclude that $C[\sigma \cup \mathbf{f}(\sigma)] = 1$.

□

Proposition 5. *If D is a proto-dependency scheme such that every true PCNF formula has a D-model, then $Q(D)$-resolution is sound: a PCNF formula Φ is false if and only if there is a $Q(D)$-resolution refutation of Φ.*

Proof. Let D be a proto-dependency scheme such that every true PCNF formula has a D-model. The empty clause is trivially unsatisfiable, so by Lemma 9 no true PCNF formula can have a $Q(D)$-resolution refutation. This proves the "if" part. The "only if" part follows immediately from Theorem 3 and the fact that every Q-resolution refutation is a $Q(D)$-resolution refutation. □

Accordingly, to prove soundness of $Q(D^{\mathrm{std}})$-resolution it is sufficient to establish the following result, which is due to Bubeck [20, Theorem 5.3.3].

Proposition 6. *Every true PCNF formula has a D^{std}-model.*

Theorem 4. *A PCNF formula Φ is false if and only if there is a $Q(D^{\mathrm{std}})$-resolution refutation ($Q(D^{\mathrm{rst}})$-resolution refutation) of Φ.*

Proof. Immediate from Proposition 5 in combination with Proposition 6. By Lemma 8, the result can equivalently be stated in terms of $Q(D^{\mathrm{rst}})$-resolution refutations. □

5.2.2 The Reflexive Resolution-Path Dependency Scheme

In this subsection we show that $Q(D^{\mathrm{rrs}})$-resolution is sound. Although we conjecture that every true PCNF formula has a D^{rrs}-countermodel, we currently do not know whether this is the case. As a consequence, we cannot adopt the proof strategy used in the previous subsection. Instead, we will show how to *rewrite* $Q(D^{\mathrm{rrs}})$-resolution refutations into ordinary Q-resolution refutations and prove the following result.

Proposition 7. *Given a PCNF formula Φ and a $Q(D^{\mathrm{rrs}})$-refutation $\mathcal{T}$ of Φ, one can compute a Q-resolution refutation of Φ of size at most $3^{|\mathcal{T}|}$.*

Recall that $\forall(D)$-reduction is a generalization of $\forall$-reduction that can remove a universal literal ℓ from a clause C provided that C does not contain an existential literal ℓ' such that $(\mathrm{var}(\ell), \mathrm{var}(\ell')) \in D_\Phi$. Thus $\forall(D)$-reduction can sometimes remove a universal literal $\ell \in C$ even when there is an existential literal $\ell' \in C$ that *blocks* ℓ (that is, where $(\ell, \ell') \in R_\Phi$). We refer to such an application of $\forall(D)$-reduction as a *strict $\forall(D)$-reduction*. We suppress the proto-dependency scheme D in this notation and simply speak of *strict reductions* when D is clear from the context.

Algorithm outline. To turn a $Q(D^{\mathrm{rrs}})$-resolution refutation $\mathcal{T}$ of a PCNF formula Φ into a Q-resolution refutation of Φ, Algorithm 3 recursively gets rid of strict reductions, starting with what we call an *outermost* strict reduction. Relative to $\mathcal{T}$, a strict reduction is *outermost* if the universal variable u removed by this strict reduction is leftmost in the quantifier prefix among the variables removed by strict reductions in $\mathcal{T}$. Suppose ℓ is the universal literal removed by an outermost strict reduction. We have to consider two cases.

1. Suppose the complementary literal $\overline{\ell}$ is not contained in any clause appearing as a vertex label on the path from the strict reduction to the root of $\mathcal{T}$. Then we simply omit this strict reduction and add a $\forall(D^{\mathrm{rrs}})$-reduction at the root of $\mathcal{T}$. If the conclusion of $\mathcal{T}$ does not contain any literals blocking ℓ, the $\forall(D^{\mathrm{rrs}})$-reduction is in fact an instance of "ordinary" $\forall$-reduction. This condition is satisfied by a refutation, and it will be maintained for subderivations and their outermost strict reductions in the recursion step.

2. Otherwise, it follows from the properties of D^{rrs} that the derivation must contain a resolution step $\star \odot_{\ell'} \star$ on an existential literal ℓ' such that $(\ell', \ell) \in R_\Phi$ (see Lemma 18). We "drop" a lowermost (i.e., one that is closest to the root) such resolution step to the root of the derivation. This may introduce ℓ' or $\overline{\ell'}$ to the clauses on the path from the resolution step to the root. But since the strict reduction picked in the first step is outermost, these literals will not interfere with strict reductions. Moreover, because the resolution step is lowermost, every clause on the path contains an existential pivot variable y such that y blocks u. Thus every $\forall(D^{\mathrm{rrs}})$-reduction on this path is in fact a strict reduction and the resulting derivation is a $Q(D^{\mathrm{rrs}})$-resolution derivation.

In this way, we obtain a $Q(D^{\mathrm{rrs}})$-resolution derivation whose immediate subderivations are (a) strictly smaller than the original refutation (although the overall size of the refutation may increase) and which (b) do not contain "new" strict reductions (we will define a preorder on derivations to make this notion precise). We get to a Q-resolution refutation by running the algorithm on these subderivations to rewrite them into Q-resolution derivations and adding a final resolution (or $\forall$-reduction step), if necessary. We illustrate this rewriting procedure with the following example.[4]

Example 2. Consider the PCNF formula

$$\Phi = \exists e_1 \forall u \exists e_2 \exists e_3 \, (u \vee e_2) \wedge (\neg u \vee e_3) \wedge (e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_2)$$

The refutation $\mathcal{T}_1$ in Figure 5.5 is a $Q(D^{\mathrm{rrs}})$-resolution refutation of Φ. Both $\forall(D^{\mathrm{rrs}})$-reductions are strict reductions, and both are outermost, so the algorithm can start by removing either of them. Suppose the algorithm picks the strict reduction on the left. The literal $\neg u$ does not occur on the path from the root of the derivation to the strict reduction, so the first of the above cases applies. We move the $\forall(D^{\mathrm{rrs}})$-reduction to the root, resulting in the derivation $\mathcal{T}_2$, and proceed with the subderivation rooted immediately above the root. The $\forall(D^{\mathrm{rrs}})$-reduction removing $\neg u$ is the only (outermost) strict reduction appearing in this derivation. Since the conclusion of the subderivation contains the complementary literal u, the second of the above cases applies. We find that the resolution with pivot e_1 is a lowermost resolution step such that $(e_1, u) \in R_\Phi$. "Dropping" this resolution step to the root of the derivation leads to the Q-resolution refutation $\mathcal{T}_3$.

Definition 27 (Strict Reduction)**.** Let D be a proto-dependency scheme, let Φ be a PCNF formula, and let $\mathcal{T}$ be a $Q(D)$-resolution derivation from Φ. If π is a position of

[4]Example 2 also demonstrates that known rewrite strategies for removing long-distance resolution steps from Q-resolution proofs [8, 51] cannot be applied to remove strict reductions from $Q(D^{\mathrm{rrs}})$-resolution refutations. If a long-distance resolution step leads to a clause containing a universal variable u in both polarities, one can assume that the pivot variable does not block u. In a refutation, the literals blocking u have to be resolved out eventually, so one can remove the long-distance resolution step by successively lowering it [8] or by (recursively) resolving out blocking literals using clauses resolved closer to the root of the derivation [51]. Resolving the clauses $(u \vee e_2)$ and $(\neg u \vee \neg e_2)$ in refutation $\mathcal{T}_1$ would amount to a long-distance resolution step. But e_2 is both the pivot variable and the variable blocking u in the premises, so we cannot further lower this resolution step.

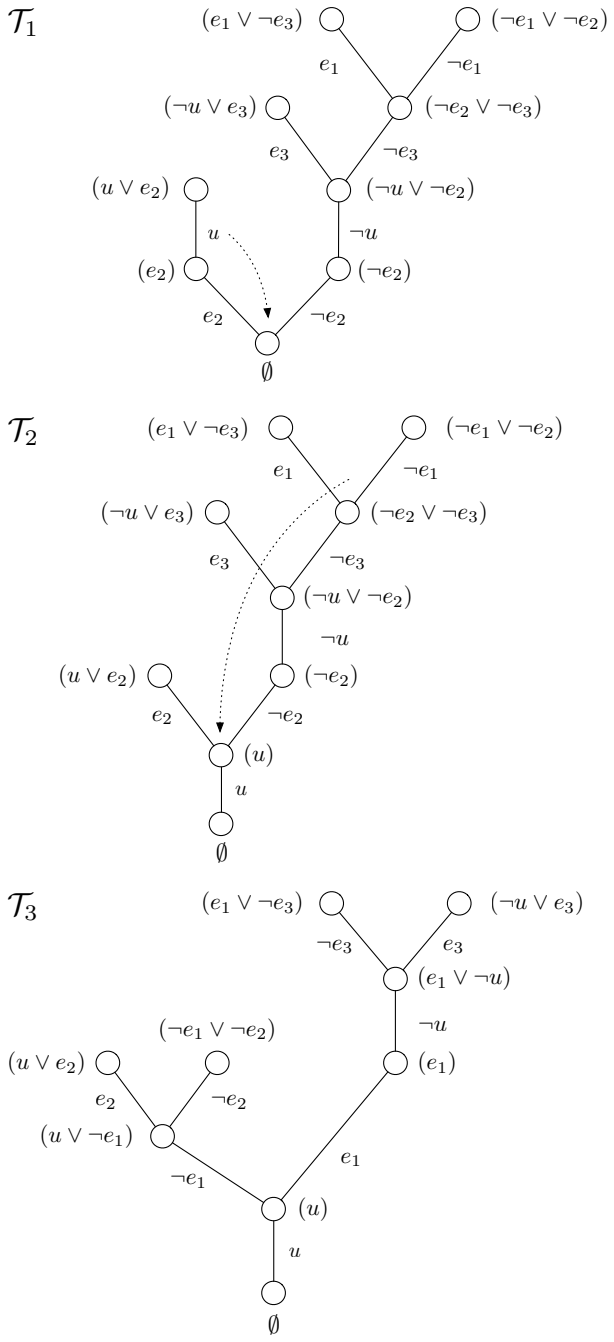

Figure 5.5: Rewriting a $Q(D^{\text{rrs}})$-resolution refutation into a Q-resolution refutation (Example 2).

$\mathcal{T}$ such that $\mathcal{T}[\pi] = \star\|_\ell$ and the conclusion of $\mathcal{T}[\pi]$ contains an existential literal ℓ' such that $(\ell, \ell') \in R_\Phi$, then $\mathcal{T}[\pi]$ is a *strict* $\forall(D)$*-reduction* of $\mathcal{T}$ (on literal ℓ, with respect to Φ). If $\mathcal{T}[\pi] = \star\|_\ell$ is a strict $\forall(D)$-reduction of $\mathcal{T}$ and $\mathcal{T}$ does not contain a strict $\forall(D)$-reduction on a literal ℓ' such that $(\ell', \ell) \in R_\Phi$, then $\mathcal{T}[\pi]$ is an *outermost* strict $\forall(D)$-reduction of $\mathcal{T}$.

Rewriting a derivation may cause literals to disappear from its conclusion, so that resolution or $\forall$-reduction steps with the original conclusion as a premise may become inapplicable. To suppress explicit case distinctions needed for situations of this kind we introduce "lazy" versions of the $\odot$ and $\|$ operations as follows (cf. [62]). Let $\mathcal{T}_1, \mathcal{T}_2$, and $\mathcal{T}$ be $Q(D)$-derivations of clauses C_1, C_2, and C. We define

$$\mathcal{T}_1 \odot_\ell^L \mathcal{T}_2 = \begin{cases} \mathcal{T}_1 \odot_\ell \mathcal{T}_2 & \text{if } \mathcal{T}_1 \odot_\ell \mathcal{T}_2 \text{ is defined,} \\ \mathcal{T}_1 & \text{otherwise, if } \ell \notin C_1, \\ \mathcal{T}_2 & \text{otherwise, if } \bar{\ell} \notin C_2; \end{cases}$$

$$\mathcal{T}\|_\ell^L = \begin{cases} \mathcal{T}\|_\ell & \text{if } \mathcal{T}\|_\ell \text{ is defined,} \\ \mathcal{T} & \text{otherwise, if } \ell \notin C; \end{cases}$$

In order to describe Algorithm 3 and prove its correctness (and termination) we are going to define and characterize the following two rewriting operations:

1. Substitution (Definition 30 and Lemma 11).
2. Dropping a resolution step (Algorithm 2 and Lemma 15).

The second operation can in turn be represented by a successive "lowering" of a resolution step (Algorithm 1, Lemmas 13 and 14).[5]

For the remainder of this subsection, let D be an arbitrary, but fixed proto-dependency scheme, and let Φ be an arbitrary, but fixed PCNF formula. We would like to make statements to the effect that rewriting operations do not create "new" strict reductions. To make this idea formal we introduce the following relations on derivations.

Definition 28. Let $\mathcal{T}_1 \approx \star\|_\ell$ and $\mathcal{T}_2 \approx \star\|_\ell$ be $Q(D)$-resolution derivations from Φ with conclusions C_1 and C_2, respectively. The derivation $\mathcal{T}_1$ *ℓ-subsumes* $\mathcal{T}_2$, $\mathcal{T}_1 \subseteq_\ell \mathcal{T}_2$ in symbols, if every literal in C_1 that blocks ℓ is contained in C_2. We say that $\mathcal{T}_1$ *subsumes* $\mathcal{T}_2$ and write $\mathcal{T}_1 \subseteq \mathcal{T}_2$ whenever $C_1 \subseteq C_2$.

Definition 29. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be $Q(D)$-resolution derivations from Φ. We write $\mathcal{T}_1 \preceq \mathcal{T}_2$ if, for every position α of $\mathcal{T}_1$ such that $\mathcal{T}_1[\alpha] \approx \star\|_\ell$, there exists a position β of $\mathcal{T}_2$ such that $\mathcal{T}_2[\beta] \approx \star\|_\ell$ and $\mathcal{T}_1[\alpha] \subseteq_\ell \mathcal{T}_2[\beta]$.

The relation $\preceq$ defines a preorder on $Q(D)$-resolution derivations from Φ with the following properties.

[5] This lowering operation essentially corresponds to the rewrite rules presented in [8] for turning long-distance resolution proofs into ordinary Q-resolution proofs, the only difference being that the (easy) cases covered in lines 5–8 of Algorithm 1 are omitted in [8].

Lemma 10. *Let $\mathcal{T}$ and $\mathcal{S}$ be $Q(D)$-resolution derivations from Φ.*

(1) $\mathcal{T}[\alpha] \preceq \mathcal{T}$ for every position α of $\mathcal{T}$.

(2) If $\mathcal{T} \approx \mathcal{T}_1 \odot_\ell \mathcal{T}_2$, $\mathcal{T}_1 \preceq \mathcal{S}$, and $\mathcal{T}_2 \preceq \mathcal{S}$, then $\mathcal{T} \preceq \mathcal{S}$.

(3) If $\mathcal{S} \approx \mathcal{S}'\|_\ell$ such that $\mathcal{S}' \preceq \mathcal{T}$ and there exists a position α of $\mathcal{T}$ such that $\mathcal{S} \subseteq_\ell \mathcal{T}[\alpha]$ then $\mathcal{S} \preceq \mathcal{T}$.

(4) If $\mathcal{T}[\alpha] \approx \star\|_\ell$ is a strict reduction and $\mathcal{T} \preceq \mathcal{S}$ then there is a position β of $\mathcal{S}$ such that $\mathcal{S}[\beta] \approx \star\|_\ell$ is a strict reduction.

Proof.

(1) The statement follows from the observation that $\mathcal{T}[\alpha][\beta] = \mathcal{T}[\alpha\beta]$ for any position β of $\mathcal{T}[\alpha]$.

(2) If $\mathcal{T}[\alpha] \approx \star\|_{\ell'}$ then $\alpha \neq \varepsilon$ and hence $\alpha = \ell\beta$ or $\overline{\ell}\beta$ for some position β. Assume without loss of generality that $\alpha = \ell\beta$. Since $\mathcal{T}_1 \preceq \mathcal{S}$ there must be a position γ of $\mathcal{S}$ such that $\mathcal{S} \approx \star\|_{\ell'}$ and $\mathcal{T}[\alpha] = \mathcal{T}_1[\beta] \subseteq_{\ell'} \mathcal{S}[\gamma]$.

(3) Let $\mathcal{S}[\gamma] \approx \star\|_\ell$. There are two cases. If $\gamma = \varepsilon$ then $\mathcal{S}[\varepsilon] \subseteq_\ell \mathcal{T}[\alpha]$. Otherwise, $\gamma = \ell\beta$ for some position β. Then $\mathcal{S}[\gamma] = \mathcal{S}'[\beta]$ and since $\mathcal{S}' \preceq \mathcal{T}$ there must be a position δ of $\mathcal{T}$ such that $\mathcal{T}[\delta] \approx \star\|_\ell$ and $\mathcal{S}'[\beta] \subseteq_\ell \mathcal{T}[\delta]$.

(4) Let $\mathcal{T}[\alpha] \approx \star\|_\ell$ and let C be the premise of the corresponding $\forall(D)$-reduction. If $\mathcal{T} \preceq \mathcal{S}$ then there is a position β of $\mathcal{S}$ such that $\mathcal{S}[\beta] \approx \star\|_\ell$ and $\mathcal{T}[\alpha] \subseteq_\ell \mathcal{S}[\beta]$. Since $\mathcal{T}[\alpha]$ is a strict reduction there must be a literal $\ell' \in C$ that blocks ℓ. By $\mathcal{T}[\alpha] \subseteq_\ell \mathcal{S}[\beta]$ the literal ℓ' must also be contained in the premise of $\mathcal{S}[\beta]$. That is, $\mathcal{S}[\beta]$ is a strict reduction.

□

Definition 30 (Substitution)**.** Let $\mathcal{T}$ and $\mathcal{S}$ be $Q(D)$-resolution derivations from Φ. Given a position α of $\mathcal{T}$ we define $\mathcal{T}[\alpha \leftarrow \mathcal{S}]$ as follows.

$$\mathcal{T}[\alpha \leftarrow \mathcal{S}] = \begin{cases} \mathcal{S} & \text{if } \alpha = \varepsilon, \\ \mathcal{T}_1[\gamma \leftarrow \mathcal{S}] \odot_\ell^L \mathcal{T}_2 & \text{if } \mathcal{T} = \mathcal{T}_1 \odot_\ell \mathcal{T}_2 \text{ and } \alpha = \ell\gamma, \\ \mathcal{T}'[\gamma \leftarrow \mathcal{S}]\|_\ell^L & \text{if } \mathcal{T} = \mathcal{T}'\|_\ell \text{ and } \alpha = \ell\gamma; \end{cases}$$

Let $\mathcal{T}$ be a $Q(D)$-resolution derivation from Φ, let α be a position of $\mathcal{T}$, and let ℓ be a literal. We say that $\mathcal{T}$ *does not contain ℓ below α* if, for every proper prefix γ of α, the conclusion of $T[\gamma]$ does not contain ℓ.

Lemma 11. *Let $\mathcal{T}$ be a $Q(D)$-resolution derivation from Φ of a clause C such that $\mathcal{T}[\alpha] \approx \mathcal{T}'\|_\ell$. If $\mathcal{T}$ does not contain $\overline{\ell}$ below α then $\mathcal{T}[\alpha \leftarrow \mathcal{T}']$ is a $Q(D)$-resolution derivation of a clause $C' \subseteq C \cup \ell$ and $\mathcal{T}[\alpha \leftarrow \mathcal{T}'] \preceq \mathcal{T}$.*

Proof. The derivation $\mathcal{T}[\alpha \leftarrow \mathcal{T}']$ simply omits the $\forall(D)$-reduction step on ℓ at position α, introducing ℓ to clauses on the path from α to the root of the derivation (not necessarily all the way to the root, as there may be another $\forall(D)$-reduction step on that may remove ℓ). By assumption, $\mathcal{T}$ does not contain $\overline{\ell}$ below α, so the result will be a $Q(D)$-resolution derivation. □

Lemma 12. *Let $\mathcal{T}$ and $\mathcal{S}$ be $Q(D)$-resolution derivations from Φ. Let α be a position of $\mathcal{T}$ such that such that $\mathcal{S} \subseteq \mathcal{T}[\alpha]$ and $\mathcal{S} \preceq T[\alpha]$. Then $\mathcal{T}[\alpha \leftarrow \mathcal{S}] \subseteq \mathcal{T}$ and $\mathcal{T}[\alpha \leftarrow \mathcal{S}] \preceq \mathcal{T}$.*

Proof. The proof is by induction on the length of α. For $\alpha = \varepsilon$ the lemma holds trivially. Assume it holds for positions of length up to k and let $\alpha = \ell\beta$ be a position of length $k + 1$. We distinguish two cases.

1. If $\mathcal{T} \approx \mathcal{T}_1 \odot_\ell \mathcal{T}_2$ then $\mathcal{T}[\alpha \leftarrow \mathcal{S}] \approx \mathcal{T}_1' \odot_\ell^L \mathcal{T}_2$, where $\mathcal{T}_1' \approx \mathcal{T}_1[\beta \leftarrow S]$. By induction hypothesis $\mathcal{T}_1'$ is a $Q(D)$-resolution derivation such that $\mathcal{T}_1' \subseteq \mathcal{T}_1$ and $\mathcal{T}_1' \preceq \mathcal{T}_1$. It follows that $\mathcal{T}_1' \odot_\ell^L \mathcal{T}_2 \subseteq \mathcal{T}$. Moreover, $\mathcal{T}_1' \odot_\ell^L \mathcal{T}_2 \preceq \mathcal{T}$ holds by Lemma 10 (2).

2. If $\mathcal{T} \approx \mathcal{T}'\|_\ell$ then $\mathcal{T}[\alpha \leftarrow \mathcal{S}] \approx \mathcal{T}'[\beta \leftarrow \mathcal{S}]\|_\ell^L$. By induction hypothesis $\mathcal{T}'[\beta \leftarrow \mathcal{S}]$ is a $Q(D)$-resolution derivation such that (a) $\mathcal{T}'[\beta \leftarrow \mathcal{S}] \subseteq \mathcal{T}'$ and (b) $\mathcal{T}'[\beta \leftarrow \mathcal{S}] \preceq \mathcal{T}'$. Using the induction hypothesis, it follows from (a) that $\mathcal{T}'[\beta \leftarrow \mathcal{S}]\|_\ell^L$ is a $Q(D)$-resolution derivation such that $\mathcal{T}'[\beta \leftarrow \mathcal{S}]\|_\ell^L \subseteq \mathcal{T}$. To prove the second part of the statement, we distinguish two cases. Suppose $\mathcal{T}[\alpha \leftarrow \mathcal{S}] \approx \mathcal{T}'[\beta \leftarrow \mathcal{S}]\|_\ell^L \approx \mathcal{T}'[\beta \leftarrow \mathcal{S}]$, that is, the literal ℓ is not contained in the conclusion of $\mathcal{T}'[\beta \leftarrow \mathcal{S}]$. We have $\mathcal{T}' \preceq \mathcal{T}$ by Lemma 10 (1) and $\mathcal{T}'[\beta \leftarrow \mathcal{S}] \preceq \mathcal{T}'$ by (b), so $\mathcal{T}[\alpha \leftarrow \mathcal{S}] \preceq \mathcal{T}$ by transitivity of $\preceq$. Otherwise, the literal ℓ is actually reduced and $\mathcal{T}[\alpha \leftarrow \mathcal{S}] \approx \mathcal{T}'[\beta \leftarrow \mathcal{S}]\|_\ell^L \approx \mathcal{T}'[\beta \leftarrow \mathcal{S}]\|_\ell$. Since $\mathcal{T}'[\beta \leftarrow \mathcal{S}]\|_\ell^L \subseteq \mathcal{T}$ (proved above), in particular $\mathcal{T}'[\beta \leftarrow \mathcal{S}]\|_\ell \subseteq_\ell \mathcal{T}$; moreover, $\mathcal{T}' \preceq \mathcal{T}$ by Lemma 10 (1) and $\mathcal{T}'[\beta \leftarrow \mathcal{S}] \preceq \mathcal{T}'$ by (b), so $\mathcal{T}'[\beta \leftarrow \mathcal{S}] \preceq \mathcal{T}$ by transitivity of $\preceq$. Applying Lemma 10 (3), we conclude that $\mathcal{T}'[\beta \leftarrow \mathcal{S}]\|_\ell \preceq \mathcal{T}$.

□

For the lowering operation (Algorithm 1), we distinguish two cases based on whether the resolution step is lowered past a $\forall(D)$-reduction step (Lemma 13) or another resolution step (Lemma 14).

Lemma 13. *Let $\mathcal{T} \approx (\star \odot_a \star)\|_b$ be a $Q(D)$-resolution derivation from Φ such that a does not block b. Then $\texttt{lower}(\mathcal{T}, b)$ is a $Q(D)$-resolution derivation from Φ such that $\texttt{lower}(\mathcal{T}, b) \subseteq \mathcal{T}$ and $\texttt{lower}(\mathcal{T}, b) \preceq \mathcal{T}$.*

Proof. It is readily verified that $\texttt{lower}(\mathcal{T}, b)$ is a $Q(D)$-derivation with the same conclusion as $\mathcal{T}$, so $\texttt{lower}(\mathcal{T}, b) \subseteq \mathcal{T}$. It may contain new $\forall(D)$-reduction steps $\mathcal{T}_1\|_b$ and $\mathcal{T}_2\|_b$, but since a does not block b we have $T_1\|_b \subseteq_b (\mathcal{T}_1 \odot_a \mathcal{T}_2)\|_b$ as well as $\mathcal{T}_2\|_b \subseteq_b (\mathcal{T}_1 \odot_a \mathcal{T}_2)\|_b$ and we conclude that $\texttt{lower}(\mathcal{T}, b) \preceq \mathcal{T}$. □

Function `lower` $(\mathcal{T}, b)$
  input: A $Q(D)$-resolution derivation $\mathcal{T}$ and a literal b.
  if $\mathcal{T} \approx (\mathcal{T}_1 \odot_a \mathcal{T}_2) \odot_b \mathcal{T}_3$ **then**
    let C_i *be the conclusion of* $\mathcal{T}_i$ *for* $i \in \{1, 2, 3\}$
    if $a \notin C_3$ *and* $\overline{a} \notin C_3$ **then**
      return $(\mathcal{T}_1 \odot_b^L \mathcal{T}_3) \odot_a (\mathcal{T}_2 \odot_b^L \mathcal{T}_3)$
    else if $a \in C_3$ **then**
      return $\mathcal{T}_1 \odot_b^L \mathcal{T}_3$
    else
      return $\mathcal{T}_2 \odot_b^L \mathcal{T}_3$
  else if $\mathcal{T} \approx (\mathcal{T}_1 \odot_a \mathcal{T}_2)\|_b$ **then**
    return $\mathcal{T}_1\|_b^L \odot_a \mathcal{T}_2\|_b^L$
  else
    return $\mathcal{T}$

Algorithm 1: Lowering a resolution step.

Lemma 14. *Let* $\mathcal{T} \approx (\mathcal{T}_1 \odot_a \mathcal{T}_2) \odot_b \mathcal{T}_3$ *be a* $Q(D)$*-resolution derivation from* Φ*. Then* $\mathtt{lower}(\mathcal{T}, b)$ *is a* $Q(D)$*-resolution derivation such that* $\mathtt{lower}(\mathcal{T}, b) \subseteq \mathcal{T}$ *and* $\mathtt{lower}(\mathcal{T}, b) \preceq \mathcal{T}$*.*

Proof. Since every clause of Φ is non-tautological, the conclusion of every subderivation of $\mathcal{T}$ must be non-tautological. Keeping this in mind, it is straightforward to check that $\mathtt{lower}(\mathcal{T}, b)$ is a $Q(D)$-resolution derivation from Φ and $\mathtt{lower}(\mathcal{T}, b) \subseteq \mathcal{T}$. Moreover, we have $\mathcal{T}_i \preceq \mathcal{T}$ for $i \in \{1, 2, 3\}$ by Lemma 10 (1). In combination with Lemma 10 (2) this yields $\mathtt{lower}'(\mathcal{T}, b) \preceq \mathcal{T}$. □

Function `drop` $(\mathcal{T}, \alpha, a)$
  input: A $Q(D)$-resolution derivation $\mathcal{T}$, a position α of $\mathcal{T}$, and a literal a.
  if $\alpha = \varepsilon$ **then**
    return $\mathcal{T}$
  else if $\alpha = b\beta$ **then**
    $\mathcal{S} := \mathtt{drop}(\mathcal{T}[b], \beta, a)$
    $\mathcal{S}' := \mathcal{T}[b \leftarrow \mathcal{S}]$
    if $\mathcal{S} \approx \star \odot_a \star$ **then**
      return $\mathtt{lower}(\mathcal{S}', b)$
    else
      return $\mathcal{S}'$

Algorithm 2: "Dropping" a resolution step.

Let $\mathcal{T}$ be a $Q(D)$-resolution derivation from Φ, let a be an existential literal, and let α be a position of $\mathcal{T}$. We say that a *does not block in* $\mathcal{T}$ *below* α if, for every prefix β

of α, whenever $\mathcal{T}[\beta] = \star\|_b$ for some literal b, then a does not block b. If a literal removed by an application of resolution does not block below the resolution step, we can can "drop" the resolution step by repeatedly lowering it (Algorithm 2).

Lemma 15. *Let $\mathcal{T}$ be a $Q(D)$-resolution derivation from Φ, and let $\mathcal{T}[\alpha] \approx \star \odot_a \star$ such that a does not block in $\mathcal{T}$ below α. Then $\mathcal{T}' = \mathtt{drop}(\mathcal{T}, \alpha, a)$ is a $Q(D)$-resolution derivation from Φ such that $\mathcal{T}' \subseteq \mathcal{T}$, $\mathcal{T}' \preceq \mathcal{T}$, and such that at least one of the following conditions holds.*

1. *$\mathcal{T}' \approx \mathcal{T}_1 \odot_a \mathcal{T}_2$ and $|\mathcal{T}_1| < |\mathcal{T}|$ as well as $|\mathcal{T}_2| < |\mathcal{T}|$*
2. *$|\mathcal{T}'| < |\mathcal{T}|$*

Proof. We proceed by induction on the length of α. If $\alpha = \varepsilon$ then $\mathcal{T}' \approx \mathcal{T}$ and the lemma holds. Now suppose $\alpha = b\beta$. By induction hypothesis $\mathcal{S} = \mathtt{drop}(\mathcal{T}[b], \beta, a)$ is a $Q(D)$-resolution derivation such that $\mathcal{S} \subseteq \mathcal{T}[b]$ and $\mathcal{S} \preceq \mathcal{T}[b]$. We can apply Lemma 12 to conclude that $\mathcal{S}' \subseteq \mathcal{T}$ and $\mathcal{S}' \preceq \mathcal{T}$, where $\mathcal{S}' = \mathcal{T}[b \leftarrow \mathcal{S}]$. We now distinguish two cases. If $\mathcal{S} \neq \star \odot_a \star$ then $\mathcal{S}$ must satisfy condition 2 by induction hypothesis, for it cannot satisfy condition 1. That is, $|\mathcal{S}| < |T[b]|$. It follows that $|\mathcal{T}'| = |\mathcal{S}| < |\mathcal{T}|$ and $\mathcal{T}'$ satisfies condition 2 as required. Moreover, $\mathcal{S} \neq \star \odot_a \star$ implies $\mathcal{T}' = \mathcal{S}'$ and so $\mathcal{T}' \subseteq \mathcal{T}$ and $\mathcal{T}' \preceq \mathcal{T}$ as claimed. Otherwise, $\mathcal{S} \approx \mathcal{S}_1 \odot_a \mathcal{S}_2$ and $|\mathcal{S}_1| < |\mathcal{T}[b]|$ as well as $|\mathcal{S}_2| < |\mathcal{T}[b]|$ by induction hypothesis. Suppose $\mathcal{S}' \approx \mathcal{S}\|_b$. Since a does not block in $\mathcal{T}$ below α it follows from Lemma 13 that $\mathtt{lower}(\mathcal{S}', b) = \mathcal{S}_1\|_b^L \odot_a \mathcal{S}_2\|_b^L$ is a $Q(D)$-resolution derivation such that $\mathtt{lower}(\mathcal{S}', b) \subseteq \mathcal{T}$ and $\mathtt{lower}(\mathcal{S}', b) \preceq \mathcal{T}$. Moreover, we must have $\mathcal{T} = \mathcal{T}[b]\|_b$, so that $|\mathcal{T}| = |\mathcal{T}[b]| + 1$. Since $|\mathcal{S}_i| < |\mathcal{T}[b]|$ and $|\mathcal{S}_i\|_b^L| \leq |\mathcal{S}_i| + 1$ this yields $|\mathcal{S}_i\|_b^L| < |\mathcal{T}|$ for $i \in \{1, 2\}$, and condition 2 is satisfied. Otherwise, $\mathcal{S}' \approx \mathcal{S} \odot_b \mathcal{T}''$, where $\mathcal{T}'' = \mathcal{T}[\bar{b}]$. Then $\mathcal{T}' = \mathtt{lower}(\mathcal{S}', b)$ is a $Q(D)$-resolution derivation such that $\mathcal{T}' \subseteq \mathcal{T}$ and $\mathcal{T}' \preceq \mathcal{T}$ by Lemma 14. To verify that one of the two conditions above is satisfied we have to consider the three possible outcomes of the case distinction in lines 3 to 9 of the algorithm. If $\mathcal{T}' = (\mathcal{S}_1 \odot_b^L \mathcal{T}'')$ or $\mathcal{T}' = (\mathcal{S}_2 \odot_b^L \mathcal{T}'')$ condition 2 is satisfied since $|\mathcal{S}_i| < |\mathcal{T}[b]|$ for $i \in \{1, 2\}$ and $|\mathcal{T}| = 1 + |\mathcal{T}[b]| + |\mathcal{T}''|$; otherwise, $\mathcal{T}' = (\mathcal{S}_1 \odot_b^L \mathcal{T}'') \odot_a (\mathcal{S}_2 \odot_b^L \mathcal{T}'')$, and condition 1 is satisfied. □

We now establish a correspondence between the structure of $Q(D)$-resolution derivations and resolution paths. More specifically, we will show that under certain conditions, literals appearing in a $Q(D)$-resolution derivation $\mathcal{T}$ are connected through resolution paths via the set $\mathit{resvar}(\mathcal{T})$ of variables appearing as pivot variables in $\mathcal{T}$. The following result captures the key insight about this correspondence (cf. [120, 107]).

Lemma 16. *Let $\mathcal{T}$ be a $Q(D)$-resolution derivation of a clause C from Φ. If $a, b \in C$ are distinct literals there is a resolution path from a to b via $\mathit{resvar}(\mathcal{T})$.*

Proof. We proceed by induction on the height of $\mathcal{T}$. If $\mathcal{T}$ has height 0 then C is a clause of Φ and the sequence ab is a resolution path. Suppose the claim holds for derivations of height up to k and let $\mathcal{T}$ have height $k+1$. There are two cases. If $\mathcal{T} \approx \mathcal{T}'\|_\ell$ then a and b must already be contained in the conclusion C' of $\mathcal{T}'$. Because $\mathcal{T}'$ has height k, we can

Function `normalize`$(\Phi, \mathcal{T})$
  input: A PCNF formula Φ and a $Q(D^{\text{rrs}})$-derivation $\mathcal{T}$ from Φ.
  if $\mathcal{T}$ *does not contain a strict reduction* **then**
    return $\mathcal{T}$
  else if $\mathcal{T} \approx \mathcal{S}\|_a$ **then**
    return $\texttt{normalize}(\Phi, \mathcal{S})\|_a^L$
  else if $\mathcal{T} \approx \mathcal{T}_1 \odot_a \mathcal{T}_2$ **then**
    let $\mathcal{T}[\alpha] \approx \mathcal{T}'\|_b$ *be an outermost strict reduction of* $\mathcal{T}$
    if $\mathcal{T}$ *does not contain* $\overline{b}$ *below* α **then**
      $\mathcal{S} := \mathcal{T}[\alpha \leftarrow \mathcal{T}']$
      return $\texttt{normalize}(\Phi, \mathcal{S})\|_b^L$
    else
      let γ *be a shortest position such that* $\mathcal{T}[\gamma] \approx \star \odot_c \star$ *and* $(c, b) \in R_\Phi$
      $\mathcal{S} := \texttt{drop}(\mathcal{T}, \gamma, c)$
      if $\mathcal{S} \approx \mathcal{S}_1 \odot_c \mathcal{S}_2$ **then**
        return $\texttt{normalize}(\Phi, \mathcal{S}_1) \odot_c^L \texttt{normalize}(\Phi, \mathcal{S}_2)$
      else
        return $\texttt{normalize}(\Phi, \mathcal{S})$

Algorithm 3: Converting $Q(D^{\text{rrs}})$-derivations to Q-resolution derivations.

apply the induction hypothesis to conclude that there is a resolution path from a to b via $\mathit{resvar}(\mathcal{T}') = \mathit{resvar}(\mathcal{T})$. Otherwise, $\mathcal{T} \approx \mathcal{T}_1 \odot_\ell \mathcal{T}_2$. Let C_1 and C_2 denote the conclusions of $\mathcal{T}_1$ and $\mathcal{T}_2$. Assume without loss of generality that $a \in C_1$ and $b \in C_2$. Since $\ell \in C_1$ and $\overline{\ell} \in C_2$ and $\mathcal{T}_1$ and $\mathcal{T}_2$ both have height at most k, we can apply the induction hypothesis to conclude that there must be a resolution path p' from a to ℓ via $\mathit{resvar}(\mathcal{T}_1)$ and a resolution path p'' from $\overline{\ell}$ to b via $\mathit{resvar}(\mathcal{T}_2)$. By Lemma 4, the sequence $p'p''$ is a resolution path from a to b via $\mathit{resvar}(\mathcal{T}_1) \cup \mathit{resvar}(\mathcal{T}_2) \cup \{\text{var}(\ell)\} = \mathit{resvar}(\mathcal{T})$. □

Lemma 17. *Let $\mathcal{T}$ be a $Q(D)$-resolution derivation of clause C from Φ. Let α be a position of $\mathcal{T}$ and let C' be the conclusion of $\mathcal{T}[\alpha]$. If there are literals a, b such that $a \neq b$, $b \in C$, $a \in C'$, and $a \notin \mathit{resvar}(\mathcal{T})$, then there exists a resolution path p from a to b via $\mathit{resvar}(\mathcal{T})$. Moreover, if $b \notin C'$ then p goes through an existential variable $e \in \text{var}(C') \cap \mathit{resvar}(\mathcal{T})$.*

Proof. The proof is by induction on the length of α. If $\alpha = \varepsilon$ then $a, b \in C$ and we apply Lemma 16 to obtain the desired resolution path. Since $b \in C' = C$ this concludes the proof the base case. Now let $\alpha = c\beta$. If c is a universal literal then $\mathcal{T} \approx \mathcal{T}'\|_c$ and the conclusion of $\mathcal{T}'$ must already contain b. Moreover, the conclusion C'' of $\mathcal{T}'[\beta] = \mathcal{T}[\alpha]$ contains a and β is obviously shorter than α. We apply the induction hypothesis to obtain the desired resolution path p from a to b via $\mathit{resvar}(\mathcal{T}') = \mathit{resvar}(\mathcal{T})$. Otherwise, c is an existential literal and $\mathcal{T} \approx \mathcal{T}_1 \odot_c \mathcal{T}_2$. Let C_1 and C_2 denote the conclusions of $\mathcal{T}_1$ and $\mathcal{T}_2$. If $b \in C_1$ we can again apply the induction hypothesis to obtain the desired

resolution path. Otherwise, we have $b \in C_2$. Since $a \notin \mathit{resvar}(\mathcal{T})$ we also have $a \neq c$. Moreover, $c \in C_1$. By induction hypothesis, there is a resolution path p' from a to c via $\mathit{resvar}(\mathcal{T})$. Because $b \in C$ we also have $\overline{c} \neq b$. Since $\overline{c}, b \in C_2$, we can use Lemma 16 to get a resolution path p'' from $\overline{c}$ to b via $\mathit{resvar}(\mathcal{T}_2)$. By Lemma 4, the sequence $p = p'p''$ is a resolution path from a to b via $\mathit{resvar}(\mathcal{T}_1) \cup \mathit{resvar}(\mathcal{T}_2) \cup \{\mathrm{var}(c)\} = \mathit{resvar}(\mathcal{T})$. If $c \notin C'$ then the resolution path p' must go through an existential variable $e \in \mathrm{var}(C') \cap \mathit{resvar}(\mathcal{T})$ and obviously p goes through e as well. Otherwise, $\mathrm{var}(c) \in C'$ and p goes through $\mathrm{var}(c) \in \mathit{resvar}(\mathcal{T})$ by construction. □

Lemma 18. *Let $\mathcal{T}$ be a $Q(D^{\mathrm{rrs}})$-derivation from Φ with conclusion C. If there is a universal literal $a \in C$ and there is a position α of $\mathcal{T}$ such that $\mathcal{T}[\alpha] \approx \star\|_{\overline{a}}$ then there is a position β of $\mathcal{T}$ such that $\mathcal{T}[\beta] \approx \star \odot_b \star$ and $(b, a) \in R_\Phi$.*

Proof. Assume for a contradiction that there is a universal literal $a \in C$ and a position α such that $\mathcal{T}[\alpha] \approx \mathcal{T}'\|_{\overline{a}}$ but there is no position β of $\mathcal{T}$ such that $\mathcal{T}[\beta] \approx \star \odot_b \star$ and $(b, a) \in R_\Phi$. Since $\overline{a} \neq a$, $a \in C$, and $\overline{a} \in C'$, where C' is the conclusion of $\mathcal{T}'$, we can apply Lemma 17 to conclude that there is a resolution path p from $\overline{a}$ to a via $\mathit{resvar}(\mathcal{T})$. Moreover, since $a \in C$ we have $\overline{a} \notin C$, so p must go through an existential variable $e \in \mathrm{var}(C') \cap \mathit{resvar}(\mathcal{T})$. That is, $p = \overline{a}, \ldots, \ell_e, \overline{\ell_e}, \ldots, a$ for some literal $\ell_e \in \{e, \neg e\}$. By Lemma 4, the sequences $p' = \overline{a}, \ldots, \ell_e$ and $p'' = \overline{\ell_e}, \ldots, a$ are resolution paths of Φ via $\mathit{resvar}(\mathcal{T})$. In other words, the pair $\{\mathrm{var}(a), e\}$ is a resolution-path dependency pair with respect to $\mathit{resvar}(\mathcal{T})$. Since there is no position β of $\mathcal{T}$ such that $\mathcal{T}[\beta] \approx \star \odot_b \star$ and $(b, a) \in R_\Phi$ we have $\mathit{resvar}(\mathcal{T}) \subseteq R_\Phi(a) \setminus \mathrm{var}_\forall(\Phi)$. It follows that $(\mathrm{var}(a), e) \in D^{\mathrm{rrs}}_\Phi$. But $\mathcal{T}[\alpha]$ is a $\forall(D)$-reduction step and thus $(\mathrm{var}(a), e) \notin D^{\mathrm{rrs}}_\Phi$, a contradiction. □

For the next lemma, we generalize the notion of an existential literal blocking a universal literal to a *clause C blocking a universal literal b*. By this we simply mean that C contains a literal blocking u. Moreover, will speak of a literal (or clause) as *blocking a $\forall(D)$-reduction step*, meaning that the literal (or clause) blocks the literal removed by the $\forall(D)$-reduction.

