FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Advanced Dependency Analysis for QBF

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Mgr. Tomáš Peitl
Registration Number 01528158

to the Faculty of Informatics

at the TU Wien

Advisor: Prof. Dr. Stefan Szeider
Second advisor: Prof. Dr. Uwe Egly

The dissertation has been reviewed by:

Olaf Beyersdorff	Luca Pulina

Vienna, 13th September, 2019

Tomáš Peitl

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Erklärung zur Verfassung der Arbeit

Mgr. Tomáš Peitl
Favoritenstraße 9, 1040 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, 13. September 2019

Tomáš Peitl

Acknowledgements

I want to thank my supervisor Stefan Szeider, and my colleague and de-facto other supervisor Friedrich Slivovsky, for guidance and mentoring, for introducing me to the real academic world with all its peculiarities, and not least for introducing me to QBF. Another person from whom I learned a great deal about QBF is Florian Lonsing, whom I thank for helpful discussions related to QCDCL and QBF in general, and for a great course on the topic.

I was kindly supported by the FWF via the grants P27721 and W1255-N23, and I am particularly thankful for the ability to spend two fantastic months at the University of British Columbia in Vancouver, Canada; to the FWF and DK LogiCS, who paid, to Stefan, who suggested it, and to Holger Hoos, who hosted me and taught me a lot about machine learning. Unfortunately, I cannot say I thank God for the weather there.

I am indebted to my family, who supported me and let me grow throughout my life, and to all of my amazing teachers and mentors, in school or otherwise, who helped shape me into who I am, and whose list I dare not spell out because I would inevitably end up missing a shameful number, but to whom all goes the lion's share of credit for any successful accomplishment of mine, such as this thesis, if that is what it is considered.

A huge thank you and admiration goes to my wife Katarína, who selflessly agreed to go on this journey with me, who always stood by me, and who has lately, while I was busy dealing with reluctant formulas and slowly assembling this thesis, been taking care of a much more important thing, which is not a thing at all, a problem that, to use the parlance of complexity theory, is known to be ALLTHETIME-hard, a complexity class in comparison with which QBF's PSPACE pales.

Kurzfassung

In den letzten 20 Jahren waren wir Zeugen des Aufstiegs von extrem effizienten Entscheidungsprozeduren für das aussagenlogische Erfüllbarkeitsproblem (Satisfiability, SAT). Obwohl SAT aufgrund seiner NP-Vollständigkeit theoretisch als quasi unlösbar gilt, können dank dieser sogenannten SAT-Solver Probleme aus Gebieten wie der Automatisierung von elektronischen Designs, der Software- und Hardwareverifikation, oder der Programmsynthese heute in vielen Fällen gelöst werden.

Von dieser Erfolgsgeschichte inspiriert, hat sich die Forschung Formalismen zugewandt, die eine noch größere Bandbreite von Problemen prägnant ausdrücken können. Ein Beispiel für eine solche logische Sprache sind quantifizierte boolesche Formeln (QBF), die den Gegenstand der vorliegenden Dissertation bilden. Quantifizierte boolesche Formeln erweitern aussagenlogische Formeln durch Quantoren, die über die boolesche Domäne rangieren. Das Erfüllbarkeitsproblem von QBF ist PSPACE-vollständig.

Die Schachtelung von Quantoren führt zu Abhängigkeiten zwischen den Variablen einer QBF. In vielen Fällen sind jedoch syntaktisch indizierte Abhängigkeiten semantisch überflüssig. Mit dem Begriff Abhängigkeitsanalyse sind Verfahren bezeichnet, die solche falschen Abhängigkeiten zum Zweck der effizienteren Lösung von QBFs identifizieren.

Die bis vor Kurzem allgemeinste Methode zur Abhängigkeitsanalyse war der Einsatz von sogenannten Abhängigkeitsschemata. Ein Abhängigkeitsschema ist eine Abbildung, die einer Formel eine Menge von Abhängigkeiten zwischen den Variabeln zuordnet.

Die Resultate dieser Arbeit stehen hauptsächlich im Zusammenhang mit suchbasierten QBF-Solvern, die Quantified Conflict-Driven Clause Learning (QCDCL) zur Grundlage haben. QCDCL-Solver können Beweise in einem als Long-Distance Q-Resolution bekannten Beweissystem erzeugen. Die Hauptergebnisse der vorliegenden Arbeit können folgendermaßen zusammengefasst werden:

- Wir geben hinreichende Bedingungen dafür an, wann ein Abhängigkeitsschema im Zusammenspiel mit QCDCL und Long-distance Q-Resolution korrekt ist und Zertifikatextraktion in Polynomialzeit erlaubt.
- Wir stellen mit “Dependency Learning” eine neue Methode zur Abhängigkeitsanalyse vor, die orthogonal zum Einsatz von Abhängigkeitsschemata steht und die das dynamische Erlernen von Abhängigkeiten zur Laufzeit ermöglicht.

- Wir beweisen, dass die Korrekheit von aus Long-Distance Q-Resolution-Beweisen extrahierten Zertifikaten in Polynomialzeit überprüft werden kann.
- Wir berichten von aus der Implementierung eines Solver-Portfolios für QBFs im QCIR-Format gewonnenen Erkenntnissen. Hier zeigt sich, dass die Laufzeit von QBF-Solvern mittels zweier Schlüsseleigenschaften von Formeln mit großer Genaugkeit vorhergesagt werden kann.
- Wir präsentieren eine dynamische Implementierung des stärksten bekannten Abhängigkeitsschemas, in der Dependency Learning mit dem Einsatz von Abhängigkeitsschemata kombiniert wird.
- Wir zeigen, dass die zwei grundlegenden Beweisregeln von Long-Distance Q-Resolution, nämlich universale Reduktion und Merging, inkommensurabel sind: Beweissysteme, die jeweils nur eine dieser Regeln verwenden, sind hinsichtlich ihrer Beweiskomplexität unvergleichbar.

Abstract

The last two decades have seen the rise of extremely efficient solvers for the satisfiability problem of propositional logic (SAT). While SAT is, as an NP-complete problem, considered theoretically intractable, the solvers have become so efficient that solving hard search and optimization problems from areas such as electronic design automation, software and hardware verification, or program synthesis via SAT encodings is now not only a feasible approach, but often the state of the art. Inspired by this success story, researchers started working on formalisms that could succinctly express an even wider range of problems than SAT can. One example, which we study, are quantified Boolean formulas (QBF).

Quantified Boolean formulas generalize propositional logic with quantification, and the satisfiability problem for QBFs is PSPACE-complete. As a consequence of quantifier nesting in QBFs, dependencies between variables arise. In many cases, however, some dependencies that appear to be present syntactically, are spurious semantically. Dependency analysis is the process of identifying spurious dependencies and using that additional information to solve a QBF more efficiently. Prior to this thesis, the most general method for dependency analysis were dependency schemes. A dependency scheme is a mapping which takes a formula and returns a set of dependencies.

The results in this thesis are related mainly to search-based QBF solvers that implement quantified conflict-driven constraint learning (QCDCL). The trace of a run of a QCDCL solver can be interpreted as a proof in long-distance Q-resolution. Our results can be summarized as follows:

- we identify a sufficient condition for when a dependency scheme admits sound use and certificate extraction in QCDCL with long-distance Q-resolution;
- we propose a new method for dependency analysis, called dependency learning, that is orthogonal to dependency schemes, and in which dependencies are computed dynamically on the fly as opposed to statically upfront;
- we show that certificates extracted from long-distance Q-resolution proofs can be verified in polynomial time;
- we report on an implementation of a portfolio for circuit QBFs, and identify a pair of key formula features that are excellent predictors of solver performance;

- we present a dynamic implementation of the strongest known sound dependency scheme and demonstrate that dependency learning can be used together with dependency schemes;
- we prove that two basic components of long-distance Q-resolution, universal reduction and merging, have incomparable strength, i.e., the proof systems without one of the rules are incomparable.

Contents

CHAPTER 1

Introduction

Almost half a century has passed since Stephen Cook [Coo71] and Leonid Levin [Lev73] independently discovered the existence of NP-complete problems, and in doing so laid the foundations for what is today known as *computational complexity theory*. Nowadays, computational complexity theory is an absolutely essential component of theoretical computer science. Of particular interest is the relationship between two complexity classes: P and NP. The class P contains all those problems that are solvable in polynomial time, and hence are considered tractable. The class NP, on the other hand, contains all those problems whose solutions can be verified in polynomial time. While P traditionally embodies problems that can be solved efficiently, NP-complete problems, the hardest of problems in NP, represent inherent intractability. The question whether $P = NP$, which has become known as the "P vs NP" problem and on whose solution a million-dollar bounty has been placed, is perhaps the single most important open problem in computer science. Under the widely-held assumption that $P \neq NP$, NP-complete problems admit no polynomial-time algorithms, and hence are not efficiently solvable, or *tractable*, in that sense.

The most prominent NP-complete problem, and also the one first discovered by Cook and Levin, is no doubt SAT—the satisfiability problem of propositional logic. SAT asks, given a propositional formula, for a satisfying assignment to its variables. As the canonical NP-complete problem, SAT *should* be believed to be intractable, and hence one would expect that no-one is actually solving SAT algorithmically in practice.

Well, that is not quite the case. Starting in the late 1990s [MS96], though building on techniques that go back to even before Cook and Levin [DP60, DLL62], algorithms and solvers have been developed that have scaled up to real-world instances with millions of variables and clauses. SAT technology has been successfully deployed in areas such as formal verification [VWM15], pure mathematics [HKM16, Heu18], and, through its optimization variant MaxSAT, also in optimization settings. When presented with an NP-complete problem, the canonical reaction today is no longer to throw your hands

up in despair, but instead to encode it into SAT and have it solved without thinking twice about it. This is not to say that the process of encoding is simple, but rather that if it is done properly, it is often expected that the SAT solver will solve the instance. The success of SAT solvers has changed the perception of hardness for many—it is not uncommon to consider NP to be the class of problems that are *practically* tractable.

It is worth taking a moment to reflect on *why* SAT solvers have been so successful. The rigorous answer is we do not know—to this day we have no satisfactory explanation for the gap between the theoretical worst-case exponential explosion and the practical real-world near-linear running time. Nevertheless, it is clear that without the aggressive search-space-pruning techniques the solvers would not get far. The single most successful complete (i.e. able to certify both satisfiability and unsatisfiability) algorithm for SAT is *Conflict-Driven Clause Learning (CDCL)* [MLM09]. CDCL combines brute-force search with reasoning about forced assignments and learning of new clauses. A CDCL solver terminates once it finds a satisfying assignment or, if the formula is unsatisfiable, when it learns the empty clause, at which point it has implicitly found a *resolution* proof that shows unsatisfiability. When we speak of SAT solvers in the context of artificial intelligence, it is precisely this learning, which can be seen as the "intelligent" component.

With the frontiers of practical tractability on the move, it is only natural that researchers started looking for other problems, theoretically assumed to be even harder than SAT, which could be attacked with similar methods and success. It should not come as a surprise that that is also how this thesis came to be. We are interested in *quantified Boolean formulas (QBF)*, which are propositional formulas with quantifiers. In SAT, we are looking for a satisfying assignment to the variables of formula, which is the same as quantifying all variables existentially and then asking if the resulting QBF is true. From that perspective, QBFs simply additionally allow universal quantifiers.

1.1 Background on QBF

It turns out that the extra quantifier makes quite a difference. If we allow its unrestricted use, which is what we typically do, the problem of evaluating QBFs is complete for PSPACE—the class of problems decidable in polynomial space. Hence, QBFs can succinctly capture many relevant problems for which only exponentially-sized encodings into SAT are assumed to exist, such as unbounded model checking, planning, or two-player games.

QBF is no longer a new research field. The first theoretical results that continue to influence today's solvers date back to 1995 [KKF95], while the first solvers spawned at the beginning of the millennium [ZM02b, ZM02c]. Since then, many different decision procedures and solvers have been developed [BL10, KSGC10, JKMSC12, RT15, JM15, Ten16, Jan18b, Jan18a]. Most of them take one of two approaches: they either use (careful) expansion/abstraction into SAT; or they build upon a generalization of CDCL for QBF

called *Quantified Conflict-Driven Constraint Learning (QCDCL)*. [1] In this thesis we focus on QCDCL and theoretical questions that arise from it.

A salient feature of QBFs are dependencies between variables. Consider this simple (in some sense the simplest non-trivial possible) formula:

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

This formula is *closed* (every variable is bound by a quantifier) and in *prenex normal form*—the quantifiers are all stacked up in the front—which is the kind of formulas that we will work with. In fact, it is also in *prenex conjunctive normal form (PCNF)*, which means that *the matrix*—the quantifier-free part— is a conjunction of clauses. A clause is a finite disjunction of literals, which in turn are either variables, like x, or negations thereof, like $\neg x$.

If x is set to 0, then we can set y to 0 as well, and the formula is satisfied. Similarly, if x is set to 1, then we can set y to 1, and again, the formula is satisfied. Therefore, it is true that for all x there is a y such that the matrix is satisfied, and hence the formula is true. But, the choice of y *depends* on x. We must set y to the same value as x, and hence no single value for y will suffice. This is how dependencies can be understood intuitively.

Dependencies are tightly related to another important notion, namely that of *models* and *countermodels*. In the previous example, we had to set y to the same value as x in order to satisfy the matrix of the formula. The function $f_y(x) = x$ that performs this is called a model of the formula. A countermodel is an analogous function for a universal variable in case the formula is false. Models and countermodels are also known as *winning strategies* or simply *strategies*. This comes from looking at QBFs as two-player games where the existential player attempts to satisfy the matrix, the universal player attempts to falsify it, and they take turns in the order of the *quantifier prefix*. A model is then exactly a winning strategy for the existential player, and similarly a countermodel is a winning strategy for the universal player.

All dependencies as given by the quantifier prefix are not always necessary. Consider a formula built from two variable-disjoint copies of the previous example:

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

Here y' really only depends on x', and not on x or y—the latter dependencies are spurious. However, in order to solve this formula with QCDCL, we must assign variables in the order of the quantifier prefix. So, for every assignment to x and y we must check from scratch that for all x' there exists a y' to satisfy the matrix, even though the y' only depends on x' and not on x and y.

[1] QCDCL is also known as QDPLL, harking back to CDCL's precursor DPLL, and the acronym QCDCL can also stand for Quantified Conflict-Driven/Directed Clause Learning, and in that case there is also a counterpart called Solution-Driven Cube Learning (SDCL). In order to avoid this naming mishmash we will stick to QCDCL and mean all of these ingredients.

Even worse, consider the formula

$$\mathcal{Q}_\Phi \mathcal{Q}_\Psi .\ \varphi \wedge \psi,$$

where φ is a hard unsatisfiable propositional formula with all its variables quantified existentially in $\mathcal{Q}_\Phi$ and $\Psi = \mathcal{Q}_\Psi.\psi$ is an easy false QBF that is variable-disjoint from Φ. The conjunction is clearly false, and it could be quickly solved using the easy part Ψ, but QCDCL has to respect the prefix, which says that the variables of Φ must go first. Hence, even if the overall formula is objectively easy, it will be difficult for QCDCL because it has to focus on the hard part.

Examples like the ones above inspired researchers to analyze dependencies in greater detail. Of course, in general all the dependencies prescribed by the quantifier prefix are necessary, and there is nothing that can be done about that. In practice, however, formulas can contain *spurious dependencies*, and if a solver can detect that fact, it can use it to its advantage. If a solver (or a preprocessor for that matter) manipulates or reasons about the quantifier prefix in any non-trivial way, we say that it performs *dependency analysis*.

The systematic study of dependencies in QBF started with the work of Samer and Szeider [SS09] on *dependency schemes*. A dependency scheme is a mapping that takes a formula and returns a set of pairs of its variables—the actual dependencies. Dependency schemes were originally introduced as a means to perform reorderings of the quantifier prefix in order to find smaller sets of variables whose every assignment simplifies the formula to a tractable form. Not long afterwards, however, Biere and Lonsing [BL09,BL10] showed how dependency schemes can be used to enhance a QCDCL solver. Their solver DepQBF uses the *standard dependency scheme* to relax the ordering imposed by the prefix.

Following the successful implementation of a dependency scheme in a QBF solver, the doors were open for research to take dependency analysis for QBF solving to the next level. Two of the most immediate problems were to ensure the soundness of QCDCL with a dependency scheme, and to develop and implement stronger dependency schemes which can detect more spurious dependencies.

To tackle the first challenge, researchers turned to proof theory for QBF. It was already known that constraints learned by QCDCL can be modeled using a proof system called Q-resolution [KKF95], a generalization of propositional resolution. Implementing QCDCL learning with Q-resolution, however, is not completely straightforward and if not done carefully may even lead to an exponential running time. While there is a somewhat more complicated algorithm that can learn constraints in linear time with Q-resolution [Gel12], in this thesis we focus on a different approach. Implemented already in the first QCDCL solvers, but properly analyzed only more recently [ELW13], *long-distance Q-resolution* generalizes Q-resolution by allowing tautological resolvents in some cases.

Suppose the clauses $C_1 \vee x$ and $C_2 \vee \neg x$ are to be resolved. Propositional resolution requires that there is no *clashing variable*, i.e., a variable y with $y \in C_1$ and $\neg y \in C_2$ or

$\neg y \in C_1$ and $y \in C_2$. This is also equivalent to requiring that the *resolvent* $C_1 \vee C_2$ is non-tautological. Q-resolution additionally stipulates that the *pivot* x is an existential variable. In long-distance Q-resolution, there may be a clashing universal variable u given that it is right of x in the quantifier prefix.

Long-distance Q-resolution has a number of advantages over Q-resolution: it is arguably easier to implement in QCDCL; and it allows for exponentially shorter proofs of some formulas. On the other hand, like Q-resolution, long-distance Q-resolution enjoys linear-time strategy extraction, i.e., there is an algorithm that, given a long-distance Q-resolution proof/refutation, computes a winning strategy in linear time. This is an important property of proof systems for applications, where the simple answer of whether a formula is true of false is often not sufficient.

In order to account for the use of dependency schemes in QCDCL, Slivovsky and Szeider defined the proof system Q(D)-resolution, a parameterized version of Q-resolution [SS16b]. Q-resolution has two rules: the aforementioned resolution on existential pivots; and *universal reduction*. Suppose $C \vee u$ is a clause where every variable $x \in C$ is left of u in the prefix. If $C \vee u$ does not become satisfied by the variables from C, then the universal player will surely win by falsifying u. Hence, already C must be satisfied and u can be *reduced* from $C \vee u$.

In Q(D)-resolution, resolution remains the same, but universal reduction receives a boost—a universal literal u may be reduced from $C \vee u$ as long as nothing in C depends on u according to the dependency scheme D. This is in line with the previous intuition: we may not play C-variables depending on the value of u, so we must satisfy C regardless of u. Slivovsky and Szeider proved that Q(D)-resolution is sound when parameterized by the *reflexive resolution-path dependency scheme*, also known as $\mathrm{D}^{\mathrm{rrs}}$ [SS16b]. This is the strongest known soundness result for a dependency scheme in Q-resolution, and it also implies that DepQBF is sound.

At the same time, $\mathrm{D}^{\mathrm{rrs}}$ happens to be the answer to the second research problem stated above—to develop new, stronger dependency schemes. However, even though stronger dependency schemes had been developed, there had been no efficient implementations after DepQBF.

It was roughly at this moment in the history of QBF that the author of this thesis joined the field. Let us summarize some of the main open problems as we saw them at the time.

- The soundness of Q(D)-resolution had been established for $\mathrm{D}^{\mathrm{rrs}}$, but what about efficient strategy extraction?
- Could long-distance Q-resolution be combined with dependency schemes?
- What is the relative proof complexity of the proof systems with dependency schemes?
- How to efficiently implement the reflexive resolution-path dependency scheme?
- What about some other, possibly stronger techniques for dependency analysis?

In the following chapters, we will give answers to these and other questions.

1.2 Contributions

In Chapter 3 we define LDQ(D)-resolution, a combination of long-distance Q-resolution with dependency schemes. We define a sufficient condition, called *normality*, on the dependency scheme D, under which LDQ(D)-resolution is both sound and admits polynomial-time strategy extraction. We also show that D^{rrs} is normal.

In Chapter 4, we introduce dependency learning for QCDCL, a novel technique for dependency analysis. QCDCL with dependency learning starts with an empty dependency relation D, assuming every pair of variables is independent. The solver proceeds just like DepQBF, making assignments and inferences according to its dependency relation, but when it comes to constraint learning, it forgets about the dependencies and attempts to learn a constraint like an ordinary QCDCL solver, using long-distance Q-resolution. This may not always be possible because of the way the solver made previous assignments. If the solver is unable to derive a learned constraint, it identifies a missing dependency, adds it to D, backtracks, and continues.

Following up, in Chapter 5, we show how to harness the architecture of dependency learning in order to implement the reflexive resolution-path dependency scheme in a QCDCL solver. The core idea is to compute certain dependencies on demand during dependency conflicts. This way only the most relevant dependencies—the ones that are to be learned—are calculated, which prevents the quadratic blow-up resulting from computing the entire dependency relation upfront. Thus, we also show that the two techniques for dependency analysis can be combined.

Chapters 6 and 7 take a detour away from the world of dependencies. In Chapter 6, we show that QBF *certificates*—which is just another name for strategies—computed from long-distance Q-resolution (and hence also Q-resolution) proofs can be validated in polynomial time. In many applications where the strategy is required it is often desirable that the strategy be verified for correctness. Prior to this result, this step had required a call to a SAT solver.

Chapter 7 also stays on the topic of long-distance Q-resolution. We show how semantic (strategy-based) lower-bound techniques introduced for Q-resolution can be used to obtain lower bounds for fragments of long-distance Q-resolution. In particular, we show that two very natural fragments of long-distance Q-resolution—Q-resolution and long-distance Q-resolution without universal reduction—are incomparable, i.e., neither polynomially simulates the other. This means that the two core abilities of long-distance Q-resolution, namely merging and reduction, are of incomparable power.

Chapter 8 is a report on our work on constructing an algorithm selector for a solver portfolio for QBFs whose matrix is a general circuit. A portfolio is a collection of solvers, and an algorithm selector is a procedure that, for a given formula, selects the most promising solver based on the formula's features. We have found that only two features

suffice to predict solver performance very accurately, and to construct a very good algorithm selector.

1.3 List of Papers and Software

While every chapter ends with a note of the papers containing the research presented in that chapter, we also list all the papers used in this thesis here for reference.

- Peitl, Slivovsky, and Szeider: Long-distance Q-resolution with Dependency Schemes, Journal of Automated Reasoning, 2019 [PSS19b, PSS16]
- Peitl, Slivovsky, and Szeider: Dependency Learning for QBF, Journal of Artificial Intelligence Research, 2019 [PSS19a, PSS17]
- Peitl, Slivovsky, and Szeider: Polynomial-Time Validation of QCDCL Certificates, SAT 2018—the 21st International Conference on the Theory and Practice of Satisfiability Testing [PSS18]
- Hoos, Peitl, Slivovsky, and Szeider: Portfolio-Based Algorithm Selection for QBF, CP 2018—the 24th International Conference on Principles and Practice of Constraint Programming [HPSS18]
- Peitl, Slivovsky, and Szeider: Combining Resolution-Path Dependencies with Dependency Learning, SAT 2019—the 22nd International Conference on the Theory and Practice of Satisfiability Testing [PSS19c]
- Peitl, Slivovsky, and Szeider: Proof Complexity of Fragments of Long-distance Q-resolution, SAT 2019—the 22nd International Conference on the Theory and Practice of Satisfiability Testing [PSS19d]

We also include links to software created as part of our work.

- The QCDCL-based dependency-learning QBF solver Qute: `https://github.com/perebor/qute`
- The certificate extractor and validation proof generator `qrp2rup`: `https://www.ac.tuwien.ac.at/research/certificates`

CHAPTER 2

Preliminaries

The central object studied in this thesis are *quantified Boolean formulas* (QBF). In this chapter we introduce the notation and the basic notions associated with propositional and quantified Boolean logic, as well as some QBF proof systems that we will study, and their relationship to QBF solving.

2.1 Propositional Logic

We assume that a countably infinite set containing (propositional) *variables* is at our disposal. A *literal* is a variable x or a negated variable $\neg x$. If x is a variable, we write $\overline{x} = \neg x$ and $\overline{\neg x} = x$, and let $var(x) = var(\neg x) = x$. We sometimes call literals x and $\neg x$ the positive and negative *polarity* of the variable x respectively. If S is a set of literals, we write $\overline{S}$ for the set $\{\,\overline{\ell} : \ell \in S\,\}$ and let $var(S) = \{\,var(\ell) : \ell \in S\,\}$.

An *(truth) assignment* to a set X of variables is a mapping $\tau : X \to \{0,1\}$. We also refer to the values 0 and 1 as *false* and *true*, respectively. We write $[X]$ for the set of truth assignments to X, and extend $\tau : X \to \{0,1\}$ to literals by letting $\tau(\neg x) = 1 - \tau(x)$ for $x \in X$. An assignment $\sigma : X' \to \{0,1\}$ is an *extension* of the assignment $\tau : X \to \{0,1\}$ if $X \subseteq X'$ and $\forall x \in X$ we have $\sigma(x) = \tau(x)$. The *restriction* of an assignment $\tau : X \to \{0,1\}$ to a set $Y \subseteq X$ of variables is the assignment $\tau|_Y : Y \to \{0,1\}$ such that τ is an extension of $\tau|_Y$. An assignment $\tau : X \to \{0,1\}$ can also be represented as the set $\tau^{-1}(1) = \{\,x \in X : \tau(x) = 1\,\} \cup \{\,\neg x \in \overline{X} : \tau(x) = 0\,\}$ of literals it makes true, or even as a sequence of those literals if there is a particular order on the variables in X. In particular, if S is a set of literals such that no two literals in S share a variable, then S also represents the assignment that sets all literals of S to 1, and $\overline{S}$ represents the assignment that sets all literals of S to 0.

We consider Boolean *circuits* over $\{\neg, \wedge, \vee, 0, 1\}$ and write $var(\varphi)$ for the set of variables occurring in a circuit φ. If φ is a circuit and $\tau : X \to \{0,1\}$ an assignment, $\varphi[\tau]$ denotes

the circuit obtained by replacing each variable $x \in X \cap var(\varphi)$ by $\tau(x)$ and propagating constants. An assignment $\tau : X \to \{0,1\}$ is a *total (full) assignment* to (the variables of) a circuit φ if $var(\varphi) \subseteq X$, otherwise it is a *partial assignment*. A circuit φ is *satisfiable* if there is an assignment τ such that $\varphi[\tau] = 1$, otherwise it is *unsatisfiable*.

A *clause* (*term*) is a circuit consisting of a disjunction (conjunction) of literals. We write $\bot$ for the empty clause and $\top$ for the empty term. We call a clause *tautological* (and a term *contradictory*) if it contains the same variable negated as well as unnegated. A *CNF formula* (*DNF formula*) is a circuit consisting of a conjunction (disjunction) of non-tautological clauses (non-contradictory terms). Whenever convenient, we treat clauses and terms as sets of literals, and CNF and DNF formulas as sets of sets of literals. For a CNF (DNF) formula φ, we define $\overline{\varphi}$ to be the DNF (CNF) formula $\{\, \overline{S} : S \in \varphi \,\}$.

While restricting clauses, terms, and CNF and DNF formulas by assignments is defined the same way as for general circuits, it will be an operation we will use extensively, and so we detail it here. Let $\tau : X \to \{0,1\}$ be a truth assignment. The restriction $S[\tau]$ of a clause (term) S by τ is defined as follows: if there is a literal $\ell \in S \cap (X \cup \overline{X})$ such that $\tau(\ell) = 1$ ($\tau(\ell) = 0$) then $S[\tau] = 1$ ($S[\tau] = 0$). Otherwise, $S[\tau] = S \setminus (X \cup \overline{X})$. The restriction $\varphi[\tau]$ of a CNF formula φ by the assignment τ is defined $\varphi[\tau] = \{\, C[\tau] : C \in \varphi, C[\tau] \neq 1 \,\}$, and similarly, the restriction $\varphi[\tau]$ of a DNF formula φ by the assignment τ is defined $\varphi[\tau] = \{\, T[\tau] : T \in \varphi, T[\tau] \neq 0 \,\}$. Here, we define the empty CNF formula $\textsc{cnf}(\emptyset)$ to be equivalent to 1, and $\textsc{dnf}(\emptyset)$ to equal 0.

A *unit clause* is a clause containing a single literal. A CNF formula ψ is derived from a CNF formula φ by the *unit clause rule* if (ℓ) is a unit clause of φ and $\psi = \varphi[\ell \mapsto 1]$. *Unit propagation* in a CNF formula consists of repeated application of the unit clause rule. Unit propagation is said to *derive* the literal ℓ in a CNF formula φ if a CNF formula ψ with $(\ell) \in \psi$ can be derived from φ by unit propagation. We say that unit propagation causes a *conflict* if $\bot$ can be derived by unit propagation. If unit propagation does not cause a conflict the set of literals that can be derived by unit propagation induces an assignment. The *closure* of an assignment τ with respect to unit propagation (in φ) is τ combined with the set of literals derivable by unit propagation in $\varphi[\tau]$.

A clause C has the *reverse unit propagation* (RUP) property with respect to a CNF formula φ if unit propagation in $\varphi[\overline{C}]$ causes a conflict. If C is RUP with respect to φ, then φ and $\varphi \wedge C$ are *equisatisfiable*, i.e. they are either both satisfiable, or both unsatisfiable. A *RUP proof of unsatisfiability* of a CNF formula φ is a sequence $C_1, \ldots, C_m$ of clauses such that $C_m = \bot$ and each clause C_i has the RUP property with respect to $\varphi \cup \{C_1, \ldots, C_{i-1}\}$, for $1 \leq i \leq m$.

The problem of deciding the satisfiablity of a given Boolean circuit (SAT) is NP-complete [Coo71]. It is well known that any circuit can be transformed into an equisatisfiable CNF formula of size linear in the size of the circuit [Tse68], and hence also deciding satisfiability of CNF formulas is NP-complete.

2.2 Quantified Boolean Formulas

We consider quantified Boolean formulas that are *closed*, i.e. with no free variables, and in *prenex normal form*. Therefore, a QBF is denoted by $\Phi = \mathcal{Q}.\varphi$, where φ is a circuit and $\mathcal{Q} = Q_1 x_1 \dots Q_n x_n$ is a sequence such that $Q_i \in \{\forall, \exists\}$ and x_i is a variable. The fact that Φ is closed means that $var(\varphi) \subseteq \{x_1, \dots, x_n\}$, and the prenex normal form refers to φ being quantifier-free. Hence we define $var(\Phi)$ as $var(\varphi)$ and for $x_i \in var(\Phi)$, we define the quantifier type $q(x_i) = Q_i$. We say that x is *existential* if $q(x) = \exists$ and *universal* if $q(x) = \forall$. The set of existential (universal) variables occurring in Φ is denoted $var_\exists(\Phi)$ ($var_\forall(\Phi)$). We call φ the *matrix* of Φ and $\mathcal{Q}$ the *(quantifier) prefix* of Φ. The quantifier prefix $\mathcal{Q}$ gives rise to the strict linear order $<_\Phi$ on $var(\Phi)$ defined as $x_i <_\Phi x_j \iff i < j$. We extend $<_\Phi$ to a strict partial order on literals by putting $\ell <_\Phi \ell' \iff var(\ell) <_\Phi var(\ell')$ and drop the subscript whenever Φ is understood. For $x \in var(\Phi)$ we let $R_\Phi(x) = \{\, y \in var(\Phi) : x <_\Phi y \,\}$ and $L_\Phi(x) = \{\, y \in var(\Phi) : y <_\Phi x \,\}$ denote the sets of variables *to the right* and *to the left* of x in Φ, respectively.

If τ is an assignment, then $\Phi[\tau]$ denotes the QBF $\mathcal{Q}'.\varphi[\tau]$, where $\mathcal{Q}'$ is the quantifier prefix obtained from $\mathcal{Q}$ by deleting variables that do not occur in $\varphi[\tau]$. The semantics of a PCNF formula Φ is defined as follows. If Φ does not contain any variables, then Φ is true if its matrix is empty and false if its matrix contains the empty clause $\bot$. Otherwise, let $\Phi = Qx\mathcal{Q}.\varphi$. If $Q = \exists$ then Φ is true if $\Phi[x \mapsto 0]$ is true or $\Phi[x \mapsto 1]$ is true, and if $Q = \forall$ then Φ is true if both $\Phi[x \mapsto 0]$ and $\Phi[x \mapsto 1]$ are true.

We define an equivalence relation $\cong_\Phi$ on $var(\Phi)$ by putting

$$x \cong_\Phi y \iff q(x) = q(y) \wedge \\ \forall z \in var(\Phi)\ (x <_\Phi z \wedge z <_\Phi y) \vee (y <_\Phi z \wedge z <_\Phi x) \implies q(z) = q(x),$$

and refer to the equivalence classes as *(quantifier) blocks*. Because quantifier blocks are intervals (in fact maximal intervals of variables of the same quantifier type) in the strict linear order $<_\Phi$, the order carries over to a strict linear order $<_{\cong_\Phi}$ by putting $[x]_{\cong_\Phi} <_{\cong_\Phi} [y]_{\cong_\Phi} \iff x <_\Phi y$. That, in turn, gives rise to a strict partial order on $var(\Phi)$ defined by $x \prec_\Phi y \iff [x]_{\cong_\Phi} <_{\cong_\Phi} [y]_{\cong_\Phi}$. We similarly extend $\prec_\Phi$ to a strict partial order on literals. We let $\mathcal{B}(\Phi)$ be the number of quantifier blocks of Φ and number the blocks in the order of $<_{\cong_\Phi}$ as $X_1, \dots, X_{\mathcal{B}(\Phi)}$, i.e. $X_i <_{\cong_\Phi} X_j \iff i < j$. We define the *(quantifier) depth* $\delta(x)$ of a variable $x \in X_i$ as $\delta(x) = i$.

If the quantifier type of each variable is known, the quantifier prefix is uniquely determined by the order $<_\Phi$, which is a linear extension of $\prec_\Phi$. However, any linear extension of $\prec_\Phi$ leads to a formula with the same truth value. In other words, since the order of variables within a quantifier block is semantically irrelevant, we will sometimes write QBFs in the form $\Phi = Q_1 X_1, \dots, Q_n X_k.\, \varphi$ with $Q_i \neq Q_{i+1}$, where each X_i is a quantifier block. In this notation, we have that $x \prec_\Phi y \iff x \in X_i, y \in X_j, i < j$.

Apart from $\prec_\Phi$, we will also make use of its sub-relation $\prec^q_\Phi$ defined as

$$x \prec^q_\Phi y \iff x \prec_\Phi y \wedge q(x) \neq q(y).$$

Let $\Phi = \mathcal{Q}.\varphi$ be a QBF. A *model* of Φ is an indexed family $\{f_e\}_{e \in var_\exists(\Phi)}$ of functions $f_e : [L_\Phi(e)] \to \{0,1\}$ such that $\varphi[\tau] = 1$ for every assignment $\tau : var(\Phi) \to \{0,1\}$ satisfying $\tau(e) = f_e(\tau|_{L_\Phi(e)})$ for $e \in var_\forall(\Phi)$. A *countermodel* of Φ is an indexed family $\{f_u\}_{u \in var_\forall(\Phi)}$ of functions $f_u : [L_\Phi(u)] \to \{0,1\}$ such that $\varphi[\tau] = 0$ for every assignment $\tau : var(\Phi) \to \{0,1\}$ satisfying $\tau(u) = f_u(\tau|_{L_\Phi(u)})$ for $u \in var_\forall(\Phi)$.

Theorem 1 (Folklore)**.** *A QBF is true if, and only if, it has a model, and false if, and only if, it has a countermodel.*

Models and countermodels are tightly related to a game-theoretic perspective of QBFs. A given QBF can be viewed as a game between two players—the *existential player* and the *universal player*, sometimes abbreviated simply as *Existential* and *Universal*. The players take turns assigning variables in the order of the quantifier prefix. Existential wins if at the end the matrix is satisfied, while Universal wins if it is falsified. In this game-theoretic terminology, a model (countermodel) is a winning *strategy* for Existential (Universal).

In most cases, we deal with QBFs in *PCNF*, i.e. when the matrix φ is a CNF. Dually, a *PDNF* formula is one where the matrix is in DNF. The *size* of a PCNF (PDNF) formula $\Phi = \mathcal{Q}.\varphi$ is defined as $|\Phi| = \sum_{S \in \varphi} |S|$. For a PCNF (PDNF) formula $\Phi = \mathcal{Q}.\varphi$, we define $\overline{\Phi} = \overline{\mathcal{Q}}.\overline{\varphi}$, where $\overline{\mathcal{Q}}$ is the quantifier prefix that results from $\mathcal{Q}$ by changing the quantifier type of each variable. Naturally, we have that Φ is true if, and only if, $\overline{\Phi}$ is false. For technical reasons we require that PCNF (PDNF) formulas contain no tautological clauses (contradictory terms). This is a safe assumption, since tautological clauses and contradictory terms can simply be removed from any matrix without a change of truth value.

The problem of deciding whether a given QBF is true or false is PSPACE-complete [SM73]. Using Tseitin conversion, one can transform the matrix of a QBF into CNF, and quantifying the auxiliary variables existentially and rightmost in the prefix results in a truth-value equivalent formula. Similarly, one can transform the matrix into a DNF and quantify the auxiliary variables universally and rightmost, and obtain an equivalent formula. Hence, deciding the truth of both PCNF and PDNF is also PSPACE-complete.

2.3 Q-resolution and other Proof Systems

An important part of the theoretical study of quantified Boolean formulas consists in the study of their *proof systems*. While a wide range of proof systems that model different solving paradigms have been proposed, in this work we focus on *Q-resolution* [KKF95], a generalization of propositional resolution to QBF, and systems that are derived from it, most prominently *long-distance Q-resolution* [ZM02b, BJ12]. The rules of Q-resolution and long-distance Q-resolution are given in Figures 2.1 and 2.2, respectively. These proof systems, as well as others that will be introduced later, operate on PCNF formulas.

A (long-distance) Q-resolution *derivation* from a PCNF formula Φ in its most basic form is a sequence of clauses $\mathcal{P} = C_1, \ldots, C_m$ such that each clause either appears in the matrix of Φ, or can be derived from clauses appearing earlier in the sequence using (long-distance) resolution or universal reduction. The *size* of $\mathcal{P}$ is defined as $|\mathcal{P}| := \sum_{i=1}^{m} |C_i|$. A derivation of the empty clause is called a *refutation*, and one can show that a PCNF formula is false, if, and only if, it has a (long-distance) Q-resolution refutation [KKF95, BJ12]. Hence, both Q-resolution and long-distance Q-resolution are sound and (refutationally) complete proof systems for PCNF formulas.

Q-resolution and long-distance Q-resolution can only be used to show that a given PCNF is false. In order to be able to prove a formula true, dual proof systems called *(long-distance) Q-consensus*, were introduced. They operate on terms instead of clauses, and are obtained by swapping the roles of existential and universal variables (the analogue of universal reduction for terms is called *existential reduction*).

$$\frac{C_1 \vee e \qquad \neg e \vee C_2}{C_1 \vee C_2}\ \text{(resolution)}$$

The *resolution* rule allows the derivation of $C_1 \vee C_2$ from clauses $C_1 \vee e$ and $\neg e \vee C_2$, provided that the *pivot* variable e is existential and that $C_1 \vee C_2$ is not a tautology. The clause $C_1 \vee C_2$ is called the resolvent of $C_1 \vee e$ and $\neg e \vee C_2$.

$$\frac{C}{C \setminus \{\ell_u\}}\ \text{(universal reduction)}$$

The *universal reduction* rule admits the deletion of a universal literal ℓ_u from a clause C under the condition that $e \prec^q var(\ell_u)$ for each existential variable $e \in var(C)$.

Figure 2.1: Q-resolution.

The case of (long-distance) Q-consensus operating on PDNF formulas is completely dual, however, the case of proving PCNF formulas to be true is slightly more involved. In particular, since (long-distance) Q-consensus operates on terms and the input PCNF formula only contains clauses, some *initial terms* need to be generated. There are two ways how this is done in practice: either a DNF representation of the matrix is obtained at the beginning using Tseitin conversion; or initial terms are generated on the fly by *model generation* [GNT06] from (full) assignments that satisfy the (CNF) matrix. [1] Hence, a (long-distance) Q-consensus derivation from a PCNF formula Φ is a sequence $T_1, \ldots, T_m$ of terms such that each term is either an initial term (i.e. it is either in the DNF representation of Φ, or it is a *generated model*), or can be derived from previous terms by consensus or existential reduction. A PCNF formula is true, if, and only if, the

[1] Model generation was shown to require an exponential number of initial terms in the worst case [JM17].

$$\frac{C_1 \vee e \qquad \neg e \vee C_2}{C_1 \vee C_2} \text{ (long-distance resolution)}$$

The *resolution* rule allows the derivation of $C_1 \vee C_2$ from clauses $C_1 \vee e$ and $\neg e \vee C_2$, provided that the *pivot* variable e is existential and that $e \prec^q var(\ell_u)$ for each universal literal $\ell_u \in C_1$ such that $\overline{\ell_u} \in C_2$. The clause $C_1 \vee C_2$ is called the resolvent of $C_1 \vee e$ and $\neg e \vee C_2$.

$$\frac{C}{C \setminus \{u, \neg u\}} \text{ (universal reduction)}$$

The *universal reduction* rule admits the deletion of a universal variable u from a clause C under the condition that $e \prec^q u$ for each existential variable $e \in var(C)$.

Figure 2.2: Long-distance Q-resolution.

empty term can be derived from a DNF representation of its matrix by (long-distance) Q-consensus. Hence, both Q-consensus and long-distance Q-consensus are sound and complete proof systems for PCNF and PDNF formulas.

Derivations as defined here will be sufficient for Chapter 6. However, in certain other places, we will need a more detailed view of derivations in proof systems that generalize Q-resolution. Therefore, a definition of derivations as certain directed acyclic graphs (*proof DAGs*) labeled with clauses will be presented at the beginning of Chapter 3. These two definitions are not in conflict—a derivation as a sequence of clauses can be seen as a topological ordering of a derivation represented as a proof DAG. By looking at a derivation as a sequence we simply abstract from the details of the immediate relationships between the clauses.

From the work of Balabanov and Jiang, and subsequently Balabanov et al. [BJ12,BJJW15] we know that Q-resolution and long-distance Q-resolution refutations can be used to compute winning strategies (countermodels, but also models in the case of Q-consensus and long-distance Q-consensus) in linear time in the size of the refutation. We call this that (long-distance) Q-resolution admits linear-time *strategy extraction*.

2.4 QBF Solving and QCDCL

Most of the work in this thesis is concerned with—or at least motivated by—improving QBF solving (deciding the truth value of a given closed prenex QBF). In the closely related field of SAT solving (checking the satisfiability of propositional formulas), a single solving paradigm has been established as the state of the art, namely *Conflict-Driven Clause Learning (CDCL)* [MS96, MLM09]. CDCL is a family of algorithms whose central

idea is to exhaustively enumerate the space of possible variable assignments, learning implied clauses along the way to prune the search space. Informally, CDCL repeatedly branches and applies unit propagation until a conflict is reached, at which point it identifies a reason for the conflict and backtracks appropriately, possibly undoing more than just the last decision before the conflict.

In contrast, in QBF there are at least two competing paradigms: expansion/instantiation-based solving; and *Quantified Conflict-Driven Constraint Learning (QCDCL)*, a generalization of CDCL. While both of these paradigms have their advantages and disadvantages—as is witnessed for instance by separations between their underlying proof systems—in this work we focus solely on QCDCL and the theory that surrounds it.

Starting from an input PCNF formula, QCDCL generates ("learns") *constraints*—clauses and terms—until it produces an empty constraint. Every clause learned by QCDCL can be derived from the input formula by (long-distance) Q-resolution, and every term learned by QCDCL can be derived by (long-distance) Q-consensus [GNT06, ELW13]. This certifies the correctness of the algorithm—if the solver learns the empty term, the formula must be true, and if it learns the empty clause, the formula must be false.

One can think of QCDCL solving as proceeding in rounds. Along with a set of clauses and terms, the solver maintains an assignment σ (the *trail*). During each round, this assignment is extended by quantified Boolean constraint propagation (QBCP) and—possibly—branching.

