

國立臺灣大學電子工程學研究所
Graduate Institute of Electronics Engineering National
Taiwan University



PhD thesis

Certification in Quantified Decision Procedures

量詞布林公式之歸結證明系統
與正反模型萃取

Valeriy Balabanov
包偉力

Advisor: professor Jie-Hong Roland Jiang
教授：江介宏

February 2015

中華民國 104 年 2 月

Acknowledgements



This work is a result of more than five years cooperation with professor Jie-Hong Roland Jiang - my advisor, teacher and friend. He guided both my research and my graduate life with careful advices and always useful suggestions. Synergetic research with professor Jiang on Quantified Boolean Formulas (which is the main topic of this thesis) led to many interesting discoveries, whose practical value was experimentally confirmed.

I would also like to thank professors Bow-Yaw Wang, Fang Yu, Yu-Fang Chen and Chung-Yang Ric Huang for their kind agreement to be on my oral defense committee, for sparing their limited free time, and for giving valuable suggestions regarding the content of this work. Another word of appreciation goes to all the members of ALCom Lab for sophisticated discussions and debates that made my work and life in lab surprisingly exciting.

Last, but definitely the most, I would like to thank my family for their support in my (sometimes difficult) journey. My dear parents Balabanov Igor and Balabanova Tetyana always were there for me, and undoubtedly I will always be there for them too. My twin brothers Oleg and Oleksandr helped me to stay in tune by their sharp riddles and interesting ideas, for which I am very grateful too. Also thank all my friends for their great support.

Valeriy Balabanov

*National Taiwan University
January 2015*

Certification in Quantified Decision Procedures

Student: Valeriy Balabanov Advisor: Prof. Jie-Hong Roland Jiang

Graduate Institute of Electronics Engineering
National Taiwan University

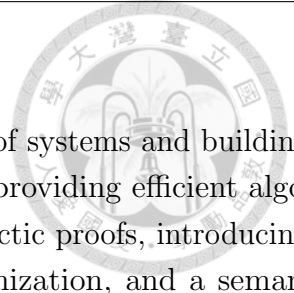
Abstract

Satisfiability (SAT) plays one of the key roles in the modern automated reasoning. For the past decade a huge success in SAT solving was achieved by researchers, and nowadays SAT-based decision algorithms scale well for many real-world applications. Quantified Boolean Formulas (QBFs) extend formulas of propositional logic by allowing existential and universal quantifications over variables. Quantifiers make QBF more expressive and allow a more compact constraints representation compared to SAT. In addition to problems encountered in formal verification, QBF has applications in planning, two-player games, electronic design automation, and other fields of computer science.

Many methods have been proposed for efficient SAT solving, including search-based solving with different learning techniques (e.g., conflict driven clause learning), syntactic rewriting methods (e.g., resolution rules), various preprocessing techniques (e.g., variable elimination), and so on. Many of these techniques were then adapted for QBF evaluation (e.g., extensions of search-based algorithms, introduction of Q-resolution, etc). Despite its similarities to SAT solving, evaluating QBFs has its unique features related to the quantification nature.

Certification of QBFs is of a great interest: Not only it confirms the answer provided by the solver (e.g., by providing syntactic Q-resolution proof), but more importantly directly provides information required in many synthesis applications (e.g., by returning a semantic model). Semantic QBF certificates can serve as a winning strategy in two-player games, the functional representation of a relation in a determinization problem, and other applications. Despite the importance of QBF evaluation and certification, solving algorithms remain immature, and many related issues are not well studied yet.

In this work we provide an overview of different QBF solving and certifying techniques, and propose novel approaches that experimentally show improvements over existing ones.



Our main contributions include developing new QBF proof systems and building corresponding syntactic certificates based on Q-resolution, providing efficient algorithms for semantic certificates extraction from various syntactic proofs, introducing new optimization heuristics for semantic certificates minimization, and a semantic classification method for dependency QBFs (DQBFs). Along with theoretical achievements, conducted experiments show that proposed methods can be useful in modern applications, therefore forming solid practical values.