Lemma 19. *Let $\mathcal{T}$ be a $Q(D^{\mathrm{rrs}})$-derivation of C from Φ such that C does not block a strict reduction of $\mathcal{T}$. Then* `normalize`$(\Phi, \mathcal{T})$ *(see Algorithm 3) is a Q-resolution derivation of size at most $3^{|T|}$ and* `normalize`$(\Phi, \mathcal{T}) \subseteq \mathcal{T}$.

Proof. By induction on the size of $\mathcal{T}$. If $\mathcal{T}$ has only one vertex it cannot contain a strict reduction and the algorithm simply returns $\mathcal{T}$ (line 2). Suppose the lemma holds for derivations of size strictly less than $|\mathcal{T}|$. If $\mathcal{T}$ does not contain a strict reduction it is already a Q-resolution derivation (line 2). Otherwise, if $\mathcal{T} \approx \mathcal{S}\|_a$ (line 5) we can apply the induction hypothesis to conclude that the derivation $\mathcal{S}' =$ `normalize`$(\Phi, \mathcal{S})$ is a Q-resolution derivation from Φ such that `normalize`$(\Phi, \mathcal{S}) \subseteq \mathcal{S}$. It follows that `normalize`$(\Phi, \mathcal{T}) \approx$ `normalize`$(\Phi, \mathcal{S})\|^L_a$ is a Q-resolution derivation and `normalize`$(\Phi, \mathcal{T}) \subseteq \mathcal{T}$. Suppose $\mathcal{T} \approx \mathcal{T}_1 \odot_a \mathcal{T}_2$ (line 6). Let α be a position such that $\mathcal{T}[\alpha] \approx \mathcal{T}'\|_b$ is an outermost strict reduction of $\mathcal{T}$ (line 7). There are two cases. (a) If $\mathcal{T}$ does not contain $\overline{b}$ below α (line 8) then it follows from Lemma 11 that $\mathcal{S} = \mathcal{T}[\alpha \leftarrow \mathcal{T}']$ is a $Q(D^{\mathrm{rrs}})$-resolution

derivation of a clause $C' \subseteq C \cup b$ and $\mathcal{S} \preceq \mathcal{T}$. By assumption, C does not block a strict reduction of $\mathcal{T}$. Since the clause C' only contains an additional universal literal, it does not block a strict reduction of $\mathcal{T}$, so in particular it does not block a strict reduction of $\mathcal{S} \preceq \mathcal{T}$. Moreover, $|\mathcal{S}| < |\mathcal{T}|$, so we can apply the induction hypothesis and conclude that $\texttt{normalize}(\Phi, \mathcal{S})$ is a Q-resolution derivation and $\texttt{normalize}(\Phi, \mathcal{S}) \subseteq \mathcal{S}$. It follows that $\texttt{normalize}(\Phi, \mathcal{T}) = \texttt{normalize}(\Phi, \mathcal{S})\|_b^L$ is a Q-resolution derivation and $\texttt{normalize}(\Phi, \mathcal{T}) \subseteq \mathcal{T}$ (line 10). (b) If $\mathcal{T}$ contains $\overline{b}$ below α then by Lemma 18 there must be a (shortest) position γ of $\mathcal{T}$ such that $\mathcal{T}[\gamma] \approx \star \odot_c \star$ and $(c, b) \in R_\Phi$ (line 12). We claim that c does not block in $\mathcal{T}$ below γ. Towards a contradiction assume that there is a proper prefix β of γ such that $\mathcal{T}[\beta] \approx \star\|_d$ and $(d, c) \in R_\Phi$. Since $\mathcal{T} \approx \star \odot_a \star$ and γ is the shortest position such that $\mathcal{T}[\gamma] \approx \star \odot_c \star$ and $(c, b) \in R_\Phi$, every clause appearing as a label on the path from the root of $\mathcal{T}$ to the conclusion of $\mathcal{T}[\gamma]$ has to contain an existential literal e such that $(b, e) \in R_\Phi$. In particular, the conclusion of $\mathcal{T}[\beta]$ has to contain such a literal e. We thus have $(d, c), (c, b), (b, e) \in R_\Phi$ and $(d, e) \in R_\Phi$ by transitivity of R_Φ. That is, e blocks d and $\mathcal{T}[\beta]$ must be a strict reduction. But $\mathcal{T}[\alpha] \approx \mathcal{T}'\|_b$ is an outermost strict reduction of $\mathcal{T}$ and $(d, b) \in R_\Phi$, a contradiction. We conclude that c does not block in $\mathcal{T}$ below γ. We can therefore apply Lemma 15 to conclude that the derivation $\mathcal{S} = \texttt{drop}(\mathcal{T}, \gamma, c)$ satisfies $\mathcal{S} \subseteq \mathcal{T}$ and $\mathcal{S} \preceq \mathcal{T}$ (line 13). Since $\mathcal{T}'\|_b$ is an outermost strict reduction of $\mathcal{T}$, $\mathcal{S} \preceq \mathcal{T}$, and $(c, b) \in R_\Phi$, neither c nor $\neg c$ block a strict reduction of $\mathcal{S}$. If $\mathcal{S} \approx \mathcal{S}_1 \odot_c \mathcal{S}_2$ then $|\mathcal{S}_1| < |\mathcal{T}|$ and $|\mathcal{S}_2| < |\mathcal{T}|$ by Lemma 15 and the conclusions of $\mathcal{S}_1$ and $\mathcal{S}_2$ do not block strict reductions of $\mathcal{S}$ by choice of c. By induction hypothesis $\texttt{normalize}(\Phi, \mathcal{S}_1)$ and $\texttt{normalize}(\Phi, \mathcal{S}_2)$ are Q-resolution derivations such that $\texttt{normalize}(\Phi, \mathcal{S}_1) \subseteq \mathcal{S}_1$ and $\texttt{normalize}(\Phi, \mathcal{S}_2) \subseteq \mathcal{S}_2$. We conclude that $\texttt{normalize}(\Phi, \mathcal{T}) = \texttt{normalize}(\Phi, \mathcal{S}_1) \odot_c^L \texttt{normalize}(\Phi, \mathcal{S}_2)$ is a Q-resolution derivation and $\texttt{normalize}(\Phi, \mathcal{T}) \subseteq \mathcal{T}$ (line 15). Otherwise, $|\mathcal{S}| < |\mathcal{T}|$ by Lemma 15 and by induction hypothesis $\texttt{normalize}(\Phi, \mathcal{T}) = \texttt{normalize}(\Phi, \mathcal{S})$ is a Q-resolution derivation such that $\texttt{normalize}(\Phi, \mathcal{T}) \subseteq \mathcal{T}$ (line 17).

It remains to prove that $\texttt{normalize}(\Phi, \mathcal{T})$ satisfies the size bound claimed in the statement of the lemma. Again we proceed by induction on the size of $\mathcal{T}$. In the base case $\mathcal{T}$ does not contain a strict reduction so $\texttt{normalize}(\Phi, \mathcal{T}) = \mathcal{T}$ and the bound holds. Suppose the bound holds for every $Q(D^{\text{rrs}})$-resolution derivations of size strictly smaller than $|\mathcal{T}|$.

- If $\mathcal{T}$ does not contain a strict reduction again $\texttt{normalize}(\Phi, \mathcal{T}) = \mathcal{T}$ and the size bound holds.
- Suppose $\mathcal{T} \approx \mathcal{S}\|_a$ and $\texttt{normalize}(\Phi, \mathcal{T}) = \texttt{normalize}(\Phi, \mathcal{S})\|_a^L$. Then by induction hypothesis $|\texttt{normalize}(\Phi, \mathcal{S})| \leq 3^{|\mathcal{S}|}$. Since $|\mathcal{S}| = |\mathcal{T}| - 1$ we obtain

$$\begin{aligned} |\texttt{normalize}(\Phi, \mathcal{S})\|_a^L| &\leq |\texttt{normalize}(\Phi, \mathcal{S})| + 1 \\ &\leq 3^{|\mathcal{S}|} + 1 \\ &= 3^{|\mathcal{T}|-1} + 1 \\ &\leq 3^{|\mathcal{T}|}. \end{aligned}$$

- Let α be a position of $\mathcal{T}$ such that $\mathcal{T}[\alpha] = \mathcal{T}\|_b$ and let $\mathcal{S} = \mathcal{T}[\alpha \leftarrow \mathcal{T}']$. Suppose $\texttt{normalize}(\Phi, \mathcal{T}) = \texttt{normalize}(\Phi, \mathcal{S})\|_b^L$. We have $|\mathcal{S}| \leq |\mathcal{T}| - 1$ and $|\texttt{normalize}(\Phi, \mathcal{S})| \leq 3^{|\mathcal{S}|}$ by induction hypothesis. Overall, we get

$$\begin{aligned} |\texttt{normalize}(\Phi, \mathcal{S})\|_b^L| &\leq |\texttt{normalize}(\Phi, \mathcal{S})| + 1 \\ &\leq 3^{|\mathcal{S}|} + 1 \\ &\leq 3^{|\mathcal{T}|-1} + 1 \\ &\leq 3^{|\mathcal{T}|}. \end{aligned}$$

- Finally, let γ be a position of $\mathcal{T}$ such that $\mathcal{T}[\gamma] \approx \star \odot_c \star$ and assume c does not block in $\mathcal{T}$ below γ. Let $\mathcal{S} = \texttt{drop}(\mathcal{T}, \gamma, c)$. We again consider two cases. (1) If $\mathcal{S} \approx \mathcal{S}_1 \odot_c \mathcal{S}_2$ then by Lemma 15 we have $|\mathcal{S}_i| < |\mathcal{T}|$ for $i \in \{1, 2\}$. By induction hypothesis we have $|\texttt{normalize}(\Phi, \mathcal{S}_i)| \leq 3^{|\mathcal{S}_i|}$ for $i \in \{1, 2\}$ and thus $|\texttt{normalize}(\Phi, \mathcal{S}_i)| < 3^{|\mathcal{T}|}$ for $i \in \{1, 2\}$. As a consequence we get

$$\begin{aligned} |\texttt{normalize}(\Phi, \mathcal{S}_1) \odot_c^L \texttt{normalize}(\Phi, \mathcal{S}_2)| &\leq 3^{|\mathcal{S}_1|} + 3^{|\mathcal{S}_2|} + 1 \\ &\leq 3 \cdot 3^{|\mathcal{T}|-1} \\ &= 3^{|\mathcal{T}|}. \end{aligned}$$

  (2) Otherwise, $|\mathcal{S}| < |\mathcal{T}|$ by Lemma 15. Using the induction hypothesis, we conclude that $|\texttt{normalize}(\Phi, \mathcal{S})| \leq 3^{|\mathcal{S}|} < 3^{|\mathcal{T}|}$.

□

Proposition 7 follows immediately from Lemma 19 and the observation that the empty clause trivially does not block a strict reduction. We can now proceed to prove soundness of $Q(D^{\text{rrs}})$-resolution at last.

Theorem 5. *A PCNF formula Φ is false if and only if there is a $Q(D^{\text{rrs}})$-resolution refutation of Φ.*

Proof. If there is a $Q(D^{\text{rrs}})$-resolution refutation of Φ then by Proposition 7 there is a Q-resolution refutation of Φ. By Theorem 3 that means the formula Φ is false. Conversely, if Φ is false then there is a Q-resolution refutation of Φ by Theorem 3, and this refutation is trivially a $Q(D^{\text{rrs}})$-resolution refutation of Φ. □

Since D^{rrs} is strictly more general than D^{std} this result entails Theorem 4.

5.3 Soundness of Q(D)-term Resolution

To prove soundness of $Q(D^{\text{res}})$-term resolution and $Q(D^{\text{rst}})$-term resolution, we follow a strategy that is dual to the one adopted in proving soundness of $Q(D^{\text{std}})$-resolution and show that these systems preserve a subset of a formula's countermodels.

Definition 31. Let D be a proto-dependency scheme and let Φ be a PCNF formula. A complete countermodel $\mathbf{f}$ of Φ is a *D-countermodel of* Φ if each function $f_u \in \mathbf{f}$ is independent of every existential variable e with $(e, u) \notin D_\Phi$.

The following two results are essentially dual to Lemma 9 and Proposition 5.

Lemma 20. *Let D be a proto-dependency scheme and $\Phi = \mathcal{Q}.\varphi$ a PCNF formula. Let $\mathbf{f}$ be a D-countermodel of Φ and let $\mathcal{S} = T_1, \ldots, T_k$ be a $Q(D)$-term resolution derivation of a term T from Φ. Then $T[\sigma \cup \mathbf{f}(\sigma)] = 0$ for every assignment σ to the existential variables of Φ.*

Proof. We prove the lemma by induction on k. If $k = 1$ then $T_k = T$ must be derived from Φ by the model generation rule, so $T \cap C$ for every $C \in \varphi$. Let $\sigma : \mathrm{var}_\exists(\Phi) \to \{0, 1\}$. Since $\mathbf{f}$ is a countermodel of Φ, there must be a $C \in \varphi$ such that $C[\sigma \cup \mathbf{f}(\sigma)] = 0$. In particular, $(\sigma \cup \mathbf{f}(\sigma))(\ell) = 0$ for every literal $\ell \in C \cap T$, so $T[\sigma \cup \mathbf{f}(\sigma)] = 0$. Suppose the lemma holds for every $k' < k$. There are three cases. (a) If T is derived from Φ by the model generation rule we proceed as in the base case. (b) $T = T_k$ is derived from T_i and T_j by resolution, where $i, j < k$. By induction hypothesis $T_i[\sigma \cup \mathbf{f}(\sigma)] = T_j[\sigma \cup \mathbf{f}(\sigma)] = 0$. Let u be the (universal) pivot variable. Assume without loss of generality that $u \in T_i$ and $\neg u \in T_j$. If $(\sigma \cup \mathbf{f}(\sigma))(u) = 1$ then there is a literal $\ell \in T_i$ with $\ell \neq u$ such that $(\sigma \cup \mathbf{f}(\sigma))(\ell) = 0$. Then $T[\sigma \cup \mathbf{f}(\sigma)] = 0$ because $\ell \in T$. Otherwise, $(\sigma \cup \mathbf{f}(\sigma))(u) = 0$ and there must be a literal $\ell \in T_j$ such that $(\sigma \cup \mathbf{f}(\sigma))(\ell) = 0$. Again, ℓ is contained in T and so $T[\sigma \cup \mathbf{f}(\sigma)] = 0$. (c) $T = T_k$ is derived from $T_i = T_k \wedge \ell$ by $\exists(D)$-reduction, where $i < k$. We distinguish two cases. (a) If $\sigma(\ell) = 1$ then it follows from the induction hypothesis that there is a literal $\ell' \in T_i$ such that $\ell' \neq \ell$ and $(\sigma \cup \mathbf{f}(\sigma))(\ell') = 0$. Then $\ell' \in T$ and $T[\sigma \cup \mathbf{f}(\sigma)] = 0$. (b) Otherwise, $\sigma(\ell) = 0$. Consider the assignment σ' such that $\sigma'(\ell) = 1$ and $\sigma'(e) = \sigma(e)$ for every existential variable $e \neq \mathrm{var}(\ell)$. By an argument parallel to the one for case (a) there must be a literal $\ell' \in T_i$ such that $\ell' \neq \ell$ and $(\sigma' \cup \mathbf{f}(\sigma'))(\ell') = 0$. Let $x = \mathrm{var}(\ell')$. Suppose x is existential. Then $x \neq \mathrm{var}(\ell)$ (since T_i is non-contradictory) and $\sigma(x) = \sigma'(x)$. So $(\sigma \cup \mathbf{f}(\sigma))(\ell') = 0$ and thus $T[\sigma \cup \mathbf{f}(\sigma)] = 0$. Now suppose x is universal. By definition of $\exists(D)$-reduction we must have $(\mathrm{var}(\ell), x) \notin D_\Phi$. Because $\mathbf{f}$ is a D-countermodel of Φ the function f_x must be independent of $\mathrm{var}(\ell)$. It follows that $f_x(\sigma|_{L_x^\Phi}) = f_x(\sigma'|_{L_x^\Phi})$ and thus $(\sigma \cup \mathbf{f}(\sigma))(\ell') = (\sigma' \cup \mathbf{f}(\sigma')(\ell') = 0$, which implies $T[\sigma \cup \mathbf{f}(\sigma)] = 0$. □

Proposition 8. *If D is a proto-dependency scheme such that every false PCNF formula has a D-countermodel, then $Q(D)$-term resolution is sound: a PCNF formula Φ is true if and only if there is a $Q(D)$-term resolution proof of Φ.*

Proof. Let D be a proto-dependency scheme such that every false PCNF formula has a D-countermodel. The empty term is trivially satisfied by any truth assignment, so by Lemma 20 no false PCNF formula can have a $Q(D)$-term resolution proof. This proves the "if" part. The "only if" part follows immediately from Theorem 3 and the fact that every Q-term resolution proof is a $Q(D)$-term resolution proof. □

5.3.1 The Resolution-Path Dependency Scheme

To prove soundness of $Q(D^{\text{res}})$-resolution, we are going to show that every false PCNF formula has a D^{res}-countermodel (Proposition 9). Given a false PCNF formula Φ, we will construct this countermodel from a Q-resolution refutation of Φ. To simplify the construction, we will start from an ordered Q-resolution refutation – we can assume that false formulas have ordered Q-resolution refutations without loss of generality: the refutations used in the original completeness proof of Q-resolution are in fact ordered [24]. We first define a countermodel that is not necessarily complete (Lemma 22). Using a connection between the structure of Q-resolution derivations and resolution-path dependencies established earlier (Lemma 17), we then prove that the model function of a universal variable u in this countermodel has the following property: it takes on the same value for any two assignments that differ only on existential variables that u does not depend on, according to D^{res}_{Φ} (Lemma 23). From there, it is straightforward to extend the countermodel to a complete countermodel that is in fact a D^{res}-countermodel of Φ (Lemma 24).

Given an ordered Q(D)-resolution refutation and an assignment, there is a unique leaf position in the derivation such that every existential literal occurring in this position is falsified by the assignment. To construct a countermodel, our goal is to set universal variables in such a way as to also falsify universal literals occurring in this position.

Definition 32 (Extension)**.** Let Φ be a PCNF formula, let $\mathcal{T}$ be an ordered Q-resolution refutation of Φ, and let π be a position of $\mathcal{T}$. For an assignment $\tau : E \to \{0,1\}$, where $E \subseteq \text{var}_{\exists}(\Phi)$, we define the *$\tau$-extension* of π in $\mathcal{T}$ (with respect to Φ) as the (unique) longest sequence $\rho = \ell_1, \ldots, \ell_k$ of literals such that $\pi\rho$ is a position of $\mathcal{T}$ and such that $\tau(\ell_i) = 0$ or $\text{var}(\ell_i) \in \text{var}_{\forall}(\Phi)$ for each $i \in [k]$.

Lemma 21. *Let Φ be a PCNF formula and let $\mathcal{T}$ be an ordered Q-resolution refutation of Φ. Further, let $\tau : E \to \{0,1\}$ be a truth assignment, where $E \subseteq \text{var}_{\exists}(\Phi)$, and let π be a position of $\mathcal{T}$ such that $\tau(\ell) = 0$ for each existential literal ℓ in the conclusion of $\mathcal{T}[\pi]$. Then $\tau(\ell) = 0$ for each existential literal ℓ in the conclusion of $\mathcal{T}[\pi\rho]$, where ρ is the τ-extension of π in $\mathcal{T}$.*

Proof. Let $\rho = \ell_1, \ldots, \ell_k$, and let C and C' denote the conclusions of $\mathcal{T}[\pi]$ and $\mathcal{T}[\pi\rho]$, respectively. We have $C' \subseteq C \cup \{\ell_1, \ldots, \ell_k\}$ and $\tau(\ell_i) = 0$ for each existential literal ℓ_i, for $i \in [k]$. In combination with the assumption that $\tau(\ell) = 0$ for each existential literal $\ell \in C$, this proves the lemma. □

For the remainder of this subsection let $\Phi = \mathcal{Q}.\varphi$ be an arbitrary, but fixed, false PCNF formula, and let E and U denote its sets of existential and universal variables, respectively. Moreover, let $\mathcal{T}$ be an ordered Q-resolution refutation of Φ.

Definition of the countermodel. We define a family $\mathbf{f} = \{f_u\}_{u \in U}$ of partial functions $f_u : 2^{L_u^\Phi} \to \{0,1\}$ as

$$f_u(\sigma) := \begin{cases} 0 & \text{if } u \text{ occurs in the } \sigma\text{-extension of } \varepsilon \text{ in } \mathcal{T}, \\ 1 & \text{if } \neg u, \text{ but not } u, \text{ occurs in the } \sigma\text{-extension of } \varepsilon \text{ in } \mathcal{T}, \\ \text{undefined} & \text{otherwise;} \end{cases}$$

Lemma 22. *The set* $\mathbf{f}$ *is a countermodel of* Φ*.*

Proof. Let $\sigma : E \to \{0,1\}$ be a truth assignment, let π be the σ-extension of ε in $\mathcal{T}$, and let C denote the conclusion of $\mathcal{T}[\pi]$. Because σ assigns every existential variable the clause C must be an input clause. We claim that $f_u(\sigma|_{L_u^\Phi})$ is defined for each universal variable $u \in \mathrm{var}(C)$ and $C[\sigma \cup \mathbf{f}(\sigma)] = 0$. If C does not contain universal variables this follows from Lemma 21. Otherwise, let $\ell \in C$ be a universal literal. The literal ℓ must occur in π because $\mathcal{T}$ is a refutation. In fact, ℓ must occur in the σ'-extension of ε in $\mathcal{T}$, where $\sigma' = \sigma|_{L_u^\Phi}$, because $\mathcal{T}$ is ordered. We now distinguish two cases. (a) If $\ell = u$ then $f_u(\sigma') = 0$ by construction of f_u. (b) Otherwise, if $\ell = \neg u$ then u cannot occur in the σ'-extension of ε in $\mathcal{T}$, as $\mathcal{T}$ is ordered and each of its positions can contain at most one of the literals u and $\neg u$. It follows that $f_u(\sigma') = 1$ by construction of f_u. We conclude that every universal literal of C is set to 0 by $\mathbf{f}(\sigma)$, so $C[\sigma \cup \mathbf{f}(\sigma)] = 0$ as claimed. □

To show that $\mathbf{f}$ can be extended to a complete D^{res}-countermodel we introduce auxiliary notation: for each $u \in U$ we define a relation $\sim_u$ on the set of assignments with domain L_u^Φ by letting $\sigma \sim_u \tau$ if $\sigma(e) = \tau(e)$ for every existential variable e such that $(e,u) \in D_\Phi^{\mathrm{res}}$. Informally, if $\sigma \sim_u \tau$ the assignments σ and τ agree on every existential variable that u may depend on. Observe that $\sim_u$ is an equivalence relation.

Lemma 23. *Let* $u \in U$ *be a universal variable. If* f_u *is defined for two assignments* $\sigma, \tau : L_u^\Phi \to \{0,1\}$ *and* $\sigma \sim_u \tau$ *then* $f_u(\sigma) = f_u(\tau)$*.*

Proof. Let $\sigma, \tau : L_u^\Phi \to \{0,1\}$ be assignments such that $f_u(\sigma) = 0$ and $f_u(\tau) = 1$. We have to show that $\sigma \nsim_u \tau$. By construction of $\mathbf{f}$, the literal u has to occur on the σ-extension π of ε in $\mathcal{T}$ and $\neg u$ has to be on τ-extension π' of ε in $\mathcal{T}$. Because $\mathcal{T}$ is ordered, every position can contain at most one of the literal u and $\neg u$, so there must be an existential literal ℓ such that ℓ occurs in π but not in π', and such that $\bar{\ell}$ occurs in π' but not in π. Consider the prefix ρ of π ending with the literal ℓ. Let $\mathcal{T}' = \mathcal{T}[\rho]$, let C denote the conclusion of $\mathcal{T}[\rho]$, and let C' denote the conclusion of $\mathcal{T}[\pi]$. We have $u \neq \ell$, $\ell \in C$, $u \in C'$, and $u \notin \mathit{resvar}(\mathcal{T})$, so by Lemma 17 the literal ℓ and u are connected in Φ via $\mathit{resvar}(\mathcal{T}')$. Let $e = \mathrm{var}(\ell)$. Because $\mathcal{T}$ is ordered we have $\mathit{resvar}(\mathcal{T}') \subseteq R_\Phi(e) \setminus (\mathrm{var}_\forall(\Phi) \cup \{e,u\})$. A parallel argument shows that $\bar{\ell}$ and $\neg u$ are connected in Φ via $R_\Phi(e) \setminus (\mathrm{var}_\forall(\Phi) \cup \{e,u\})$. Combining these statements yields $(e,u) \in D_\Phi^{\mathrm{res}}$. By choice of ℓ we have $\sigma(\ell) = 0$ and $\tau(\bar{\ell}) = 0$, whence $\sigma(e) \neq \tau(e)$. That is, $(e,u) \in D_\Phi^{\mathrm{res}}$ but $\sigma(e) \neq \tau(e)$, so $\sigma \nsim_u \tau$. □

We now construct a family of functions $\mathbf{g} = \{g_u\}_{u \in U}$ which extends $\mathbf{f}$ to a complete countermodel of Φ. For every $u \in U$ and $\sigma : L_u^\Phi \to \{0,1\}$ we define

$$g_u(\sigma) = \begin{cases} f_u(\tau) & \text{if there is a } \tau \sim_u \sigma \text{ and } f_u(\tau) \text{ is defined,} \\ 1 & \text{otherwise;} \end{cases}$$

Lemma 24. *The set* $\mathbf{g}$ *is a* D^{res}*-countermodel of* Φ*.*

Proof. It is immediate from Lemma 22 that $\mathbf{g}$ is a countermodel of Φ, and $\mathbf{g}$ is complete by construction. Let $u \in U$ and $e \in E$ such that $(e,u) \notin D_\Phi^{\mathrm{res}}$. Towards a contradiction, let $\sigma, \tau : L_u^\Phi \to \{0,1\}$ be truth assignments such that $\sigma(y) = \tau(y)$ for each $y \in L_u^\Phi \setminus \{e\}$, but $g_u(\sigma) = 0$ and $g_u(\tau) = 1$. By construction of $\mathbf{g}$ there has to be an assignment $\sigma' : L_u^\Phi$ such that $\sigma' \sim_u \sigma$ and $g_u(\sigma) = f_u(\sigma') = 0$. Note that $\sigma \sim_u \tau$ and recall that $\sim_u$ is an equivalence relation. It follows that $\sigma' \sim_u \tau$. Since $g_u(\tau) = 1$ this implies that there has to be another truth assignment τ' such that $\tau' \sim_u \tau$ and $g_u(\tau) = f_u(\tau') = 1$. But $\sigma' \sim_u \tau'$ by transitivity of $\sim_u$, so we can apply Lemma 23 to conclude that $f_u(\sigma') = f_u(\tau')$, a contradiction. □

Since Φ was chosen arbitrarily, we obtain the following result.

Proposition 9. *Every false PCNF formula has a* D^{res}*-countermodel.*

Theorem 6. *A PCNF formula* Φ *is true if and only if there is a* $Q(D^{\mathrm{res}})$*-term resolution proof of* Φ*.*

Proof. Immediate from Proposition 8 in combination with Proposition 9. □

By Proposition 1 the resolution-path dependency scheme is more general than the standard dependency scheme, so we obtain the following result as a corollary.

Corollary 1. *A PCNF formula* Φ *is true if and only if there is a* $Q(D^{\mathrm{std}})$*-term resolution proof of* Φ*.*

5.3.2 The Refined Standard Dependency Scheme

Recall that the refined standard dependency scheme and the resolution-path dependency scheme are incomparable (Proposition 1). In particular, there are formulas Φ and pairs $(x,y) \in \mathrm{var}_\exists(\Phi) \times \mathrm{var}_\forall(\Phi)$ such that $(x,y) \in D_\Phi^{\mathrm{res}}$ but $(x,y) \notin D_\Phi^{\mathrm{rst}}$, so soundness of $Q(D^{\mathrm{res}})$-term resolution does not imply soundness of $Q(D^{\mathrm{rst}})$-term resolution.

To prove soundness of $Q(D^{\mathrm{rst}})$-term resolution we will use the same proof strategy as in the previous subsection and show that every false PCNF formula has a D^{rst}-countermodel. The starting point for constructing this countermodel is the canonical countermodel extraction algorithm from Q-resolution proofs [8]. This algorithm defines a model function f_u for a universal variable u roughly as follows: for each application of the $\forall$-reduction rule that removes a literal ℓ with $\mathrm{var}(\ell) = u$, it makes sure that f_u maps ℓ to false whenever the remaining literals in the premise of the $\forall$-reduction step evaluate to false.

For instance, if $\forall$-reduction is applied to a clause $(e \vee u)$, where e is an existential variable, then the algorithm ensures that $f_u(\tau) = 0$ whenever $\tau(e) = 0$, for an assignment τ to existential variables. Since "ordinary" $\forall$-reduction can be applied only if a clause contains no blocking existential variable, the definition of f_u may use the values assigned to existential variables occurring in the premise. The fact that the premise may contain another universal variable v makes things slightly more complicated in that the behavior of f_v and f_u must be coordinated. But this coordination can always be achieved due to the nesting of quantifier scopes in the prefix: if $u \in R(v)$ then f_u can "see" the assignment passed to f_v as an argument and infer the value of f_v; otherwise, $v \in R(u)$ and f_v can infer the value of f_u. In constructing a D^{rst}-countermodel from a Q-resolution refutation we cannot use the same coordination strategy since there may be an existential variable e such that (e, v) is in the refined standard dependency relation but (e, u) is not, or vice versa.

In order to concisely define our D^{rst}-countermodel, we introduce additional notation. Let Φ be a PCNF formula, let $\mathcal{T}$ be a Q-resolution refutation of Φ, and let π be a position of $\mathcal{T}$.

- We let $\mathit{clause}_{\mathcal{T}}(\pi)$ denote the conclusion of $\mathcal{T}[\pi]$, and write $\exists\text{-}\mathit{clause}_{\mathcal{T}}(\pi)$ for its restriction to existential literals, that is
$$\exists\text{-}\mathit{clause}_{\mathcal{T}}(\pi) = \mathit{clause}_{\mathcal{T}}(\pi) \cap (\mathrm{var}_{\exists}(\Phi) \cup \overline{\mathrm{var}_{\exists}(\Phi)}).$$
- Moreover, we define $\forall\text{-}\mathit{prefixes}_{\Phi}(\pi)$ as the set of prefixes of π that end in a universal literal. Formally,
$$\forall\text{-}\mathit{prefixes}_{\Phi}(\pi) = \{\, \sigma\ell : \sigma\ell \text{ is a prefix of } \pi \text{ and } \mathrm{var}(\ell) \in \mathrm{var}_{\forall}(\Phi) \,\}.$$
- Finally, we let
$$\forall\text{-}\mathit{premises}_{\mathcal{T},\Phi}(\pi) = \bigcup_{\rho \in \forall\text{-}\mathit{prefixes}_{\Phi}(\pi)} \exists\text{-}\mathit{clause}_{\mathcal{T}}(\rho).$$

Lemma 25. *Let Φ be a PCNF formula and let $\mathcal{T}$ be an ordered Q-resolution refutation of Φ. Let $\pi = \sigma\ell_u$ be a position of $\mathcal{T}$ such that $u \in \mathrm{var}_{\forall}(\Phi)$, where $u = \mathrm{var}(\ell_u)$. Then $\mathrm{var}(\forall\text{-}\mathit{premises}_{\mathcal{T},\Phi}(\pi)) \subseteq \overline{D^{\mathrm{rst}}_{\Phi}(u)}$.*

Proof. Let $e \in \mathrm{var}(\forall\text{-}\mathit{premises}_{\mathcal{T},\Phi}(\pi))$. Then there is a prefix $\rho \in \forall\text{-}\mathit{prefixes}_{\Phi}(\pi)$ of π and a literal ℓ_e with $\mathrm{var}(\ell_e) = e$ such that $e \in \mathrm{var}(\mathit{clause}_{\mathcal{T}}(\rho))$. Let $\mathcal{T}' = \mathcal{T}[\rho]$. Because u is universal we have $u \notin \mathit{resvar}(\mathcal{T}')$ and $\ell_e \neq \ell_u$. Thus we can apply Lemma 17 to conclude that the literals ℓ_u and ℓ_e are connected in Φ via $\mathit{resvar}(\mathcal{T}')$. It now follows from Lemma 3 that Φ contains an (e, u)-dependency pair with respect to $\mathit{resvar}(\mathcal{T}')$. We claim that $\mathit{resvar}(\mathcal{T}') \subseteq R^{\diamond}_{\Phi}(e) \setminus \mathrm{var}_{\forall}(\Phi)$. Obviously $\mathit{resvar}(\mathcal{T}') \cap \mathrm{var}_{\forall}(\Phi) = \emptyset$, so we only have to show that $\mathit{resvar}(\mathcal{T}') \subseteq R^{\diamond}_{\Phi}(e)$. Since $\rho \in \forall\text{-}\mathit{prefixes}_{\Phi}(\pi)$ there is a position γ of $\mathcal{T}$ and a literal ℓ_v (not necessarily distinct from ℓ_u) such that $\rho = \gamma\ell_v$ and $v \in \mathrm{var}_{\forall}(\Phi)$, where $\mathrm{var}(\ell_v) = v$. Since $\mathcal{T}$ is a refutation ℓ_e has to be resolved out eventually, and because $\mathcal{T}$ is ordered we have $(e, v) \in R_{\Phi}$ as well as $\mathit{resvar}(\mathcal{T}') \subseteq R_{\Phi}(v)$. That is, $\mathit{resvar}(\mathcal{T}') \subseteq R^{\diamond}_{\Phi}(e)$ and the claim holds, so $(e, u) \in D^{\mathrm{rst}}_{\Phi}$. □

For the remainder of this subsection, let Φ denote a fixed PCNF formula and let $\mathcal{T}$ be a fixed, ordered Q-resolution refutation of Φ. To construct a D^{rst}-countermodel, we define a total order $<_{\mathcal{T}}$ on the set of positions of $\mathcal{T}$. Let $\Phi = Q_1 x_1 \dots Q_n x_n.\varphi$. We encode each position π of $\mathcal{T}$ as an n-digit ternary numeral in the following way. For $i \in [n]$, the ith digit of the numeral associated with a position π is 0 if π does not contain an x_i-literal, 1 if π contains the literal x_i, and 2 if π contains the literal $\neg x_i$. For each position π, let $\mathit{value}(\pi)$ be the value of the numeral associated with π. We define the relation $<_{\mathcal{T}}$ on positions of $\mathcal{T}$ by letting

$$\pi <_{\mathcal{T}} \rho \iff \mathit{value}(\pi) < \mathit{value}(\rho).$$

This relation is a total order on the set of positions of $\mathcal{T}$. Additionally, it has the following property.

Lemma 26. *Let $\pi_1\ell_1$ and $\pi_2\ell_2$ be positions of $\mathcal{T}$ such that $\operatorname{var}(\ell_1) = \operatorname{var}(\ell_2)$ and $\pi_1\ell_1 <_{\mathcal{T}} \pi_2\ell_2$. If $\pi_1\ell_1\pi_1'$ and $\pi_2\ell_2\pi_2'$ are positions of $\mathcal{T}$ then $\pi_1\ell_1\pi_1' <_{\mathcal{T}} \pi_2\ell_2\pi_2'$.*

Definition of the countermodel. We define a family $\mathbf{f} = \{f_u\}_{u \in \operatorname{var}_\forall(\Phi)}$ of model functions for the universal variables of Φ. For each $u \in \operatorname{var}_\forall(\Phi)$ and $\tau : L_u^\Phi \to \{0,1\}$ we define $f_u(\tau)$ as follows. Let

$$\mathit{positions}(u) = \{\, \pi : \pi = \sigma\ell \text{ is a position of } \mathcal{T} \text{ and } \operatorname{var}(\ell) = u \,\}.$$

That is, $\mathit{positions}(u)$ is the set of positions of Φ followed by a $\forall$-reduction step involving variable u. The function f_u first computes the set Π_τ defined as

$$\Pi_\tau = \{\, \pi \in \mathit{positions}(u) : \forall\text{-}\mathit{premises}_{\mathcal{T},\Phi}(\pi)[\tau] = 0 \,\}.$$

If $\pi \in \Pi_\tau$ and π' is a prefix of π such that $\mathcal{T}[\pi']$ is a $\forall$-reduction step, then every existential literal in the premise of this $\forall$-reduction step is mapped to 0 by τ. Using the set Π_τ, we define f_u as follows.

- If $\Pi_\tau = \emptyset$ we let $f_u(\tau|_{L_u^\Phi}) = 0$.
- Otherwise, if $\pi\ell \in \Pi_\tau$ is the smallest (with respect to $<_{\mathcal{T}}$) position in Π_τ, then

$$f_u(\tau|_{L_u^\Phi}) = \begin{cases} 0 & \text{if } \ell = u, \\ 1 & \text{otherwise;} \end{cases}$$

By Lemma 25 the set Π_τ only depends on the restricted assignment $\tau|_X$, where $X = \overline{D_\Phi^{\text{rst}}}(u)$, so the function f_u is independent of variables in $\operatorname{var}_\exists(\Phi) \setminus \overline{D_\Phi^{\text{rst}}}(u)$ by construction.

Lemma 27. *The set $\mathbf{f}$ is a D^{rst}-countermodel of Φ.*

Proof. We first show that $\mathbf{f}$ is a countermodel of Φ. Assume towards a contradiction that there is an assignment $\tau : var_\exists(\Phi) \to \{0,1\}$ such that $\tau \cup \mathbf{f}(\tau)$ satisfies φ. Let π be the smallest (with respect to $<_\mathcal{T}$) leaf position of $\mathcal{T}$ such that $\tau(\ell) = 0$ for each existential literal $\ell \in \mathit{clause}_\mathcal{T}(\pi)$ and such that $\forall\text{-}\mathit{premises}_{\mathcal{T},\Phi}(\pi)[\tau] = 0$. The τ-extension of ε in $\mathcal{T}$ satisfies this condition by Lemma 21, so such a position π must exist. By assumption, the clause $C \in \varphi$ is satisfied by $\tau \cup \mathbf{f}(\tau)$, so there must be a universal literal $\ell' \in C$ such that $\mathbf{f}(\tau)(\ell') = 1$. Let $u = \mathrm{var}(\ell')$. There has to be a position $\rho \in \mathit{positions}(u)$ such that ρ is a prefix of π, since $\mathcal{T}$ is a refutation and the literal ℓ' has to be removed by $\forall$-reduction eventually. By choice of π we have $\forall\text{-}\mathit{premises}_{\mathcal{T},\Phi}(\rho)[\tau] = 0$. Since ℓ' evaluates to 1, by construction of f_u there has to be another position $\rho' \in \mathit{positions}(u)$ such that $\rho' <_\mathcal{T} \rho$ and $\forall\text{-}\mathit{premises}_{\mathcal{T},\Phi}(\rho')[\tau] = 0$. Let α be the τ-extension of ρ' in $\mathcal{T}$, and let $\pi' = \rho'\alpha$. It follows from Lemma 21 that $\tau(\ell) = 0$ for each existential literal $\ell \in \mathit{clause}_\mathcal{T}(\pi')$ and that $\forall\text{-}\mathit{premises}_{\mathcal{T},\Phi}(\pi')[\tau] = 0$. Since $\rho, \rho' \in \mathit{positions}(u)$, $\rho' <_\mathcal{T} \rho$, and ρ' and ρ are prefixes of π' and π, respectively, we can apply Lemma 26 to conclude that $\pi' <_\mathcal{T} \pi$, a contradiction. So there cannot be an assignment $\tau : var_\exists(\Phi) \to \{0,1\}$ such that $\tau \cup \mathbf{f}(\tau)$ satisfies φ and $\mathbf{f}$ is a countermodel of Φ. Moreover, $\mathbf{f}$ is complete and each function $f_u \in \mathbf{f}$ is independent of variables in $\mathrm{var}_\exists(\Phi) \setminus \overline{D^{\mathrm{rst}}_\Phi}(u)$ by construction, so $\mathbf{f}$ is even a D^{rst}-countermodel of Φ. □

Proposition 10. *Every false PCNF formula has a complete D^{rst}-countermodel.*

Theorem 7. *A PCNF formula Φ is true if and only if there is a $Q(D^{\mathrm{rst}})$-term resolution proof of Φ.*

Proof. Immediate from Proposition 8 in combination with Proposition 10. □

Summary

Motivated by the use of dependency schemes in the PCNF solver DepQBF, we introduced and studied two families of resolution proof systems for PCNF formulas that take a dependency scheme D as a parameter: Q(D)-resolution, a generalization of Q-resolution, and Q(D)-term resolution, a generalization of Q-term resolution. We gave soundness proofs for several instances of these proof systems:

- We showed that $Q(D^{\mathrm{rst}})$-resolution and $Q(D^{\mathrm{rst}})$-term resolution are sound. These are the proof systems used for proof generation by DepQBF in its current version [82]. These results provide the first rigorous argument for correctness of the algorithm implemented by this solver.
- Beyond that, we proved soundness of $Q(D^{\mathrm{rrs}})$-resolution and $Q(D^{\mathrm{res}})$-term resolution. Since these systems are, respectively, stronger than $Q(D^{\mathrm{rst}})$-resolution and incomparable to $Q(D^{\mathrm{rst}})$-term resolution, the implementation of these dependency schemes could lead to performance gains in future incarnations of DepQBF.

CHAPTER 6

Quantifier Reordering

Dependency schemes can be used to manipulate the order of variables in the prefix of a PCNF formula while preserving its truth value. In fact, the notion of a dependency scheme was originally defined with this application in mind [102]. This definition requires only that dependency schemes support an operation called quantifier *shifting* (see Section 6.1). Here, we will consider proto-dependency schemes that support a more general operation: every reordering that does not change the relative order of pairs of variables occurring in the dependency relation must be truth value-preserving. Calling dependency schemes of this kind *permutation dependency schemes*, we prove the following results:

1. The resolution-path dependency scheme D^{res} is a permutation dependency scheme (Theorem 9).

2. The refined standard dependency scheme D^{rst} is a permutation dependency scheme (Theorem 10).

Because every proto-dependency scheme that is less general than a permutation dependency scheme is a permutation dependency scheme as well (Proposition 13), this entails that every proto-dependency scheme introduced in Chapter 3 is a permutation dependency scheme.

As an application, we will combine permutation dependency schemes with a linear-time algorithm that computes a quantifier prefix with the smallest number of quantifier alternations among reorderings compatible with a given dependency relation. The number of quantifier alternations of a PCNF formula is an indicator of hardness for the associated instance of QSAT, as witnessed by the following results:

a) QSAT is complete for the kth level of the polynomial hierarchy when restricted to PCNF formulas with k alternations [112].

b) Without a bound on quantifier alternations, QSAT is PSPACE-hard even for formulas of bounded pathwidth [4].

c) QSAT is fixed-parameter tractable if both the treewidth and the number of quantifier alternations are taken as parameters, but the function bounding the complexity in the parameters is a tower of exponentials with height corresponding to the number of quantifier alternations (unless P=NP) [90].