Quantified Boolean constraint propagation consists of exhaustive application of universal and existential reduction in combination with unit assignments.[2] More specifically, QBCP reports a clause C as falsified if $C[\sigma] \neq 1$ and universal reduction can be applied to $C[\sigma]$ to obtain the empty clause. Dually, a term T is considered satisfied if $T[\sigma] \neq 0$ and $T[\sigma]$ can be reduced to the empty term. A clause C is *unit* under σ if $C[\sigma] \neq 1$ and universal reduction yields the clause $C' = (\ell)$, for some existential literal ℓ such that $var(\ell)$ is unassigned. Dually, a term T is *unit* under σ if $T[\sigma] \neq 0$ and existential reduction can be applied to obtain a term $T' = (\ell)$ containing a single universal literal ℓ. If $C = (\ell)$ is a unit clause, then the assignment σ has to be extended by ℓ in order not to falsify C, and if $T = (\ell)$ is a unit term, then σ has to be extended by $\bar{\ell}$ in order not to satisfy T. If several clauses or terms are unit under σ, QBCP nondeterministically picks one and extends the assignment accordingly. This is repeated until a constraint is empty or no unit constraints remain.

If QBCP does not lead to an empty constraint, the assignment σ is extended by *branching*. That is, the solver chooses an unassigned variable x such that every variable y with $y \prec^q x$ is assigned, and extends the assignment σ by x or $\neg x$.

The resulting assignment can be partitioned into so-called *decision levels*. The decision level of a literal ℓ in σ is the number of literals in σ that were assigned by branching and not later than ℓ. The decision level of an assignment σ is the decision level of its last

[2] We do not consider the pure literal rule as part of QBCP.

assigned literal. Note that each decision level greater than 0 can be associated with a unique variable assigned by branching.

Eventually, the assignment maintained by QCDCL must falsify a clause or satisfy a term. [3] When this happens (this is called a *conflict*), the solver proceeds to *conflict analysis* to derive a learned constraint C. Initially, C is the falsified clause (satisfied term), called the *conflict clause (term)*. The solver finds the existential (universal) literal in C that was assigned last by QBCP, and the antecedent clause (term) R responsible for this assignment. A new constraint is derived by resolving C and R and applying universal (existential) reduction. This is done repeatedly until the resulting constraint C is either empty or *asserting*. A clause (term) S is asserting if there is a unique existential (universal) literal $\ell \in S$ with maximum decision level among literals in S, its decision level is greater than 0, the corresponding decision variable is existential (universal), and every universal (existential) variable $y \in var(S)$ such that $y \prec^q var(\ell)$ is assigned at a lower decision level. Such a literal is called an *asserting literal*. Once an asserting constraint has been found, it is added to the solver's set of constraints. Finally, QCDCL *backtracks*, undoing variable assignments until it reaches a decision level computed from the learned constraint. It is not hard to see that an asserting constraint becomes unit after this backtracking, with the asserting literal being the unit literal.

Example 1. We present a run of QCDCL on a simple PCNF formula Φ shown below.

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

We use the notation $x \stackrel{d}{=} c$ to denote that variable x that is assigned value c by decision. If x is assigned c by unit propagation, we write $x = c$. A possible run of QCDCL on Φ looks as follows. The formula does not contain any unit clauses so QCDCL has to make a decision, and the variable x is the only variable eligible for a decision. Setting $x \stackrel{d}{=} c$ satisfies the first clause and turns the third clause into the unit clause (z). Setting $z = 1$ turns the second clause into the unit clause (y), and setting $y = 1$ satisfies the matrix. This allows us to derive $(x \wedge z \wedge y)$ as an initial term using the model generation rule. By applying existential reduction we can derive the unit term (x) as a learned term. QCDCL then backtracks to decision level 0 and assigns $x = 0$. This turns the first clause into the unit clause $(\neg y)$ and results in the assignment $y = 0$. Now the second clause simplifies to $(\neg z)$ and the algorithm assigns $z = 0$. The resulting assignment satisfies the matrix and we derive $(\neg x \wedge \neg y \wedge \neg z)$ as an initial term. Existential reduction leads to the unit term $(\neg x)$ which is resolved with the term (x) responsible for assigning $x = 0$. The resolvent is the empty term and QCDCL returns TRUE.

[3] If a PCNF is being solved and we do not have a DNF representation of the matrix, it can happen that the assignment satisfies all clauses without satisfying a term. In that case, we generate a model, which we add to the database of learned terms, and which is satisfied by the current assignment.

2.5 Dependency Schemes

In QCDCL, the quantifier prefix imposes constraints on the order of variable assignments: a variable may be assigned only if it occurs in the leftmost quantifier block with unassigned variables; else the method is not sound. However, this is often more restrictive than necessary. For instance, variables from variable-disjoint subformulas may be assigned in any order. Intuitively, a variable can be assigned as long as it is guaranteed *not to depend* on any unassigned variable. This is the intuition underlying a generalization of QCDCL implemented in the solver DepQBF [BL10, Lon12]. DepQBF uses a *dependency scheme* [SS09] to compute an overapproximation of variable dependencies.

In this section we focus on what dependency schemes *are*. We will deal with the issue of how to *use* dependency schemes in a QCDCL solver in Section 2.6. A dependency scheme is a mapping that maps a PCNF formula Φ to a strict partial order of its variables that refines the order $\prec_\Phi$ on variables in the quantifier prefix.[4]

Definition 1 (Dependency Scheme)**.** *A* dependency scheme *is a mapping*

$$D : \Phi \mapsto D_\Phi \subseteq \prec_\Phi^q$$

that maps a PCNF formula Φ to the dependency relation D_Φ *of Φ with respect to D.*

The mapping which simply returns the prefix ordering of an input formula can be thought of as a baseline dependency scheme:

Definition 2 (Trivial Dependency Scheme)**.** *The* trivial dependency scheme D^{trv} *associates each PCNF formula Φ with the relation $D^{\text{trv}}_\Phi = \prec^q_\Phi$.*

Dependency schemes can be partially ordered based on their dependency relations: if the dependency relation computed by a dependency scheme D_1 is a subset of the dependency relation computed by a dependency scheme D_2 for each PCNF formula, then D_1 is *more general* than D_2.

By definition, D^{trv} is the least general dependency scheme. The more general a dependency scheme, the more it relaxes the prefix ordering, and so the more useful it is. Currently, (aside from the trivial dependency scheme) DepQBF supports the so-called *standard dependency scheme* [SS09].[5] We will work with the more general *reflexive resolution-path dependency scheme* [SS16b], a variant of the resolution-path dependency scheme [VG11, SS12]. This dependency scheme computes an over-approximation of variable dependencies based on whether two variables are connected by a (pair of) resolution path(s).

Definition 3 (Resolution Path)**.** *Let $\Phi = \mathcal{Q}.\varphi$ be a PCNF formula and let X be a set of variables. A* resolution path *from ℓ_1 to ℓ_{2k} in Φ is a sequence $\pi = \ell_1, \ldots, \ell_{2k}$ of literals satisfying the following properties:*

[4] The original definition of dependency schemes [SS09] is more restrictive than the one given here, but the additional requirements are irrelevant for the purposes of this thesis.

[5] Strictly speaking, it uses a refined version of the standard dependency scheme [Lon12, p.49].

1. *for all* $i \in \{1, \ldots, k\}$, *there is a* $C_i \in \varphi$ *such that* $\ell_{2i-1}, \ell_{2i} \in C_i$,
2. *for all* $i \in \{1, \ldots, k\}$, $var(\ell_{2i-1}) \neq var(\ell_{2i})$,
3. *for all* $i \in \{1, \ldots, k-1\}$, $\overline{\ell_{2i}} = \ell_{2i+1}$.

If additionally

4. *for all* $i \in \{1, \ldots, k-1\}$, $\{\ell_{2i}, \ell_{2i+1}\} \subseteq X \cup \overline{X}$,

then we say that π *is a resolution path* via X. *If* $\pi = \ell_1, \ldots, \ell_{2k}$ *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* $i \in \{1, \ldots, k-1\}$ *we say that* π goes through $var(\ell_{2i})$ *and* $var(\ell_{2i}), 1 \leq i < k$ *are the* connecting variables of π.

Resolution paths can be understood in terms of walks in the *implication graph* of a formula [SS16a].

Definition 4 (Implication graph)**.** *Let* $\Phi = \mathcal{Q}.\varphi$ *be a PCNF formula. The* implication graph *of* Φ*, denoted by* $IG(\Phi)$ *is the directed graph with vertex set* $var(\Phi) \cup \overline{var(\Phi)}$ *and edge set* $\{ (\overline{\ell}, \ell') : \text{ there is a } C \in \varphi \text{ such that } \ell, \ell' \in C \text{ and } \ell \neq \ell' \}$.

Lemma 1 ([SS16a])**.** *Let* Φ *be a PCNF formula and let* $\ell, \ell' \in var(\Phi) \cup \overline{var(\Phi)}$ *be distinct literals. The following statements are equivalent:*

1. $\ell, \ell_1, \overline{\ell_1}, \ldots, \ell_k, \overline{\ell_k}, \ell'$ *is a resolution path from* ℓ *to* ℓ',
2. $\overline{\ell}, \ell_1, \ldots, \ell_k, \ell'$ *is a path in* $IG(\Phi)$.

One can think of a resolution path as a potential chain of implications: if each clause C_i contains exactly two literals, then assigning ℓ_1 to 0 requires setting ℓ_{2k} to 1. If, in addition, there is such a path from $\overline{\ell_1}$ to $\overline{\ell_{2k}}$, then ℓ_1 and ℓ_{2k} have to be assigned opposite values. The resolution path dependency scheme identifies variables connected by a pair of resolution paths as potentially dependent on each other. We call a pair of variables connected in this way a *dependency pair*.

Definition 5 (Dependency pair)**.** *Let* Φ *be a PCNF formula and* $x, y \in var(\Phi)$. *We say* $\{x, y\}$ *is a* resolution-path dependency pair *of* Φ *with respect to* $X \subseteq var_\exists(\Phi)$ *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.

It remains to determine the set X of variables with respect to which a pair x, y of variables needs to be connected to induce a dependency. For $x \prec_\Phi y$, the original *resolution-path dependency scheme* only included dependency pairs $\{x, y\}$ connected with respect to existential variables to the right of x, excluding x and y. It turns out that this dependency scheme can be used for reordering the quantifier prefix [SS16a] but does not lead to a sound generalization of Q-resolution as required for use within a QCDCL-solver [SS16b]. By dropping the restriction that x and y must not appear on the resolution paths inducing a dependency pair, we obtain the *reflexive resolution-path dependency scheme*, which yields a sound generalization of Q-resolution [SS16b].

Definition 6 (Proper Resolution Path)**.** *Let ℓ, ℓ' be two literals of a PCNF formula Φ such that $\delta(\ell') < \delta(\ell)$. A resolution path from ℓ to ℓ' is called* proper*, if it is a resolution path via $R_\Phi(var(\ell')) \cap var_\exists(\Phi)$. If there is a proper resolution path from ℓ to ℓ', we say that ℓ and ℓ' are* properly connected *(in Φ).*

Definition 7 (Proper dependency pair)**.** *Let Φ be a PCNF formula and $x, y \in var(\Phi)$, $\delta(x) < \delta(y)$. We say $\{x, y\}$ is a* proper *resolution-path dependency pair of Φ if at least one of the following conditions holds:*

- *x and y, as well as $\neg x$ and $\neg y$, are properly connected in Φ.*
- *x and $\neg y$, as well as $\neg x$ and y, are properly connected in Φ.*

Definition 8. *The* reflexive resolution-path dependency scheme *is the mapping $\mathcal{D}^{rrs}$ that assigns to each PCNF formula $\Phi = \mathcal{Q}.\varphi$ the relation*

$$\mathcal{D}^{rrs}_\Phi = \{\, x \prec^q_\Phi y : \{x, y\} \text{ is a proper resolution-path dependency pair of } \Phi \,\}.$$

2.6 QCDCL With Dependency Schemes

DepQBF uses a dependency relation to determine the order in which variables can be assigned: if y is a variable and there is no unassigned variable x such that (x, y) is in the dependency relation, then y is considered ready for assignment. DepQBF also uses the dependency relation to generalize the $\forall$-reduction rule used in clause learning [BL10]—if no existential literals depend on a given universal literal in a clause, that universal literal may be removed. When QCDCL is generalized by a dependency scheme this way, however, its soundness requires a new argument. Consider the following true PCNF formula:

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

and a dependency scheme D that says that y does not depend on x. Thanks to the independence, the x-literals can be reduced from both clauses, leading to unit propagation assigning $y = 0$ from the first clause, which falsifies the second clause. The decision level is 0 (no decisions have been made yet), and hence the falsified clause is not asserting and

QCDCL must attempt to resolve it with the antecedent clause, i.e. the one responsible for propagating $\neg y$. After applying generalized universal reduction, the falsified clause becomes y, and resolving it with $(x \vee \neg y)$ and reducing yields the empty clause. Thus, the formula, which is true, is refuted using a suitable dependency scheme.

In fact, it is easy to see that unless a QBF is satisfiable as a propositional formula in its existential variables, disregarding the universal variables completely, there is always a dependency scheme that allows it to be refuted. Hence, a particular dependency scheme D must be "approved" before it can be used in QCDCL solving. Similarly to how ordinary QCDCL learns clauses (and terms) that can be derived using Q-resolution, it turns out that QCDCL using a dependency scheme uses a generalized proof system called Q(D)-resolution (and Q(D)-consensus), which takes a dependency scheme as a parameter. Studying this proof system can provide the necessary soundness guarantees for the dependency scheme.

Finally, ordinary QCDCL corresponds to QCDCL with the trivial dependency scheme, and likewise, Q-resolution corresponds to Q($\mathrm{D}^{\mathrm{trv}}$)-resolution, as will become apparent from the definition in Section 2.7.

2.7 Q(D)-resolution

In order to generalize Q-resolution by a dependency scheme D, we reinterpret the condition for universal reduction that uses the prefix order $\prec_\Phi^q$ with the relation D_Φ defined by the dependency scheme. The rules of the resulting proof system are shown in Figure 2.3. In particular, we note that since the resolution rule does not take the prefix into account, it remains unchanged in the dependency-enhanced version of the proof system.

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

An *input clause* $C \in \varphi$ can be used as an axiom. From two clauses $C_1 \vee e$ and $\neg e \vee C_2$, where e is an existential variable, the *resolution* rule can derive the clause $C_1 \vee C_2$, provided that it is not a tautology.

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

The *universal reduction* rule admits the deletion of a universal literal ℓ_u from a clause C under the condition that $(evar(\ell_u)) \notin D_\Phi$ for each existential variable e in C.

Figure 2.3: Derivation rules of Q(D)-resolution for a PCNF formula $\Phi = \mathcal{Q}.\varphi$.

As was mentioned in the previous section, Q(D)-resolution is not necessarily sound—that

depends on the dependency scheme. However, it was shown that both Q($\mathrm{D}^{\mathrm{rrs}}$)-resolution and Q($\mathrm{D}^{\mathrm{rrs}}$)-consensus are sound proof systems [SS16b]. If a dependency scheme D_1 is more general than a dependency scheme D_2, then every Q(D_2)-resolution step is also a Q(D_1)-resolution step, and consequently, every Q(D_2)-resolution derivation is also a Q(D_1)-resolution derivation, and soundness and other properties of Q(D_1)-resolution such as strategy extraction carry over to Q(D_2)-resolution. In this case, we have that Q($\mathrm{D}^{\mathrm{std}}$)-resolution and Q($\mathrm{D}^{\mathrm{std}}$)-consensus, and therefore also DepQBF, are sound.

CHAPTER 3

Long-distance Q-resolution with Dependency Schemes

In Chapter 2 we mentioned the various proof systems that build on Q-resolution, such as long-distance Q-resolution or Q(D)-resolution. One combination that was missing is that of *long-distance Q(D)-resolution*, i.e. long-distance Q-resolution parameterized by a dependency scheme. Therefore, it is natural to ask, whether dependency schemes can be used in combination with long-distance Q-resolution.

We provide an affirmative answer to that question by giving a sufficient condition under which a given dependency scheme D gives rise to a sound version of long-distance Q(D)-resolution. In fact, under our condition the resulting proof system is not only sound, but it also admits polynomial-time strategy extraction. We apply the general result by showing that D^{rrs} fulfils our condition, and therefore long-distance $Q(D^{rrs})$-resolution is sound and admits efficient strategy extraction.

Our result shows that a QCDCL solver can employ both learning based on long-distance Q-resolution, and a (suitable) dependency scheme. This has several advantages—long-distance Q-resolution is easier to implement and can potentially offer speedups on certain formulas. While an exponential separation is not known in general, a separation between $Q(D^{std})$-resolution and long-distance $Q(D^{std})$-resolution follows easily from the separation between Q-resolution and long-distance Q-resolution, as we discuss in Section 3.7. We further analyze the benefit of combining long-distance Q-resolution with dependency schemes experimentally by enabling both features in the QCDCL solver DepQBF [BL10]. While it does not afford a major boost of performance, there are formulas where this approach fares best.

3.1 Long-distance Q(D)-resolution

Both Q-resolution and Q(D)-resolution only allow for the derivation of non-tautological clauses, that is, clauses that do not contain a literal negated as well as unnegated. Long-distance Q-resolution (to recall the rules, see Figure 2.2) is a variant of Q-resolution that admits tautological clauses in certain cases [BJ12]. Variants of QCDCL that allow for learned clauses to be tautological [ZM02c, ZM02a] have been shown to generate proofs in long-distance Q-resolution [ELW13].

In long-distance Q-resolution, when a tautological clause is created by resolution, a variable that appears in both polarities must be to the right of the pivot variable. We generalize this by requiring that the pivot be independent of a tautological variable to obtain *long-distance Q(D)-resolution* (LDQ(D)-resolution). The derivation rules of LDQ(D)-resolution are shown in Figure 3.1.[1] Here, as in the rest of the chapter, D denotes an arbitrary dependency scheme.

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

An *input clause* $C \in \varphi$ can be used as an axiom. From two clauses $C_1 \vee e$ and $\neg e \vee C_2$, where e is an existential variable, the *(long-distance) resolution* rule can derive the clause $C_1 \vee C_2$, provided that $(u, e) \notin D_\Phi$ for each universal variable u with $u \in C_1$ and $\overline{u} \in C_2$ (or vice versa), and that $C_1 \vee C_2$ does not contain an existential variable in both polarities.

$$\frac{C}{C \setminus \{u, \neg u\}}\ \text{(∀-reduction)}$$

The ∀*-reduction* rule derives the clause $C \setminus \{u, \neg u\}$ from C, where $u \in var(C)$ is a universal variable such that $(u, e) \notin D_\Phi$ for every existential variable $e \in var(C)$.

Figure 3.1: Derivation rules of LDQ(D)-resolution for a PCNF formula $\Phi = \mathcal{Q}.\varphi$.

A derivation in a proof system consists of repeated applications of the derivation rules to derive a clause from the clauses of an input formula. Here, derivations will be represented by node-labeled directed acyclic graphs (DAGs). More specifically, we require these DAGs to have a unique sink (that is, a node without outgoing edges) and each of their nodes to have at most two incoming edges. We further assume an ordering on the in-neighbors (or parents) of every node with two incoming edges—that is, each node has a "first" and a "second" in-neighbor. Referring to such DAGs as *proof DAGs*, we define the following two operations to represent resolution and ∀-reduction:

[1] The resolution rule as defined here is more general than the one considered in an earlier version of this work [PSS16], in that we admit complementary universal literals to be "merged" as long as the pivot is independent according to D (rather than D^{trv}). This definition—which is required to capture proofs generated by DepQBF (see Example 2 in Section 3.2)—was proposed in independent work by Beyersdorff and Blinkhorn [BB16].

1. If ℓ is a literal and $\mathcal{P}_1$ and $\mathcal{P}_2$ are proof DAGs with distinct sinks v_1 and v_2, then $\mathcal{P}_1 \odot_\ell \mathcal{P}_2$ is the proof DAG consisting of the union of $\mathcal{P}_1$ and $\mathcal{P}_2$ along with a new sink v that has two incoming edges, the first one from v_1 and the second one from v_2. Moreover, if C_1 is the label of v_1 in $\mathcal{P}_1$ and C_2 is the label of v_2 in $\mathcal{P}_2$, then v is labeled with the clause $(C_1 \setminus \{\ell\}) \cup (C_2 \setminus \{\overline{\ell}\})$.

2. If u is a variable and $\mathcal{P}$ is a proof DAG with a sink w labeled with C, then $\mathcal{P} - u$ denotes the proof DAG obtained from $\mathcal{P}$ by adding an edge from w to a new node v such that v is labeled with $C \setminus \{u, \neg u\}$.

Definition 9 (Derivation)**.** *An* LDQ(D)-resolution derivation *(or* LDQ(D)-derivation*) of a clause C from a PCNF formula $\Phi = \mathcal{Q}.\varphi$ is a proof DAG $\mathcal{P}$ satisfying the following properties.*

- *Source nodes are labeled with input clauses from φ.*
- *If a node with label C has parents labeled C_1 and C_2 then C can be derived from C_1 and C_2 by (long-distance) resolution.*
- *If a node labeled with a clause C has a single parent with label C' then C can be derived from C' by $\forall$-reduction with respect to the dependency scheme D.*

We refer to these nodes as input nodes*, resolution nodes, and* $\forall$-reduction nodes*, respectively.*

Let $\mathcal{P}$ be an LDQ(D)-derivation from a PCNF formula Φ. The (clause) label of the sink node is called the *conclusion* of $\mathcal{P}$, denoted $Cl(\mathcal{P})$. If the conclusion of $\mathcal{P}$ is the empty clause then we refer to $\mathcal{P}$ as an *LDQ(D)-refutation* of Φ. For a node v of $\mathcal{P}$, the *subderivation* (of $\mathcal{P}$) rooted at v is the proof DAG induced by v and its ancestors in $\mathcal{P}$. It is straightforward to verify that the resulting proof DAG is again an LDQ(D)-derivation from Φ. For convenience, we will identify (sub)derivations with their sinks. The *size* of $\mathcal{P}$, denoted $|\mathcal{P}|$, is the total number of literal occurrences in clause labels of $\mathcal{P}$.

3.2 QCDCL with Dependency Schemes Generates LDQ(D)-Proofs

In this section, we present a version of the QCDCL algorithm that uses dependency schemes [BL10, Lon12] and performs constraint learning based on long-distance Q-resolution [ZM02b, ELW13].Generalizing an argument by Egly, Lonsing, and Widl [ELW13], we will show that this algorithm produces LDQ(D)-proofs when using a dependency scheme D.

Starting from an input PCNF formula Φ, QCDCL generates ("learns") constraints—clauses and terms—until it produces an empty constraint, at which point it returns TRUE (if the empty term is learned) or FALSE (if the empty clause is learned).

One can think of QCDCL solving as proceeding in rounds. Along with a set of clauses and terms, the solver maintains an assignment σ (the *trail*, which we will represent by a sequence of literals in the order of their assignment). During each round, this assignment is extended by quantified Boolean constraint propagation (QBCP) and—possibly—branching.

Quantified Boolean constraint propagation (with dependency scheme D) consists in the exhaustive application of universal and existential reduction (relative to D_Φ) in combination with unit assignments.[2] More specifically, QBCP reports a clause C as falsified if $C[\sigma] \neq 1$ and universal reduction can be applied to $C[\sigma]$ to obtain the empty clause. Dually, a term T is considered satisfied if $T[\sigma] \neq 0$ and $T[\sigma]$ can be reduced to the empty term. A clause C is *unit* under σ if $C[\sigma] \neq 1$ and universal reduction yields a clause $C' = (\ell)$, for some existential literal ℓ such that $var(\ell)$ is unassigned. Dually, a term T is *unit* under σ if $T[\sigma] \neq 0$ and existential reduction can be applied to obtain a term $T' = (\ell)$ containing a single universal literal ℓ. If $C = (\ell)$ is a unit clause, then the assignment σ has to be extended by ℓ in order not to falsify C, and if $T = (\ell)$ is a unit term, then σ has to be extended by $\overline{\ell}$ in order not to satisfy T. If several clauses or terms are unit under σ, QBCP nondeterministically picks one and extends the assignment accordingly. This is repeated until a constraint is empty or no unit constraints remain.

If QBCP does not lead to an empty constraint, the assignment σ is extended by *branching*. That is, the solver chooses an unassigned variable y such that every variable x with $(x, y) \in D_\Phi$ is assigned, and extends the assignment σ by y or $\neg y$.

The resulting assignment can be partitioned into so-called *decision levels*. The decision level of an assignment σ is the number of literals in σ that were assigned by branching. The decision level of a literal ℓ in σ is the decision level of the prefix of σ that ends with ℓ. Note that each decision level greater than 0 can be associated with a unique variable assigned by branching.

Eventually, the assignment maintained by QCDCL must falsify a clause or satisfy a term. When this happens (we call this a *conflict*), the solver proceeds to *conflict analysis* to derive a learned constraint C. Initially, C is the falsified clause (satisfied term), called the *conflict clause (term)*. The solver finds the existential (universal) literal in C that was assigned last by QBCP, and the antecedent clause (term) R responsible for this assignment. A new constraint is derived by resolving C and R and applying universal (existential) reduction (again, relative to D_Φ). This is done repeatedly until the resulting constraint C is *asserting*. A clause (term) S is asserting if there is a unique existential (universal) literal $\ell \in S$ with maximum decision level (greater than zero) among literals in S, the corresponding decision variable is existential (universal), and every universal (existential)

[2] For simplicity, we do not consider the pure literal rule as part of QBCP.

variable $y \in var(S)$ such that $(y, var(\ell)) \in D_\Phi$ is assigned at a lower decision level (an asserting constraint becomes unit after backtracking). Once an asserting constraint has been found, it is added to the solver's set of constraints. Finally, QCDCL *backtracks*, undoing variable assignments until reaching a decision level computed from the learned constraint.

Pseudocode for the main QCDCL loop is shown as Algorithm 1, and pseudocode for conflict analysis is shown as Algorithm 2. We now formally state and prove a correspondence between clauses learned by QCDCL and LDQ(D)-resolution.

Proposition 1. *Every clause learned by Algorithm 1 given an input PCNF Φ can be derived from Φ by LDQ(D)-resolution.*

Proof. We will show that each learned clause constructed during conflict analysis can be derived by LDQ(D)-resolution from input clauses or previously learned clauses. In addition, we prove an invariant saying that each such clause C can be reduced to the empty clause under the trail assignment restricted to variables up to and including the existential variable assigned last among those in C. Formally, if $\ell_1 \dots \ell_k$ is the trail and $i = \max\{\, j : \overline{\ell_j} \in C, 1 \leq j \leq k \,\}$, then the restricted trail is $\ell_1 \dots \ell_i$. If such an i does not exist—which can happen only if C contains no existential variable—the restricted trail is empty. Let τ denote the assignment corresponding to the restricted trail. We want to show that $C[\tau]$ simplifies to the empty clause by $\forall$-reduction. The proof is by induction on the number of resolution operations performed by Algorithm 2.

The base case with C being the conflict clause is trivial. For the inductive step, suppose that C can be reduced to the empty clause under the restricted trail assignment $\tau = \ell_1 \dots \ell_i$. Thus $\overline{\ell_i} \in C$ is the existential literal falsified last among literals in C and conflict analysis would resolve C with the antecedent R of ℓ_i next. We have to show the resolvent satisfies the above invariant and that this resolution operation is permissible in LDQ(D)-resolution. Let $\tau' = \ell_1 \dots \ell_{i-1}$ denote the trail at the time when ℓ_i was propagated. Clause R is unit under τ', that is, $R[\tau']$ reduces to (ℓ_i). In particular, we have $\tau'(\ell) = 0$ for every existential literal $\ell \in R \setminus \{\ell_i\}$ and every universal literal $\ell \in R$ assigned by τ'. Since $C[\tau]$ reduces to the empty clause by assumption, we further must have $\tau(\ell) = 0$ for every existential literal and every assigned universal literal of C. We conclude that $\tau'(\ell) = 0$ for every existential literal and every assigned universal literal in the resolvent $C' = (C \cup R) \setminus \{\ell_i, \overline{\ell_i}\}$. This property is clearly not affected by unassigning universal variables or unassigning existential variables not occurring in C', so C' reduces to the empty clause under its corresponding restricted trail assignment, proving that the invariant is preserved. Furthermore, it entails that any literal $\ell \in R$ such that $\overline{\ell} \in C$ must be universal and unassigned by τ'. Since $R[\tau']$ can be reduced to (ℓ_i) by $\forall$-reduction, we must have $(var(\ell), var(\ell_i)) \notin D_\Phi$ for each such literal ℓ, so C' can be derived from C and R in LDQ(D)-resolution. □

Algorithm 1 QCDCL with Dependency Scheme D

```
1:  procedure SOLVE(Φ)
2:      compute D_Φ
3:      while TRUE do
4:          conflict = QBCP()
5:          if conflict == NONE then
6:              DECIDE()
7:          else
8:              constraint, btlevel = ANALYZECONFLICT(conflict)
9:              if ISEMPTY(constraint) then
10:                 if ISTERM(constraint) then
11:                     return TRUE
12:                 else
13:                     return FALSE
14:                 end if
15:             else
16:                 ADDLEARNEDCONSTRAINT(constraint)
17:                 BACKTRACK(btlevel)
18:             end if
19:         end if
20:     end while
21: end procedure
```

Algorithm 2 Conflict Analysis with Long-Distance Q-resolution

```
1:  procedure ANALYZECONFLICT(conflict)
2:      constraint = GETCONFLICTCONSTRAINT(conflict)
3:      while NOT ASSERTING(constraint) do
4:          pivot = GETPIVOT(constraint)
5:          reason = GETANTECEDENT(pivot)
6:          constraint = RESOLVE(constraint, reason, pivot)
7:          constraint = REDUCE(constraint)
8:      end while
9:      btlevel = GETBACKTRACKLEVEL(constraint)
10:     return constraint, btlevel
11: end procedure
```

As in the case of QCDCL without dependency schemes [GNT06, ELW13] an analogue of this result can be proved for learned terms and a dual proof system ("Q-consensus") that operates on terms instead of clauses.

The proof of Proposition 1 uses the fact that two clauses $C_1 \vee u \vee e$ and $C_2 \vee \neg u \vee \neg e$ can be resolved on variable e even if $u < e$ as long as $(u, e) \notin D_\Phi$. The following example illustrates that this generalization of the resolution rule is necessary for LDQ(D)-resolution to trace QCDCL with dependency schemes and long-distance Q-resolution.

Example 2. Consider the formula $\Phi = \exists z_0 \exists z_1 \forall x \exists y \exists z_2 \exists z_3 \exists a \exists b . \varphi \wedge \psi$, where

$$\varphi = \underbrace{(\overline{z_1} \vee x \vee z_2 \vee \overline{a})}_{C_1} \wedge \underbrace{(z_1 \vee y \vee z_2)}_{C_2} \wedge \underbrace{(\overline{x} \vee \overline{y} \vee z_3 \vee \overline{b})}_{C_3} \wedge \underbrace{(\overline{z_2} \vee z_0)}_{C_4} \wedge \underbrace{(\overline{z_2} \vee \overline{z_0})}_{C_5} \wedge \underbrace{(\overline{y} \vee \overline{z_3})}_{C_6},$$

and ψ consists of auxiliary clauses

$$\psi = (a) \wedge (b) \wedge (\overline{x} \vee a) \wedge (\overline{x} \vee b).$$

The clauses in ψ are there simply to enforce that a and b are set to true and (in conjunction with φ) that $(x, a), (x, b) \in \mathrm{D}^{\mathrm{rrs}}_{\Phi}$. It is straightforward to check that the set of dependencies computed by the reflexive resolution-path dependency scheme is

$$\mathrm{D}^{\mathrm{rrs}}_{\Phi} = \{(z_0, x), (z_1, x), (x, a), (x, b)\}.$$

That is, the dependency scheme identifies the syntactic dependencies $(x, y), (x, z_2)$, and (x, z_3) as spurious.

We now construct a possible trace of QCDCL on Φ with $\mathrm{D}^{\mathrm{rrs}}$ and learning based on long-distance Q-resolution. At decision level 0 the unit clauses in ψ are propagated, setting $a = 1$ and $b = 1$. This does not lead to further propagation and QCDCL proceeds with the decision $y \overset{d}{=} 0$. Note that y can be assigned before x because $(x, y) \notin \mathrm{D}^{\mathrm{rrs}}_{\Phi}$. This assignment does not lead to any literals being propagated, so the algorithm makes another decision $z_2 \overset{d}{=} 0$. Now clause C_2 simplifies to the unit clause (z_1) and $z_1 = 1$ is propagated. Clause C_1 only contains x under the resulting assignment and we have a conflict. Conflict analysis first resolves clauses C_1 and C_2 to obtain the clause $C_{12} = (x \vee y \vee z_2 \vee \neg a)$. Variable a depends on x, so $\forall$-reduction cannot be applied. Since z_2 is the only variable from the second decision level in clause C_{12} and z_2 does not depend on x, C_{12} is asserting and the clause is learned by QCDCL. Backtracking undoes the decision involving z_2 and propagates $z_2 = 1$ instead. As a result, C_4 simplifies to (z_0), unit propagation assigns $z_0 = 1$, and clause C_5 is falsified. Conflict analysis resolves C_4 and C_5 to derive the learned (unit) clause $C_{45} = (\overline{z_2})$, which causes QCDCL to backtrack to decision level 0 and propagate $z_2 = 0$. Now clause C_{12} simplifies to $(x \vee y)$. Since y is independent of x we can apply $\forall$-reduction to obtain the unit clause (y) which propagates the assignment $y = 1$. Clause C_6 in turn becomes unit and propagates $z_3 = 0$. As a result, clause C_3 simplifies to $(\overline{x})$ and reduces to the empty clause by $\forall$-reduction. Conflict analysis resolves C_3 and C_6 so as to obtain the clause $(\overline{x} \vee \overline{y} \vee \overline{b})$. Variable b depends on x, so $\forall$-reduction cannot be applied. Next, clause $(\overline{x} \vee \overline{y} \vee \overline{b})$ is resolved with C_{12} to derive $(x \vee \overline{x} \vee z_2 \vee \overline{a} \vee \overline{b})$. Note that this resolution step is permissible since $(x, y) \notin \mathrm{D}^{\mathrm{rrs}}_{\Phi}$. Further resolution steps involving unit clauses yield the clause $(x \vee \overline{x})$, which can be reduced to the empty clause, so that QCDCL terminates with return value FALSE.

3.3 Soundness of and Strategy Extraction for LDQ($\mathbf{D^{rrs}}$)

A PCNF formula can be associated with an evaluation game played between an existential and a universal player. These players take turns assigning quantifier blocks in the order

of the prefix. The existential player wins if the matrix evaluates to 1 under the resulting variable assignment, while the universal player wins if the matrix evaluates to 0. One can show that the formula is true (false) if and only if the existential (universal) player has a winning strategy in this game, and this winning strategy is a (counter)model.

Goultiaeva, Van Gelder and Bacchus [GVGB11] proved that a Q-resolution refutation can be used to compute winning moves for the universal player in the evaluation game. The idea is that Universal maintains a "restriction" of the refutation by the assignment constructed in the evaluation game, which is a refutation of the restricted formula.

For assignments made by the existential player, the universal player only needs to consider each instance of resolution whose pivot variable is assigned: one of the premises is not satisfied and can be used to (re)construct a refutation.

When it is Universal's turn, the quantifier block for which she needs to pick an assignment is leftmost in the restricted formula. This means that $\forall$-reduction of these variables is blocked by any of the remaining existential variables and can only be applied to a purely universal clause. In a Q-resolution refutation, these variables must therefore be reduced at the very end, and because Q-resolution does not permit tautological clauses, only one polarity of each universal variable from the leftmost block can appear in a refutation. It follows that Universal can maintain a Q-resolution refutation by assigning variables from the leftmost block in such a way as to map the associated literals to 0, effectively deleting them from the remaining Q-resolution refutation.

In this manner, the universal player can maintain a refutation until the end of the game, when all variables have been assigned. At that point, a refutation consists only of the empty clause, which means that the assignment chosen by the two players falsifies a clause of the original matrix and universal has won the game.

Egly, Lonsing, and Widl [ELW13] observed that this argument goes through even in the case of long-distance Q-resolution, since a clause containing both u and $\neg u$ for a universal variable u can only be derived by resolving on an existential variable to the left of u, but no such existential variable exists if u is from the leftmost block.

In this section, we will prove that this argument can be generalized to LDQ($\mathrm{D}^{\mathrm{rrs}}$)-refutations. We illustrate this correspondence with an example:

Example 3. Consider the PCNF formula

$$\Phi = \exists x\, \forall u\, \exists e, y \quad (x \vee u \vee \overline{y}) \wedge (\overline{x} \vee \overline{u} \vee \overline{y}) \wedge (x \vee y) \wedge (\overline{x} \vee e) \wedge (\overline{u} \vee y) \wedge (\overline{y} \vee e)$$

Figure 3.2 shows an LDQ($\mathrm{D}^{\mathrm{rrs}}$) -refutation of Φ. The only universal variable is u, so a strategy for the universal player in the evaluation game associated with Φ has to determine an assignment to u given an assignment to x, the only (existential) variable preceding u. The figure illustrates how to compute the assignment to u for the two possible assignments $\tau : \{x\} \to \{0, 1\}$ from the restriction of the refutation by τ. In both cases, only one polarity of u occurs in the restricted refutation and therefore it

is easy for Universal to determine the correct assignment. Notice that in one of the cases, a generalized ∀-reduction node remains present in the restriction—this shows that we cannot limit ourselves to looking at the final reduction step in the proof when looking for the variables to assign (as is the case with ordinary Q-resolution refutations, cf. [GVGB11]).

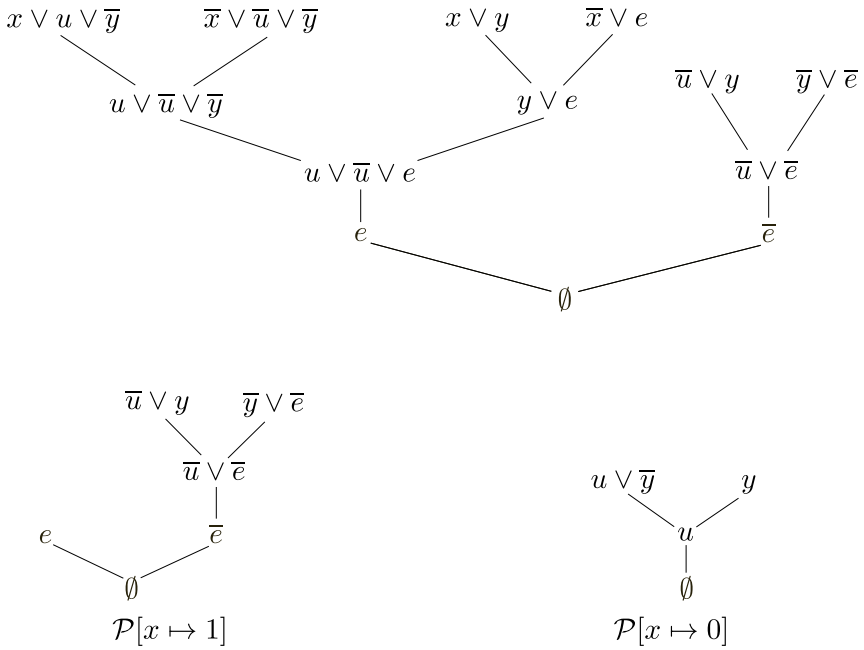

Figure 3.2: An LDQ($\mathrm{D^{rrs}}$)-refutation of the formula Φ from Example 3 (above) and two restrictions (below).

The key property that allows Universal to maintain a refutation in the above example is that universal variables from the leftmost quantifier block may appear in at most one polarity. We now show that this property is in fact sufficient for soundness of LDQ(D)when combined with a natural monotonicity property of dependency schemes.

Definition 10. *A dependency scheme D is* monotone *if $D_{\Phi[\tau]} \subseteq D_\Phi$ for every PCNF formula Φ and every assignment τ to a subset of $var(\Phi)$. We say that D is* simple *if, for every PCNF formula $\Phi = \forall X \mathcal{Q}.\varphi$, every LDQ(D)-derivation $\mathcal{P}$ from Φ, and every universal variable $u \in X$, u or $\overline{u}$ does not appear in $\mathcal{P}$. A dependency scheme D is* normal *if it is both monotone and simple.*

As in the case of Q-resolution, Universal's move for a particular quantifier block can be computed from the assignment corresponding to the previous moves and the refutation

in polynomial time. Since every polynomial-time algorithm can be implemented by a family of polynomially-sized circuits, and because these circuits can even be computed in polynomial time [AB09, p.109], it follows that LDQ(D)admits polynomial-time strategy extraction when D is normal. We present an explicit construction with a more specific bound on the runtime.

Theorem 2. *Let D be a normal dependency scheme. There is an algorithm that computes a countermodel of a PCNF formula Φ with n variables from an LDQ(D)-refutation $\mathcal{P}$ of Φ in time $O(|\mathcal{P}| \cdot n)$.*

As an application of this general result, we will prove that the reflexive resolution-path dependency scheme is normal in Section 3.4.1 below.

Theorem 3. $\mathrm{D}^{\mathrm{rrs}}$ *is normal.*

Corollary 1. *There is a polynomial-time algorithm that, given a PCNF formula Φ and an* LDQ($\mathrm{D}^{\mathrm{rrs}}$)*-refutation of Φ, computes a countermodel of Φ.*

This result immediately carries over to the less general standard dependency scheme.

Corollary 2. *There is a polynomial-time algorithm that, given a PCNF formula Φ and an LDQ(D^{std})-refutation of Φ, computes a countermodel of Φ.*

In combination with Proposition 1, these results imply soundness of both proof systems.

Corollary 3. *The systems LDQ(D^{std})-resolution and LDQ($\mathrm{D}^{\mathrm{rrs}}$)-resolution are sound.*

3.4 Proof of Theorem 2

We begin by formally defining the "restriction" of an LDQ(D)-derivation by an assignment, which is a straightforward generalization of this operation for Q-resolution derivations [GVGB11].[3] The result of restricting a derivation is either a derivation or the object $\top$, which can be interpreted as representing the tautological clause containing every literal. Accordingly, we stipulate that $\ell \in \top$ for every literal ℓ.

Definition 11 (Restriction)**.** *Let Φ be a PCNF formula and let $\mathcal{P}$ be an LDQ(D)-derivation from Φ. Further, let $X \subseteq var(\Phi)$ and let $\tau : X \to \{0,1\}$ be a truth assignment. The* restriction of $\mathcal{P}$ by τ*, in symbols $\mathcal{P}[\tau]$, is defined as follows.*

1. *If $\mathcal{P}$ is an input node there are two cases. If $Cl(\mathcal{P})[\tau] = 1$ then $\mathcal{P}[\tau] = \top$. Otherwise, $\mathcal{P}[\tau]$ is the proof DAG consisting of a single node labeled with $Cl(\mathcal{P})[\tau]$.*

[3] Our definition slightly differs from the original for the resolution rule: if restriction removes the pivot variable from both premises, we simply pick the (restriction of the) first premise as the result (rather than the clause containing fewer literals). This simplifies the certificate extraction argument given below.

2. *If $\mathcal{P} = \mathcal{P}_1 \odot_\ell \mathcal{P}_2$, that is, if $\mathcal{P}$ is a resolution node, we distinguish four cases:*
   a) *If $\ell \notin Cl(\mathcal{P}_1[\tau])$ then $\mathcal{P}[\tau] = \mathcal{P}_1[\tau]$.*
   b) *If $\ell \in Cl(\mathcal{P}_1[\tau])$ and $\overline{\ell} \notin Cl(\mathcal{P}_2[\tau])$ then $\mathcal{P}[\tau] = \mathcal{P}_2[\tau]$.*
   c) *If $\ell \in Cl(\mathcal{P}_1[\tau])$, $\overline{\ell} \in Cl(\mathcal{P}_2[\tau])$, and $\mathcal{P}_1[\tau] = \top$ or $\mathcal{P}_2[\tau] = \top$, we let $\mathcal{P}[\tau] = \top$.*
   d) *If $\ell \in Cl(\mathcal{P}_1[\tau])$, $\overline{\ell} \in Cl(\mathcal{P}_2[\tau])$, $\mathcal{P}_1[\tau] \neq \top$, and $\mathcal{P}_2[\tau] \neq \top$, we define $\mathcal{P}[\tau] = \mathcal{P}_1[\tau] \odot_\ell \mathcal{P}_2[\tau]$.*
3. *If $\mathcal{P} = \mathcal{P}' - u$, that is, if $\mathcal{P}$ is a $\forall$-reduction node, we distinguish three cases:*
   a) *If $\mathcal{P}'[\tau] = \top$ then $\mathcal{P}[\tau] = \top$.*
   b) *If $\mathcal{P}'[\tau] \neq \top$ and $u \notin var(Cl(\mathcal{P}'[\tau]))$ then $\mathcal{P}[\tau] = \mathcal{P}'[\tau]$.*
   c) *If $\mathcal{P}'[\tau] \neq \top$ and $u \in var(Cl(\mathcal{P}'[\tau]))$ then $\mathcal{P}[\tau] = \mathcal{P}'[\tau] - u$.*

If D is a monotone dependency scheme, LDQ(D)-refutations are preserved under restriction by an existential assignment (cf. [GVGB11, Lemma 4]). This is stated in the following lemma, which can by proved by a straightforward induction on the structure of the LDQ(D)-derivation.