量詞布林公式之歸結證明系統 與正反模型萃取



學生：包偉力

教授：江介宏

國立臺灣大學電子工程學研究所

摘要

可滿足性 (Satisfiability, SAT) 測試在當代的自動化推理領域扮演著關鍵的角色。在近十年眾多研究人員的努力下，SAT 求解技術已臻成熟而能有效擴展解決大型問題，成功地被廣泛運用於各式業界的邏輯求解問題上。相較於 SAT，量化布林公式 (Quantified Boolean Formula, QBF) 允許存在性量詞與所有性量詞對變量的量化設限，因而擴展了命題邏輯公式的表達力，能更簡潔地表示約束式。除了在正規驗證的應用，QBF 在電子設計自動化、人工智慧規劃問題、對弈賽局等計算機科學領域皆有廣泛的應用。

許多 SAT 的有效求解方法已被提出，包括具各種學習技術的搜尋方法 (例如，衝突驅動的條件學習)，語法重寫方法 (例如，歸結規則)，不同的預處理技術 (例如，純文本消除)，等等。許多這些技術被修改採用於 QBF 求解 (例如，基於學習搜尋的演算法，Q-歸結規則等擴展)。儘管其與 SAT 求解有相似處，QBF 求解因其所具的量化功能有獨特的特殊性，QBF 求解在計算複雜度上屬 PSPACE-complete，為多項式階層中最困難的問題，被視為比 SAT 求解等的 NP-complete 類別更困難許多。

QBFs 的認證近來引起極大的關注：它(QBF 語法認證)不僅於用來驗證 QBF 求解的正確性，(QBF 語意認證)更能夠直接地提供許多合成應用上所需的資訊。QBF 的語意認證可以是，例如，在兩人對弈賽局中的穩贏策略，在關係確定化問題中的函數式答案，和其它實際對應。縱使 QBF 的求解和認證有極高的重要性，其相關的演算法仍不成熟，許多關鍵問題仍然沒有很好的解法。

在本論文，我們總覽各式 QBF 求解和認證技術，提出新的 QBF 證明系統和認證方法，並以實驗證實新方法相較於現有方法的優越性。我們的主要貢獻包括開發，基於 Q-歸結規則，新的 QBF 證明系統與其語法認證、開發高效的演算法自擴展的語法認證提取語意認證、開發優化演算法化簡語意認證、開發 Dependency QBF (DQBF) 語意分類方法等成果。除理論貢獻外，我們還以實驗表明所提出方法在當前應用上的具體實用價值。



Contents

Acknowledgements	i
Abstract	ii
Chinese Abstract	v
1 Preamble	2
1.1 Motivation	2
1.2 Our contributions and structure of the thesis	4
2 Introduction to SAT and QBF	6
2.1 Boolean algebra	6
2.1.1 Truth Tables	7
2.1.2 And Inverter Graphs (AIGs)	8
2.1.3 Binary Decision Diagrams (BDDs)	9
2.1.4 Conjunctive and Disjunctive Normal Forms (CNF and DNF) .	10
2.2 Satisfiability problem (SAT)	11
2.2.1 Preliminaries	11
2.2.2 Satisfiability equivalence and Tseitin encoding	13
2.2.3 Solving SAT	15
2.2.4 Heuristics in SAT solving	17
2.2.5 SAT applications	18
2.3 Quantified Boolean Formulae (QBF)	19
2.3.1 Preliminaries	20
2.3.2 QBF semantics	22
2.3.3 Tseitin transformation for QBF	24
2.3.4 Q-resolution	25
2.3.5 Solving QBF	27
2.3.6 QBF certificates applications	30



3	Syntactic certificates and QBF	32
3.1	Q-resolution proof system	32
3.1.1	Introduction revisited	33
3.1.2	Intractability of Q-resolution	35
3.2	Extensions of Q-resolution proof system	40
3.2.1	LQ- and QU-resolution proof systems	40
3.2.2	Transformation of LQ-resolution proofs to Q-resolution proofs	41
3.2.3	Incomparability of LQ- and QU-resolutions	48
3.3	New resolution proof systems	54
3.3.1	Introducing LQU and LQU+	54
3.3.2	Superiority and Limitation of LQU and LQU+	55
4	Semantic certificates and QBF	59
4.1	Models and countermodels	60
4.1.1	Skolemization	60
4.1.2	Model generation using Skolemization	60
4.2	Extraction of Skolem and Herbrand functions from Q-resolution proofs	62
4.2.1	Counter-model Construction from Clause-Resolution Proofs .	62
4.2.2	Model Construction from Cube-Resolution Proofs	70
4.2.3	Experimental evaluation	73
4.3	Flexibilities and Optimization of Skolem and Herbrand Functions . .	77
4.3.1	Effective pure literals detection	77
4.3.2	Optimization with Flexibility of Topological Ordering	81
4.3.3	Postponing forall-reduction	82
4.3.4	Don't care minimization	87
4.3.5	Effects of variable sorting on certificate size	89
4.3.6	Experimental evaluation	90
4.4	Extension of the extraction algorithm to other proof systems	94
4.4.1	Polynomial extraction of Skolem and Herbrand functions from LQ(U+)-resolution proofs	94
4.4.2	Experimental evaluation	104
5	Beyond Quantified Boolean Formulae	109
5.1	Dependency Quantified Boolean Formulas	109
5.1.1	Introduction to Henkin quantifiers	109
5.1.2	DQBF preliminaries	111
5.1.3	Applications and prior work	113