Paired with a tractable permutation dependency scheme, the alternation minimization algorithm leads to a polynomial-time preprocessing procedure that may reduce the number of quantifier alternations of a PCNF formula:

3. Let D be a tractable permutation dependency scheme. For every PCNF formula Φ, a reordering Ψ of Φ satisfying the following two properties can be computed in polynomial time: (A) Ψ is compatible with the dependency relation D_Φ, and (B) among reorderings with this property, Ψ has the fewest quantifier alternations (Theorem 11).

The example in Section 5.1 proves that Q(D)-resolution need not be sound for a permutation dependency scheme D. In Section 6.3, we prove that the converse holds, however:

4. Let D be a proto-dependency scheme such that $Q(D)$-resolution and $Q(D)$-term resolution are sound. Then D is a permutation dependency scheme (Proposition 15).

6.1 Permutation Dependency Schemes

For the purposes of this chapter, dependency relations can be interpreted as sets of constraints on the relative order of variables: a pair (x, y) in the dependency relation indicates that changing the relative order of x and y in the quantifier prefix may change the truth value of the formula. We consider formulas obtained from a given PCNF formula by reordering the quantifier prefix while observing these constraints.

Definition 33. Let D be a proto-dependency scheme, and let $\Phi = \mathcal{Q}.\varphi$ and $\Psi = \mathcal{Q}'.\varphi$ be PCNF formulas such that $\mathcal{Q}'$ is a permutation of $\mathcal{Q}$. If $D_\Phi \subseteq R_\Psi$ we call Ψ a *D-permutation* of Φ. If, in addition, $\mathcal{Q}'$ is a transposition of $\mathcal{Q}$ then Ψ is a *D-transposition* of Φ.

Definition 34 (Permutation Dependency Scheme)**.** A *permutation dependency scheme* (*transposition dependency scheme*) is a proto-dependency scheme D such that $\Phi \equiv \Psi$ for every PCNF formula Φ and every D-permutation (D-transposition) Ψ of Φ.

Example 3. Consider the PCNF formula $\Phi = \mathcal{Q}.\varphi$, where

$$\mathcal{Q} = \forall x_1 \exists y_1 \forall x_2 \exists y_2 \forall x_3 \exists y_3, \text{ and}$$
$$\varphi = (\neg x_1 \vee y_1) \wedge (\neg y_1 \vee x_2) \wedge (\neg x_2 \vee y_2) \wedge (\neg y_2 \vee x_3) \wedge (\neg x_3 \vee y_3).$$

The trivial dependency relation of this formula is

$$\{(x_1, y_1), (x_1, y_2), (x_1, y_3), (x_2, y_2), (x_2, y_3), (x_3, y_3),\\ (y_1, x_2), (y_1, x_3), (y_2, x_3)\},$$

so the only D^{trv}-permutation of Φ is the formula Φ itself. On the other hand, for the resolution-path dependency scheme the dependency relation D^{res}_Φ is empty and the formula $\forall x_1 \forall x_2 \forall x_3 \exists y_1 \exists y_2 \exists y_3 . \varphi$ is an example of a D^{res}-permutation of Φ.

Permutation dependency schemes are a special case of so-called *cumulative dependency schemes* which are defined in terms of quantifier *shifting* [102].

Definition 35 (Shifting)**.** Let $\Phi = \mathcal{Q}.\varphi$ be a PCNF formula and $X \subseteq \mathrm{var}(\Phi)$. The PCNF formula Ψ is obtained from Φ by *(down)-shifting* X, in symbols $\Psi = S^{\downarrow}(\Phi, X)$, if $\Psi = \mathcal{Q}'.\varphi$ and $\mathcal{Q}'$ is a permutation of $\mathcal{Q}$ such that the following conditions hold:

1. $X = R_\Psi(x)$ for some $x \in \mathrm{var}(\Phi) = \mathrm{var}(\Psi)$,
2. $(x, y) \in R_\Psi$ if and only if $(x, y) \in R_\Phi$ for all $x, y \in X$, and
3. $(x, y) \in R_\Psi$ if and only if $(x, y) \in R_\Phi$ for all $x, y \in \mathrm{var}(\Phi) \setminus X$.

Definition 36 (Dependency scheme)**.** A proto-dependency scheme D is a *dependency scheme* if $\Phi \equiv S^{\downarrow}(\Phi, D^*_\Phi(x))$ for every PCNF formula Φ and every $x \in \mathrm{var}(\Phi)$.

Definition 37 (Cumulative)**.** A dependency scheme D is *cumulative* if the equivalence $\Phi \equiv S^{\downarrow}(\Phi, D^*_\Phi(X))$ holds for every PCNF formula Φ and $X \subseteq \mathrm{var}(\Phi)$.

Cumulative dependency schemes were originally introduced in context with *backdoor sets* of PCNF formulas [102]. A strong backdoor set of a PCNF formula is a set of variables with the following property: for every assignment to the variables in this set, the remaining formula belongs to a certain class (for example, the class of PCNF formulas with Horn matrices). A strong backdoor set to a tractable PCNF class can be used algorithmically by instantiating the input formula with every possible assignment of the backdoor variables and solving the remaining (tractable) instance in polynomial time. If the backdoor set is (upward) closed under the dependency relation computed by a cumulative dependency scheme, we can first shift the corresponding variables towards the front of the quantifier prefix and this evaluation strategy is sound. If these backdoor sets can be found efficiently, this leads to a fast decision algorithm for PCNF formulas with small backdoor sets.

Proposition 11. *Every permutation dependency scheme is a cumulative dependency scheme.*

Proof. Let D be a permutation dependency scheme and Φ a PCNF formula. Let $X \subseteq \mathrm{var}(\Phi)$ and let $\Psi = S^{\downarrow}(\Phi, D^*_\Phi(X))$. We claim that Ψ is a D-permutation of Φ. To prove this, we have to show that $(x, y) \in R_\Psi$ whenever $(x, y) \in D_\Phi$. Let $(x, y) \in D_\Phi$. If

$x \in D^*_\Phi(X)$ then also $y \in D^*_\Phi(X)$ and so $(x, y) \in R_\Psi$ by Condition 2 of Definition 35. Suppose $x \notin D^*_\Phi(X)$. If $y \notin D^*_\Phi(X)$ then $(x, y) \in R_\Psi$ by Condition 3. If $y \in D^*_\Phi(X)$ then there is a $z \in \mathrm{var}(\Phi)$ such that $D^*_\Phi(X) = R_\Psi(z)$ by Condition 1. We have $x \notin R_\Psi(z)$ and thus $(x, z) \in R_\Psi$. Because R_Ψ is transitive and $(z, y) \in R_\Psi$ this implies $(x, y) \in R_\Psi$. This proves the claim. Since D is a permutation dependency scheme it follows that $\Psi \equiv \Phi$. So D is a cumulative dependency scheme. □

Every permutation dependency scheme is trivially a transposition dependency scheme. Even though transpositions are a very restricted subclass of permutations, a converse can be proved in presence of the following property.

Definition 38 (Monotone)**.** A proto-dependency scheme D is *monotone* if it satisfies the following condition for every PCNF formula Φ, every D-permutation Ψ of Φ, and every $x \in \mathrm{var}(\Phi)$: if $R_\Psi(x) \subseteq R_\Phi(x)$ then $D_\Psi(x) \subseteq D_\Phi(x)$.

Proposition 12. *Every monotone transposition dependency scheme is a permutation dependency scheme.*

Proof. Let D be a monotone transposition dependency scheme and let $\Phi = \mathcal{Q}.\varphi$ and $\Psi = \mathcal{Q}'.\varphi$ be PCNF formulas such that Ψ is a D-permutation of Φ. The prefix $\mathcal{Q}'$ can be obtained from $\mathcal{Q}$ by sorting subsequences of increasing length according to the order in $\mathcal{Q}'$, as follows. Assuming that the quantifier/variable pairs $Q_i x_i$ up to $Q_n x_n$ are already sorted according to $\mathcal{Q}'$, we insert the pair $Q_{i-1}x_{i-1}$ at the correct position (according to $\mathcal{Q}'$) by moving it to the right. Let $\mathcal{Q}_i$ denote the prefix where $Q_i x_i$ up to $Q_n x_n$ are already sorted according to $\mathcal{Q}'$, let $\Phi_i = \mathcal{Q}_i.\varphi$ be the corresponding PCNF formula, and let $R_i = R_{\Phi_i}$. Formally, $\mathcal{Q}_i$ is the unique permutation of $\mathcal{Q}$ such that $R_i = R'_i \cup R''_i \cup R'''_i$, where

$$
\begin{aligned}
R'_i &= R_\Phi \cap (\{x_1, \ldots, x_{i-1}\} \times \{x_1, \ldots, x_{i-1}\}), \\
R''_i &= R_\Psi \cap (\{x_i, \ldots, x_n\} \times \{x_i, \ldots, x_n\}), \text{ and} \\
R'''_i &= \{x_1, \ldots, x_{i-1}\} \times \{x_i, \ldots, x_n\}.
\end{aligned}
$$

That is, the first part of $\mathcal{Q}_i$ agrees with $\mathcal{Q}$, but the second part is ordered according to $\mathcal{Q}'$. We now show that $\Phi_i \equiv \Phi_{i-1}$ for every $1 < i \leq n$. Since $\Phi_n = \Phi$ and $\Phi_1 = \Psi$, the proposition then follows. So pick any i such that $1 < i \leq n$. We can obtain $\mathcal{Q}_{i-1}$ from $\mathcal{Q}_i$ by moving $Q_{i-1}x_{i-1}$ to the right as far as necessary, and this operation can in turn be represented as a sequence of transpositions where $Q_{i-1}x_{i-1}$ is moved one position to the right. Suppose k is the number of transpositions needed. For $0 \leq j \leq k$, let $\mathcal{Q}'_j$ be the prefix where $Q_{i-1}x_{i-1}$ has been moved j positions to the right in $\mathcal{Q}_i$, let $\mathcal{H}_j = \mathcal{Q}'_j.\varphi$, and let $R'_j = R_{\mathcal{H}_j}$. We claim that $\mathcal{H}_j \equiv \mathcal{H}_{j+1}$ for $0 \leq j < k$. Let $0 \leq j < k$, and let y be the variable immediately to the right of x_{i-1} in $\mathcal{Q}'_j$, such that y trades places with x_{i-1} in $\mathcal{Q}'_{j+1}$. Then $(y, x_{i-1}) \in R_\Psi$ and so $(x_{i-1}, y) \notin D_\Phi$ because Ψ is a D-permutation of Φ. By construction of $\mathcal{Q}'_j$ we have $R'_j(x_{i-1}) \subseteq R_i(x_{i-1})$, and by construction of $\mathcal{Q}_i$ we have $R_i(x_{i-1}) = R_\Phi(x_{i-1})$. So $R'_j(x_{i-1}) \subseteq R_\Phi(x_{i-1})$, and since D is monotone, this implies $D'_j(x_{i-1}) \subseteq D_\Phi(x_{i-1})$. In particular, $(x_{i-1}, y) \notin D_{\mathcal{H}_j}$. It follows from D being

a transposition dependency scheme that $\mathcal{H}_j \equiv \mathcal{H}_{j+1}$. This proves the claim. Because $\mathcal{H}_0 = \Phi_i$ and $\mathcal{H}_k = \Phi_{i-1}$, we conclude that $\Phi_i \equiv \Phi_{i-1}$. □

We now apply this result to prove that the resolution-path dependency scheme is a permutation dependency scheme. The following result is due to Van Gelder [120, Theorem 4.7]. Since he offers two definitions of the resolution-path dependency scheme and the theorem only holds for one of them (see Example 4), we include our own proof below.

Theorem 8. *The resolution-path dependency scheme is a transposition dependency scheme.*

Proof. Let $\Phi = \mathcal{Q}.\varphi$ and let $\Psi = \mathcal{Q}'.\varphi$ be a D^{res}-transposition of Φ such that

$$\mathcal{Q} = Q_1 x_1 \dots Q_{i-1} x_{i-1} Q_i x_i Q_{i+1} x_{i+1} \dots Q_n x_n, \text{ and}$$
$$\mathcal{Q}' = Q_1 x_1 \dots Q_{i-1} x_{i-1} Q_{i+1} x_{i+1} Q_i x_i \dots Q_n x_n.$$

We have to prove that Φ is false if and only if Ψ is false. We will assume that x_i is universal and x_{i+1} is existential and show that Φ is false if Ψ is false (the remaining cases are immediate). It is sufficient to show that for every truth assignment $\tau : \{x_1, \dots, x_{i-1}\} \to \{0, 1\}$ the formula $\Phi[\tau]$ has a Q-resolution refutation if $\Psi[\tau]$ has a Q-resolution refutation; here, $\Phi[\tau]$ and $\Psi[\tau]$ denote the PCNF formulas obtained from Φ and Ψ, respectively, by applying the assignment τ to the matrix φ and omitting redundant variables from the quantifier prefixes. Let $\tau : \{x_1, \dots, x_{i-1}\} \to \{0, 1\}$ be an assignment and let $\mathcal{T}$ be a Q-resolution refutation of $\Psi[\tau]$. Assume without loss of generality that $\mathcal{T}$ is ordered. The only derivation step admissible in $\mathcal{T}$ that cannot occur in a refutation of $\Phi[\tau]$ is $\forall$-reduction that removes variable x_i from a clause that contains x_{i+1} or $\neg x_{i+1}$. If $\mathcal{T}$ contains no such step, $\mathcal{T}$ is already a refutation of $\Phi[\tau]$ and we are done. Otherwise, because $\mathcal{T}$ is ordered, the final derivation step has to be a resolution step on variable x_{i+1}, that is, $\mathcal{T} \approx \mathcal{T}_1 \odot_{\ell_{i+1}} \mathcal{T}_2$ for a literal ℓ_{i+1} with $\text{var}(\ell_{i+1}) = x_{i+1}$. If the derivation $\mathcal{T}_j$ contains a $\forall$-reduction step on ℓ_i with $\text{var}(\ell_i) = x_i$, we must have $\mathcal{T}_j \approx \mathcal{T}'_j \| _{\ell_i}$ since $\mathcal{T}$ is ordered, for $j \in \{1, 2\}$. To show that we can get rid of this $\forall$-reduction, we have to consider two cases.

1. If $\mathcal{T}_1 \approx \mathcal{T}'_1 \| \ell_i$ such that $\text{var}(\ell_i) = x_i$ but neither $\mathcal{T}_2 \approx \star \| _{\ell_i}$ nor $\mathcal{T}_2 \approx \star \| _{\overline{\ell_i}}$ then $\mathcal{T}' = (\mathcal{T}'_1 \odot_{\ell_{i+1}} \mathcal{T}_2) \| _{\ell_i}$ is a Q-resolution refutation of $\Phi[\tau]$.
2. Otherwise, if $\mathcal{T}_1 \approx \mathcal{T}'_1 \| \ell_i$ such that $\text{var}(\ell_i) = x_i$ and $\mathcal{T}_2 \approx \mathcal{T}'_2 \| _{\ell_i}$ then $\mathcal{T}' = (\mathcal{T}'_1 \odot_{\ell_{i+1}} \mathcal{T}'_2) \| _{\ell_i}$ is a Q-resolution refutation of $\Phi[\tau]$.

This case distinction is exhaustive – suppose $\mathcal{T}_1 \approx \mathcal{T}'_1 \| \ell_i$ such that $\text{var}(\ell_i) = x_i$ and $\mathcal{T}_2 \approx \mathcal{T}'_2 \| _{\overline{\ell_i}}$. Then the conclusion of $\mathcal{T}'_1$ contains the literals ℓ_i and ℓ_{i+1}, and the conclusion of $\mathcal{T}_2$ contains the literals $\overline{\ell_i}$ and $\overline{\ell_{i+1}}$. Using Lemma 16 and the fact that $\mathcal{T}$ is ordered, it would follow that the literals ℓ_i and ℓ_{i+1} are connected in Ψ with respect to $\mathit{resvar}(\mathcal{T}_1) \subseteq R_\Psi(x_i) \setminus \text{var}_\forall(\Psi)$, and the literals $\overline{\ell_i}$ and $\overline{\ell_{i+1}}$ are connected in Ψ with respect to $\mathit{resvar}(\mathcal{T}_2) \subseteq R_\Psi(x_i) \setminus \text{var}_\forall(\Psi)$. That is, $\{x_i, x_{i+1}\}$ would be an

irreflexive resolution-path dependency pair of Ψ with respect to $R_\Psi(x_i) \setminus \mathrm{var}_\forall(\Psi)$. Since $R_\Psi(x_i) \subseteq R_\Phi(x_i)$ the pair $\{x_i, x_{i+1}\}$ would also be an irreflexive resolution-path dependency pair of Φ with respect to $R_\Phi(x_i) \setminus \mathrm{var}_\forall(\Phi)$ and $(x_i, x_{i+1}) \in D^{\mathrm{res}}_\Phi$, in contradiction with our assumption that Ψ is a D^{res}-transposition of Φ. □

Example 4. Consider the PCNF formula $\Phi = \mathcal{Q}.\varphi$, where

$$\mathcal{Q} = \forall u \exists e \exists v \forall x \exists y \exists z, \text{ and}$$
$$\varphi = \underbrace{(u \vee y)}_{C_1} \wedge \underbrace{(\neg y \vee \neg x \vee v)}_{C_2} \wedge \underbrace{(\neg v \vee x \vee z)}_{C_3} \wedge \underbrace{(\neg z \vee e)}_{C_4} \wedge \underbrace{(\neg u \vee \neg e)}_{C_5}.$$

The sequence $s = u, y, \neg y, v, \neg v, z, \neg z, e$ is a resolution path in Φ via $R_\Phi(u) \setminus (\mathrm{var}_\forall(\Phi) \cup \{e\}) = \{v, y, z\}$. In fact, it is the only resolution path via $\{x, y, z\}$ connecting u and e. The literals $\neg u$ and $\neg e$ are trivially connected, so $\{u, e\}$ is a resolution path dependency pair with respect to $\{v, y, z\}$ and $(u, e) \in D^{\mathrm{res}}_\Phi$. This reflects a "real" dependency of e on u: changing the relative order of u and e in the prefix of Φ results in a formula Ψ that is unsatisfiable while Φ is satisfiable. By Condition A of Definition 8, if $\ell_1, \ldots, \ell_{2k}$ is a resolution path there has to be a sequence $C_1, \ldots, C_k$ of clauses such that $\ell_{2i-1}, \ell_{2i} \in C_i$ for each $i \in [k]$. Let us call such a sequence a *witnessing clause sequence*. For the resolution path s we have $c = C_1, C_2, C_3, C_4$ as a unique witnessing clause sequence. Suppose we were to restrict Definition 8 and require a resolution path to have a witnessing clause sequence such that every pair of consecutive clauses has a non-tautological resolvent (as in Definition 4.1 of [120]). The resulting proto-dependency scheme D would not be a transposition dependency scheme: the clauses C_2 and C_3 do not have a non-tautological resolvent, so s would not be a resolution path and $(u, e) \notin D_\Phi$; thus Ψ would be a D-transposition of Φ but $\Phi \not\equiv \Psi$.

Lemma 28. *The resolution path dependency scheme is monotone.*

Proof. The result follows from the observation that every resolution path in a PCNF formula Φ via X is a resolution path in Φ via Y if $X \subseteq Y \subseteq \mathrm{var}(\Phi)$. □

Having proved Theorem 8 and Lemma 28, we can now apply Proposition 12 to obtain the following result.

Theorem 9. *The resolution path dependency scheme is a permutation dependency scheme.*

Owing to the following observation, this entails that every dependency scheme less general than the resolution-path dependency scheme is a permutation dependency scheme as well.

Proposition 13. *Let D^1 be a permutation (transposition) dependency scheme and let D^2 be proto-dependency scheme such that $D^1 \subseteq D^2$. Then D^2 is a permutation (transposition) dependency scheme.*

Proof. Let Φ be a PCNF formula and let Ψ be a D^2-permutation (D^2-transposition) of Φ. Since $D^1_\Phi \subseteq D^2_\Phi$ by definition of $\subseteq$ (Definition 12), the formula Ψ is also a D^1-permutation (D^1-transposition) of Φ. But D^1 is a permutation (transposition) dependency scheme, so $\Phi \equiv \Psi$. □

We proceed to showing that the refined standard dependency scheme is a permutation dependency scheme. This result generalizes an observation to the effect that the refined standard dependency scheme is a cumulative dependency scheme [102, Remark 3]. Unfortunately, we cannot use Proposition 12 because the refined standard dependency scheme is not monotone, as illustrated by the following example.

Example 5. Consider the formula

$$\Phi = \exists y_1 \exists y_2 \forall x_1 \exists y_3 \forall x_2 \exists y_4 . (y_1 \vee y_2) \wedge (y_2 \vee y_3) \wedge (y_3 \vee x_2) \wedge (x_1 \vee x_2 \vee y_4).$$

The formula Φ contains a (y_1, x_2)-dependency pair only with respect to $\{y_2, y_3\}$. Since $y_2 \notin R^\diamond_\Phi(y_1)$ it follows that $(y_1, x_2) \notin D^{\mathrm{rst}}_\Phi$. It is straightforward to verify that $(y_2, x_1) \notin D^{\mathrm{rst}}_\Phi$, so the formula

$$\Psi = \exists y_1 \forall x_1 \exists y_2 \exists y_3 \forall x_2 \exists y_4 . (y_1 \vee y_2) \wedge (y_2 \vee y_3) \wedge (y_3 \vee x_2) \wedge (x_1 \vee x_2 \vee y_4)$$

is a D^{rst}-permutation of Φ. As Ψ and Φ have the same matrix, Ψ also contains a (y_1, x_2)-dependency pair with respect to $\{y_2, y_3\}$. But $\{y_2, y_3\} \subseteq R^\diamond_\Psi(y_1)$, so $(y_1, x_2) \in D^{\mathrm{rst}}_\Psi$. At the same time, $R_\Psi(y_1) \subseteq R_\Phi(y_1)$, so D^{rst} is not monotone.

To prove that D^{rst} is a permutation dependency scheme, we can instead make the following observation.

Proposition 14. *If D is a proto-dependency scheme such that every true PCNF formula has a D-model and every false formula has a D-countermodel, then D is a permutation dependency scheme.*

Proof. Let D be a proto-dependency scheme and let Φ be a PCNF formula. Suppose Φ is true and has a D-model $\mathbf{f} = \{f_e\}_{e \in \mathrm{var}_\exists(\Phi)}$. For each $e \in \mathrm{var}_\exists(\Phi)$, let $f'_e : 2^{\overline{D_\Phi}(e)} \to \{0,1\}$ be a function defined as follows. Given an assignment $\tau : \overline{D_\Phi}(e) \to \{0,1\}$ we let $f'_e(\tau) = f_e(\tau \cup \sigma_e)$, where $\sigma_e : L^\Phi_e \setminus \overline{D_\Phi}(e) \to \{0,1\}$ is an arbitrary truth assignment. Because $\mathbf{f}$ is a D-model, each function $f_e \in \mathbf{f}$ is independent of $\mathrm{var}_\forall(\Phi) \setminus \overline{D_\Phi}(e)$ and the function f'_e does not depend on our choice of σ_e. Now let Ψ be a D-permutation of Φ. For each $e \in \mathrm{var}_\exists(\Psi) = \mathrm{var}_\exists(\Phi)$, let $g_e : 2^{L^\Psi_e} \to \{0,1\}$ be a model function defined as $g_e(\tau) = f'_e(\tau|_{\overline{D_\Phi}(e)})$ for $\tau : L^\Psi_e \to \{0,1\}$. Because Ψ is a D-permutation of Φ we have $D_\Phi \subseteq R_\Psi$ and in particular $\overline{D_\Phi}(e) \subseteq L^\Psi_e$ for every $e \in \mathrm{var}_\exists(\Psi)$. We claim that $\mathbf{g} = \{g_e\}_{e \in \mathrm{var}_\exists(\Psi)}$ is a model of Ψ. Let $\tau : \mathrm{var}_\forall(\Psi) \to \{0,1\}$ be a truth assignment. Let φ denote the matrix of Φ and Ψ and let $e \in \mathrm{var}_\exists(\Psi)$. Recall that the definition of f'_e is independent of our choice of $\sigma_e : L^\Phi_e \setminus \overline{D_\Phi}(e) \to \{0,1\}$, so in particular we could have chosen $\sigma_e = \tau|_{L^\Phi_e \setminus \overline{D_\Phi}(e)}$. Then

$$f'_e(\tau|_{\overline{D_\Phi}(e)}) = f_e(\tau|_{\overline{D_\Phi}(e)} \cup \sigma_e) = f_e(\tau|_{L^\Phi_e}).$$

We have $g_e(\tau|_{L_e^\Psi}) = f'_e(\tau|_{\overline{D_\Phi(e)}})$ by construction, so $g_e(\tau|_{L_e^\Psi}) = f_e(\tau|_{L_e^\Phi})$. It follows that $\mathbf{g}(\tau) = \mathbf{f}(\tau)$. Since $\mathbf{f}$ is a model we have $\varphi[\tau \cup \mathbf{f}(\tau)] = 1$, so $\varphi[\tau \cup \mathbf{g}(\tau)] = 1$ and $\mathbf{g}$ is indeed a model of Ψ. A similar argument shows that if Φ is false and has a D-countermodel then every D-permutation of Φ has a countermodel. We conclude that if every true formula has a D-model and every false formula has a D-countermodel, then every D-permutation of a PCNF formula Φ has the same truth value as Φ and D is a permutation dependency scheme. □

Using results from Chapter 5, this allows us to establish the following.

Theorem 10. *The refined standard dependency scheme is a permutation dependency scheme.*

Proof. By Proposition 10, every false PCNF formula has a D^{rst}-countermodel, and by Proposition 6 and Lemma 8, every true PCNF formula has a D^{rst}-model. This allows us to apply Proposition 14 and conclude that D^{rst} is a permutation dependency scheme. □

6.2 Minimizing Quantifier Alternations

We define the *alternation depth* of a PCNF formula Φ, in symbols $\mathit{depth}(\Phi)$, as the number of quantifier blocks minus one, that is

$$\mathit{depth}(\Phi) = \max_{x \in \mathrm{var}(\Phi)} \{\mathit{qdepth}(x)\} - 1.$$

We will use dependency schemes to define a more general notion of alternation depth. For this purpose, we interpret dependency relations as directed acyclic graphs in the following way.

Definition 39 (Dependency DAG)**.** Let Φ be a PCNF formula and let D be a proto-dependency scheme. The *dependency DAG* of Φ with respect to D, denoted $G(\Phi, D)$, is the directed graph with vertex set $\mathrm{var}(\Phi)$ and edge set D_Φ.

Observe that R_Φ is a total order and $D_\Phi \subseteq R_\Phi$, so dependency DAGs are indeed acyclic, as stated by the following lemma.

Lemma 29. *Let D be a proto-dependency scheme and Φ a PCNF formula. The dependency DAG of Φ with respect D is acyclic.*

Definition 40 (D-Alternation depth)**.** Let D be a proto-dependency scheme. The *D-alternation depth* of a PCNF formula Φ is the length of a longest path in $G(\Phi, D)$.

We now show that the alternation depth of a formula is simply its D^{trv}-alternation depth.

Lemma 30. *If Φ is a PCNF formula then $\mathit{depth}(\Phi)$ is equivalent to the D^{trv}-alternation depth of Φ.*

Proof. Let Φ be a PCNF formula with D^{trv}-alternation depth l and $\mathit{depth}(\Phi) = k$. By definition of the alternation depth, there has to be a sequence $x_1, \ldots, x_{k+1}$ of variables such that for each $i \in [k]$ we have $x_i \in \mathrm{var}(\Phi)$, $q(x_i) \neq q(x_{i+1})$, and $(x_i, x_{i+1}) \in R_\Phi$. It follows that $(x_i, x_{i+1}) \in D^{\mathrm{trv}}_\Phi$ for each $i \in [k]$, so $x_1, \ldots, x_{k+1}$ is a path in $G(\Phi, D^{\mathrm{trv}})$ and thus $k \leq l$. On the other hand, let $y_1, \ldots, y_{l+1}$ be a path in $G(\Phi, D^{\mathrm{trv}})$. Then $q(y_i) \neq q(y_{i+1})$ for each $i \in [l]$ and $(y_i, y_{i+1}) \in R_\Phi$. From this we get $\mathit{qdepth}(y_i) < \mathit{qdepth}(y_{i+1})$ for each $i \in [l]$ and thus $l \leq k$. □

The next result states that the D-alternation depth of a PCNF formula Φ is a lower bound on the alternation depth of any D-permutation of Φ.

Lemma 31. *Let Φ be a PCNF formula and D a proto-dependency scheme. If there is a path $x_1, \ldots, x_{k+1}$ in $G(\Phi, D)$, the alternation depth of a D-permutation of Φ is at least k. If there is a path $y_1 \ldots y_{k+1}$ in $G(\Phi, D)$ such that $q(x_1) \neq q(y_1)$, the alternation depth of a D-permutation of Φ is at least $k + 1$.*

Proof. Let $x_1, \ldots, x_{k+1}$ be a path in $G(\Phi, D)$ and let Ψ be a D-permutation of Φ. Since D is a proto-dependency scheme we have $D_\Phi \subseteq D^{\mathrm{trv}}_\Phi$, so $(x_i, x_{i+1}) \in D^{\mathrm{trv}}_\Psi$ and thus (x_i, x_{i+1}) is an edge of $G(\Psi, D^{\mathrm{trv}})$ for each $i \in [k]$. That is, $x_1, \ldots, x_{k+1}$ is a path in $G(\Psi, D^{\mathrm{trv}})$, so the alternation depth of Ψ is at least k by Lemma 30. Suppose there exists a path $y_1, \ldots, y_{k+1}$ in $G(\Phi, D)$ such that $q(x_1) \neq q(y_1)$. Then $y_1, \ldots, y_{k+1}$ is a path in $G(\Psi, D^{\mathrm{trv}})$ as well. Assume without loss of generality that $(y_1, x_1) \in R_\Psi$. Then $(y_1, x_1) \in D^{\mathrm{trv}}_\Psi$ and (y_1, x_1) is an edge of $G(\Psi, D^{\mathrm{trv}})$, so $y_1, x_1, \ldots, x_{k+1}$ is a path of length $k+1$ in $G(\Psi, D^{\mathrm{trv}})$ and Ψ has alternation depth at least $k + 1$ by Lemma 30. □

We now present an algorithm (Algorithm 4) that matches this lower bound and computes a D-permutation of minimum alternation depth. Starting with an empty quantifier prefix, Algorithm 4 either removes all existential or all universal source vertices/variables (vertices without incoming edges) from the dependency DAG, appending them to the prefix in arbitrary order. If there are both existentially and universally quantified source vertices/variables, the algorithm picks a quantifier type with a vertex that is the initial vertex of a longest path. This way, it computes a topological ordering of the dependency DAG with as few quantifier alternations as possible.

The dependency DAG can be constructed in linear time from the dependency relation. The body of the while loop is executed at most $|\mathrm{var}(\Phi)|$ times, and each line in the body of the while loop can be implemented so as to run in polynomial time. Overall, we get a polynomial-time algorithm. In fact, the algorithm even runs in linear time.

Lemma 32. *Given a PCNF formula Φ and the relation D_Φ for some proto-dependency scheme D, Algorithm 4 runs in time $O(n + m)$, where $n = |\mathrm{var}(\Phi)|$ and $m = |D_\Phi|$.*

Proof. Let D be a proto-dependency scheme and Φ be PCNF formula. Consider an execution of Algorithm 4 on input (Φ, D_Φ). We are going to assume that $G(\Phi, D)$ is represented by separate adjacency lists for outgoing and incoming edges (such a representation can be computed from Φ and D_Φ in time $O(n + m)$). By processing the vertices of $G(\Phi, D)$ in reverse-topological order, we can compute the number of incoming

Input: A PCNF formula $\Phi = \mathcal{Q}.\varphi$ and a dependency relation D_Φ.
Output: A D-permutation of Φ.

$G \leftarrow G(\Phi, D)$;
$\mathcal{Q}' \leftarrow \varepsilon$;
while $V(G) \neq \emptyset$ **do**
  $x \leftarrow$ initial vertex of a longest path in G;
  $X \leftarrow \{\, v \in V(G) : q(v) = q(x) \text{ and } N_G^-(v) = \emptyset \,\}$;
  $\mathcal{Q}'' \leftarrow Qx_1 \dots Qx_{|X|}$ where $Q = q(x)$ and $X = \{x_1, \dots, x_{|X|}\}$;
  $\mathcal{Q}' \leftarrow \mathcal{Q}'\mathcal{Q}''$;
  $G \leftarrow G[V(G) \setminus X]$;
end
return $\mathcal{Q}'.\varphi$

Algorithm 4: Computing a D-permutation of minimum alternation depth.

edges $in(x)$, as well as the length $l(x)$ of a maximum-length path in $G(\Phi, D)$ starting at x, for each $x \in \mathrm{var}(\Phi)$. This can be done in time $O(n + m)$. We can then compute the sets $X_\forall = \{\, x \in \mathrm{var}_\forall(\Phi) : in(x) = 0 \,\}$ and $X_\exists = \{\, x \in \mathrm{var}_\exists(\Phi) : in(x) = 0 \,\}$ of existentially and universally quantified variables without incoming edges in time $O(n)$.

For each vertex x, we also store a flag $active(x) \in \{0, 1\}$ to encode the characteristic function of $V(G)$ for the remaining (induced) subgraph G of $G(\Phi, D)$. To check whether $V(G) \neq \emptyset$, we maintain a counter c that we initially set to n. Initially, $active(x)$ is set to 1 for each $x \in \mathrm{var}(\Phi)$. Finally, we sort the variables in Φ with respect to $l(x)$, in decreasing order. Because $0 \leq l(x) \leq n - 1$ for each $x \in \mathrm{var}(\Phi)$, we can do this in linear time by creating a bin B_l for each l with $0 \leq l \leq n - 1$, putting each $x \in \mathrm{var}(\Phi)$ in bin $B_{l(x)}$, and then emptying the bins in descending order. Let $x_1, \dots, x_n$ be the corresponding sequence of variables. We maintain a pointer p to an element in this sequence that we set to 1 initially. After this initialization phase, the following properties hold:

(a) x_p points to the initial vertex of a longest path in G.

(b) $X_\forall$ and $X_\exists$ are the sets of universally and existentially quantified variables without incoming edges in G.

(c) The value of c is the number of vertices in G.

(d) The values of *active* encodes the characteristic function of $V(G)$.

(e) If $x \in V(G)$ then $l(x)$ corresponds to the length of a longest path in G starting from x.

(f) For each $x \in \mathrm{var}(\Phi)$ the value of $in(x)$ is the number of in-neighbors of x in G, that is, $in(x) = |N_G^-(x)|$.

To maintain these properties, we implement the while loop in the following way. Let $Q = q(x_p)$. We set $X := X_Q$ as well as $X_Q := \emptyset$. We then go through X in an arbitrary order $y_1, \ldots, y_{|X|}$ and do the following for each y_i. We set $\mathit{active}(y_i) := 0$, $P' := P'Qy_i$, and decrement the counter c by one. For each $y \in N_G^+(y_i)$ we set $\mathit{in}(y) := \mathit{in}(y) - 1$. Whenever $\mathit{in}(y)$ reaches 0 we add y to $X_{q(y)}$. Finally, if $c > 0$, we increase p until $\mathit{active}(x_p) = 1$. This implementation of the loop body maintains Conditions (c), (d), and (f). Let G and G' denote the remaining subgraphs before and after an execution of the loop, respectively. Similarly, let α and α' denote the values of variable α before and after. Let $\overline{\exists} = \forall$ and $\overline{\forall} = \exists$. The set $X'_{\overline{q(x_p)}}$ contains the vertices in $X_{\overline{q(x_p)}}$ plus any vertex in G all whose incoming edges are contained in X, which is just the set of vertices of G' without incoming edges. There cannot be a vertex x in G' such that x has no incoming edges and $q(x) = q(x_p)$ (since the ones in G are contained in X and removal of X does not create new ones), so $X_{q(x_p)}$ is indeed the empty set and Condition (b) is maintained. Moreover, if $x \in V(G')$, then $l(x)$ is still the length of a longest path in G' starting at x: since G' contains no additional edges, the length of a longest path starting from x cannot be bigger than $l(x)$. Let $x, y_1, \ldots, y_{l(x)}$ be a path in G. By assumption, $x \in V(G') \subseteq V(G)$, so each y_i with $i \in [l(x)]$ has an incoming edge in G and therefore $y_i \notin X$. This means $x, y_1, \ldots, y_{l(x)}$ is a path in G', so $l(x)$ is indeed the length of a longest path in G' starting from x, and Condition (e) is preserved. Because $l(x_i) \geq l(x_j)$ for $1 \leq i < j \leq n$, the vertex $x_{p'}$ is the initial vertex of a longest path in G' (provided that $V(G) \neq \emptyset$), so Condition (a) holds as well.

If G is acyclic then G' – which is a subgraph of G – is also acyclic. Accordingly, since $G(\Phi, D)$ is acyclic by Lemma 29, at any time during the execution the remaining subgraph of $G(\Phi, D)$ is acyclic. Thus the set X computed in the loop body and then removed from the digraph is always nonempty, so the algorithm must terminate. More specifically, every vertex $v \in G(\Phi, D)$ is put into either $X_\forall$ or $X_\exists$ exactly once, and each outgoing edge of v is considered once in updating in. The pointer p assumes each value in $\{0, \ldots, n-1\}$ at most once. We conclude that the overall runtime is in $O(n+m)$. □

The next lemma states that the algorithm is correct.

Lemma 33. *Let Φ be a PCNF formula and let D be a proto-dependency scheme. On input (Φ, D_Φ), Algorithm 4 computes a D-permutation of Φ with minimum alternation depth.*

Proof. Let $\Phi = \mathcal{Q}.\varphi$. By Lemma 32, Algorithm 4 terminates and outputs a D-permutation $\Psi = \mathcal{Q}'.\varphi$ of Φ. Let $r > 0$ be the number of times the while loop is executed. For $i \in [r]$, we let X_i be the set of vertices removed from the digraph during the i'th execution of the loop, and set $V_i = \cup_{j=1}^{i} X_j$. Let $G_i = G(\Phi, D)[\mathrm{var}(\Phi) \setminus V_i]$. Note that G_i is the digraph remaining after i executions of the loop body.

We first prove that Ψ is a D-permutation of Φ. Let $G'_i = G(\Phi, D)[V_i]$, let $E_i = E(G'_i)$, let $\mathcal{Q}'_i$ be the subsequence of $\mathcal{Q}'$ constructed after i executions, and let R_i denote $R_{\mathcal{Q}'_i}$. We prove that $E_i \subseteq R_i$ for $i \in [r]$ by induction on i. The set E_0 is empty so the claim trivially holds for $i = 0$. Let $0 < i \leq r$ and suppose the claim holds for every $0 \leq j < i$. Since

$R_{i-1} \subseteq R_i$ we immediately get $E_{i-1} \subseteq R_i$. Suppose there is an edge $(v, w) \in E_i$ such that $(v, w) \notin R_i$. At least one of its endpoints must be in X_i since otherwise $(v, w) \in E_{i-1}$. If $v \notin X_i$ and $w \in X_i$ then $v \in X_j$ for some $j < i$ and $(v, w) \in R_i$, a contradiction. It cannot be the case that both $v \in X_i$ and $w \in X_i$ since $q(v) = q(w)$ by choice of X_i, but G'_i, being a subgraph of $G(\Phi, D)$, only contains edges between variables associated with different quantifiers. The only case remaining is $v \in X_i$ but $w \notin X_i$. Then $w \in X_j$ for some $1 \leq j < i$. By choice of X_j, the variable w does not have any incoming edges in G_{j-1}. But $v \notin V_{j-1}$ and so $(v, w) \in E(G_{j-1})$, a contradiction. We conclude that $E_i \subseteq R_i$ for each $0 \leq i \leq r$. In particular, $E_r = D_\Phi \subseteq R_r = R_\Psi$, so Ψ is a D-permutation of Φ.

For $i \in [r]$, let l_i be the length of a longest path in G_{i-1}. We claim that for each $i \in [r]$, it holds that $l_i = r - i$ if every variable that is the initial vertex of a longest path in G_{i-1} is existential or every initial vertex of a longest path in G_{i-1} is universal, and $l_i + 1 = r - i$ otherwise. Consider the case $i = r$ first. Since X_r is the last set removed by the algorithm, all remaining variables must be of the same type, so we have to show that $l_r = r - r = 0$. Since $X_r = V(G_{r-1})$, for any two variables $v, w \in V(G_{r-1})$ we have $q(v) = q(w)$. But G_{r-1} is a subgraph of $G(\Phi, D)$, which only contains edges between variables of associated with different quantifiers. So G_{r-1} must be edgeless and $l_r = 0$. Let $1 \leq i < r$ and suppose the statement holds for all j such that $i < j \leq r$. Assume first (1) that any two initial vertices of a maximum-length path in G_{i-1} are quantified in the same way, and let $x_1, \ldots, x_{l_i+1}$ be a path in G_{i-1}. Initial vertices of longest paths in G_{i-1} cannot have incoming edges, which in combination with (1) implies that they must be contained in X_i. It follows that a maximum-length path in G_i must be strictly shorter than l_i. The sequence $x_2, \ldots, x_{l_i+1}$ is a path in G_i, so $l_{i+1} = l_i - 1$. Suppose there is a path $y_2, \ldots, y_{l_i+1}$ in G_i such that $q(y_2) \neq q(x_2)$. That is, $q(y_2) = q(x_1)$ and there must be a $y_1 \in V(G_{i-1})$ such that $(y_1, y_2) \in E(G_{i-1})$, since otherwise $y_2 \in X_i$ and the vertex would not be contained in G_i. But then $y_1, \ldots, y_{l_i+1}$ is a (longest) path in G_{i-1} and $q(y_1) \neq q(x_1)$, in contradiction with (1). So the initial vertices of any two longest paths in G_i must be associated with the same quantifier and we can apply the induction hypothesis to conclude that $l_{i+1} = r - (i + 1)$. In combination with $l_{i+1} = l_i - 1$ this yields $r - i - 1 = l_i - 1$ and thus $l_i = r - i$.

Now suppose (2) there are longest paths $x_1, \ldots, x_{l_i+1}$ and $y_1, \ldots, y_{l_i+1}$ in G_{i-1} such that $q(x_1) \neq q(y_1)$. Without loss of generality assume that $x_1 \in X_i$, so that X_i contains all variables $x \in V(G_{i-1})$ without incoming edges and $q(x) = q(x_1)$. The sequence $y_1, \ldots, y_{l_k+1}$ is still a path in G_i because none of its vertices can be in X_i, on pain of contradiction. In fact, it is a longest path in G_i and $l_{i+1} = l_i$. If G_i would contain a path $z_1, \ldots, z_{l_i+1}$ with $q(z_1) \neq q(y_1)$ then this would have been a path G_{i-1} such that $q(z_1) = q(x_1)$, which in turn implies $z_1 \in X_i$, a contradiction. Applying the induction hypothesis, we get $l_{i+1} = r - (i + 1)$ and conclude that $l_i + 1 = r - i$. This proves the claim.