Lemma 2. *Let D be a monotone dependency scheme, let $\mathcal{P}$ be an LDQ(D)-derivation from a PCNF formula Φ, let $E \subseteq var_\exists(\Phi)$, and let $\tau : E \to \{0,1\}$ be an assignment. If $\mathcal{P}[\tau] = \top$ then $Cl(\mathcal{P})[\tau] = 1$. Otherwise, $\mathcal{P}[\tau]$ is an LDQ(D)-derivation from $\Phi[\tau]$ such that $Cl(\mathcal{P}[\tau]) \subseteq Cl(\mathcal{P})[\tau]$.*

Proof. The proof is by induction on the structure of $\mathcal{P}$.

1. If $\mathcal{P}$ is an input node then $\mathcal{P}[\tau] = \top$ iff $Cl(\mathcal{P})[\tau] = 1$ and $Cl(\mathcal{P}[\tau]) = Cl(\mathcal{P})[\tau]$ otherwise, so the statement holds trivially.
2. If $\mathcal{P} = \mathcal{P}_1 \odot_\ell P_2$ is a resolution node we distinguish four cases:
   a) If $\ell \notin Cl(\mathcal{P}_1[\tau])$, then $\mathcal{P}[\tau] = \mathcal{P}_1[\tau]$ and
   $$Cl(\mathcal{P}_1[\tau]) = Cl(\mathcal{P}_1[\tau]) \setminus \{\ell\} \subseteq Cl(\mathcal{P}_1)[\tau] \setminus \{\ell\} \subseteq Cl(\mathcal{P})[\tau],$$
   where the first inclusion holds by induction hypothesis and the second inclusion follows from the definition of the resolution rule.
   b) If $\ell \in Cl(\mathcal{P}_1[\tau])$ and $\overline{\ell} \notin Cl(\mathcal{P}_2[\tau])$ then $\mathcal{P}[\tau] = \mathcal{P}_2[\tau]$ and the statement follows via a symmetric argument.
   c) If $\ell \in Cl(\mathcal{P}_1[\tau])$, $\overline{\ell} \in Cl(\mathcal{P}_2[\tau])$, and $\mathcal{P}_1[\tau] = \top$ or $\mathcal{P}_2[\tau] = \top$ then we have $\mathcal{P}[\tau] = \top$. Assume without loss of generality that $\mathcal{P}_1[\tau] = \top$. Then $Cl(\mathcal{P}_1)[\tau] = 1$ by induction hypothesis. Let $\ell' \in Cl(\mathcal{P}_1)$ be a literal such that $\tau(\ell') = 1$. We distinguish two cases. If $\ell \neq \ell'$ then $\ell' \in Cl(\mathcal{P})$ and

$Cl(\mathcal{P})[\tau] = 1$. Otherwise, $\tau(\overline{\ell'}) = \tau(\overline{\ell}) = 0$, and we must have $\mathcal{P}_2[\tau] = \top$ since $\overline{\ell} \in Cl(\mathcal{P}_2[\tau])$. By induction hypothesis, there has to be another literal $\ell'' \neq \overline{\ell}$ such that $\ell'' \in Cl(\mathcal{P}_2)$ and $\tau(\ell'') = 1$. The literal ℓ'' is contained in $Cl(\mathcal{P})$ as well, so $Cl(\mathcal{P})[\tau] = 1$.

d) If $\ell \in Cl(\mathcal{P}_1[\tau])$, $\overline{\ell} \in Cl(\mathcal{P}_2[\tau])$, $\mathcal{P}_1[\tau] \neq \top$, and $\mathcal{P}_2[\tau] \neq \top$, then $\mathcal{P}[\tau] = \mathcal{P}_1[\tau] \odot_\ell \mathcal{P}_2[\tau]$ and $\mathcal{P}[\tau] \neq \top$. By the induction hypothesis, $\mathcal{P}_1[\tau]$ is an LDQ(D)-derivation from $\Phi[\tau]$ such that $Cl(\mathcal{P}_1[\tau]) \subseteq Cl(P_1)[\tau]$, and $\mathcal{P}_2[\tau]$ is an LDQ(D)-derivation from $\Phi[\tau]$ such that $Cl(\mathcal{P}_2[\tau]) \subseteq Cl(P_2)[\tau]$. Monotonicity of D ensures that after restriction, the resolution step is still sound and thus $\mathcal{P}[\tau]$ is an LDQ(D)-derivation from $\Phi[\tau]$ as well and

$$\begin{aligned} Cl(\mathcal{P}[\tau]) &= Cl(\mathcal{P}_1[\tau] \odot_\ell \mathcal{P}_2[\tau]) \\ &= Cl(\mathcal{P}_1[\tau]) \cup Cl(\mathcal{P}_2[\tau]) \setminus \{\ell, \overline{\ell}\} \\ &\subseteq Cl(\mathcal{P}_1)[\tau] \cup Cl(\mathcal{P}_2)[\tau] \setminus \{\ell, \overline{\ell}\} = Cl(\mathcal{P})[\tau]. \end{aligned}$$

3. If $\mathcal{P} = \mathcal{P}' - u$ is a reduction node, we have to distinguish two cases:

a) If $\mathcal{P}'[\tau] = \top$ then $\mathcal{P}[\tau] = \top$ by definition. By induction hypothesis $Cl(\mathcal{P}')[\tau] = 1$ and since τ does not assign u, we get $Cl(\mathcal{P})[\tau] = 1$ as well.

b) If $\mathcal{P}[\tau] \neq \top$ then $\mathcal{P}'[\tau] \neq \top$ by definition of the restriction operation. By induction hypothesis, $\mathcal{P}'[\tau]$ is an LDQ(D)-derivation from $\Phi[\tau]$ such that $Cl(\mathcal{P}'[\tau]) \subseteq Cl(\mathcal{P}')[\tau]$. If $u \notin var(Cl(\mathcal{P}'[\tau]))$ then $\mathcal{P}[\tau] = \mathcal{P}'[\tau]$ and the statement holds. Otherwise, if $u \in var(Cl(\mathcal{P}'[\tau]))$ then $\mathcal{P}[\tau] = \mathcal{P}'[\tau] - u$ and thus

$$\begin{aligned} Cl(\mathcal{P}[\tau]) &= Cl(\mathcal{P}'[\tau]) \setminus \{u, \neg u\} \\ &\subseteq Cl(\mathcal{P}')[\tau] \setminus \{u, \neg u\} = (Cl(\mathcal{P}') \setminus \{u, \neg u\})[\tau] = Cl(\mathcal{P})[\tau], \end{aligned}$$

where the last but one equality holds because τ does not assign u. To see that $\mathcal{P}[\tau] = \mathcal{P}'[\tau] - u$ is a valid $\forall$-reduction node, note that $Cl(\mathcal{P}'[\tau]) \subseteq Cl(\mathcal{P}')$ by induction hypothesis and observe that $D_{\Phi[\tau]} \subseteq D_\Phi$.

□

Above, we argued that the universal player can use an LDQ(D)-refutation for a normal dependency scheme D in order to compute winning moves in the evaluation game associated with a PCNF formula and that this can be used to compute a countermodel of the formula in polynomial time. We now prove this directly, by showing how to construct a circuit implementing a countermodel from an LDQ(D)-refutation.

We begin by describing auxiliary circuits simulating the restriction operation. Let $\Phi = Q_1 X_1 \dots Q_k X_k . \varphi$ be a PCNF formula and let $\mathcal{P}$ be a refutation of Φ. For each quantifier block X_i, each subderivation $\mathcal{S}$ of $\mathcal{P}$, and each literal ℓ, we will construct

circuits $\textsc{top}^i_{\mathcal{S}}$ and $\textsc{contains}^i_{\mathcal{S},\ell}$ with inputs from $X = \bigcup_{j<i} X_j$ such that, for every assignment $\sigma : X \to \{0,1\}$,

$$\textsc{top}^i_{\mathcal{S}}[\sigma] = 1 \iff \mathcal{S}[\sigma] = \top \tag{3.1}$$

$$\textsc{contains}^i_{\mathcal{S},\ell}[\sigma] = 1 \iff \ell \in Cl(\mathcal{S}[\sigma]) \tag{3.2}$$

We first describe our construction and then prove that it satisfies the above properties in Lemma 3. Let $\mathcal{S}$ be an input node. We let

$$\textsc{top}^1_{\mathcal{S}} := \bigvee \left(Cl(\mathcal{S}) \cap (X_1 \cup \overline{X_1}) \right),$$

and define $\textsc{top}^i_{\mathcal{S}}$ for $1 < i \leq k$ as

$$\textsc{top}^i_{\mathcal{S}} := \textsc{top}^{i-1}_{\mathcal{S}} \vee \bigvee \left(Cl(\mathcal{S}) \cap (X_i \cup \overline{X_i}) \right).$$

Moreover, for $1 \leq i \leq k$ we define $\textsc{contains}^i_{\mathcal{S},\ell}$ as

$$\textsc{contains}^i_{\mathcal{S},\ell} = \begin{cases} 1 & \text{if } \ell \in Cl(\mathcal{S}) \setminus (X \cup \overline{X}), \\ \textsc{top}^i_{\mathcal{S}} & \text{otherwise.} \end{cases}$$

For non-input nodes, we proceed as follows. If $\mathcal{S} = \mathcal{S}_1 \odot_q \mathcal{S}_2$, we define $\textsc{top}^i_{\mathcal{S}}$ as

$$\textsc{top}^i_{\mathcal{S}} = (\textsc{contains}^i_{\mathcal{S}_1,q} \wedge \textsc{top}^i_{\mathcal{S}_2}) \vee (\textsc{contains}^i_{\mathcal{S}_2,\overline{q}} \wedge \textsc{top}^i_{\mathcal{S}_1}),$$

and if $\mathcal{S} = \mathcal{S}' - u$, we let

$$\textsc{top}^i_{\mathcal{S}} := \textsc{top}^i_{\mathcal{S}'}.$$

For the $\textsc{contains}^i_{\mathcal{S},\ell}$ circuit, we distinguish two cases. Let ℓ be a literal and $\mathcal{S}$ a derivation. If $\ell \notin Cl(\mathcal{S})$ we simply let

$$\textsc{contains}^i_{\mathcal{S},\ell} := \textsc{top}^i_{\mathcal{S}}.$$

Otherwise, if $\ell \in Cl(\mathcal{S})$, we have to consider two cases. First, if $\mathcal{S} = \mathcal{S}_1 \odot_q \mathcal{S}_2$, we let

$$\begin{aligned} \textsc{contains}^i_{\mathcal{S},\ell} = & \textsc{top}^i_{\mathcal{S}} \vee \\ & (\neg\textsc{contains}^i_{\mathcal{S}_1,q} \wedge \textsc{contains}^i_{\mathcal{S}_1,\ell}) \vee \\ & (\textsc{contains}^i_{\mathcal{S}_1,q} \wedge \neg\textsc{contains}^i_{\mathcal{S}_2,\overline{q}} \wedge \textsc{contains}^i_{\mathcal{S}_2,\ell}) \vee \\ & (\textsc{contains}^i_{\mathcal{S}_1,q} \wedge \textsc{contains}^i_{\mathcal{S}_2,\overline{q}} \wedge (\textsc{contains}^i_{\mathcal{S}_1,\ell} \vee \textsc{contains}^i_{\mathcal{S}_2,\ell})). \end{aligned}$$

Second, if $\mathcal{S} = \mathcal{S}' - u$, then

$$\textsc{contains}^i_{\mathcal{S},\ell} := \textsc{contains}^i_{\mathcal{S}',\ell}.$$

To implement the winning strategy for Universal sketched above, we further construct circuits $\text{POLARITY}_{\mathcal{S},u}$ for each node $\mathcal{S}$ of $\mathcal{P}$ and each universal variable $u \in var_\forall(\Phi)$, such that, for each assignment $\tau : L_\Phi(u) \to \{0,1\}$,

$$\text{POLARITY}_{\mathcal{S},u}[\tau] = 1 \iff u \text{ occurs in } \mathcal{S}[\tau]. \tag{3.3}$$

Let $u \in X_i$ be a universal variable from the ith quantifier block. If $\mathcal{S}$ is an input node, we simply define

$$\text{POLARITY}_{\mathcal{S},u} := \text{CONTAINS}^i_{\mathcal{S},u},$$

and if $\mathcal{S} = \mathcal{S}' - u$ is a $\forall$-reduction node, we let

$$\text{POLARITY}_{\mathcal{S},u} := \text{POLARITY}_{\mathcal{S}',u}.$$

If $\mathcal{S} = \mathcal{S}_1 \odot_q \mathcal{S}_2$, then

$$\begin{aligned}\text{POLARITY}_{\mathcal{S},u} :=& (\neg\text{CONTAINS}^i_{\mathcal{S}_1,q} \wedge \text{POLARITY}_{\mathcal{S}_1,u}) \vee \\ & (\text{CONTAINS}^i_{\mathcal{S}_1,q} \wedge \neg\text{CONTAINS}^i_{\mathcal{S}_2,\bar{q}} \wedge \text{POLARITY}_{\mathcal{S}_2,u}) \vee \\ & (\text{CONTAINS}^i_{\mathcal{S}_1,q} \wedge \text{CONTAINS}^i_{\mathcal{S}_2,\bar{q}} \wedge (\text{POLARITY}_{\mathcal{S}_1,u} \vee \text{POLARITY}_{\mathcal{S}_2,u})).\end{aligned}$$

Lemma 3. *Let $\Phi = Q_1X_1 \dots Q_kX_k.\varphi$ be a PCNF formula and $\mathcal{P}$ an LDQ(D)-derivation from Φ. For each $1 \le i \le k$, each literal ℓ, every $u \in var_\forall(\Phi) \cap X_i$, and every truth assignment $\sigma : \bigcup_{j=1}^{i-1} X_j \to \{0,1\}$, $\text{TOP}^i_\mathcal{P}$ satisfies (3.1), $\text{CONTAINS}^i_{\mathcal{P},\ell}$ satisfies (3.2), and $\text{POLARITY}_{\mathcal{P},u}$ satisfies (3.3).*

Proof. Let $X = X_1 \cup \dots \cup X_{i-1}$. As (3.1) and (3.2) are related, we will prove them first. We will use induction on the structure of $\mathcal{P}$, with the induction hypothesis that (3.1) and (3.2) hold. The inductive step will be carried out in two phases. In the first phase, we prove that (3.1) holds and in the second phase we use this additional information to prove that (3.2) holds as well.

1. Let $\mathcal{P}$ be an input node. By Definition 11 we have $\mathcal{P}[\sigma] = \top$ if, and only if, $Cl(\mathcal{P})[\sigma] = 1$. Since σ only assigns variables in X, this is the case if, and only if, $\text{TOP}^i_\mathcal{P}[\sigma] = 1$, so (3.1) holds.

2. Let $\mathcal{P} = \mathcal{P}_1 \odot_q \mathcal{P}_2$ such that (3.1) and (3.2) hold for $\mathcal{P}_1$ and $\mathcal{P}_2$. We distinguish several cases.

   a) $q \notin Cl(\mathcal{P}_1[\tau])$. Then $\mathcal{P}[\tau] = \mathcal{P}_1[\tau]$. Since $q \notin Cl(\mathcal{P}_1[\tau])$, it cannot be the case that $\mathcal{P}_1[\tau] = \top$ and so $\mathcal{P}[\tau] \neq \top$ as well. By the induction hypothesis, we have $\text{CONTAINS}^i_{\mathcal{P}_1,q}[\tau] = 0$ and also $\text{TOP}^i_{\mathcal{P}_1}[\tau] = 0$ which means $\text{TOP}^i_\mathcal{P}[\tau] = 0$ as required.

b) $q \in Cl(\mathcal{P}_1[\tau])$ and $\overline{q} \notin Cl(\mathcal{P}_2[\tau])$. Then $\mathcal{P}[\tau] = \mathcal{P}_2[\tau]$. Since $\overline{q} \notin Cl(\mathcal{P}_2[\tau])$, we cannot have $\mathcal{P}_2[\tau] = \top$ and thus $\mathcal{P}[\tau] \neq \top$ as well. By the induction hypothesis, we have $\textsc{contains}^i_{\mathcal{P}_2,\overline{q}}[\tau] = 0$ and also $\textsc{top}^i_{\mathcal{P}_2}[\tau] = 0$ which means $\textsc{top}^i_{\mathcal{P}}[\tau] = 0$ as required.

c) $q \in Cl(\mathcal{P}_1[\tau])$ and $\overline{q} \in Cl(\mathcal{P}_2[\tau])$ and $\mathcal{P}_1[\tau] = \top$ or $\mathcal{P}_2[\tau] = \top$. Then $\mathcal{P}[\tau] = \top$ and by induction hypothesis, we have $\textsc{contains}^i_{\mathcal{P}_1,q}[\tau] = 1$ as well as $\textsc{contains}^i_{\mathcal{P}_2,\overline{q}}[\tau] = 1$, and $\textsc{top}^i_{\mathcal{P}_1}[\tau] = 1$ or $\textsc{top}^i_{\mathcal{P}_2}[\tau] = 1$. In any case, $\textsc{top}^i_{\mathcal{P}}[\tau] = 1$.

d) $q \in Cl(\mathcal{P}_1[\tau])$ and $\overline{q} \in Cl(\mathcal{P}_2[\tau])$ and $\mathcal{P}_1[\tau] \neq \top$ and $\mathcal{P}_2[\tau] \neq \top$. Then $\mathcal{P}[\tau] = \mathcal{P}_1[\tau] \odot_q \mathcal{P}_2[\tau] \neq \top$. By induction hypothesis, we have $\textsc{top}^i_{\mathcal{P}_1}[\tau] = 0$ and $\textsc{top}^i_{\mathcal{P}_2}[\tau] = 0$, which ensures $\textsc{top}^i_{\mathcal{P}}[\tau] = 0$.

3. Let $\mathcal{P} = \mathcal{P}' - u$. From the definitions, we can immediately see that $\mathcal{P}'[\tau] = \top \iff \mathcal{P}[\tau] = \top$ and $\textsc{top}^i_{\mathcal{P}'} = \textsc{top}^i_{\mathcal{P}}$ which proves (3.1).

We have proved that $\mathcal{P}[\tau] = \top \iff \textsc{top}^i_{\mathcal{P}}[\tau] = 1$, and it can be easily checked that, by definition, $\textsc{top}^i_{\mathcal{P}} \Rightarrow \textsc{contains}^i_{\mathcal{P},\ell}$ for every literal ℓ. Therefore, if $\mathcal{P}[\tau] = \top$, (3.2) holds and in the following, we can restrict ourselves to the cases when $\mathcal{P}[\tau] \neq \top$. Also, we can restrict ourselves to the cases when ℓ (the literal in question) actually belongs to $Cl(\mathcal{P})$, because otherwise $\textsc{contains}^i_{\mathcal{P},\ell} = \textsc{top}^i_{\mathcal{P}}$ and in that case (3.2) clearly holds.

1. Let $\mathcal{P}$ be an input node. We may assume $\mathcal{P}[\tau] \neq \top$ and $\ell \in Cl(\mathcal{P})$ by the above. By definition, we can easily see that $\textsc{contains}^i_{\mathcal{P},\ell}[\tau] = 1$ if, and only if, $\ell \in Cl(\mathcal{P}[\tau])$.

2. Let $\mathcal{P} = \mathcal{P}_1 \odot_q \mathcal{P}_2$ such that (3.1) and (3.2) hold for $\mathcal{P}_1$ and $\mathcal{P}_2$. We distinguish several cases.

   a) $q \notin Cl(\mathcal{P}_1[\tau])$. By the induction hypothesis, we have $\textsc{contains}^i_{\mathcal{P}_1,q}[\tau] = 0$. Also $\mathcal{P}[\tau] = \mathcal{P}_1[\tau]$ and

   $$\ell \in Cl(\mathcal{P}[\tau]) \iff \ell \in Cl(\mathcal{P}_1[\tau]) \iff \textsc{contains}^i_{\mathcal{P}_1,\ell}[\tau],$$

   where the second equivalence holds by induction hypothesis. Since we have $\textsc{contains}^i_{\mathcal{P}_1,q}[\tau] = 0$, we can write

   $$\textsc{contains}^i_{\mathcal{P}_1,\ell}[\tau] \iff \neg\textsc{contains}^i_{\mathcal{P}_1,q}[\tau] \wedge \textsc{contains}^i_{\mathcal{P}_1,\ell}[\tau].$$

   Because $\textsc{contains}^i_{\mathcal{P}_1,q}[\tau] = 0$ and $\textsc{top}^i_{\mathcal{P}}[\tau] = 0$, the only disjunct in the definition of $\textsc{contains}^i_{\mathcal{P},\ell}[\tau]$ that can possibly be satisfied is the second one, so that

   $$\textsc{contains}^i_{\mathcal{P},\ell}[\tau] \iff \neg\textsc{contains}^i_{\mathcal{P}_1,q}[\tau] \wedge \textsc{contains}^i_{\mathcal{P}_1,\ell}[\tau],$$

   which establishes (3.2).

b) $q \in Cl(\mathcal{P}_1[\tau])$ and $\overline{q} \notin Cl(\mathcal{P}_2[\tau])$. By the induction hypothesis, we have $\textsc{contains}^i_{\mathcal{P}_1,q}[\tau] = 1$ and $\textsc{contains}^i_{\mathcal{P}_2,\overline{q}}[\tau] = 0$. An argument symmetric to the one for the preceding case can be used to show (3.2).

c) $q \in Cl(\mathcal{P}_1[\tau])$ and $\overline{q} \in Cl(\mathcal{P}_2[\tau])$ and $\mathcal{P}_1[\tau] = \top$ or $\mathcal{P}_2[\tau] = \top$. In this case $\mathcal{P}[\tau] = \top$ which has already been taken care of (see above).

d) $q \in Cl(\mathcal{P}_1[\tau])$ and $\overline{q} \in Cl(\mathcal{P}_2[\tau])$ and $\mathcal{P}_1[\tau] \neq \top$ and $\mathcal{P}_2[\tau] \neq \top$. Then $\mathcal{P}[\tau] = \mathcal{P}_1[\tau] \odot_q \mathcal{P}_2[\tau]$ and since we have restricted ourselves to the case when $\ell \in Cl(\mathcal{P})$ (see above), we have

$$\begin{aligned}\ell \in Cl(\mathcal{P}[\tau]) &\iff \ell \in Cl(\mathcal{P}_1[\tau]) \vee \ell \in Cl(\mathcal{P}_2[\tau]) \\ &\iff \textsc{contains}^i_{\mathcal{P}_1,\ell}[\tau] \vee \textsc{contains}^i_{\mathcal{P}_2,\ell}[\tau],\end{aligned}$$

where the final equivalence follows from the induction hypothesis. It is straightforward to verify that the last expression in turn is equivalent to the fourth disjunct in the definition of $\textsc{contains}^i_{\mathcal{P},\ell}$ being satisfied, and since this is the only disjunct that can be satisfied in this case, we conclude that (3.2) holds.

By Definition 11, $\mathcal{P}[\sigma] = \top$ if, and only if, $\mathcal{P}'[\sigma] = \top$, and $\ell \in Cl(\mathcal{P}[\sigma])$ if, and only if, $\ell \in Cl(\mathcal{P}'[\sigma])$, for each literal $\ell \in Cl(\mathcal{P})$. Since (3.1) and (3.2) hold for $\mathcal{P}'$ by induction hypothesis, these properties must hold for $\mathcal{P}$ as well.

Let us now turn to the proof of (3.3).

1. If $\mathcal{P}$ is an input node we have

$$u \in \mathcal{P}[\tau] \iff u \in Cl(\mathcal{P}[\tau]) \iff \textsc{contains}^i_{\mathcal{P},u}[\tau] = \textsc{polarity}_{\mathcal{P},u}[\tau]$$

by what we proved previously and the definition of $\textsc{polarity}_{\mathcal{P},u}$ for input nodes (and the fact that a literal appears in a derivation that consists of a single input node iff it occurs in the clause of associated with that node).

2. Let $\mathcal{P} = \mathcal{P}_1 \odot_q \mathcal{P}_2$.

a) $q \notin Cl(\mathcal{P}_1[\tau])$. Then $\mathcal{P}[\tau] = \mathcal{P}_1[\tau]$ and by the induction hypothesis, we have

$$u \text{ appears in } \mathcal{P}[\tau] \iff u \text{ appears in } \mathcal{P}_1[\tau] \iff \textsc{polarity}_{\mathcal{P}_1,u}[\tau] = 1.$$

Using (3.2), it is readily verified that $\textsc{polarity}_{\mathcal{P},u}[\tau] = \textsc{polarity}_{\mathcal{P}_1,u}[\tau]$.

b) $q \in Cl(\mathcal{P}_1[\tau])$ and $\overline{q} \notin Cl(\mathcal{P}_2[\tau])$. Here, (3.3) can be proved using an argument symmetric to one for the previous case.

c) $q \in Cl(\mathcal{P}_1[\tau])$ and $\overline{q} \in Cl(\mathcal{P}_2[\tau])$ and $\mathcal{P}_1[\tau] = \top$ or $\mathcal{P}_2[\tau] = \top$. Then $\mathcal{P}[\tau] = \top$, so u appears in $\mathcal{P}[\tau]$. Without loss of generality, let $\mathcal{P}_1[\tau] = \top$. By the induction hypothesis we have $\textsc{polarity}_{\mathcal{P}_1,u}[\tau] = 1$, which, along with the assumptions for this case and (3.2), implies that $\textsc{polarity}_{\mathcal{P},u}$ is satisfied by the last disjunct.

d) $q \in Cl(\mathcal{P}_1[\tau])$ and $\overline{q} \in Cl(\mathcal{P}_2[\tau])$ and $\mathcal{P}_1[\tau] \neq \top$ and $\mathcal{P}_2[\tau] \neq \top$. In this case u appears in $\mathcal{P}[\tau]$ if, and only if, it appears in $\mathcal{P}_1[\tau]$ or in $\mathcal{P}_2[\tau]$. Using the induction hypothesis and (3.2), one can verify that this is the case if, and only if, $\text{POLARITY}_{\mathcal{P},u}[\tau] = 1$.

□

These auxiliary circuits can be efficiently constructed in a top-down manner, from the input nodes to the conclusion. By a careful analysis, we obtain the following:

Lemma 4. *There is an algorithm that, given a PCNF formula Φ and an LDQ(D)-derivation $\mathcal{P}$ from Φ, computes the circuits $\text{POLARITY}_{\mathcal{P},u}$ for every universal variable u in time $O(|\mathcal{P}| \cdot n)$, where $n = |var(\Phi)|$.*

Proof. The algorithm first sorts clauses according to a fixed order of literals. Let k be the number of quantifier blocks in the prefix of Φ. There is at most one circuit $\text{TOP}^i_{\mathcal{P}}$ for each node $\mathcal{S}$ of $\mathcal{P}$ and each $1 \leq i \leq k$. Similarly, there is at most one circuit $\text{CONTAINS}^i_{\mathcal{S},\ell}$ for each node $\mathcal{S}$ of $\mathcal{P}$, each $1 \leq i \leq k$, and each literal $\ell \in Cl(\mathcal{S})$.

Once $\text{TOP}^i_{\mathcal{S}}$ has been computed for each $1 \leq i \leq k$, the circuits $\text{CONTAINS}^i_{\mathcal{S},\ell}$ can easily be constructed for each $1 \leq i \leq k$ and every literal $\ell \in Cl(\mathcal{S})$. Overall, this can be done in time

$$O(|Cl(\mathcal{S})| \cdot k) \leq O(|Cl(\mathcal{S})| \cdot n).$$

Assume that the circuits $\text{CONTAINS}^i_{\mathcal{S},\ell}$ are stored in lists following the order of literals in $Cl(\mathcal{S})$. Then for each node $\mathcal{S}$, the circuits $\text{TOP}^i_{\mathcal{S}}$ and $\text{CONTAINS}^i_{\mathcal{S},\ell}$ associated with $\mathcal{S}$ can again be computed in time $O(|Cl(\mathcal{S})| \cdot n)$, so that overall, these circuits can be computed in time $O(|\mathcal{P}| \cdot n)$ for all nodes of $\mathcal{P}$. Having computed the circuits CONTAINS and TOP, the circuits $\text{POLARITY}_{\mathcal{S},u}$ can be computed for each node $\mathcal{S}$ and each universal variable $u \in var_\forall(\Phi)$ in time $O(|\mathcal{P}| \cdot n)$. □

Using Lemma 2, we can spell out the argument sketched at the beginning of this section and prove that for normal dependency schemes D, the universal player can maintain an LDQ(D)-refutation throughout the evaluation game by successively restricting an initial LDQ(D)-refutation by both players' moves and assigning universal variables from the leftmost remaining block X so as to falsify the (unique) literals from X remaining in the refutation. Lemma 3 tells us that the POLARITY circuits can be used to implement this strategy. In order to put things together, we will need the following two lemmas, which tell us that successive restriction and bulk restriction in fact yield the same result.

Lemma 5. *Let $\mathcal{P}$ be an LDQ(D)-derivation from a PCNF formula Φ, let τ_1, τ_2 be two assignments to disjoint sets of variables. Then $\mathcal{P}[\tau_1][\tau_2] = \mathcal{P}[\tau_1 \cup \tau_2]$.*

Proof. By induction on the structure of the derivation. If $\mathcal{P}$ is an input node, we have $Cl(\mathcal{P}[\tau]) = Cl(\mathcal{P})[\tau] = Cl(\mathcal{P})[\tau_1][\tau_2] = Cl(\mathcal{P}[\tau_1][\tau_2])$ and since both derivations consist of a single node with the same label, they are in fact equal. For derivations created by the operations, the equality is trivially preserved. □

Lemma 6. *Let D be a normal dependency scheme, let $\Phi = Q_1 X_1 \dots Q_k X_k.\varphi$ be a PCNF formula, let $\mathcal{P}$ be an LDQ(D)-refutation of Φ. Let X_i be a universal quantifier block and let $\tau : \bigcup_{j=1}^{i-1} X_j \to \{0,1\}$ be an assignment. If $\mathcal{P}[\tau]$ is an LDQ(D)-refutation of $\Phi[\tau]$, then $\mathcal{P}[\tau \cup \sigma]$ is an LDQ(D)-refutation of $\Phi[\tau \cup \sigma]$, where $\sigma : X_i \to \{0,1\}$ is the assignment such that $\sigma(u) = \neg\text{POLARITY}_{\mathcal{P},u}[\tau]$ for each $u \in X_i$.*

Proof. Assume $\mathcal{P}[\tau]$ is an LDQ(D)-refutation of $\Phi[\tau]$. Let $u \in X_i$. Because D is simple, variable u appears in $\mathcal{P}[\tau]$ in at most one polarity. If u does not appear in $\mathcal{P}[\tau]$ at all, the restriction $\mathcal{P}[\tau][\sigma]$ does not depend on $\sigma(u)$. Otherwise, there is a unique literal ℓ with $var(\ell) = u$ that appears in $\mathcal{P}[\tau]$. By Lemma 3, $\text{POLARITY}_{\mathcal{P},u}[\tau] = 1$ iff u appears in $\mathcal{P}[\tau]$, so $\sigma(u) = \neg\text{POLARITY}_{\mathcal{P},u}[\tau] = 0$ if $\ell = u$ and $\sigma(u) = 1$ if $\ell = \neg u$. It is a straightforward consequence that $\mathcal{P}[\tau][\sigma]$ can be obtained from $\mathcal{P}[\tau]$ by deleting every occurrence of a variable $u \in X_i$ and omitting instances of $\forall$-reduction that become redundant as a result. Because D is monotone, the restriction $\mathcal{P}[\tau][\sigma]$ is an LDQ(D)-refutation of $\Phi[\tau \cup \sigma]$, and $\mathcal{P}[\tau][\sigma] = \mathcal{P}[\tau \cup \sigma]$ by Lemma 5. □

With that, we are ready to prove the final statement.

Lemma 7. *Let D be a normal dependency scheme, let $\mathcal{P}$ be an LDQ(D)-refutation of a PCNF formula Φ. Then the family $\{f_u\}_{u \in var_\forall(\Phi)}$ of functions $f_u = \neg\text{POLARITY}_{\mathcal{P},u}$ is a countermodel of Φ.*

Proof. Let $\Phi = Q_1 X_1 \dots Q_k X_k.\varphi$ and let $\tau : var(\Phi) \to \{0,1\}$ be a truth assignment such that $\tau(u) = f_u\left(\tau|_{L_\Phi(u)}\right)$ for each universal variable u. Let $X_{<i} = \bigcup_{j=1}^{i-1} X_j$, and let $\tau_i = \tau|_{X_{<i}}$ for each $1 \leq i \leq k+1$. We claim that $\mathcal{P}[\tau_i]$ is an LDQ(D)-refutation of $\Phi[\tau_i]$ for $1 \leq i \leq k+1$. The assignment τ_1 is empty so $\mathcal{P}[\tau_1] = \mathcal{P}$ and $\Phi[\tau_1] = \Phi$ so the statement holds in that case. Suppose the claim holds for i such that $1 \leq i \leq k$. If $Q_i = \exists$, then $\mathcal{P}[\tau_i][\tau|_{X_i}]$ is an LDQ(D)-refutation of $\Phi[\tau_{i+1}]$ by Lemma 2, and $\mathcal{P}[\tau_i][\tau|_{X_i}] = \mathcal{P}[\tau_{i+1}]$ by Lemma 5. Otherwise, $Q_i = \forall$ and $\mathcal{P}[\tau_{i+1}]$ is an LDQ(D)-refutation of $\Phi[\tau_{i+1}]$ by Lemma 6. This completes the proof of the claim. In particular, we now have that $\mathcal{P}[\tau_{k+1}] = \mathcal{P}[\tau]$ is an LDQ(D)-refutation of $\Phi[\tau_{k+1}] = \Phi[\tau]$. Because $\Phi[\tau]$ does not contain any variables, the only way $\Phi[\tau]$ can have a refutation is that its matrix contains the empty clause, which means that $\varphi[\tau] = \{\emptyset\}$. □

Theorem 2 follows immediately from Lemma 4 and Lemma 7.

3.4.1 The Reflexive Resolution-Path Dependency Scheme is Normal

In order to prove Theorem 3 and show that $\mathrm{D}^{\mathrm{rrs}}$ is normal, we will need some insight into the relationship between resolution paths and $\mathrm{LDQ}(\mathrm{D}^{\mathrm{rrs}})$-derivations. For a formula Φ and a universal literal u, we will denote by $T_u(\Phi)$ the set of existential literals e such that $u \prec^q_\Phi e$ and e is reachable from u by a proper resolution path. Since in the following we consider u from the leftmost quantifier block, the condition for properness means just that the connecting variables are existential.

Lemma 8. *Let $\Phi = \forall X \mathcal{Q}.\varphi$ be a PCNF formula and let u be a universal literal with $var(u) \in X$. Let $C_1, C_2 \in \varphi$ be clauses such that for some existential literal x, $x \in C_1$ and $\overline{x} \in C_2$, and let $C = C_1 \cup C_2 \setminus \{x, \overline{x}\}$. Then $T_u(\Phi) = T_u(\mathcal{Q}.\varphi \cup \{C\})$.*

Proof. Let $\Phi' = \mathcal{Q}.\varphi \cup \{C\}$. Of course, by adding clauses to a formula, we preserve all existing resolution paths, so $T_u(\Phi) \subseteq T_u(\Phi')$. We will prove that the opposite inclusion holds as well. Let $e \in T_u(\Phi')$ and let π be a resolution path in Φ' certifying this. If π is also a resolution path in Φ, we are done. If it is not, it must be because it performs a *C-transition*, namely it contains two consecutive literals l_1, l_2 such that $var(l_1) \neq var(l_2)$, $l_1, l_2 \in C$, but $\{l_1, l_2\} \not\subseteq C_1$ and $\{l_1, l_2\} \not\subseteq C_2$. In this case, without loss of generality, we have $l_1 \in C_1$ and $l_2 \in C_2$. Let π_1 be the prefix of π up to and including l_1 and π_2 be the suffix of π starting with l_2. Let π' be the concatenation of π_1, x, $\overline{x}$, and π_2. It is clearly a valid resolution path and it uses one fewer C-transitions than π. Iterating this process, we can remove all C-transitions from π to obtain a resolution path in Φ. The resulting resolution path has the same endpoints and therefore certifies that $e \in T_u(\Phi)$. □

The previous lemma implies that when considering reachability from an outermost universal literal in a formula Φ, we can use clauses derived from Φ by $\mathrm{LDQ}(\mathrm{D}^{\mathrm{rrs}})$-resolution as well. Indeed, adding clauses produced by the resolution rule does not change the set of reachable literals by Lemma 8, and adding clauses produced by universal reduction clearly does not even create new resolution paths. Particularly, if two literals ever appear together in a derived clause, there is a resolution path between them. This is summarized by the following corollary.

Corollary 4. *Let $\mathcal{P}$ be an $\mathrm{LDQ}(\mathrm{D}^{\mathrm{rrs}})$-derivation from a PCNF formula $\Phi = \forall X \mathcal{Q}.\varphi$ and let $u \in X, u \in Cl(\mathcal{P})$. Then for all existential literals $e \in Cl(\mathcal{P})$, there is a resolution path from u to e in Φ.*

As a first step towards proving Theorem 3, we will prove that both polarities of an outermost universal literal cannot appear together in a single clause of a derivation.

Lemma 9. *Let $\mathcal{P}$ be an $\mathrm{LDQ}(\mathrm{D}^{\mathrm{rrs}})$-derivation from a PCNF formula $\Phi = \forall X \mathcal{Q}.\varphi$ and let $u \in X$. Then $u \notin Cl(\mathcal{P})$ or $\neg u \notin Cl(\mathcal{P})$.*

Proof. Towards a contradiction, suppose $u, \neg u \in Cl(\mathcal{P})$. Since input clauses do not contain both polarities of any literal, there must be a resolution step inside the derivation,

which merges u and $\neg u$ into one clause. Let $\mathcal{P}' = \mathcal{P}_1 \odot_x \mathcal{P}_2$ be such a step. Then, without loss of generality, $x, u \in Cl(\mathcal{P}_1)$ and $\neg x, \neg u \in Cl(\mathcal{P}_2)$ and by Corollary 4, there is a resolution path from u to x and from $\neg u$ to $\neg x$, i.e. $(u, x) \in \mathrm{D}^{\mathrm{rrs}}_{\Phi}$. However, if x depends on u, opposite polarities of u cannot be merged in a resolution step with the pivot x, a contradiction. □

We will further assume that the derivation considered in the proof of Theorem 3 is in the normal form given by the following lemma.

Lemma 10. *Let $\mathcal{P}$ be an LDQ($\mathrm{D}^{\mathrm{rrs}}$)-derivation from a PCNF formula $\Phi = \forall X \mathcal{Q}.\varphi$, let $u \in X$ be a universal variable such that both u and $\neg u$ appear in $\mathcal{P}$, and let $Y = var_{\exists}(\Phi)$. There exists an LDQ($\mathrm{D}^{\mathrm{rrs}}$)-derivation $\mathcal{P}'$ from a PCNF formula $\Phi' = \forall u \exists Y.\varphi'$ such that $\mathcal{P}'$ contains both u and $\neg u$ but only $\forall$-reduction steps with respect to one polarity of u.*

Proof. Let φ' be the result of removing all universal variables except u from φ. Removing universal variables does not introduce new resolution paths, so $\mathrm{D}^{\mathrm{rrs}}_{\Phi'} \subseteq \mathrm{D}^{\mathrm{rrs}}_{\Phi}$. The derivation $\mathcal{P}'$ can be obtained from $\mathcal{P}$ in the following way. We first delete all occurrences of universal variables other than u from $\mathcal{P}$, along with $\forall$-reduction steps involving such variables. The result is an LDQ($\mathrm{D}^{\mathrm{rrs}}$)-derivation from Φ', and it still contains both u and $\neg u$. Next, we choose a subderivation containing both u and $\neg u$ such that none of its proper subderivations contains both literals. By Lemma 9, the conclusion C of this derivation can contain at most one of u and $\neg u$. If $u \in C$ we omit all $\forall$-reduction steps involving u. Otherwise, we omit all $\forall$-reduction steps involving $\neg u$. This yields the desired LDQ($\mathrm{D}^{\mathrm{rrs}}$)-derivation $\mathcal{P}'$ from Φ'. □

Using Lemma 9 and Lemma 10, we can proceed to finish the proof of Theorem 3.

Proof of Theorem 3. Towards a contradiction, consider an LDQ($\mathrm{D}^{\mathrm{rrs}}$)-derivation from a formula $\Phi = \forall X \mathcal{Q}.\varphi$ and let $u \in X$ be such that both polarities of u occur in this derivation. Let $Y = var_{\exists}(\Phi)$ and let $\mathcal{P}$ denote the simplified derivation given by Lemma 10. Assume without loss of generality that $\mathcal{P}$ is a tree (any derivation can be turned into a tree-like derivation by copying proof nodes), and that $\mathcal{P}$ does not contain $\forall$-reduction steps involving $\neg u$. Since $\neg u$ occurs in $\mathcal{P}$ but is not reduced, $\neg u$ must occur in the conclusion of $\mathcal{P}$. Thus u cannot occur in the conclusion by Lemma 9. Since u is present in the derivation $\mathcal{P}$, this means there must be a reduction step on u somewhere in $\mathcal{P}$. As u is the only universal variable and we omitted all reduction steps on $\neg u$, all reduction steps in $\mathcal{P}$ are on u and $\mathcal{P}$ must have the form depicted in Figure 3.3, where $\mathcal{P}_n = \mathcal{P}_{n+1} - u$ is a lowermost reduction step on u and the subsequent resolutions are on pivots $x_n, \ldots, x_1$. Let $C_0 = Cl(\mathcal{P}), C_i = Cl(\mathcal{P}_i), C'_i = Cl(\mathcal{P}'_i)$. The clauses $C'_1, \ldots, C'_n, C_{n+1}$ are derived by LDQ($\mathrm{D}^{\mathrm{rrs}}$)-resolution and by Lemma 8 we know that we can use them to show resolution-path connections as if they were input clauses. By the transformations we considered we know that starting from an arbitrary LDQ($\mathrm{D}^{\mathrm{rrs}}$)-derivation we can obtain a valid

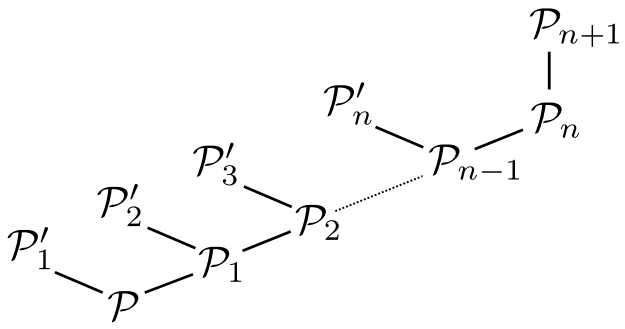

Figure 3.3: Shape of the derivation constructed in the proof of Theorem 3.

LDQ($\mathrm{D}^{\mathrm{rrs}}$)-derivation in this form, so any contradiction we derive from here means a contradiction with the assumption that an LDQ($\mathrm{D}^{\mathrm{rrs}}$)-derivation contains both polarities of a universal variable from the outermost block, thus proving Theorem 3. With that, we are ready to finish the proof.

We will prove that there is a resolution path from $\neg u$ to u going through an existential literal in C_{n+1}, which is in contradiction with the soundness of reduction of u from C_{n+1}. Let us consider *open* resolution paths, i.e. resolution paths without their final literal. If an open resolution path ends in a literal ℓ of clause C, we say that the path *leads* to the clause C. By induction on n, we will prove that there is an open resolution path from $\neg u$ which leads to the clause C_n. If $n = 1$, we have the path $\neg u, \neg x_1, x_1$. For $n > 1$, let π be the open path leading to C_{n-1} and let ℓ be its last literal. Then either $\ell \in C_n$, in which case we have an open path leading to C_n, or $\ell \in C'_n$, in which case we have the open path $\pi, \neg x_n, x_n$ leading to C_n. An open path that leads to C_n also leads to C_{n+1}, because those two clauses only differ in the presence of u and therefore can be closed by the literal u to obtain the required resolution path. □

3.5 Experiments

To gauge the potential of clause learning based on LDQ($\mathrm{D}^{\mathrm{std}}$), we ran experiments with the search-based solver DepQBF[4] in version 5.0. By default, DepQBF supports proof generation only in combination with the trivial dependency scheme—in that case, it generates Q-resolution or long-distance Q-resolution proofs (depending on whether long-distance resolution is enabled). However, by uncommenting a few lines in the source code, proof generation can also be enabled with the standard dependency scheme, and this option can even be combined with long-distance resolution. This leads to the solver generating Q($\mathrm{D}^{\mathrm{std}}$)-resolution or LDQ($\mathrm{D}^{\mathrm{std}}$)-resolution proofs (see Section 3.2).