5.2	DQBF properties	115
5.2.1	Semantic classification of DQBF	115
5.2.2	Formula Expansion on Existential Variables	122
5.2.3	Prenex and Non-prenex Conversion	123
5.3	DQ-resolution	125
6	Conclusions and future work	130
	List of Figures	134
	List of Tables	135
	Bibliography	140



Chapter 1

Preamble

1.1 Motivation

More than for fifty years logicians are studying Boolean satisfiability problem (SAT). It has a huge theoretical importance to the mathematics and theoretical computer science, as it is one of the core NP-complete problems [12]. Many problems in complexity theory have reductions to SAT and its derivatives. Besides its theoretical value, SAT is one of the most widely studied problems due to its numerous applications to automated reasoning [12]. Tremendous improvement in SAT-solving has been made for the past decade, and nowadays SAT lies at the heart of practically every hardware verification tool. It is impossible to imagine automated IC design and verification today without SAT solving. Equivalence checking, model checking, logic optimization and many other problems effectively exploit SAT [12]. Other sections of verification, such as testing, also make use of SAT, as in, for example, automatic test pattern generation. Despite the great success of SAT, it has limitations. SAT cannot directly deal with quantifiers, it is bounded by its complexity class in terms of the representative power, and it has limited use when it comes to function derivation.

Figure 1.1 shows the dependence of CPU transistor count on the production year of the design. It verifies the Moore's law, which suggests that transistor count doubles every two years. This emergence of silicon technology has lead to ICs with billions of transistors in 2008, and each year it pushes the effort spent for their verification higher and higher.