The alternation depth of Ψ is $r - 1$ by construction. Let k be the maximum length of a path in $G(\Phi, D)$. (a) If there are paths $x_1 \ldots x_{k+1}$ and $y_1, \ldots, y_{k+1}$ in $G(\Phi, D)$ such that $q(x_1) \neq q(x_2)$. By Lemma 31, the alternation depth of any D-permutation of Φ

is at least $k+1$, and by the above claim $l_1 + 1 = r - 1$. Note that l_1 is the length of a longest path in $G_0 = G(\Phi, D)$, so $l_1 = k$. We conclude that $r - 1 = k + 1$ and Ψ is a D-permutation of Φ with minimum alternation depth. (b) Otherwise, every length k path in $G(\Phi, D)$ starts with a variable associated with the same quantifier. In that case, the above claim yields $l_1 = r - 1$, i.e. $k = r - 1$. By Lemma 31, the alternation depth of any D-permutation of Φ is at least k, so again Ψ is a D-permutation of Φ with minimum alternation depth. □

In combination with a tractable permutation dependency scheme, we can use Algorithm 4 to preprocess PCNF formulas and possibly reduce the alternation depth, as stated in the following theorem.

Theorem 11. *Let D be a tractable permutation dependency scheme. Given a PCNF formula Φ, a D-permutation of Φ with minimum alternation depth can by computed in polynomial time.*

Proof. The algorithm first invokes a polynomial-time algorithm for computing D_Φ and then runs Algorithm 4 on (Φ, D_Φ) in linear time (Lemma 32). By Lemma 33, this yields a D-permutation of Φ with minimum alternation depth. □

6.3 Reordering and Generalized Resolution

Example 1 of Section 5.1 shows that Q(D)-resolution is not always sound for permutation dependency schemes: as we proved in this chapter, the resolution-path dependency scheme is a permutation dependency scheme, but the example shows that $Q(D^{\text{res}})$-resolution is unsound. In other words, this shows that the combination of reordering according to D^{res} and Q-resolution does not simulate $Q(D^{\text{res}})$-resolution.

We now show that the converse holds: Q(D)-resolution simulates reordering with a permutation dependency scheme plus Q-resolution, and Q(D)-term resolution simulates reordering with a permutation dependency scheme plus Q-term resolution.

Lemma 34. *Let D be a permutation dependency scheme and let $\Phi = \mathcal{Q}.\varphi$ be a PCNF formula. Let Ψ be a D-permutation of Φ and let $\mathcal{T}$ be a Q-resolution derivation from Ψ. Then $\mathcal{T}$ is a $Q(D)$-resolution derivation from Φ.*

Proof. By induction on the structure of $\mathcal{T}$. If $\mathcal{T} \approx \triangle(C)$ then $C \in \varphi$ since Ψ and Φ have the same matrix, so $\mathcal{T}$ is a $Q(D)$-resolution derivation from Φ. If $\mathcal{T} \approx \mathcal{T}_1 \odot_\ell \mathcal{T}_2$ and $\mathcal{T}_1$ as well as $\mathcal{T}_2$ are $Q(D)$-resolution derivations from Φ, then $\mathcal{T}$ is a $Q(D)$-resolution derivation from Φ. Finally, suppose $\mathcal{T} \approx \mathcal{T}'\|_\ell$. Let C denote the conclusion of $\mathcal{T}'$ and let $x = \text{var}(\ell)$. By definition of Q-resolution, we must have $(x, y) \notin D^{\text{trv}}_\Psi$ for each existential variable $y \in \text{var}(C)$, or equivalently, $(x, y) \notin R_\Psi$ for each existential variable $y \in \text{var}(C)$. Because Ψ is a D-permutation of Φ, this implies $(x, y) \notin D_\Phi$ for each existential variable $y \in \text{var}(C)$. Thus $\mathcal{T} \approx \mathcal{T}'\|_\ell$ is a $\forall(D)$-reduction step, and $\mathcal{T}$ is a $Q(D)$-resolution derivation from Φ. □

Using a similar argument, one can also prove the following.

Lemma 35. *Let D be a permutation dependency scheme and let Φ be a PCNF formula. Let Ψ be a D-permutation of Φ and let $\mathcal{T}$ be a Q-term resolution derivation from Ψ. Then $\mathcal{T}$ is a $Q(D)$-term resolution derivation from Φ.*

When combined, these lemmas lead to the following result.

Proposition 15. *Let D be a proto-dependency scheme such that a PCNF formula Φ has a $Q(D)$-resolution refutation only if Φ is false, and a $Q(D)$-term resolution proof only if Φ is true. Then D is a permutation dependency scheme.*

Proof. Let Φ be a PCNF formula and let Ψ be a D-permutation of Φ. If Ψ is false, then there exists a Q-resolution refutation of Ψ, which by Lemma 34 is a $Q(D)$-resolution refutation of Φ. By assumption, this means Φ is false as well. If Ψ is true, then there is a Q-term resolution proof of Ψ, which by Lemma 35 is a $Q(D)$-term resolution proof of Φ. It follows from our assumption that Φ is true as well. We conclude that Ψ is true if and only if Φ is true, so D is a permutation dependency scheme. □

Summary

In this chapter, we studied proto-dependency schemes as a means to reordering the quantifier prefix of a PCNF formula while preserving its truth value. With this application in mind, we introduced the notion of a permutation dependency scheme and showed that in fact, every proto-dependency scheme presented in Chapter 3 is a permutation dependency scheme. We presented a linear-time algorithm that, given a dependency relation, finds a compatible reordering with the smallest number of quantifier alternations. When combined with a tractable permutation dependency scheme, this algorithm yields a polynomial-time preprocessing routine that may reduce the number of quantifier alternations of an input PCNF formula. Finally, we showed that every proto-dependency scheme for which Q(D)-resolution and Q(D)-term resolution are sound is a permutation dependency scheme.

CHAPTER 7

Conclusion

In this part, we established a series of new results on dependency schemes for QBF: we showed that the resolution-path dependency scheme is tractable, proved that the use of several known dependency schemes in QDPLL solvers is sound, and presented an application of dependency schemes to the problem of minimizing the number of quantifier alternations of a PCNF formula. We conclude by sketching the most important challenges for future research.

The main selling point for dependency schemes is their use in the solver DepQBF. The implementation of the (refined) standard dependency scheme leads to increased performance, but it introduces problems of its own. Our results show that the proof systems used by DepQBF in certificate generation are sound, but verifying the correctness of the answer produced by a solver is arguably not the main motivation for generating certificates. In applications such as model checking or planning, QSAT solvers are expected to return models of true formulas and countermodels of false formulas. Propositional formulas encoding models or countermodels can be efficiently (in linear time) extracted from ordinary Q-resolution proofs [8]. Unfortunately, the corresponding algorithm does not work for Q(D)-resolution proofs – the presence of $\forall(D)$-reduction simply breaks the correspondence between (counter)models and proofs.

One way of dealing with this problem is to convert a Q(D)-resolution proof into an ordinary Q-resolution proof before running the extraction algorithm. For the reflexive resolution-path dependency scheme this could be done using the rewriting algorithm presented in Chapter 5. However, our current analysis only yields an exponential upper bound on this algorithm's runtime. From a theoretical perspective, we are faced with the following dilemma: if we show that Q(D)-resolution proofs can be turned into Q-resolution proofs by means of a polynomial-time algorithm, then the use of this dependency scheme in DepQBF cannot pay off in terms of substantially shorter proofs. If proofs in Q(D)-resolution *are* substantially shorter than Q-resolution proofs, we spend the time saved in solving on proof rewriting. It is possible that rewriting can be avoided altogether by developing fast model extraction algorithms that work directly

on Q(D)-resolution proofs. The recent discovery of algorithms for model extraction from long-distance Q-resolution shows that fast model extraction is possible for proof systems even if they cannot be efficiently simulated in Q-resolution [117]. In general, we cannot have both short proofs and fast model extraction, however [10].

Expansion of universally quantified variables is an operation that is used in QBF preprocessing [21] as well as in standalone solvers [12, 83, 72]. Dependency schemes (and related techniques) can be used to prune the set of existential variables that have to be copied upon expansion. In particular, this is known to be the case for the standard dependency scheme and a variant of the triangle dependency scheme [21, 20]. Whether this is also possible for the reflexive resolution-path dependency scheme is currently open. Recently introduced proof systems that capture the power of expansion [73, 71, 67] may provide a starting point for generalizing our proof-theoretic argument for soundness of $Q(D^{\text{rrs}})$-resolution.

The default input format for QSAT solvers is PCNF. Formulas that are not already in this format have to be converted before they can be passed to these solvers, a process that may result in a loss of structural information. While general QBFs are completely symmetric with respect to truth and falsity, CNF representations are biased towards detecting falsified clauses [1, 98]. This observation has lead to the development of several techniques for symmetric reasoning about satisfiability and unsatisfiability [126, 98, 60, 79] that are often subsumed under the term *dual propagation* [59]. These include dedicated non-CNF solvers [38, 60, 79] as well as approaches that rely on representations that combine CNF and DNF encodings [126, 98]. The latter can be implemented in PCNF solvers by a clever use of the constraint learning mechanisms already present in most modern solvers [63]. Although formula structure can be partially reconstructed from PCNF encodings [61], dual propagation works best when the original non-CNF representation of a formula is available. The recent introduction of a standard encoding format (QCIR) for general QBFs (in prenex normal form) may lead to a workflow for QSAT solving where an input formula is transformed into a combined CNF-DNF representation that is used to seed the constraint database(s) of a PCNF solver. Common techniques for carrying out this transformation process generate PCNF formulas for which the standard dependency relation and even (in the case of Tseitin conversion) the resolution-path dependency relation coincides with the trivial dependency relation. It looks like combining the use of a non-CNF input format with dependency schemes will require new ideas.

Related work. The standard dependency scheme originated from techniques for reducing the cost of Shannon expansion. Upon expanding a variable x in a formula Φ, the formula Φ is replaced by two copies of itself: one where x is assigned 0, and one in which x is assigned 1. This step can be repeated until every variable has been assigned, or until the formula contains only one existentially quantified variables, at which point the instance can be solved by an off-the-shelf SAT solver. To mitigate the increase in formula size caused by quantifier expansion, one can perform anti-prenexing, or miniscoping, to narrow the scope of the variable that is to be expanded [5]. For the special case of

expanding an innermost universal variable x of a PCNF formula, Biere observed that only existential variables that are suitably connected to x must be copied [12]. This was generalized to universal variables of arbitrary depth by Bubeck and Kleine Büning [21]. Their definition of dependent existential variables coincides with those identified by the standard dependency scheme, introduced independently by Samer and Szeider [101, 102].

There is a related line of work on offsetting the negative effects of prenexing [39, 9, 52]. Quantifier shifting can be applied to perform prenexing in a way that reduces the number of quantifier alternations of the resulting PCNF formula [39], and information on the nesting of scopes in a non-prenex formula can be used to increase the performance of PCNF solvers for the prenexed version [52]. Alternatively, the structure of quantifier scopes can be partially reconstructed from PCNF formulas [9]. For a detailed account of how the standard dependency scheme generalizes this technique, see [14, 82].

Samer defined a variant of dependency schemes for the Quantified Constraint Satisfaction Problem (QCSP), based on a notion of semantic independence [100]. Translated back from QCSP to QSAT, a set Y of existential variables is independent, according to his definition, from a universal variable x, if there is a model in which the model function for each $y \in Y$ is independent of the assignment to x. If this is the case, the variables in Y need not be copied upon expanding x. Dependency schemes are mappings D satisfying the following condition: if Φ is a PCNF formula, $Y \subseteq \mathrm{var}_\exists(\Phi)$ is a set of existential variables, and $x \in \mathrm{var}_\forall(\Phi)$ is a universal variable, then $D_\Phi(Y, x) \subseteq Y$ is a set such that $Y \setminus D_\Phi(Y, x)$ is independent from x [100, p.518]. Samer proved that the standard dependency scheme, suitably adapted to this setting, is a dependency scheme according to this definition, while the triangle dependency scheme is not. He proposed a variant of the triangle dependency scheme which is a dependency scheme in the above sense, and used it as the base case in a hierarchy of dependency schemes of increasing generality. Whether Q(D)-resolution is sound for these dependency schemes is an open question.

One-sided dependencies as studied by Bubeck amount to a different relaxation of the triangle dependency scheme where the relevant X-paths may go through the dependent existential variable [20]. Stated in our terminology, he showed that every true PCNF formula has a D-model, where D is the proto-dependency scheme that computes one-sided dependencies.

Resolution-path dependencies, along with quadrangle and strict standard dependencies (see Section 3.4), were introduced by Van Gelder – essentially, he proved that the resolution-path dependency scheme is a transposition dependency scheme [120]. He further made a claim to the effect that $\forall(D^{\mathrm{res}})$-reduction is sound (Theorem 4.9). However, the counterexample to soundness of $Q(D^{\mathrm{res}})$-resolution presented in Section 5.1 can be used to show that this claim does not hold: adding the clause $(\neg u \vee \neg y)$ derived from $(\neg x \vee \neg u \vee \neg y)$ by $\forall(D^{\mathrm{res}})$-reduction to the formula Φ from Example 1 results in a false formula.

Dependency Quantified Boolean Formulas (DQBFs) are a generalization of QBFs in which every existential variable is annotated with a subset of universal variables [93]. A DQBF is satisfiable if there is a model in which the model function for each existential variable only depends on the assignment to universal variables in its annotation. Balabanov, Chiang, and Jiang defined a generalization of Q-resolution to DQBF they call DQ-resolution [7]. Like Q(D)-resolution, DQ-resolution uses a generalized version of $\forall$-reduction that can remove a universal variable x from a clause C as long as C does not contain existential variables depending on x – that is, as long as x does not appear in the annotation of any existential variable occurring in C.

Part II

Propositional Model Counting

CHAPTER 8

Introduction

The propositional model counting problem (#SAT) asks for the number $\#F$ of satisfying truth assignments of a propositional formula F in conjunctive normal form (CNF). Aside from being a well-studied problem in theoretical computer science, the close relation of #SAT to several problems in probabilistic reasoning has prompted research into efficient algorithms for model counting. For instance, the probability of a formula F being true under a uniform distribution of truth assignments can be computed as $\#F/2^n$, and reasoning in probabilistic graphical models such as Bayesian networks [92] can be reduced to a generalization of #SAT known as *weighted model counting*[1] [6, 105].

Model counting is at least as hard as propositional satisfiability (SAT), and there is strong evidence that the problem is in fact significantly harder. In practice, #SAT has turned out to be much more difficult to solve than SAT, and the largest instances whose model count can be feasibly determined are orders of magnitude smaller than those handled by state-of-the-art SAT solvers [56]. This is the case for both exact model counters [16, 74, 35, 104, 115] and the (randomized) approximation algorithms on which research has focused in recent years [124, 55, 53, 81, 26, 41] (though the latter scale much better).

The empirical hardness of model counting goes hand in hand with findings in complexity theory. #SAT is complete for the class #P introduced by Valiant to study the complexity of counting problems [119, 118]. Roughly speaking, a counting problem is in #P if the number of solutions of an instance corresponds to the number of accepting runs of a nondeterministic Turing machine for the associated decision problem [119]. The counting versions of most well-known NP-complete problems are complete for the class #P. Surprisingly, there are problems which can be decided in polynomial time whose counting versions are complete for #P, such as computing the number of perfect

[1] Weighted model counting is a variant of #SAT where each propositional variable x comes with a weight $w(x) \in [0, 1]$. Under an assignment τ, a variable x has weight $w(x)$ if $\tau(x) = 1$ and $1 - w(x)$ if $\tau(x) = 0$. The weight of an assignment is the product of the weights of individual variables, and the task is to compute the sum over the weights of satisfying assignments.

matchings in a bipartite graph [118]. Here, completeness is defined with respect to *counting reductions*. A counting reduction from problem $\mathcal{A}$ to problem $\mathcal{B}$ consists of two parts: a polynomial-time algorithm that transforms an instance A of $\mathcal{A}$ into an instance B of $\mathcal{B}$, and another polynomial-time algorithm for retrieving the number of solutions of A from the number of solutions of B.

In general, problems in #P are highly intractable. In a seminal paper [116], Toda proved that any problem in the polynomial hierarchy can be solved by a polynomial-time algorithm with access to a #P oracle – in fact, a single query to a #P oracle is sufficient. For many applications, computing an estimate of the number of satisying assignments of a formula would be good enough, but it is NP-hard even to approximate the model count of a formula within a factor of $2^{n^{1-\varepsilon}}$ for any $\varepsilon > 0$ [97, 127]. Furthermore, unless NP can be solved by probabilistic polynomial-time algorithms with one-sided error (that is, unless NP = RP), #SAT does not admit a fully polynomial randomized approximation scheme (FPRAS) [127].[2]

These results have inspired research into what restrictions make #SAT tractable. Surprisingly, the problem remains hard for fragments for which SAT is in P or even trivial, such as Horn, 2CNF, or monotone formulas [97]. Again, approximation offers no relief, as it is NP-hard to approximate the number of satisfying truth assignments of a formula with n variables to within a factor $2^{n^{1-\varepsilon}}$ for any $\varepsilon > 0$ for Horn and monotone 2CNF formulas [97, 127].

On the other hand, restrictions on *structural parameters* have led to a series of positive results [6, 114, 44, 87, 103, 47, 49, 91, 108]. A structural parameter of a formula F corresponds to a graph parameter (or invariant) of a graph associated with F, a paradigmatic example being the treewidth of the primal graph (the primal graph contains a vertex for each variable of the input formula, and an edge between any pair of variables that co-occur in a clause). #SAT admits efficient algorithms with respect to several structural parameters, with runtime bounds of the form $f(k)n^c$ or $n^{f(k)}$, where, k is the value of the structural parameter, c is a constant, and f is a function independent of the instance. Bounds of both shapes yield polynomial-time tractability when k is considered a constant, but algorithms with bounds of the former kind (so-called *fixed parameter tractable* algorithms [37, 45]) perform better for large instances with small parameter values.

Here, we focus on structural parameters defined in terms of graph *width* measures [69]. Width measures can be characterized through decompositions which organize graphs in a tree-like structure. In the case of structural parameters, these decompositions correspond to decompositions of an input formula. Given a decomposition witnessing small width, one can sometimes efficiently solve #SAT by means of dynamic programming along the following lines: the algorithm maintains a "table" for each node in the decomposition, where an entry in the table corresponds to the cardinality of a class of partial assignments; table entries are computed in a bottom up manner, starting from the leaves of the

[2] An FPRAS for a counting problem is an algorithm that, for any ε and δ, approximates the solution count of an instance within a factor of $1 - \varepsilon$ with probability $1 - \delta$, and runs in time polynomial in ε, δ, and the size of the instance.

decomposition; the model count of the input formula can be computed from the table associated with the root node. Choosing classes of partial assignments in such a way as to keep tables small but store enough information to retrieve the model count at the root is the main challenge in designing such algorithms.

Structural parameters can be partially ordered in the following way. We say that a parameter p *strictly dominates* parameter q if there is a function $f : \mathbb{R} \to \mathbb{R}$ so that $p(F) \leq f(q(F))$ for every formula F, but not vice versa. With respect to this order, structural parameters based on width measures can be organized in a natural hierarchy (see Chapter 10). We study the complexity of #SAT with respect to the modular treewidth of the incidence graph (recall that the incidence graph of a formula has variables and clauses as its vertices, and contains an edge joining a variable x with a clause C if x occurs in C) and the symmetric clique-width of the incidence graph. The main contribution of this part are dynamic programming algorithms running in time $n^{f(k)}$ for computing the model count of formulas of size n and *modular incidence treewidth* k or *symmetric incidence clique-width* k. This implies polynomial-time tractability of classes of formulas for which these parameters are bounded by a constant.

The most general structural parameters in this hierarchy for which #SAT was previously known to be tractable are *incidence treewidth* and *signed incidence clique-width*, both of which admit algorithms with runtime bounds of the form $f(k)n^c$. We prove that such bounds cannot be obtained with respect to modular incidence treewidth or symmetric incidence clique-width, subject to a well-established hypothesis in the area of parameterized complexity [37, 45].

Modular incidence treewidth strictly dominates incidence treewidth and is incomparable with signed incidence clique-width, while symmetric incidence clique-width strictly dominates both, and our algorithms allow us to efficiently solve #SAT for classes of formulas for which algorithms exploiting small incidence treewidth or signed incidence clique-width require exponential time.

These results are obtained through a combination of dynamic programming and the representation of truth assignments by their projections. By a *projection* of a truth assignment τ onto a formula F we mean the subset of clauses of F satisfied by τ [76]. Observe that the projection of a satisfying assignment onto F is F itself, and that the projection onto F of the union of assignments (that agree on the intersection of their domains) is just the union of their individual projections onto F. We use these properties to combine standard dynamic programming techniques [103, 47] with tables storing information on partial assignments in terms of projections, and show that the size of these tables is in $O(m^{f(k)})$ for formulas with m clauses and modular incidence treewidth k or symmetric clique-width k.

Organization of Part II. In Chapter 9, we review preliminaries and notation. Chapter 10 surveys complexity results for #SAT with respect to structural parameters and organizes these parameters according to a notion of strict domination. The main contributions of this part are presented in Chapter 11 and Chapter 12, where we prove that #SAT is polynomial-time tractable for classes of formulas of bounded modular incidence

treewidth and classes of formulas of bounded symmetric clique-width, respectively. We conclude and give an overview of related work in Chapter 13.

The main contributions of this part have been published, in preliminary form, as part of conference proceedings:

- An algorithm for #SAT parameterized by modular incidence treewidth (with a similar runtime bound as the one presented in Chapter 11) was published in the proceedings of STACS 2013 [91].
- Tractability of #SAT for classes of formulas of bounded symmetric clique-width (Chapter 12, Theorem 16), was proved in a paper published in the proceedings of ISAAC 2013 [108].

CHAPTER 9

Preliminaries

We write $\mathbb{N}$ to denote the set non-negative integers. If $n > 0$ is a non-negative integer, we write $[n]$ for the set $\{1, \ldots, n\}$.

Graphs. A *directed graph*, or *digraph*, for short, is a pair $G = (V, E)$, where V is a finite set and $E \subseteq V \times V$ is an irreflexive binary relation. The elements of V and E are called the *vertices* and *edges* of G, respectively. A *subdigraph* of G is a digraph $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$. The subdigraph of G *induced* by a subset $V' \subseteq V$ of vertices is the digraph $G[V'] = (V', E \cap (V' \times V'))$. If E is symmetric then G is a *graph*. In this case, we may write vw to denote an edge $(v, w) \in E$. If $G = (V, E)$ is a graph and $v \in V$ is a vertex of G, we let $N(v)$ denote the set $\{ w \in V : vw \in E \}$ of *neighbors* of v in G. Let $G = (V, E)$ be a graph. A *path* (in G) from v to w is a sequence $v_1, \ldots, v_k$ such that $v_i \in V$ for each $i \in [k]$ and $v_i v_{i+1} \in E$ for each $i \in [k-1]$. If there is a path from v to w for every pair of vertices $v, w \in V$ then G is *connected*. The graph G is *bipartite* if V can be split into (disjoint) subsets U and W such that for each edge $uw \in E$, $u \in U$ and $w \in W$.

A *tree* (*rooted at* r) is a graph that can be constructed using the following rules. The graph $(\{r\}, \emptyset)$ is tree. If $T = (V, E)$ is a tree, $v \in V$, and $w \notin V$, then $G' = (V \cup \{w\}, E \cup \{vw\})$ is a tree. Let $T = (V, E)$ be a tree rooted at r. We call r the *root* of T. If $vw \in E$ and v occurs on the (unique) path from w to r, then w is a *child* of v in T. If $v \in V$ does not have a child then v is a *leaf*. We write $L(T)$ to denote the set of leaves of T. We call T a *binary* tree if every vertex that is not a leaf has exactly to children. If $v \in V$ occurs on the path from w to r in G we say w is *below* v. The vertices of a tree are sometimes called *nodes*.

Let $\mathcal{C}$ be a class of graphs and let f be a mapping (invariant under isomorphisms) that associates each graph G with a non-negative real number. We say $\mathcal{C}$ *has bounded* f if there is a c such that $f(G) \leq c$ for every $G \in \mathcal{C}$. Otherwise, we say that $\mathcal{C}$ *has unbounded* f.

Formulas. A *literal* is a negated or unnegated variable. If x is a variable, we write $\overline{x} = \neg x$ and $\overline{\neg x} = x$, and let $\mathrm{var}(x) = \mathrm{var}(\neg x) = x$. If L is a set of literals, we write $\overline{L}$ for the set $\{\, \overline{\ell} : \ell \in L \,\}$. For a set X of variables we let $\mathit{lit}(X) = X \cup \overline{X}$. A *clause* is a finite disjunction of literals. We call a clause *tautological* if it contains the same variable negated as well as unnegated. A *(CNF) formula* is a finite conjunction of non-tautological clauses. Whenever convenient, we treat clauses and terms as sets of literals, and a CNF formula as a set of sets of literals. If C is clause, we let $\mathrm{var}(C)$ be the set of variables occurring (negated or unnegated) in C. For a CNF formula F we let $\mathrm{var}(F) = \bigcup_{C \in F} \mathrm{var}(C)$. The *length* of a formula F is $\sum_{C \in F} |C|$.

Let F be a formula. The *incidence graph* of F the bipartite graph $I(F)$ with vertex set $\mathrm{var}(F) \cup F$ and edge set $\{\, Cx : C \in F \text{ and } x \in \mathrm{var}(C) \,\}$. Two vertices are *twins* if they have the same neighbors in $I(F)$. The equivalence classes of the twin relation are called *modules*. If v and w are twins of $I(F)$, then either $v, w \in \mathrm{var}(F)$ or $v, w \in F$, as $I(F)$ does not contain isolated vertices. Accordingly, each module of $I(F)$ either contains only variables or only clauses. If $\mathbf{x} \subseteq \mathrm{var}(F)$ is a module of $I(F)$, then $\mathbf{x}$ is a *variable module* of F. If $\mathcal{C} \subseteq F$ is a module of $I(F)$, then $\mathcal{C}$ is a *clause module* of F. The *modular incidence graph* of F is the bipartite graph $I^*(F)$ whose vertices are the variable and clause modules of F, and which contains an edge $\mathcal{C}\mathbf{x}$ if Cx is an edge in the incidence graph for $C \in \mathcal{C}$ and $x \in \mathbf{x}$.

For a set X of variables, a *truth assignment* (or simply *assignment*) is a mapping $\tau : X \to \{0,1\}$. We extend τ to literals by letting $\tau(\neg x) = 1 - \tau(x)$. An assignment τ *satisfies* a clause C if there is a literal $\ell \in C$ such that $\tau(\ell) = 1$. The assignment $\varepsilon : \emptyset \to \{0,1\}$ is called the *empty assignment*. If an assignment τ satisfies each clause C of a formula F, then τ *satisfies* F. If F is a formula and $\tau : \mathrm{var}(F) \to \{0,1\}$ satisfies F, then F is *satisfiable* and τ is a *satisfying assignment* of F. The satisfiability (SAT) problem is that of testing whether a given formula has a satisfying assignment. Propositional model counting (#SAT) is a generalization of SAT that asks for the number $\#F$ of satisfying assignments of a given formula F.

If $\tau : X \to \{0,1\}$ is a truth assignment and $Y \subseteq X$, then the *restriction of* τ *to* Y, in symbols $\tau|_Y$, is the truth assignment $\tau' : Y \to \{0,1\}$ such that $\tau'(x) = \tau(x)$ for each $x \in Y$. If $\tau : X \to \{0,1\}$ and $\sigma : Y \to \{0,1\}$ are truth assignments and $\tau(x) = \sigma(x)$ for each $x \in X \cap Y$, then the *union* of τ and σ is the truth assignment $\tau \cup \sigma : X \cup Y \to \{0,1\}$, where $(\tau \cup \sigma)(x) = \tau(x)$ if $x \in X$, and $(\tau \cup \sigma)(y) = \sigma(y)$ otherwise.

Tree decompositions. Let $G = (V(G), E(G))$ be a finite, undirected graph without self-loops or multiple edges. A *tree decomposition* of G is a triple $\mathcal{T} = (T, \chi, r)$, where $T = (V(T), E(T))$ is a tree rooted at r and $\chi : V(T) \to 2^{V(G)}$ is a labeling of the vertices of T by subsets of $V(G)$ (called *bags*) such that the following three conditions hold:

1. $\bigcup_{t \in V(T)} \chi(t) = V(G)$,
2. for each edge $uv \in E(G)$, there is a node $t \in V(T)$ with $\{u, v\} \subseteq \chi(t)$,
3. for each vertex $x \in V(G)$, the set of nodes t with $x \in \chi(t)$ forms a connected subtree of T.

The *width* of a tree decomposition (T, χ, r) is the size of a largest bag $\chi(t)$ minus 1. The *treewidth* of G is the minimum width over all possible tree decompositions of G. A tree decomposition (T, χ, r) is *nice* if T is a binary tree such that the nodes of T belong to one of the following four types:

A. a *leaf node* t is a leaf of T,

B. an *introduce node* t has one child t' and $\chi(t) \setminus \{v\} = \chi(t')$ for some vertex $v \in V(G)$,

C. a *forget node* t has one child t' and $\chi(t') \setminus \{v\} = \chi(t)$ for some vertex $v \in V(G)$,

D. a *join node* t has two children t_1, t_2 and $\chi(t) = \chi(t_1) = \chi(t_2)$.

Kloks [80] showed that every tree decomposition of a graph G can be converted in linear time to a nice tree decomposition, such that the size of the largest bag does not increase, and the corresponding tree has at most $4|V(G)|$ nodes.

Let F be a formula. We call the treewidth of the modular incidence graph $I^*(F)$ the *modular incidence treewidth* of F. Let $\mathcal{T} = (T, \chi, r)$ be a tree decomposition of $I^*(F)$. For $t \in V(T)$, we write $\chi_c(t)$ and $\chi_v(t)$ to denote the sets of clause modules and variable modules in $\chi(t)$, respectively. We further assume an arbitrary ordering of the variable modules of $I^*(F)$, and let $\chi_{\mathbf{v}}$ be the mapping which, for each node t of T, returns a tuple containing the variable modules in $\chi_v(t)$ ordered accordingly. For each node t of T, we let F_t denote the formula consisting of the union of clause modules appearing in bags of nodes below t in T (this includes the bag of t). Similarly, X_t denotes the union of variable modules appearing in bags of nodes at or below t. We further let $X_t^b = \cup\chi_v(t)$ and $F_t^b = \cup\chi_c(t)$ denote the union of variable and clause modules appearing in the bag of t, respectively.

Decomposition trees. We review decomposition trees following the presentation in [23]. Let $G = (V, E)$ be a graph. A *decomposition tree* for G is a pair (T, δ), where T is a rooted binary tree and $\delta : L(T) \to V$ is a bijection. For a subset $X \subseteq V$ let $\overline{X} = V \setminus X$. We associate every edge $e \in E(T)$ with a bipartition P_e of V obtained as follows. If T_1 and T_2 are the components obtained by removing e from T, we let $P_e = (L(T_1), L(T_2))$. Note that $L(T_2) = \overline{X}$ for $X = L(T_1)$. A function $f : 2^V \to \mathbb{R}$ is *symmetric* if $f(X) = f(\overline{X})$ for all $X \subseteq V$. Let $f : 2^V \to \mathbb{R}$ be a symmetric function. The f-width of (T, δ) is the maximum of $f(X) = f(\overline{X})$ taken over the bipartitions $P_e = (X, \overline{X})$ for all $e \in E(T)$. The f-width of G is the minimum of the f-widths of the decomposition trees of G.

Let $A(G)$ stand for the *adjacency matrix* of G, that is, the $V \times V$ matrix $A(G) = (a_{vw})_{v \in V, w \in V}$ such that $a_{vw} = 1$ if $vw \in E$ and $a_{vw} = 0$ otherwise. For $X, Y \subseteq V$, let $A(G)[X, Y]$ denote the $X \times Y$ submatrix $(a_{vw})_{v \in X, w \in Y}$. The *cut-rank* function $\rho_G : 2^V \to \mathbb{R}$ of G is defined as

$$\rho_G(X) = \mathit{rank}(A(G)[X, V \setminus X]),$$

where *rank* is the rank function of matrices over $\mathbb{Z}_2$. The row and column ranks of any matrix are equivalent, so this function is symmetric. The *rank-width* of a decomposition

tree (T, δ) of G, denoted *rankw*(T, δ), is the ρ_G-width of (T, δ), and the *rank-width* of G, denoted *rankw*(G), is the ρ_G-width of G.

Let X be a proper nonempty subset of V. We define an equivalence relation $\equiv_X$ on X as

$$x \equiv_X y \text{ iff, for every } z \in V \setminus X,\ xz \in E \Leftrightarrow yz \in E.$$

The *index* of X in G is the cardinality of $X/\equiv_X$, that is, the number of equivalence classes of $\equiv_X$. We let $index_G : 2^V \to \mathbb{R}$ be the function that maps each proper nonempty subset X of V to its index in G. We now define the function $\iota_G : 2^V \to \mathbb{R}$ as

$$\iota_G(X) = \max(index_G(X), index_G(V \setminus X)).$$

This function is trivially symmetric. The *index* of a decomposition tree (T, δ) of G, denoted *index*(T, δ), is the ι_G-width of (T, δ). The *symmetric clique-width* of G, denoted *scw*(G), is the ι_G-width of G [29].

Symmetric clique-width and rank-width are closely related graph parameters. In fact, the index of a decomposition tree can be bounded in terms of its rank-width.

Lemma 36. *For every graph G and decomposition tree (T, δ) of G, rankw$(T, \delta) \leq$ index$(T, \delta) \leq 2^{rankw(T,\delta)}$.*

Proof. Let $G = (V, E)$ be a graph and X be a nonempty proper subset of V. For every pair of vertices $x, y \in X$ the rows of $A(G)[X, V \setminus X]$ with indices x and y are identical if and only if $x \equiv_X y$. So $index_G(X)$ is precisely the number of distinct rows of $A(G)[X, V \setminus X]$, which is an upper bound on the rank of $A(G)[X, V \setminus X]$ over $\mathbb{Z}_2$. Symmetrically, $index_G(V \setminus X)$ is the number of distinct columns of $A(G)[X, V \setminus X]$, which is also an upper bound on the rank. So $\rho_G(X) \leq \iota_G(X)$, which proves the left inequality. The rank of $A(G)[X, V \setminus X]$ is the cardinality of a basis for the matrix's row (column) space. That is, each of its row (column) vectors can be represented as a linear combination of $\rho_G(X)$ row (column) vectors. Over $\mathbb{Z}_2$, any linear combination can be obtained using only 0 and 1 as coefficients. Accordingly, there can be at most $2^{\rho(X)}$ distinct rows (columns) in $A(G)[X, V \setminus X]$. So $\iota_G(X) \leq 2^{\rho_G(X)}$, and the right inequality follows. □

Corollary 2. *For every graph G, rankw$(G) \leq$ scw$(G) \leq 2^{rankw(G)}$.*

Runtime bounds for the dynamic programming algorithm presented below are more naturally stated in terms the index of the underlying decomposition tree than in terms of its rank-width. However, to the best of our knowledge, there is no polynomial-time algorithm for computing decomposition trees of minimum index directly – instead, we will use the following result to compute decomposition trees of minimum rank-width.

Theorem 12 ([40])**.** *Let $k \in \mathbb{N}$ be a constant and $n \geq 2$. For an n-vertex graph G, we can output a decomposition tree of rank-width at most k or confirm that the rank-width of G is larger than k in time $O(n^3)$.*

CHAPTER 10

Taxonomy of Structural Parameters

This chapter gives an overview and comparison of structural parameters (more specifically, structural parameters based on width measures) for #SAT.

Formally, a (structural) parameter of CNF formulas is a function p that maps each formula to a non-negative real number. If $\mathcal{C}$ is a class of CNF formulas and p is a parameter, we say $\mathcal{C}$ *has bounded* p if there is a constant c such that $p(F) \leq c$ for each $F \in \mathcal{C}$; otherwise, if there is no such constant, $\mathcal{C}$ is said to have *unbounded* p.

In comparing parameters, we use the following terminology [103]. For parameters p and q, we say that p *dominates* q if there is a function f such that $p(F) \leq f(q(F))$ for every formula F. We say that p is *strictly dominates* q if p dominates q but not the other way around. Two parameters are *equivalent* if they dominate each other, and *incomparable* if neither dominates the other.

10.1 Treewidth

Definition 41. Let F be a CNF formula. The *primal graph* of F is the graph $G = (V, E)$, where $V = \mathrm{var}(F)$ and $E = \{\, xy : \text{ there is a clause } C \in F \text{ such that } x, y \in \mathrm{var}(C) \,\}$.

Primal treewidth. The *primal treewidth* of a formula is the treewidth of its primal graph. #SAT can be solved in time $O(2^k kn^2)$ for formulas with n variables and primal treewidth k [103]. A bound of $2^{O(k)}n^{O(1)}$ can be obtained for formulas with n variables and primal graphs of *branchwidth* k [6]. Branchwidth is a graph parameter essentially equivalent to treewidth [96].

Incidence treewidth. The *incidence treewidth* of a formula is the treewidth of its incidence graph. Recall that the incidence graph $I(F)$ of a formula F is the bipartite

graph with vertices $\mathrm{var}(F) \cup F$ and edges $\{\, xC : C \in F \text{ and } x \in \mathrm{var}(C) \,\}$. The models of a formula F of length l with incidence treewidth k can be counted in time $O(4^k l)$ [44]. A different algorithm achieves a bound of $O(2^k ks(n+m))$ for formulas with n variables, m clauses, maximum clause size s, and incidence treewidth k [103]. Bounds of the form $f(k)\, l^c$ on the time complexity of #SAT for formulas of incidence treewidth k and length l, where c is a constant and f is a function independent of the input, can also be obtained using algorithmic meta-theorems of Courcelle, Makowsky and Rotics [31, 32], but the function f is at least doubly exponential in k [44].

The following result is well known (see, for instance Gottlob and Pichler [58]).

Proposition 16. *Incidence treewidth strictly dominates primal treewidth.*

Proof. We first argue that incidence treewidth dominates primal treewidth. To see this, let F be a CNF formula and consider a tree decomposition $\mathcal{T} = (T, \chi, r)$ of its primal graph of width k. For each clause $C \in F$, the set $\mathrm{var}(C)$ of variables occurring in C forms a clique in the primal graph. It follows from the properties of a tree decomposition that there has to be a node t of T such that $\mathrm{var}(C) \subseteq \chi(t)$. By adding C to the bag of t, we obtain a bag that covers every edge incident to C in $I(F)$. By doing this for each $C \in F$ (copying tree nodes if necessary), we get a tree decomposition of the incidence graph of F of width at most $k+1$.

Let $F_n = \{x_1, \ldots, x_n\}$ be the CNF formula consisting of a single clause on n variables. The primal graph of F_n is an n-clique, whereas $I(F)$ is a tree, so the class $\mathcal{C} = \{\, F_n : n > 0 \,\}$ has unbounded primal treewidth but bounded incidence treewidth. We conclude that incidence treewidth strictly dominates primal treewidth. □

Modular incidence treewidth. Recall that the modular incidence treewidth of a formula F is the treewidth of the modular incidence graph $I^*(F)$, that is, the incidence graph after contraction of modules. In Chapter 11, we will show that #SAT can be solved in time $O(l^{2k+7})$ for CNF formula of length l and modular incidence treewidth k (Theorem 13).

It is not hard to see that modular incidence treewidth strictly dominates incidence treewidth, as stated in the following proposition.

Proposition 17. *Modular incidence treewidth strictly dominates incidence treewidth.*

Proof. Let F be a formula. The modular incidence graph $I^*(F)$ is isomorphic to an induced subgraph of $I(F)$, so the treewidth of $I^*(F)$ is at most the treewidth of $I(F)$. That is, modular incidence treewidth dominates incidence treewidth. Now consider the class $\mathcal{C} = \{\, F_n : n \geq 1 \,\}$, where the formula F_n consists of n clauses $C_1, \ldots, C_n$ such that $\mathrm{var}(C_i) = \{x_1, \ldots, x_n\}$ for each $i \in [n]$. The set $\{x_1, \ldots, x_n\}$ is a variable module of F and the set $\{C_1, \ldots, C_n\}$ is a clause module of F, so $I^*(F_n)$ consists of a single edge and has treewidth 1. On the other hand, the incidence graph $I(F_n)$ is a complete bipartite graph of treewidth n. So $\mathcal{C}$ has unbounded incidence treewidth but bounded modular incidence treewidth. We conclude that modular incidence treewidth strictly dominates incidence treewidth. □

10.2 Clique-width

Signed incidence clique-width. Clique-width is a graph parameter based on graph grammars [30]. Directed clique-width is a variant of clique-width for digraphs [33]. The signed incidence graph is obtained from the incidence graph by orientating its edges so as to indicate positive or negative occurrences of variables.

Definition 42. Let F be a CNF formula. The *signed incidence graph* of F is the digraph $G = (V, E)$, where $V = \mathrm{var}(F) \cup F$ and

$$E = \{\, (x, C) \in V \times V : x \in C \,\} \cup \{\, (C, x) \in V \times V : \neg x \in C \,\}.$$

The directed clique-width of a digraph is defined as follows.

Definition 43 (Directed clique-width)**.** Let S be a finite set. An *S-labeled graph* is a pair (G, λ) where G is a directed graph and λ is a mapping $\lambda : V(G) \to S$, called an *S-labeling of G*. Let $\mathbf{G} = (G, \lambda)$ be an S-labeled graph and let $a, b \in S$ with $a \neq b$. We define the following two unary operations on labeled graphs:

- We let $\alpha_{a,b}(\mathbf{G}) = (G', \lambda)$, where G' is the graph obtained from G by introducing an edge (v, w) for each pair of vertices $v, w \in V(G)$ such that $\lambda(v) = a$ and $\lambda(w) = b$.
- We let $\rho_{a\to b}(\mathbf{G}) = (G, \lambda')$, where $\lambda'(v) = b$ if $\lambda(v) = a$ and $\lambda'(v) = \lambda(v)$ otherwise, for $v \in V(G)$.

For a set $U \subseteq V(G)$ of vertices, we let $\mathbf{G}[U]$ denote the S-labeled graph $(G[U], \lambda|_U)$, where $\lambda|_U$ denotes the restriction of λ to U. The disjoint union $\mathbf{G_1} \oplus \mathbf{G_2}$ of two S-labeled graphs $\mathbf{G_1}$ and $\mathbf{G_2}$ is defined in the obvious way.