We compared the performance of DepQBF in four configurations,[5] each using a different

[4] `http://lonsing.github.io/depqbf/`

[5] As a sanity check, we verified that all configurations that were able to solve a particular instance returned the same result.

proof system for constraint learning:

1. Long-distance Q-resolution with ∀/∃-reduction according to D^{std} (LDQD).
2. Long-distance Q-resolution with ordinary ∀/∃-reduction (LDQ).
3. Q-resolution with ∀/∃-reduction according to D^{std} (QD).
4. Ordinary Q-resolution (Q).

These experiments were performed on a cluster with Intel Xeon E5649 processors at 2.53 GHz running 64-bit Linux. We set time and memory limits of 900 seconds and 4 GB, respectively. Instances were taken from two tracks of the QBF Gallery 2014: the *applications* track consisting of 6 instance families and a total of 735 formulas, and the *QBFLib* track consisting of 276 formulas.

For our first set of experiments, we disabled dynamic QBCE (Quantified Blocked Clause Elimination), a technique introduced with version 5.0 of DepQBF [LBB+15]. We further used bloqqer[6] (version 037) with default settings as a preprocessor. Since LDQ(D^{std}) generalizes both long-distance Q-resolution and Q(D^{std})-resolution, we were expecting a performance increase with LDQ(D^{std})-learning compared to learning based on the other proof systems. However, all four configurations showed virtually identical performance on both the application and QBFlib benchmark sets in terms of total runtime and instances solved within the time limit (see Table 3.1).

Application track

Configuration	Solved	True	False	Time
LDQD	377	186	191	343455
LDQ	377	186	191	345459
QD	377	183	194	343928
Q	376	182	194	345914

QBFLib track

Configuration	Solved	True	False	Time
LDQD	130	69	61	140743
LDQ	131	69	62	141646
QD	129	67	62	140975
Q	127	65	62	142679

Table 3.1: Solved instances, solved true instances, solved false instances, and total runtime in seconds (including timeouts) with preprocessing (but without QBCE).

[6] http://fmv.jku.at/bloqqer/

To get a more detailed picture, we broke down the results for the application track by instance family, limiting ourselves to instances that were solved by at least one configuration. The barplot in Figure 3.4 shows that there are considerable differences in performance between solver configurations for individual instances families, with each solver configuration being outperformed by another configuration on at least one family.

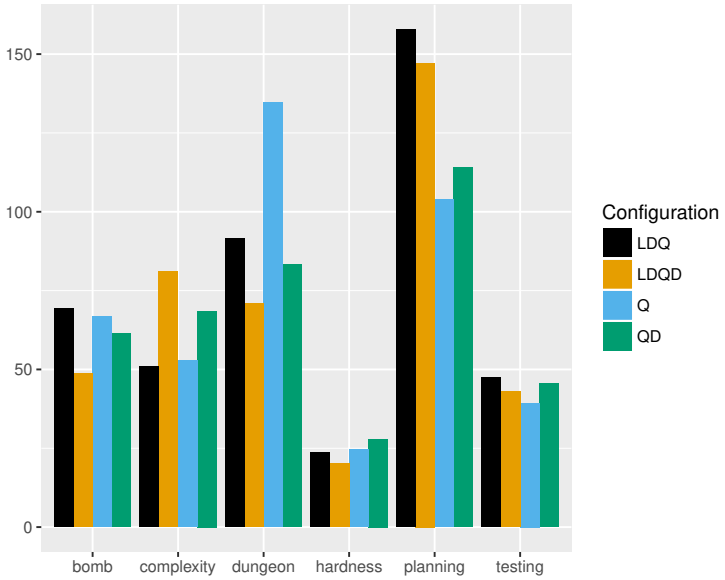

Figure 3.4: Average runtime in seconds (y-axis) for instances from the application track for each instance family (x-axis), by solver configuration (with preprocessing, but without dynamic QBCE). Here, we only considered instances that were solved by at least one configuration.

For our second set of experiments, we turned on dynamic QBCE. This led to a significant performance increase both in terms of number of instances solved within the time limit and total runtime for both benchmark sets, a result that is consistent with the findings in [LBB+15]. However, as far as the performance of LDQ(D^{std})-learning is concerned, the application and QBFlib tracks differed significantly for this experiment. While LDQ(D^{std})-learning fared *worst* among the configurations both with respect to instances solved and total runtime on the application track, it was the *best* configuration for the QBFlib track in both respects (see Table 3.2). Figure 3.5 shows that using the standard dependency scheme was beneficial both with and without long-distance resolution for the QBFlib instances.

For our final set of experiments, we left dynamic QBCE enabled but *disabled* preprocessing for the application track, as this was shown to lead to a performance *increase* in the case of learning with ordinary Q-resolution [LBB+15]. Indeed, this resulted in a performance increase across the board (see Table 3.3). Moreover, LDQ(D^{std})-learning was the best configuration in terms of instances solved (on par with Q(D^{std})-resolution) as well as in

Application track

Configuration	Solved	True	False	Time
LDQD	385	195	190	339143
LDQ	388	201	187	336739
QD	392	201	191	334965
Q	389	198	191	337141

QBFLib track

Configuration	Solved	True	False	Time
LDQD	145	75	70	132567
LDQ	133	64	69	141682
QD	137	70	67	134150
Q	129	62	67	142399

Table 3.2: Results with preprocessing and dynamic QBCE.

Configuration	Solved	True	False	Time
LDQD	440	223	217	287012
LDQ	435	223	212	291574
QD	440	225	215	291661
Q	437	221	216	337141

Table 3.3: Results for the application track with QBCE (but without preprocessing).

terms of overall runtime. Moreover, LDQ(D^{std})-learning was the best configuration in terms of instances solved (on par with Q(D^{std})-resolution) as well as in terms of overall runtime.

3.6 Related Work

QCDCL with learning based on long-distance Q-resolution was first described by Zhang and Malik [ZM02a]. They presented an argument for the soundness of using tautological clauses (respectively, contradictory terms) within their algorithm but did not study long-distance Q-resolution as a proof system. Lacking a sound theoretical foundation, the use of tautological clauses in QCDCL was abandoned in favour of more complicated methods for constraint learning that avoid their generation [GNT06, Gel12, LEVG13].

Interest in long-distance Q-resolution was renewed when Balabanov and Jiang [BJ12] introduced the proof system we presented in Section 3.1 (restricted to the trivial dependency scheme) and proved its soundness. Egly, Lonsing, and Widl [ELW13] showed that a family of formulas known to be hard for Q-resolution [KKF95] admits short

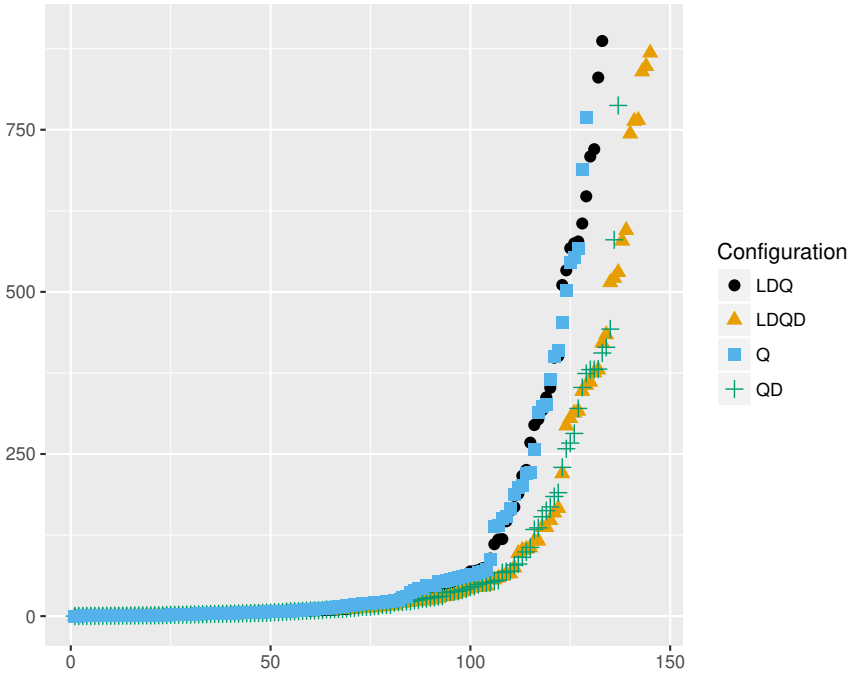

Figure 3.5: Solved instances from the QBFLib track (x-axis) sorted by runtime (y-axis), by solver configuration (with preprocessing and dynamic QBCE).

long-distance Q-resolution proofs. They also demonstrated that QCDCL with learning based on long-distance resolution generates long-distance Q-resolution proofs and presented a new version of DepQBF that implements this algorithm. Finally, they showed that a long-distance Q-resolution proof can be interpreted as a winning strategy in the evaluation game associated with a QBF, generalizing a result by Goultiaeva, Van Gelder, and Bacchus [GVGB11]. Even though these results established a solid theoretical framework for the use of long-distance Q-resolution within QCDCL, they were regarded as unsatisfactory by some since they did not provide an intuitive account of the semantics of individual tautological clauses. Such an account was subsequently presented by Balabanov, Janota, Jiang, and Widl [BJJW15], who showed that tautological literals $u, \neg u \in C$ can be interpreted as proxies for "phase functions" that determine whether a variable or its negation is present in clause C based on the values assigned to pivot variables appearing in the derivation of C. The authors used this interpretation to generalize the linear-time strategy extraction algorithm of Balabanov and Jiang [BJ12] to long-distance Q-resolution proofs.

Recently and independently of this work, Beyersdorff and Blinkhorn investigated the soundness of Q-resolution proof systems parameterized by dependency schemes [BB16]. They define a property of dependency schemes D—*full exhibition*—which ensures that a certain version of long-distance Q(D)-resolution is sound, and show that the reflexive

resolution-path dependency scheme has that property.

In a nutshell, a dependency scheme D is fully exhibited if every true QBF Φ has a model $\{f_e\}_{e \in var_\exists(\Phi)}$ such that f_e may only depend on a universal variable u if $(u, e) \in D_\Phi$ (such models have elsewhere been referred to as *D-models* [Sli15]). It is fairly straightforward to show that Q(D)-resolution is sound if D has this property, but generalizing this result to proof systems with long-distance resolution presents a challenge. Beyersdorff and Blinkhorn show that full exhibition is sufficient for soundness of a restricted version of LDQ(D)-resolution, where complementary universal literals that are "merged" by resolution must be annotated with the (existential) pivot variable, and universal reduction can be applied only if every existential variable occurring in the premise or the annotation of a universal variable is independent of the universal variable to be reduced. However, it is uncertain whether proofs generated by DepQBF with LDQ(D)-learning satisfy this additional restriction.

How full exhibition relates to our normality property is not entirely clear. Beyersdorff and Blinkhorn prove that full exhibition is *not* sufficient for soundness of LDQ(D)-resolution as defined here. In combination with Theorem 2, this shows that dependency schemes that are fully exhibited need not be normal. Whether there are normal dependency schemes that are not fully exhibited, on the other hand, remains open. Indeed, there is some evidence to the effect that normality entails full exhibition: consider a dependency scheme D that is not fully exhibited, and let $\Phi = \forall u \mathcal{Q}.\varphi$ be a true QBF that does not have a D-model. Suppose u is the only universal variable of Φ. In this restricted case, the (non-)existence of a D-model can be expressed as a QBF Ψ by simply shifting existentials independent of u to the left. Because Φ does not have a D-model, Ψ must be false and admit a Q-resolution refutation $\mathcal{P}$, which is also an LDQ(D)-refutation of Φ. Because Φ is true, LDQ(D)-resolution must be unsound and so D cannot be normal by Theorem 2. Obviously, the assumption that u is the only universal variable of Φ is very restrictive, but since we can suppose that D is monotone (recall that a dependency scheme is normal if it is both simple and monotone), there is hope that the argument for an arbitrary QBF can be reduced to this case by instantiating with a suitable variable assignment.

3.7 Summary and Discussion

We have defined LDQ(D)-resolution, the proof system that combines long-distance Q-resolution with dependency schemes. We defined normality of a dependency scheme, and showed that if D is normal, then LDQ(D)-resolution is sound and admits polynomial-time strategy extraction. As an application of this general result, we showed that D^{rrs} is normal.

The results of Section 3.2 and Section 3.3 establish a partial soundness proof of QCDCL with learning based on LDQ(D^{std}): we now know that we can trust such a solver when it outputs "false". To prove that "true" answers can be trusted as well, one has to show soundness of quantified term resolution (Q-consensus) when combined with the standard dependency scheme and long-distance resolution. The reason this does not follow from

the results proved here is that they rely on a correspondence of Q-resolution derivations with dependency-inducing resolution paths that is not immediate for terms generated from an input PCNF: there is a correspondence of Q-consensus derivations with dual "resolution paths" that connect such terms, but these "resolution paths" do not induce resolution-path dependencies in the input PCNF.

The experiments in Section 3.5 indicate that we should not expect significant performance gains when switching from learning with Q($\mathrm{D}^{\mathrm{std}}$)-resolution to LDQ($\mathrm{D}^{\mathrm{std}}$). This is in spite of the fact that, from a purely theoretical perspective, LDQ($\mathrm{D}^{\mathrm{std}}$) is a stronger proof system: a well-studied class of QBFs introduced by Kleine Büning, Karpinski, and Flögel requires exponentially-sized Q-resolution proofs [KKF95] but admits short long-distance Q-resolution refutations [ELW13], and since the standard dependency scheme does not offer any improvement over trivial dependencies on these formulas (see [BB17]) we obtain an exponential separation of LDQ($\mathrm{D}^{\mathrm{std}}$)-resolution from Q($\mathrm{D}^{\mathrm{std}}$)-resolution. From a practical point of view, the main benefit of using LDQ($\mathrm{D}^{\mathrm{std}}$)-resolution over Q($\mathrm{D}^{\mathrm{std}}$)-resolution is that conflict analysis is much simpler (cf. [ELW13]). A learned constraint can be obtained from a conflict simply by resolving variables in the reverse order of their propagation (see Section 3.2). Methods that avoid the generation of tautological clauses (contradictory terms) during learning are significantly more involved [GNT06, Gel12, LEVG13].

We have shown that LDQ($\mathrm{D}^{\mathrm{rrs}}$)-refutations allow for polynomial-time strategy extraction. In practice, the corresponding algorithm generates circuits that are frequently larger by an order of magnitude than the refutations provided as input. It is unclear whether this increase in size can be avoided by careful engineering alone or only by using a different approach. Faster (linear time) strategy extraction algorithms are known for "ordinary" Q-resolution and long-distance Q-resolution [BJ12, BJJW15]. Unfortunately, their underlying idea of setting universal variables so as to falsify the premise of some $\forall$-reduction step no longer works when dependency schemes enter the mix: generalized $\forall$-reduction may remove a universal variable u even in the presence (in the premise) of an existential variable e such that $u < e$ and the universal player can only be sure to falsify the premise if the e-literal is false, but she does not know the value of e at the time of assigning u. We believe that developing linear-time strategy extraction algorithms for Q(D)-resolution or LDQ(D)-resolution is going to require a better understanding of the power of these proof systems vis-à-vis Q-resolution and long-distance Q-resolution [BB17].

Publication Notes

The research in this chapter appeared in a paper published in the proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing (SAT 2016) [PSS16], and in full in a paper published in Volume 63 (2019) of the Journal of Automated Reasoning [PSS19b].

CHAPTER 4

Dependency Learning for QBF

We have seen how the prefix ordering $\prec^q$ constrains QCDCL in its selection of branching variables as well as in the application of unit propagation. Dependency schemes can be used to grant the solver additional freedom, however, with greater freedom comes greater responsibility—in this case to ensure that the dependency scheme is sound. In Chapter 3 we gave a sufficient condition for a dependency scheme D in order for LDQ(D)-resolution to be sound and admit polynomial-time strategy extraction. The argument why D^{rrs} satisfies this condition, and thus LDQ(D^{rrs})-resolution is sound, was tedious and long-winded. This is an inherent weakness of dependency schemes—each new dependency scheme or integration with another solving technique, such as long-distance Q-resolution, requires a new soundness proof. Moreover, even though we established that LDQ(D^{rrs})-resolution admits polynomial-time strategy extraction, the algorithm is not as practical as its counterpart for long-distance Q-resolution, and would most likely be unsuitable for an application where the strategy is of vital importance.

In this chapter we describe a different, novel approach to dependency analysis called *dependency learning*. Dependency learning is a modification of QCDCL which allows the solver to exploit independence between variables while retaining long-distance Q-resolution as its underlying proof system, enjoying its soundness and fast strategy extraction without the need for additional arguments. As we can observe from the experimental evidence, dependency learning appears to identify far more independent pairs of variables than even the reflexive resolution-path dependency scheme, providing yet another argument in its favor. Compared to dependency schemes, it is also much simpler to implement, requiring virtually no special data structures, other than an adaptation of a watched-literal scheme, which is anyway already present in a QCDCL solver.

Algorithm 3 QCDCL with Dependency Learning

```
1:  procedure SOLVE( )
2:      D = ∅
3:      while TRUE do
4:          conflict = QBCP()
5:          if conflict == NONE then
6:              DECIDE()
7:          else
8:              constraint, btlevel = ANALYZECONFLICT(conflict)
9:              if constraint != NONE then
10:                 if ISEMPTY(constraint) then
11:                     return ISTERM(constraint)
12:                 else
13:                     ADDLEARNEDCONSTRAINT(constraint)
14:                 end if
15:             end if
16:             BACKTRACK(btlevel)
17:         end if
18:     end while
19: end procedure
```

4.1 QCDCL with Dependency Learning

Similarly to depednency schemes, for dependency learning the solver maintains a set $D \subseteq \prec^q$ of variable dependencies. Both QBCP and the decision rule must be modified as follows:

- QBCP() uses universal and existential reduction relative to D. Universal reduction relative to D removes each universal variable u from a clause C such that there is no existential variable $e \in var(C)$ with $(u, e) \in D$ (existential reduction relative to D is defined dually).
- DECIDE() may assign any variable y such that there is no unassigned variable x with $(x, y) \in D$ (note that $(x, y) \in D$ implies $x \prec^q y$).

This is how DepQBF uses the dependency relation D computed by a dependency scheme in propagation and decisions [BL10]. Unlike DepQBF, QCDCL with dependency learning does *not* use the generalized reduction rules during conflict analysis (RESOLVE and REDUCE in lines 7 and 8 refer to resolution and reduction as defined in Figure 2.2). As a consequence, the algorithm cannot always construct a learned constraint during conflict analysis (see Example 4 below). Such cases are dealt with in lines 9 through 12 of ANALYZECONFLICT (Algorithm 4):

- EXISTSRESOLVENT(*constraint*, *reason*, *pivot*) determines whether the resolvent of *constraint* and *reason* exists.

Algorithm 4 Conflict Analysis with Dependency Learning

```
1:  procedure analyzeConflict(conflict)
2:      constraint = getConflictConstraint(conflict)
3:      while not asserting(constraint) do
4:          pivot = getPivot(constraint)
5:          reason = getAntecedent(pivot)
6:          if existsResolvent(constraint, reason, pivot) then
7:              constraint = resolve(constraint, reason, pivot)
8:              constraint = reduce(constraint)
9:          else
10:             illegal_merges = illegalMerges(constraint, reason, pivot)
11:             D = D ∪ { (v, pivot) : v ∈ illegal_merges }
12:             return none, decisionLevel(pivot)
13:         end if
14:     end while
15:     btlevel = getBacktrackLevel(constraint)
16:     return constraint, btlevel
17: end procedure
```

- If this is not the case, there has to be a variable v (universal for clauses, existential for terms) satisfying the following condition: $v \prec^q \mathit{pivot}$ and there exists a literal $\ell \in \mathit{constraint}$ with $\mathit{var}(\ell) = v$ and $\bar{\ell} \in \mathit{reason}$. The set of such variables is computed by illegalMerges. For each such variable, a new dependency is added to D. No learned constraint is returned by conflict analysis, and the backtrack level (*btlevel*) is set so as to cancel the decision level at which *pivot* was assigned.

The criteria for a constraint to be asserting must also be slightly adapted: a clause (term) S is asserting with respect to D if there is a unique existential (universal) literal $\ell \in S$ with maximum decision level among literals in S, its decision level is greater than 0, the corresponding decision variable is existential (universal), and every universal (existential) variable $y \in \mathit{var}(S)$ such that $(y, \mathit{var}(\ell)) \in D$ is assigned (again, this corresponds to the definition of asserting constraints used in DepQBF [Lon12, p.119]). Finally, in the main QCDCL loop, we have to implement a case distinction to account for the fact that conflict analysis may not return a constraint (line 9 in Algorithm 3).

Example 4. We revisit the PCNF formula Φ from Example 1 to illustrate a run of QCDCL with dependency learning.

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

Initially, the set D of learned dependencies is empty. Accordingly, universal reduction relative to D would simplify the first clause to $(\neg y)$ and the third clause to (z). The algorithm thus assigns $y = 0$ and $z = 1$, falsifying the second clause. Conflict analysis first

resolves the second clause with the third clause (which is responsible for propagating the last literal in the falsified clause) to obtain the clause $(\neg x \vee y)$. Since universal reduction is performed according to the prefix order during conflict analysis, the literal $\neg x$ cannot be removed from this clause even though there is no dependency of y on x. Next, conflict analysis attempts to resolve $(\neg x \vee y)$ with $(x \vee \neg y)$. These clauses do not have a resolvent in long-distance Q-resolution since $x \prec^q y$. Variable x is identified as involved in an illegal merge, the dependency (x, y) is added to D, and the solver backtracks before decision level 0 (the level where $\neg y$ was propagated), undoing all assignments. Because of the learned dependency $(x, y) \in D$ the first clause can no longer be simplified by universal reduction, but the third clause still simplifies to (z) and $z = 1$ is propagated. This simplifies the second clause to (y) and $y = 1$ is propagated. Thus the first clause becomes (x) and universal reduction results in a conflict. Conflict analysis resolves the first and second clause to derive $(x \vee \neg z)$. Because $x \prec^q z$ universal reduction cannot simplify this clause. Conflict analysis then attempts to resolve $(x \vee \neg z)$ and $(\neg x \vee z)$. These clauses do not have a resolvent and x is identified as the cause of an illegal merge. The dependency (x, z) is added to D and the solver backtracks to remove decision level 0 where z was propagated. Since $D = \{(x, y), (x, z)\}$ now contains all possible dependencies of Φ, QCDCL with dependency learning behaves exactly like ordinary QCDCL from that point onward.

4.2 Soundness and Termination

Soundness of QCDCL with dependency learning is an immediate consequence of the following observation.

Observation 1. *Every constraint learned by QCDCL with dependency learning can be derived from the input formula by long-distance Q-resolution or Q-consensus.*

To prove termination, we argue that the algorithm learns a new constraint or a new dependency after each conflict. Just as in QCDCL, every learned constraint is asserting, so learning does not introduce duplicate constraints.

Observation 2. *QCDCL with dependency learning never learns a constraint already present in the database.*

The only additional argument required to prove termination is one that tells us that the algorithm cannot indefinitely "learn" the same dependencies.

Lemma 11. *If QCDCL with dependency learning does not learn a constraint during conflict analysis, it learns a new dependency.*

Proof. To simplify the presentation, we are only going to consider clause learning (the proof for term learning is analogous). We first establish an invariant of intermediate clauses derived during conflict analysis: they are empty under the partial assignment

obtained by backtracking to the last literal in the assignment that falsifies an existential literal in the clause. Formally, let C be a clause and let $\sigma = (\ell_1, \ldots, \ell_k)$ be an assignment. We define $last_C(\sigma) = \max(\{\, i : 1 \leq i \leq k \text{ and } \overline{\ell_i} \in C, var(\ell_i) \in E \,\} \cup \{0\})$, where E is the set of existential variables of the input PCNF formula, and let $\sigma_C = (\ell_1, \ldots, \ell_{last_C(\sigma)})$. In particular, if $last_C(\sigma) = 0$ then σ_C is empty.

We now prove the following claim: if σ is an assignment that falsifies a clause, then, for every intermediate clause C constructed during conflict analysis, $C[\sigma_C]$ is empty after universal reduction. The proof is by induction on the number of resolution steps in the derivation of C. If C is the conflict clause, then $C[\sigma]$ reduces to the empty clause. That means $C[\sigma_C]$ can only contain universal literals and can also be reduced to the empty clause by universal reduction. Suppose C is the result of resolving clauses C' and R and applying universal reduction, where C' is an intermediate clause derived during conflict analysis and R is a clause that triggered unit propagation. The induction hypothesis tells us that $C'[\sigma_{C'}]$ reduces to the empty clause. Since the pivot literal ℓ is chosen to be the last existential literal falsified in C', we must have $\sigma_{C'} = (\ell_1, \ldots, \ell_k)$ where $\ell_k = \overline{\ell}$. Let $\tau = (\ell_1, \ldots, \ell_{k-1})$. We must have $C'[\tau] = U' \cup \{\ell\}$, where U' is a purely universal clause. Because R is responsible for propagating $\overline{\ell}$, we further must have $R[\tau] = U'' \cup \{\overline{\ell}\}$, where U'' again is a purely universal clause. It follows that their resolvent $C[\tau] = (C' \setminus \{\ell\})[\tau] \cup (R \setminus \{\overline{\ell}\})[\tau] = U' \cup U''$ is a purely universal clause. Since τ is a prefix of σ, it follows that $C[\sigma_C]$ is a purely universal clause as well and therefore empty after universal reduction. This proves the claim.

We proceed to prove the lemma. If the algorithm does not learn a clause during conflict analysis, this must be due to a failed attempt at resolving an intermediate clause C with a clause R responsible for unit propagation. That is, if e is the existential pivot variable, there must be a universal variable $u \prec e$ such that $u \in var(C) \cap var(R)$ and $\{u, \neg u\} \subseteq C \cup R$. Towards a contradiction, suppose that $(u, e) \in D$. Let σ denote the assignment that caused the conflict and assume without loss of generality that $\{u, e\} \subseteq R$ and $\{\neg u, \neg e\} \subseteq C$. Since R caused propagation of e but $(u, e) \in D$, the variable u must have been assigned before e and $\neg u \in \sigma$. As the pivot $\neg e$ is the last existential literal falsified in C, it follows that $\neg u \in \sigma_C$. Because $\neg u \in C$, this implies that the assignment σ_C satisfies C, in contradiction with the claim proved above. □

For a formula with n variables the number of dependencies is $O(n^2)$ and the number of distinct constraints is 2^{2n+1}. QCDCL runs into a conflict at least every n variable assignments, so Observation 2 and Lemma 11 imply termination.

Theorem 4. *QCDCL with dependency learning is sound and terminating.*

4.3 Experiments

To see whether dependency learning works in practice, we implemented a QCDCL solver named Qute that supports this technique.[1] We evaluated the performance of Qute in several experiments. First, we measured the number of instances solved by Qute on the union of benchmark sets from the 2016–2018 QBF Evaluations [Pul16]. We compared these numbers with those of the best performing publicly available solvers for each input type. In a second experiment, we computed the dependency sets given by the standard dependency scheme [SS09, BL09] and the reflexive resolution-path dependency scheme [VG11, SS16b] for preprocessed instances, and compared their sizes to the number of dependencies learned by Qute. Finally, we revisit an instance family which is known to be hard to solve for QCDCL [Jan16] and show they pose no challenge to Qute. In fact, we reinforce the last experimental result by a formal proof that QCDCL with dependency learning can indeed solve instances from this family efficiently. For our experiments, we used a cluster with Intel Xeon E5649 processors at 2.53 GHz running 64-bit Linux.

4.3.1 Decision Heuristic, Restart Strategy, and Constraint Deletion Policy

We briefly describe a few design choices for key components of Qute. The exact values of parameters (parameter names are shown in italics) used in the experiments are listed in Table 4.1.

- We rely on a version of the variable move-to-front (VMTF) heuristic for selecting decision variables [Rya04, BF15]. VMTF maintains a list of variables and selects a decision variable that is closest to the head of the list. Upon learning a new constraint, variables occurring in the constraint are moved to the front of the list.

- Restarts are determined by a simple inner-outer restart scheme [Bie08]. A restart is triggered every time the conflict counter reaches a number referred to as the *inner restart distance*. After every restart, the inner restart distance is multiplied by a fixed *restart multiplier* and the conflict counter is reset. After a number of restarts corresponding to the *outer restart distance*, the inner restart distance is reset and the outer restart distance is multiplied by the *restart multiplier*.

- Qute keeps limits on the number of learned clauses and terms, respectively. Upon hitting the limit for clauses or terms, the corresponding constraints are ordered lexicographically according to their literal blocks distance (LBD) [AS09] in increasing order and activity [ES03] in decreasing order. A fraction (determined by the *clause deletion ratio* and *term deletion ratio*) of these constraints is then deleted starting from the back of the list, skipping constraints that are locked because they are the antecedent of a literal on the current trail. The limit on the number of learned

[1] `http://github.com/perebor/qute`

clauses or terms is then increased by a fixed constant (the *clause database increment* and *term database increment*, respectively).

4.3.2 Solved Instances for QBF Evaluation 2016-2018 Benchmark Sets

In our first experiment, we used the prenex non-CNF (QCIR, [JKS16]) benchmark sets from the 2016-2018 QBF Evaluation, consisting in total of 1240 formulas. Time and memory limits were set to 10 minutes and 4 GB, respectively. The results are summarized in Table 4.2 and Figure 4.1. Qute's performance is competitive with other state-of-the-art circuit solvers, and this appears to be in large part due to dependency learning: when dependency learning is deactivated, the number of solved instances drops significantly.

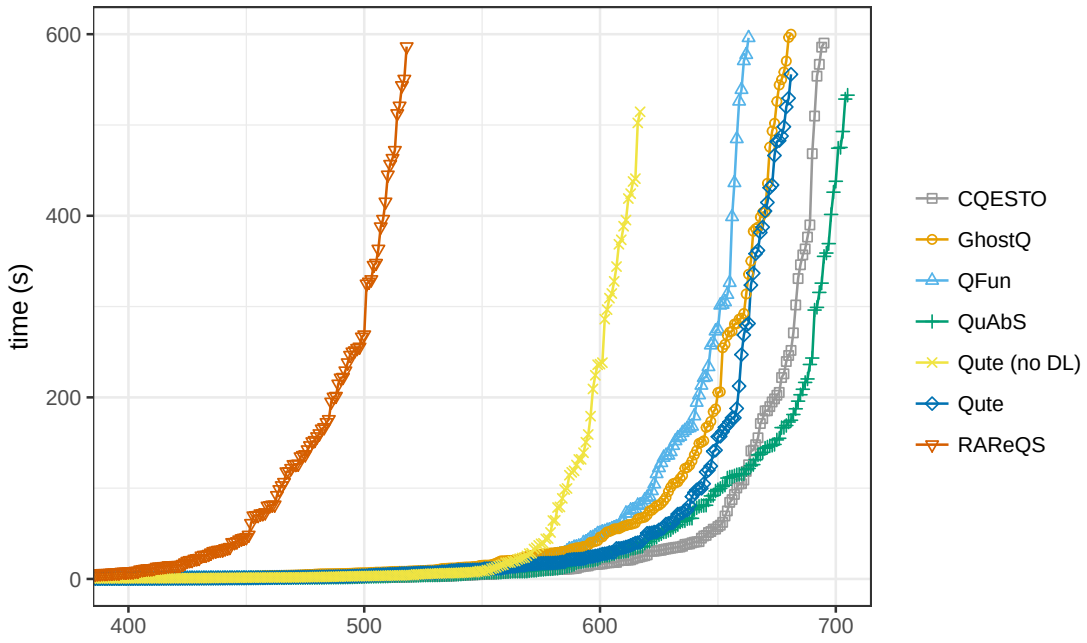

Figure 4.1: Solved instances from the 2016-2018 QBF Evaluation prenex non-CNF (QCIR) benchmark sets (x-axis) sorted by runtime (y-axis).

It is folklore within the QBF community that the number of quantifier alternations has a strong influence on solver performance. Generally speaking, expansion/abstraction solvers tend to do better on instances with few alternations, whereas QCDCL solvers are at an advantage on instances with many alternations. Standard benchmark sets contain many instances with only a single alternation [LE18], presumably because many problems of interest can be encoded in such formulas. Figure 4.2 shows the number of solved prenex non-CNF instances broken down by the number of quantifier alternations. While the

Table 4.1: Command line parameters for Qute used in the experiments.

parameter	value	description
`--decision-heuristic`	VMTF	Decision heuristic, see Section 4.3.
`--restarts`	inner-outer	Restart strategy, see Section 4.3.
`--inner-restart-distance`	250	Initial number of conflicts before restart.
`--outer-restart-distance`	20	Initial number of restarts before outer restart.
`--restart-multiplier`	2.5	Multiplier for inner restart distance (upon restart) and outer restart distance (upon outer restart).
`--initial-clause-DB-size`	1000	Initial limit on learned clauses.
`--initial-term-DB-size`	4000	Initial limit on learned terms.
`--clause-DB-increment`	1000	Upon reaching the current limit on the number of learned clauses, increase the limit by this value.
`--term-DB-increment`	500	Upon reaching the current limit on the number of learned terms, increase the limit by this value.
`--clause-removal-ratio`	0.4	Fraction of learned clauses to delete upon reaching the current limit.
`--term-removal-ratio`	0.3	Fraction of learned terms to delete upon reaching the current limit.
`--LBD-threshold`	5	Only delete constraints with LBD greater than this value.
`--constraint-activity-decay`	0.99	Multiply constraint activities with this value after each conflict.
`--constraint-activity-inc`	-2	Add this value to the activity score of a constraint whenever it is seen during conflict analysis.
`--dependency-learning`	all	Add all variables involved in an illegal merge as learned dependencies.
`--phase-heuristic`	invJW	Heuristic for choosing the assignment of a decision variable based on positive and negative literal occurrences.
`--model-generation`	depQBF	Use DepQBF-style model generation for PCNF instances: pick the first satisfying literal in each clause, with a preference for existential literals.

Table 4.2: Instances from the 2016-2018 QBF Evaluation prenex non-CNF (QCIR) benchmark sets solved within 10 minutes.

solver	total	sat	unsat
QuAbS	705	313	392
CQESTO	695	310	385
Qute	681	315	366
GhostQ	681	296	385
Qfun	663	296	367
Qute (no DL)	617	281	336
RaReQS	518	218	300

number of instances with up to 100 alternations solved by Qute with dependency learning is slightly subpar (even compared to Qute without dependency learning), dependency learning shines when it comes to the subset of instances with the highest number (100+) of quantifier alternations.[2] This is also clearly visible in Figure 4.3, which shows the runtimes of Qute with and without dependency learning for individual instances. On average, dependency learning incurs a slight performance penalty for instances solved by both configurations but leads to many more solved instances among those with at least 100 quantifier alternations.

Many of these formulas have a number of quantifier alternations that is close to the overall number of variables. Apparently, most of the corresponding variable dependencies are spurious, and dependency learning allows Qute to ignore them.

For our second experiment, we used the prenex CNF (PCNF) benchmark sets from the 2016-2018 QBF Evaluations consisting of 1314 instances. Time and memory limits were again set to 10 minutes and 4 GB. We performed this experiment twice: with and without preprocessing using HQSpre [WRMB17]. In order not to introduce variance in overall runtime through preprocessing, each instance was preprocessed only once and solvers were run on the preprocessed instances with a timeout corresponding to the overall timeout minus the time spent on preprocessing.

Since Qute shows good performance on QCIR instances, we included configurations that perform partial circuit reconstruction using *qcir-conv*[3] and then solve the resulting QCIR instance.

The results obtained without using HQSpre are shown on the left hand side of Table 4.3. When using qcir-conv, Qute solves significantly more instances with dependency learning than without dependency learning. Without qcir-conv, the difference is less pronounced,

[2]This matches our experimental results on portfolio-based algorithm selection for QCIR, where selectors favored Qute with dependency learning over other solvers for instances with many quantifier alternations (see Chapter 8).

[3]`http://www.cs.cmu.edu/~wklieber/qcir-conv.py`

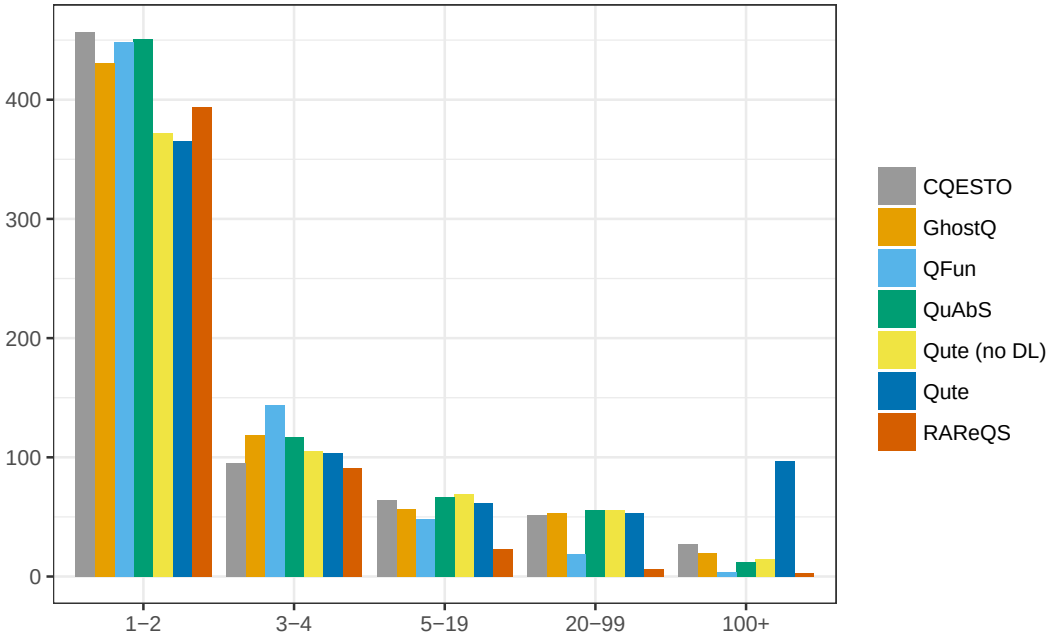

Figure 4.2: Solved instances from the 2016-2018 QBF Evaluation Prenex non-CNF (QCIR) benchmark sets (y-axis) by number of quantifier alternations (x-axis).

but dependency learning remains beneficial. Overall, we see that circuit reconstruction (also used internally by GhostQ [KSGC10] substantially increases the performance of Qute.

Table 4.3: Instances from the QBF Evaluation 2016-2018 prenex CNF (PCNF) benchmark sets solved within 10 minutes without preprocessing (left) and with preprocessing using HQSpre (right). Configurations labeled with +CR use partial circuit reconstruction.

solver	total	sat	unsat
GhostQ	752	350	402
Qute +CR	712	315	397
Qute (no DL)+CR	667	313	354
DepQBF	637	259	378
CaQE	549	216	333
RaReQS	510	152	358
Qute	499	166	333
Qute (no DL)	488	172	316

solver	total	sat	unsat
CaQE	978	442	536
RaReQS	945	410	535
DepQBF	888	396	492
Qute +CR	871	374	497
Qute (no DL)+CR	866	377	489
Qute (no DL)	858	381	477
GhostQ	833	369	464
Qute	827	347	480

The results including preprocessing with HQSpre are shown on the right hand side of

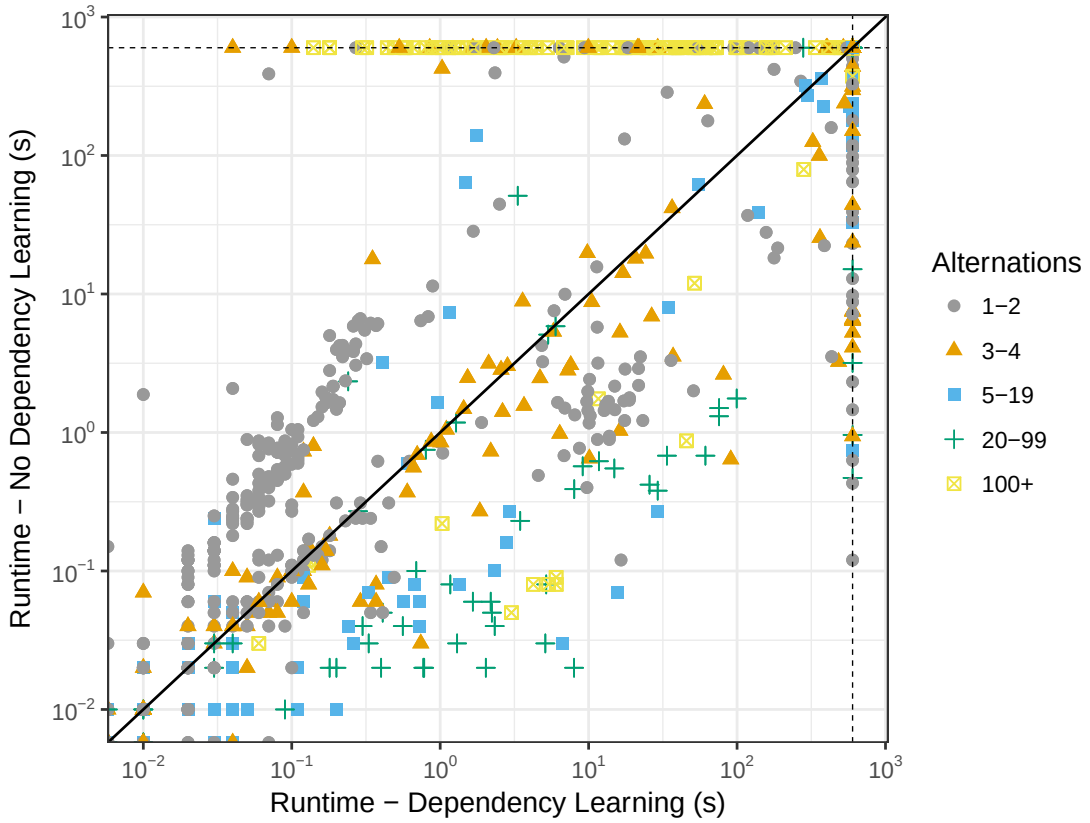

Figure 4.3: Runtimes of Qute with and without dependency learning on the 2016-2018 QBF Evaluation Prenex non-CNF (QCIR) benchmark sets, by number of quantifier alternations.

Table 4.3. The power of the preprocessor strikes the eye: any solver/configuration solves more instances when paired with HQSpre, than any other without it. While dependency learning does not seem to provide as much as in the previous cases, it is still part of the best-performing configuration of Qute.

4.3.3 Learned Dependencies Compared to Dependency Relations

To get an idea of how well QCDCL with dependency learning is able to exploit independence, we compared the number of dependencies learned by Qute with the number of standard and resolution-path dependencies for instances from the PCNF benchmark set after preprocessing with HQSpre. We only considered instances with at least one quantifier alternation after preprocessing. Qute was run with a 10 minute timeout (excluding preprocessing). If an instance was not solved we used the number of dependencies learned

within that time limit.[4]

Summary statistics are shown in Table 4.4. On average, the standard dependency scheme only provides a mild improvement over trivial dependencies. The reflexive resolution-path dependency scheme does better, but the set of trivial dependencies it can identify as spurious is still small in many cases. The fraction of learned dependencies is much smaller than either dependency relation on average, and the median fraction of trivial dependencies learned is even below 1%.

This indicates that proof search in QCDCL with dependency learning is less constrained than in QCDCL with either dependency scheme: since QCDCL is allowed to branch on a variable x only if every variable that x depends on has already been assigned, decision heuristics are likely to have a larger pool of variables to choose from if fewer dependencies are present.

Table 4.4: Learned dependencies, standard dependencies, and reflexive resolution-path dependencies for instances preprocessed by HQSpre, as a fraction of trivial dependencies.

dependencies	mean	median	variance
standard	0.929	1.000	0.029
resolution-path	0.628	0.798	0.139
learned	0.033	0.007	0.004

4.3.4 Dependency Learning on Hard Instances for QCDCL

For our third experiment, we chose a family of instances CR_n recently used to show that ordinary QCDCL does not simulate tree-like Q-resolution [Jan16]. Since the hardness of these formulas is tied to QCDCL not propagating across quantifier levels, they represent natural test cases for QCDCL with dependency learning. We recorded the number of backtracks required to solve CR_n by Qute with and without dependency learning, for $n \in \{1, ..., 50\}$. As a reference, we used DepQBF.[5] For this experiment, we kept the memory limit of 4 GB but increased the timeout to one hour. The results are summarized in Figure 4.4. As one would expect, Qute without dependency learning and DepQBF were only able to solve instances up to $n = 7$ and $n = 8$, respectively. Furthermore, it is evident from the plot that the number of backtracks grows exponentially with n for both solvers. By contrast, Qute with dependency learning was able to solve all instances within

[4] We cannot rule out that, for unsolved instances, Qute would have to learn a larger fraction of trivial dependencies before terminating. However, the solver tends to learn most dependencies at the beginning of a run, with the fraction of learned trivial dependencies quickly converging to a value that does not increase much until termination.