1.1. Motivation

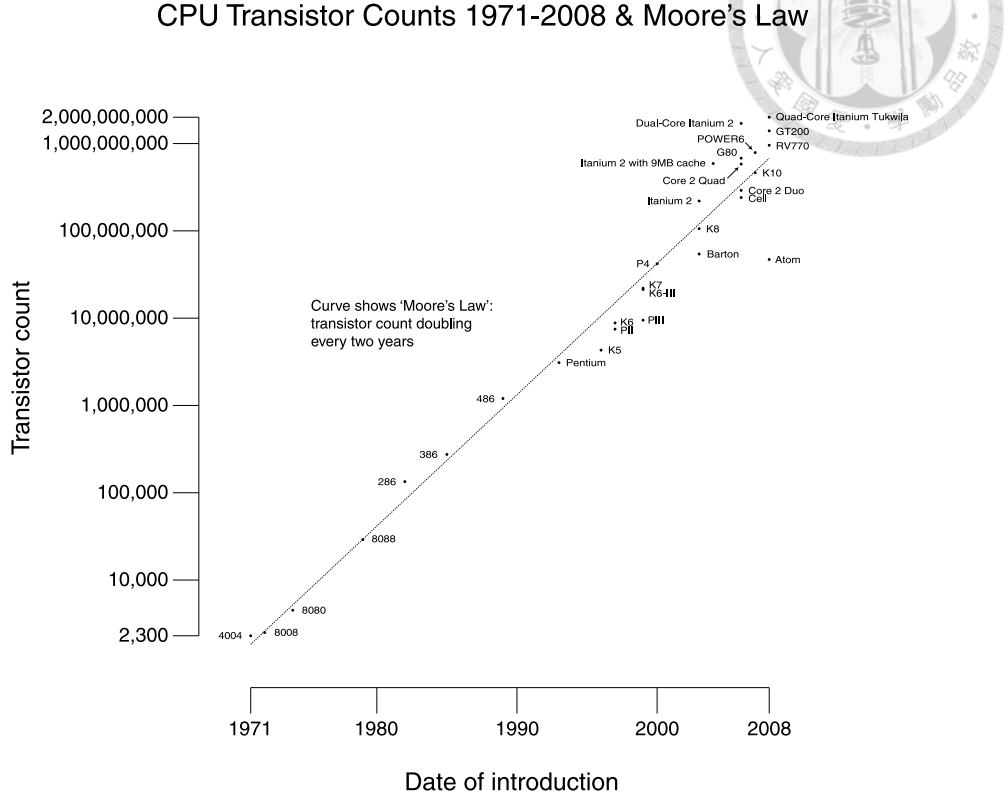


Figure 1.1: Transistor count and Moore's law.

Innovational saturation in SAT-solving is close therefore one has to look beyond SAT for new algorithms. Quantified Boolean formulae (QBF) stands for a wider class of Boolean formulas that allows quantification over free variables. QSAT - is the decision problem related to QBF (in the context we might simply call it as a “QBF solving” problem, as it is more intuitive despite its informality). QSAT is PSPACE-complete problem, and has higher complexity power compare to SAT. QBF overcomes SAT limitations: it syntactically allows quantifiers, semantically supports multi-agent scenarios and is richer in terms of encoding function-based problems.

Certification plays an important role in both SAT and QSAT [12]. Usually it is more desirable to provide a certificate for a given by solver answer. Not only certificate can serve as a proof for the solver's correctness, but it also can be used by designers. In propositional satisfiability certificate for a true formula is a satisfying assignment to variables that can, for example, indicate the false path leading to a bug in bounded model checking. True certificates are easy to verify as by the definition of NP problems they have to be polynomial in the problem size. More difficult is to provide and verify a certificate for false instances. Up to date, the most

commonly accepted certificates for false SAT instances are the, so-called, resolution proofs. Unfortunately worst-case exponential lower bounds have been proven for the size of resolution proofs. Nevertheless, in practice resolution proofs can be efficiently used, for example, for the interpolant extraction in image overapproximation based algorithms in model checking.

For QBF the situation is more complicated. Certificates can be represented either as functional (counter)models or syntactic (resolution-like) proofs. Certificates generation is more complicated, but functional certificates are as valuable as their SAT counterparts. QBF certificates may represent strategies for multi-agent scenarios, plans in planning or scheduling problems, or functional representation of Boolean relations used for synthesis. QBF certification will be the main topic of this work.

1.2 Our contributions and structure of the thesis

Yearly, QBF solving and certification attracts more and more researchers. On contrary to propositional satisfiability QBF algorithms are far less well understood. Prior to our work QBF certification picture was much incomplete. Semantic certificates mainly existed only for true QBFs, despite of the obvious symmetry when compared to their false counterparts. Nature of Q-resolution was unexplored, and its derivative proof systems (such as long-distance resolution for example) were not formalized. The link between semantic and syntactic certificates was missing as well. This work provides an extensive study of *certification in quantified decision procedures*. We study of Q-resolution and its derivatives, formalize long-distance resolution, and propose new, stronger, resolution-based proof systems. For semantic certificates, we introduce novel approaches for certificate generation from resolution proofs (and its derivatives), we show experimentally that our approach significantly outperforms existing direct methods. Further we also characterize and implement different optimization techniques to improve the extracted certificate quality. In this work we do not also limit ourselves to QBF and provide a certification study for dependency quantified formulas as well.

This work is organized as follows (in parentheses we specify our publications that formed a basis for the content of the corresponding chapter).