Let $\mathbf{G} = (G, \lambda)$ and $\mathbf{G'} = (G', \lambda')$ be S-labeled graphs. We write $\mathbf{G} \to \mathbf{G'}$ if $V(G') = V(G)$ and one of the following conditions holds:

1. $\mathbf{G} = \mathbf{G_1} \oplus \mathbf{G_2}, \mathbf{G'} = \mathbf{G_1} \oplus \rho_{a\to b}(\mathbf{G_2})$ or
2. $\mathbf{G} = \mathbf{G_1} \oplus \mathbf{G_2}, \mathbf{G'} = \mathbf{G_1} \oplus \alpha_{a,b}(\mathbf{G_2})$,

where $\mathbf{G_1}, \mathbf{G_2}$ are S-labeled graphs and $a, b \in S$ with $a \neq b$. An *S-construction*[33] of a labeled graph $\mathbf{G}$ is a sequence $\mathbf{G_0}, \mathbf{G_1}, \ldots, \mathbf{G_n}$ of S-labeled graphs satisfying the following properties:

1. $\mathbf{G_0} = (G_0, \lambda_0)$ such that $E(G_0) = \emptyset$;
2. $\mathbf{G_i} \to \mathbf{G_{i+1}}$ for $0 \leq i < n$;
3. $\mathbf{G_n} = \mathbf{G}$.

The *directed clique-width* of a digraph G is the minimum cardinality of a set S such that there is an S-labeling λ of G and an S-construction of (G, λ).

The *signed incidence clique-width* of a formula is the directed clique-width of its signed incidence graph. Given a CNF formula F of length l and signed incidence clique-width k, its model count $\#F$ can be computed in time $f(k)\,l^3$ [44].

To prove that signed incidence clique-width strictly dominates incidence treewidth, we use the following lemma, which is a special case of a result by Courcelle and Olariu [33].

Lemma 37 (Courcelle and Olariu [33])**.** *Signed incidence clique-width dominates incidence treewidth.*

We will prove that there is a class of formulas of bounded signed incidence clique-width but unbounded incidence treewidth in Corollary 3 below. Jointly, these results imply the following.

Proposition 18. *Signed incidence clique-width strictly dominates incidence treewidth.*

To prove that modular incidence treewidth and signed incidence clique-width are incomparable, we first show that there are classes of formulas of bounded signed incidence clique-width but unbounded modular treewidth. We define the formula ψ_m for each $m \geq 1$ as follows.

Example 6. Let $x_1, \ldots, x_m, y_1, \ldots, y_m$ be $2m$ distinct variables. We let ψ_m consist of the clauses $C_i = \{y_i, x_1, \ldots, x_m\}$ for $1 \leq i \leq m$, along with m singleton clauses $\{x_1\}, \ldots, \{x_m\}$.

Lemma 38. *The class* $\{\, \psi_m : m \geq 1 \,\}$ *has bounded signed incidence clique-width but unbounded modular incidence treewidth.*

Proof. Let $m \geq 1$. The incidence graph $I(\psi_m)$ of ψ_m has no modules containing more than one vertex, so the modular incidence treewidth and the incidence treewidth of ψ_m coincide. Since $I(\psi_m)$ contains the complete bipartite graph $K_{m,m}$ as a subgraph, its treewidth is at least m, that is, the modular incidence treewidth of ψ_m is at least m.

For the second part of the statement, we show that there is an S-construction of (I_m, λ_1), where I_m denotes the signed incidence graph of ψ_m, λ_1 is an S-labeling of I_m such that $\lambda_1(v) = 1$ for each $v \in V(I_m)$, and $S = \{1, 2, 3, 4\}$. For $i \in [m]$, let $D_i = \{x_i\}$. Then

$$\psi_m = \{\, C_i, D_i : 1 \leq i \leq m \,\},$$
$$V(I_m) = \bigcup_{i=1}^{m} \{x_i, y_i, C_i, D_i\},$$

and

$$E(I_m) = \bigcup_{i=1}^{m} \{(x_i, D_i), (y_i, C_i)\} \cup \{\, (x_i, C_j) : 1 \leq i, j \leq m \,\}.$$

Let $G_0 = (V_0, E_0)$ denote the digraph where $V_0 = V(I_m) = V$ and $E_0 = \emptyset$, and let λ denote the S-labeling of G_0 given by

$$\lambda(D_i) = 1, \lambda(x_i) = 2, \lambda(C_i) = 3, \lambda(y_i) = 4,$$

for $1 \leq i \leq m$. The graph $\mathbf{G_0} = (G_0, \lambda)$ will be the initial graph in our S-construction. For $1 \leq i \leq m$, the graph $\mathbf{G_i}$ is simply the graph obtained from $\mathbf{G_{i-1}}$ by adding an edge from x_i to D_i. This can be expressed as

$$\mathbf{G_i} = \mathbf{G_{i-1}}[V \setminus \{x_i, D_i\}] \oplus \alpha_{2,1}(\mathbf{G_{i-1}}[\{x_i, D_i\}]). \tag{10.1}$$

Under the assumption that

$$\mathbf{G_{i-1}} = \mathbf{G_{i-1}}[V \setminus \{x_i, D_i\}] \oplus \mathbf{G_{i-1}}[\{x_i, D_i\}], \tag{10.2}$$

this implies that $\mathbf{G_{i-1}} \to \mathbf{G_i}$. Condition (10.2) is satisfied initially since G_0 does not contain any edges, and (10.1) ensures that it remains true for $1 \leq i \leq m$.

For $m + 1 \leq i \leq 2m$, we let $\mathbf{G_i}$ be the digraph obtained from $\mathbf{G_{i-1}}$ by adding an edge from y_i to C_i. That is,

$$\mathbf{G_i} = \mathbf{G_{i-1}}[V \setminus \{y_i, C_i\}] \oplus \alpha_{4,3}(\mathbf{G_{i-1}}[\{y_i, C_i\}]). \tag{10.3}$$

Again, this implies that $\mathbf{G_{i-1}} \to \mathbf{G_i}$ provided that

$$\mathbf{G_{i-1}} = \mathbf{G_{i-1}}[V \setminus \{y_i, C_i\}] \oplus \mathbf{G_{i-1}}[\{y_i, C_i\}]. \tag{10.4}$$

It follows from the construction of $\mathbf{G_m}$ and (10.3) that condition (10.4) is satisfied for each $m + 1 \leq i \leq 2m$. We let $\mathbf{G_{2m+1}} = (G_{2m+1}, \lambda) = \alpha_{2,3}(\mathbf{G_{2m}})$, introducing all edges (x_i, C_j) with $1 \leq i, j \leq m$. Since $G_{2m+1} = I_m$ and $\mathbf{G_i} \to \mathbf{G_{i+1}}$ for $0 \leq i \leq 2m$, we find that the sequence $\mathbf{G_0}, \mathbf{G_1}, \ldots, \mathbf{G_{2m+1}}$ is indeed an S-construction of (I_m, λ). □

Corollary 3. *The class* $\{ \psi_m : m \geq 1 \}$ *has bounded signed incidence clique-width but unbounded incidence treewidth.*

Proof. By Lemma 38, the class $\{ \psi_m : m \geq 1 \}$ has bounded signed incidence clique-width but unbounded modular treewidth. As modular incidence treewidth dominates incidence treewidth by Proposition 17, it follows that this class has unbounded incidence treewidth. □

Next, we show that there is a class of formulas of bounded modular incidence treewidth but unbounded signed incidence clique-width.

Example 7. For $m \geq 1$, let $x_1, \ldots, x_m$ be distinct variables. The formula φ_m is defined as the set of clauses $C_{i,j}$ for $1 \leq i, j \leq m$ and $i \neq j$, where

$$C_{i,j} = (\{x_1, \ldots, x_m\} \setminus \{x_i, x_j\}) \cup \{\neg x_i, \neg x_j\}.$$

Lemma 39. *The class* $\{ \varphi_m : m \geq 1 \}$ *has bounded modular incidence treewidth.*

Proof. The incidence graph of φ_m corresponds to the complete bipartite graph $K_{n,m}$ for $n = \binom{m}{2}$, which has treewidth m. Contracting all modules reduces $K_{n,m}$ to a single edge, so the modular incidence treewidth of φ_m is 1. □

Lemma 40 (Fischer, Makowsky, and Ravve [44])**.** *The class* $\{\varphi_m : m \geq 1\}$ *has unbounded signed incidence clique-width.*

In combination, Lemma 38, Lemma 39, and Lemma 40 prove the following.

Proposition 19. *Modular incidence treewidth and signed incidence clique-width are incomparable.*

Symmetric incidence clique-width. Clique-width can be defined in a similar way as directed clique-width, with an operation that adds undirected edges instead of directed edges between vertex classes with distinct labels [30]. As in the case of treewidth, many graph problems become tractable when restricted to graphs of bounded clique-width [31]. The clique-width analogue of a tree decomposition is called a *k-expression*. Unfortunately, it is NP-hard to compute an optimal k-expression for a given graph – in fact, it is not even known whether for fixed $k \geq 4$, there is a polynomial-time algorithm for deciding whether an input graph has clique-width at most k [43].

The difficulty of computing clique-width directly led to the introduction of rank-width (see Chapter 9), a graph parameter that is bounded for a graph class $\mathcal{C}$ if and only if $\mathcal{C}$ has bounded clique-width, but which can be computed efficiently [89]. There are several other graph parameters that are equivalent to clique-width, in particular symmetric clique-width [29], Boolean-width [23], and NLC-width [123]. We found that the dynamic programming algorithm described in Chapter 12 is most naturally presented in terms of symmetric clique-width (see Chapter 9), so we focus on this parameter. Defining the *symmetric incidence clique-width* of a formula as the symmetric clique-width of its incidence graph, we show that the model count of a formula F of length l and symmetric clique-width k can be computed in time $f(k)\, l^3 + l^{O(2^k)}$ (Theorem 16), where f is a function independent of F.

Symmetric incidence clique-width strictly dominates both modular incidence treewidth and signed incidence clique-width.

Lemma 41. *Symmetric incidence clique-width dominates signed incidence clique-width and modular incidence treewidth.*

Proof. A graph of treewidth k has clique-width at most $2^{k+1} + 1$ and clique-width is invariant under contraction of modules [33]. Moreover, a graph of clique-width k has symmetric clique-width at most 2^k [29]. It follows that symmetric incidence clique-width dominates modular incidence treewidth.

If G is the graph obtained from a directed graph G' by ignoring the directions of edges in G', then the clique-width of G is at most the directed clique-width of G' [33]. We conclude that symmetric incidence clique-width dominates signed incidence clique-width. □

In combination with Proposition 19, this yields the following result.

Proposition 20. *Symmetric incidence clique-width strictly dominates signed incidence clique-width and modular incidence treewidth.*

10.3 Hypertree-width

The most general structural parameters we consider are α-hypertree-width and β-hypertree-width [58]. These parameters are based on a generalization of treewidth to hypergraphs [57].

Definition 44 (Hypergraph)**.** A *hypergraph* is a pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a finite set and $\mathcal{E}$ is a family of nonempty subsets of $\mathcal{V}$. Elements of $\mathcal{V}$ and $\mathcal{E}$ are called *vertices* and *hyperedges* of $\mathcal{G}$, respectively. A hypergraph obtained from $\mathcal{G}$ by deleting vertices and hyperedges is called a *subhypergraph* of $\mathcal{G}$.

Definition 45 (Hypertree decomposition)**.** Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a hypergraph. A *hypertree decomposition* of $\mathcal{G}$ is a quadruple $\mathcal{T} = (T, \chi, \lambda, r)$, where $T = (V, E)$ is a tree, $r \in V$ is the *root* of T, and $\chi : V \to 2^{\mathcal{V}}$ and $\lambda : V \to 2^{\mathcal{E}}$ are labeling functions satisfying the following conditions:

(I) For every $e \in \mathcal{E}$, there is a node $t \in V$ such that $e \subseteq \chi(t)$.

(II) For every $v \in \mathcal{V}$, the set $\{\, t \in V : v \in \chi(t) \,\}$ induces a connected subtree of T.

(III) For every node $t \in V$, $\chi(t) \subseteq \bigcup \lambda(t)$.

(IV) For every node $t \in V$, if a vertex $v \in \mathcal{V}$ occurs in a hyperedge $e \in \lambda(t)$ and $v \in \chi(t')$ for some node t' below t (with respect to r), then $v \in \chi(t)$.

We define the *width* of a hypertree decomposition (T, χ, λ, r), where $T = (V, E)$, as $\max\{\, |\lambda(t)| : t \in V \,\}$. The *hypertree-width* of a hypergraph $\mathcal{G}$, in symbols $hw(\mathcal{G})$, is the minimum width of a hypertree decomposition of $\mathcal{G}$.

Ignoring the polarities of variable occurrences, we identify a CNF formula with a hypergraph and define its α-hypertree-width as follows.

Definition 46 (Associated hypergraph)**.** The *associated hypergraph* of a formula F is $\mathcal{H}(F) = (\mathrm{var}(F), \{\, \mathrm{var}(C) : C \in F \,\})$.

Definition 47 (α-hypertree-width)**.** The *α-hypertree-width* of a CNF formula F is the hypertree-width of its associated hypergraph $\mathcal{H}(F)$.

Treewidth and clique-width are *hereditary* graph parameters. That is, for every graph G, the treewidth (clique-width) of an induced subgraph of G is at most the treewidth (clique-width) of G. If F' is a formula obtained from a formula F by deleting variables or clauses of F, then $I(F')$ is an induced subgraph of $I(F)$. It follows that the incidence treewidth (clique-width) of F' is at most the incidence treewidth (clique-width) of F.

The structural parameter just defined does not have this property. In fact, a CNF formula F can easily be turned into an *α-acyclic* formula, that is, a formula of α-hypertree-width 1, by adding a single clause containing all variables of F. This observation can be used to show that bounds on the α-hypertree-width do not make #SAT any easier.

Proposition 21 (Samer and Szeider [103])**.** *#SAT remains #P-hard when restricted to α-acyclic formulas.*

Proof. Let F be a CNF formula. Consider the formula $F' = F \cup \{\mathrm{var}(F)\}$ obtained by adding a single clause containing all variables of F. Verify that F' is α-acylic: the quadruple (T, χ, λ, r) with $T = (\{r\}, \emptyset)$ such that $\chi(r) = \mathrm{var}(F)$ and $\lambda(r) = \{\mathrm{var}(F)\}$ is a hypertree-decomposition of F'. Moreover, the number of satisfying assignments $\#F$ can be computed from $\#F'$ as follows. If the assignment $\tau : \mathrm{var}(F) \to \{0, 1\}$ such that $\tau(x) = 0$ for each $x \in \mathrm{var}(F)$ does not satisfy F (which can be checked in polynomial time), then $\#F = \#F'$. Otherwise, $\#F = \#F' + 1$. □

A hereditary version of hypertree-width can be defined as follows [58].

Definition 48 (β-hypertree-width)**.** The *β-hypertree-width* of a hypergraph $\mathcal{G}$ is defined $\max\{\, hw(\mathcal{G}' : \mathcal{G}' \text{ is a subhypergraph of } \mathcal{G}\,\}$. The *$\beta$-hypertree-width* of a formula F is the β-hypertree-width of its associated hypergraph $\mathcal{H}(F)$.

Proposition 22. *α-hypertree-width strictly dominates β-hypertree-width.*

Proof. It is trivial that α-hypertree-width dominates β-hypertree-width. We have already seen that adding a single large hyperedge containing all vertices turns any hypergraph into an α-acyclic hypergraph, so the result follows from the fact that there are classes of hypergraphs of unbounded α-hypertree-width (see, for instance, Grohe and Marx [64], Example 9). □

While not as general as α-hypertree-width, the next two results prove that β-hypertree-width strictly dominates incidence clique-width.

Lemma 42 (Gottlob and Pichler [58])**.** *β-hypertree-width dominates incidence clique-width.*

Hypergraphs of β-hypertree-width 1 have been studied under the name of *β-acyclic hypergraphs* in database theory [42]. One can show that even *β-acyclic formulas*, that is, formulas whose associated hypergraphs are β-acyclic, can have arbitrary large clique-width.

Lemma 43 (Gottlob and Pichler [58])**.** *The class of β-acyclic formulas is of unbounded incidence clique-width.*

Proposition 23. *β-hypertree-width strictly dominates incidence clique-width.*

One can show that satisfiability of β-acyclic formulas can be decided in polynomial time [88]. This tractability result was recently generalized to #SAT [19]. However, it is currently open whether #SAT is polynomial-time tractable for classes of formulas of bounded β-hypertree-width.

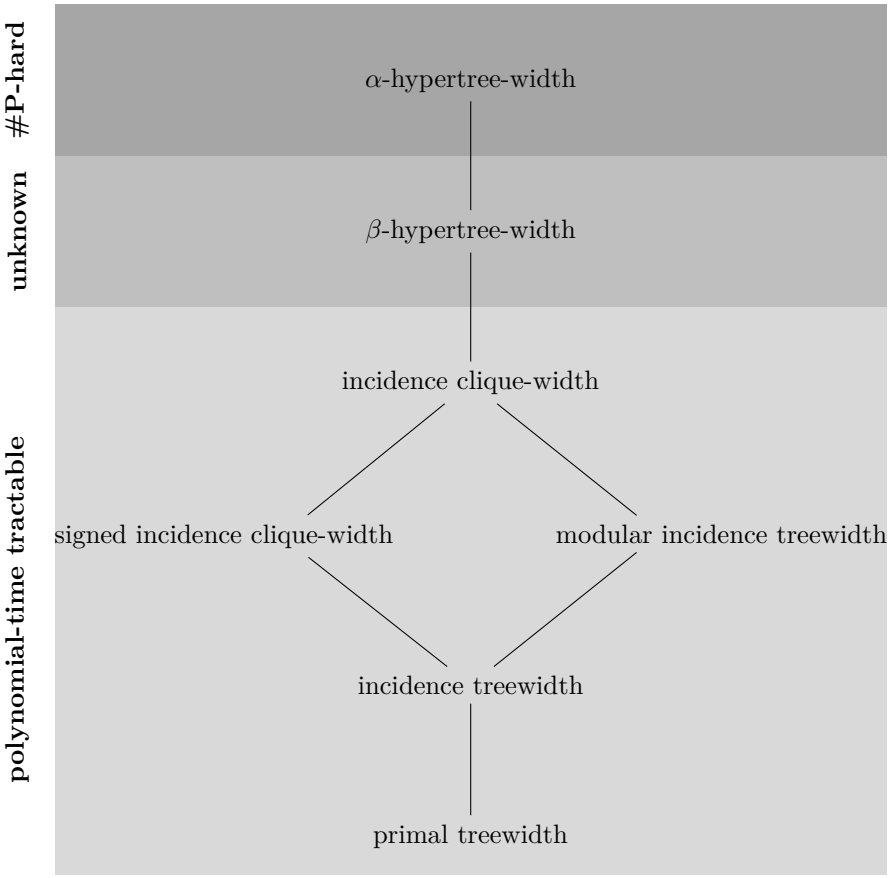

Figure 10.1: Hasse diagram of structural parameters ordered by strict domination. Regions indicate complexity of #SAT for classes of formulas where the respective parameters are bounded by a constant.

Summary

In this chapter, we surveyed structural parameters of CNF formulas based on width measures for graphs and hypergraphs. Saying that parameter p *strictly dominates* parameter q if p is bounded for any class of formulas for which is q is bounded, but not the other way around, we ordered structural parameters by strict domination. For each parameter, we further considered the complexity of #SAT for classes of formulas for which this parameters is bounded by a constant. Together, the results comparing structural parameters presented in this chapter yield the Hasse diagram shown in Figure 10.1.

CHAPTER 11

Modular Treewidth

This chapter proves tractability of #SAT for formula classes of bounded modular incidence treewidth. The modular incidence treewidth of a formula is the treewidth of its incidence graph after contraction of modules. Recall that a module in a graph is a set S of vertices such that for any vertex $v \notin S$, every vertex in S is a neighbor of v or every vertex in S is a non-neighbor of v. Contraction of modules, that is, replacing each module by a single vertex, is an important preprocessing step for a wide range of combinatorial optimization problems [66]. For #SAT, it allows us to show tractability of classes of formulas for which dynamic programming algorithms on tree decompositions of the (uncontracted) incidence graph would require exponential time. More specifically, we prove the following result (Theorem 13):

> #SAT can be solved in time $O(l^{2k+7})$ on CNF formulas of modular incidence treewidth k and length l.

The above time bound yields polynomial-time tractability of formulas whose modular incidence treewidth k is bounded by a constant, but the order of the corresponding polynomial is a (linear) function in k. The following result on SAT shows that under the widely believed assumption that FPT$\neq$W[1], this dependence on k cannot be avoided (Theorem 14).

> SAT is W[1]-hard, when parameterized by the modular incidence treewidth of the input formula.

Theorem 14 can in fact be proven by using the same reduction as the one in the proof of the result from Ordyniak et al. [88] that states that SAT is W[1]-hard when parameterized by the β-hypertree-width.

We illustrate the notion of contracting modules with an examples. Let $F_{m,n}$ be the formula constructed as follows. Given a set $\{x_1, \ldots, x_n\}$ of variables, $F_{m,n}$ that consists of m distinct clauses, each containing n literals over $\{x_1, \ldots, x_n\}$, so that every variable

occurs in every clause. It is easy to see that $F_{m,n}$ has exactly $2^n - m$ satisfying truth assignments. The vertices of the incidence graph of $F_{m,n}$ (the bipartite graph whose vertex classes consist of variables and clauses, and a variable is exactly adjacent to those clauses it occurs in) can be partitioned into two modules. By contracting these modules, the incidence graph of $F_{m,n}$ reduces to a single edge, so the modular incidence treewidth of $F_{m,n}$ is one. The incidence treewidth of $F_{m,n}$, on the other hand, is $\min(m, n)$.

11.1 Projections and Modules

A key underlying idea for our algorithm is to classify assignments according to their *projections* [76]. For an assignment $\tau : X \to \{0,1\}$, the projection $F(\tau)$ of τ onto a set F of clauses is just the set of clauses in F satisfied by τ, that is

$$F(\tau) = \{\, C \in F : C[\tau] = 1 \,\}.$$

We write $\mathsf{proj}(F, X)$ for the set of projections of assignments τ onto F, that is

$$\mathsf{proj}(F, X) = \{\, F(\tau) : \tau : X \to \{0,1\} \,\}.$$

Projections have the following simple properties, which we will use in proofs later on:

(I) An assignment $\tau : X \to \{0,1\}$ satisfies F if and only if $F(\tau) = F$.

(II) Given two assignments $\tau : X \to \{0,1\}$ and $\sigma : Y \to \{0,1\}$ such that $\tau(x) = \sigma(x)$ for each $x \in X \cap Y$, the projection of their union is simply the union of their projections, that is
$$F(\tau \cup \sigma) = F(\tau) \cup F(\sigma).$$

(III) If $\tau : X \to \{0,1\}$ is an assignment and F, F' are sets of clauses, then
$$(F \cup F')(\tau) = F(\tau) \cup F'(\tau).$$

There is an algorithm for #SAT which runs in time polynomial in the number of projections and the length of an input formula [76]. Since the number of projections of a formula can be exponential in its length even if the incidence graph is a path, we cannot directly use this algorithm for our purposes.

Example 8. To wit, consider the formula F defined as follows. Let $n > 1$ be a natural number, let $\{x_1, \ldots, x_n\}$ be a set of variables, and let $C_i = \{x_i, x_{i+1}\}$ for $i \in [n-1]$. We define F as
$$F = \{\, C_i : 1 \leq i < n \,\}.$$
The incidence graph of F is path, but F has at least $2^{\lceil n/2 \rceil}$ projections: setting every variable x_i with an even index $i \in [n]$ to 0, we are left with a formula consisting of $\lceil n/2 \rceil$ distinct unit clauses, which has $2^{\lceil n/2 \rceil}$ projections.

The algorithm presented in Section 11.2 relies on the fact that clause modules have few projections and that formulas have few projections with respect to variable modules. This essentially follows from the observation that a truth assignment $\tau : X \to \{0,1\}$ does not satisfy a unique clause C with $\mathrm{var}(C) = X$. For a set of clauses containing additional variables, such a clause corresponds to an equivalence class of the following relation.

Definition 49. For a set X of variables, we define the relation $\sim_X$ on clauses as

$$C \sim_X C' \iff C \cap \mathit{lit}(X) = C' \cap \mathit{lit}(X).$$

It is straightforward to verify that $\sim_X$ is an equivalence relation. We now prove a simple result about this relation which will allow us to establish a connection between projections and equivalence classes of $\sim_X$ in Lemma 45.

Lemma 44. *Let $\mathcal{C}$ be a set of clauses and X a set of variables such that $X \cap \mathrm{var}(C) = X \cap \mathrm{var}(C')$ for each pair $C, C' \in \mathcal{C}$. Let $\tau : X \to \{0,1\}$ be an assignment and that does not satisfy $C \in \mathcal{C}$. Then τ does not satisfy a clause $C' \in \mathcal{C}$ if and only if $C \sim_X C'$.*

Proof. Let $\tau : X \to \{0,1\}$ be an assignment and let $C \in \mathcal{C}$ be a clause not satisfied by τ. Suppose $C' \in \mathcal{C}$ is not satisfied by τ. Let $\ell \in C \cap \mathit{lit}(X)$ and let $x = \mathrm{var}(\ell)$. By assumption, $\mathrm{var}(C) \cap X = \mathrm{var}(C') \cap X$, so $x \in \mathrm{var}(C')$. Let $\ell' \in C'$ such that $\mathrm{var}(\ell') = x$. We have $\tau(\ell) = \tau(\ell') = 0$ because C and C' are not satisfied by τ, so $\ell' = \ell$. It follows that $C \cap \mathit{lit}(X) \subseteq C' \cap \mathit{lit}(X)$. A symmetric argument shows that $C \cap \mathit{lit}(X) \supseteq C' \cap \mathit{lit}(X)$, so $C \sim_X C'$. Conversely, if $C \sim_X C'$ then $C \cap \mathit{lit}(X) = C' \cap \mathit{lit}(X)$ is not satisfied by τ, so C' is not satisfied by τ. □

Lemma 45. *Let $\mathcal{C}$ be a set of clauses and X a set of variables such that $X \cap \mathrm{var}(C) = X \cap \mathrm{var}(C')$ for each pair $C, C' \in \mathcal{C}$. Then*

$$\mathsf{proj}(\mathcal{C}, X) \setminus \{\mathcal{C}\} = \{\, \mathcal{C} \setminus S : S \in \mathcal{C}/{\sim_X} \,\}.$$

Proof. If $\tau : X \to \{0,1\}$ does not satisfy $\mathcal{C}$ and $S = \mathcal{C} \setminus \mathcal{C}(\tau)$ is the set of clauses not satisfied by τ, then $S \in \mathcal{C}/{\sim_X}$ by Lemma 44. Conversely, if $S \in \mathcal{C}/{\sim_X}$ and $C \in S$, then an assignment $\tau : X \to \{0,1\}$ such that $\tau(\ell) = 0$ for each literal $\ell \in C \cap \mathit{lit}(X)$ does not satisfy C, so its projection is $\mathcal{C}(\tau) = \mathcal{C} \setminus S$ by Lemma 44. □

Lemma 46. *Let $\mathcal{C}$ be a set of clauses and X a set of variables such that $X \cap \mathrm{var}(C) = X \cap \mathrm{var}(C')$ for each pair $C, C' \in \mathcal{C}$. There is an algorithm that, given $\mathcal{C}$ and X, computes the set $\mathsf{proj}(\mathcal{C}, X)$ in time $O(l^2 + l|X|)$, where l denotes the length of $\mathcal{C}$.*

Proof. By Lemma 45 we have $\mathsf{proj}(\mathcal{C}, X) \setminus \{\mathcal{C}\} = \{\, \mathcal{C} \setminus S : S \in \mathcal{C}/{\sim_X} \,\}$. The algorithm first computes the set $Y = X \cap \mathrm{var}(\mathcal{C})$. Assuming an order on variables such that the variables in Y precede the remaining variables, it then creates a dictionary of clauses of $\mathcal{C}$, each of which is represented as a list of literals, ordered according to the order on variables. This can be accomplished in time $O(l \log(l))$. The set of equivalence classes $E = \mathcal{C}/{\sim_X}$ can then be computed in time $O(l|Y|)$. To compute $\mathsf{proj}(\mathcal{C}, X) \setminus \mathcal{C}$, the

algorithm computes the difference $\mathcal{C} \setminus S$ for each $S \in E$, which can be done in time $O(|E|l) = O(l^2)$. Finally, to determine whether $\mathcal{C} \in \mathsf{proj}(\mathcal{C}, X)$, the algorithm determines whether $|E| < 2^{|Y|}$. Letting $c = |E| + |Y|$, this can be done in time $O(c\,log(c))$. □

The number of assignments to a variable module with a given projection is easy to count. We now define a function $f_{\mathbf{x}}^F$ that does just that.

Definition 50. If $\mathbf{x}$ is a variable module of a formula F, we let

$$F_{\mathbf{x}} = \{\, C \in F : \mathbf{x} \subseteq \mathrm{var}(C) \,\},$$

and define a function $f_{\mathbf{x}}^F : 2^F \to \mathbb{N}$ as follows. For $S \subseteq F$, we let

$$f_{\mathbf{x}}^F(S) = \begin{cases} 2^{|\mathbf{x}|} - |F_{\mathbf{x}}/{\sim_{\mathbf{x}}}| & \text{if } S = F_{\mathbf{x}}, \\ 1 & \text{if } S \in \mathsf{proj}(F, \mathbf{x}) \setminus F_{\mathbf{x}}, \\ 0 & \text{otherwise.} \end{cases}$$

We omit the formula F in the superscript and simply write $f_{\mathbf{x}}$ if F is clear from the context.

Lemma 47. *For each variable module* $\mathbf{x}$ *of a formula* F *and each set* $S \subseteq F$*,*

$$f_{\mathbf{x}}(S) = |\{\, \tau : \mathbf{x} \to \{0,1\} : F(\tau) = S \,\}|.$$

Proof. If $S \notin \mathsf{proj}(F, \mathbf{x})$ then there is no assignment $\tau : \mathbf{x} \to \{0,1\}$ such that $F(\tau) = S$. By Lemma 45, $S \in \mathsf{proj}(F, \mathbf{x}) \setminus F_{\mathbf{x}}$ if and only if $F_{\mathbf{x}} \setminus S \in F_{\mathbf{x}}/{\sim_{\mathbf{x}}}$. If $\tau, \sigma : \mathbf{x} \to \{0,1\}$ are assignments such that $F(\tau) = F(\sigma) = S$ and $S \neq F_{\mathbf{x}}$, then there is a clause $C \in F_{\mathbf{x}} \setminus S$ not satisfied by τ and σ. Then $\tau(\ell) = \sigma(\ell) = 0$ for each $\ell \in C \cap \mathit{lit}(\mathbf{x})$ and thus $\tau(x) = \sigma(x)$ for each $x \in \mathrm{var}(C) \cap \mathbf{x}$. Because $C \in F_{\mathbf{x}}$ we have $\mathbf{x} \subseteq \mathrm{var}(C)$ and thus $\tau = \sigma$. That is, for each $S \in \mathsf{proj}(F, \mathbf{x}) \setminus F_{\mathbf{x}}$, there is a unique assignment $\tau : \mathbf{x} \to \{0,1\}$ such that $F(\tau) = S$. It follows that the remaining $2^{|\mathbf{x}|} - |F_{\mathbf{x}}/{\sim_{\mathbf{x}}}|$ assignments satisfy $F_{\mathbf{x}}$. □

11.2 Proof of Tractability

This section presents a dynamic programming algorithm for #SAT that runs on (nice) tree decompositions of the *modular* incidence graph. In combination with a standard algorithm for computing tree decompositions of small width, this algorithm allows us to prove tractability of #SAT parameterized by modular incidence treewidth.

We begin by describing the quantities (the "records") computed as intermediate results by our algorithm.

Definition 51. Let F be a formula and let $\mathcal{T} = (T, \chi, r)$ be a nice tree decomposition of its modular incidence graph $I^*(F)$. Let $t \in V(T)$ be a node and let $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \ldots, \mathbf{x_k})$. If S is a subset of F and $\mathbf{S} = (S_1, \ldots, S_k)$ is a k-tuple of subsets of F, we define $n_t^{\mathcal{T}}(S, \mathbf{S}) = |N_t^{\mathcal{T}}(S, \mathbf{S})|$, where $N_t^{\mathcal{T}}(S, \mathbf{S})$ is the set of truth assignments $\tau : X_t \to \{0,1\}$ with the following properties:

(A) $F(\tau) = S$,

(B) $F(\tau|_{\mathbf{x_i}}) = S_i$ for each $i \in [k]$, and

(C) $F_t \setminus F_t^b \subseteq F(\tau)$.

We omit the superscript and simply write $N_t(S, \mathbf{S})$ and $n_t(S, \mathbf{S})$ if the tree decomposition $\mathcal{T}$ is clear from the context.

The following lemma establishes necessary conditions for the set $N_t(S, \mathbf{S})$ to be nonempty.

Lemma 48. *Let F be a formula and let $\mathcal{T} = (T, \chi, r)$ be a nice tree decomposition of $I^*(F)$. Let $t \in V(T)$ and let $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \ldots, \mathbf{x_k})$. If S is a subset of F and $\mathbf{S} = (S_1, \ldots, S_k)$ is a k-tuple of subsets of F such that $n_t(S, \mathbf{S}) > 0$, then $F_t \setminus F_t^b \subseteq S$ and $f_{\mathbf{x_i}}(S_i) > 0$ for each $i \in [k]$.*

Proof. Let S be a subset of F and let $\mathbf{S} = (S_1, \ldots, S_k)$ be a k-tuple of subsets of F such that $n_t(S, \mathbf{S}) > 0$. Let $\tau \in N_t(S, \mathbf{S})$ and let $\tau_i = \tau|_{\mathbf{x_i}}$ for $i \in [k]$. Then $S = F(\tau)$ by (A) and $F_t \setminus F_t^b \subseteq F(\tau)$ by (C). Moreover, $F(\tau_i) = S_i$ for each $i \in [k]$ by (B), so $f_{\mathbf{x_i}}(S_i) > 0$ for each $i \in [k]$ by Lemma 47. □

11.2.1 Propagation Lemmas

Let F be a formula and let $\mathcal{T} = (T, \chi, r)$ be a nice tree decomposition of $I^*(F)$. To simplify the statements of results, we consider F and $\mathcal{T}$ to be fixed for the remainder of this subsection. We prove a series of what we call "propagation lemmas" that show how the values $n_t(S, \mathbf{S})$ can be obtained from the values $n_{t'}(S', \mathbf{S}')$ for children t' of t. Since $\mathcal{T}$ is a *nice* tree decomposition, each node $t \in V(T)$ is either a leaf node, an introduce node, a forget node, or a join node.

Owing to the properties of projections, assignments $\tau \cup \sigma$ and $\tau' \cup \sigma$ have the same projections onto a formula F as long as $F(\tau) = F(\tau')$. As a consequence, membership of an assignment $\tau \in N_t(S, \mathbf{S})$ is robust under replacing the restriction $\tau|_{\mathbf{x}}$ with a different assignment $\tau'|_{\mathbf{x}}$, as long as $F(\tau'|_{\mathbf{x}}) = F(\tau|_{\mathbf{x}})$. This allows us to represent sets $N_t(S, \mathbf{S})$ as products of sets of the following form.

Definition 52. Let $t \in V(T)$ and let $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \ldots, \mathbf{x_k})$. For a subset S of F and a k-tuple $\mathbf{S} = (S_1, \ldots, S_k)$ of subsets of F, we define the set $A_t^-(S, \mathbf{S})$ as

$$A_t^-(S, \mathbf{S}) = \{\, \sigma : X_t \setminus X_t^b \to \{0, 1\} : S = \bigcup_{i=1}^{k} S_i \cup F(\sigma) \,\}.$$

For each $i \in [k]$, we further let $A_t^i(S_i)$ denote the set

$$A_t^i(S_i) = \{\, \tau_i : \mathbf{x_i} \to \{0, 1\} : F(\tau_i) = S_i \,\}.$$

Finally, we let

$$A_t(S, \mathbf{S}) = \prod_{i=1}^{k} A_t^i(S_i) \times A_t^-(S, \mathbf{S}).$$

We prove that the set $A_t(S, \mathbf{S})$ is just an alternative representation of the set $N_t(S, \mathbf{S})$.

Lemma 49. *Let $t \in V(T)$ and let $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \ldots, \mathbf{x_k})$. Let S be a subset of F such that $F_t \setminus F_t^b \subseteq S$, and let $\mathbf{S} = (S_1, \ldots, S_k)$ be a k-tuple of subsets of F. Moreover, let $\tau : X_t \to \{0, 1\}$ be an assignment, let $\sigma = \tau|_{X_t \setminus X_t^b}$, and let $\tau_i = \tau|_{\mathbf{x_i}}$ for each $i \in [k]$. Then*

$$\tau \in N_t(S, \mathbf{S}) \iff (\tau_1, \ldots, \tau_k, \sigma) \in A_t(S, \mathbf{S}).$$

Proof. Suppose $\tau \in N_t(S, \mathbf{S})$. By (B), we have $F(\tau_i) = S_i$ and thus $\tau_i \in A_t^i(S_i)$ for each $i \in [k]$. Using (A), we get

$$S = F(\tau) = \bigcup_{i=1}^{k} F(\tau_i) \cup F(\sigma) = \bigcup_{i=1}^{k} S_i \cup F(\sigma),$$

so $\sigma \in A_t^-(S, \mathbf{S})$ and $(\tau_1, \ldots, \tau_k, \sigma) \in A_t(S, \mathbf{S})$. For the converse, suppose $(\tau_1, \ldots, \tau_k, \sigma) \in A_t(S, \mathbf{S})$. Then $\tau_i \in A_t^i(S_i)$ and thus $F(\tau_i) = S_i$ for each $i \in [k]$, so (B) is satisfied. Moreover, $\sigma \in A_t^-(S, \mathbf{S})$, so

$$F(\tau) = \bigcup_{i=1}^{k} F(\tau_i) \cup F(\sigma) = \bigcup_{i=1}^{k} S_i \cup F(\sigma) = S$$

and (A) holds. By assumption, $F_t \setminus F_t^b \subseteq S$, so (C) holds as well. We conclude that $\tau \in N_t(S, \mathbf{S})$. □

Under conditions that are satisfied if $N_t(S, \mathbf{S})$ is nonempty (Lemma 48), this allows us to relate the sizes of sets $A_t^-(S, \mathbf{S})$ and $N_t(S, \mathbf{S})$ in the following way.

Lemma 50. *Let $t \in V(T)$ such that $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \ldots, \mathbf{x_k})$. Let S be a subset of F such that $F_t \setminus F_t^b \subseteq S$, and let $\mathbf{S} = (S_1, \ldots, S_k)$ be a k-tuple of subsets of F such that $f_{\mathbf{x_i}}(S_i) > 0$ for each $i \in [k]$. Then*

$$|A_t^-(S, \mathbf{S})| = \frac{n_t(S, \mathbf{S})}{\prod_{i=1}^{k} f_{\mathbf{x_i}}(S_i)}. \tag{11.1}$$

Proof. It is immediate from Lemma 49 that $|A_t(S, \mathbf{S})| = n_t(S, \mathbf{S})$. In combination with the definition of $A_t(S, \mathbf{S})$, this yields

$$n_t(S, \mathbf{S}) = |A_t(S, \mathbf{S})| = \prod_{i=1}^{k} |A_t^i(S_i)||A_t^-(S, \mathbf{S})|.$$

By Lemma 47, $\prod_{i=1}^{k} |A_t^i(S_i)| = \prod_{i=1}^{k} f_{\mathbf{x_i}}(S_i)$. By assumption, $\prod_{i=1}^{k} f_{\mathbf{x_i}}(S_i) > 0$ and (11.1) follows. □

If P is a set and $\mathbf{S} = (S_1, \ldots, S_k)$ is a k-tuple of sets, we write $\mathbf{S} : P$ for the $(k+1)$-tuple $(S_1, \ldots, S_k, P)$ obtained by appending P to $\mathbf{S}$. To prove the propagation lemma for variable introduce nodes, we first establish the following result.

Lemma 51. *Let $t \in V(T)$ be a variable introduce node with child $t' \in V(T)$, and let $\chi_{\mathbf{v}}(t') = (\mathbf{x_1}, \ldots, \mathbf{x_k})$ and $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \ldots, \mathbf{x_k}, \mathbf{x})$. If S and P are subsets of F and $\mathbf{S} = (S_1, \ldots, S_k)$ is a k-tuple of subsets of F, then*

$$|A_t^-(S, \mathbf{S} : P)| = \sum_{S' \colon S' \cup P = S} |A_{t'}^-(S', \mathbf{S})|. \tag{11.2}$$

Proof. We first show that

$$A_t^-(S, \mathbf{S} : P) = \bigcup_{S' \colon S' \cup P = S} A_{t'}^-(S', \mathbf{S}). \tag{11.3}$$

Let $\sigma \in A_t^-(S, \mathbf{S} : P)$. Then

$$\bigcup_{i=1}^{k} S_i \cup P \cup F(\sigma) = S, \tag{11.4}$$

or equivalently

$$(\bigcup_{i=1}^{k} S_i \cup F(\sigma)) \cup (P \setminus (\bigcup_{i=1}^{k} S_i \cup F(\sigma))) = S.$$

Because the union on the left-hand side is disjoint, this gives

$$\bigcup_{i=1}^{k} S_i \cup F(\sigma) = S \setminus (P \setminus (\bigcup_{i=1}^{k} S_i \cup F(\sigma))).$$

Let $S' = S \setminus (P \setminus (\bigcup_{i=1}^{k} S_i \cup F(\sigma)))$. Then

$$\bigcup_{i=1}^{k} S_i \cup F(\sigma) = S',$$

so $\sigma \in A_{t'}^-(S', \mathbf{S})$, and

$$S' \cup P = (S \setminus (P \setminus (\bigcup_{i=1}^{k} S_i \cup F(\sigma)))) \cup P = S,$$

since $P \subseteq S$ by (11.4). For the converse, let $\sigma \in A_{t'}^-(S', \mathbf{S})$ such that $S' \cup P = S$. Then

$$P \cup \bigcup_{i=1}^{k} S_i \cup F(\sigma) = P \cup S' = S,$$

so $\sigma \in A_t^-(S, \mathbf{S})$. For each assignment $\sigma : X_{t'} \setminus X_{t'}^b \to \{0, 1\}$, there is a unique set S' such that $\sigma \in A_{t'}^-(S', \mathbf{S})$, so the union on the right-hand side of (11.3) is disjoint and (11.2) follows. □

For variable introduce and forget nodes, we will assume that the variable module $\mathbf{x}$ introduced or forgotten is the last element in the tuple given by $\chi_{\mathbf{v}}$. The proofs below can be easily adapted to cases where $\mathbf{x}$ is in a different position.