[5] For the sake of comparing with Qute in prefix mode, we disabled features recently added to DepQBF such as dynamic quantified blocked clause elimination [LBB+15] and oracle calls to the expansion-based solver Nenofex.

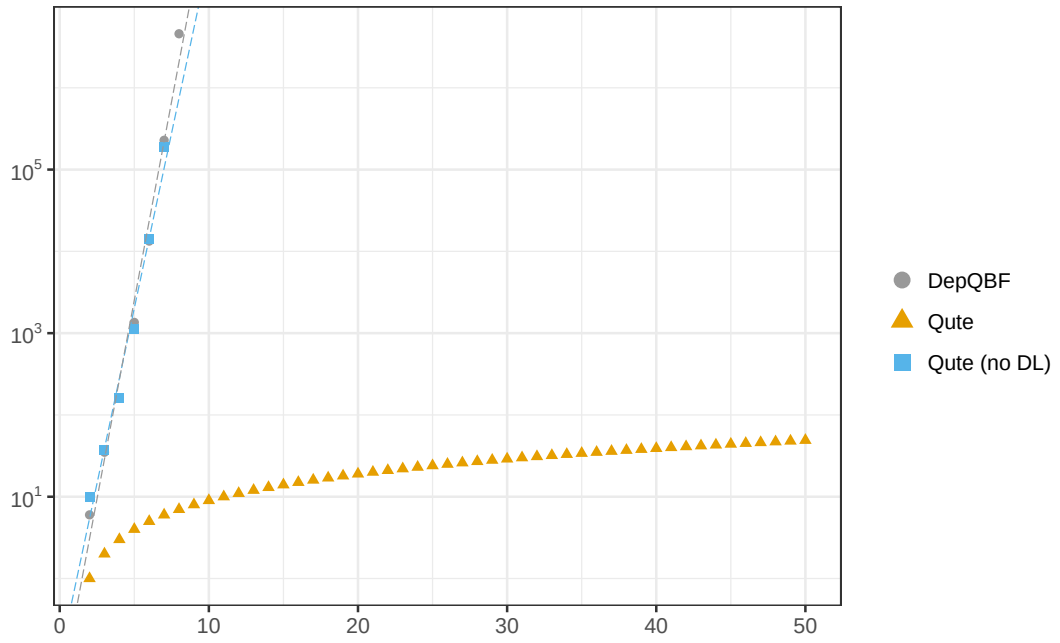

Figure 4.4: Backtracks for instances CR_n based on the completion principle [Jan16], as a function of n.

the timeout. In the next section, we will formally prove that QCDCL with dependency learning can find short proofs of the formulas CR_n for all n.

4.4 An Exponential Speedup over QCDCL

We will now show that there is a run of QCDCL with dependency learning on CR_n that terminates in time polynomial in n. The corresponding short proof of CR_n that is found is, up to a symmetry, the one presented by Janota [Jan16]. Let us first recall the definition of CR_n.

Definition 12. *Let $n \in \mathbb{N}$, and let $\mathcal{X} = \{ x_{ij} : 1 \leq i, j \leq n \}$, and $\mathcal{L} = \{ a_i, b_i : 1 \leq i \leq n \}$ be sets of variables. The formula CR_n has the prefix $\exists\mathcal{X}\forall z\exists\mathcal{L}$, and the matrix consisting of the following clauses:*

$$A_{ij} = x_{ij} \vee z \vee a_i \qquad \text{for } 1 \leq i, j \leq n$$
$$B_{ij} = \neg x_{ij} \vee \neg z \vee b_j \qquad \text{for } 1 \leq i, j \leq n$$
$$A = \bigvee_{i=1}^{n} \neg a_i \qquad B = \bigvee_{j=1}^{n} \neg b_j$$

Lemma 12. *Assume that the current trail of a QCDCL solver with dependency learning contains only literals from the set $\{a_1, \ldots, a_n\}$. Then, assigning any a_i that is not on*

the trail to false causes unit propagation to derive a conflict, and the clause $(z \vee a_i)$ *is derived during conflict analysis.*

Proof. Assigning a_i to false causes the clause A_{ij} to propagate x_{ij} for $j = 1, \ldots, n$. In turn, assigning x_{ij} to true causes the clause B_{ij} to propagate b_j for $j = 1, \ldots, n$ (because there is no dependency of b_j on z). This causes a conflict with the clause B.

During conflict analysis, we will resolve B with the clauses B_{ij} on $b_1, \ldots, b_n$, effectively removing all $\neg b_j$ from B, and introducing $\neg x_{i1}, \ldots, \neg x_{in}$ and $\neg z$. At that point, $\neg z$ is trailing and can be reduced. Afterwards, we will resolve away all the $\neg x_{ij}$ using A_{ij}, and we will end up with the clause $(z \vee a_i)$ as required. Note that each intermediate clause contains at least two literals from the set $\{a_i, \neg b_1, \ldots, \neg b_n, \neg x_{i1}, \ldots, \neg x_{in}\}$ which are existential and assigned at the same decision level as a_i (which is the highest decision level) so none of these clauses can be asserting. □

Consider the following run of QCDCL with dependency learning. Each clause contains at least two existential literals, so (even with an empty dependency relation) no variable is assigned by unit propagation at decision level 0. As the first decision, the variable a_1 is assigned to false. By Lemma 12 the clause $(z \vee a_1)$ is derived during conflict analysis. This clause is asserting, so it is learned, the algorithm backtracks to decision level 0, and unit propagation sets a_1 to true. This is repeated for $a_1, \ldots, a_{n-1}$. During propagation of the assignment $a_1 \wedge \cdots \wedge a_{n-1}$ the clause A propagates $\neg a_n$, which by Lemma 12 again leads to a conflict and the derivation of the clause $(z \vee a_n)$ during conflict analysis. This time, because $\neg a_n$ is assigned at decision level 0, the clause $(z \vee a_n)$ is not asserting. Thus conflict analysis proceeds to resolve $(z \vee a_n)$ with A, and then subsequently with the clauses $(z \vee a_i)$ for $i = 1, \ldots, n-1$. This yields the unit clause (z), which is reduced to the empty clause, resulting in QCDCL terminating and outputting FALSE.

This run requires n invocations of propagation and conflict analysis. Since both unit propagation and conflict analysis can be carried out in polynomial time, we get a polynomial bound on the overall runtime.

Theorem 5. *QCDCL with dependency learning can solve* CR_n *in polynomial time.*

Since ordinary QCDCL requires time exponential in n to solve CR_n [Jan16], dependency learning achieves an exponential speedup on these instances.

4.5 An Interpretation of Learned Dependencies

Dependencies in QBF naturally arise in a semantic context. The formula $\forall x \exists y . \varphi(x, y)$ says that we can choose a value for y, *depending on* x, such that $\varphi(x, y)$ evaluates to true. In other words, the assertion that this formula is true is equivalent to saying that there is a function f_y, which depends on x as its input, and which chooses values for y so that φ evaluates to true. Such a function is called a *model* of the formula, and one way

of thinking about dependencies in a given formula is to think about the dependencies exploited by a model/all models of the formula. For instance, in this example, it could be the case that a constant function f_y that does not exploit the information about x at all suffices to make $\varphi(x, y)$ evaluate to true—in such a case we would say that the dependency of y on x, while present *syntactically*, is spurious *semantically*.

QCDCL with dependency learning does not directly extract semantic information about the formula it is solving in the sense outlined in the previous paragraph—it simply relaxes the rules of QCDCL, and refines the relaxation (by learning a dependency) to avoid an illegal merge during constraint learning. Nevertheless, there is something that can be said about the relationship between learned and "actual", semantic dependencies of a formula. In this section, we formalize what we mean by semantic dependencies, and study how they relate to learned dependencies.

Our notion of semantic dependencies is based on changes (or lack thereof) in the truth value of a formula when certain variables are shifted around in the prefix. As an example, consider the formula

$$\Phi = \forall u \forall x \exists z \forall v \exists y \exists e.\varphi$$

for some matrix φ. We define the y-x shift of Φ as the formula

$$\Phi' = \forall u \exists y \forall x \exists z \forall v \exists e.\varphi,$$

i.e., the variable y is moved "just in front of" x, maintaining the relative order of other variables. This is formalized in the definition below.

Definition 13. *Let $\Phi = Q_1 x_1 \dots Q_n x_n.\varphi$ be a QBF, and let $x = x_k$ and $y = x_m$, $k < m$. Let $g_k^m(j)$ for $j = 1, \dots, n$ be defined in the following way:*

$$g_k^m(j) = \begin{cases} j & \text{if } j < k \text{ or } m < j \\ m & \text{if } j = k \\ j - 1 & \text{otherwise} \end{cases}$$

The y-x shift of Φ *is the formula $\Phi_x^y = Q_{g_k^m(1)} x_{g_k^m(1)} \dots Q_{g_k^m(n)} x_{g_k^m(n)}.\varphi$.*

Notice that the mapping g_k^m is a permutation of the set $\{1, \dots, n\}$, and the prefix $Q_{g_k^m(1)} x_{g_k^m(1)} \dots Q_{g_k^m(n)} x_{g_k^m(n)}$ is what results from $Q_1 x_1 \dots Q_n x_n$ when y is shifted just in front of x. This notion can naturally be extended to shifting in front of arbitrary sets of variables.

Definition 14. *Let $\Phi = Q_1 x_1 \dots Q_n x_n.\varphi$ be a QBF, let $y = x_m$, and $\emptyset \neq X \subseteq var(\Phi)$. Let $k = \min_{x_j \in X} j < m$. The y-X* shift of Φ *is the formula $\Phi_X^y = \Phi_{x_k}^y$.*

For a given variable y, any set of variables on which a model for y is allowed to depend (syntactically), i.e., a set of variables left of y and of opposite quantifier type, will be called a *syntactic dependency set*.

Definition 15. *Let $\Phi = \mathcal{Q}.\varphi$ be a QBF, $X \subseteq var(\Phi)$, and $y \in var(\Phi) \setminus X$. If $x \prec_\Phi y$ for each $x \in X$, then we say that X is a* syntactic dependency set *of y. If $X = \{ x \}$, we say that x is a* syntactic dependency *of y.*

Finally, we define *potential* and *critical* semantic dependencies. Intuitively, X is a *critical dependency set* of y if moving y in front of X in the prefix makes the formula change its truth value. X is a *potential dependency set* of y if the player who owns y loses when y is shifted in front of X, but does not necessarily win when y is in its original place. This is akin to a necessary condition—X must be left of y if there should be a chance to win, but maybe not even that is sufficient. Every critical dependency set is also a potential dependency set.

Definition 16. *Let $\Phi = \mathcal{Q}.\varphi$ be a QBF, y one of its existential (universal) variables, and X a syntactic dependency set of y. If Φ_X^y is false (true), then we say that X is a* potential dependency set *of y. If, moreover, Φ is true (false), then we say that X is a* critical dependency set *of y. If $X = \{ x \}$, we say that x is a* potential *or* critical dependency *of y, respectively.*

We note that in the above definition, a critical (potential) dependency is a critical (potential) dependency set of cardinality 1, and that this is different from being an element of a larger critical (potential) dependency set. It can be easily seen that any superset of a critical (potential) dependency set is also a critical (potential) dependency set, but the same does not necessarily hold for subsets.

Example 5. Consider the formula

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

Clearly Φ is true, as is witnessed by the model $f_y(x) = x$, and x is a syntactic dependency of y. If we shift y in front of x, we get the formula $\Phi_x^y = \exists y \forall x\ (x \vee \neg y) \wedge (\neg x \vee y)$, which is false, witnessed by the countermodel $f_x(y) = \neg y$. Hence, we conclude that x is a critical dependency of y.

Example 6. Consider the formula

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

If we shift y in front of x, we get the formula $\Phi_x^y = \exists y \forall x\ (x \vee \neg y) \wedge (x \vee y)$, which is false, witnessed by the countermodel $f_x(y) = 0$. Hence, x is a potential dependency of y. However, in this case Φ itself is false, and so x is not a critical dependency of y.

The results of this section show that learning certain seemingly necessary dependencies can be avoided, and provide a characterization of learned dependencies and the context in which they are learned. Theorem 6 says that QCDCL with dependency learning can solve formulas with many critical dependencies (critical dependency sets of size 1) without learning any of them. Theorem 7 on the other hand says that any learned dependency must be contained in a potential dependency set in a restriction of the input formula.

Theorem 6. *For every $n \in \mathbb{N}$ there is a QBF Φ_n with $O(n^2)$ variables and $\Omega(n^2)$ critical dependencies, and a run of QCDCL with dependency learning on Φ_n that terminates in polynomial time and learns no dependency.*

Proof. Let $\Phi_n = \mathrm{CR}_n$ (see Definition 12). The required run of QCDCL with dependency learning has already been presented in the previous section. It remains to show that there are indeed the required critical dependencies.

Let $1 \leq i_0, j_0 \leq n$, and consider the QBF $(\Phi_n)^z_{x_{ij}}$, which results from swapping x_{ij} and z in the prefix of Φ_n. It can be verified that the following set of (mostly constant) functions is a model of $(\Phi_n)^z_{x_{ij}}$:

$$\begin{aligned} x_{i_0 j} &= 1 && \text{for } j \neq j_0 \\ x_{i j_0} &= 0 && \text{for } i \neq i_0 \\ x_{i_0 j_0} &= \neg z \\ a_i = b_j &= 1 && \text{for } i \neq i_0 \wedge j \neq j_0 \\ a_{i_0} = b_{j_0} &= 0 \end{aligned} \quad ,$$

and hence $(\Phi_n)^z_{x_{ij}}$ is true. Since Φ_n is false, we get that $x_{i_0 j_0}$ is a critical dependency of z, and there are n^2 choices of i_0, j_0. □

Theorem 7. *Assume QCDCL with dependency learning is solving the formula Φ and learns a dependency of a variable y on a variable x. Let τ be the current trail assignment, and let σ be τ restricted to literals of the same quantifier type as y. Then there is $X \subseteq var(\Phi)$, such that $x \in X$ and X is a potential dependency set of y in $\Phi[\sigma]$.*

Proof. Without loss of generality, let us assume that x is universal and y is existential. Consider QCDCL with dependency learning in the state when it learns a new dependency of y on x. The learned dependency stems from a failed attempt at long-distance resolution of two clauses R, the clause that became unit during search, and C, the intermediate learned clause, derived from Φ, over the pivot y, without loss of generality $y \in R$. Let τ be the trail assignment at the point when the resolution of R and C is attempted, i.e., up to but excluding y, and let σ be τ restricted to existential literals. Since R was unit under τ and propagated y, and C is the intermediate learned clause, which must contain only universal literals after the application of $\tau \cup \{y\}$ (cf. proof of Lemma 11), we have that y is the only existential literal in $R[\sigma]$ and $\overline{y}$ is the only existential literal in $C[\sigma]$. The reason why long-distance resolution of R and C fails is that there is at least one universal variable $x' \prec y$ blocking the resolution, i.e., without loss of generality, $x' \in R$ and $\overline{x'} \in C$. Let X be the set of all universal variables blocking the resolution of R and C, clearly a syntactic dependency set of y. We will show that σ and X satisfy the stated conditions.

First, by definition of dependency learning, the learned dependency x is in the set X. We need to prove that X is a potential dependency set of y in $\Phi[\sigma]$, i.e., we need to show that

$\Phi[\sigma]^y_X$ is false. We will do that by restricting the long-distance Q-resolution derivations of the clauses R and C, which are implicitly generated by QCDCL, by the assignment σ. Since neither R, nor C are satisfied by σ, and σ only assigns existential variables, the restricted derivations derive clauses $R' \subseteq R[\sigma]$ and $C' \subseteq C[\sigma]$ (Lemma 2). If no literal on y is present in either of R' and C', we already have a long-distance Q-resolution derivation of a purely universal clause, and so $\Phi[\sigma]$, and therefore also $\Phi[\sigma]^y_X$, is false (shifting an existential variable to the left can only make a true formula false, not the other way around). Otherwise, consider the shifted prefix in $\Phi[\sigma]^y_X$. With this prefix, we can reduce all literals that are possibly blocking the resolution of R' and C', and hence we can derive the empty clause. Since shifting the existential variable y left in the prefix does not invalidate any previous reductions or resolution steps, we have a valid long-distance Q-resolution refutation of $\Phi[\sigma]^y_X$, and hence X is really a potential dependency set of y in $\Phi[\sigma]$. □

Example 7. Theorem 7 can be illustrated with the following example. Consider the formula

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

which essentially says *if not z, then $y = x$*. One of the models of Φ is $f_z = 1$, which remains a model even if we shift y in front of x. Hence, x is not even a potential dependency of y. However, assume QCDCL with dependency learning is solving this formula and starts with the decision $z \overset{d}{=} 0$. Subsequently, because no dependencies have been learned yet, unit propagation w.l.o.g. sets $y = 1$, and the clause $(z \vee x \vee \overline{y})$ becomes falsified by universal reduction. Conflict analysis will attempt to resolve the (only) two clauses of Φ, which is however prevented by x, and a dependency of y on x will have to be learned.

Theorem 7 says there is a σ and a potential dependency set X of y in $\Phi[\sigma]$. In particular, $\sigma = \neg z$, the decision made by the solver, and $X = \{ x \}$. Therefore, we could also paraphrase Theorem 7 in the following way: if a dependency of y on x is learned, then even if perhaps x is not contained in any potential dependency set of y in the original formula Φ, it is definitely contained in a potential dependency set of Φ restricted by the "current trail assignment", which can be thought of as the actual formula that the solver is solving at the moment of learning the dependency. In this example, at the moment of learning the dependency of y on x, z is assigned to false, and hence the restricted formula says $y = x$, and so learning the dependency is justified.

Note that this example also shows that it is unlikely that we could provide a stronger condition for learned dependencies than Theorem 7, because even when solving a fairly trivial formula with no potential dependency sets dependencies can still be learned.

4.6 Summary and Discussion

In this section, we introduced dependency learning for QCDCL, a novel method for dependency analysis. Dependency learning has several advantages over the use of

dependency schemes within QCDCL.

First, it is arguably easier to implement. The integration of the standard dependency scheme with DepQBF required the development of data structures for the succinct representation of standard dependencies [BL09], and no such compact representation is currently known for the resolution-path dependency scheme. By contrast, dependency sets in Qute are encoded in arrays containing a list of variables.

Second, our experiments indicate that proof search is less constrained since Qute typically learns only a fraction of the dependencies computed by dependency schemes.

Third, by keeping long-distance Q-resolution as the underlying proof system, QCDCL with dependency learning is amenable to a simple correctness proof and enjoys linear-time strategy extraction.

Blinkhorn and Beyersdorff [BB17] offered a strong proof-theoretic argument in favor QCDCL with dependency schemes over "vanilla" QCDCL by proving an exponential separation of Q(D^{rrs})-resolution and ordinary Q-resolution, where Q(D^{rrs})-resolution is the proof system used by QCDCL with the reflexive resolution-path dependency scheme D^{rrs} and a form of constraint learning that avoids long-distance resolution [LEVG13]. However, this separation was proved against a class of formulas introduced by Kleine Büning, Karpinski, and Flögel [KKF95] which is known to have short proofs in long-distance Q-resolution [ELW13], so it does not speak about the relative strength of QCDCL with the resolution-path dependency scheme and QCDCL with dependency learning.

In our experiments, Qute performed much better when presented with non-CNF input. In particular, dependency learning was most effective on the prenex non-CNF (QCIR) benchmark set, accounting for a 10% increase in the number of solved instances. Even for PCNF formulas, the best configuration(s) used tools for partially recovering circuit structure from CNF. This is consistent with the fact that dependency learning had a more limited effect on Qute when preprocessing was used, since preprocessing is known to adversely affect circuit reconstruction [GB13], and so it would impair Qute's best-performing configurations. Whether this bias towards non-CNF representations is inherent to QCDCL with dependency learning or an artifact of other design choices implemented in our solver remains to be seen.

Sometimes, the additional freedom afforded by dependency learning appears to be detrimental to performance. In particular, this seems to be the case when quantifier alternations reflect strict temporal or logical dependencies, such as in formulas encoding two-player games. Freed from the restrictions normally imposed by the quantifier prefix, decision heuristics originally devised for SAT steer Qute into regions of the search space that make little sense semantically and force it to learn many useless constraints and dependencies. We see two ways of addressing this issue.

First, we want to design new decision heuristics that take the quantifier prefix into account, possibly by penalizing out-of-order assignments.

Second, we plan to develop techniques for initializing learned dependencies with a small set of dependencies that might help steer proof search in the right direction. For instance, Qute uses Tseitin conversion to obtain a set of initial clauses and terms from non-CNF (QCIR) instances. We found that assigning a Tseitin variable before a variable used in its definition often results in learning a dependency, so that it pays off to simply include dependencies of a Tseitin variable on the variables used in its definition from the start. For similar reasons, it might make sense to include dependencies induced by implicit variable definitions [LM08]. Efficient techniques for detecting implicit variable definitions have been developed for preprocessing in propositional model counting [LLM16, IMMV16].

It is worth noting that dependency learning supports the removal of learned dependencies just as well as their addition. Although we did not encounter instances where the set of learned dependencies grows so large that it significantly affects performance, it is possible that the management of learned dependencies causes an overhead during longer solver runs. For that reason, removing learned dependencies at regular intervals in analogy to constraint deletion might be beneficial.

Publication Notes

The research in this chapter (except Sections 4.4 and 4.5) appeared in a paper published in the proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT 2017) [PSS17], and in full in a paper published in Volume 65 (2019) of the Journal of Artificial Intelligence Research [PSS19a].

CHAPTER 5

Combining Resolution-Path Dependencies with Dependency Learning

In Chapter 3, we showed that LDQ(D^{rrs})-resolution, the proof system that models QCDCL solving with long-distance Q-resolution and the reflexive resolution-path dependency scheme, is sound, and hence can be used for solving if the dependency scheme can be computed efficiently enough. In contrast, in Chapter 4, we introduced dependency learning, a new technique for dependency analysis that is orthogonal to dependency schemes. We argued that dependency learning offers an array of advantages over dependency schemes, such as soundness for free, empirically better independence detection, and native support for fast strategy extraction. In fact, the very idea of dependency learning was conceived in order to address the shortcomings of dependency schemes. However, that does not mean that dependency learning cannot be used in conjunction with dependency schemes.

In this chapter, we show exactly that—how to incorporate both dependency learning and dependency schemes into a QCDCL solver. Moreover, we present the first practically efficient implementation of the reflexive resolution-path dependency scheme. In order to achieve that, we give up on computing the entire dependency relation upfront like DepQBF does, and instead query certain dependencies on the fly, during dependency conflicts. Recall that a dependency conflict occurs when the solver attempts to resolve two clauses that do not have a resolvent in long-distance Q-resolution. If the blocking variables are in fact independent according to D^{rrs}, the resolution step is valid in LDQ(D^{rrs})-resolution. Hence, dependency conflicts are a great place at which to compute dependencies. Once computed, however, the dependencies are kept for future use in both dependency conflicts, as well as universal reduction.

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

An *input clause* $C \in \varphi$ can be used as an axiom. From two clauses $C_1 \vee e$ and $\neg e \vee C_1$, where e is an existential variable, the *(long-distance) resolution* rule can derive the clause $C_1 \vee C_2$, provided that $(u, e) \notin \mathcal{D}^{\text{rrs}}_{\Phi}$ for each universal variable u with $u \in C_1$ and $\overline{u} \in C_2$ (or vice versa), and that $C_1 \vee C_2$ does not contain an existential variable in both polarities.

$$\frac{C}{C \setminus \{u, \neg u\}}\ \text{(generalized } \forall\text{-reduction)}$$

The $\forall$*-reduction* rule derives the clause $C \setminus \{u, \neg u\}$ from C, where $u \in var(C)$ is a universal variable such that $(u, e) \notin \mathcal{D}^{\text{rrs}}_{\Phi}$ for every existential variable $e \in var(C)$.

Figure 5.1: Derivation rules of LDQ($\mathcal{D}^{\text{rrs}}$)-resolution for a PCNF formula $\Phi = \mathcal{Q}.\varphi$.

5.1 Reflexive Resolution-Path Dependency Scheme

Recall Definition 8 of the reflexive resolution path dependency scheme from Section 2.5. When D^{rrs} is used in QCDCL solving, the solver learns clauses in a generalization of long-distance Q-resolution called LDQ(D^{rrs})-resolution. Figure 5.1 shows the proof rules of LDQ(D^{rrs})-resolution (cf. Figure 3.1). Soundness of LDQ(D^{rrs})-resolution has been established by Corollary 3. We note that the soundness of the corresponding LDQ(D^{rrs})-consensus for terms still remains as an open problem. In our experiments with the proof system, we have been able to independently verify the truth value of all formulas by a different QBF solver.

5.2 Using Resolution-Path Dependencies in Practice

The major issue with implementing any dependency scheme for use in a QBF solver is the fact that the size of the dependency relation is inherently worst-case quadratic in the number of variables—all pairs of variables of opposite quantifier type potentially need to be stored. QBFs of interest often contain hundreds of thousands of variables, and therefore any procedure with quadratic complexity is infeasible. DepQBF overcomes this by identifying equivalence classes of variables with identical dependency information, and storing only one chunk of data per equivalence class [BL09]. This compressed form, however, is specifically tailored to the standard dependency scheme, and cannot directly be transferred to other dependency schemes.

5.2.1 Dynamically Applying $\mathcal{D}^{\text{rrs}}$

In order to avoid the quadratic blowup, we take a different approach. We do not aim at computing the entire dependency relation, but instead compute parts of it on demand,

when a *dependency conflict* occurs.

Dependency conflicts in clause learning in QCDCL with dependency learning take place in the following way (in this entire section we focus on the case of clauses, but the case of term learning is dual): the solver attempts to resolve two clauses, C_1 and C_2, over a pivot variable e, but there is a non-empty set of universal variables U, such that

$$\forall u \in U \; u \prec e, (u \in C_1 \wedge \overline{u} \in C_2) \vee (\overline{u} \in C_1 \wedge u \in C_2).$$

These variables are blocking the resolution step, as is shown in the pseudocode snippet in Algorithm 4. The reason why this occurs is that the solver mistakenly assumed e not to depend on any $u \in U$, and this erroneous assumption is now to be rectified by learning the dependency of e on at least one variable from U.

Algorithm 5 Conflict Analysis with DL and a Dependency Scheme

```
1:  procedure analyzeConflict(conflict)
2:      constraint = getConflictConstraint(conflict)
3:      while not asserting(constraint) do
4:          pivot = getPivot(constraint)
5:          reason = getAntecedent(pivot)
6:          if existsResolvent(constraint, reason, pivot) then
7:              constraint = resolve(constraint, reason, pivot)
8:              constraint = reduce(constraint)
9:          else // dependency conflict
10:             U = illegalMerges(constraint, reason, pivot)
11:             rrs_deps[pivot] = getDependencies(pivot)
12:             U = U ∩ rrs_deps[pivot]
13:             if U = ∅ then
14:                 goto 7
15:             else
16:                 D = D ∪ { (v, pivot) : v ∈ U }
17:                 return none, decisionLevel(pivot)
18:             end if
19:         end if
20:     end while
21:     btlevel = getBacktrackLevel(constraint)
22:     return constraint, btlevel
23: end procedure
```

We can conveniently insert a dynamically computed dependency scheme at this moment. Before any dependency of e is learned, the dependencies of e according to the dependency scheme are computed. Any $u \in U$ that turns out to be independent of e can be removed from the set of blocking variables. If everything in U is independent, no dependency needs to be learned, and conflict analysis can proceed by performing a resolution step in

LDQ($\mathcal{D}^{\text{rrs}}$)-resolution, in which all $u \in U$ are merged over e. If some variables in U turn out to be actual dependencies of e, at least one of them has to be learned as usual. The modification to the conflict analysis process is shown in Algorithm 5.

The computed dependencies of e are then stored and re-used in any future dependency conflicts featuring e as the pivot variable, as well as in strengthening the reduction rule.

Soundness of QCDCL with dependency learning *and* the reflexive resolution-path dependency scheme follows from the soundness of long-distance Q($\mathcal{D}^{\text{rrs}}$)-resolution, the underlying proof system used by the algorithm.

5.2.2 Dynamically Computing $\mathcal{D}^{\text{rrs}}$

When computing resolution-path connections, it is natural to start with a variable v, and compute all variables which depend on v. This is because in this case, the set of connecting variables that can form proper resolution paths is fixed—all existential variables right of v are permitted—and the task of finding everything that depends on v is reducible to reachability in a single directed graph. However, since a dependency conflict may feature any number of blocking variables, we would potentially need to perform the search many times in order to check each dependency. It would be preferable to compute all dependencies of the pivot variable instead. However, since for every blocking variable $u \in U$, the set of allowed connecting variables may be different, we cannot reduce the task of finding all dependencies of the pivot e to just reachability in a single directed graph, and we need a different approach.[1]

Definition 17. *Let Φ be a PCNF formula, ℓ a literal of Φ, and $w_\ell : var(\Phi) \cup \overline{var(\Phi)} \to \mathbb{R} \cup \{\pm\infty\}$ the mapping defined by*

$$w_\ell(l) = \begin{cases} \infty & \text{if } l = \bar{\ell}, \\ \delta(\ell) & \text{if } l \neq \bar{\ell} \text{ and } var(l) \text{ is existential}, \\ -\infty & \text{otherwise.} \end{cases}$$

The depth-implication graph for Φ at ℓ, *denoted $DIG(\Phi, \ell)$ is the weighted version of $IG(\Phi)$ where the weight of an edge (ℓ_1, ℓ_2) is defined as $w(\ell_1, \ell_2) = w_\ell(\ell_1)$.*

For a path π in a weighted directed graph G, the *width* of π is defined as the minimum weight over all edges of π. The following theorem relates resolution paths in a formula with *widest paths* in its depth-implication graph.

Theorem 8. *Let ℓ, ℓ' be two literals of a PCNF formula Φ such that $\delta(\ell') < \delta(\ell)$. There is a proper resolution path from ℓ to ℓ' if, and only if, the widest path from $\bar{\ell}$ to ℓ' in $DIG(\Phi, \ell)$ has width larger than $\delta(\ell')$.*

[1] This is the case regardless of the quantifier type of the pivot, the issue is that different targets in the set of blocking variables can be reached using different connecting variables.

Proof. Let $\pi = \ell, \ell_2, \ldots, \ell_{2k-1}, \ell'$ be a proper resolution path, and let

$$\pi' = \overline{\ell}, \ell_2, \ldots, \ell_{2k-2}, \ell'$$

be the corresponding path in $\mathrm{DIG}(\Phi, \ell)$ (by Lemma 1). The width of π' is defined as

$$\begin{aligned} w(\pi') &= \min\left\{w(\overline{\ell}, \ell_2), \ldots, w(\ell_{2k-2}, \ell')\right\} \\ &= \min\left\{w_\ell(\overline{\ell}), \ldots, w_\ell(\ell_{2k-2})\right\}. \end{aligned}$$

Since $w_\ell(\overline{\ell}) = \infty$ and π is proper and hence none of its connecting variables are universal, we have that $w(\pi') = \min\{\delta(\ell_2), \ldots, \delta(\ell_{2k-2})\} > \delta(\ell')$, where the inequality follows from π being proper.

Conversely, let $\pi' = \overline{\ell_1}, \ell_2, \ldots, \ell_k, \ell'$ be a path of width greater than $\delta(\ell')$, and let $\pi = \ell_1, \ell_2, \overline{\ell_2}, \ldots, \ell_k, \overline{\ell_k}, \ell'$ be the corresponding resolution path. Since $w(\pi') > \delta(\ell')$, no connecting variables in π can be universal, and they all have to be right of ℓ', hence π is proper. □

Naively applying the algorithm from [SS16a] would result in an overall quadratic running time needed to determine all dependencies of a given variable v. Using Theorem 8 we can reduce the task to two searches for widest paths, and obtain a much more favourable time bound.

Theorem 9. *Given a variable v of a PCNF formula Φ, all resolution-path dependencies, i.e., the set $\{\, x \in var(\Phi) : (x, v) \in \mathcal{D}^{rrs}_\Phi \,\}$, can be computed in time $O(\|\Phi\| \log \|\Phi\|)$.*

Proof. In order to find out whether a given candidate variable x is a dependency of v, one has to determine whether there is a pair of proper resolution paths, either from v to x and from $\overline{v}$ to $\overline{x}$, or from v to $\overline{x}$ and from $\overline{v}$ to x. Theorem 8 tells us that the existence of proper resolution paths is equivalent to existence of wide paths. A generalization of Dijkstra's algorithm can compute widest paths from a single source to all destinations in a given graph in quasilinear time [KP06]. The key observation is that the entire computation is performed within two graphs, namely $\mathrm{DIG}(\Phi, v)$ and $\mathrm{DIG}(\Phi, \overline{v})$. By computing all widest paths from both v and $\overline{v}$, and then subsequently checking for which candidate variables x both polarities of x are reached by a wide enough path, we can find all dependencies of v.

By using the clause-splitting trick like in [SS16a] we can, in linear time, obtain an equisatisfiable formula Φ' with $var(\Phi) \subseteq var(\Phi')$ such that the resolution-path connections between variables of Φ are the same. Since Φ' has bounded clause size, we get that the number of edges in $\mathrm{IG}(\Phi')$ is $O(\|\Phi'\|) = O(\|\Phi\|)$, and the stated running time is then simply the running time of Dijkstra's algorithm. □

5.3 Experiments

We modified the dependency-learning solver Qute so as to perform the procedure described above—when a dependency is about to be learned, resolution-path dependencies of the pivot variable are computed, and all blocking variables that turned out to be spurious dependencies are eliminated. Furthermore, the computed dependencies are kept for re-use in future dependency conflicts featuring the same pivot variable, as well as to be used in generalized ∀-reduction.

We evaluated our solver on a cluster of 16 machines each having two 10-core Intel Xeon E5-2640 v4, 2.40GHz processors and 160GB of RAM, running Ubuntu 16.04. We set the time limit to 900 seconds and the memory limit to 4GB. As our benchmark set, we selected the QDIMACS instances available in the QBF Library[2] [GNPT05]. We first preprocessed them using the preprocessor HQSpre[3] [WRMB17] with a time limit of 400 seconds, resulting in a set of 14893 instances not solved by HQSpre. Out of these instances, we further identified the set of easy instances as those solved within 10 seconds by each of the following solvers: CaQE[4] 3.0.0 [RT15], DepQBF[5] 6.03 [BL10], QESTO[6] 1.0 [JM15], Qute[7] 1.1 [PSS19a], and RaReQS[8] 1.1 [JKMSC12]. We decided to focus only on instances not solved by at least one of these solvers in under 10 seconds, as it arguably makes little sense to try and push state of the art for formulas that can already be solved in almost no time regardless of the choice of the solver. That left us with a set of 11262 instances.

Table 5.1 and Figure 5.2 show the comparison between plain Qute and the version which implements the dependency scheme (Qute-$\mathcal{D}^{\text{rrs}}$). The version with the dependency scheme solved 176 (roughly 4.5%) more instances than the version without. The scatter plot in Figure 5.2 deserves further attention. While the overall number of solved instances is higher for Qute-$\mathcal{D}^{\text{rrs}}$, the plot is skewed towards Qute-$\mathcal{D}^{\text{rrs}}$. We attribute this to a small overhead associated with the use of the dependency scheme, which is most apparent for the easiest formulas. The plot also shows that there are a few formulas solved by the plain version, but not by Qute-$\mathcal{D}^{\text{rrs}}$. This is only partly due to the additional time spent computing resolution paths, and is, in our opinion, in much larger part due to the heuristics being led off the right track towards a proof of the formula.

We found two families of instances where the increase in number of solved instances is even more significant, as is documented in Table 5.1. Particularly on the *matrix multiplication* and *reduction finding* benchmarks the dependency scheme provides a tremendous boost of performance, resulting in almost four times as many solved instances.

[2] http://www.qbflib.org/
[3] https://projects.informatik.uni-freiburg.de/users/4
[4] https://www.react.uni-saarland.de/tools/caqe
[5] https://github.com/lonsing/depqbf
[6] http://sat.inesc-id.pt/~mikolas/sw/qesto
[7] https://github.com/perebor/qute
[8] http://sat.inesc-id.pt/~mikolas/sw/areqs

Table 5.1: Number of instances solved by plain Qute vs Qute using the reflexive resolution-path dependency scheme on the 'matrix multiplication' and 'reduction finding' families of formulas, as well as on all instances.

	MM-family	RF-family	all instances
# of instances	334	2269	11262
solved by Qute (SAT / UNSAT)	34 (4/30)	423 (140/283)	3959 (1467/2492)
solved by Qute-$\mathcal{D}^{\text{rrs}}$ (SAT / UNSAT)	123 (4/119)	484 (144/340)	4135 (1489/2646)

5.4 Summary and Discussion

We presented the first practical implementation of $\mathcal{D}^{\text{rrs}}$ in a QBF solver. Thus, we have demonstrated that the strongest known tractable sound dependency scheme can be efficiently used in QBF solving. Our approach shows that dependency schemes can be fruitfully combined with dependency learning. Our algorithm for the computation of all resolution-path dependencies of a given variable may also be of independent interest.

While the additional prefix relaxation that comes from $\mathcal{D}^{\text{rrs}}$ is no cure-all for the hardness of QBF, we have found families of formulas where it provides a significant speedup. In particular, the use of the dependency scheme turned out very beneficial on the 'matrix multiplication' and 'reduction finding' classes, which are both practically relevant applications and further improvement using QBF would be valuable.

A possible direction for future work is to try to further improve the time bound of our algorithm for computing the resolution-path dependencies of a variable either by using data structures more suitable for this concrete scenario, or by preprocessing the formula. A succinct, possibly implicit, representation of $\mathcal{D}^{\text{rrs}}$ for use in other solver architectures would also be very interesting.

Publication Notes

The research in this chapter appeared in a paper published in the proceedings of the 22nd International Conference on Theory and Applications of Satisfiability Testing (SAT 2019) [PSS19c].

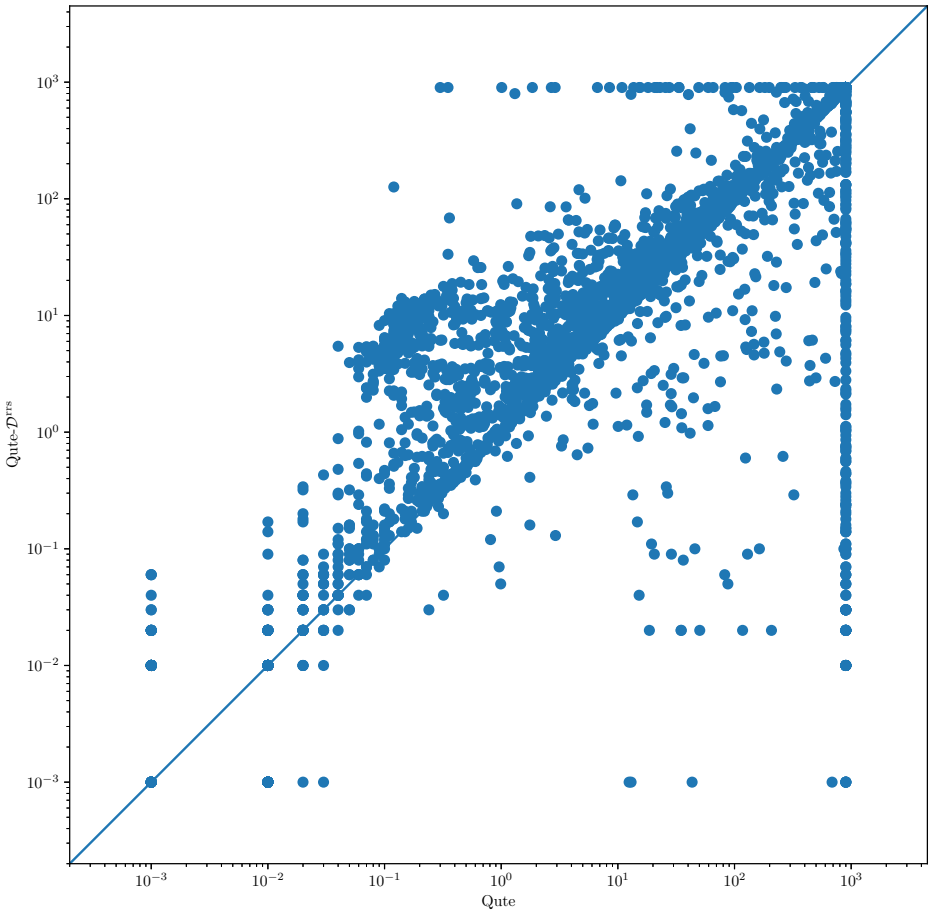

Figure 5.2: Runtimes of Qute with and without $\mathcal{D}^{\text{rrs}}$on all instances.

CHAPTER 6

Polynomial-Time Validation of QCDCL Certificates

In QBF solving, whether with or without dependency analysis, our main goal is to determine the truth value of a given formula. This is the basic ability that any solver must have, and we also frame questions in computational complexity theory in these terms—as decision problems. However, once we want to actually apply QBF to real world problems, we often find that it is necessary to get more from the solver than just a plain yes/no answer. Such is the case when we encode tasks from software verification and want to see the faulty counterexample, when we solve a synthesis problem and want to get the synthesized program, or when we encode a 2-player game and are interested in the winning strategy.

Additional information provided by a QBF solver is called a *certificate*—it certifies in an independently verifiable way that the solver's answer is correct. A certificate can have several forms, but we will use the term to refer to models and countermodels.

A certificate has two purposes: to provide a means of checking the correctness of the solver; and to serve itself as an object of interest, as described above. In both cases it is desirable to *validate* the certificate, i.e. to check that it is really correct. This is a complex process which ends with a query about the truth value of a propositional formula. We show how certificates from QCDCL solvers can be validated in polynomial time, without the need for calling a SAT solver in the last step.

We begin this chapter by describing the setting of the problem of QBF certificate validation. Then, in Sections 6.2 and 6.3, we present an algorithm that computes a RUP proof that can be used to replace the final call to the SAT solver by a simple proof check.

6.1 QBF Certificate Validation

For the sake of simplicity, we will only focus on false PCNF formulas. The results generalize to true formulas by duality, which will be discussed in Section 6.4.

Let φ be a CNF formula, let C be a boolean circuit. The substitution of C into φ, denoted by $\varphi[C]$, is simply the CNF formula φ in conjunction with a CNF encoding of C (which may contain additional auxiliary variables). Let $\Phi = \mathcal{Q}.\varphi$ be a false QBF in PCNF, let C be a boolean circuit whose inputs are existential variables of Φ and whose outputs are universal variables of Φ. The task of verifying that C is a countermodel of Φ is to verify that $\varphi[C]$ is unsatisfiable.

Some QCDCL QBF solvers are capable of outputting a *trace* that contains a (long-distance) Q-resolution refutation of the formula solved. From this refutation, a countermodel circuit can be computed by the Balabanov-Jiang (BJ) algorithm [BJ12], or by the extended Balabanov-Jiang-Janota-Widl (BJJW) algorithm [BJJW15] for long-distance Q-resolution. Let $\Phi = \mathcal{Q}.\varphi$ be a QBF, let $\mathcal{P}$ be a (long-distance) Q-resolution refutation of it, let $\textsc{cc}(\mathcal{P})$ be the countermodel circuit computed by the appropriate version of BJ/BJJW. The CNF formula that results from substitution of $\textsc{cc}(\mathcal{P})$ into φ as described in the previous paragraph, i.e., $\varphi[\textsc{cc}(\mathcal{P})]$, is denoted by $\Phi[\mathcal{P}]$, and is called the *validation formula* for the QBF Φ and the proof $\mathcal{P}$. This is the formula that must be checked for unsatisfiability in order to verify the correctness of the certificate $\textsc{cc}(\mathcal{P})$. We will now present a way how to directly compute a RUP proof for the validation formula out of the proof $\mathcal{P}$, thus obviating the need to use a SAT solver and making validation checks solvable in polynomial time.