- In Chapter 2 we introduce necessary background for SAT and QBF. More precisely, Section 2.1 is devoted to Boolean algebra and various representations of a Boolean function. Section 2.2 introduces satisfiability testing problem (SAT), basic solving techniques, syntactic derivation rules and applications. In Section 2.3 we give a formal definition to quantified Boolean formulae (QBF), show QBF examples, solving techniques, and provide a rough introduction into QBF certification.
- Chapter 3 is devoted for study of syntactic QBF certificates (chapter extends [8]).
In Section 3.1 we review Q-resolution as a proof system over QBF language, give a formal proof for its soundness and completeness. Further we study how modern search-based QBF solvers construct Q-resolution proofs and discuss it on some examples. In Section 3.2 we show existing extensions of Q-resolution proof system, and complexity relations between them. In Section 3.3 we propose yet novel extensions of Q-resolution proof system, and discuss their superiority and limitations in details.
- In Chapter 4 we give a brief study of semantic QBF certificates (chapter extends [4], [5], [6] and [7]).
Section 4.1 revisits the definition of models and countermodels as semantic certificates for QBF, and overviews some of the existing methods for their direct construction. In Section 4.2 we establish the connection between semantic and syntactic certificates, derive conversion algorithms and do extensive experimental evaluation. In Section 4.3 we show various optimizations to proposed in Section 4.2 algorithms. In Section 4.4 we extend conversion algorithms to derivatives of resolution proof system.
- Chapter 5 provides an extensive study of dependency quantified Boolean formulas (DQBF) (chapter extends [2] and [3]).
In Section 5.1 we give a formal introduction and necessary background for DQBFs. In Section 5.2 we provide a new semantic classification of DQBFs with examples. In Section 5.3 we show an extension of QBF syntactic certificates to DQBF and discuss its completeness and soundness issues.
- In Chapter 6 we conclude this work and outline open research questions for future work.



Chapter 2

Introduction to SAT and QBF


Laconically through this chapter we introduce basic concepts of Boolean algebra, go through various representations of a function over Boolean domain, and conclude by a brief formal introduction of satisfiability problem (SAT) and quantified Boolean formulae (QBF).

2.1 Boolean algebra

In the modern world Boolean algebra plays an important role not only because of numerous applications of formal logic to mathematics, but also due to its significance for theoretical computer science, and for the different aspects of integrated chips production. Boolean algebra represents a significant part of mathematics, in which (Boolean) variables are restricted to take only the truth values *true* or *false*, or equivalently denoted as 0 and 1 respectively.

Function in Boolean algebra is a subject to the Boolean domain, and ranged over truth values. Formally, Boolean n -ary function ϕ over variables x_1, \dots, x_n is a mapping $\phi : \{0, 1\}^n \rightarrow \{0, 1\}$, i.e. for each value $\{0, 1\}$ of each input, f either evaluates to 0 or to 1. There are many different ways to represent Boolean functions. Depending on the application some of them are more efficient than the others. All the representations might be subdivided into three groups: *expressions* (or *formulas*), *tables* and *graphs*. If there are no quantifiers present in a Boolean formula (i.e.

2.1. Boolean algebra



x	$\neg x$
0	1
1	0

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Figure 2.1: Truth tables representing key Boolean operations.

variables are free), and each assignment to its domain variables determines its unique value then it is said to be *propositional*. Each propositional Boolean formula can be split into a sequence of Boolean *operations*. In contrast to more commonly used in other algebras binary operations of addition and multiplication, Boolean algebra has *logic connectives*, e.g. binary operators *disjunction* (denoted as “ \vee ”), *conjunction* (denoted as “ \wedge ”), *implication* (denoted as “ \rightarrow ” or “ \leftarrow ”), and their derivatives. The only nonidentical Boolean unary operation is *negation* (denoted either as “ \neg ” or an overline). The rest of this section will be fully devoted to introduce various representations of Boolean functions together with the necessary definitions, and also discuss both their advantages and disadvantages. Also for convenience reasons through our work, when referring to a variable, function, formula, or a relation, we shall drop “Boolean”, and if not otherwise stated in the context, consider them within the Boolean algebra domain.

2.1.1 Truth Tables

Truth table is a natural representation of a Boolean function. Each cell of a truth table simply represents the value taken by the function with respect to the given assignment of its input variables. Truth tables for the key Boolean unary and binary operations are depicted in Figure 2.1.

Please note, that truth tables form a *canonical* representation of Boolean functions, meaning that the set of existing Boolean functions is isomorphic to the set of existing truth tables. Also note that for n -ary function there are 2^n entries to its corresponding truth table, since each variable can be assigned either true or false. Therefore there are exactly 2^{2^n} distinct truth tables representing n -ary Boolean functions and consequently 2^{2^n} distinct n -ary Boolean functions.