Lemma 52 (Variable introduce node)**.** *Let $t, t' \in V(T)$ such that t' is the child of t in T and such that $\chi_{\mathbf{v}}(t') = (\mathbf{x_1}, \ldots, \mathbf{x_k})$ and $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \ldots, \mathbf{x_k}, \mathbf{x})$. If S and P are subsets of F and $\mathbf{S} = (S_1, \ldots, S_k)$ is a k-tuple of subsets of F, then*

$$n_t(S, \mathbf{S} : P) = f_{\mathbf{x}}(P) \sum_{S' \colon S' \cup P = S} n_{t'}(S', \mathbf{S}). \tag{11.5}$$

Proof. We distinguish two cases. First, assume that $F_t \setminus F_t^b \subseteq S$ and that $f_{\mathbf{x_i}}(S_i) > 0$ for each $i \in [k]$ as well as $f_{\mathbf{x}}(P) > 0$. Then we can apply Lemma 50 and Lemma 51 to obtain

$$\frac{n_t(S, \mathbf{S} : P)}{\prod_{i=1}^{k} f_{\mathbf{x_i}}(S_i)\, f_{\mathbf{x}}(P)} = |A_t^-(S, \mathbf{S} : P)| = \sum_{S' \colon S' \cup P = S} |A_{t'}^-(S', \mathbf{S})|. \tag{11.6}$$

By assumption, $f_{\mathbf{x}}(P) > 0$, so it follows from Lemma 47 that $P \in \mathsf{proj}(F, \mathbf{x})$. Because $\mathcal{T}$ is a tree decomposition of $I^*(F)$ we have $\mathbf{x} \cap \mathrm{var}(F_t \setminus F_t^b) = \emptyset$, so an assignment to $\mathbf{x}$ cannot satisfy a clause in $F_t \setminus F_t^b$. Because P is the projection of an assignment to $\mathbf{x}$ onto F, this implies $P \cap (F_t \setminus F_t^b) = \emptyset$. Combining this with the assumption that $F_t \setminus F_t^b \subseteq S$, we conclude that $F_t \setminus F_t^b = F_{t'} \setminus F_{t'}^b \subseteq S'$ for any set S' such that $S = S' \cup P$. This allows us to use Lemma 50 again and write the sum in the rightmost expression of (11.6) as

$$\begin{aligned} \sum_{S' \colon S' \cup P = S} |A_{t'}^-(S', \mathbf{S})| &= \sum_{S' \colon S' \cup P = S} \frac{n_{t'}(S', \mathbf{S})}{\prod_{i=1}^{k} f_{\mathbf{x_i}}(S_i)} \\ &= \frac{1}{\prod_{i=1}^{k} f_{\mathbf{x_i}}(S_i)} \sum_{S' \colon S' \cup P = S} n_{t'}(S', \mathbf{S}). \end{aligned}$$

In combination with (11.6), we get

$$\frac{n_t(S, \mathbf{S} : P)}{\prod_{i=1}^{k} f_{\mathbf{x_i}}(S_i)\, f_{\mathbf{x}}(P)} = \frac{1}{\prod_{i=1}^{k} f_{\mathbf{x_i}}(S_i)} \sum_{S' \colon S' \cup P = S} n_{t'}(S', \mathbf{S}).$$

Multiplying with $\prod_{i=1}^{k} f_{\mathbf{x_i}}(S_i)$, we obtain (11.5).

Suppose the assumption for the first case does not hold. If $F_t \setminus F_t^b \not\subseteq S$ then $n_t(S, \mathbf{S} : P) = 0$ by Lemma 48. Since $F_{t'} \setminus F_{t'}^b = F_t \setminus F_t^b$, we also have $n_{t'}(S', \mathbf{S}) = 0$ for every $S' \subseteq S$, so both sides of (11.5) evaluate to 0 in this case. If $f_{\mathbf{x}}(P) = 0$ then the right-hand side of (11.5) is 0 and $n_t(S, \mathbf{S} : P) = 0$ by Lemma 48, so again both sides of (11.5) evaluate to 0. Finally, suppose there is an $i \in [k]$ such that $f_{\mathbf{x_i}}(S_i) = 0$. It again follows from Lemma 48 that $n_t(S, \mathbf{S} : P) = 0$ and $n_{t'}(S', \mathbf{S}) = 0$ for any $S' \subseteq F$, so (11.5) holds. □

Lemma 53 (Variable forget node)**.** *Let $t, t' \in V(T)$ such that t' is the child of t in T and such that $\chi_{\mathbf{v}}(t') = (\mathbf{x_1}, \ldots, \mathbf{x_k}, \mathbf{x})$ and $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \ldots, \mathbf{x_k})$. If S is a subset of F and $\mathbf{S} = (S_1, \ldots, S_k)$ is a tuple of subsets of F, then*

$$n_t(S, \mathbf{S}) = \sum_{P \subseteq F} n_{t'}(S, \mathbf{S} : P). \tag{11.7}$$

Proof. We prove that $N_t(S, \mathbf{S}) = U$, where the set U is defined as

$$U = \bigcup_{P \subseteq F} n_{t'}(S, \mathbf{S} : P).$$

Let $\tau \in N_t(S, \mathbf{S})$ and let $P = F(\tau|_{\mathbf{x}})$. Since $X_t = X_{t'}$ and $F_t \setminus F_t^b = F_{t'} \setminus F_{t'}^b$, it follows from our choice of τ and P that $\tau \in n_{t'}(S, \mathbf{S} : P)$. In particular, $\tau \in U$. The converse is immediate. □

Lemma 54 (Clause introduce node)**.** *Let $t, t' \in V(T)$ be nodes such that t' is the child of t in T and such that $\chi_c(t) = \chi_c(t') \cup \{\mathcal{C}\}$ for some clause module $\mathcal{C}$ of F. Then*

$$n_t(S, \mathbf{S}) = n_{t'}(S, \mathbf{S}). \tag{11.8}$$

Proof. Since $\chi_v(t) = \chi_v(t')$, $X_t = X_{t'}$, and

$$F_t \setminus F_t^b = (F_{t'} \cup \mathcal{C}) \setminus (F_{t'}^b \cup \mathcal{C}) = F_{t'} \setminus F_{t'}^b,$$

the conditions for membership in $N_t(S, \mathbf{S})$ and $N_{t'}(S, \mathbf{S})$ coincide. □

Lemma 55 (Clause forget node)**.** *Let $t, t' \in V(T)$ be nodes such that t' is the child of t in T and such that $\chi_c(t) = \chi_c(t') \setminus \{\mathcal{C}\}$ for some clause module $\mathcal{C}$ of F. Further, let $\chi_{\mathbf{v}}(t) = \chi_{\mathbf{v}}(t') = (\mathbf{x_1}, \ldots, \mathbf{x_k})$. Let S be a subset of F and let $\mathbf{S}$ be a k-tuple of subsets of F. Then*

$$n_t(S, \mathbf{S}) = \begin{cases} n_{t'}(S, \mathbf{S}) & \text{if } \mathcal{C} \subseteq S, \\ 0 & \text{otherwise.} \end{cases} \tag{11.9}$$

Proof. Since $\mathcal{C} \subseteq F_t \setminus F_t^b$, it follows from Lemma 48 that $n_t(S, \mathbf{S}) = 0$ if $\mathcal{C} \nsubseteq S$. Otherwise, if $\mathcal{C} \subseteq S$ then $N_{t'}(S, \mathbf{S}) \subseteq N_t(S, \mathbf{S})$, since any assignment $\tau : X_t \to \{0, 1\}$ such that $F_{t'} \setminus F_{t'}^b \subseteq F(\tau)$ and $S = F(\tau)$ satisfies $F_t \setminus F_t^b = (F_{t'} \setminus F_{t'}^b) \cup \mathcal{C} \subseteq F(\tau)$; the inclusion $N_t(S, \mathbf{S}) \subseteq N_{t'}(S, \mathbf{S})$ is trivial, so $N_t(S, \mathbf{S}) = N_{t'}(S, \mathbf{S})$ in this case. □

To simplify the proof of the propagation lemma for join nodes, we first establish the following result.

Lemma 56. *Let $t, t', t'' \in V(T)$ such that t' and t'' are the children of t in $\mathcal{T}$, and let $\chi_{\mathbf{v}}(t) = \chi_{\mathbf{v}}(t') = \chi_{\mathbf{v}}(t'') = (\mathbf{x_1}, \ldots, \mathbf{x_k})$. If S is a subset of F and $\mathbf{S} = (S_1, \ldots, S_k)$ is a k-tuple of subsets of F, then*

$$|A_t^-(S, \mathbf{S})| = \sum_{\substack{S', S'': \\ S = S' \cup S''}} |A_{t'}^-(S', \mathbf{S})||A_{t''}^-(S'', \mathbf{S})|. \tag{11.10}$$

Proof. We first show that

$$A_t^-(S,\mathbf{S}) = \bigcup_{\substack{S',S'':\\ S=S'\cup S''}} \{\,\sigma'\cup\sigma'' : \sigma'\in A_{t'}^-(S',\mathbf{S}), \sigma''\in A_{t''}^-(S'',\mathbf{S})\,\}. \tag{11.11}$$

Let $\sigma \in A_t^-(S,\mathbf{S})$, and let σ' and σ'' denote the restrictions of σ to $X_{t'}$ and $X_{t''}$, respectively. Defining $S' = \bigcup_{i=1}^k S_i \cup F(\sigma')$ and $S'' = \bigcup_{i=1}^k S_i \cup F(\sigma'')$, we immediately get $\sigma' \in A_{t'}^-(S',\mathbf{S})$ and $\sigma'' \in A_{t''}^-(S'',\mathbf{S})$. Moreover,

$$S'\cup S'' = \bigcup_{i=1}^k S_i \cup F(\sigma') \cup F(\sigma'') = \bigcup_{i=1}^k S_i \cup F(\sigma'\cup\sigma'') = \bigcup_{i=1}^k S_i \cup F(\sigma) = S,$$

where the last equality follows from $\sigma \in A_t^-(S,\mathbf{S})$. This proves that the left-to-right inclusion of (11.11). For the right-to-left inclusion, let $\sigma' \in A_{t'}^-(S',\mathbf{S})$ and $\sigma'' \in A_{t''}^-(S'',\mathbf{S})$ such that $S = S'\cup S''$. Then

$$\bigcup_{i=1}^k S_i \cup F(\sigma'\cup\sigma'') = (\bigcup_{i=1}^k S_i \cup F(\sigma')) \cup (\bigcup_{i=1}^k S_i \cup F(\sigma'')) = S'\cup S'' = S,$$

that is, $\sigma'\cup\sigma'' \in A_t^-(S,\mathbf{S})$. We conclude that (11.11) holds. For assignments $\sigma' : X_{t'} \setminus X_{t'}^b \to \{0,1\}$ and $\sigma'' : X_{t''} \setminus X_{t'}^b \to \{0,1\}$, the sets S' and S'' such that $\sigma' \in A_{t'}(S',\mathbf{S})$ and $\sigma'' \in A_{t'}(S'',\mathbf{S})$ are unique. Accordingly, the union on the right hand side of (11.11) is disjoint, and (11.10) follows. □

Lemma 57. *Let $t,t',t'' \in V(T)$ such that t' and t'' are the children of t in T, and let $\chi_{\mathbf{v}}(t) = \chi_{\mathbf{v}}(t') = \chi_{\mathbf{v}}(t'') = (\mathbf{x_1},\ldots,\mathbf{x_k})$. Let S be a subset of F and let $\mathbf{S} = (S_1,\ldots,S_k)$ be a k-tuple of subsets of F such that $F_t \setminus F_t^b \subseteq S$ and $f_{\mathbf{x_i}}(S_i) > 0$ for each $i \in [k]$. Then*

$$n_t(S,\mathbf{S}) = \frac{1}{\prod_{i=1}^k f_{\mathbf{x}_i}(S_i)} \sum_{\substack{S',S'':\\ S=S'\cup S''}} n_{t'}(S',\mathbf{S})\; n_{t''}(S'',\mathbf{S}). \tag{11.12}$$

Proof. From Lemma 50 and Lemma 56, we get

$$\frac{n_t(S,\mathbf{S})}{\prod_{i=1}^k f_{\mathbf{x_i}}(S_i)} = |A_t^-(S,\mathbf{S})| = \sum_{\substack{S',S'':\\ S=S'\cup S''}} |A_{t'}^-(S',\mathbf{S})||A_{t''}^-(S'',\mathbf{S})|. \tag{11.13}$$

We argue that the sum on the right-hand side of (11.13) can be restricted to pairs S', S'' of sets such that $F_{t'} \setminus F_{t'}^b \subseteq S'$ and $F_{t''} \setminus F_{t''}^b \subseteq S''$. Suppose both $A_{t'}^-(S',\mathbf{S})$ and $A_{t''}^-(S'',\mathbf{S})$ are nonempty and $S'\cup S'' = S$. Let $\sigma' \in A_{t'}^-(S',\mathbf{S})$ and let $\sigma'' \in A_{t''}^-(S'',\mathbf{S})$. Then $S' = \bigcup_{i=1}^k S_i \cup F(\sigma')$, $S'' = \bigcup_{i=1}^k S_i \cup F(\sigma'')$, and $S = \bigcup_{i=1}^k S_i \cup F(\sigma') \cup F(\sigma'')$. By assumption, $F_t \setminus F_t^b \subseteq S$, so

$$F_t \setminus F_t^b = (F_{t'} \setminus F_{t'}^b) \cup (F_{t''} \setminus F_{t''}^b) \subseteq \bigcup_{i=1}^k S_i \cup F(\sigma') \cup F(\sigma''). \tag{11.14}$$

Because $\mathcal{T}$ is a tree decomposition of $I^*(F)$ we have $(X_{t'} \setminus X^b_{t'}) \cap \mathrm{var}(F_{t''} \setminus F^b_{t''}) = \emptyset$ and symmetrically, $(X_{t''} \setminus X^b_{t''}) \cap \mathrm{var}(F_{t'} \setminus F^b_{t'}) = \emptyset$. Thus, σ' cannot satisfy a clause in $F_{t''} \setminus F^b_{t''}$ and σ'' cannot satisfy a clause in $F_{t'} \setminus F^b_{t'}$, so $(F_{t''} \setminus F^b_{t''}) \cap F(\sigma') = \emptyset$ and $(F_{t'} \setminus F^b_{t'}) \cap F(\sigma'') = \emptyset$. In combination with (11.14), this implies $F_{t'} \setminus F^b_{t'} \subseteq \bigcup_{i=1}^k S_i \cup F(\sigma') = S'$ and $F_{t''} \setminus F^b_{t''} \subseteq \bigcup_{i=1}^k S_i \cup F(\sigma'') = S''$ as claimed. We can now use Lemma 50 to write the right-hand side of (11.13) equivalently as

$$\begin{aligned}\sum_{\substack{S',S'':\\ S=S'\cup S''}} |A^-_{t'}(S',\mathbf{S})||A^-_{t''}(S'',\mathbf{S})| &= \sum_{\substack{S',S'':\\ S=S'\cup S''}} \frac{n_{t'}(S',\mathbf{S})}{\prod_{i=1}^k f_{\mathbf{x_i}}(S_i)} \frac{n_{t''}(S'',\mathbf{S})}{\prod_{i=1}^k f_{\mathbf{x_i}}(S_i)} \\ &= \frac{1}{(\prod_{i=1}^k f_{\mathbf{x_i}}(S_i))^2} \sum_{\substack{S',S'':\\ S=S'\cup S''}} n_{t'}(S',\mathbf{S})\, n_{t''}(S'',\mathbf{S}).\end{aligned}$$

In combination with (11.13), we get

$$\frac{n_t(S,\mathbf{S})}{\prod_{i=1}^k f_{\mathbf{x_i}}(S_i)} = \frac{1}{(\prod_{i=1}^k f_{\mathbf{x_i}}(S_i))^2} \sum_{\substack{S',S'':\\ S=S'\cup S''}} n_{t'}(S',\mathbf{S})\, n_{t''}(S'',\mathbf{S}).$$

Multiplying both sides with $\prod_{i=1}^k f_{\mathbf{x_i}}(S_i)$, we obtain (11.12). □

Lemma 58 (Leaf node)**.** *Let $t \in V(T)$ be a leaf node and with $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \ldots, \mathbf{x_k})$. If S is a subset of F and $\mathbf{S} = (S_1, \ldots, S_k)$ is a k-tuple of subsets of F, then*

$$n_t(S,\mathbf{S}) = \begin{cases} 0 & \text{if } S \neq \bigcup_{i=1}^k S_i, \\ \prod_{i=1}^k f_{\mathbf{x_i}}(S_i) & \text{otherwise.} \end{cases} \tag{11.15}$$

Proof. Let S be a subset of F and let $\mathbf{S} = (S_1, \ldots, S_k)$ be a k-tuple of subsets of F. Suppose $N_t(S,\mathbf{S})$ is nonempty and let $\tau \in N_t(S,\mathbf{S})$. Let $\tau_i = \tau|_{\mathbf{x_i}}$ for each $i \in [k]$. We have $S_i = F(\tau_i)$ for each $i \in [k]$ by (B), and $S = F(\tau)$ by (A), so

$$S = F(\tau) = \bigcup_{i=1}^k F(\tau_i) = \bigcup_{i=1}^k S_i.$$

Now suppose $S = \bigcup_{i=1}^k S_i$. Since t is a leaf node we have $F_t \setminus F^b_t = \emptyset \subseteq S$. If there is an $i \in [k]$ such that $f_{\mathbf{x_i}}(S_i) = 0$ then $n_t(S,\mathbf{S}) = 0$ by Lemma 48 and $\Pi_{i=1}^k f_{\mathbf{x_i}}(S_i) = 0$. Otherwise, Lemma 50 yields

$$|A^-_t(S,\mathbf{S})| = \frac{n_t(S,\mathbf{S})}{\prod_{i=1}^k f_{\mathbf{x_i}}(S_i)}.$$

Since $X_t \setminus X^b_t = \emptyset$, the empty assignment ε is the only assignment to $X_t \setminus X^b_t$. We have $F(\varepsilon) = \emptyset$ and $S = \bigcup_{i=1}^k S_i$ by assumption, so $\varepsilon \in A^-_t(S,\mathbf{S})$ and thus $|A^-_t(S,\mathbf{S})| = 1$. □

If $t \in V(T)$ is a leaf node and $\chi_v(t) = \emptyset$, the empty assignment ε is contained in $N_t(\emptyset, ())$. The condition on S in (11.15) trivializes to $S = \emptyset$, and the empty product correctly computes the cardinality of $N_t(\emptyset, ())$ as 1.

Lemma 59 (Root node)**.** *Let* $\chi_{\mathbf{v}}(r) = (\mathbf{x_1}, \ldots, \mathbf{x_k})$*. We have*

$$\#F = \sum_{\mathbf{S} \in F^k} n_r(F, \mathbf{S}). \tag{11.16}$$

Proof. We prove that the set of satisfying assignments of F is

$$U = \bigcup_{\mathbf{S} \in F^k} N_r(F, \mathbf{S}).$$

If $\tau \in N_r(F, \mathbf{S})$ then $F(\tau) = F$ and $X_r = \mathrm{var}(F)$, so τ is a satisfying assignment of F. For the converse, let $\tau : \mathrm{var}(F) \to \{0, 1\}$ be a satisfying assignment of F, let $S_i = F(\tau|_{\mathbf{x_i}})$ for $i \in [k]$, and let $\mathbf{S} = (S_1, \ldots, S_k)$. We have $F(\tau) = F$ and $F_r \setminus F_r^b \subseteq F$, so $\tau \in N_r(F, \mathbf{S})$. The sets $N_r(F, \mathbf{S})$ and $N_r(F, \mathbf{S}')$ are disjoint for distinct $\mathbf{S}$ and $\mathbf{S}'$, so

$$|U| = \sum_{\mathbf{S} \in F^k} n_r(F, \mathbf{S}).$$

□

11.2.2 The Algorithm

We now describe our dynamic programming algorithm MTW-ModelCount for computing the number $\#F$ of satisfying assignments of an input formula F. MTW-ModelCount takes as input a formula F and a tree decomposition $\mathcal{T} = (T, \chi, r)$ of its modular incidence graph $I^*(F)$.

Data structures. We assume the algorithm has access to a dictionary M_t for each $t \in V(T)$. If $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \ldots, \mathbf{x_k})$, M_t is addressed by pairs $(S, \mathbf{S})$, where S is a subset of F and $\mathbf{S}$ is a k-tuple of subsets of F. The value of $M_t[S, \mathbf{S}]$ is a non-negative integer. The dictionary M_t implements three operations.

- One can *create* a new (zero-valued) entry $M_t[S, \mathbf{S}]$. If the entry already exists, this operation has no effect.
- One can *search* for an entry $M_t[S, \mathbf{S}]$ and retrieve its value. If the entry does not exist, this operation returns 0.
- One can *set* the value of an entry $M_t[S, \mathbf{S}]$. If the entry does not exist, this operation has no effect.

We further assume that for each variable module $\mathbf{x}$ of F, there is a dictionary representing the function $f_{\mathbf{x}}$ which lets the algorithm search for and retrieve the value $f_{\mathbf{x}}(P)$ for a given projection $P \in \mathsf{proj}(F, \mathbf{x})$.

Initialization. For each variable module $\mathbf{x}$ of F, the algorithm precomputes the set $\mathsf{proj}(F, \mathbf{x})$ and the values $f_{\mathbf{x}}(P)$ for each projection $P \in \mathsf{proj}(F, \mathbf{x})$, and creates the dictionary representing $f_{\mathbf{x}}$.

For each node $t \in V(T)$, it then initializes the dictionary M_t as follows. Let $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \ldots, \mathbf{x_d})$, and let $\chi_c(t) = \{\mathcal{C}_1, \ldots, \mathcal{C}_e\}$. The algorithm computes the set $\mathsf{proj}(\mathcal{C}_j, X_t)$ for each $i \in [e]$. It then goes through all $(d+e)$-tuples $(S_1, \ldots, S_d, S'_1, \ldots, S'_e)$ with $S_i \in \mathsf{proj}(F, \mathbf{x_i})$ for each $i \in [d]$ and $S'_j \in \mathsf{proj}(\mathcal{C}_j, X_t)$ for each $j \in [e]$, and creates the entry $M_t[S, \mathbf{S}]$, where $\mathbf{S} = (S_1, \ldots, S_d)$, and

$$S = (F_t \setminus F_t^b) \cup \bigcup_{i=1}^{e} S'_i \cup \bigcup_{i=1}^{d} S_i.$$

Processing phase. After initialization, the algorithm processes nodes of T in a bottom-up manner, starting from the leaves. It finds a node $t \in V(T)$ whose children have already been processed, and then processes t based on its type:

1. Let t be a leaf node with $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \ldots, \mathbf{x_k})$. The algorithm goes through all k-tuples $\mathbf{S} = (S_1, \ldots, S_k)$ such that $S_i \in \mathsf{proj}(F, \mathbf{x_i})$ for $i \in [k]$, setting

$$M_t[\bigcup_{i=1}^{k} S_i, \mathbf{S}] := \prod_{i=1}^{k} f_{\mathbf{x}}(S_i).$$

2. Let t be a variable introduce node with child t', and let $\chi_v(t) = \chi_v(t') \cup \{\mathbf{x}\}$. For each projection $P \in \mathsf{proj}(F, \mathbf{x})$ and each entry $M_{t'}[S', \mathbf{S}]$, the algorithm

   i. computes the set $S := S' \cup P$

   ii. and sets
   $$M_t[S, \mathbf{S} : P)] := M_t[S, \mathbf{S} : P] + f_{\mathbf{x}}(P) M_{t'}[S', \mathbf{S}].$$

3. Let t be a variable forget node with child t'. For each entry $M_{t'}[S, \mathbf{S} : P]$, the algorithm sets
$$M_t[S, \mathbf{S}] := M_t[S, \mathbf{S}] + M_{t'}[S, \mathbf{S} : P].$$

4. Let t be a clause introduce node with child t'. For each entry $M_{t'}[S, \mathbf{S}]$, the algorithm sets $M_t[S, \mathbf{S}] := M'_t[S, \mathbf{S}]$.

5. Let t be a clause forget node with child t', and let $\chi_c(t) = \chi_c(t') \setminus \{\mathcal{C}\}$ for a clause module $\mathcal{C}$ of F. For each entry $M_{t'}[S, \mathbf{S}]$ such that $\mathcal{C} \subseteq S$, the algorithm sets $M_t[S, \mathbf{S}] := M_{t'}[S, \mathbf{S}]$.

6. Let t be a join node with children t' and t'', and let $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \ldots, \mathbf{x_k})$. The algorithm goes through each pair $M_{t'}[S', \mathbf{S}]$, $M_{t''}[S'', \mathbf{S}]$ of entries, where $\mathbf{S} = (S_1, \ldots, S_k)$, and executes the following steps:

   i. It computes the union $S := S' \cup S''$

ii. as well as the product $p := \prod_{i=1}^{k} f_{\mathbf{x_i}}(S_i)$ and quotient $q := \frac{M_{t'}[S', \mathbf{S}]}{p}$,

iii. and sets

$$M_t[S, \mathbf{S}] := M_t[S, \mathbf{S}] + qM_{t''}[S'', \mathbf{S}].$$

Output. After all nodes have been processed, the algorithm computes and outputs $\sum_{\mathbf{S}} M_r[F, \mathbf{S}]$.

11.2.3 Correctness

In this subsection, we will show that MTW-ModelCount correctly computes the number of satisfying assignments. We begin by arguing that the algorithm creates an entry $M_t[S, \mathbf{S}]$ in its initialization phase whenever $n_t(S, \mathbf{S}) > 0$. To this end, we first establish the following result.

Lemma 60. *Let F be a formula and let $\mathcal{T} = (T, \chi, r)$ be a tree decomposition of $I^*(F)$. Let $t \in V(T)$ with $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \ldots, \mathbf{x_d})$ and $\chi_c(t) = \{\mathcal{C}_1, \ldots, \mathcal{C}_e\}$. If S is a subset of F and $\mathbf{S} = (S_1, \ldots, S_d)$ is a d-tuple of subsets of F such that $n_t(S, \mathbf{S}) > 0$, then there is a projection $S'_i \in \mathsf{proj}(\mathcal{C}_i, X_t)$ for each $i \in [e]$ such that*

$$S = (F_t \setminus F_t^b) \cup \bigcup_{i=1}^{e} S'_i \cup \bigcup_{i=1}^{d} S_i.$$

Proof. Let S be a subset of F and let $\mathbf{S} = (S_1, \ldots, S_d)$ be a d-tuple of subsets of F such that $n_t(S, \mathbf{S}) > 0$. Let $\tau \in N_t(S, \mathbf{S})$. We can write $F(\tau)$ as

$$F(\tau) = (F \setminus F_t)(\tau) \cup (F_t \setminus F_t^b)(\tau) \cup F_t^b(\tau). \tag{11.17}$$

The projection $(F \setminus F_t)(\tau)$ can be decomposed further into

$$(F \setminus F_t)(\tau) = (F \setminus F_t)(\tau|_{X_t^b}) \cup (F \setminus F_t)(\tau|_{X_t \setminus X_t^b}).$$

Because $\mathcal{T}$ is a tree decomposition of $I^*(F)$, no variable from $X_t \setminus X_t^b$ occurs in $F \setminus F_t$, so this simplifies to

$$(F \setminus F_t)(\tau) = (F \setminus F_t)(\tau|_{X_t^b}).$$

Inserting into (11.17), we get

$$F(\tau) = (F \setminus F_t)(\tau|_{X_t^b}) \cup (F_t \setminus F_t^b)(\tau) \cup F_t^b(\tau).$$

By (C), τ must satisfy $(F_t \setminus F_t^b)$, so we obtain

$$F(\tau) = (F \setminus F_t)(\tau|_{X_t^b}) \cup (F_t \setminus F_t^b) \cup F_t^b(\tau). \tag{11.18}$$

We can write $F(\tau)$ redundantly as

$$F(\tau) = F(\tau) \cup F(\tau|_{X_t^b}).$$

Inserting the right hand side of (11.18) for $F(\tau)$, we get

$$\begin{aligned} F(\tau) &= (F \setminus F_t)(\tau|_{X_t^b}) \cup (F_t \setminus F_t^b) \cup F_t^b(\tau) \cup F(\tau|_{X_t^b}) \\ &= (F_t \setminus F_t^b) \cup F_t^b(\tau) \cup F(\tau|_{X_t^b}) \\ &= (F_t \setminus F_t^b) \cup \bigcup_{i=1}^{e} \mathcal{C}_i(\tau) \cup \bigcup_{i=1}^{d} F(\tau|_{\mathbf{x_i}}) \\ &= (F_t \setminus F_t^b) \cup \bigcup_{i=1}^{e} \mathcal{C}_i(\tau) \cup \bigcup_{i=1}^{d} S_i. \end{aligned}$$

Choosing $S_i' = \mathcal{C}_i(\tau)$ for $i \in [e]$, the lemma follows. □

Lemma 61. *Let F be a formula and let $\mathcal{T} = (T, \chi, r)$ be a tree decomposition of $I^*(F)$. Let $t \in V(T)$ such that $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \ldots, \mathbf{x_k})$, let S be a subset of F, and let $\mathbf{S} = (S_1, \ldots, S_k)$ be a k-tuple of subsets of F. If $n_t(S, \mathbf{S}) > 0$ then* MTW-ModelCount *creates an entry $M_t[S, \mathbf{S}]$ during its initialization phase given F and $\mathcal{T}$ as input.*

Proof. Immediate from Lemma 60 and the definition of MTW-ModelCount. □

Lemma 62. *Let F be a formula and let $\mathcal{T} = (T, \chi, r)$ be a tree decomposition of $I^*(F)$, and let $t \in V(T)$. If* MTW-ModelCount *is executed on input F and $\mathcal{T}$, then $M_t[S, \mathbf{S}] = n_t(S, \mathbf{S})$ for each subset S of F and each tuple $\mathbf{S}$ of subsets of F after the algorithm is done processing t.*

Proof. MTW-ModelCount starts processing a node t only if its children have already been processed. To prove the lemma, we assume that it holds for the children of t and show that it holds for t as well under this assumption. Keep in mind that $M_t[S, \mathbf{S}] = 0$ if the dictionary M_t has no entry for $(S, \mathbf{S})$, and that entries are set to 0 initially.

1. Let t be a leaf node with $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \ldots, \mathbf{x_k})$, let S be a subset of F, and let $\mathbf{S} = (S_1, \ldots, S_k)$ be a k-tuple of subsets of F. If $S_i \in \mathsf{proj}(F, \mathbf{x_i})$ for each $i \in [k]$ and $S = \bigcup_{i=1}^{k} S_i$, then the entry $M_t[S, \mathbf{S}]$ is created during the initialization phase, and

$$\begin{aligned} M_t[S, \mathbf{S}] &= \prod_{i=1}^{k} f_{\mathbf{x}}(S_i) && \text{(by definition of the algorithm)} \\ &= n_t(S, \mathbf{S}) && \text{(by Lemma 58).} \end{aligned}$$

   Otherwise, if $S_i \notin \mathsf{proj}(F, \mathbf{x_i})$ for an $i \in [k]$ then $n_t(S, \mathbf{S}) = 0$ by Lemma 48, and if $S \neq \bigcup_{i=1}^{k} S_i$ then $n_t(S, \mathbf{S}) = 0$ by Lemma 58. Moreover, the entry $M_t[S, \mathbf{S}]$ either does not exist and in any case is not updated, so $M_t[S, \mathbf{S}] = 0$.

2. Let t be a variable introduce node with child t', and let $\chi_{\mathbf{v}}(t') = (\mathbf{x_1}, \ldots, \mathbf{x_k})$ as well as $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \ldots, \mathbf{x_k}, \mathbf{x})$. Let S and P be subsets of F, and let $\mathbf{S} = (S_1, \ldots, S_k)$ be a k-tuple of subsets of F. By assumption, the lemma holds for t', so Lemma 52 yields

$$n_t(S, \mathbf{S} : P) = f_{\mathbf{x}}(P) \sum_{S' : S' \cup P = S} M_{t'}[S', \mathbf{S}]. \tag{11.19}$$

We argue that the right-hand side of (11.19) is equal to $M_t[S, \mathbf{S} : P]$ after the algorithm is done processing t. If $P \in \mathsf{proj}(F, \mathbf{x})$ and there exists an entry $M_{t'}[S', \mathbf{S}] > 0$ such that $S = S' \cup P$, then $n_t(S, \mathbf{S} : P) > 0$ by (11.19), so the entry $M_t[S, \mathbf{S} : P]$ must have been created during the initialization phase by Lemma 61. For a given set S', there is a unique set S such that $S = S' \cup P$, so the value of $M_t[S, \mathbf{S} : P]$ after processing matches the right-hand side of (11.19). If $P \notin \mathsf{proj}(F, \mathbf{x})$ or there is no entry $M_{t'}[S', \mathbf{S}] > 0$ such that $S = S' \cup P$ then $n_t(S, \mathbf{S} : P) = 0$ by (11.19). The algorithm does not update $M_t[S, \mathbf{S}]$, so $M_t[S, \mathbf{S}] = 0$ holds after processing.

3. Let t be a variable forget node with child t' with $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \ldots, \mathbf{x_k})$ and $\chi_{\mathbf{v}}(t') = (\mathbf{x_1}, \ldots, \mathbf{x_k}, \mathbf{x})$. Let S be a subset of F and let $\mathbf{S} = (S_1, \ldots, S_k)$ is a k-tuple of subsets of F. By Lemma 53 and the assumption that the lemma holds for t',

$$n_t(S, \mathbf{S}) = \sum_{P \subseteq F} M_{t'}[S, \mathbf{S} : P]. \tag{11.20}$$

Accordingly, if there is an entry $M_{t'}[S, \mathbf{S} : P] > 0$ then $n_t(S, \mathbf{S}) > 0$ and an entry for $(S, \mathbf{S})$ in M_t must have been created during initialization by Lemma 61. After processing t, the value of this entry matches the right-hand side of (11.20). If there is no non-zero entry $M_{t'}[S, \mathbf{S} : P]$ then the entry $M_t[S, \mathbf{S}]$ is not updated by the algorithm if it exists at all, so $M_t[S, \mathbf{S}] = 0$. By (11.20) we have $n_t(S, \mathbf{S}) = 0$, so $M_t[S, \mathbf{S}] = n_t(S, \mathbf{S})$ after processing.

4. Let t be a clause introduce node with child t' and $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \ldots, \mathbf{x_k})$. Let S be a subset of F and let $\mathbf{S} = (S_1, \ldots, S_k)$ be a k-tuple of subsets of F. By assumption, $M_{t'}[S, \mathbf{S}] = n_{t'}(S, \mathbf{S})$, and $n_t(S, \mathbf{S}) = n_{t'}(S, \mathbf{S})$ by Lemma 54. Thus if $M_{t'}[S, \mathbf{S}] > 0$ then $n_t(S, \mathbf{S}) > 0$, so the entry $M_t[S, \mathbf{S}]$ must have been created during initialization by Lemma 61, and it is updated correctly as $M_t[S, \mathbf{S}] = M_{t'}[S, \mathbf{S}]$. Otherwise, either there is no entry for $(S, \mathbf{S})$ in M_t or it satisfies $M_t[S, \mathbf{S}] = 0 = M_{t'}[S, \mathbf{S}]$ after updating.

5. Let t be a clause forget node with child t', and let $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \ldots, \mathbf{x_k})$ as well as $\chi_c(t) = \chi_c(t') \setminus \{\mathcal{C}\}$. Let S be a subset of F and let $\mathbf{S} = (S_1, \ldots, S_k)$ be a k-tuple of subsets of F. By assumption, $M_{t'}[S, \mathbf{S}] = n_{t'}(S, \mathbf{S})$. By Lemma 55, $n_t(S, \mathbf{S}) = n_{t'}(S, \mathbf{S})$ if $\mathcal{C} \subseteq S$, and $n_t(S, \mathbf{S}) = 0$ otherwise. If $M_{t'}[S, \mathbf{S}] > 0$ and $\mathcal{C} \subseteq S$, then $n_t[S, \mathbf{S}] > 0$. Thus the entry $M_t[S, \mathbf{S}]$ must have been created during initialization by Lemma 61, and it satisfies $M_t[S, \mathbf{S}] = M_{t'}[S, \mathbf{S}]$ after updating. Otherwise, either M_t has no entry for $(S, \mathbf{S})$ or it satisfies $M_t[S, \mathbf{S}] = 0 = M_{t'}[S, \mathbf{S}]$ after updating.

6. Let t be a join node with children t' and t''. Let $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \ldots, \mathbf{x_k})$, let S be a subset of F, and let $\mathbf{S} = (S_1, \ldots, S_k)$ be a k-tuple of subsets of F. If $F_t \setminus F_t^b \not\subseteq S$ or there exists an $i \in [k]$ such that $f_{\mathbf{x_i}}(S_i) = 0$, then $n_t(S, \mathbf{S}) = 0$ by Lemma 48. By definition of $f_{\mathbf{x_i}}$ we have $S_i \notin \mathsf{proj}(F, \mathbf{x_i})$, so MTW-ModelCount does not create an entry for $(S, \mathbf{S})$ in M_t during initialization and $M_t[S, \mathbf{S}] = 0$.

   Otherwise, $F_t \setminus F_t^b \subseteq S$ and $f_{\mathbf{x_i}}(S_i) > 0$ for each $i \in [k]$, so by Lemma 57 and the assumption that the lemma holds for t' and t'',

$$n_t(S, \mathbf{S}) = \frac{1}{\prod_{i=1}^{k} f_{\mathbf{x_i}}(S_i)} \sum_{\substack{S,S'': \\ S=S' \cup S''}} M_{t'}[S', \mathbf{S}]\, M_{t''}[S'', \mathbf{S}]. \tag{11.21}$$

   Accordingly, if there are non-zero entries $M_{t'}[S', \mathbf{S}]$ and $M_{t''}[S'', \mathbf{S}]$ with $S = S' \cup S''$, then $n_t(S, \mathbf{S}) > 0$ and the entry $M_t[S, \mathbf{S}]$ must have been created during initialization by Lemma 61. Given subsets S', S'' of F, their union $S = S' \cup S''$ is unique, so the value of $M_t[S, \mathbf{S}]$ after processing matches the right-hand side of (11.21). Otherwise, if there are no non-zero entries $M_{t'}[S', \mathbf{S}]$ and $M_{t''}[S'', \mathbf{S}]$ such that $S = S' \cup S''$, then $n_t(S, \mathbf{S}) = 0$ by (11.21) and either there is no entry for $(S, \mathbf{S})$ in M_t, or the entry was created but not updated, and $M_t[S, \mathbf{S}] = 0$ in either case.

□

Having established this lemma, correctness follows easily.

Lemma 63. *Given a formula F and a tree decomposition $\mathcal{T} = (T, \chi, r)$ of $I^*(F)$ as input, the algorithm* MTW-ModelCount *computes the number $\#F$ of satisfying assignments of F.*

Proof. Let F be a formula and let $\mathcal{T} = (T, \chi, r)$ be a tree decomposition of $I^*(F)$. It is easily verified that the algorithm terminates. A more detailed runtime analysis is given in Lemma 65 below. Let $\chi_{\mathbf{v}}(r) = (\mathbf{x_1}, \ldots, \mathbf{x_k})$. By Lemma 62 we have $M_r[F, \mathbf{S}] = n_r(F, \mathbf{S})$ for each k-tuple $\mathbf{S}$ of subsets of F once the algorithm has processed r. Using Lemma 59, we conclude that value returned by MTW-ModelCount is $\#F$. □

11.2.4 Runtime Analysis

We now analyze the runtime of MTW-ModelCount. The number of satisfying assignments of a formula can be exponential in the number of its variables, and we cannot automatically assume that arithmetic operations on numbers of this size can be performed in constant time. To account for this, we introduce constants $\delta(n)$ denoting the time required for operations on $(n+1)$-bit integers (the number of models of an input formula with n variables can be anywhere between 0 and 2^n, so we need an additional bit), and give runtime bounds including these constants. More specifically, we assume that $\delta(n)$ is an upper bound on the time required for addition, subtraction, division, and multiplication of two $(n+1)$-bit integers. Using standard algorithms we obtain $\delta(n) = O(n^2)$, which is sufficient for our analysis.

Lemma 64. *Let F be a formula with n variables and let $\mathcal{T} = (T, \chi, r)$ be a tree decomposition of $I^*(F)$. Each arithmetic operation performed by* MTW-ModelCount *on input $(F, \mathcal{T})$ requires time $\delta(n)$.*

Proof. We argue that every operand of an arithmetic operation is an integer in the interval $[0, 2^n]$. By Lemma 62 and definition of $n_t(S, \mathbf{S})$, the value of $M_t[S, \mathbf{S}]$ is an integer in the interval $[0, 2^n]$ after a node $t \in V(T)$ has been processed. For each variabe module $\mathbf{x}$ of F and subset S of F, $f_{\mathbf{x}}(S)$ is in the interval $[0, 2^{|\mathbf{x}|}]$. It is an easy consequence that operands of arithmetic operations performed during initialization, as well as during processing of leaf, introduce, and forget nodes, are in the interval $[0, 2^n]$. The formula F has at most 2^n satisfying assignments, so the output can be computed using integers in the interval $[0, 2^n]$ as well. Let t be a join node with $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \ldots, \mathbf{x_k})$. Let S, S', and S'' be subsets of F such that $S = S' \cup S''$, and let $\mathbf{S} = (S_1, \ldots, S_k)$ be a k-tuple of subsets of F. Since n is the sum of cardinalities of variable modules of F and $f_{\mathbf{x}}(S)$ is in the interval $[0, 2^{|\mathbf{x}|}]$ for each variable module $\mathbf{x}$ of F, the product

$$p = \prod_{i=1}^{k} f_{\mathbf{x_i}}(S_i)$$

is in the interval $[0, 2^n]$. By Lemma 62 and Lemma 50,

$$q = \frac{M_{t'}[S', \mathbf{S}]}{p} = \frac{M_{t'}[S', \mathbf{S}]}{\prod_{i=1}^{k} f_{\mathbf{x_i}}(S_i)} = |A^-_{t'}(S', \mathbf{S})|$$

is an integer in the interval $[0, 2^n]$. We conclude that operands of assignments and arithmetic operations performed by MTW-ModelCount are integers in the interval $[0, 2^n]$, which can be represented using $n + 1$ bits. □

Lemma 65. *Let F be formula with n variables, m clauses, and length l. Let $\mathcal{T} = (T, \chi, r)$ be a tree decomposition of $I^*(F)$ of width k. On input $(F, \mathcal{T})$* MTW-ModelCount *runs in time*

$$O(|T|(m+1)^{2k+2} l^3 \log(l)).$$

Proof. The dictionaries for M_t, $\mathsf{proj}(F, \mathbf{x})$, and $f_{\mathbf{x}}$ can be implemented (for instance, using red-black trees [28]) so that each operation takes time $O(K \log(S) + V)$, where K is the maximum length of a key, V is the maximum length of the binary representation of a value, and S is the maximum number of entries. By Lemma 64, $V \leq n + 1$. Assuming a fixed order on the clauses of F, we can represent a subset S of F by its characteristic function using m bits. The number of entries in the dictionaries for $\mathsf{proj}(F, \mathbf{x})$ and $f_{\mathbf{x}}$ is at most $m+1$. Thus operations on a dictionary for $\mathsf{proj}(F, \mathbf{x})$ take time $O(m \log(m)) = O(l \log(l))$, and operations on a dictionary for $f_{\mathbf{x}}$ take time $O(m \log(m) + n) = O(l \log(l))$. A pair $(S, \mathbf{S})$ consisting of a subset S of F and a d-tuple $\mathbf{S}$ of subsets of F can be represented by the sequence of characteristic functions using $(d + 1)m$ bits. Verify that for each $t \in V(T)$, the number of entries $M_t[S, \mathbf{S}]$ created by the algorithm during initialization

is at most $(m+1)^{k+1}$. To see this, let $t \in V(T)$ such that $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \dots \mathbf{x_r})$ and $\chi_c(t) = \{\mathcal{C}_1, \dots, \mathcal{C}_s\}$. Since $|\mathsf{proj}(F, \mathbf{x_i})| \leq m+1$ for each $i \in [r]$ and $|\mathsf{proj}(\mathcal{C}_i, X_t)| \leq m+1$ for each $j \in [s]$, MTW-ModelCount creates at most $(m+1)^{r+s} \leq (m+1)^{k+1}$ distinct entries $M_t[S, \mathbf{S}]$. Since $\mathbf{S}$ can be a $(k+1)$-tuple in the worst case, we conclude that each operation on a dictionary M_t takes time $O((k+2)m(k+1)log(m)+n) = O(l^3 log(l))$. We now give a bound on the time required for initialization.