6.2 RUP Proofs from Ordinary Q-Resolution

We will begin by describing a countermodel, and in particular its CNF version obtained by the Tseitin conversion, computed by BJ. For a full explanation of the algorithm we refer to the original paper [BJ12]. We illustrate the certificate extraction process on this example formula

$$\begin{aligned}\exists x_1, x_2\ \forall y\ \exists z \quad & (x_1 \vee x_2 \vee y \vee z) \wedge (x_1 \vee x_2 \vee \overline{z}) \wedge (x_1 \vee \overline{x_2}) \\ & \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{y} \vee \overline{z}) \wedge (\overline{x_1} \vee \overline{x_2} \vee z) \wedge (\overline{x_1} \vee x_2)\end{aligned}$$

along with its Q-resolution refutation:

(1)	$x_1 \vee x_2 \vee y \vee z$	(input)
(2)	$\overline{x_1} \vee \overline{x_2} \vee \overline{y} \vee \overline{z}$	(input)
(3)	$x_1 \vee x_2 \vee \overline{z}$	(input)
(4)	$\overline{x_1} \vee \overline{x_2} \vee z$	(input)
(5)	$x_1 \vee \overline{x_2}$	(input)
(6)	$\overline{x_1} \vee x_2$	(input)
(7)	$x_1 \vee x_2 \vee y$	(1, 3)
(8)	$x_1 \vee x_2$	(7)
(9)	$\overline{x_1} \vee \overline{x_2} \vee \overline{y}$	(2, 4)
(10)	$\overline{x_1} \vee \overline{x_2}$	(9)
(11)	x_1	(5, 8)
(12)	$\overline{x_1}$	(6, 10)
(13)	$\bot$	(11, 12)

Let $\mathcal{P}$ be a Q-resolution refutation of a formula $\Phi = \mathcal{Q}.\varphi$. BJ processes the clauses of $\mathcal{P}$ forward (we assume a topological ordering of the proof), and every time a conclusion R of a reduction step $R = R' - L$ (read the set of literals L is reduced from the clause R' to obtain the clause R) is encountered, for every literal ℓ from L either the clause R (if ℓ is positive) or the term $\overline{R}$ (if ℓ is negative) is pushed to what is called the *countermodel array* of $var(\ell)$ (cf. [BJ12]). At the end, the arrays represent the countermodel functions for their respective variables, in the following way:

Let u be a universal variable, and let its countermodel array have the entries $X_1, \ldots, X_n$. This array is interpreted by constructing a set of partial circuits. Let $f^u_n = X_n$. Then we define

$$f^u_k = \begin{cases} X_k \wedge f^u_{k+1} & \text{if } X_k \text{ is a clause,} \\ X_k \vee f^u_{k+1} & \text{if } X_k \text{ is a term,} \end{cases}$$

and finally $f^u = f^u_1$. The circuit f^u represents the countermodel function for the variable u. Intuitively, these circuits find the first reduction step whose conclusion is falsified, and set all of the reduced literals in the premise so that they are falsified too, which ensures that the falsified clause is implied by the conjunction of input clauses and hence at least one of those is falsified too.

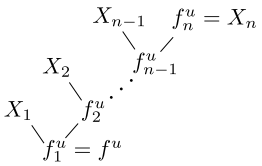

Figure 6.1: Schematic depiction of a countermodel circuit extracted by BJ. Each f_i is either an "and" or an "or" gate, depending on the context.

Let us see what this means on the example formula and proof. There is only one universal variable, so we will only build one countermodel array. Processing the clauses forward, the first conclusion of a reduction step that we encounter is (8), y is reduced in positive polarity, so we push the clause $(x_1 \vee x_2)$ to the countermodel array. Next, we encounter the conclusion (10), here y is reduced in negative polarity, so we push the negation of the conclusion $(\overline{x_1} \vee \overline{x_2})$, the term $(x_1 \wedge x_2)$. There are no more reduction steps, so the final countermodel array for y is $[(x_1 \vee x_2), (x_1 \wedge x_2)]$. According to the interpretation above, this results in the circuit $y = ((x_1 \vee x_2) \wedge (x_1 \wedge x_2)) = (x_1 \wedge x_2)$. It can be easily verified that this is indeed a countermodel for the formula.

Let us now examine how the circuit f^u can be translated into CNF for substitution into Φ. We can observe that the circuit f^u has a nested structure, in which first the values of all of the X_k are evaluated, which are then further processed by the circuit to obtain the value for u. Every X_k is either a clause or a term corresponding to a conclusion of a reduction step in $\mathcal{P}$. Let $R_1, \ldots, R_N$ be all conclusions of reduction steps in $\mathcal{P}$, in the same order as they appear in the proof. Then for every X_k there is i_k such that $X_k = R_{i_k}$

or $X_k = \overline{R_{i_k}}$. Let us define variables $g_i = R_i$ for $1 \leq i \leq N$ using the set of clauses

$$G = \{\{(\overline{g_i} \vee R_i)\} \cup \{(g_i \vee \bar{\ell}) \mid \ell \in R_i\} \mid 1 \leq i \leq N\}.$$

Rather than encoding each countermodel circuit using its X_k members, we will leverage the fact that X_k is either equivalent to g_{i_k} or to $\overline{g_{i_k}}$ and replace it by the suitable polarity. This way, the recursive definitions of f_k^u boil down to

$$f_n^u = \begin{cases} g_{i_n} & \text{if } X_n \text{ is a clause,} \\ \overline{g_{i_n}} & \text{if } X_n \text{ is a term,} \end{cases}$$

and for $1 \leq k < n$

$$f_k^u = \begin{cases} g_{i_k} \wedge f_{k+1}^u & \text{if } X_k \text{ is a clause,} \\ \overline{g_{i_k}} \vee f_{k+1}^u & \text{if } X_k \text{ is a term.} \end{cases}$$

At this point, since the countermodel arrays are populated in the order of the proof, we can observe the following:

Observation 3. *Whenever g_{i_k} and $g_{i_{k'}}$ appear in the same circuit and $k < k'$, i.e., g_{i_k} comes before $g_{i_{k'}}$ in the corresponding countermodel array, then also $i_k < i_{k'}$, i.e., the reduction step corresponding to g_{i_k} also comes before the one corresponding to $g_{i_{k'}}$.*

Using the simplified circuits with the variables g_i, we can finally produce an encoding into CNF. By using the Tseitin conversion, we get the clauses

$$F_n^u = \begin{cases} (f_n^u \vee \overline{g_{i_n}}) \wedge (\overline{f_n^u} \vee g_{i_n}) & \text{if } X_n \text{ is a clause,} \\ \underbrace{(f_n^u \vee g_{i_n})}_{F_{n,1}^u} \wedge \underbrace{(\overline{f_n^u} \vee \overline{g_{i_n}})}_{F_{n,2}^u} & \text{if } X_n \text{ is a term,} \end{cases}$$

and for $1 \leq k < n$

$$F_k^u = \begin{cases} (f_k^u \vee \overline{g_{i_k}} \vee \overline{f_{k+1}^u}) \wedge (\overline{f_k^u} \vee g_{i_k}) \wedge (\overline{f_k^u} \vee f_{k+1}^u) & \text{if } X_k \text{ is a clause,} \\ \underbrace{(\overline{f_k^u} \vee \overline{g_{i_k}} \vee f_{k+1}^u)}_{F_{k,1}^u} \wedge \underbrace{(f_k^u \vee g_{i_k})}_{F_{k,2}^u} \wedge \underbrace{(f_k^u \vee \overline{f_{k+1}^u})}_{F_{k,3}^u} & \text{if } X_k \text{ is a term.} \end{cases}$$

In our running example, we have two reduction steps, there are therefore two definitions of g-variables, namely $g_1 = (x_1 \vee x_2)$ and $g_2 = (\overline{x_1} \vee \overline{x_2})$. If we replace the actual entries in the countermodel array by the g-variables, we get the array $[g_1, \overline{g_2}]$ and the corresponding circuit $y = g_1 \wedge \overline{g_2}$. Its CNF encoding is

$$(y \vee \overline{g_1} \vee g_2) \wedge (\overline{y} \vee g_1) \wedge (\overline{y} \vee \overline{g_2}).$$

Starting from a formula $\Phi = \mathcal{Q}.\varphi$ and its Q-resolution refutation $\mathcal{P}$, G will denote the set of clauses defining the g_i and F will denote the set of clauses F_k^u (for all universals

u and appropriate k) defining the countermodel. The validation formula $\Phi[\mathcal{P}]$ is then $\varphi \wedge G \wedge F$ and we will now present a RUP proof for it.

We will need the following notation. Let x, y be variables of a propositional formula φ, let τ be an assignment to variables of φ. We write $x \cong^{\varphi}_{\tau} y$ if, for every extension σ of τ that defines x or y, either unit propagation in $\varphi[\sigma]$ causes a conflict or $\sigma'(x) = \sigma'(y)$, where σ' is the closure of σ with respect to unit propagation. If φ is understood from the context, we may drop the superscript, likewise, if τ is the empty assignment, we may drop the subscript.

Lemma 13. *Let u be a universal variable of Φ whose countermodel array has n entries and the corresponding g-variables are $g_{i_1}, \ldots, g_{i_n}$. For $1 \leq k \leq n$ let τ_k be a partial assignment (to variables of $\Phi[\mathcal{P}]$) which sets $g_{i_1}, \ldots, g_{i_{k-1}}$ to true. Then $f^u \cong_{\tau_k} f^u_k$.*

Proof. We can see that the clauses $F^u_{j,2}[\tau_k]$ are satisfied by g_{i_k} and $\overline{g_{i_k}}$ disappears from $F^u_{j,1}[\tau_k]$ for $1 \leq j < k$. The clauses $F^u_{j,1}[\tau_k]$ and $F^u_{j,3}[\tau_k]$ we are left with encode precisely $f^u_j \cong f^u_{j+1}$. Together, we have that under the assignment τ_k, $f^u = f^u_1 \cong f^u_k$, or in other words $f^u \cong_{\tau_k} f^u_k$. □

The following lemma asserts that the intuition about how countermodel circuits find the first falsified conclusion and set the variable accordingly is indeed true.

Lemma 14. *For $1 \leq i \leq N$ let τ_i be a partial assignment (to variables of $\Phi[\mathcal{P}]$) which sets $g_1, \ldots, g_{i-1}$ to true and g_i to false. Let ℓ be a universal literal that is reduced in the reduction step leading to R_i. Under the assignment τ_i unit propagation (in $\Phi[\mathcal{P}]$) causes a conflict or derives $\overline{\ell}$.*

Proof. Let us assume unit propagation does not cause a conflict. Let $u = var(\ell)$, g_i occurs in the countermodel array of u as some g_{i_k}. If ℓ is positive, $F^u_{k,2}$ together with $\overline{g_{i_k}}$ propagate $\overline{f^u_k}$. If ℓ is negative, $F^u_{k,2}$ together with $\overline{g_{i_k}}$ propagate f^u_k. We can use Observation 3 to see that all $g_{i_{k'}}$ with $k' < k$ are set to true and Lemma 13 applies, so that $f^u \cong f^u_k$ and the value for u propagated is false if ℓ is negative and true if ℓ is positive. Either way, this means that $\overline{\ell}$ is propagated. □

With Lemma 14, we can describe how to construct a RUP proof for $\Phi[\mathcal{P}]$ from $\mathcal{P}$.

Theorem 10. *Let $\mathcal{P}$ be a Q-resolution refutation of the formula $\Phi = \mathcal{Q}.\varphi$. Then there exists a RUP proof of unsatisfiability of the validation formula $\Phi[\mathcal{P}]$ of size $O(|\mathcal{P}|)$, and this proof can be computed in $O(|\mathcal{P}|)$ time.*

Proof. Let $\mathcal{P}'$ be $\mathcal{P}$ with each conclusion R_i replaced by the unit clause (g_i), and with the input clauses omitted. We claim that $\mathcal{P}'$ is a RUP proof of unsatisfiability of $\Phi[\mathcal{P}]$. Since resolvents are always RUP with respect to their premises we only need to verify that all (g_i) are RUP too. Let $R_i = R'_i - L$ be a reduction step, let $\ell \in L$ be one of the

universal literals reduced to obtain R_i, let $u = var(l)$. We need to prove that setting (g_i) to false causes a conflict by unit propagation. At the time when (g_i) is inserted into the proof, all (g_j) with $j < i$ have already been inserted and since they are unit clauses, all g_j with $j < i$ are set to true by unit propagation. Adding to that the assignment $\overline{g_i}$, the conditions of Lemma 14 are satisfied and so either unit propagation causes a conflict (in which case we are done), or $\overline{\ell}$ is propagated. Since ℓ was chosen without loss of generality, all literals in L are propagated to false, and since $\overline{g_i}$ trivially propagates all literals of R_i to false, R'_i is falsified and a conflict is reached as required. Clearly, the size of $\mathcal{P}'$ is bounded by the size of $\mathcal{P}$, and it can be computed in time $O(|\mathcal{P}|)$ as the amount of work per each clause of $\mathcal{P}$ is proportional to its size. □

For example, the RUP proof constructed according to Theorem 10 from the example Q-resolution proof would consist of the following sequence of clauses:

$$(x_1 \vee x_2 \vee y),\ (g_1),\ (\overline{x_1} \vee \overline{x_2} \vee \overline{y}),\ (g_2),\ (x_1),\ (\overline{x_1}),\ \bot$$

6.3 RUP Proofs from Long-Distance Q-Resolution

With long-distance Q-resolution, we cannot directly use the clauses of the refutation in the RUP proof as we did in the proof of Theorem 10, because these clauses may be tautological. Instead, we adopt the approach that was used in the paper of Balabanov et al. [BJJW15] in order to generalize BJ to long-distance Q-resolution proofs. The following definition is taken from the paper of Balabanov et al. [BJJW15], with a slight change of notation.

Definition 18. *Let $\mathcal{P}$ be a long-distance Q-resolution refutation of the QBF $\Phi = \mathcal{Q}.\varphi$. Let $C \in \mathcal{P}$ be a clause, $\ell \in C$ a literal and $u = var(\ell)$. The* phase function *of the variable u in the clause C, denoted by $u^\phi(C)$, is a boolean function defined recursively as follows:*

- *if C is an input clause, then $u^\phi(C) = 1$ if $\ell = u$, otherwise $u^\phi(C) = 0$*
- *if C is the result of application of universal reduction on the clause C', $u^\phi(C) = u^\phi(C')$*
- *if C is the resolvent of C_1 and C_2 on the pivot literal p, $p \in C_1$, $\overline{p} \in C_2$, then if $u \notin var(C_1)$, then $u^\phi(C) = u^\phi(C_2)$, if $u \notin var(C_2)$ or $u^\phi(C_1) = u^\phi(C_2)$, then $u^\phi(C) = u^\phi(C_1)$, otherwise $u^\phi(C) = (p \wedge u^\phi(C_2)) \vee (\overline{p} \wedge u^\phi(C_1))$*

The effective literal *of ℓ in C, denoted by $\ell^\epsilon(C)$, is a literal that satisfies $\ell^\epsilon(C) \Leftrightarrow (u \Leftrightarrow u^\phi(C))$. The* shadow clause *of C is the clause $C^\sigma = \bigvee_{\ell \in C} \ell^\epsilon(C)$.*

The phase function intuitively tells us, under a given assignment to previous variables in the quantifier prefix, what is the phase in which a given universal variable would have appeared in a given clause, had we restricted the proof using that assignment. The

effective literal is a literal which, based on an assignment to previous existential variables, is equivalent to the polarity of its variable indicated by the phase function. Note that in the case when the phase function is constant, i.e. 0 or 1, the effective literal of any literal is simply the literal itself. In such cases we say that the literal is *unmerged*. Literals that are not unmerged are *merged*.

We will now present a description of the countermodel computed by BJJW from a long-distance Q-resolution refutation. In order to do that, we adapt the notation from Section 6.2. Let $\mathcal{P}$ be a long-distance Q-resolution refutation of a formula $\Phi = \mathcal{Q}.\varphi$. The conclusions of reduction steps in $\mathcal{P}$, in the same order as they appear, are denoted by $R_1, \ldots, R_N$. The variables g_i, $1 \leq i \leq N$, are now equivalent to the shadow clauses ${R_i}^\sigma$ instead of R_i themselves. Since BJJW keeps track of the phase function of every universal variable in every clause, we will use a variable $u^\phi(C)$ to denote the output of the phase function. We will also have variables $\ell^\epsilon(C)$ for the effective literals. In the case of unmerged literals, this will simply be ℓ. By H we will denote the conjunction of all clauses that encode the circuits which define phase variables and effective literals.

The partial countermodel circuits f_k^u from the previous section are slightly more complicated now. Let $R_i = R'_i - L$ be a reduction step, let $\ell \in L$ be a literal that is being reduced, let $u = var(\ell)$. If ℓ is unmerged, ${R_i}^\sigma$ is pushed into the countermodel array of u, similarly as in the case of ordinary Q-resolution. However, if ℓ is merged, we first require that both ℓ and $\overline{\ell}$ be reduced at the same time (merged literals arise from merges, so they are always in both polarities in a clause), and as such two entries are pushed into the countermodel array of u, namely ${R_i}^\sigma \vee \overline{u^\phi(R'_i)}$ and right afterwards $\overline{{R_i}^\sigma} \wedge \overline{u^\phi(R'_i)}$. The intuition for why these entries are added is the following: if the phase $u^\phi(R'_i)$ of ℓ in R'_i is positive, and the (shadow clause of the) conclusion is falsified, set u to false, otherwise if the phase is negative and the conclusion is falsified, set u to true, each time falsifying the effective literal $\ell^\epsilon(R'_i)$. This is analogous to the ordinary case, where when the conclusion is falsified, the reduced literal is set so that it is falsified, only in this case we falsify the effective literal.

Now, for the sake of simplicity of presentation, we will treat unmerged literals the same way as merged ones. This means that even for unmerged reduced literals we push two entries into the countermodel array, $R_i \vee \overline{u^\phi(R'_i)}$ and $\overline{R_i} \wedge \overline{u^\phi(R'_i)}$. It is easy to see that if $u^\phi(R'_i) = 1$, the term becomes falsified and the clause reduces to just R_i, while if $u^\phi(R'_i) = 0$, the clause becomes satisfied and the term reduces to just $\overline{R_i}$. In each case, the circuit is equivalent to what we would have produced by pushing just the one entry as previously.

Let $X_1, \ldots, X_{2n}$ be the entries in the countermodel array of a universal variable u. Each X_{2k-1} is ${R_{i_k}}^\sigma \vee \overline{u^\phi(R'_{i_k})}$ and X_{2k} is $\overline{{R_{i_k}}^\sigma} \wedge \overline{u^\phi(R'_{i_k})}$. We have already defined $g_i = {R_i}^\sigma$, but since each entry in the countermodel array still contains two variables even after replacing ${R_{i_k}}^\sigma$ with g_{i_k}, we will define the auxiliary variables $f'^u_{2k-1} = g_{i_k} \vee \overline{u^\phi(R'_{i_k})}$ and

$f'^u_{2k} = \overline{g_{i_k}} \wedge \overline{u^\phi(R'_{i_k})}$ using the following sets of clauses (for $1 \leq k \leq n$):

$$F'^u_{2k-1} = \overbrace{(\overline{f'^u_{2k-1}} \vee g_{i_k} \vee \overline{u^\phi(R'_{i_k})})}^{F'^u_{2k-1,1}} \wedge \overbrace{(f'^u_{2k-1} \vee \overline{g_{i_k}})}^{F'^u_{2k-1,2}} \wedge \overbrace{(f'^u_{2k-1} \vee u^\phi(R'_{i_k}))}^{F'^u_{2k-1,3}}$$
$$F'^u_{2k} = \underbrace{(f'^u_{2k} \vee g_{i_k} \vee u^\phi(R'_{i_k}))}_{F'^u_{2k,1}} \wedge \underbrace{(\overline{f'^u_{2k}} \vee \overline{g_{i_k}})}_{F'^u_{2k,2}} \wedge \underbrace{(\overline{f'^u_{2k}} \vee \overline{u^\phi(R'_{i_k})})}_{F'^u_{2k,3}}$$

Let F' be the conjunction of all F'^u_k for all universal variables u and all appropriate k. The following is immediate from the clauses F'.

Observation 4. *Setting g_{i_k} to true causes unit propagation to set f'^u_{2k-1} and $\overline{f'^u_{2k}}$.*

Finally, we are ready to present the set F of clauses which encode the countermodel circuit:

$$F^u_{2n,1} = (f^u_{2n} \vee \overline{f'^u_{2n}}), \quad F^u_{2n,2} = (\overline{f^u_{2n}}, f'^u_{2n}),$$

and for $1 \leq k < 2n$

$$F^u_k = \begin{cases} \underbrace{(f^u_k \vee \overline{f'^u_k} \vee \overline{f^u_{k+1}})}_{F^u_{k,1}} \wedge \underbrace{(\overline{f^u_k}, f'^u_k)}_{F^u_{k,2}} \wedge \underbrace{(\overline{f^u_k}, f^u_{k+1})}_{F^u_{k,3}} & \text{if } k \text{ is odd,} \\ \underbrace{(\overline{f^u_k} \vee f'^u_k \vee f^u_{k+1})}_{F^u_{k,1}} \wedge \underbrace{(f^u_k, \overline{f'^u_k})}_{F^u_{k,2}} \wedge \underbrace{(f^u_k, \overline{f^u_{k+1}})}_{F^u_{k,3}} & \text{if } k \text{ is even.} \end{cases}$$

Similarly as before, let F be the conjunction of all F^u_k for all appropriate u and k, and let G be the conjunction of the clauses defining the equivalences $g_i \Leftrightarrow R_i{}^\sigma$. Then, the validation formula for Φ and $\mathcal{P}$ is $\Phi[\mathcal{P}] = \varphi \wedge F \wedge F' \wedge G \wedge H$.

The following are analogues of Lemmas 13 and 14.

Lemma 15. *Let u be a universal variable of Φ whose countermodel array has $2n$ entries and the corresponding g-variables are $g_{i_1}, \ldots, g_{i_{2n}}$. For $1 \leq k \leq 2n$ let τ_k be a partial assignment (to variables of $\Phi[\mathcal{P}]$) which sets $g_{i_1}, \ldots, g_{i_{k-1}}$ to true. Then $f^u \cong_{\tau_k} f^u_{2k-1}$.*

Proof. Let $1 \leq j < k$. Applying Observation 4, we see that f'^u_{2j-1} and $\overline{f'^u_{2j}}$ are propagated, in each case, inspecting the restricted clauses that remain, we see that $f^u_{2j-1} \cong_{\tau_k} f^u_{2j}$ and $f^u_{2j} \cong_{\tau_k} f^u_{2j+1}$. Altogether, we get $f^u \cong_{\tau_k} f^u_k$. □

Lemma 16. *For $1 \leq i \leq N$ let τ_i be a partial assignment (to variables of $\Phi[\mathcal{P}]$) which sets $g_1, \ldots, g_{i-1}$ to true and g_i to false. Let u be a universal variable of Φ in whose countermodel g_i appears as some g_{i_k}. Let R_i be the corresponding reduction step, obtained from R'_i. Then, under either of the assignments $\tau_i \cup u^\phi(R'_i)$ and $\tau_i \cup \overline{u^\phi(R'_i)}$, unit propagation (in $\Phi[\mathcal{P}]$) causes a conflict or derives $\overline{u^\epsilon(R'_i)}$.*

Proof. Assume unit propagation does not cause a conflict. Let us assume $u^\phi(R'_i)$ first. Since we have $\overline{g_{i_k}} \wedge u^\phi(R'_i)$, the clause $F'^u_{2k-1,1}$ propagates $\overline{f'^u_{2k-1}}$, which in turn propagates $\overline{f^u_{2k-1}}$. Since $g_1, \ldots, g_{i-1}$ are set to true, Lemma 15 applies and the value of f^u_{2k-1} is propagated for the value of u, meaning $\overline{u}$ is propagated. Together with the assumption $u^\phi(R'_i)$, we have that the effective literal $u^\epsilon(R'_i)$ is set to false by unit propagation.

If on the other hand we assume $\overline{u^\phi(R'_i)}$, f'^u_{2k-1} is propagated from $F'^u_{2k-1,3}$, which means that the restricted clauses F^u_{2k-1} now encode $f^u_{2k-1} \cong f^u_{2k}$. Also, $F'^u_{2k,1}$ propagates f'^u_{2k}, which in turn propagates f^u_{2k}. Since $g_1, \ldots, g_{i-1}$ are set to true, Lemma 15 applies and the value of f^u_{2k} is propagated for the value of u, meaning u is propagated. Together with the assumption $\overline{u^\phi(R'_i)}$, we have that the effective literal $u^\epsilon(R'_i)$ is set to false by unit propagation. □

While in the case of ordinary Q-resolution, the resolvent of two clauses is always RUP with respect to those clauses, this is not true in the case of long-distance Q-resolution. This is due to the fact that if a merge occurs, a fresh effective literal is introduced in the resolvent, and just falsifying this new fresh literal without the knowledge of the value of the corresponding phase variable does not cause the effective literals in the premises of the resolution step to become falsified. Therefore, we first prove that a set of extra clauses can be derived from the definitions of phase functions and effective literals. These clauses will then empower unit propagation to deal with merged effective literals the same way as with unmerged ones.

Let C be the resolvent of C_1 and C_2 on the pivot literal $p \in C_1$ (and $\overline{p} \in C_2$). Let $\ell \in C_1$, $\overline{\ell} \in C_2$, $u = var(\ell)$ be a universal literal such that $u^\phi(C_1) \neq u^\phi(C_2)$, i.e. u is being merged in this resolution step. Then the clauses E^u_{C,C_1} and E^u_{C,C_2} are defined as follows:

$$E^u_{C,C_1} = (u^\epsilon(C) \vee p \vee \overline{u^\epsilon(C_1)}), \quad E^u_{C,C_2} = (u^\epsilon(C) \vee \overline{p} \vee \overline{u^\epsilon(C_2)}).$$

We will denote by E the set of all $E^u_{C,D}$ for appropriate premise D, resolvent C, and merged literal u. The clauses of E will provide us with a direct relationship between successive effective literals of one variable. They express one direction of the conditional dependence of an effective literal on the previous effective literals—if an effective literal is false, then based on the value of the pivot variable, the corresponding previous effective literal must be false too.

Lemma 17. *All clauses of E are derivable by RUP from H. The combined size of the RUP proofs is $O(|\mathcal{P}|)$ and they are computable in $O(|\mathcal{P}|)$ time.*

Proof. Let $E^u_{C,D} \in E$, let $p \in D$ be the pivot literal. It can be easily verified by unit propagation on the definitions of phase functions and effective literals that the following is the required RUP proof:

$$(u^\epsilon(C) \vee p \vee \overline{u^\epsilon(D)} \vee u^\phi(C)),\ (E^u_{C,D})$$

Clearly, per each resolution step, these proofs only take up constant space and are computable in constant time, resulting in an overall linear bound. □

We now state the main result of this section.

Theorem 11. *Let $\mathcal{P}$ be a long-distance Q-resolution refutation of the formula $\Phi = \mathcal{Q}.\varphi$. Then there exists a RUP proof of unsatisfiability of the validation formula $\Phi[\mathcal{P}]$ of size $O(|\mathcal{P}|)$, and this proof can be computed in $O(|\mathcal{P}|)$ time.*

Proof. Let F, F', G, H be the sets of clauses described above. Let $\mathcal{P}_E$ be the combined RUP proof of all clauses of E from H from Lemma 17. For each reduction step $R_i = R'_i - L$, let $V_L = \{u_1, \ldots, u_s\}$ be some ordering of $var(L)$, $s = |var(L)|$. We define the variables $g_{i,j}$ for each $1 \leq j \leq s$ recursively in the following way (recall that g_i is defined equal to ${R_i}^\sigma$):

$$g_{i,s} = g_i$$
$$g_{i,j} = g_{i,j+1} \vee {u_{j+1}}^\epsilon(R'_i) \text{ for } 1 \leq j < s$$

Essentially, $g_{i,j}$ is equivalent to $\left(R'_i - \{u_1, \overline{u_1}, \ldots, u_j, \overline{u_j}\}\right)^\sigma$, or in other words, to an intermediate shadow clause with only a subset of the variables reduced. Let $\mathcal{P}'$ be the sequence of shadow clauses of $\mathcal{P}$ with each ${R_i}^\sigma$ replaced by the following sequence of clauses:

$$(g_{i,1} \vee {u_1}^\phi(R'_i)),\ (g_{i,1}),\ \ldots,\ (g_{i,s} \vee {u_s}^\phi(R'_i)),\ (g_{i,s}),$$

and with $C \vee p$ inserted before each resolvent C in whose resolution step on the pivot literal p a variable is being merged. Then, we claim that $\mathcal{P}_E$ followed by $\mathcal{P}'$ is a linear-size linear-time-computable RUP proof of unsatisfiability of $\Phi[\mathcal{P}]$. Since by Lemma 17 $\mathcal{P}_E$ satisfies the requirements, it only remains to verify that $\mathcal{P}'$ is a valid RUP proof of the required size. Moreover, since we have already proven the clauses E by $\mathcal{P}_E$, we can use them to argue the soundness of $\mathcal{P}'$.

The rest of the proof will proceed in two stages. In the first stage, we verify that the shadow clauses of the resolvents as they are introduced into the RUP proof are RUP indeed, in the second stage we verify the soundness of reduction steps.

For the first stage, we will use the clause set E. It is sufficient to check the RUP property of resolvents in whose resolution steps a variable is merged, the ordinary resolvents are trivially RUP. Let C be the resolvent of C_1 and C_2 on the pivot p, $p \in C_1$. Let $u_1, \ldots, u_m$ be the variables that are merged in this reduction step. First, we have to verify that $C^\sigma \vee p$ is RUP. Setting all literals of C^σ and p to false falsifies everything in ${C_1}^\sigma$, except the effective literals ${u_1}^\epsilon(C_1), \ldots, {u_m}^\epsilon(C_1)$. Let $1 \leq j \leq m$ and consider the clause $E^{u_j}_{C,C_1}$. By assumption, ${u_j}^\epsilon(C)$ and p are set to false, which causes $E^{u_j}_{C,C_1}$ to propagate $\overline{{u_j}^\epsilon(C_1)}$. This is done for all u_j, altogether falsifying the clause ${C_1}^\sigma$ and meaning $C^\sigma \vee x$ is indeed RUP. In order to see that C^σ is RUP once we have $C^\sigma \vee p$ consider that setting C^σ to false sets p to true by unit propagation. Now, for $1 \leq j \leq m$, ${u_j}^\epsilon(C)$ is set to false and p to true, so the clause $E^{u_j}_{C,C_2}$ propagates $\overline{{u_j}^\epsilon(C_2)}$, ultimately falsifying ${C_2}^\sigma$.

Let $R_i = R'_i - L$ be a reduction step, let $V_L = \{u_1, \ldots, u_s\}$. We will prove by induction on s that all of the intermediate clauses introduced for this reduction step are RUP. If $s = 1$, we have that g_i occurs in the countermodel array of u_1 as some g_{i_k}. When $g_{i,1}$ is set to false, g_i is set to false by unit propagation, and so is ${R_i}^\sigma$. If in addition ${u_1}^\phi(R'_i)$ is set to false, either a conflict is reached, or ${u_1}^\epsilon(R'_i)$ is set to false by unit propagation by Lemma 16. Since ${R'_i}^\sigma = {R_i}^\sigma \vee {u_1}^\epsilon(R'_i)$ and ${R_i}^\sigma$ is falsified, we have that ${R'_i}^\sigma$ is now falsified too. This means that the clause $(g_{i,1} \vee {u_1}^\phi(R'_i))$ is RUP. To see that afterwards the clause $(g_{i,1})$ is RUP, consider that in this case the previous clause will propagate ${u_1}^\phi(R'_i)$, by Lemma 16 ${u_1}^\epsilon(R'_i)$ is set to false by unit propagation, and ${R'_i}^\sigma$ is again falsified.

For $s > 1$ we argue similarly using $g_{i,s}$. When $g_{i,s}$ is set to false, g_i is set to false by unit propagation, and so is ${R_i}^\sigma$. If in addition ${u_s}^\phi(R'_i)$ is set to false, either a conflict is reached, or ${u_s}^\epsilon(R'_i)$ is set to false by unit propagation by Lemma 16. Since $g_{i,s-1} = g_{i,s} \vee {u_s}^\epsilon(R'_i)$, unit propagation sets $g_{i,s-1}$ to false, a conflict. To see that afterwards the clause $(g_{i,s})$ is RUP, consider that in this case the previous clause will propagate ${u_1}^\phi(R'_i)$, by Lemma 16 ${u_j}^\epsilon(R'_i)$ is set to false by unit propagation, and $g_{i,s-1}$ is again falsified.

This concludes the proof of soundness of $\mathcal{P}'$. As for the time and space claims, consider that for each resolvent and reduction step, only a linear number of literals (in the size of the resolvent or premise of reduction step) is introduced into the proof. These literals can easily be computed in linear time as we only need to know the phase variables, effective literals, and the $g_{i,j}$ variables, which we create freshly while processing the reduction step. □

Finally, let us point out that even though we presented concrete CNF encodings for many of the circuits, other encodings can work as well. Namely, it is sufficient if the encodings contain the g-variables (because these are present in the RUP proof) and satisfy the unit-propagation properties of the lemmas.

6.4 True Formulas

In this section we show how to derive analogues of Theorems 10 and 11 for true formulas. Let us start with the case of a (long-distance) Q-consensus proof $\mathcal{P}$ of a true PDNF formula $\Phi = \mathcal{Q}.\varphi$. In this case the validation formula $\Phi[\mathcal{P}]$ for the model $\textsc{cc}(\mathcal{P})$ is the DNF φ in disjunction with $\textsc{dnf}(\textsc{cc}(\mathcal{P}))$. The task of validation of the model $\textsc{cc}(\mathcal{P})$ is to check that $\Phi[\mathcal{P}]$ is valid, and checking the validity of $\Phi[\mathcal{P}]$ is equivalent to checking that the CNF $\overline{\Phi[\mathcal{P}]}$ is unsatisfiable.

Theorem 12. *Let $\mathcal{P}$ be a long-distance Q-consensus proof of the PDNF formula $\Phi = \mathcal{Q}.\varphi$. Then there exists a RUP proof of unsatisfiability of the negated validation formula $\overline{\Phi[\mathcal{P}]}$ of size $O(|\mathcal{P}|)$, and it can be computed in $O(|\mathcal{P}|)$ time.*

Proof. We observe that the countermodels extracted by BJ/BJJW from $\mathcal{P}$ and from its negation $\overline{\mathcal{P}}$ (the sequence of negated terms of $\mathcal{P}$) are in fact the same (we have not

discussed the variants of BJ/BJJW for true formulas here, but check the definitions in [BJ12, BJJW15] to see that this trivially holds), which means that their CNF and DNF encodings are negations of one another. This means that

$$\overline{\Phi[\mathcal{P}]} = \overline{\varphi \vee \textsc{dnf}(\textsc{cc}(\mathcal{P}))} = \overline{\varphi} \wedge \overline{\textsc{dnf}(\textsc{cc}(\mathcal{P}))} = \overline{\varphi} \wedge \textsc{cnf}(\textsc{cc}(\overline{\mathcal{P}})) = \overline{\Phi}[\overline{\mathcal{P}}],$$

and we can apply Theorem 11 on $\overline{\Phi}$ and $\overline{\mathcal{P}}$. □

For a Q-consensus proof $\mathcal{P}$ of a true PCNF formula $\Phi = \mathcal{Q}.\varphi$ let us first clarify what the validation formula looks like. We would need to check the validity of $\varphi \vee \textsc{dnf}(\textsc{cc}(\mathcal{P}))$, but φ is a CNF and $\textsc{cc}(\mathcal{P})$ must be encoded as a DNF for validity checking. Therefore, we need to first transform φ to DNF using the Tseitin transformation as follows. Suppose $\varphi = C_1 \wedge \cdots \wedge C_n$. We will define the clause variables $c_i = C_i$ and represent $\textsc{dnf}(\varphi)$ as follows:

$$\textsc{dnf}(\varphi) = \bigvee_{i=1}^{n} \left[(c_i \wedge \overline{C_i}) \vee \bigvee_{\ell \in C_i} (\overline{c_i} \wedge \ell)\right] \vee (c_1 \wedge \cdots \wedge c_n).$$

The validation formula $\Phi[\mathcal{P}]$ is then $\textsc{dnf}(\varphi) \vee \textsc{dnf}(\textsc{cc}(\mathcal{P}))$. As before, instead of checking the validity of $\Phi[\mathcal{P}]$, we will check the unsatisfiability of $\overline{\Phi[\mathcal{P}]}$.

Theorem 13. *Let $\mathcal{P}$ be a long-distance Q-consensus proof of the PCNF formula $\Phi = \mathcal{Q}.\varphi$ with the set of initial terms μ. If every clause from $\overline{\mu}$ is RUP with respect to $\overline{\textsc{dnf}(\varphi)}$, then there exists a RUP proof of unsatisfiability of the negated validation formula $\overline{\Phi[\mathcal{P}]}$ of size $O(|\mathcal{P}|)$, and it can be computed in $O(|\mathcal{P}|)$ time.*

Proof. Let $M = \mathcal{Q}.\mu$ be the PDNF consisting of the initial terms. Using Theorem 12, we obtain a RUP proof for the negated validation formula $\overline{M[\mathcal{P}]} = \overline{M}[\overline{\mathcal{P}}] = \overline{\mu} \wedge \textsc{cnf}(\textsc{cc}(\overline{\mathcal{P}}))$. By prepending $\overline{\mu}$ to this proof, we obtain a RUP proof of $\overline{\textsc{dnf}(\varphi)} \wedge \textsc{cnf}(\textsc{cc}(\overline{\mathcal{P}})) = \overline{\Phi}[\overline{\mathcal{P}}]$ of size $O(|\mathcal{P}| + |\mu|) = O(|\mathcal{P}|)$. □

There are two common ways of obtaining initial terms. One is to transform the CNF φ to DNF [JM17], in which case there is nothing to prove, because the negated initial terms are directly members of $\overline{\textsc{dnf}(\varphi)}$ and therefore RUP. The other way is to produce hitting sets of the clauses of φ. In this case, since every initial term is a hitting set of the clauses $C_1, \ldots, C_n$, we have that for every initial term I and for every clause C_i, there is always a clause of $\textsc{cnf}(\overline{\varphi})$ of the form $(c_i \vee \overline{\ell})$, such that $\ell \in I$. Therefore, by assuming the negation of a negated initial term, i.e. the term itself, unit propagation will propagate c_i for all i, which in turn causes a conflict with the clause $(\overline{c_1} \vee \cdots \vee \overline{c_n})$. Therefore, every clause in $\neg\mu$ is indeed RUP with respect to $\overline{\textsc{dnf}(\varphi)}$ and Theorem 13 applies.

Finally, in the paragraph above we mentioned that initial terms are hitting sets of the clauses of φ (in one of the cases). In fact, this need not always be true, since the hitting sets might have existential reduction applied to them first according to the *model generation* rule [GNT06]. Since it is no problem for the QBF solver to output the original hitting set without applying existential reduction, but very difficult (NP-hard in general)

for the proof-checker to recover it, we suggest to strengthen the conditions on the QRP proof format by requiring that the initial terms be full hitting sets. If this condition is not met our algorithm may fail to produce valid RUP proofs for true PCNF formulas. Fortunately DepQBF always generated terms that happened to be full hitting sets in our experiments.

6.5 Experiments

We implemented the algorithm of Theorem 11, which generalizes Theorem 10, in a tool called `qrp2rup` (`https://www.ac.tuwien.ac.at/research/certificates/`) and evaluated the performance compared to various other approaches to certificate validation. In particular, since our tool is also capable of emitting deletion information for DRAT-trim, we evaluated the following six configurations of certificate extractors and validators:

- `qrp2rup` with *deletion* information and validation by DRAT-trim,
- `qrp2rup` without deletion information (*plain*) and validation by DRAT-trim,
- `qrp2rup` and validation by Lingeling (ignoring the RUP proof),
- `qrp2rup` and validation by Glucose (ignoring the RUP proof),
- QBFcert and validation by Lingeling,
- QBFcert and validation by Glucose.

We also experimented with configurations of DRAT-trim that used forward checking (instead of the default backward checking), but excluded the results due to systematically inferior performance. Note that since QBFcert cannot handle long-distance Q-resolution, only the first four configurations were used for the experiments with long-distance proofs. To produce both ordinary and long-distance Q-resolution proofs, we used DepQBF 6.03 in a configuration that allowed tracing (i.e., with most of the advanced techniques off) with a cut-off time of 900 CPU seconds and a memory limit of 4GB. The validation process was limited to 1800 CPU seconds and 7GB of memory. The experiments were run on a cluster of heterogeneous machines running 64-bit Ubuntu 16.04.3 LTS (GNU/Linux 4.10.0-42). We evaluated the tools on the PCNF benchmark sets from the QBF Evaluations 2017, 2016, and 2010. The numbers of true and false validated instances for each configuration and benchmark set are reported in the tables below. The column "total" reports the total number of proofs for true and false formulas produced by DepQBF.

The results indicate that our approach is beneficial mainly on true formulas, but performs well across the board. Interestingly, even though QBFcert tends to produce smaller certificates than `qrp2rup`, Glucose performs worse on them. QBFcert internally uses AIG-based optimizations to shrink the certificates, and it is conceivable that these optimizations hurt Glucose's performance.

		QBFcert SAT solver		`qrp2rup` SAT solver		`qrp2rup` DRAT-trim	
year	total	Lingeling	Glucose	Lingeling	Glucose	deletion	plain
2010	162+230	88+215	88+216	88+225	92+**228**	**99**+224	**99**+223
2016	157+206	124+196	123+197	116+202	128+**203**	**136**+202	**136**+200
2017	18+62	**12**+58	**12**+58	11+62	**12**+**63**	**12**+**63**	**12**+62

Table 6.1: Ordinary Q-resolution proofs: number of true+false formulas validated.

		`qrp2rup`+SAT-solver		`qrp2rup`+DRAT-trim	
year	total	Lingeling	Glucose	deletion	plain
2010	149+222	93+215	95+**217**	**100**+215	**100**+215
2016	160+250	120+197	131+**200**	**137**+196	**137**+196
2017	17+59	12+**59**	**13**+**59**	**13**+**59**	**13**+**59**

Table 6.2: Long-distance Q-resolution proofs: number of true+false formulas validated.

6.6 Summary and Discussion

We have presented a way of using (long-distance) Q-resolution/Q-consensus proofs in the process of validating QBF certificates. Our approach does not require a SAT call and comes with a polynomial runtime guarantee. Since it allows us to generate proofs in a format that is routinely used to verify the answers produced by SAT solvers and that has prompted the development of formally verified checkers [Lam17, HHKW17, CFHH+17], we can have a high degree of confidence in the correctness of certificates validated in this manner.

However, one subtle challenge remains. When constructing the validation formula $\Phi[\mathcal{P}]$, we take the matrix of Φ and append a CNF encoding of the countermodel. In principle, if we instead appended a small unsatisfiable CNF formula such as $(x) \wedge (\overline{x})$, we could be led to believe that it represents a countermodel when in reality it is much more restrictive than a countermodel is allowed to be (a formula that does not encode a set of functions). It would be desirable to have a way of checking that what we appended to the original matrix is indeed a set of functions (with the correct dependencies) for universal variables. This may require formal verification of parts of the certificate extraction algorithm.

A potential limitation of our approach is that it is sensitive to certain aspects of the CNF encoding of the countermodel to be validated, and therefore does not necessarily work with certificates extracted by other tools. However, our method ought to be compatible with simple circuit-level simplifications of certificates. Moreover, we hope to improve performance by generating GRAT [Lam17] proofs of validation formulas as future work.

Publication Notes

The research in this chapter appeared in a paper published in the proceedings of the 21st International Conference on Theory and Applications of Satisfiability Testing (SAT 2018) [PSS18].

CHAPTER 7

Proof Complexity of Fragments of Long-distance Q-resolution

Proof complexity is an invaluable tool in the study of the limits (and strengths) of QBF solvers. Analyzing even simple versions of QCDCL can be very complicated, let alone when decision heuristics and other details enter the field. If, however, we can argue that a given solver searches for proofs in some proof system, and that a given formula only has exponentially long proofs in that proof system (a *lower bound*), then we obtain bulletproof evidence that the solver is going to perform poorly no matter what heuristics it uses.

We present lower bounds and general lower-bound techniques for several fragments of long-distance Q-resolution in this chapter. We also observe that a previously known upper bound can be rephrased in stronger terms in our context, yielding a separation of two important fragments of long-distance Q-resolution: Q-resolution, i.e., the fragment that has universal reduction but no merging; and *reductionless* long-distance Q-resolution, i.e., the fragment that has merging but no reduction.