- Let $\mathbf{x}$ be a variable module of F. The set $\mathsf{proj}(F, \mathbf{x})$ can be computed in time $O(l^2 + nl) = O(l^2)$ by Lemma 46. Filling the dictionary for $\mathsf{proj}(F, \mathbf{x})$ takes time $O((m+1)l\log(l)) = O(l^2 \log(l))$.

  For each $S \in \mathsf{proj}(F, \mathbf{x}) \setminus \{F_{\mathbf{x}}\}$, we have $f_{\mathbf{x}}(S) = 1$. If $F_{\mathbf{x}} \in \mathsf{proj}(F, \mathbf{x})$, it follows from Lemma 45 that $|F_{\mathbf{x}}/{\sim_{\mathbf{x}}}| = |\mathsf{proj}(F, \mathbf{x})| - 1$, and this number can be computed in time $O(m\log(m))$. As $f_{\mathbf{x}}(F_{\mathbf{x}}) = 2^{|\mathbf{x}|} - |F_{\mathbf{x}}/{\sim_{\mathbf{x}}}|$, from there we can compute $f_{\mathbf{x}}(F_{\mathbf{x}})$ in time $\delta(n)$. Overall, populating the dictionary for $f_{\mathbf{x}}$ takes time $O(l^2 log(l) + m\log(m) + \delta(n)) = O(l^2 log(l))$.

- By Lemma 46, the set $\mathsf{proj}(\mathcal{C}, X_t)$ can be computed in time $O(l^2)$ for each node $t \in V(T)$ and clause module $\mathcal{C} \in \chi_c(t)$. Since $|\chi_c(t)| \leq k+1$, for a given $t \in V(T)$ we can compute all sets $\mathsf{proj}(\mathcal{C}, X_t)$ for $\mathcal{C} \in \chi_c(t)$ in time $O(kl^2)$. Given the characteristic function of s subsets of F, we can compute the characteristic function of their union in time $O(sm)$. Thus for each $t \in V(T)$, filling the dictionary M_t takes time

$$O(kl^2 + (m+1)^{k+1}((k+1)m + l^3\log(l))) = O((m+1)^{k+1} l^3 \log(l)).$$

Overall, initialization takes time

$$O(nl^2\log(n) + |T|(m+1)^{k+1} l^3 \log(l)) = O(|T|(m+1)^{k+1} l^3 \log(l)).$$

We proceed to give bounds on the time required to process an individual node:

1. Let t be a leaf node with $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \dots, \mathbf{x_d})$. There are at most $(m+1)^{k+1}$ tuples $\mathbf{S} = (S_1, \dots, S_d)$ such that $S_i \in \mathsf{proj}(F, \mathbf{x_i})$ for each $i \in [d]$. For each such tuple $\mathbf{S}$, we can compute the union $S = \bigcup_{i=1}^{d} S_i$ in time $O(lk)$. Using the dictionaries for $f_{\mathbf{x_i}}$, the product of the $f_{\mathbf{x_i}}(S_i)$ for $i \in [d]$ can be computed in time $O(kl\log(l) + \log(k)\delta(n))$. Updating $M_t[S, \mathbf{S}]$ takes time $O(l^3\log(l))$. Since $k, n \leq l$, we conclude that the overall processing time for a leaf node is

$$O((m+1)^{k+1}(l^2\log(l) + \log(k)\delta(n) + l^3\log(l)) = O((m+1)^{k+1} l^3 \log(l)).$$

2. Let t be a variable introduce node with child t', and let $\chi_v(t) = \chi_v(t') \cup \{\mathbf{x}\}$. There are at most $(m+1)^{k+1}$ pairs $(P, (S, \mathbf{S}))$ consisting of a projection $P \in \mathsf{proj}(F, \mathbf{x})$ and an entry $M_{t'}[S', \mathbf{S}]$. The union $S = S' \cup P$ can be computed in time $O(l)$. The time required for retrieving $f_{\mathbf{x}}(P)$ and updating $M_t[S, \mathbf{S}]$ is in $O(l\log(l) + l^3\log(l) + \delta(n)) = O(l^3\log(l))$. Thus the overall processing time for t is in

$$O((m+1)^{k+1} l^3 \log(l)).$$

3. Let t be a variable forget node with child t', and let $\chi_v(t) = \chi_v(t') \setminus \{\mathbf{x}\}$. There are at most $(m+1)^{k+1}$ non-zero entries $M_{t'}[S, \mathbf{S} : P]$. Updating an entry $M_t[S, \mathbf{S}]$ takes time $O(l^3 \log(l) + \delta(n)) = O(l^3 \log(l))$. We conclude that processing t takes time

$$O((m+1)^{k+1} l^3 \log(l)).$$

4. To process a clause introduce node, the algorithm has to go through at most $(m+1)^{k+1}$ non-zero entries $M_{t'}[S, \mathbf{S}]$ and update the corresponding entry $M_t[S, \mathbf{S}]$, which can be done in time $O((m+1)^{k+1} l^3 \log(l))$.

5. Let t be a clause forget node with child t', and let $\chi_c(t) = \chi_c(t') \setminus \{\mathcal{C}\}$. There are at most $(m+1)^{k+1}$ non-zero entries $M_{t'}[S, \mathbf{S}]$. We can check whether $\mathcal{C} \subseteq S$ in time $O(l)$, and update the entry $M_t[S, \mathbf{S}]$ in time $O(l^3 \log(l))$. Overall, processing t takes time
$$O((m+1)^{k+1} l^3 \log(l)).$$

6. Let t be a join node with children t' and t'', and let $\chi_{\mathbf{v}}(t) = (\mathbf{x_1}, \ldots, \mathbf{x_r})$. There are at most $(m+1)^{2r} \leq (m+1)^{2k+2}$ pairs of entries $M_{t'}[S', \mathbf{S}]$ and $M_{t''}[S'', \mathbf{S}']$. Determining whether $\mathbf{S} = \mathbf{S}'$ takes time $O(kl)$. The union $S = S' \cup S''$ can be computed in time $O(l)$. We can compute the product p in time $O(kl \log(l) + \log(k)\delta(n))$ and the quotient q in time $\delta(n)$. Updating $M_t[S, \mathbf{S}]$ takes time $O(l^3 \log(l) + \delta(n)) = O(l^3 \log(l))$. Overall, the time spent on t is in
$$O((m+1)^{2k+2} l^3 \log(l). \tag{11.22}$$

After all nodes have been processed, the algorithm computes the sum of at most $(m+1)^{k+1}$ entries $M_r[F, \mathbf{S}]$ in time $O((m+1)^{k+1}\delta(n))$. With (11.22) as an upper bound for the processing time of individual nodes, we get an overall runtime bound of

$$O(|T|(m+1)^{2k+2} l^3 \log(l)).$$

□

Theorem 13. *#SAT can be solved in time $O(l^{2k+7})$ on CNF formulas that have modular incidence treewidth at most k and length l.*

Proof. Let F be a formula with n variables, m clauses, length l, and modular incidence treewidth at most k. We first construct $I(F)$ and then contract all its modules in order to obtain $I^*(F)$. This can be done in $O(l^2)$ time. A tree decomposition of $I^*(F)$ of width at most k can be obtained in time $O(n^{k+2}) = O(l^{k+2})$ [2]. This tree decomposition can be converted in time $O(l^2)$ into a nice tree decomposition $\mathcal{T} = (T, \chi, r)$ of width at most k with at most $4(n+m)$ nodes [80]. Using the algorithm MTW-ModelCount with input F and $\mathcal{T}$, we can then compute $\#F$ in time

$$\begin{aligned} O(|T|(m+1)^{2k+2} l^3 \log(l)) &= O((n+m)(m+1)^{2k+2} l^3 \log(l)) \\ &= O((m+1)^{2k+2} l^4 \log(l)) \end{aligned}$$

by Lemma 63 and Lemma 65. If $l = m$ then each clause of F contains exactly one literal and $\#F \in \{0, 1\}$ can easily be computed in linear time. Otherwise, $m + 1 \leq l$ and we get an overall runtime bound of

$$O(l^2) + O(l^{k+2}) + O(l^{2k+2} l^4 \log(l)) = O(l^{2k+7}).$$

□

11.3 W[1]-Hardness of SAT

To prove W[1]-hardness of SAT parameterized by modular incidence treewidth, we can reuse a reduction for proving that SAT is W[1]-hard when parameterized by the β-hypertree width [88].

Theorem 14. *SAT is* W[1]*-hard when parameterized by the modular incidence treewidth of the input formula.*

Proof. A *clique* in a graph is a subset of vertices that are mutually adjacent. A k-partite graph is *balanced* if its k partition classes are of the same size. A *partitioned clique* of a balanced k-partite graph $G = (V_1, \ldots, V_k, E)$ is a clique K with $|K \cap V_i| = 1$ for $i = 1 \ldots, k$. We devise a parameterized reduction from the following problem, which is W[1]-complete [94].

> Partitioned Clique
>
> *Instance:* A balanced k-partite graph $G = (V_1, \ldots, V_k, E)$.
>
> *Parameter:* The integer k.
>
> *Question:* Does G have a partitioned clique?

Before we describe the reduction we introduce some auxiliary concepts. For any three variables z, x_1, x_2, let $F(z, x_1, x_2)$ denote the formula consisting of the clauses

$$\{z, x_1, \overline{x_2}\}, \{z, \overline{x_1}, x_2\}, \{z, \overline{x_1}, \overline{x_2}\}, \{\overline{z}, x_1, x_2\}, \{\overline{z}, \overline{x_1}, \overline{x_2}\}.$$

This formula has exactly three satisfying assignments, corresponding to the vectors 000, 101, and 110. Hence each satisfying assignment sets at most one out of x_1 and x_2 to true, and if one of them is set to true, then z is set to true as well ("$z = x_1 + x_2$"). Taking several instances of this formula we can build a "selection gadget." Let $x_1, \ldots, x_m$ and $z_1, \ldots, z_{m-1}$ be variables. We define $F^{=1}(x_1, \ldots, x_m; z_1, \ldots, z_{m-1})$ as the union of $F(z_1, x_1, x_2)$, $\bigcup_{i=2}^{m-1} F(z_i, z_{i-1}, x_{i+1})$, and $\{\{z_{m-1}\}\}$. Now each satisfying assignment of this formula sets exactly one variable out of $\{x_1, \ldots, x_m\}$ to true, and, conversely, for each $1 \leq i \leq m$ there exists a satisfying assignment that sets exactly x_i to true and all other variables from $\{x_1, \ldots, x_m\}$ to false.

Now we describe the reduction. Let $G = (V_1, \ldots, V_k)$ be a balanced k-partite graph for $k \geq 2$. We write $V_i = \{v_1^i, \ldots, v_n^i\}$. We construct a formula F. As the variables of F

we take the vertices of G plus new variables z^i_j for $1 \leq i \leq k$ and $1 \leq j \leq n-1$. We put $F = \bigcup_{i=0}^{k} F_i$ where the formulas F_i are defined as follows: F_0 contains for any $u \in V_i$ and $v \in V_j$ $(i \neq j)$ with $uv \notin E$ the clause $C_{u,v} = \{\,\overline{u}, \overline{v}\,\} \cup \{\,w : w \in (V_i \cup V_j) \setminus \{\,u, v\,\}\,\}$; for $i > 0$ we define $F_i = F^{=1}(v^i_1, \ldots, v^i_n; z^i_1, \ldots, z^i_{n-1})$. To prove the theorem it suffices to show the following two claims.

Claim 1. The modular incidence treewidth of F is at most $\binom{k}{2} + 1$.

For $1 \leq i \leq k$, let $D^i_1, \ldots, D^i_n$ be the vertices in $I(F)$ with $N(D^i_1) = \{z^i_1, v^i_1, v^i_2\}$, $N(D^i_j) = \{z^i_j, z^i_{j-1}, v^i_{j+1}\}$ for $2 \leq j \leq n-1$ and $N(D^i_n) = \{z^i_{n-1}\}$. In $I(F)$, the set of vertices $C_{u,v}$ can be partitioned into modules $\mathcal{C}_1, \ldots, \mathcal{C}_m$, where $m \leq \binom{k}{2}$. By deleting these modules, we obtain a graph $I'(F)$ that consists of k connected components corresponding to the subgraphs of $I(F)$ induced by $\{v^i_1, \ldots, v^i_n, z^i_1, \ldots, z^i_{n-1}, D^i_1, \ldots, D^i_n\}$ for $1 \leq i \leq k$. Note that these components are trees, so the treewidth of $I'(F)$ is 1. Thus the graph obtained from $I'(F)$ by contracting modules has treewidth 1. We can turn the corresponding tree decomposition into a tree decomposition of $I^*(F)$ that has width $m + 1 \leq \binom{k}{2} + 1$ by simply adding the set of m clause modules $\{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ to each bag.

Claim 2. G has a partitioned clique if and only if F is satisfiable.

To prove Claim 2, we first suppose that G has a partitioned clique K. We define a partial truth assignment $\tau : V \to \{0, 1\}$ by setting $\tau(v) = 1$ for $v \in K$, and $\tau(v) = 0$ for $v \notin K$. This partial assignment satisfies F_0, and it is easy to extend τ to a satisfying truth assignment of F. Conversely, suppose that F has a satisfying truth assignment τ. Because of the formulas F_i, $1 \leq i \leq k$, τ sets exactly one variable $v^i_{j_i} \in V_i$ to true. Let $K = \{v^1_{j_1}, \ldots, v^k_{j_k}\}$. The clauses in F_0 ensure that $v^i_{j_i}$ and $v^{i'}_{j_{i'}}$ are adjacent in G for each pair $1 \leq i < i' \leq k$, hence K is a partitioned clique of G. This proves Claim 2. □

Summary

Modular incidence treewidth combines treewidth and module contraction, a powerful preprocessing technique used in combinatorial optimization. In this chapter, we proved that there is an algorithm for #SAT that runs in time $l^{O(k)}$ on formulas of length l and modular incidence treewidth k. This proves that in particular, #SAT is polynomial-time tractable for classes of formulas of bounded modular incidence treewidth. We gave strong evidence that this runtime bound cannot be improved to a bound of the shape $f(k)\, l^c$, where c is a constant and f is a computable function independent of the input, by showing that already SAT is $W[1]$-hard when parameterized by the modular incidence treewidth.

CHAPTER 12

Symmetric Clique-Width

In this chapter, we prove an upper bound on the complexity of #SAT parameterized by symmetric incidence clique-width, that is, by the symmetric clique-width [29] of the incidence graph. Specifically, we prove the following result (Theorem 16):

> There is a function $f : \mathbb{N} \to \mathbb{N}$ such that #SAT can be solved in time $f(k)\, l^3 + l^{O(2^k)}$ on CNF formulas of symmetric incidence clique-width k and length l.

For classes of formulas of bounded symmetric incidence clique-width, we obtain a polynomial-time model counting algorithm. Since symmetric incidence clique-width strictly dominates modular incidence treewidth, this result entails the result on the polynomial-time tractability of #SAT for classes of formulas of bounded modular incidence treewidth from the previous chapter. Symmetric clique-width is a parameter that is closely related to clique-width, rank-width, and Boolean-width: a class of graphs has bounded symmetric clique-width *iff* it has bounded clique-width *iff* it has bounded rank-width *iff* it has bounded Boolean-width. Our result therefore carries over to incidence clique-width, rank-width, and Boolean-width.

The above result is obtained by means of dynamic programming on a decomposition tree. As in the algorithm presented in the previous chapter, we represent truth assignments by projections. However, we cannot afford to keep a record for each projection $F(\tau)$ of a partial assignments τ onto the input formula F, as the number of distinct projections can be exponential in the length of F. To deal with this, we pair information on partial solutions with an "expectation from the outside" [22, 46, 47]. The underlying idea is that the information that has to recorded for a partial solution can be reduced if one includes an "expectation" about what this partial solution will be combined with to form a complete solution. In our case, we do not record the projection $F_p(\tau)$ of a partial assignment τ onto the subformula F_p already processed – instead, we keep track of whether a partial assignment τ satisfies the subformula F_p when combined with an assignment σ to the remaining variables with a particular projection $F_p(\sigma)$. We show

that the number of projections of outside variables onto a subformula with m clauses is bounded by $(m+1)^k$, where k is the symmetric incidence clique-width, so that the number of cases to consider (and records to keep) is polynomial for fixed k.

Owing to the following result (cf. Theorem 14), we cannot expect to get k out of the exponent of l in the runtime bound of our algorithm.[1]

Theorem 15 ([88])**.** *SAT, parameterized by the symmetric clique-width of the incidence graph of the input formula, is W[1]-hard.*

Given a formula F and a decomposition tree $\mathcal{T}$ of $I(F)$, our dynamic programming algorithm runs in time $l^{O(w)}$, where w is the index of $\mathcal{T}$. We get $O(2^k)$ in the exponent of l in the bound of Theorem 16 because we have to approximate the symmetric clique-width of $I(F)$ through its rank-width. Efficient algorithms for computing decomposition trees of (approximately) optimal index (matching the graph's symmetric clique-width) would allow us to obtain a better bound.

Below, we present an algorithm for counting the models of an input formula F via dynamic programming on a decomposition tree of its incidence graph $I(F)$. To simplify the statements of intermediate results, we fix a formula F of length l with $|F| = m$ clauses and a decomposition tree (T, δ) of $I(F)$ with $\mathit{index}(T, \delta) = k$. For a node $z \in V(T)$, let T_z denote the maximal subtree of T rooted at z. We write var_z for the set of variables $\mathrm{var}(F) \cap \delta(L(T_z))$ and F_z for the set of clauses $F \cap \delta(L(T_z))$. Moreover, we write $\overline{F_z} = F \setminus F_z$ and $\overline{\mathrm{var}_z} = \mathrm{var}(F) \setminus \mathrm{var}_z$.

The information maintained for each node $z \in V(T)$ of the decomposition tree is a collection of shapes [47], along with the number of truth assignments $\sigma : \mathrm{var}_z \to \{0,1\}$ in each shape.

Definition 53 (Shape)**.** Let $z \in V(T)$, let $\mathit{out}_z \subseteq \overline{F_z}$, and let $\mathit{in}_z \subseteq F_z$. We call the pair $(\mathit{out}_z, \mathit{in}_z)$ a *shape (for z)*, and say an assignment $\tau \in 2^{\mathrm{var}_z}$ is *of shape* $(\mathit{out}_z, \mathit{in}_z)$ if it satisfies the following conditions.

(i) $\overline{F_z}(\tau) = \mathit{out}_z$.

(ii) For each clause $C \in F_z$, the assignment τ satisfies C or $C \in \mathit{in}_z$.

If $\mathit{out}_z \in \mathsf{proj}(\overline{F_z}, \mathrm{var}_z)$ and $\mathit{in}_z \in \mathsf{proj}(F_z, \overline{\mathrm{var}_z})$ then the shape $(\mathit{out}_z, \mathit{in}_z)$ is *proper*. We denote the set of shapes for $z \in V(T)$ by $\mathsf{shapes}(z)$ and write $N_z(s)$ to denote the set of assignments in 2^{var_z} of shape $s \in \mathsf{shapes}(z)$. Moreover, we let $n_z(s) = |N_z(s)|$.

An assignment can have multiple shapes, so shapes do not partition assignments into equivalence classes. The model count of F corresponds to the number of assignments with a particular shape at the root node.

[1] To be precise, the hardness proof in [88] is stated in terms of clique-width. But since the clique-width of a graph is at most twice its symmetric clique-width [29], the result carries over to symmetric clique-width.

Lemma 66. *A truth assignment $\tau \in 2^{\mathrm{var}(F)}$ satisfies F if and only if it has shape $(\emptyset, \emptyset)$. Moreover, the shape $(\emptyset, \emptyset)$ is proper.*

Proof. Observe that $\mathrm{var}_r = \mathrm{var}(F)$, and let $\tau \in 2^{\mathrm{var}_r}$. Suppose τ satisfies F. Since $\overline{F_r}$ is empty, we immediately get $\overline{F_r}(\tau) = \emptyset$, so τ satisfies condition (i). Moreover τ satisfies every clause of $F = F_r$, so condition (ii) is satisfied as well. For the converse, suppose τ has shape $(\emptyset, \emptyset)$. It follows from condition (ii) that τ must satisfy $F_r = F$. To see that $(\emptyset, \emptyset)$ is proper note that $\overline{F_r}(\sigma) = \emptyset$ for any $\sigma \in 2^{\mathrm{var}_r}$, and that $2^{\overline{\mathrm{var}_r}}$ contains only the empty assignment $\epsilon : \emptyset \to \{0, 1\}$ with $F_r(\epsilon) = \emptyset$. □

Let $x, y, z \in V(T)$ such that x and y are the children of z, and let s_x, s_y, s_z be shapes for x, y, z, respectively. The assignments in $N_x(s_x)$ and $N_y(s_y)$ contribute to $N_z(s_z)$ if certain conditions are met. These are captured by the following definition.

Definition 54. Let $x, y, z \in V(T)$ such that x and y are the children of z. We say two shapes $(out_x, in_x) \in \mathsf{shapes}(x)$ and $(out_y, in_y) \in \mathsf{shapes}(y)$ *generate* the shape $(out_z, in_z) \in \mathsf{shapes}(z)$ whenever the following conditions are satisfied.

(1) $out_z = (out_x \cup out_y) \cap \overline{F_z}$

(2) $in_x = (in_z \cup out_y) \cap F_x$

(3) $in_y = (in_z \cup out_x) \cap F_y$

We write $\mathsf{generators}_z(s)$ for the set of pairs in $\mathsf{shapes}(x) \times \mathsf{shapes}(y)$ that generate $s \in \mathsf{shapes}(z)$.

Lemma 67. *Let $x, y, z \in V(T)$ such that x and y are the children of z, and let $\tau_x : \mathrm{var}_x \to \{0, 1\}$ be of shape $(out_x, in_x) \in \mathsf{shapes}(x)$ and $\tau_y : \mathrm{var}_y \to \{0, 1\}$ be of shape $(out_y, in_y) \in \mathsf{shapes}(y)$. If (out_x, in_x) and (out_y, in_y) generate the shape $(out_z, in_z) \in \mathsf{shapes}(z)$, then $\tau_x \cup \tau_y$ is of shape (out_z, in_z). Moreover, if (out_z, in_z) is proper then (out_x, in_x) and (out_y, in_y) are proper.*

Proof. Suppose (out_x, in_x) and (out_y, in_y) generate (out_z, in_z). Let $\tau = \tau_x \cup \tau_y$. We have

$$\overline{F_z}(\tau_z) = \overline{F_z}(\tau_x) \cup \overline{F_z}(\tau_y) = (out_x \cap \overline{F_z}) \cup (out_y \cap \overline{F_z}) = out_z,$$

so τ satisfies condition (i). To verify that condition (ii) is satisfied as well, let $C \in F_z = F_x \cup F_y$. Assume without loss of generality that $C \in F_x$. Suppose τ does not satisfy C. Then in particular τ_x does not satisfy C, so we must have $C \in in_x$ because τ_x is of shape (out_x, in_x). But τ_y does not satisfy C either, so $C \notin out_y$. Combining these statements, we get $C \in in_x \setminus out_y$. Because (out_x, in_x) and (out_y, in_y) generate (out_z, in_z) we have $in_x = (in_z \cup out_y) \cap F_x$ by condition (2), so $C \in in_z$.

The assignments τ_x and τ_y are of shapes (out_x, in_x) and (out_y, in_y), so $out_x \in \mathsf{proj}(\overline{F_x}, \mathrm{var}_x)$ and $out_y \in \mathsf{proj}(\overline{F_y}, \mathrm{var}_y)$ by condition (i). Suppose (out_z, in_z) is proper. Then there is an assignment $\rho : \overline{\mathrm{var}_z} \to \{0, 1\}$ such that $in_z = F_z(\rho)$. The shapes

(out_x, in_x) and (out_y, in_y) generate (out_z, in_z), so $in_x = (in_z \cup out_y) \cap F_x$, that is, $in_x = (F_z(\rho) \cup \overline{F_y}(\tau_y)) \cap F_x$. Equivalently, $in_x = (F_z(\rho) \cap F_x) \cup (\overline{F_y}(\tau_y) \cap F_x)$. Since $F_x \subseteq F_z$ and $F_x \subseteq \overline{F_y}$ this can be rewritten once more as $in_x = F_x(\rho) \cup F_x(\tau_y)$. The domains $\overline{\mathrm{var}_z}$ of ρ and var_y of τ_y are disjoint, so $F_x(\rho) \cup F_x(\tau_y) = F_x(\rho \cup \tau_y)$. Because $\overline{\mathrm{var}_z} \cup \mathrm{var}_y = \overline{\mathrm{var}_x}$ it follows that $in_x \in \mathsf{proj}(F_x, \overline{\mathrm{var}_x})$ and (out_x, in_x) is proper. A symmetric argument shows that (out_y, in_y) is proper. □

Corollary 4. *Let $x, y, z \in V(T)$ such that x and y are the children of z in T, and let $s \in \mathsf{shapes}(z)$ be proper. Suppose $s_x \in \mathsf{shapes}(x)$ and $s_y \in \mathsf{shapes}(y)$ generate s and both $N_x(s_x)$ and $N_y(s_y)$ are nonempty. Then s_x and s_y are proper.*

Lemma 68. *Let $x, y, z \in V(T)$ such that x and y are the children of z, and let $\tau : \mathrm{var}_z \to \{0, 1\}$ be a truth assignment of shape $(out_z, in_z) \in \mathsf{shapes}(z)$. Let $\tau_x = \tau|_{\mathrm{var}_x}$ and $\tau_y = \tau|_{\mathrm{var}_y}$. There are unique shapes $(out_x, in_x) \in \mathsf{shapes}(x)$ and $(out_y, in_y) \in \mathsf{shapes}(y)$ generating (out_z, in_z) such that τ_x has shape (out_x, in_x) and τ_y has shape (out_y, in_y).*

Proof. We define $out_x = \overline{F_x}(\tau_x)$, $out_y = \overline{F_y}(\tau_y)$ and

$$in_x = (in_z \cap F_x) \cup F_x(\tau_y), in_y = (in_z \cap F_y) \cup F_y(\tau_x).$$

We prove that (out_x, in_x) and (out_y, in_y) generate (out_z, in_z). Since τ has shape (out_z, in_z) by condition (i) we have $out_z = \overline{F_z}(\tau)$. We further have $\overline{F_z}(\tau) = \overline{F_z}(\tau_x) \cup \overline{F_z}(\tau_y)$ by choice of τ_x and τ_y. Because $\overline{F_z} \subseteq \overline{F_x}$ and $\overline{F_z} \subseteq \overline{F_y}$ we get

$$\overline{F_z}(\tau) = (\overline{F_x}(\tau_x) \cap \overline{F_z}) \cup (\overline{F_y}(\tau_y) \cap \overline{F_z})$$

and thus

$$\overline{F_z}(\tau) = (out_x \cup out_y) \cap \overline{F_z}.$$

That is, condition (1) is satisfied. From $F_x \subseteq \overline{F_y}$ and $F_y \subseteq \overline{F_x}$ it follows that $F_x(\tau_y) = \overline{F_y}(\tau_y) \cap F_x$ and $F_y(\tau_x) = \overline{F_x}(\tau_x) \cap F_y$. Thus $F_x(\tau_y) = out_y \cap F_x$ and $F_y(\tau_x) = out_x \cap F_y$ by construction of out_x and out_y. By inserting in the definitions of in_x and in_y we get $in_x = (in_z \cap F_x) \cup (out_y \cap F_x)$ and $in_y = (in_z \cap F_y) \cup (out_x \cap F_y)$, so conditions (2) and (3) are satisfied as well. We conclude that (out_x, in_x) and (out_y, in_y) generate (out_z, in_z).

We proceed to showing that τ_x is of shape (out_x, in_x). Condition (i) is satisfied by construction. To see that condition (ii) holds, pick any $C \in F_x$ not satisfied by τ_x. If τ_y satisfies C, then $C \in F_x(\tau_y) \subseteq in_x$. Otherwise, $\tau = \tau_x \cup \tau_y$ does not satisfy C. Since τ is of shape (out_z, in_z) this implies $C \in in_z$. Again we get $C \in in_x$ as $in_z \cap F_x \subseteq in_x$. The proof that τ_y has shape (out_y, in_y) is symmetric.

To show uniqueness, let $(out'_x, in'_x) \in \mathsf{shapes}(x)$ and $(out'_y, in'_y) \in \mathsf{shapes}(y)$ generate (out_z, in_z), and suppose τ_x has shape (out'_x, in'_x) and τ_y has shape (out'_y, in'_y). From condition (i) we immediately get $out'_x = \overline{F_x}(\tau_x) = out_x$ and $out'_y = \overline{F_y}(\tau_y) = out_y$. Since the pairs (out'_x, in'_x),(out'_y, in'_y) and (out_x, in_x), (out_y, in_y) both generate (out_z, in_z), it follows from condition (2) that $in'_x = in_x$ and $in'_y = in_y$. □

Lemma 69. *Let $x, y, z \in V(T)$ such that x and y are the children of z in T, and let $s \in \mathsf{shapes}(z)$. The following equality holds.*

$$n_z(s) = \sum_{(s_x,s_y)\in \mathsf{generators}_z(s)} n_x(s_x)\, n_y(s_y) \tag{12.1}$$

Proof. Let $M(s) = \bigcup_{(s_x,s_y)\in\mathsf{generators}_z(s)} N_x(s_x) \times N_y(s_y)$. We first show that the function $f : \tau \mapsto (\tau|_{\mathrm{var}_x}, \tau|_{\mathrm{var}_y})$ is a bijection from $N_z(s)$ to $M(s)$. By Lemma 68 for every $\tau \in N_z(s)$ there is a pair $(s_x, s_y) \in \mathsf{generators}_z(s)$ such that $\tau|_{\mathrm{var}_x} \in N_x(s_x)$ and $\tau|_{\mathrm{var}_y} \in N_y(s_y)$. So f is *into*. By Lemma 67, for every pair of assignments $\tau_x \in N_x(s_x), \tau_y \in N_y(s_y)$ with $(s_x, s_y) \in \mathsf{generators}_z(s)$ the assignment $\tau_x \cup \tau_y$ is in $N_z(s)$. Hence f is *surjective*. It is easy to see that f is *injective*, so f is indeed a bijection.

We prove that $|M(s)|$ is equivalent to the right hand side of (12.1). Since $|N_x(s_x) \times N_y(s_y)| = n_x(s_x)\, n_y(s_y)$ for every pair $(s_x, s_y) \in \mathsf{generators}_z(s)$, we only have to show that the sets $N_x(s_x) \times N_y(s_y)$ and $N_x(s'_x) \times N_y(s'_y)$ are disjoint for distinct pairs of shapes $(s_x, s_y), (s'_x, s'_y) \in \mathsf{generators}_z(s)$. Let $(s_x, s_y), (s'_x, s'_y) \in \mathsf{generators}_z(s)$ and suppose $(N_x(s_x)\times N_y(s_y))\cap(N_x(s'_x)\times N_y(s'_y))$ is nonempty. Let $(\tau_x, \tau_y) \in (N_x(s_x)\times N_y(s_y))\cap (N_x(s'_x) \times N_y(s'_y))$. The function f is a bijection, so $\tau_x \cup \tau_y \in N_z(s)$. By Lemma 68 there is at most one pair $(s''_x, s''_y) \in \mathsf{generators}_z(s)$ of shapes such that $\tau_x \in N_x(s''_x)$ and $\tau_y \in N_y(s''_y)$, so $(s_x, s_y) = (s''_x, s''_y) = (s'_x, s'_y)$. □

Corollary 5. *Let $x, y, z \in V(T)$ such that x and y are the children of z in T, and let $s \in \mathsf{shapes}(z)$ be proper. Let $P = \{\, (s_x, s_y) \in \mathsf{generators}_z(s) : s_x \text{ and } s_y \text{ are proper} \,\}$. The following equality holds.*

$$n_z(s) = \sum_{(s_x,s_y)\in P} n_x(s_x)\, n_y(s_y) \tag{12.2}$$

Proof. By Corollary 4 the product $n_x(s_x)n_y(s_y)$ is nonzero only if s_x and s_y are proper, for any pair $(s_x, s_y) \in \mathsf{generators}_z(s)$. In combination with (12.1) this implies (12.2). □

Corollary 5 in combination with Lemma 66 implies that, for each $z \in V(T)$, it is enough to compute the values $n_z(s)$ for *proper* shapes $s \in \mathsf{shapes}(z)$. To turn this insight into a polynomial time dynamic programming algorithm, we still have to show that the number of proper shapes in $\mathsf{shapes}(z)$ can be polynomially bounded, and that the set of such shapes can be computed in polynomial time. We will achieve this by specifying a subset of $\mathsf{shapes}(z)$ for each $z \in V(T)$ that contains all proper shapes and can be computed in polynomial time.

We define families neigh_z and $\overline{\mathsf{neigh}_z}$ of sets of variables for each node $z \in V(T)$, as follows.

$$\mathsf{neigh}_z = \{\, X \subseteq \mathrm{var}_z : \exists C \in \overline{F_z} \text{ such that } X = \mathrm{var}_z \cap \mathrm{var}(C) \,\}$$
$$\overline{\mathsf{neigh}_z} = \{\, X \subseteq \overline{\mathrm{var}_z} : \exists C \in F_z \text{ such that } X = \overline{\mathrm{var}_z} \cap \mathrm{var}(C) \,\}$$

The next lemma follows from the definition of a decomposition tree's index.

Lemma 70. *For every node $z \in V(T)$, $\max(|\mathsf{neigh}_z|, |\overline{\mathsf{neigh}_z}|) \leq k$.*

Let $z \in V(T)$. For each $X \in \mathsf{neigh}_z$, let $\overline{F_z^X}$ denote the set of clauses $C \in \overline{F_z}$ such that $X \subseteq \mathrm{var}(C)$. Symmetrically, for $Y \in \overline{\mathsf{neigh}_z}$ let F_z^X denote the set of clauses $C \in F_z$ such that $Y \subseteq \mathrm{var}(C)$. Let f be a function that maps every set $X \in \mathsf{neigh}_z$ to some projection $f(X) \in \mathsf{proj}(\overline{F_z^X}, X)$. We denote the set of such functions by $\mathsf{outfunctions}(z)$. Symmetrically, we let $\mathsf{infunctions}(z)$ denote the set of functions g that map every set $Y \in \overline{\mathsf{neigh}_z}$ to some projection $g(Y) \in \mathsf{proj}(F_z^Y, Y)$.

Lemma 71. *For every $z \in V(T)$, $|\mathsf{outfunctions}(z)| \leq (m+1)^k$ as well as $|\mathsf{infunctions}(z)| \leq (m+1)^k$.*

Proof. By Lemma 45 (proved in Chapter 11) the cardinality of $\mathsf{proj}(\overline{F_z^X}X,)$ is bounded by $m+1$ for every $X \in \mathsf{neigh}_z$. In combination with Lemma 70 this yields $|\mathsf{outfunctions}(z)| \leq (m+1)^k$. The proof of $|\mathsf{infunctions}(z)| \leq (m+1)^k$ is symmetric. □

Let $\mathsf{union}(f)$ denote $\bigcup_{X \in \mathsf{dom}(f)} f(X)$, where $\mathsf{dom}(f)$ is the domain of f. We define the set of *restricted* shapes for $z \in V(T)$ as follows.

$$\mathsf{shapes}^{\mathsf{res}}(z) = \{\, (\mathit{out}, \mathit{in}) \in \mathsf{shapes}(z) : \exists f \in \mathsf{outfunctions}(z) \text{ s.t. } \mathit{out} = \mathsf{union}(f) \text{ and } \exists g \in \mathsf{infunctions}(z) \text{ s.t. } \mathit{in} = \mathsf{union}(g) \,\}$$

Every pair $(f, g) \in \mathsf{outfunctions}(z) \times \mathsf{infunctions}(z)$ uniquely determines a shape in $\mathsf{shapes}^{\mathsf{res}}(z)$. Accordingly, Lemma 71 allows us to bound the cardinality of $\mathsf{shapes}^{\mathsf{res}}(z)$ as follows.

Corollary 6. *For any $z \in V(T)$, $|\mathsf{shapes}^{\mathsf{res}}(z)| \leq (m+1)^{2k}$.*

The following result states that every proper shape is a restricted shape.

Lemma 72. *Let $z \in V(T)$ and let $s \in \mathsf{shapes}(z)$ be proper. Then $s \in \mathsf{shapes}^{\mathsf{res}}(z)$.*

Proof. Let $s = (\mathit{out}, \mathit{in})$. We show that there are functions $f \in \mathsf{outfunctions}(z)$ and $g \in \mathsf{infunctions}(z)$ such that $\mathit{out} = \mathsf{union}(f)$ and $\mathit{in} = \mathsf{union}(g)$. Because s is proper we have $\mathit{out} \in \mathsf{proj}(\overline{F_z}, \mathrm{var}_z)$ and $\mathit{in} \in \mathsf{proj}(F_z, \overline{\mathrm{var}_z})$, so there are assignments $\sigma : \mathrm{var}_z \to \{0,1\}$ and $\tau : \overline{\mathrm{var}_z} \to \{0,1\}$ such that $\mathit{out} = \overline{F_z}(\sigma)$ and $\mathit{in} = F_z(\tau)$. We define f as follows. For each $X \in \mathsf{neigh}_z$, let $f(X) = \overline{F_z^X}(\sigma|_X)$. The assignment σ is defined on $X \subseteq \mathrm{var}_z$, so $\sigma|_X$ has domain X and $f(X) \in \mathsf{proj}(\overline{F_z^X}, X)$. That is, $f \in \mathsf{outfunctions}(z)$. Symmetrically, we let $g(Y) = F_z^Y(\tau|_Y)$ for each $Y \in \overline{\mathsf{neigh}_z}$. Since τ is defined on $Y \subseteq \overline{\mathrm{var}_z}$, the assignment $\tau|_Y$ has domain Y and $g(Y) \in \mathsf{proj}(F_z^Y, Y)$, so $g \in \mathsf{infunctions}(z)$.

Pick an arbitrary clause $C \in \overline{F_z}$ and let $X = \mathrm{var}(C) \cap \mathrm{var}_z$. We show that $C \in \mathit{out}$ if and only if $C \in \mathsf{union}(f)$. Suppose $C \in \mathit{out}$ and keep in mind that $\mathit{out} = \overline{F_z}(\sigma)$. The assignment σ has domain var_z, so $\sigma|_X$ satisfies C because σ does. That is, $C \in \overline{F_z}(\sigma|_X)$. By choice of X we have $C \in F_z^X$, so $C \in \overline{F_z}(\sigma|_X) \cap F_z^X$. Since $F_z^X \subseteq F_z$ we get

$$\overline{F_z}(\sigma|_X) \cap F_z^X = \overline{F_z^X}(\sigma|_X).$$

So $C \in \overline{F_z^X}(\sigma|_X) = f(X)$ and thus $C \in \mathsf{union}(f)$. For the converse, suppose $C \in \mathsf{union}(f)$. That is, $C \in f(Y)$, where $f(Y) = \overline{F_z^Y}(\sigma|_Y)$ for some $Y \in \mathsf{neigh}_z$. Then in particular $C \in \overline{F_z}(\sigma) = \mathit{out}$. We conclude that $\mathsf{union}(f) = \mathit{out}$. The proof of $\mathsf{union}(g) = \mathit{in}$ is symmetric. □

This shows that if we can determine the values $n_z(s)$ for every $z \in V(T)$ and $s \in \mathsf{shapes}^{\mathsf{res}}(z)$, we can determine the values $n_z(s')$ for every proper shape $s' \in \mathsf{shapes}(z)$. More specifically, as long as we can determine lower bounds for $n_z(s)$ for every $s \in \mathsf{shapes}^{\mathsf{res}}(z)$ and the exact values of $n_z(s)$ for proper s, we can compute the correct values for all proper shapes for every tree node.

Definition 55. For $z \in V(T)$, a *lower bounding function* (for z) associates with each $s \in \mathsf{shapes}^{\mathsf{res}}(z)$ a value $l_z(s)$ such that $l_z(s) \leq n_z(s)$ and $l_z(s) = n_z(s)$ if s is proper.

Let $x, y, z \in V(T)$ such that x and y are the children of z. For each shape $s \in \mathsf{shapes}(z)$, we write

$$\mathsf{restricedgen}_z(s) = \mathsf{generators}_z(s) \cap (\mathsf{shapes}^{\mathsf{res}}(x) \times \mathsf{shapes}^{\mathsf{res}}(y)).$$

Lemma 73. *Let $x, y, z \in V(T)$ such that x and y are the children of z. Let l_x and l_y be lower bounding functions for x and y. Let l_z be the function defined as follows. For each $s \in \mathsf{shapes}^{\mathsf{res}}(z)$, we let*

$$l_z(s) = \sum_{(s_x, s_y) \in \mathsf{restricedgen}_z(s)} l_x(s_x)\, l_y(s_y). \tag{12.3}$$

Then l_z is a lower bounding function for z.

Proof. The inequality $l_z(s) \leq n_z(s)$ follows from the containments $\mathsf{shapes}^{\mathsf{res}}(x) \subseteq \mathsf{shapes}(x)$ and $\mathsf{shapes}^{\mathsf{res}}(y) \subseteq \mathsf{shapes}(y)$, in combination with (12.1) and the fact that l_x and l_y are lower bounding functions for x and y. By Lemma 72 the set $\mathsf{restricedgen}_z(s)$ contains all pairs $(s_x, s_y) \in \mathsf{generators}_z(s)$ such that s_x and s_y are proper. It follows from Corollary 5 and $l_x(s_x) = n_x(s_x)$, $l_y(s_y) = n_y(s_y)$ for proper s_x, s_y that $l_z(s) \geq n_z(s)$ and thus $l_z(s) = n_z(s)$ for proper s. We conclude that l_z is a lower bounding function for z. □

Lemma 74. *There is a polynomial p (independent of F) such that for any $z \in V(T)$, the set $\mathsf{shapes}^{\mathsf{res}}(z)$ can be computed in time $m^{2k}p(l)$.*

Proof. To compute $\mathsf{shapes}^{\mathsf{res}}(z)$, we compute all pairs $(\mathsf{union}(f), \mathsf{union}(g))$ for $(f, g) \in \mathsf{outfunctions}(z) \times \mathsf{infunctions}(z)$. To compute the set neigh_z, we run through all clauses $C \in \overline{F_z}$ and determine $\mathrm{var}(C) \cap \mathrm{var}_z$. This can be done in time polynomial in l, and the same holds for the set $\overline{\mathsf{neigh}_z}$. A function $f \in \mathsf{outfunctions}(z)$ maps each $X \in \mathsf{neigh}_z$ to a set $f(X) \in \mathsf{proj}(\overline{F_z^X}, X)$. By Lemma 46 (Chapter 11) the set $\mathsf{proj}(\overline{F_z^X}, X)$ can by computed in time polynomial in l for each $X \in \mathsf{neigh}_z$. Going through all pairs $(f, g) \in \mathsf{outfunctions}(z) \times \mathsf{infunctions}(z)$ amounts to going through all possible combinations of choices of $f(X) \in \mathsf{proj}(\overline{F_z^X}, X)$ for every $X \in \mathsf{neigh}_z$ and $g(Y) \in \mathsf{proj}(F_z^Y, Y)$ for each

$Y \in \overline{\mathsf{neigh}_z}$, of which there are at most $(m+1)^{2k}$ many. For each such pair (f,g) we compute the sets $\mathsf{union}(f)$ and $\mathsf{union}(g)$, which can be done in time polynomial in l. □

Lemma 75. *Let $x, y, z \in V(T)$ such that x and y are the children of z. Let $s_x \in \mathsf{shapes}(x)$, $s_y \in \mathsf{shapes}(y)$, and $s_z \in \mathsf{shapes}(z)$. It can be decided in time $O(l)$ whether s_x and s_y generate s_z.*

Proof. We have to check conditions (1) to (3), which can easily be done in time linear in l since the sets of clauses involved have cardinality at most $m \leq l$. □

Lemma 76. *For any leaf node $z \in V(T)$ a lower bounding function for z can be computed in time $O(l)$.*

Proof. Every leaf $z \in V(T)$ is either associated with a clause $C \in F$ or a variable $v \in \mathrm{var}(F)$. In the first case, $\mathrm{var}_z = \emptyset$ and so $\mathsf{neigh}_z = \emptyset$ if $\overline{F_z} = \emptyset$ or $\mathsf{neigh}_z = \{\emptyset\}$. It follows that the set $\mathsf{outfunctions}(z)$ only contains the empty function or the function f with domain $\{\emptyset\}$ such that $f(\emptyset) = \emptyset$. For the set $\overline{\mathsf{neigh}_z}$ we get $\overline{\mathsf{neigh}_z} = \{\mathrm{var}(C)\}$ for the unique clause $C \in F_z$. Since $F_z^{\mathrm{var}(C)} = \{C\}$ we have $\mathsf{proj}(F_z^{\mathrm{var}(C)}, \mathrm{var}(C)) = \{\{C\}, \emptyset\}$ and thus $\mathsf{infunctions}(z) = \{g, g'\}$, where g is the function with domain $\{\mathrm{var}(C)\}$ such that $g(\mathrm{var}(C)) = \{C\}$ and g' is the function with domain $\{\mathrm{var}(C)\}$ such that $g'(\mathrm{var}(C)) = \emptyset$. It follows that $\mathsf{shapes}^{\mathsf{res}}(z)$ only contains the shapes $(\emptyset, \emptyset)$ and $(\emptyset, \{C\})$. The set var_z is empty, so 2^{var_z} contains only the empty assignment which does not satisfy any clause. Hence $n_z((\emptyset, \emptyset)) = 0$ and $n_z((\emptyset, \{C\})) = 1$.

In the second case, $\mathrm{var}_z = \{v\}$ for some variable $v \in \mathrm{var}(F)$. Since $F_z = \emptyset$ we have $\overline{\mathsf{neigh}_z} = \emptyset$ and so $\mathsf{infunctions}(z)$ only contains the empty function. The set neigh_z contains $\{v\}$, and the empty set if there is a clause $C \in F$ with $v \notin \mathrm{var}(C)$. We get $\mathsf{proj}(\overline{F_z^{\{v\}}}, \{v\}) = \{F_v^+, F_v^-\}$, where F_v^+ is the set of clauses of F with a positive occurrence of v, and F_v^- is the set of clauses F with a negative occurrence of v. Moreover, $\mathsf{proj}(\overline{F_z^{\emptyset}}, \emptyset) = \{\emptyset\}$. It follows that $\mathsf{shapes}^{\mathsf{res}}(z) = \{(F_v^+, \emptyset), (F_v^-, \emptyset)\}$. The set 2^{var_z} only contains the assignments τ_0 with $\tau_0(v) = 0$ and τ_1 with $\tau_1(v) = 1$, and $\overline{F_z}(\tau_0) = F_v^-$ and $\overline{F_z}(\tau_1) = F_v^+$. This implies $n_z((F_v^+, \emptyset)) = 1$ and $n_z((F_v^-, \emptyset)) = 1$.

For both cases the set $\mathsf{shapes}^{\mathsf{res}}(z)$ and the values $n_z(s)$ for each $s \in \mathsf{shapes}^{\mathsf{res}}(z)$ can be computed in time $O(l)$. These values provide a lower bounding function for z. □

Lemma 77. *There is a polynomial p (independent of F) such that for any inner node $z \in V(T)$, a lower bounding function for z can be computed in time $m^{6k}p(l)$ from lower bounding functions for the children of z.*

Proof. By Lemma 74, there is a polynomial q (independent of z) such that the set $\mathsf{shapes}^{\mathsf{res}}(z)$ can be computed in time $O(m^{2k}q(l))$. Let x and y denote the children of z, and let l_x and l_y be lower bounding functions for x and y. We compute a lower bounding function l_z for z as follows. Initially, we set $l_z(s_z) = 0$ for all $s_z \in \mathsf{shapes}^{\mathsf{res}}(z)$. We then run through all triples of shapes $s_x \in \mathsf{shapes}^{\mathsf{res}}(x)$, $s_y \in \mathsf{shapes}^{\mathsf{res}}(y)$, and $s_z \in \mathsf{shapes}^{\mathsf{res}}(z)$ and check whether s_x and s_y generate s_z. If that is the case, we add $l_x(s_x)\, l_y(s_y)$ to $l_z(s_z)$.

Correctness follows from Lemma 73 and the fact that l_x, l_y are lower bounding functions for x and y. The bound on the runtime is obtained as follows. By Corollary 6 there are at most $(m+1)^{6k}$ triples (s_x, s_y, s_z) of shapes that have to be considered. For each one, one can decide whether s_x and s_y generate s_z in time $O(l)$ by Lemma 75. Depending on the outcome of that decision we may have to multiply two integers $l_x(s_x)$ and $l_y(s_y)$, adding the result to $l_z(s_z)$. These values are bounded from above by $2^{|\operatorname{var}(F)|} \leq 2^l$, so their binary representations have size $O(l)$ and these arithmetic operations can be carried out in time $O(l^2)$. □

Lemma 78. *There is a polynomial p (independent of F) such that a lower bounding function for z can be computed for every $z \in V(T)$ in time $m^{6k}p(l)$.*

Proof. By Lemma 76, a lower bounding function for a leaf of T can be computed in time $O(l)$. The number of leaves of T is in $O(l)$, so we can compute lower bounding functions for all of them in time $O(l^2)$. By Lemma 77, we can then compute lower bounding functions for each inner node $z \in V(T)$ in a bottom up manner. For each inner node z, a lower bounding function can computed in time $m^{6k}q(l)$ by Lemma 77, where q is a polynomial independent of z. The number of inner nodes of T is in $O(l)$, so this requires $O(m^{6k} l\, q(l))$ time in total. □

Proposition 24. *There is a polynomial p and an algorithm that, given a formula F and a decomposition tree (T, δ) of $I(F)$, computes the number of satisfying assignments of F in time $m^{6k}p(l)$. Here, m denotes the number of clauses of F, l denotes the length of F, and $k = \mathit{index}(T, \delta)$.*

Proof. By Lemma 78 a lower bounding function l_r for the root r of T can be computed in time $m^{6k}\, q(l)$, where q is a polynomial independent of F. By Lemma 66, the value $n_r((\emptyset, \emptyset))$ corresponds to the number of satisfying total truth assignments of F, and the shape $(\emptyset, \emptyset)$ is proper. Since l_r is a lower bounding function for r it follows that $l_r((\emptyset, \emptyset)) = n_r((\emptyset, \emptyset))$. □

In combination with an algorithm for computing a decomposition tree of an input formula's incidence graph, we get an algorithm for #SAT.

Theorem 16. *There is a function $f : \mathbb{N} \to \mathbb{N}$ such that #SAT can be solved in time $f(k)\, l^3 + l^{O(2^k)}$ on CNF formulas of symmetric incidence clique-width k and length l.*

Proof. It follows from Theorem 12 and Lemma 36 that there exist a function $f : \mathbb{N} \to \mathbb{N}$ and an algorithm that, given a graph G with n vertices and symmetric clique-width k, computes a decomposition tree (T, δ) such that $\mathit{index}(T, \delta) \leq 2^k$ in time $f(k)\, n^3$. Let F be a CNF formula of symmetric incidence clique-width k and length l. We use this algorithm to compute a decomposition tree (T, δ) of the incidence graph $I(F)$ in time $f(k)\, l^3$. From F and (T, δ), the number of satisfying total truth assignments of F can be computed in time $l^{O(2^k)}$ by Proposition 24. □

Summary

We have shown that #SAT is polynomial-time tractable for classes of formulas with incidence graphs of bounded symmetric clique-width (or bounded clique-width, or bounded rank-width). More specifically, we proved that the model count of a CNF formula of length l and symmetric clique-width k can be computed in time $f(k)\,l^3 + l^{O(2^k)}$. Since already SAT, parameterized by symmetric incidence clique-width, is W[1]-hard, we cannot expect to get rid of the dependence on k in the exponent of l. However, fast algorithms for computing (near) optimal decomposition trees for an input formula's incidence graph might allow us to replace the term $l^{O(2^k)}$ in the runtime bound by $l^{O(k)}$.

Using a canonical transformation of tree decompositions into branch decompositions, the algorithm presented in this chapter can also be used to show that the model count of a formula of length l and modular incidence treewidth k can be computed in time $l^{O(k)}$, albeit with a factor in the exponent that is worse than the one in the bound proved in Chapter 11.

CHAPTER 13

Conclusion

We proved new results on the complexity of #SAT with respect to structural parameters, presenting algorithms that compute the model count of formulas F with modular incidence treewidth k or symmetric incidence clique-width k in time $n^{f(k)}$. In particular, #SAT is polynomial-time tractable for classes of formulas where these parameters are bounded by a constant.

It appears that our techniques cannot be applied to prove tractability of #SAT parameterized by the β-hypertree-width, and the complexity of #SAT with respect to this parameter remains open, though polynomial-time tractability was recently proved for the special case of β-acyclic formulas using a clever generalization of Davis-Putnam resolution [19].

Our interest in the complexity of #SAT with respect to ever more general structural parameters is mainly theoretical. In closing, we would like to point out an advantage of symmetric incidence clique-width over less general structural parameters which may serve as a more practical motivation. As mentioned in the introduction to this part, inference in Bayesian networks can be reduced to (weighted) model counting. The underlying undirected graph of a Bayesian network often has small treewidth, but this structure is not preserved by standard reductions such as the one by Sang, Beame, and Kautz [105]. More precisely, the incidence treewidth of the resulting formula can be much larger than the treewidth of the Bayesian network: for each conditional probability table of the network, the reduction introduces a complete bipartite subgraph of essentially the same size. On the other hand, one can show that the reduction preserves symmetric incidence clique-width. More specifically, a decomposition tree of the Bayesian network can be turned into a decomposition tree of the formula's incidence graph without increasing its width (index). Accordingly, it suffices to compute a good decomposition of the Bayesian network. Observations of this kind may help tackle one of the main challenges in applying these algorithms to real-world instances: computing good decompositions as a starting point for dynamic programming [70, 11, 68].

Recent advances in practical model counting are based on randomized approximation algorithms [55, 81, 26, 41]. These algorithms use randomly generated parity constraints to partition the space of satisfying assignments into small cells; using a SAT solver as an oracle, the model count of an individual cell can be determined in order to obtain an estimate of the number of satisfying assignments overall. For this approach to work, the constraints must involve sufficiently many variables to ensure that models are distributed evenly across cells, but large parity constraints can be detrimental to the performance of SAT solvers. Finding suitable constraints is the fundamental problem in designing such algorithms. Surprisingly, it was shown experimentally that constraints involving only a few variables are often sufficient to obtain good estimates [53]. It would be interesting to see whether restrictions on structural parameters can help explain this phenomenon, and more generally, whether randomized approximation algorithms can benefit from structural restrictions.

Related work. The *signed rank-width* of a formula is equivalent to its signed incidence clique-width, in the sense that these parameters dominate each other. Ganian, Hlinený and Obdrzálek proved that the models of a formula can be counted by an algorithm whose runtime is single-exponential in the signed rank-width [47]. Since the signed rank-width of a formula never exceeds its signed incidence clique-width but can be exponentially smaller, this constitutes an improvement over the algorithm by Fischer, Makowsky, and Ravve, whose runtime is single-exponential in the signed incidence clique-width [44].

Saether, Telle, and Vatshelle showed that our dynamic programming algorithm for symmetric clique-width can be modified so as to precompute the set of proper shapes (see Chapter 12) for each node in the decomposition tree [99]. The resulting algorithm runs in time polynomial in the maximum number of proper shapes over all nodes, a value that essentially corresponds to what they call the *PS-width* of a decomposition tree. Defining the PS-width of a formula F as the minimum PS-width of a decomposition tree of $I(F)$, they show that the model count of F can be computed in time polynomial in the length and PS-width of F, assuming that a decomposition tree of this width is provided as part of the input. They proceed to prove that formulas whose incidence graph are interval bigraphs have small PS-width and that the corresponding decomposition trees can be found in polynomial time, so that #SAT is polynomial-time tractable for such formulas. They further showed that a formula with m clauses and an incidence graph of MIM-width k has PS-width at most m^k, where MIM-width is a parameter that strictly dominates clique-width [122]. Since it is not known whether decomposition trees of small PS-width or MIM-width can be computed efficiently, these results falls short of proving tractability of #SAT parameterized by PS-width or incidence MIM-width.

There is an orthogonal family of structural parameters based on so-called *backdoors* [48], a notion originally introduced to explain the heavy-tailed runtime distributions of DPLL-style backtrack search methods [125]. A *strong backdoor* of a formula F for a class $\mathcal{C}$ is a set S of variables such that for each assignment $\tau : S \to \{0, 1\}$, the instance $F[\tau]$ obtained by instantiating F with τ belongs to the class $\mathcal{C}$. If #SAT is polynomial-time tractable for formulas in $\mathcal{C}$ and the detection of strong backdoors for $\mathcal{C}$ is

fixed-parameter tractable, then #SAT is fixed-parameter tractable parameterized by the size of a smallest backdoor for $\mathcal{C}$. Gaspers and Szeider proved that #SAT is fixed-parameter tractable when parameterized by the size of a smallest strong backdoor for a class of bounded incidence treewidth [49]. A set $S \subseteq \mathrm{var}(F)$ is a *deletion backdoor* of F for $\mathcal{C}$ if the formula obtained by deleting the literals $x, \neg x$ for each $x \in S$ is in $\mathcal{C}$. Nishimura, Ragde, and Szeider showed that #SAT is fixed-parameter tractable parameterized by the size of a smallest deletion backdoor for the class of so-called cluster formulas [87].

Structural parameters as studied in this work can be analogously specified for the constraint satisfaction problem (CSP), using constraints instead of clauses in the definition of graphs (or hypergraphs) associated with an instance. The complexity of CSP with respect to structural parameters is well understood. For a class $\mathcal{H}$ of hypergraphs, let $\mathrm{CSP}(\mathcal{H})$ denote the class of instances of CSP with associated hypergraphs in $\mathcal{H}$. If the maximum edge size of hypergraphs in $\mathcal{H}$ is bounded, $\mathrm{CSP}(\mathcal{H})$ is polynomial-time tractable if and only if $\mathcal{H}$ has bounded treewidth, subject to a standard assumption from parameterized complexity [65]. A similar result can be proved for the counting version of CSP, subject to a weaker hypothesis [34]. Boolean constraint languages of bounded arity can be expressed as propositional formulas of bounded clause size with constant overhead and without changing the associated hypergraph. Accordingly, these results imply tractability of #SAT (respectively, SAT) for classes of bounded (primal) treewidth and bounded clause size. For the case of unbounded arities, this correspondence breaks down. In this case, $\mathrm{CSP}(\mathcal{H})$ is fixed-parameter tractable, parameterized by the size of the query, if and only if $\mathcal{H}$ has bounded *submodular width*, assuming the Exponential Time Hypothesis [85]. Submodular width is a parameter that strictly dominates α-hypertree-width, but as we saw in Chapter 10, #SAT is #P-complete already for α-acyclic formulas, and a similar reduction proves NP-hardness of SAT for α-acyclic formulas [103]. This discrepancy is caused by the fact that clauses provide a very succinct representation of (certain) constraint relations: represented explicitly, a clause over n variables corresponds to a constraint of size $2^n - 1$. Chen and Grohe classified the complexity of CSP for more succinct *generalized DNF* representations and unbounded arity, showing that *bounded incidence width modulo homomorphic equivalence* is a necessary and sufficient condition for tractability [27].

Bibliography

[1] C. Ansótegui, C. P. Gomes, and B. Selman. The Achilles' heel of QBF. In M. M. Veloso and S. Kambhampati, editors, *AAAI Conference on Artificial Intelligence - AAAI 2005*, pages 275–281. AAAI Press / The MIT Press, 2005.

[2] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, 8(2):277–284, 1987.

[3] B. Aspvall, M. F. Plass, and R. E. Tarjan. A linear-time algorithm for testing the truth of certain quantified Boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.

[4] A. Atserias and S. Oliva. Bounded-width QBF is pspace-complete. *J. of Computer and System Sciences*, 80(7):1415–1429, 2014.

[5] A. Ayari and D. A. Basin. Qubos: Deciding quantified boolean logic using propositional satisfiability solvers. In M. Aagaard and J. W. O'Leary, editors, *Formal Methods in Computer-Aided Design - FMCAD 2002*, volume 2517 of *Lecture Notes in Computer Science*, pages 187–201. Springer Verlag, 2002.

[6] F. Bacchus, S. Dalmao, and T. Pitassi. Algorithms and complexity results for #SAT and Bayesian inference. In *IEEE Symposium on Foundations of Computer Science - FOCS 2003*, pages 340–351, 2003.

[7] V. Balabanov, H.-J. K. Chiang, and J.-H. R. Jiang. Henkin quantifiers and Boolean formulae: A certification perspective of DQBF. *Theoretical Computer Science*, 523:86–100, 2014.

[8] V. Balabanov and J.-H. R. Jiang. Unified QBF certification and its applications. *Formal Methods in System Design*, 41(1):45–65, 2012.

[9] M. Benedetti. Quantifier trees for QBFs. In *Theory and Applications of Satisfiability Testing - SAT 2005*, volume 3569 of *LNCS*, pages 378–385, 2005.

[10] O. Beyersdorff, L. Chew, and M. Janota. Proof complexity of resolution-based QBF calculi. *Electronic Colloquium on Computational Complexity (ECCC)*, 21:120, 2014.

[11] M. Beyß. Fast algorithm for rank-width. In A. Kucera, T. A. Henzinger, J. Nesetril, T. Vojnar, and D. Antos, editors, *Mathematical and Engineering Methods in Computer Science - MEMICS 2012*, volume 7721 of *Lecture Notes in Computer Science*, pages 82–93. Springer Verlag, 2013.

[12] A. Biere. Resolve and expand. In *Theory and Applications of Satisfiability Testing - SAT 2004*, pages 59–70, 2004.

[13] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In R. Cleaveland, editor, *Tools and Algorithms for Construction and Analysis of Systems - TACAS 1999*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer Verlag, 1999.

[14] A. Biere and F. Lonsing. A compact representation for syntactic dependencies in QBFs. In O. Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009*, volume 5584 of *Lecture Notes in Computer Science*, pages 398–411. Springer Verlag, 2009.

[15] A. Biere and F. Lonsing. Integrating dependency schemes in search-based QBF solvers. In O. Strichman and S. Szeider, editors, *Theory and Applications of Satisfiability Testing - SAT 2010*, volume 6175 of *Lecture Notes in Computer Science*, pages 158–171. Springer Verlag, 2010.

[16] E. Birnbaum and E. L. Lozinskii. The good old Davis-Putnam procedure helps counting models. *J. Artif. Intell. Res.*, pages 457–477, 1999.

[17] P. Bjesse, T. Leonard, and A. Mokkedem. Finding bugs in an alpha microprocessor using satisfiability solvers. In G. Berry, H. Comon, and A. Finkel, editors, *Computer Aided Verification - CAV 2001*, pages 454–464, 2001.

[18] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.

[19] J. Brault-Baron, F. Capelli, and S. Mengel. Understanding model counting for beta-acyclic cnf-formulas. In E. W. Mayr and N. Ollinger, editors, *Symposium on Theoretical Aspects of Computer Science - STACS 2015*, volume 30 of *LIPIcs*, pages 143–156. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[20] U. Bubeck. *Model-based transformations for quantified Boolean formulas.* PhD thesis, University of Paderborn, 2010.

[21] U. Bubeck and H. Kleine Büning. Bounded universal expansion for preprocessing QBF. In J. Marques-Silva and K. A. Sakallah, editors, *Theory and Applications of Satisfiability Testing - SAT 2007*, volume 4501 of *Lecture Notes in Computer Science*, pages 244–257. Springer Verlag, 2007.

[22] B.-M. Bui-Xuan, J. A. Telle, and M. Vatshelle. H-join decomposable graphs and algorithms with runtime single exponential in rankwidth. *Discrete Applied Mathematics*, 158(7):809–819, 2010.

[23] B.-M. Bui-Xuan, J. A. Telle, and M. Vatshelle. Boolean-width of graphs. *Theoretical Computer Science*, 412(39):5187–5204, 2011.

[24] H. K. Büning, M. Karpinski, and A. Flögel. Resolution for quantified Boolean formulas. *Information and Computation*, 117(1):12–18, 1995.

[25] M. Cadoli, M. Schaerf, A. Giovanardi, and M. Giovanardi. An algorithm to evaluate Quantified Boolean Formulae and its experimental evaluation. *Journal of Automated Reasoning*, 28(2), 2002.

[26] S. Chakraborty, K. S. Meel, and M. Y. Vardi. A scalable approximate model counter. In C. Schulte, editor, *Principles and Practice of Constraint Programming - CP 2013*, volume 8124 of *Lecture Notes in Computer Science*, pages 200–216. Springer Verlag, 2013.

[27] H. Chen and M. Grohe. Constraint satisfaction with succinctly specified relations. *J. of Computer and System Sciences*, 76(8):847–860, 2010.

[28] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, third edition, 2009.

[29] B. Courcelle. Clique-width of countable graphs: a compactness property. *Discrete Mathematics*, 276(1-3):127–148, 2004.

[30] B. Courcelle, J. Engelfriet, and G. Rozenberg. Handle-rewriting hypergraph grammars. *J. of Computer and System Sciences*, 46(2):218–270, 1993.

[31] B. Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.

[32] B. Courcelle, J. A. Makowsky, and U. Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discr. Appl. Math.*, 108(1-2):23–52, 2001.

[33] B. Courcelle and S. Olariu. Upper bounds to the clique-width of graphs. *Discr. Appl. Math.*, 101(1-3):77–114, 2000.

[34] V. Dalmau and P. Jonsson. The complexity of counting homomorphisms seen from the other side. *Theoretical Computer Science*, 329(1-3):315–323, 2004.

[35] A. Darwiche. New advances in compiling CNF into decomposable negation normal form. In R. L. de Mántaras and L. Saitta, editors, *European Conference on Artificial Intelligence - ECAI 2004*, pages 328–332. IOS Press, 2004.

[36] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.

[37] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer Verlag, New York, 1999.

[38] U. Egly, M. Seidl, and S. Woltran. A solver for QBFs in negation normal form. *Constraints*, 14(1):38–79, 2009.

[39] U. Egly, H. Tompits, and S. Woltran. On quantifier shifting for quantified Boolean formulas. In *Proc. SAT'02 Workshop on Theory and Applications of Quantified Boolean Formulas*, pages 48–61. Informal Proceedings, 2002.

[40] P. H. ený and S. il Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008.

[41] S. Ermon, C. P. Gomes, A. Sabharwal, and B. Selman. Low-density parity constraints for hashing-based discrete integration. In *International Conference on Machine Learning - ICML 2014*, volume 32 of *JMLR Proceedings*, pages 271–279, 2014.

[42] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. of the ACM*, 30(3):514–550, 1983.

[43] M. R. Fellows, F. A. Rosamond, U. Rotics, and S. Szeider. Clique-width is NP-complete. *SIAM J. Discrete Math.*, 23(2):909–939, 2009.

[44] E. Fischer, J. A. Makowsky, and E. R. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discr. Appl. Math.*, 156(4):511–529, 2008.

[45] J. Flum and M. Grohe. *Parameterized Complexity Theory*, volume XIV of *Texts in Theoretical Computer Science. An EATCS Series*. Springer Verlag, Berlin, 2006.

[46] R. Ganian and P. Hliněný. On parse trees and Myhill-Nerode-type tools for handling graphs of bounded rank-width. *Discr. Appl. Math.*, 158(7):851–867, 2010.

[47] R. Ganian, P. Hlinený, and J. Obdrzálek. Better algorithms for satisfiability problems for formulas of bounded rank-width. *Fund. Inform.*, 123(1):59–76, 2013.

[48] S. Gaspers and S. Szeider. Backdoors to satisfaction. In H. L. Bodlaender, R. Downey, F. V. Fomin, and D. Marx, editors, *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370 of *Lecture Notes in Computer Science*, pages 287–317. Springer Verlag, 2012.

[49] S. Gaspers and S. Szeider. Strong backdoors to bounded treewidth SAT. In *IEEE Symposium on Foundations of Computer Science - FOCS 2013*, pages 489–498. IEEE Computer Society, 2013.

[50] E. Giunchiglia, P. Marin, and M. Narizzano. Reasoning with quantified Boolean formulas. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185, pages 761–780. IOS Press, 2009.

[51] E. Giunchiglia, M. Narizzano, and A. Tacchella. Clause/term resolution and learning in the evaluation of Quantified Boolean Formulas. *J. Artif. Intell. Res.*, 26:371–416, 2006.

[52] E. Giunchiglia, M. Narizzano, and A. Tacchella. Quantifier structure in search-based procedures for QBFs. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 26(3):497–507, 2007.

[53] C. P. Gomes, J. Hoffmann, A. Sabharwal, and B. Selman. Short XORs for model counting: From theory to practice. In J. Marques-Silva and K. A. Sakallah, editors, *Theory and Applications of Satisfiability Testing - SAT 2007*, volume 4501 of *Lecture Notes in Computer Science*, pages 100–106. Springer Verlag, 2007.

[54] C. P. Gomes, H. Kautz, A. Sabharwal, and B. Selman. Satisfiability solvers. In *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 89–134. Elsevier, 2008.

[55] C. P. Gomes, A. Sabharwal, and B. Selman. Model counting: A new strategy for obtaining good bounds. In *AAAI Conference on Artificial Intelligence - AAAI 2006*, pages 54–61. AAAI Press, 2006.

[56] C. P. Gomes, A. Sabharwal, and B. Selman. Model counting. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185, pages 633–654. IOS Press, 2009.

[57] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. *J. of Computer and System Sciences*, 64(3):579–627, 2002.

[58] G. Gottlob and R. Pichler. Hypergraphs in model checking: acyclicity and hypertree-width versus clique-width. *SIAM J. Comput.*, 33(2):351–378, 2004.

[59] A. Goultiaeva. *Exploiting Problem Structure in QBF Solving*. PhD thesis, University of Toronto, Mar. 2014.

[60] A. Goultiaeva and F. Bacchus. Exploiting QBF duality on a circuit representation. In M. Fox and D. Poole, editors, *AAAI Conference on Artificial Intelligence - AAAI 2010*. AAAI Press, 2010.

[61] A. Goultiaeva and F. Bacchus. Recovering and utilizing partial duality in QBF. In M. Järvisalo and A. V. Gelder, editors, *Theory and Applications of Satisfiability Testing - SAT 2013*, volume 7962 of *Lecture Notes in Computer Science*, pages 83–99. Springer Verlag, 2013.

[62] A. Goultiaeva, A. V. Gelder, and F. Bacchus. A uniform approach for generating proofs and strategies for both true and false QBF formulas. In T. Walsh, editor, *International Joint Conference on Artificial Intelligence - IJCAI 2011*, pages 546–553. IJCAI/AAAI, 2011.

[63] A. Goultiaeva, M. Seidl, and A. Biere. Bridging the gap between dual propagation and CNF-based QBF solving. In E. Macii, editor, *Design, Automation and Test in Europe - DATE 13*, pages 811–814. EDA Consortium San Jose, CA, USA / ACM DL, 2013.

[64] M. Grohe and D. Marx. Constraint solving via fractional edge covers. *ACM Transactions on Algorithms*, 11(1):4, 2014.

[65] M. Grohe, T. Schwentick, and L. Segoufin. When is the evaluation of conjunctive queries tractable? In *ACM Symposium on Theory of Computing - STOC 2001*, pages 657–666, New York, 2001. ACM.

[66] M. Habib and C. Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010.

[67] M. Heule, M. Seidl, and A. Biere. A unified proof system for QBF preprocessing. In S. Demri, D. Kapur, and C. Weidenbach, editors, *International Joint Conference on Automated Reasoning - IJCAR 2014*, volume 8562 of *Lecture Notes in Computer Science*, pages 91–106. Springer Verlag, 2014.

[68] M. Heule and S. Szeider. A SAT approach to clique-width. In M. Järvisalo and A. V. Gelder, editors, *Theory and Applications of Satisfiability Testing - SAT 2013*, volume 7962 of *Lecture Notes in Computer Science*, pages 318–334. Springer Verlag, 2013.

[69] P. Hlinený, S. Oum, D. Seese, and G. Gottlob. Width parameters beyond tree-width and their applications. *The Computer Journal*, 51(3):326–362, 2008.

[70] E. M. Hvidevold, S. Sharmin, J. A. Telle, and M. Vatshelle. Finding good decompositions for dynamic programming on dense graphs. In D. Marx and P. Rossmanith, editors, *Parameterized and Exact Computation - IPEC 2011*, volume 7112 of *Lecture Notes in Computer Science*, pages 219–231. Springer Verlag, 2012.

[71] M. Janota, L. Chew, and O. Beyersdorff. On unification of QBF resolution-based calculi. *Electronic Colloquium on Computational Complexity (ECCC)*, 2014.

[72] M. Janota, W. Klieber, J. Marques-Silva, and E. M. Clarke. Solving QBF with counterexample guided refinement. In A. Cimatti and R. Sebastiani, editors, *Theory and Applications of Satisfiability Testing - SAT 2012*, volume 7317 of *Lecture Notes in Computer Science*, pages 114–128. Springer Verlag, 2012.

[73] M. Janota and J. Marques-Silva. On propositional QBF expansions and Q-resolution. In M. Järvisalo and A. V. Gelder, editors, *Theory and Applications of Satisfiability Testing - SAT 2013*, volume 7962 of *Lecture Notes in Computer Science*, pages 67–82. Springer Verlag, 2013.

[74] R. J. B. Jr. and J. D. Pehoushek. Counting models using connected components. In H. A. Kautz and B. W. Porter, editors, *AAAI Conference on Artificial Intelligence - AAAI 2000*, pages 157–162. AAAI Press / The MIT Press, 2000.

[75] T. Jussila and A. Biere. Compressing BMC encodings with QBF. *Electr. Notes Theor. Comput. Sci.*, 174(3):45–56, 2007.

[76] P. Kaski, M. Koivisto, and J. Nederlof. Homomorphic hashing for sparse coefficient extraction. In *International Symposion on Parameterized and Exact Computation - IPEC 2012*, 2012.

[77] H. Kautz and B. Selman. Pushing the envelope: planning, propositional logic, and stochastic search. In *AAAI Conference on Artificial Intelligence - AAAI 1996*, pages 1194–1201. AAAI Press, 1996.

[78] H. Kleine Büning and T. Lettman. *Propositional logic: Deduction and algorithms*. Cambridge University Press, Cambridge, 1999.

[79] W. Klieber, S. Sapra, S. Gao, and E. M. Clarke. A non-prenex, non-clausal QBF solver with game-state learning. In O. Strichman and S. Szeider, editors, *Theory and Applications of Satisfiability Testing - SAT 2010*, volume 6175 of *Lecture Notes in Computer Science*, pages 128–142. Springer Verlag, 2010.

[80] T. Kloks. *Treewidth: Computations and Approximations*. Springer Verlag, Berlin, 1994.

[81] L. Kroc, A. Sabharwal, and B. Selman. Leveraging belief propagation, backtrack search, and statistics for model counting. In L. Perron and M. A. Trick, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems - CPAIOR 2008*, volume 5015 of *Lecture Notes in Computer Science*, pages 127–141. Springer Verlag, 2008.

[82] F. Lonsing. *Dependency Schemes and Search-Based QBF Solving: Theory and Practice*. PhD thesis, Johannes Kepler University, Linz, Austria, Apr. 2012.

[83] F. Lonsing and A. Biere. Nenofex: Expanding NNF for QBF solving. In H. K. Büning and X. Zhao, editors, *Theory and Applications of Satisfiability Testing - SAT 2008*, volume 4996 of *Lecture Notes in Computer Science*, pages 196–210. Springer Verlag, 2008.

[84] A. G. M. Prasad, A. Biere. A survey of recent advances in SAT-based formal verification. *Software Tools for Technology Transfer*, 7(2):156–173, 2005.

[85] D. Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *J. of the ACM*, 60(6):42, 2013.

[86] A. Niemetz, M. Preiner, F. Lonsing, M. Seidl, and A. Biere. Resolution-based certificate extraction for QBF. In A. Cimatti and R. Sebastiani, editors, *Theory and Applications of Satisfiability Testing - SAT 2012*, volume 7317 of *Lecture Notes in Computer Science*, pages 430–435. Springer Verlag, 2012.

[87] N. Nishimura, P. Ragde, and S. Szeider. Solving #SAT using vertex covers. *Acta Informatica*, 44(7-8):509–523, 2007.

[88] S. Ordyniak, D. Paulusma, and S. Szeider. Satisfiability of acyclic and almost acyclic CNF formulas. *Theoretical Computer Science*, 481:85–99, 2013.

[89] S. Oum and P. Seymour. Approximating clique-width and branch-width. *Journal Comb. Theory B*, 96(4):514–528, 2006.

[90] G. Pan and M. Y. Vardi. Fixed-parameter hierarchies inside PSPACE. In *IEEE Symposium on Logic in Computer Science - LICS 2006*, pages 27–36. IEEE Computer Society, 2006.

[91] D. Paulusma, F. Slivovsky, and S. Szeider. Model counting for CNF formulas of bounded modular treewidth. In N. Portier and T. Wilke, editors, *Symposium on Theoretical Aspects of Computer Science - STACS 2013*, volume 20 of *LIPIcs*, pages 55–66. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

[92] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. The Morgan Kaufmann Series in Representation and Reasoning. Morgan Kaufmann, San Mateo, CA, 1988.

[93] G. Peterson, J. Reif, and S. Azhar. Lower bounds for multiplayer noncooperative games of incomplete information. *Computers & Mathematics with Applications*, 41(7-8):957 – 992, 2001.

[94] K. Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *J. of Computer and System Sciences*, 67(4):757–771, 2003.

[95] J. Rintanen. Constructing conditional plans by a theorem-prover. *J. Artif. Intell. Res.*, 10:323–352, 1999.

[96] N. Robertson and P. D. Seymour. Graph minors X. Obstructions to tree-decomposition. *J. Combin. Theory Ser. B*, 52(2):153–190, 1991.

[97] D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1-2):273–302, 1996.

[98] A. Sabharwal, C. Ansotegui, C. Gomes, J. Hart, and B. Selman. QBF modeling: Exploiting player symmetry for simplicity and efficiency. In *Theory and Applications of Satisfiability Testing - SAT 2006*, volume 4121 of *Lecture Notes in Computer Science*, pages 382–395. Springer Verlag, 2006.

[99] S. H. Sæther, J. Telle, and M. Vatshelle. Solving MaxSAT and #SAT on structured CNF formulas. In C. Sinz and U. Egly, editors, *Theory and Applications of Satisfiability Testing - SAT 2014*, volume 8561 of *Lecture Notes in Computer Science*, pages 16–31. Springer Verlag, 2014.

[100] M. Samer. Variable dependencies of quantified CSPs. In I. Cervesato, H. Veith, and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - LPAR 2008*, volume 5330 of *Lecture Notes in Computer Science*, pages 512–527. Springer Verlag, 2008.

[101] M. Samer and S. Szeider. Backdoor sets of quantified boolean formulas. In J. Marques-Silva and K. A. Sakallah, editors, *Theory and Applications of Satisfiability Testing - SAT 2007*, volume 4501 of *Lecture Notes in Computer Science*, pages 230–243, 2007.

[102] M. Samer and S. Szeider. Backdoor sets of quantified Boolean formulas. *Journal of Automated Reasoning*, 42(1):77–97, 2009.

[103] M. Samer and S. Szeider. Algorithms for propositional model counting. *J. Discrete Algorithms*, 8(1):50–64, 2010.

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

[105] T. Sang, P. Beame, and H. A. Kautz. Performing Bayesian inference by weighted model counting. In *AAAI Conference on Artificial Intelligence - AAAI 2005*, pages 475–481. AAAI Press / The MIT Press, 2005.

[106] J. P. M. Silva. The impact of branching heuristics in propositional satisfiability algorithms. In P. Barahona and J. J. Alferes, editors, *Progress in Artificial Intelligence, Portuguese Conference on Artificial Intelligence - EPIA '99*, volume 1695 of *Lecture Notes in Computer Science*, pages 62–74. Springer Verlag, 1999.

[107] F. Slivovsky and S. Szeider. Computing resolution-path dependencies in linear time. In A. Cimatti and R. Sebastiani, editors, *Theory and Applications of Satisfiability Testing - SAT 2012*, volume 7317 of *Lecture Notes in Computer Science*, pages 58–71. Springer Verlag, 2012.

[108] F. Slivovsky and S. Szeider. Model counting for formulas of bounded clique-width. In L. Cai, S. Cheng, and T. W. Lam, editors, *International Symposium on Algorithms and Computation - ISAAC 2013*, volume 8283 of *Lecture Notes in Computer Science*, pages 677–687. Springer Verlag, 2013.

[109] F. Slivovsky and S. Szeider. Variable dependencies and Q-resolution. In *Workshop on Quantified Boolean Formulas - QBF 2013*, 2013.

[110] F. Slivovsky and S. Szeider. Resolution paths and term resolution. In *Workshop on Quantified Boolean Formulas - QBF 2014*, 2014.

[111] F. Slivovsky and S. Szeider. Variable dependencies and Q-resolution. In C. Sinz and U. Egly, editors, *Theory and Applications of Satisfiability Testing - SAT 2014*, volume 8561 of *Lecture Notes in Computer Science*, pages 269–284. Springer Verlag, 2014.

[112] L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[113] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *Proc. Theory of Computing*, pages 1–9. ACM, 1973.

[114] S. Szeider. On fixed-parameter tractable parameterizations of SAT. In E. Giunchiglia and A. Tacchella, editors, *Theory and Applications of Satisfiability Testing - SAT 2003*, volume 2919 of *Lecture Notes in Computer Science*, pages 188–202. Springer Verlag, 2004.

[115] M. Thurley. sharpsat - counting models with advanced component caching and implicit BCP. In A. Biere and C. P. Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT 2006*, volume 4121 of *Lecture Notes in Computer Science*, pages 424–429. Springer Verlag, 2006.

[116] S. Toda. On the computational power of PP and +P. In *IEEE Symposium on Foundations of Computer Science - FOCS 1989*, pages 514–519. IEEE Computer Soc., 1989.

[117] M. J. Valeriy Balabanov, Jie-Hong Roland Jiang and M. Widl. Efficient extraction of QBF (counter)models from long-distance resolution proofs. In *Workshop on Quantified Boolean Formulas - QBF 2014*, 2014.

[118] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

[119] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

[120] A. Van Gelder. Variable independence and resolution paths for quantified boolean formulas. In J. Lee, editor, *Principles and Practice of Constraint Programming - CP 2011*, volume 6876 of *Lecture Notes in Computer Science*, pages 789–803. Springer Verlag, 2011.

[121] M. Y. Vardi. Boolean satisfiability: theory and engineering. *Commun. ACM*, 57(3):5, 2014.

[122] M. Vatshelle. *New Width Parameters of Graphs*. PhD thesis, University of Bergen, 2012.

[123] E. Wanke. k-NLC graphs and polynomial algorithms. *Discr. Appl. Math.*, 54(2-3):251–266, 1994. Efficient algorithms and partial k-trees.

[124] W. Wei and B. Selman. A new approach to model counting. In F. Bacchus and T. Walsh, editors, *Theory and Applications of Satisfiability Testing - SAT 2005*, volume 3569 of *Lecture Notes in Computer Science*, pages 324–339. Springer Verlag, 2005.

[125] R. Williams, C. Gomes, and B. Selman. On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search. In *Theory and Applications of Satisfiability Testing - SAT 2003*, pages 222–230, 2003.

[126] L. Zhang. Solving QBF by combining conjunctive and disjunctive normal forms. In *AAAI Conference on Artificial Intelligence - AAAI 2006*, pages 143–150. AAAI Press, 2006.

[127] D. Zuckerman. On unapproximable versions of NP-complete problems. *SIAM J. Comput.*, 25(6):1293–1304, Dec. 1996.

Friedrich Slivovsky

Institute of Computer Graphics and Algorithms
Vienna University of Technology
Favoritenstraße 9-11
A-1040 Vienna, Austria
http://www.ac.tuwien.ac.at/people/fslivovsky/

Education

- PhD Student, Technical Sciences. Vienna UT, 2009 – present.
- MSc, Computational Intelligence. Vienna UT, 2009.
- BSc, Software & Information Engineering. Vienna UT, 2006.

Publications

5. Friedrich Slivovsky and Stefan Szeider. Variable Dependencies and Q-Resolution. *Theory and Applications of Satisfiability Testing - SAT 2014.* Lecture Notes in Computer Science, vol. 8561, pp. 269-284, Springer 2014.

4. Friedrich Slivovsky and Stefan Szeider. Model Counting for Formulas of Bounded Clique-Width. *International Symposium on Algorithms and Computation - ISAAC 2013.* Lecture Notes in Computer Science, vol. 8283, pp. 677-687, Springer 2013.

3. Robert Ganian, Friedrich Slivovsky, and Stefan Szeider. Meta-kernelization with Structural Parameters. *Symposium on Mathematical Foundations of Computer Science - MFCS 2013.* Lecture Notes in Computer Science, vol. 8087, pp. 457-468, Springer 2013.

2. Daniel Paulusma, Friedrich Slivovsky, and Stefan Szeider. Model Counting for CNF Formulas of Bounded Modular Treewidth. *Symposium on Theoretical Aspects of Computer Science - STACS 2013.* Leibniz International Proceedings in Informatics, vol. 20, pp 55-66, 2013.

1. Friedrich Slivovsky and Stefan Szeider. Computing Resolution-Path Dependencies in Linear Time. *Theory and Applications of Satisfiability Testing - SAT 2012.* Lecture Notes in Computer Science, vol. 7317, pp. 58-71, Springer 2012.