The rules of reductionless long-distance Q-resolution are shown in Figure 7.1. The QBF solver GhostQ [KSGC10] uses reductionless long-distance Q-resolution for learning. Indeed, in a search-based solver that assigns variables in the order of the quantifier prefix, universal reduction is not required to derive a learned clause. One only needs to identify purely universal clauses, which are treated as if they were empty. Note, however, that while the reductions do not help in deriving a single learned clause, it is not known whether they cannot speed the solver up in the long run by learning shorter clauses. Reductionless long-distance Q-resolution was originally called Q^w*-resolution* [BJK15], but we will stick to the longer, somewhat more self-explanatory name.

We have already defined Q-resolution and long-distance Q-resolution in Chapter 2 in Figures 2.1 and 2.2. In this chapter, we will sometimes write u^* for the *merged literal*

$$\frac{}{C}\ \text{(input clause)}$$

Every input clause $C \in \varphi$ can be used as an axiom.

$$\frac{C_1 \vee e \qquad \neg e \vee C_2}{(C_1 \setminus \overline{C_2}) \vee (C_2 \setminus \overline{C_1}) \vee \{\, u^* : u \in var(C_1 \cap \overline{C_2}) \,\}}\ \text{(long-distance resolution)}$$

It is required that $e <_\Phi u$ for each variable $u \in var(C_1 \cap \overline{C_2})$ and that $C_1 \vee C_2$ does not contain an existential variable and its negation.

Figure 7.1: Derivation rules of reductionless Q-resolution for a PCNF formula $\Phi = \mathcal{Q}.\varphi$.

$u \vee \neg u$ created by long-distance resolution, and let $\overline{u^*} = u^*$ and $var(u^*) = u$.

As usual, we consider *derivations* in these proof systems which are sequences $C_1, \ldots, C_k$ of clauses such that each clause C_i is an axiom or derived from clauses appearing earlier in the sequence using one of the proof rules. The definition of a *refutation* is slightly adapted: it is a derivation of the empty clause; or, in the case of reductionless Q-resolution, a derivation of a purely universal clause. In the following sections, we will also sometimes write $C_1 \odot_x C_2$ to denote the resolvent of C_1 and C_2 on pivot x. If the pivot is understood we may simply write $C_1 \odot C_2$.

7.1 A Lower Bound for Reductionless Q-resolution

We generalize an exponential lower bound for *level-ordered* Q-resolution [JMS15] for the *completion principle* formulas CR_n. We have already used these formulas before, to prove certain properties of QCDCL with dependency learning in Chapter 4 (see Definition 12), but we repeat the definition below for convenience. A Q-resolution derivation is level-ordered if the order of pivot variables encountered on any path in the derivation follows the order in the quantifier prefix. A level-ordered Q-resolution refutation can be turned into a reductionless Q-resolution refutation simply by omitting the reduction steps.

Definition 19 ([JMS15]). *Let*

$$\mathrm{CR}_n = \exists_{1 \le i,j \le n} x_{ij}\ \forall z\ \exists_{i=1}^{n} a_i\ \exists_{j=1}^{n} b_j \quad \left(\bigwedge_{1 \le i,j \le n} A_{ij} \wedge B_{ij} \right) \wedge A \wedge B,$$

where

$$A_{ij} = x_{ij} \vee z \vee a_i, \qquad A = \overline{a_1} \vee \cdots \vee \overline{a_n},$$
$$B_{ij} = \overline{x_{ij}} \vee \overline{z} \vee b_j, \qquad B = \overline{b_1} \vee \cdots \vee \overline{b_n}.$$

We will prove the following result.

Theorem 14. *Any reductionless Q-resolution refutation of the formula* CR_n *has size at least* 2^n.

In the following we assume $n \geq 2$ (Theorem 14 obviously holds for $n = 1$), Π is a reductionless Q-resolution refutation of CR_n, C is a clause of Π, and $C_\bot$ the conclusion of Π (recall that a reductionless Q-resolution refutation ends in a purely universal clause). For the purposes of this subsection, we will consider a merged literal u^* as a shorthand for $u \vee \neg u$.

Claim 1. *For all* $1 \leq i \leq n$ *and* $1 \leq j \leq n$,

- *if* $a_i \in C$ *or* $x_{ij} \in C$ *then* $z \in C$,
- *if* $b_j \in C$ *or* $\overline{x_{ij}} \in C$ *then* $\overline{z} \in C$.

Proof. The statement holds for input clauses and universal literals are never removed from clauses. □

Claim 2. *For all* $1 \leq i \leq n$ *and* $1 \leq j \leq n$,

- *if* $\overline{a_i} \in C$ *then* $z \in C$ *or* $C = A$,
- *if* $\overline{b_j} \in C$ *then* $\overline{z} \in C$ *or* $C = B$.

Proof. The statement holds for input clauses. Let C be the resolvent of C_1 and C_2 and assume without loss of generality that $\overline{a_i} \in C_1$. By induction hypothesis either $z \in C_1$, in which case $z \in C$, or $C_1 = A$, in which case the resolution step is over some pivot a_j. That means $a_j \in C_2$, and by Claim 1 we have $z \in C_2$ and so $z \in C$. □

Claim 3.

- *For all* $1 \leq i \leq n$, *if* $\overline{z} \notin C \wedge \overline{a_i} \notin C \wedge C \neq B$, *then there is* j, *such that* $x_{ij} \in C$,
- *For all* $1 \leq j \leq n$, *if* $z \notin C \wedge \overline{b_j} \notin C \wedge C \neq A$, *then there is* i, *such that* $\overline{x_{ij}} \in C$.

Proof. By induction on the proof size. The statement clearly holds for input clauses. Let C be the resolvent of C_1 and C_2. Since $\overline{z} \notin C$, both $\overline{z} \notin C_1$ and $\overline{z} \notin C_2$. Since $\overline{a_i} \notin C$, either $\overline{a_i} \notin C_1$ or $\overline{a_i} \notin C_2$, assume the first. If $C_1 = B$, then the resolution step can only resolve away one of the literals, and so there is j such that $\overline{b_j} \in C$. By Claim 2 (since $C \neq B$) we have $\overline{z} \in C$, a contradiction. Therefore $C_1 \neq B$ and by induction hypothesis there is j such that $x_{ij} \in C_1$. Unless the resolution step is on x_{ij}, we have $x_{ij} \in C$, so it remains to prove that this is indeed not the case. If x_{ij} were the pivot, then $\overline{x_{ij}} \in C_2$, hence $\overline{z} \in C_2$, a contradiction. □

Claim 4. *The conclusion $C_\bot$ contains z^*.*

Proof. $C_\bot$ contains no existential literals and is distinct from A and B, so in order not to be in contradiction with Claim 3, the statement has to hold. □

Claim 5. *If $z, \overline{z} \in C$, then for all $1 \leq i, j \leq n$ we have $a_i, \overline{a_i}, b_j, \overline{b_j} \notin C$.*

Proof. Consider the last C that violates this implication. Clearly $C \neq C_\bot$. Therefore, there is C_0 and C_1 such that C_1 is the resolvent of C and C_0. Clearly $z, \overline{z} \in C_1$. Since C_1 no longer violates the condition, there is no literal right of z in C_1. Hence C_0 is neither A nor B. Therefore by Claim 1 and Claim 2, a literal on z is in C_0. Since C violates the implication, there is a literal right of z in C, but since there is none in C_1, the pivot variable must be right of z. That means the resolution step is illegal, a contradiction. □

Claim 6. *If C is the resolvent of C_1 and C_2 and $z, \overline{z} \in C$, then neither C_1 nor C_2 contains any of the literals $a_i, \overline{a_i}, b_j, \overline{b_j}$.*

Proof. If $z, \overline{z} \in C_1$ or $z, \overline{z} \in C_2$, then the statement follows from Claim 5. Otherwise, the resolution step merges z and $\overline{z}$. That means the pivot must be left of z, and so any of the literals $a_i, \overline{a_i}, b_j, \overline{b_j}$ would end up in C if contained in any of the premises, a contradiction with Claim 5. □

For the next claims, we need to introduce sets M and S as follows. We let

$$M = \{\, C \in \Pi : z^* \in C \,\}$$

be the set of clauses which contain a merged literal. We define S as the "boundary" of M, i.e., the set of clauses that do not contain a merged literal but have a direct descendant that does, formally

$$S = \{\, C \in \Pi : C \notin M \text{ and there are } C_0, C_1 \in \Pi \text{ s.t. } C_1 = C \odot C_0 \text{ and } C_1 \in M \,\}.$$

Claim 7. *If $C \in S$, then $a_i, \overline{a_i}, b_j, \overline{b_j} \notin C$.*

Proof. Follows from Claim 6 as $C \in S$ has a direct descendant with z^*. □

Claim 8. *If $C \in S$ then $|C \setminus \{z, \overline{z}\}| \geq n$.*

Proof. By Claim 7 and the fact that $C \notin M$ we get that all preconditions of one of the rows of Claim 3 are satisfied for C, which means that either for all i there is a j such that $x_{ij} \in C$, or for all j there is an i such that $\overline{x_{ij}} \in C$, in both cases a total of n distinct literals, none of which is a literal on z. □

of Theorem 14. Consider $\Pi' = S \cup M$. Clauses in M have direct ancestors only in M or S (any direct ancestor that is not in M is by definition in S). Since $C_\bot \in M$, Π' is a reductionless Q-resolution refutation of S. If we disregard literals on z, it is in fact a resolution refutation of the propositional formula S, which means that S is unsatisfiable. By Claim 8, every clause in S has at least n literals, and so it excludes at most 2^{n^2-n} of the assignments to the variables of S. Therefore, S must have at least 2^n clauses in order to exclude all assignments and be unsatisfiable. □

Corollary 5. *Reductionless Q-resolution does not p-simulate tree-like Q-resolution.*

Proof. Since CR_n have short proofs in tree-like Q-resolution [Jan16], the separation follows from Theorem 14. □

Remark 1. Theorem 14 has another interesting consequence. Since QCDCL with dependency learning can solve CR_n in polynomial time [PSS19a], Theorem 14 implies that in order to harness the full power of QCDCL with dependency learning, one has to perform universal reduction during clause learning. This is in contrast with "ordinary" QCDCL where universal reduction is not required to derive a learned clause [KSGC10].

7.2 Short Proofs of QParity in Reductionless Q-Resolution

In this section, we prove that the QParity formulas, which require exponentially long proofs in Q-resolution [BCJ15], have short proofs in reductionless Q-resolution. It has already been shown that these formulas have short proofs in long-distance Q-resolution [Che17, Theorem 9]. We simply observe that these proofs—which we reproduce below for the sake of completeness—are in fact reductionless.

Definition 20 ([BCJ15]). *Let* $\mathrm{QParity}_n = \exists x_1, \ldots, x_n \, \forall z \, \exists t_2, \ldots, t_n. \, \phi_n$, *where*

$$\phi_n = T_2^1 \wedge T_2^2 \wedge T_2^3 \wedge T_2^4 \wedge \left(\bigwedge_{i=3}^{n} T_i^1 \wedge T_i^2 \wedge T_i^3 \wedge T_i^4 \right) \wedge Z^1 \wedge Z^2, \text{ and}$$

$$\begin{aligned}
T_2^1 &= x_1 \vee x_2 \vee \overline{t_2}, & T_i^1 &= t_{i-1} \vee x_i \vee \overline{t_i}, & Z^1 &= t_n \vee \overline{z}, \\
T_2^2 &= x_1 \vee \overline{x_2} \vee t_2, & T_i^2 &= t_{i-1} \vee \overline{x_i} \vee t_i, & Z^2 &= \overline{t_n} \vee z. \\
T_2^3 &= \overline{x_1} \vee x_2 \vee t_2, & T_i^3 &= \overline{t_{i-1}} \vee x_i \vee t_i, & & \\
T_2^4 &= \overline{x_1} \vee \overline{x_2} \vee \overline{t_2}, & T_i^4 &= \overline{t_{i-1}} \vee \overline{x_i} \vee \overline{t_i}, & &
\end{aligned}$$

Theorem 15. *There is a reductionless Q-resolution refutation of* $\mathrm{QParity}_n$ *of size* $6n - 5$.

Proof. For $2 \leq i \leq n-1$ and $1 \leq j \leq 2$, we let $Z_i^1 = t_i \vee z^*$, $Z_i^2 = \overline{t_i} \vee z^*$, and $Z_n^j = Z^j$, and it is easy to verify that

$$Z_{i-1}^j = \left(T_i^{3j-2} \odot_{t_i} Z_i^1\right) \odot_{x_i} \left(T_i^{j+1} \odot_{t_i} Z_i^2\right).$$

Hence, we derive Z_2^1 and Z_2^2 in a total of $6(n-2)$ steps. Next, we have

$$(z^*) = \left(\left(T_2^1 \odot_{t_2} Z_2^1\right) \odot_{x_2} \left(T_2^2 \odot_{t_2} Z_2^2\right)\right) \odot_{x_1} \left(\left(T_2^4 \odot_{t_2} Z_2^1\right) \odot_{x_2} \left(T_2^3 \odot_{t_2} Z_2^2\right)\right),$$

and so the formula is refuted. The resolution steps on the x_i are sound, because $x_i < z$ for all $1 \leq i \leq n$. The total number of resolution steps is $6n - 5$. □

Corollary 6. *Q-resolution does not p-simulate reductionless Q-resolution.*

Remark 2. While strategies extracted from Q-resolution refutations (of PCNF formulas containing a single unviersal variable) correspond to bounded-depth circuits [BCJ15], Theorem 15 implies that reductionless Q-resolution proofs cannot be efficiently transformed into bounded-depth circuits.

7.3 Lower Bounds from Strategy Extraction

In this section, we will show how to extend the scope of lower bound techniques based on strategy extraction [BCJ15] to fragments of long-distance Q-resolution. We begin by observing that strategies extracted from reductionless Q-resolution proofs correspond to branching programs [BBM19].

We briefly review the definition of a branching program and refer to the book by Wegener for more details [Weg00]. A *branching program* or *binary decision diagram (BDD)* on a set X of variables is a directed acyclic graph with a unique source node and at most two sink nodes. Each node v is labelled with a variable $\lambda(v) \in X$, except for the sinks, which are labelled with 0 or 1. If there are two sink nodes, their labels must be distinct. Moreover, every node has exactly two outgoing edges labelled with 0 and 1, respectively. A path $v_1, \ldots, v_n$ from the source of a branching program to its sink is *consistent* if the label of edge (v_i, v_{i+1}) agrees with the label of edge (v_j, v_{j+1}) whenever v_i and v_j are labelled with the same variable. A consistent path corresponds to an assignment in the obvious way. A branching program B on X computes a Boolean function $f(B)$ in the following way. Let $\tau : X \to \{0,1\}$ be an assignment. We follow the (consistent) path induced by τ to a sink node v, and set $f(B)(\tau) = \lambda(v)$. The *size* of a branching program is the number of nodes in it.

Let $\pi = C_1, \ldots, C_k$ be a reductionless Q-resolution derivation from a PCNF formula Φ. For each universal variable $u \in var_\forall(\Phi)$, we construct a branching program $B_\Phi^u(\pi)$ in the following way [BJK15, BBM19]. We first introduce two nodes v_0 and v_1 with $\lambda(v_0) = 0$ and $\lambda(v_1) = 1$. We now consider the clauses C_i in the order of their derivation and associate a node v_i with each one. Depending on how clause C_i was derived, we distinguish two cases:

1. If C_i is an input clause, we let

$$v_i = \begin{cases} v_0, & \text{if } u \in C_i; \\ v_1, & \text{otherwise.} \end{cases}$$

2. If C_i is the resolvent of clauses $C_j = C'_j \vee e$ and $C_l = \neg e \vee C'_l$, there are two possibilities depending on the order of variables e and u in the prefix:

   - If $e < u$, we introduce a fresh node v_i to $B^u_\Phi(\pi)$ and label it $\lambda(v) = e$. Moreover, we add a 0-labelled edge from v_i to v_j and a 1-labelled edge from v_i to v_l.
   - Otherwise, $u < e$ and we cannot have $u \in var(C_j \cap \overline{C_l})$ by the rules of reductionless Q-resolution (see Figure 7.1). If $u \in var(C_j)$, we let $v_i = v_j$. Otherwise, we let $v_i = v_l$.

Finally, we remove all nodes that cannot be reached from v_k. The following statement is immediate from the construction.

Lemma 18. *If $\pi = C_1, \dots, C_k$ is a reductionless Q-resolution derivation from a PCNF formula Φ and $u \in var_\forall(\Phi)$ is a universal variable, then $B^u_\Phi(\pi)$ is a branching program on D^u_Φ of size at most k.*

Moreover, if the derivation π is a refutation, these branching programs compute a universal winning strategy. To show this, we first prove the following statement.

Lemma 19. *Let $\pi = C_1, \dots, C_k$ be a reductionless Q-resolution derivation from a PCNF formula Φ. Let $\tau : var_\exists(\Phi) \to \{0,1\}$ be an assignment that does not satisfy C_k, and let*

$$\sigma^\pi_\Phi = \{\, u \mapsto f(B^u_\Phi(\pi))(\tau|_{D^u_\Phi}) : u \in var_\forall(\Phi) \,\}$$

be the assignment computed by the branching programs $B^u_\Phi(\pi)$ in response. Then $C_i[\sigma^\pi_\Phi \cup \tau] = 0$ for some input clause $C_i \in \pi$.

Proof. We proceed by induction on the size k of the derivation. If $\pi = C_1$ then C_1 must be an input clause, and $B^u_\Phi(\pi)$ consists of a single node labelled 0 if $u \in C_1$ and labelled 1 if $\neg u \in C_1$. Accordingly, the function $f(B^u_\Phi(\pi))$ constantly returns an assignment that falsifies any universal literal on variable u. This proves the base case.

Suppose the statement of the lemma holds for derivations of size up to $k-1$. If C_k is an input clause, the same reasoning as in the base case applies, so suppose C_k is derived by resolution from clauses $C_i = C'_i \vee e$ and $C_j = \neg e \vee C'_j$ with $1 \le i, j < k$. Let $\pi_i = C_1, \dots, C_i$ and let $\pi_j = C_1, \dots, C_j$ be the derivations of the corresponding clauses. We claim that $\sigma^\pi_\Phi(u) = \sigma^{\pi_i}_\Phi(u)$ for each universal variable $u \in var(C_i)$ if $\tau(e) = 0$, and $\sigma^\pi_\Phi(u) = \sigma^{\pi_j}_\Phi(u)$ for each universal variable $u \in var(C_j)$ if $\tau(e) = 1$.

Choose a universal variable u and let $\tau' = \tau|_{D^u_\Phi}$ be the corresponding restriction of τ. We consider two cases. If $e < u$ it is not difficult to see that

$$f(B^u_\Phi(\pi))(\tau') = \begin{cases} f(B^u_\Phi(\pi_i))(\tau') \text{ if } \tau(e) = 0, \text{ and} \\ f(B^u_\Phi(\pi_j))(\tau') \text{ otherwise.} \end{cases}$$

Accordingly, we have $\sigma^\pi_\Phi(u) = \sigma^{\pi_i}_\Phi(u)$ if $\tau(e) = 0$ and otherwise $\sigma^\pi_\Phi(u) = \sigma^{\pi_j}_\Phi(u)$. On the other hand, if $u < e$ by construction of the branching program we have

$$f(B^u_\Phi(\pi)) = \begin{cases} f(B^u_\Phi(\pi_i)) \text{ if } u \in var(C_i), \text{ and} \\ f(B^u_\Phi(\pi_j)) \text{ otherwise.} \end{cases}$$

If $u \in var(C_i)$ then $\sigma^\pi_\Phi(u) = \sigma^{\pi_i}_\Phi(u)$. If $u \in var(C_j)$ as well then we must have $u \in C_i \cap C_j$ or $\neg u \in C_i \cap C_j$ since $u \notin var(C_i \cap \overline{C_j})$ by definition of reductionless Q-resolution. It follows that $f(B^u_\Phi(\pi_i))$ and $f(B^u_\Phi(\pi_j))$ compute the same constant function. Otherwise, if $u \notin var(C_i)$ then $\sigma^\pi_\Phi(u) = \sigma^{\pi_j}_\Phi(u)$. This proves the claim.

If $\tau(e) = 0$ then by induction hypothesis $C_l[\sigma^{\pi_i}_\Phi \cup \tau] = 0$ for an input clause $C_l \in \pi_i$. We can assume without loss of generality that π_i does not contain any universal variable besides those in the clause C_i. It follows from the claim that the assignment $\sigma^\pi_\Phi \cup \tau$ falsifies C_l as well. The case $\tau(e) = 1$ is symmetric. □

Lemma 20. *Let $\pi = C_1, \ldots, C_k$ be a reductionless Q-resolution refutation of a PCNF formula Φ. The set $\{ f(B^u_\Phi(\pi)) : u \in var_\forall(\Phi) \}$ is a universal winning strategy.*

Proof. Because π is a refutation the clause C_k must not contain existential variables. Thus every assignment $\tau : var_\exists(\Phi) \to \{0,1\}$ is an assignment that does not satisfy C_k, and by Lemma 19 the universal response σ^π_Φ (defined as in the statement of that lemma), in conjunction with the assignment τ, must falsify an input clause. □

These results allow us to translate lower bounds for branching programs to lower bounds on the size of reductionless Q-resolution refutations. Let $f : X \to \{0,1\}$ be a Boolean function, let $\varphi(X)$ be a Boolean circuit encoding f, and let u be a variable not occurring in φ. Using Tseitin transformation [Tse68], we can construct a CNF formula $\psi(X, u, Y)$ such that $\exists Y.\psi(X, u, Y)$ is logically equivalent to $\varphi(X) \neq u$. The PCNF formula $\Phi = \exists X \forall u \exists Y.\psi(X, u, Y)$ is a false PCNF formula with f as a unique universal winning strategy (cf. the lower bounds from strategy extraction for Q-resolution [BCJ15]). We call such a formula Φ a *PCNF encoding of f*.

Proposition 2. *Let Φ be a PCNF encoding of a Boolean function f such that any branching program computing f has size at least m. Then any reductionless Q-resolution refutation of Φ requires at least m clauses.*

Proof. Since f is the unique universal winning strategy for Φ, the statement follows immediately from Lemma 18 and Lemma 20. □

To the best of our knowledge, the only lower bounds on the size of general branching programs for explicitly defined Boolean functions currently known are polynomial [Nec66]. Accordingly, Proposition 2 does not yield strong lower bounds for reductionless Q-resolution. However, we can lift lower bounds for restricted classes of branching programs to lower bounds on the proof size in restricted versions of reductionless Q-resolution.

7.3.1 Regular Reductionless Q-Resolution

Every reductionless Q-resolution derivation $\pi = C_1, \ldots, C_k$ can be represented by a directed acyclic graph $G(\pi)$ on vertices $v_1, \ldots, v_k$ where v_i is labelled with C_i and there is an edge from v_j to v_i if $i < j$ and C_i is one of the clauses C_j was derived from (that is, edges are oriented from conclusions to premises). Each edge is labelled with the corresponding pivot variable.

A reductionless Q-resolution refutation $\pi = C_1, \ldots, C_k$ is *regular* if each variable occurs at most once as a label on any directed path starting from the vertex labelled with clause C_k. Each strategy function computed by such a proof corresponds to a so-called *read-once branching programs* or *free binary decision diagram (FBDD)*. A read-once branching program is a branching program where each variable is encountered at most once on any path from the source to a sink [Weg00].

Lemma 21. *Let $\pi = C_1, \ldots, C_k$ be a regular reductionless Q-resolution refutation of a PCNF formula Φ. Then $B^u_\Phi(\pi)$ is a read-once branching program of size at most k for each universal variable $u \in var_\forall(\Phi)$.*

Proof. Consider $B^u_\Phi(\pi)$ for any universal variable $u \in var_\forall(\Phi)$. By construction, the sequence of variables encountered on any path starting from the source of $B^u_\Phi(\pi)$ is a subsequence of the pivot variables seen as edge labels on any path starting from the source of $G(\pi)$. In particular, every variable occurs at most once. Since $B^u_\Phi(\pi)$ is a branching program of size at most k by Lemma 18, it is in fact a read-once branching program of size at most k. □

The *FBDD size* of a Boolean function f is the size of the smallest read-once branching program representing f. We can transfer lower bounds on the FBDD size of Boolean functions into lower bounds on the regular reductionless Q-resolution proof size of certain PCNF formulas, as stated in the next result.

Proposition 3. *Let Φ be a PCNF encoding of a Boolean function f with FBDD size m. Any regular reductionless Q-resolution refutation of Φ has size at least m.*

Proof. The statement follows from Lemma 21 and Lemma 20. □

Unlike in the case of general branching programs, strong lower bounds on the FBDD size of many explicitly defined Boolean functions are known [Weg00]. For instance, we can use the following result due to Bollig and Wegener [BW98].

Theorem 16 ([BW98]). *There is a Boolean function g in n variables that can be computed by a Boolean circuit of size $O(n^{3/2})$ but has FBDD size $\Omega(2^{\sqrt{n}})$.*

Corollary 7. *There is a Boolean function g in n variables with a PCNF encoding Φ of size polynomial in n such that any regular reductionless Q-resolution refutation of Φ has size $\Omega(2^{\sqrt{n}})$.*

7.3.2 Tree-like Long-Distance Q-Resolution

In this subsection, we are going to prove lower bounds on the size of *tree-like* long-distance Q-resolution. As in the case of reductionless Q-resolution, a long-distance Q-resolution derivation $\pi = C_1, \ldots, C_k$ can be represented by a labelled DAG $G(\pi)$. A derivation π is called *tree-like* if the DAG $G(\pi)$ is a tree.

We want to show that every tree-like long-distance Q-resolution refutation of a PCNF encoding of a Boolean function f can be efficiently turned into a bounded-depth circuit computing f. First, we generalize the construction of the branching programs B^u_Φ described at the beginning of this section to long-distance Q-resolution derivations. Let $\pi = C_1, \ldots, C_k$ be a long-distance Q-resolution derivation from a PCNF formula Φ. For each universal variable $u \in var_\forall(\Phi)$, we construct a labelled DAG $B^u_\Phi(\pi)$ in the same way as for a reductionless Q-resolution derivation, except for the following modification: if clause C_i is derived from a clause C_j by universal reduction and $u \in var(C_j) \setminus var(C_i)$, we set $v_i = v_0$, where $\lambda(v_0) = 0$. It is readily verified that we obtain a branching program of size at most k, as stated in the following lemma.

Lemma 22. *If $\pi = C_1, \ldots, C_k$ is a long-distance Q-resolution derivation from a PCNF formula Φ and $u \in var_\forall(\Phi)$ is a universal variable, then $B^u_\Phi(\pi)$ is a branching program on D^u_Φ of size at most k.*

A universal winning strategy can be computed from a long-distance Q-resolution refutation as follows [BJJW15]. We maintain a kind of *decision list* [Riv87] for each universal variable that is intended to encode a strategy function. Specifically, we consider sequences $L = (\varphi_1 \to \psi_1), \ldots, (\varphi_k \to \psi_k)$ where each of the φ_i and ψ_i are propositional formulas. Such a list, which we call a *generalized decision list*, represents a Boolean function f_L in the following way. Consider an assignment $\tau : \bigcup_{i=1}^k var(\varphi_i) \cup var(\psi_i) \to \{0,1\}$ to all the variables appearing in formulas on the list. If there is no index i with $1 \leq i \leq k$ such that τ satisfies φ_i, we define $f_L(\tau) = 1$. Otherwise, let i be the smallest index such that τ satisfies φ_i. Then $f_L(\tau) = \psi_i[\tau]$. The *size* of a decision list $L = (\varphi_1 \to \psi_1), \ldots, (\varphi_k \to \psi_k)$ is $|L| = \sum_{i=1}^k (|\varphi_i| + |\psi_i|)$.

Given a long-distance Q-resolution refutation $\pi = C_1, \ldots, C_k$ of a PCNF formula Φ, we construct a family $\mathcal{L}_\Phi(\pi) = \{ L_u : u \in var_\forall(\Phi) \}$ of generalized decision lists representing a universal winning strategy for Φ. For $Q \in \{\exists, \forall\}$, let $C_i^Q = \{ \ell \in C_i : var(\ell) \in var_Q(\Phi) \}$ denote the restriction of C_i to existential or universal literals. Moreover, for a Boolean function f, let $\phi(f)$ denote an encoding of f as a propositional formula. We consider

applications of universal reduction in the same order as they appear in the proof. Let C_i be a clause derived by universal reduction from a clause C_j, and let $u \in var(C_j) \setminus var(C_i)$. Let $\pi_i = C_1, \ldots, C_i$ and $\pi_j = C_1, \ldots, C_j$ denote the subderivations ending in clauses C_i and C_j, respectively. We add an entry

$$\left(\overline{C_i^{\exists}} \wedge \bigwedge_{v \in var(C_i^{\forall})} v \leftrightarrow \phi(f(B_\Phi^v(\pi_i))) \right) \rightarrow \phi(f(B_\Phi^u(\pi_j))) \tag{7.1}$$

at the end of the decision list L_u. Observing that the functions $f(B_\Phi^u(\pi_j))$ correspond to the (negated) *phase functions* introduced for the purpose of efficiently extracting universal winning strategies from long-distance Q-resolution refutations [BJJW15], it can be verified that the strategy functions computed by the corresponding algorithm coincide with the functions computed by the decision lists defined according to (7.1).

Proposition 4 ([BJJW15])**.** *Let π be a long-distance Q-resolution refutation of a PCNF formula Φ. The set $\{ f_{L_u} : L_u \in \mathcal{L}_\Phi(\pi) \}$ is a universal winning strategy.*

We now argue that this winning strategy can be represented by a bounded-depth circuit for certain proofs in *tree-like* long-distance Q-resolution. Specifically, we will show that this is the case for every tree-like refutation of a PCNF encoding of a Boolean function. We first observe that the branching programs B_Φ^u for tree-like proofs are decision trees. A *decision tree* is a branching program that can be turned into a tree by deleting the sink nodes.

Lemma 23. *If $\pi = C_1, \ldots, C_k$ is a tree-like long-distance Q-resolution derivation from a PCNF formula Φ, then $B_\Phi^u(\pi)$ is a decision tree for each universal variable $u \in var_\forall(\Phi)$.*

Proof. It is not difficult to see that after deleting the sink nodes labelled with 0 and 1 from $B_\Phi^u(\pi)$, the corresponding DAG can be obtained from $G(\pi)$ by deleting vertices and edges as well as contracting induced paths. Since $G(\pi)$ is a tree, the result is also a tree. □

Every decision tree can be efficiently translated into a CNF formula by taking the conjunction over the negations of its consistent paths [Riv87]. Moreover, a generalized decision list $L = (\varphi_1 \rightarrow \psi_1), \ldots, (\varphi_k \rightarrow \psi_k)$ can be represented by a circuit

$$\phi(L) = \bigvee_{i=1}^{k} \left(\left(\bigwedge_{j=1}^{i-1} \neg\varphi_j \wedge \varphi_i \right) \rightarrow \psi_i \right). \tag{7.2}$$

Lemma 24. *Let $L = (\varphi_1 \rightarrow \psi_1), \ldots, (\varphi_k \rightarrow \psi_k)$ be a generalized decision list such that d is the maximum depth of any formula φ_i and ψ_i, for $1 \leq i \leq k$. Then $\phi(L)$ is equivalent to f_L. Moreover, $\phi(L)$ has depth at most $d + 4$ and $|\phi(L)| = O(|L|^2)$.*

Let Φ be the PCNF encoding of a Boolean function f and consider a tree-like long-distance Q-resolution refutation π of Φ. Because Φ contains only a single universal variable, each entry in a decision list of $\mathcal{L}_\Phi(\pi)$ given by (7.1) simplifies to $\overline{C} \to \phi(f(B^u_\Phi(\pi_j)))$, and the right hand side of this implication can be efficiently transformed into a CNF because B^u_Φ is a decision tree by Lemma 23. We thus obtain the following result.

Proposition 5. *There is a polynomial $p(\cdot)$ and a constant d such that, for any tree-like long-distance Q-resolution refutation π of the PCNF encoding of a function f, there exists a Boolean circuit of size at most $p(|\pi|)$ and depth at most d computing f.*

Proof. By Lemma 22 and Lemma 23, each labelled DAG $B^u_\Phi(\pi')$ is a decision tree of size at most $|\pi|$ for the universal variable u of Φ and each subproof π' of π. Each such decision tree can be efficiently encoded as a CNF formula and the decision list has no more than $|\pi|$ entries of size polynomial in $|\pi|$, so it follows from Lemma 24 that there is a polynomial $p(\cdot)$ such that $\phi(L_u)$ has size at most $p(|\pi|)$ and depth at most 6. Finally, $\{f_{L_u}\}$ is a universal winning strategy for Φ by Proposition 4, so f_{L_u} must coincide with f. □

Since any bounded-depth circuit computing the n-bit parity function has size exponential in n [Hås87], Proposition 5 allows us to obtain the following exponential lower bound on the size of refutations of QParity in tree-like long-distance Q-resolution.

Theorem 17. *Any refutation of* QParity_n *in tree-like long-distance Q-resolution requires size exponential in n.*

7.4 Summary and Discussion

We studied the proof complexity of fragments of long-distance Q-resolution. We proved that reductionless Q-resolution cannot p-simulate even tree-like Q-resolution. Since reductionless Q-resolution can be used to derive learned clauses in QCDCL solvers [KSGC10], this is another indication that QCDCL[1] proofs correspond to a fairly weak fragment of (long-distance) Q-resolution [Jan16]. The QParity formulas, on the other hand, have short refutations in reductionless Q-resolution. These formulas require Q-resolution refutations of exponential size [BCJ15], so Q-resolution and reductionless Q-resolution turn out to be incomparable.

The existence of short proofs of QParity also marks the breakdown of an elegant technique for obtaining lower bounds on the size of Q-resolution refutations through strategy extraction [BCJ15]. Evidently, strategies corresponding to reductionless Q-resolution proofs do not correspond to bounded-depth circuits.

We proved that arguments based on strategy extraction can nevertheless be used to obtain lower bounds for restricted versions of long-distance Q-resolution. Specifically, we

[1] At least without techniques like dependency learning [PSS19a].

showed that *regular* reductionless Q-resolution proofs correspond to *read-once branching programs*, and that *tree-like* long-distance Q-resolution proofs correspond to *bounded-depth* circuits, allowing us to transfer known lower bounds.

Obtaining a characterization of the strategies corresponding to (even reductionless) long-distance Q-resolution refutations that could be used in obtaining lower bounds remains as an intriguing open problem.

Publication Notes

The research in this chapter appeared in a paper published in the proceedings of the 22nd International Conference on Theory and Applications of Satisfiability Testing (SAT 2019) [PSS19d].

CHAPTER 8

Portfolio-based Algorithm Selection for Circuit QBFs

This chapter documents our work on the construction of algorithm selectors for a portfolio solvers for circuit QBFs. An *algorithm selector* is a procedure which, given a vector of *features* of an input instance, picks that solver from a *portfolio* which it estimates will take the shortest time to solve the instance. Here, a portfolio is simply a set of solvers, and the features are real numbers easily computable from a formula. An algorithm selector typically learns from data of instance features and solver runs using machine learning.

The contributions of this work are the following:

- we define novel features for formulas in the QCIR input format;
- we show that these features lead to good algorithm selectors;
- and we identify a set of only two features which suffices to predict solver performance accurately, and leads to closing almost the entire performance gap to the virtual best solver.

Our portfolio comprises the QBF solvers that participated in the prenex non-CNF track of the 2017 QBF Evaluation[1] (with the exception of CQesto, which was not publicly available at the time; for all solvers, the default configurations provided by their authors were used). Their performance on the corresponding benchmark set was fairly similar, with the number of solved instances ranging from 89 (GhostQ) to 117 (QFun) out of a total 320.

[1] See `http://www.qbflib.org`.

1. QuAbS [Ten16] generalizes the concept of "clause selection" (as implemented in Qesto [JM15] and CAQE [RT15]) from clauses to subformulas. An abstraction is maintained for each quantifier block, and so-called interface literals communicate whether a subformula is satisfied or falsified at a lower (or higher) level.
2. QFun [Jan18b] generalizes counterexample-guided abstraction refinement (CEGAR) solving [JKMSC12] to circuit QBFs and uses decision tree learning to "guess" counter(models) based on recent truth assignments.
3. Qute [PSS17] is a search-based solver that implements a technique called dependency learning to ignore artificial syntactic dependencies induced by nested quantifiers.
4. GhostQ [KSGC10] is a search-based solver that utilizes so-called ghost literals for dual propagation.

AutoFolio is an algorithm selector that alleviates the burden of manually choosing the right machine learning model for a problem domain and hand-tuning hyperparameters by using algorithm configuration tools to automatically make design choices and find hyperparameter settings that work well for a particular scenario [LHHS15].

AutoFolio allows us to construct a portfolio from the above solvers with little effort. In particular, it quickly lets us create portfolios that are tuned to particular subsets of features (see Section 8.3). Our main design choice consists in defining the set of features described in the next section.

8.1 QCIR Instance Features

We consider circuit Quantified Boolean Formulas (QBFs) in prenex normal form encoded according to the "cleansed" QCIR standard [JKS16]. Each such formula is a pair $\Phi = \mathcal{Q}.\varphi$ consisting of a *quantifier prefix* $\mathcal{Q}$ and a Boolean circuit φ called the *matrix* of Φ. The quantifier prefix $\mathcal{Q}$ is a sequence $Q_1X_1 \dots Q_kX_k$ where each $Q_i \in \{\forall, \exists\}$ is a *quantifier* for $1 \leq i \leq k$ such that $Q_i \neq Q_{i+1}$ for $1 \leq i < k$, and the X_i are pairwise disjoint sets of variables called *quantifier blocks*.

The matrix φ is a Boolean circuit encoded as a sequence of gate definitions of the form

$$g = \circ(l_1, \dots, l_r)$$

where $\circ \in \{\wedge, \vee\}$, each *gate literal* l_i is either an unnegated gate variable g' (a *positive* gate literal) or a negated gate variable $\neg g'$ (a *negative* gate literal), and g' is a previously defined gate or an input gate $g' \in \bigcup_{i=1}^{k} X_i$. We refer to r as the *size* of gate g. The *depth* of a gate g is 0 if g is an input gate, and otherwise the maximum depth of a gate occurring in the definition of g plus one. A unique gate literal is identified as the output of the circuit φ.

We consider the following *static features* of QCIR instances:

1. The number n_e of existential variables.
2. The number n_u of universal variables.
3. The balance $n_e/n_u + n_u/n_e$ of existential and universal variables.
4. The number k of quantifier blocks.
5. The minimum size min_{b} of a quantifier block.
6. The maximum size max_{b} of a quantifier block.
7. The average size μ_{b} of a quantifier block.
8. The standard deviation σ_{b} of the quantifier block size.
9. The relative standard deviation $\sigma_{\mathrm{b}}/\mu_{\mathrm{b}}$ of the quantifier block size.
10. The total number pos of positive gate literals.
11. The total number neg of negative gate literals.
12. The balance $pos/neg + neg/pos$ of positive and negative gate literals.
13. The number $n_\wedge$ of AND gates.
14. The number $n_\vee$ or OR gates.
15. The maximum gate size max_{gs}.
16. The average gate size μ_{gs}.
17. The standard deviation σ_{gs} of the gate size.
18. The relative standard deviation $\sigma_{\mathrm{gs}}/\mu_{\mathrm{gs}}$ of the gate size.
19. The maximum gate depth max_d.
20. The average gate depth μ_d.
21. The standard deviation σ_d of the gate depth.
22. The relative standard deviation σ_d/μ_d of the gate depth.
23. The number n_p of gates all of whose gate literals have the same polarity (all positive or all negative).

Features that only depend on the quantifier prefix can be computed just as well for PCNF instances, and indeed some of the features 1–9 were already used in constructing the portfolio solver AQME [PT09]. The main difference between PCNF and QCIR is in the representation of the matrix and accordingly, this is where new features are required. Some of the above features (such as the numbers of AND/OR gates) can be seen as generalizations of PCNF features (number of clauses). Others, such as the maximum gate depth, only make sense for circuits.

In addition to these static features, we use several *probing features* computed by a short run of Qute (probing features are crucial for the performance of portfolios for SAT [XHHL08]):

1. The number of learned clauses.
2. The number of learned tautological clauses.
3. The number of learned terms.
4. The number of learned contradictory terms.
5. The fraction of variable assignments made by branching (the remaining assignments are due to propagation).
6. The total number of backtracks.
7. The number of backtracks due to dependency learning (a feature of Qute).
8. The number of learned dependencies as a fraction of the trivial dependencies.

8.2 Per-instance Algorithm Selection for QCIR

The experiments were conducted on a cluster where each node is equipped with 2 Intel Xeon E5-2640 v4 processors (25M Cache, 2.40 GHz) and 160GB of RAM. The machines are running 64-bit Ubuntu in version 16.04.3.

We work with the set of QCIR benchmark instances from the 2016 and 2017 QBF evaluations solved by at least one of the above solvers within 900 seconds of CPU time and 4GB of memory usage, a total of 731 instances. Figure 8.1 illustrates that there is a lot of complementarity between the component solvers. We split the 731 instances into a training set of 549 instances and a test set of 182 instances, uniformly at random. On the training set we fixed a cross-validation split into 10 folds of the same size. When we report performance of a selector on the *training* set, we in fact report cross-validation performance on this fixed split. This means that the selector was trained once on each subset of 9 folds and evaluated on the 10th one, and the results were combined. On the other hand, when we report performance on the test set, the respective selector is trained on the *entire* training set, disregarding the CV-split, and then evaluated on the

solver	Training set (549) PAR10	#solved	%closed	Test set (182) PAR10	#solved	%closed
GhostQ	2228.92	414	—	2492.61	132	—
Qfun	1922.07	433	—	2384.68	134	—
QuAbS	1641.90	450	—	1747.40	147	0%
Qute	1458.09	461	0%	1845.48	145	—
PFA	71.93	546	96.35%	171.03	179	91.01%
PF2	57.58	547	97.35%	217.16	178	88.34%
PF3	55.78	547	97.47%	165.97	179	91.30%
PFS	55.65	547	97.48%	167.53	179	91.21%
VBS	19.46	549	100%	15.35	182	100%

Table 8.1: Performance of component solvers and selectors on the training and test sets in terms of penalized average runtime (PAR10), the number of solved instances, and for selectors the extent to which they match the virtual best solver (VBS) measured as the percentage of the PAR10 gap between the single best solver (SBS) and the VBS that is closed by the selector. Training performance of selectors is CV-performance. Selectors were configured using AutoFolio in self-tuning mode for each of the feature subsets reported. PF2 is the selector configured for the best subset of 2 features, similarly PF3, PFS uses static features only, and PFA uses all features.

entire test set. The reason why we use this setup for our evaluation is the following. The standard way to evaluate the performance of AutoFolio is by using cross-validation. However, if AutoFolio is tuned to the specific CV-split, the CV performance may be an overly optimistic estimate of how well the model will generalize. Even though cross validation should still protect us from overfitting, we decided to hold out a test set even on top of that, in order to perform a sanity check of the experiment afterwards.

Each of the selectors PF* mentioned in Table 8.1 was trained using AutoFolio in self-tuning mode, with a budget of 42 000 wall-clock seconds and a bound of 50 000 runs for the algorithm configuration tool SMAC, and with a specific subset of features (see the next section and caption of Table 8.1 for details). For the SMAC-configuration phase we used the CV-split as mentioned earlier. The selectors PFA, PFS, and PF3 use an XGBoost classifier, while PF2 uses a random-forest regressor.

8.3 Which Features Matter?

It is common wisdom that high-performance per-instance algorithm selectors should have access to a large and rich set of features (see, e.g., [XHHL08]). While earlier selector designs based on ridge regression required feature selection to work well, state-of-the-art per-instance selectors make use of sophisticated machine learning techniques, such as

random forests, that are less sensitive to uninformative or correlated features. However, defining and computing features requires substantial domain expertise and often involves significant amount of work, especially since feature computation must be efficient in order to achieve good selector performance. Furthermore, selectors based on large sets of complex features can be far more difficult to understand than ones based on few and simple features. Since our full feature set for QCIR formulae, as described previously, gave rise to excellent selector performance, we decided to investigate whether similarly good performance could be obtained with fewer features.

We first trained a selector using only our static features, using AutoFolio, as described in the previous section. The resulting selector, denoted PFS in Table 8.1, performed slightly better than the selector trained using the full set of static and probing features (PFA). This was a great surprise to us in light of previous work on algorithm selection in which probing features were found to be helpful (see, e.g., [Kot16]). Since our full selector is already very close in performance to the VBS, it cannot be the case that we simply failed to come up with the right probing features, but rather that in the scenario we consider, static features are sufficient. Prompted by this finding, we decided to investigate the effect of further reducing our static features set.

In order to test what feature subsets might work well, we used the following setup. We configured AutoFolio using the static features, and we saved the resulting configuration of hyperparameters. Then, with this configuration of AutoFolio, we performed forward and backward selection on the set of static features. In forward selection, we started with the empty set of features, and at each step added a single feature, while in backward selection we started with the full set of static features, and at each step removed a feature. In both cases, the feature to be added/removed was chosen so that the resulting portfolio would have maximum performance. It is important to note here that we *did not* configure AutoFolio for each of the subsets searched in this process—instead we used the configuration that we computed as described at the beginning of this paragraph. The reason for that was to avoid the huge computational cost of configuring AutoFolio over and over again. In retrospect, this was indeed justified, as we obtained well-performing selectors for the feature subsets even this way, and we saved months of CPU time. However, note that once we found promising subsets of features by forward/backward selection, we configured AutoFolio for these subsets again, and the results of those specifically configured selectors are reported in Table 8.1.

Figure 8.2 shows the performance curve along forward/backward selection. The values of PAR10 and the number of solved instances were obtained by performing cross validation on the fixed CV-split mentioned earlier. In particular, we can see that forward selection achieves very good performance with two or three features already. The first three features picked by forward selection are *circuit depth*, *number of quantifier blocks*, and *average block size*. Since so few features turned out to yield such good selectors, we performed a brute-force search of all subsets of size 2 or 3 (again, evaluating performance with the fixed AutoFolio configuration used for forward/backward selection). This search confirmed that both the size-2 and size-3 subsets found by forward selection

were almost optimal (equal number of solved instances as with the optimal set, PAR10 within 1%). This search confirmed that the size-2 subset found by forward selection was almost optimal (second best, equal number of solved instances as with the optimal set, difference of 1.2 in PAR10), and the size-3 subset was optimal. We decided to continue the experiment with the size-2 subset found by forward selection (instead of the "optimal" one), for two reasons. Firstly, it contains the feature *circuit depth*, which is the best single predictor, but which is replaced in the optimal subset by *relative standard deviation of gate depths*, a feature that is somewhat harder to interpret. Secondly, we need to keep in mind, that not even this exhaustive search was perfect, as we did not (and could not) configure AutoFolio for each subset searched. Therefore, its results only served as a sanity check, to make sure that forward selection did not miss some great feature set, which turned out not to be the case. Hence, we went on to configure AutoFolio for the subsets {*circuit depth*, *number of quantifier blocks*}, and {*circuit depth*, *number of quantifier blocks*, *average block size*}, the results of which are shown in Table 8.1 (entries PF2 and PF3). As Table 8.1 shows, PF2 achieves virtually the same good performance as PFS, and closes almost all of the gap between SBS and VBS. This holds whether we look at the CV-evaluation on the training set, or the additional evaluation on the test set.

As a final sanity check, we evaluated the performance of selectors trained using these small sets of features on the same set of instances, but using only 3 out the 4 participating solvers (for each subset of 3 solvers). We set this experiment up in the following way: for each subset of features corresponding to one of the selectors PF*, we saved the configuration of AutoFolio that was optimized for the particular subset of features using all four solvers. We then evaluated the performance of selectors built using the saved configurations for each of the 4 size-3 solver subsets (a total of 16 selectors), in the same way as we did for Table 8.1. In order to get the theoretically best AutoFolio performance, we would have had to reconfigure AutoFolio for every pair of (solver subset, feature subset), but as before we simplified things to save computational resources. This experiment confirmed that even for different solver sets, the features *circuit depth*, and *number of quantifier blocks* are fairly robust predictors of solver performance. However, naturally, features must be tied to solvers whose performance they predict, so we cannot expect that a fixed set of features will be a universal predictor for all solver sets.

In a sense, these results are not surprising, as one would expect from complexity theory as well as from previous work that the number of quantifier blocks indeed plays an important role. Similarly, circuit depth seems to be a prominent property of circuits. However, it is indeed striking that only two, and moreover the most straightforward features of circuit QBF suffice to build such robust portfolios. We believe that this opens up a new path of thinking for both solver users and developers. Users can classify their benchmarks and pick a suitable solver more easily, while developers can take advantage of this information to build portfolios within their solvers. Believing many features are necessary to learn anything meaningful about a given instance can be discouraging from even trying. With just two features, the options are much wider—they can be understood

intuitively, or even plotted. In fact, to demonstrate how we can gain additional insight into the problem, we visualize the solver choices made both by the portfolio, as well as by the VBS. When plotting the VBS in Figure 8.4, we ignore instances where the solvers perform too similarly, because they contain more noise than information. On the other hand, we plot the portfolio choices in Figure 8.5 as a grid (of hypothetical instances), in order to discover the decision boundaries. These figures show very clearly which solvers are good for which instances. Incidentally, Figure 8.4 also reveals the fact that the QCIR instances that are available either have many quantifier blocks, or deep circuits, or neither, but not both (strictly speaking, to see that, we would need to plot all instances, but the picture has the same shape, only more noise). This should serve as a challenge to the QBF community to come up with a more complete distribution of benchmark instances.

8.4 Summary and Discussion

With the availability of tools such as AutoFolio, the task of constructing effective per-instance algorithm selectors essentially boils down to designing and implementing features that (jointly) permit to effectively identify which solver to run on any given problem instance. This can still seem daunting in view of the fact that certain domains require rich sets of quickly computable features, with a combination of static and dynamic features, in order to achieve good selector performance [XHHL08]. Our results show that this need not be the case: for circuit QBFs, two or three cheaply computable instance features are sufficient to realize most of the performance potential of a (hypothetical) perfect selector. Moreover, these features include properties of QBFs such as the number of quantifier blocks that are known to affect solver performance. Apart from corroborating the notion that quantifier alternations matter, our results show that circuit depth seems to be important. This warrants further investigation.

Our finding that simple feature sets can be effective likely applies to other problems and encourages an incremental design philosophy: start with a few simple features and add features as needed. As part of future work we hope to find other domains where this approach works well and, more generally, identify the circumstances under which this is the case.

Publication Notes

The research in this chapter appeared in a paper published in the proceedings of the 24th International Conference on Principles and Practice of Constraint Programming (CP 2018) [HPSS18].

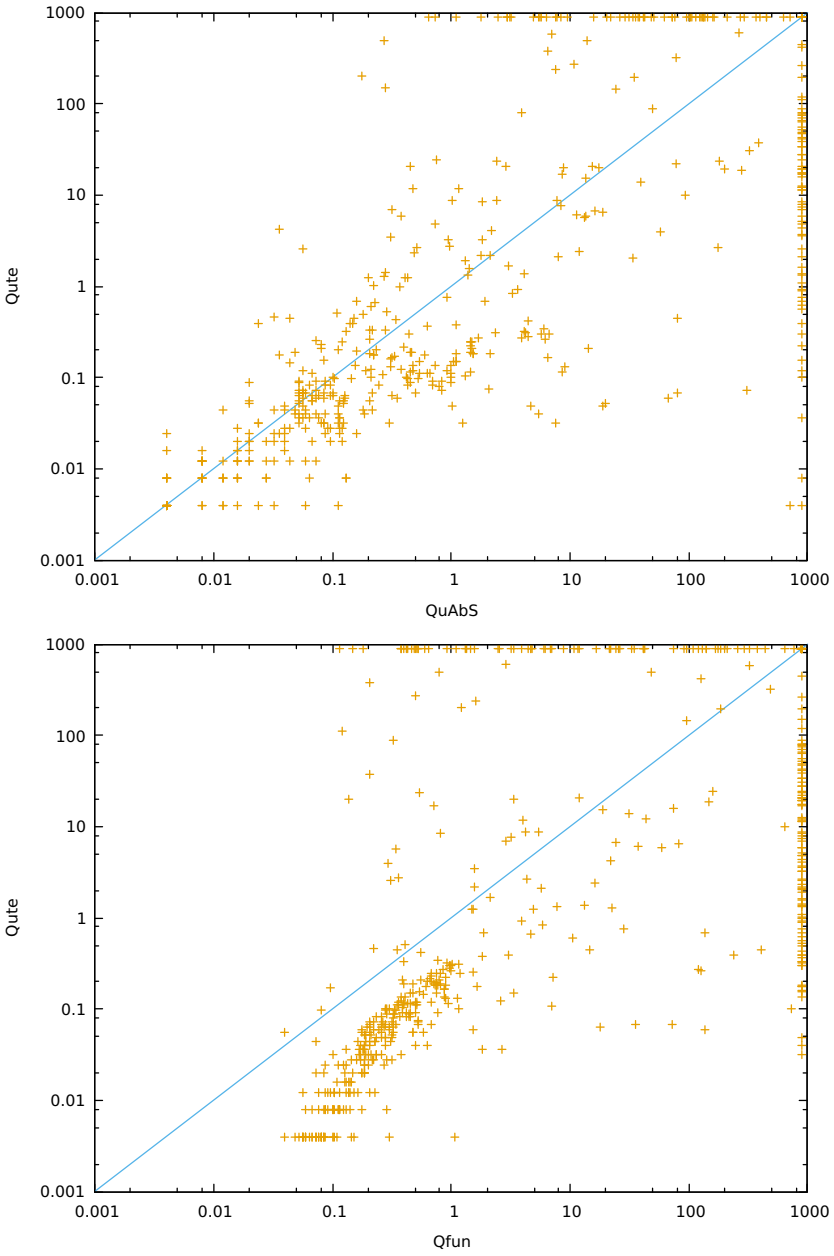

Figure 8.1: Comparisons of high-performance QBF solvers on our instance set; performance is measured as PAR 10 (penalized running times with penalty factor 10) on our reference machines. This shows that there is quite a lot of complementarity between the solvers.

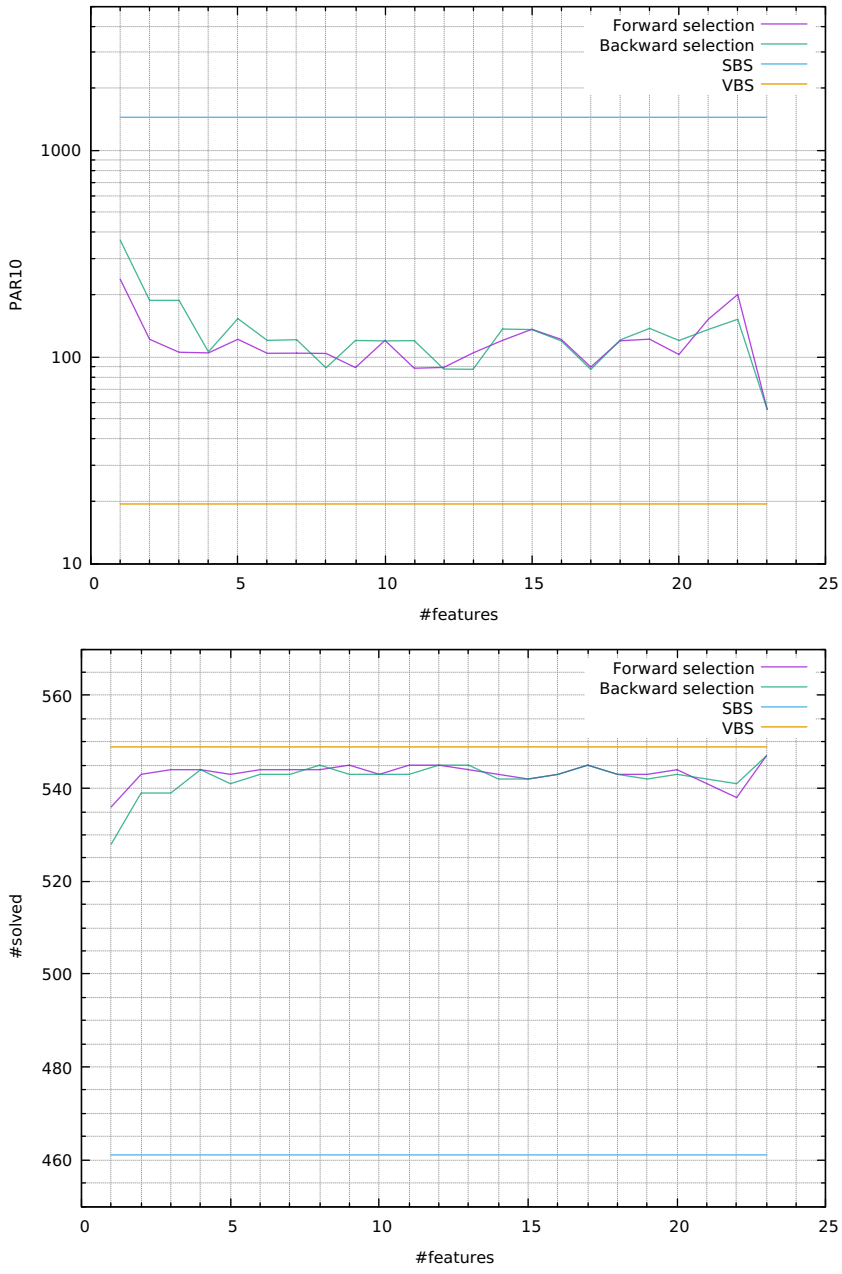

Figure 8.2: Forward and backward selection on the static features; the plots show performance based on the number of features included. Note that for the performance evaluation during forward/backward selection, AutoFolio was not automatically configured for each subset of features, but instead was once configured for the full set of static features at the beginning, and this configuration of hyperparameters was subsequently used for all features sets.

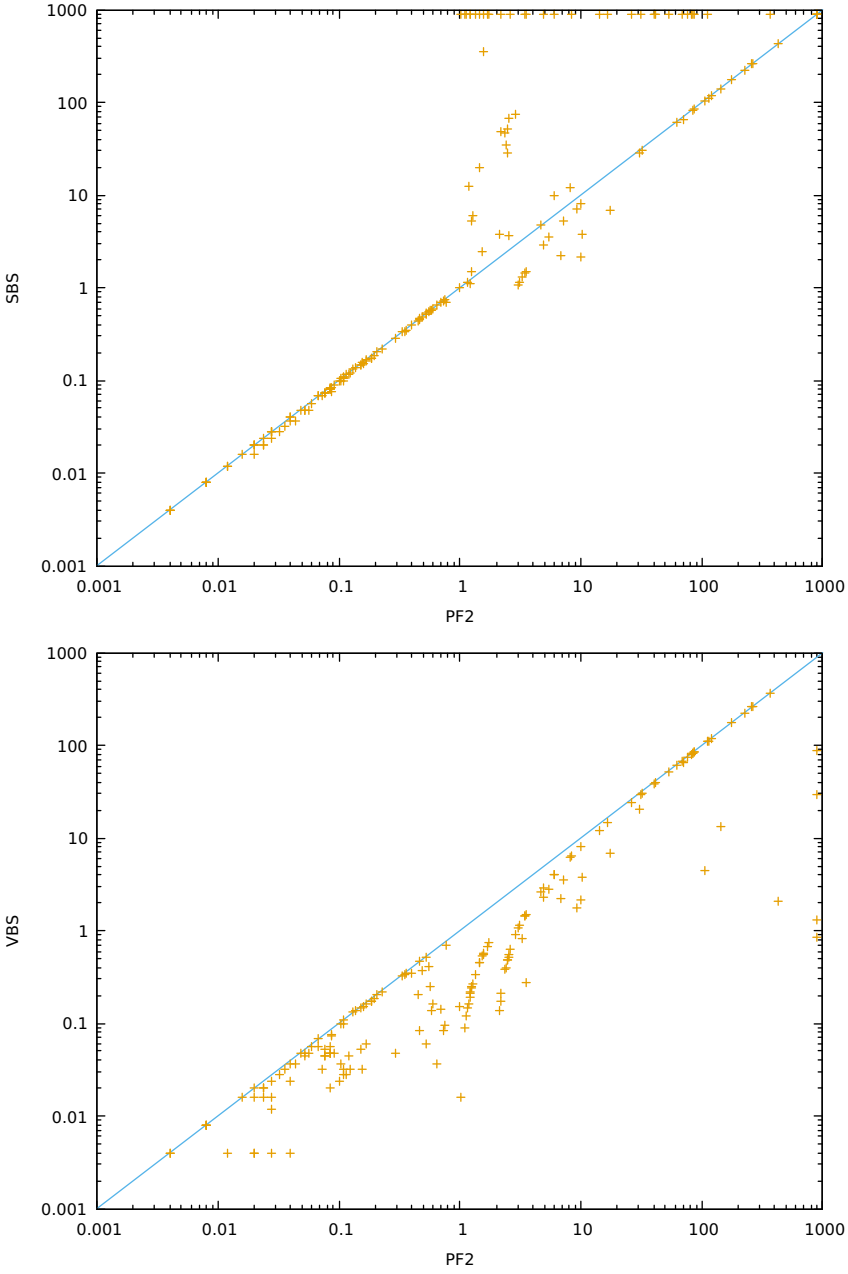

Figure 8.3: Performance of PF2 with all four solvers vs SBS and VBS.

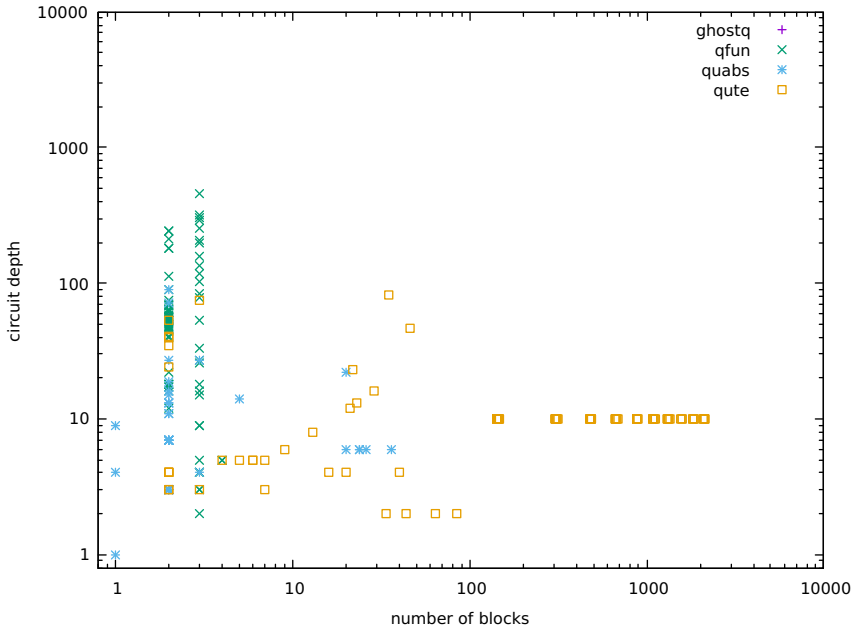

Figure 8.4: Best solver choices based on instance features. Each point represents an instance/solver pair; the coordinates correspond to the number of quantifier blocks and circuit depth of the instance, the shape and color indicate the solver that is fastest on that instance. Only instances where the fastest solver is either the only one to solve the instance, or at least ten times faster than the second fastest, are shown. This is to ensure that the figure shows only solver choices that are crucial, and to avoid instances where the solver choice is unimportant, because all of them run in similar time.

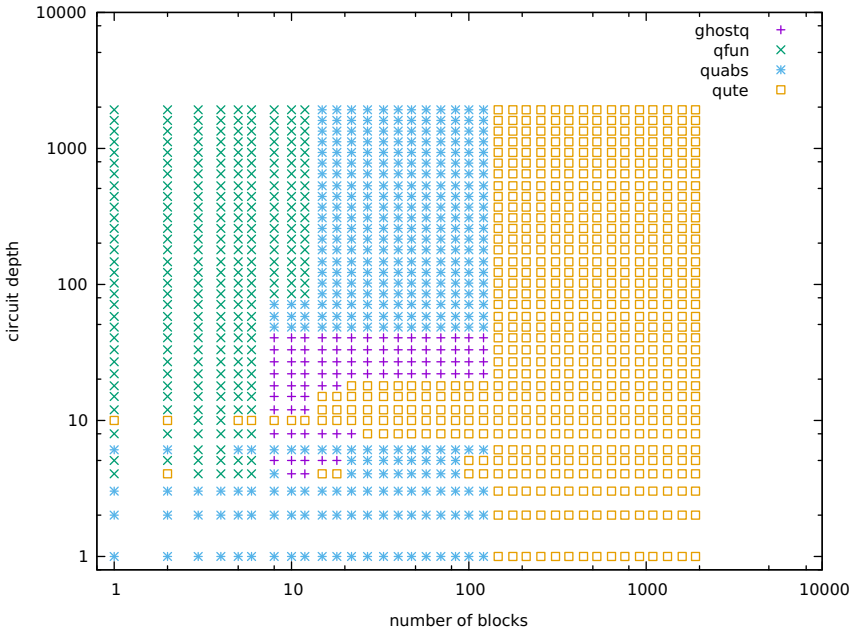

Figure 8.5: Points indicate solver choices of PF2 based on feature values.

CHAPTER 9

Conclusion

We have presented a number of results concerning the theory and practice of quantified Boolean formulas in Chapters 3 to 8. Let us now take a moment to step back and consider the big picture, how our results fit in, and some directions for future research.

In Chapter 3, we have shown that long-distance Q-resolution in QCDCL solving is compatible with the known dependency schemes, and in particular with the reflexive resolution-path dependency scheme. This is an important result, without which Chapter 5 would most likely not have been possible. Indeed, even though Q-resolution appears to be a more natural and intuitive proof system than its tautology-producing sister, this is definitely not the case from the implementation perspective. And it is not just that long-distance Q-resolution is easier to put down in code; when dependency learning comes into play, it is not clear whether an efficient implementation using Q-resolution is even possible.

The main question left open from Chapter 3 is whether, under the condition of normality of D, long-distance Q(D)-consensus is sound. Experimental results from Chapter 5 suggest that this could be the case, but a formal proof will be required in order to close this problem and guarantee soundness of the corresponding solvers for true formulas. Another important open problem is to improve the strategy extraction algorithm for LDQ(D)-resolution. While we now know that a polynomial-time algorithm exists, an entirely different approach, possibly similar to the work of Balabanov et al. [BJJW15], might be necessary to come up with a truly practically efficient algorithm.

Dependency learning, introduced in Chapter 4, is a novel technique for dependency analysis, and a powerful generalization of QCDCL. Recent results show that QCDCL requires exponential time on formulas with short Q-resolution proofs [Jan16], but with dependency learning, these formulas can be solved in polynomial time. This brings us to a different perspective on dependency learning. Instead of seeing it as a dependency-analysis method, we can look at it as a different, more liberal generalization of CDCL to QBF—one

where soundness follows not from a restriction on the order of assignments, but only from the soundness of the underlying proof system. This leads to a natural theoretical question: can QCDCL with dependency learning polynomially simulate (long-distance) Q-resolution, i.e., can it, given the right heuristic choices, find a short proof whenever one exists?

Several questions surrounding dependencies in QBF can be elegantly formulated using dependency quantified Boolean formulas (DQBF) [PRA01]. Informally speaking, in DQBF, every existential variable is assigned a set of universal variables on which it is allowed to depend. Every QBF is also a DQBF, the dependency set being simply the set of all universal variables to the left. We have not touched upon DQBF in this thesis, but it appears to be an appropriate tool to address many open problems in dependency analysis. Let us illustrate that with an example from Chapter 4.

In Section 4.5, we defined the notion of a potential or critical dependency set X of a variable y by shifting y before X in the quantifier prefix (Definitions 14 through 16). A more natural (and stronger) way to define this notion could indeed be in terms of DQBF—instead of shifting the variable y before X, we could simply remove X from the dependency set of y (shifting also removes X from the dependency set of y, but it possibly removes more than just that). However, in that case we could not use long-distance Q-resolution to argue that the restricted formula in the proof of Theorem 7 is false, because long-distance Q-resolution is not sound for DQBF in general [BCSS16]. Nevertheless, the kind of long-distance Q-resolution derivation that appears in that proof is fairly specific, and it is conceivable that a stronger, DQBF-based version of Theorem 7 can be proved by more careful analysis. In order to do that, a further, more detailed study of long-distance Q-resolution in the context of DQBF will be necessary; something we are planning to do as part of future work.

In Chapter 6, we showed how strategies extracted from long-distance Q-resolution proofs can be validated in polynomial time. This is an important step in the direction of streamlining the entire QBF workflow—from encoding, through solving, and certificate extraction, to certificate validation. The SAT community has largely achieved this goal with the DRAT proof system and its derivatives and associated proof checkers [WHH14, Lam17, CFHH+17], which have become the gold standard in certifying UNSAT answers of SAT solvers. Building on these well-tested tools, we eliminated the unpleasant worst-case exponential-time validation step by a SAT solver. Nevertheless, a lot of effort by the entire QBF community will still be necessary in order to standardize the certification of QBF solvers.

Chapter 7 provides an interesting glimpse into the inner workings of long-distance Q-resolution. The two weapons that long-distance Q-resolution has to deal with universal variables—reduction and merging—turn out to be of incomparable power. We also learned that unlike Q-resolution (in a special case), reductionless long-distance Q-resolution does not admit polynomial-time strategy extraction into bounded-depth circuits, potentially hinting at why the merging of long-distance Q-resolution allows for exponentially shorter proofs. In fact, we believe that the mystery of long-distance Q-resolution deserves more

attention, and can potentially hold answers to many questions about QCDCL solving. It is fascinating how the use of seemingly useless tautologies leads to such a rich underlying theory. We are therefore planning to dedicate further effort to the study of long-distance Q-resolution, also in the context of DQBF as mentioned above, in future work.

If Chapter 8 has a single important message, then it is probably this: the choice of a solver for a particular QBF depends first and foremost on the number of quantifier alternations. While the situation is not always completely clear-cut, for instances with a very high number of alternations QCDCL solvers tend to work well, and for instances with a very low number of alternations expansion-based solvers are preferable. An important consequence of this finding is that of benchmark selection. We should be careful not to over- or under-represent a certain class of benchmarks in order not to make a particular solving paradigm appear inferior and thus hinder its development.

QBF should ultimately serve as a technology to solve hard problems. This requires a number of tools—great solvers, standardized formats, convenient encoding methods, strategy manipulators to name a few. In this thesis, we touched upon many of these topics, and developed new algorithms and settled open theoretical problems. As it usually happens, our work has raised more new questions than it provided answers. Ultimately, the goal of this and related research is to make QBF technology capable of solving interesting real-world problems from all sorts of applications. And while that target still remains more distant than, say, with SAT, this thesis has shed light and helped zoom in on some of the important issues, and should hopefully serve as a source of both knowledge and inspiration for QBF researchers working towards that ultimate goal.

List of Figures

List of Tables

List of Algorithms

Index

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[AS09] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In Craig Boutilier, editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 399–404, 2009.

[BB16] Olaf Beyersdorff and Joshua Blinkhorn. Dependency schemes in QBF calculi: Semantics and soundness. In Michel Rueher, editor, *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016*, volume 9892 of *Lecture Notes in Computer Science*, pages 96–112. Springer Verlag, 2016.

[BB17] Joshua Blinkhorn and Olaf Beyersdorff. Shortening QBF proofs with dependency schemes. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing - SAT 2017*, volume 10491 of *Lecture Notes in Computer Science*, pages 263–280. Springer Verlag, 2017.

[BBM19] Olaf Beyersdorff, Joshua Blinkhorn, and Meena Mahajan. Building strategies into QBF proofs. In *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, pages 14:1–14:18, 2019.

[BCJ15] Olaf Beyersdorff, Leroy Chew, and Mikoláš Janota. Proof complexity of resolution-based QBF calculi. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 76–89. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[BCSS16] Olaf Beyersdorff, Leroy Chew, Renate A. Schmidt, and Martin Suda. Lifting QBF resolution calculi to DQBF. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing – SAT 2016*, pages 490–499, Cham, 2016. Springer International Publishing.

[BF15] Armin Biere and Andreas Fröhlich. Evaluating CDCL variable scoring schemes. In Marijn Heule and Sean Weaver, editors, *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, volume 9340 of *Lecture Notes in Computer Science*, pages 405–422. Springer Verlag, 2015.

[Bie08] Armin Biere. Adaptive restart strategies for conflict driven SAT solvers. In Hans Kleine Büning and Xishun Zhao, editors, *Theory and Applications of Satisfiability Testing - SAT 2008, 11th International Conference, SAT 2008, Guangzhou, China, May 12-15, 2008. Proceedings*, volume 4996 of *Lecture Notes in Computer Science*, pages 28–33. Springer Verlag, 2008.

[BJ12] Valeriy Balabanov and Jie-Hong Roland Jiang. Unified QBF certification and its applications. *Formal Methods in System Design*, 41(1):45–65, 2012.

[BJJW15] Valeriy Balabanov, Jie-Hong Roland Jiang, Mikoláš Janota, and Magdalena Widl. Efficient extraction of QBF (counter)models from long-distance resolution proofs. In Blai Bonet and Sven Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 3694–3701. AAAI Press, 2015.

[BJK15] Nikolaj Bjørner, Mikoláš Janota, and William Klieber. On conflicts and strategies in QBF. In Ansgar Fehnker, Annabelle McIver, Geoff Sutcliffe, and Andrei Voronkov, editors, *20th International Conferences on Logic for Programming, Artificial Intelligence and Reasoning - Short Presentations, LPAR 2015, Suva, Fiji, November 24-28, 2015.*, volume 35 of *EPiC Series in Computing*, pages 28–41. EasyChair, 2015.

[BL09] Arming Biere and Florian Lonsing. A compact representation for syntactic dependencies in QBFs. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009*, volume 5584 of *Lecture Notes in Computer Science*, pages 398–411. Springer Verlag, 2009.

[BL10] Armin Biere and Florian Lonsing. Integrating dependency schemes in search-based QBF solvers. In Ofer Strichman and Stefan Szeider, editors, *Theory and Applications of Satisfiability Testing - SAT 2010*, volume 6175 of *Lecture Notes in Computer Science*, pages 158–171. Springer Verlag, 2010.

[BW98] Beate Bollig and Ingo Wegener. A very simple function that requires exponential size read-once branching programs. *Inf. Process. Lett.*, 66(2):53–57, 1998.

[CFHH+17] Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt, Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified RAT verification. In Leonardo de Moura, editor, *Automated Deduction – CADE 26*, pages 220–236, Cham, 2017. Springer International Publishing.

[Che17] Leroy Nicholas Chew. *QBF proof complexity*. PhD thesis, University of Leeds, UK, 2017.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd Annual Symp. on Theory of Computing*, pages 151–158, Shaker Heights, Ohio, 1971.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *J. of the ACM*, 7(3):201–215, 1960.

[ELW13] Uwe Egly, Florian Lonsing, and Magdalena Widl. Long-distance resolution: Proof generation and strategy extraction in search-based QBF solving. In Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - LPAR 2013*, volume 8312 of *Lecture Notes in Computer Science*, pages 291–308. Springer Verlag, 2013.

[ES03] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer Verlag, 2003.

[GB13] Alexandra Goultiaeva and Fahiem Bacchus. Recovering and utilizing partial duality in QBF. In Matti Järvisalo and Allen Van Gelder, editors, *Theory and Applications of Satisfiability Testing - SAT 2013*, volume 7962 of *Lecture Notes in Computer Science*, pages 83–99. Springer Verlag, 2013.

[Gel12] Allen Van Gelder. Contributions to the theory of practical quantified Boolean formula solving. In Michela Milano, editor, *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012*, volume 7514 of *Lecture Notes in Computer Science*, pages 647–663. Springer Verlag, 2012.

[GNPT05] E. Giunchiglia, M. Narizzano, L. Pulina, and A. Tacchella. Quantified Boolean Formulas satisfiability library (QBFLIB), 2005. `www.qbflib.org`.

[GNT06] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Clause/term resolution and learning in the evaluation of Quantified Boolean Formulas. *J. Artif. Intell. Res.*, 26:371–416, 2006.

[GVGB11] Alexandra Goultiaeva, Allen Van Gelder, and Fahiem Bacchus. A uniform approach for generating proofs and strategies for both true and false QBF formulas. In Toby Walsh, editor, *Proceedings of IJCAI 2011*, pages 546–553. IJCAI/AAAI, 2011.

[Hås87] Johan Håstad. *Computational Limitations of Small-depth Circuits*. MIT Press, Cambridge, MA, USA, 1987.

[Heu18] Marijn J. H. Heule. Schur number five. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 6598–6606, 2018.

[HHKW17] Marijn Heule, Warren Hunt, Matt Kaufmann, and Nathan Wetzler. Efficient, verified checking of propositional proofs. In Mauricio Ayala-Rincón and César A. Muñoz, editors, *Interactive Theorem Proving*, pages 269–284, Cham, 2017. Springer International Publishing.

[HKM16] Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 228–245. Springer Verlag, 2016.

[HPSS18] Holger H. Hoos, Tomáš Peitl, Friedrich Slivovsky, and Stefan Szeider. Portfolio-based algorithm selection for circuit QBFs. In John N. Hooker, editor, *Principles and Practice of Constraint Programming - 24th International Conference, CP 2018*, volume 11008 of *Lecture Notes in Computer Science*, pages 195–209. Springer Verlag, 2018.

[IMMV16] Alexander Ivrii, Sharad Malik, Kuldeep S. Meel, and Moshe Y. Vardi. On computing minimal independent support and its applications to sampling and counting. *Constraints*, 21(1):41–58, 2016.

[Jan16] Mikoláš Janota. On Q-resolution and CDCL QBF solving. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 402–418. Springer Verlag, 2016.

[Jan18a] Mikoláš Janota. Circuit-based search space pruning in qbf. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing – SAT 2018*, pages 187–198, Cham, 2018. Springer International Publishing.

[Jan18b] Mikolás Janota. Towards generalization in QBF solving via machine learning. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence - AAAI 2018*. AAAI Press, 2018.

[JKMSC12] Mikoláš Janota, William Klieber, João Marques-Silva, and Edmund M. Clarke. Solving QBF with counterexample guided refinement. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing - SAT 2012*, volume 7317 of *Lecture Notes in Computer Science*, pages 114–128. Springer Verlag, 2012.

[JKS16] Charles Jordan, Will Klieber, and Martina Seidl. Non-CNF QBF solving with QCIR. In Adnan Darwiche, editor, *Beyond NP, Papers from the 2016 AAAI Workshop.*, volume WS-16-05 of *AAAI Workshops*. AAAI Press, 2016.

[JM15] Mikoláš Janota and Joao Marques-Silva. Solving QBF by clause selection. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015*, pages 325–331. AAAI Press, 2015.

[JM17] Mikoláš Janota and Joao Marques-Silva. An Achilles' heel of term-resolution. In Eugénio C. Oliveira, João Gama, Zita A. Vale, and Henrique Lopes Cardoso, editors, *Progress in Artificial Intelligence - 18th EPIA Conference on Artificial Intelligence, EPIA 2017*, volume 10423 of *Lecture Notes in Computer Science*, pages 670–680. Springer Verlag, 2017.

[JMS15] Mikoláš Janota and Joao Marques-Silva. Expansion-based QBF solving versus Q-resolution. *Theoretical Computer Science*, 577(0):25–42, April 2015.

[KKF95] H. Kleine Büning, M. Karpinski, and A. Flögel. Resolution for quantified Boolean formulas. *Information and Computation*, 117(1):12–18, 1995.

[Kot16] Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. In Christian Bessiere, Luc De Raedt, Lars Kotthoff, Siegfried Nijssen, Barry O'Sullivan, and Dino Pedreschi, editors, *Data Mining and Constraint Programming - Foundations of a Cross-Disciplinary Approach*, volume 10101 of *Lecture Notes in Computer Science*, pages 149–190. Springer, 2016.

[KP06] Volker Kaibel and Matthias Peinhardt. On the bottleneck shortest path problem. Zib-report 06-22, Zuse Institute Berlin, 2006.

[KSGC10] William Klieber, Samir Sapra, Sicun Gao, and Edmund M. Clarke. A non-prenex, non-clausal QBF solver with game-state learning. In Ofer Strichman and Stefan Szeider, editors, *Theory and Applications of Satisfiability Testing - SAT 2010*, volume 6175 of *Lecture Notes in Computer Science*, pages 128–142. Springer Verlag, 2010.

[Lam17] Peter Lammich. Efficient verified (un)sat certificate checking. In Leonardo de Moura, editor, *Automated Deduction – CADE 26*, pages 237–254. Springer International Publishing, 2017.

[LBB+15] Florian Lonsing, Fahiem Bacchus, Armin Biere, Uwe Egly, and Martina Seidl. Enhancing search-based QBF solving by dynamic blocked clause elimination. In Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings*, volume 9450 of *Lecture Notes in Computer Science*, pages 418–433. Springer Verlag, 2015.

[LE18] Florian Lonsing and Uwe Egly. Evaluating QBF solvers: Quantifier alternations matter. In John N. Hooker, editor, *Principles and Practice of Constraint Programming - 24th International Conference, CP 2018*, volume 11008 of *Lecture Notes in Computer Science*, pages 276–294. Springer Verlag, 2018.

[Lev73] Leonid Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265—266, 1973.

[LEVG13] Florian Lonsing, Uwe Egly, and Allen Van Gelder. Efficient clause learning for quantified Boolean formulas via QBF pseudo unit propagation. In Matti Järvisalo and Allen Van Gelder, editors, *Theory and Applications of Satisfiability Testing - SAT 2013*, volume 7962 of *Lecture Notes in Computer Science*, pages 100–115. Springer Verlag, 2013.

[LHHS15] Marius Thomas Lindauer, Holger H. Hoos, Frank Hutter, and Torsten Schaub. Autofolio: An automatically configured algorithm selector. *J. Artif. Intell. Res.*, 53:745–778, 2015.

[LLM16] Jean-Marie Lagniez, Emmanuel Lonca, and Pierre Marquis. Improving model counting by leveraging definability. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 751–757. IJCAI/AAAI Press, 2016.

[LM08] Jérôme Lang and Pierre Marquis. On propositional definability. *Artificial Intelligence*, 172(8-9):991–1017, 2008.

[Lon12] Florian Lonsing. *Dependency Schemes and Search-Based QBF Solving: Theory and Practice*. PhD thesis, Johannes Kepler University, Linz, Austria, April 2012.

[MLM09] João P. Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning sat solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, pages 131–153. IOS Press, 2009.

[MS96] João P. Marques-Silva and Karem A. Sakallah. GRASP - a new search algorithm for satisfiability. In *International Conference on Computer-Aided*

Design (ICCAD '96), November 10-14, 1996, San Jose, CA, USA, pages 220–227. ACM and IEEE, 1996.

[Nec66] 'E. I. Nechiporuk. A Boolean function. *Dokl. Akad. Nauk SSSR*, 169(4):765–766, 1966.

[PRA01] 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.

[PSS16] Tomáš Peitl, Friedrich Slivovsky, and Stefan Szeider. Long distance Q-resolution with dependency schemes. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 500–518. Springer Verlag, 2016.

[PSS17] Tomáš Peitl, Friedrich Slivovsky, and Stefan Szeider. Dependency learning for QBF. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing - SAT 2017*, volume 10491 of *Lecture Notes in Computer Science*, pages 298–313. Springer Verlag, 2017.

[PSS18] Tomáš Peitl, Friedrich Slivovsky, and Stefan Szeider. Polynomial-time validation of QCDCL certificates. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing – SAT 2018*, pages 253–269, Cham, 2018. Springer International Publishing.

[PSS19a] Tomáš Peitl, Friedrich Slivovsky, and Stefan Szeider. Dependency learning for QBF. *Journal of Artificial Intelligence Research*, vol. 65, 2019.

[PSS19b] Tomáš Peitl, Friedrich Slivovsky, and Stefan Szeider. Long-distance Q-resolution with dependency schemes. *Journal of Automated Reasoning*, 63(1):127–155, Jun 2019.

[PSS19c] Tomáš Peitl, Friedrich Slivovsky, and Stefan Szeider. Combining resolution-path dependencies with dependency learning. In Mikoláš Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, Lisbon, Portugal, July 9-12, 2019, Proceedings*, Lecture Notes in Computer Science. Springer Verlag, 2019. To appear.

[PSS19d] Tomáš Peitl, Friedrich Slivovsky, and Stefan Szeider. Proof complexity of fragments of long-distance Q-resolution. In Mikoláš Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, Lisbon, Portugal, July 9-12, 2019, Proceedings*, Lecture Notes in Computer Science. Springer Verlag, 2019. To appear.

[PT09] Luca Pulina and Armando Tacchella. A self-adaptive multi-engine solver for quantified Boolean formulas. *Constraints*, 14(1):80–116, 2009.

[Pul16] Luca Pulina. The ninth QBF solvers evaluation - preliminary report. In Florian Lonsing and Martina Seidl, editors, *Proceedings of the 4th International Workshop on Quantified Boolean Formulas (QBF 2016).*, volume 1719 of *CEUR Workshop Proceedings*, pages 1–13. CEUR-WS.org, 2016.

[Riv87] Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.

[RT15] Markus N. Rabe and Leander Tentrup. CAQE: A certifying QBF solver. In Roope Kaivola and Thomas Wahl, editors, *Formal Methods in Computer-Aided Design - FMCAD 2015*, pages 136–143. IEEE Computer Soc., 2015.

[Rya04] Lawrence Ryan. Efficient algorithms for clause-learning SAT solvers. Master's thesis, Simon Fraser University, 2004.

[Sli15] Friedrich Slivovsky. *Structure in #SAT and QBF*. PhD thesis, TU Wien, May 2015.

[SM73] Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time. In *Proc. Theory of Computing*, pages 1–9. ACM, 1973.

[SS09] Marko Samer and Stefan Szeider. Backdoor sets of quantified Boolean formulas. *Journal of Automated Reasoning*, 42(1):77–97, 2009.

[SS12] Friedrich Slivovsky and Stefan Szeider. Computing resolution-path dependencies in linear time. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing - SAT 2012*, volume 7317 of *Lecture Notes in Computer Science*, pages 58–71. Springer Verlag, 2012.

[SS16a] Friedrich Slivovsky and Stefan Szeider. Quantifier reordering for QBF. *Journal of Automated Reasoning*, 56(4):459–477, 2016.

[SS16b] Friedrich Slivovsky and Stefan Szeider. Soundness of Q-resolution with dependency schemes. *Theoretical Computer Science*, 612:83–101, 2016.

[Ten16] Leander Tentrup. Non-prenex QBF solving using abstraction. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016*, volume 9710 of *Lecture Notes in Computer Science*, pages 393–401. Springer Verlag, 2016.

[Tse68] G. S. Tseitin. On the complexity of derivation in propositional calculus. *Zap. Nauchn. Sem. Leningrad Otd. Mat. Inst. Akad. Nauk SSSR*, 8:23–41, 1968. Russian. English translation in J. Siekmann and G. Wrightson (eds.) *Automation of Reasoning. Classical Papers on Computer Science 1967–1970*, Springer Verlag, 466–483, 1983.

[VG11] Allen Van Gelder. Variable independence and resolution paths for quantified Boolean formulas. In Jimmy Lee, editor, *Principles and Practice of Constraint Programming - CP 2011*, volume 6876 of *Lecture Notes in Computer Science*, pages 789–803. Springer Verlag, 2011.

[VWM15] Yakir Vizel, Georg Weissenbacher, and Sharad Malik. Boolean satisfiability solvers and their applications in model checking. *Proceedings of the IEEE*, 103(11):2021–2035, 2015.

[Weg00] Ingo Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, 2000.

[WHH14] Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt. Drat-trim: Efficient checking and trimming using expressive clausal proofs. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing – SAT 2014*, pages 422–429. Springer Verlag, 2014.

[WRMB17] Ralf Wimmer, Sven Reimer, Paolo Marin, and Bernd Becker. HQSpre - an effective preprocessor for QBF and DQBF. In Axel Legay and Tiziana Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017*, volume 10205 of *Lecture Notes in Computer Science*, pages 373–390, 2017.

[XHHL08] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla: Portfolio-based algorithm selection for SAT. *J. Artif. Intell. Res.*, 32:565–606, 2008.

[ZM02a] Lintao Zhang and Sharad Malik. Conflict driven learning in a quantified Boolean satisfiability solver. In Lawrence T. Pileggi and Andreas Kuehlmann, editors, *Proceedings of the 2002 IEEE/ACM International Conference on Computer-aided Design, ICCAD 2002, San Jose, California, USA, November 10-14, 2002*, pages 442–449. ACM / IEEE Computer Society, 2002.

[ZM02b] Lintao Zhang and Sharad Malik. The quest for efficient Boolean satisfiability solvers. In D. Brinksma and K. G. Larsen, editors, *Computer Aided Verification: 14th International Conference (CAV 2002)*, volume 2404 of *Lecture Notes in Computer Science*, pages 17–36, 2002.

[ZM02c] Lintao Zhang and Sharad Malik. Towards a symmetric treatment of satisfaction and conflicts in quantified Boolean formula evaluation. In Pascal Van Hentenryck, editor, *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings*, volume 2470 of *Lecture Notes in Computer Science*, pages 200–215. Springer Verlag, 2002.