2.1. Boolean algebra

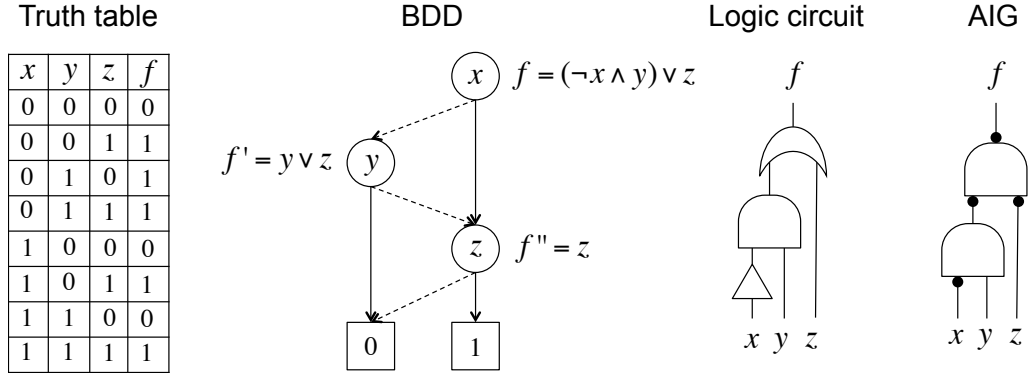


Figure 2.2: Various representations of a Boolean function.

On Figure 2.2 we show different representations of the same Boolean function, where left-most is its truth table representation.

2.1.2 And Inverter Graphs (AIGs)

To represent a Boolean function one may use elementary logic operations (or *logic gates*), to form a corresponding *logical expression* (or a *logic circuit*). The most commonly used logic gates (or sometimes referred to as *elementary gates*, or *logic connectives*) are NOT gate (\neg), AND gate (\wedge), OR gate (\vee), implication (\rightarrow), exclusive OR gate (XOR), etc. Logic circuit is not a canonical representation of a Boolean function. Given a Boolean function it has dozens of circuit representations. For a given circuit, the number of logic gates in it is called the *circuit size*, and the number of gates on the longest path from any input to the output is called the *logic depth* of a circuit. Since the number of transistors used to produce a particular design and its delay have positive correlations with its circuit size and circuit depth, respectively, designers prefer shallower and smaller circuit representations of the desired function.

And Inverter Graph (AIG) is a special kind of logic circuits containing only AND gates and NOT gates. Any Boolean function can be represented as AIG since (two input) AND and NOT gates form a functionally complete basis. Furthermore, it takes linear time to convert any logic circuit into AIG. NOT gates are usually not counted when computing the circuit size or logic depth for AIGs.

AIGs are widely used in modern Boolean logic. Convenience comes from the small

2.1. Boolean algebra

number of elementary logic gates (namely AND and NOT gates) used in the basis of the representation. This allows to effectively perform some structural rearrangements of the circuit to, for example, optimize its size.

Two right-most parts of Figure 2.2 show examples of a function, represented as logic circuit (in our case consisting of AND, OR and NOT gates) and AIG. Rounded rectangles depict AND gates, moon-shaped object is an OR gate, and triangle and small black circles represent NOT gates for general circuit and for AIG respectively.

2.1.3 Binary Decision Diagrams (BDDs)

Logic circuits and truth tables are intuitive ways to represent Boolean functions, however in practice former tend to have large size, and latter do not allow to find some functional properties of the design quickly. *Binary Decision Diagrams* were introduced to perform quick functional operations on functions with relatively small number of variables. BDDs scale much better than truth tables.

Given a Boolean function $\phi(x_1, \dots, x_n)$, it could be decomposed in the following way: $\phi = x_1 \wedge \phi(1, x_2, \dots, x_n) \vee \neg x_1 \wedge \phi(0, x_2, \dots, x_n)$, This decomposition is called *Shannon expansion*. $\phi(1, x_2, \dots, x_n)$ (alternatively denoted $\phi|_{x_1}$, ϕ_{x_1} or $\phi_{\lceil x_1}$) and $\phi(0, x_2, \dots, x_n)$ (alternatively denoted $\phi|_{\overline{x_1}}$, $\phi_{\overline{x_1}}$ or $\phi_{\lceil \overline{x_1}}$) are called a *positive cofactor* and a *negative cofactor* of ϕ respectively. Exactly the same definition of cofactors applies to Boolean formulas as well.

BDD is a directed acyclic graph (DAG), consisting of one source vertex (node), internal decision vertexes (nodes), and two terminal vertexes (nodes), namely 0-node and 1-node. Each non-terminal node is associated with a Boolean variable and a function. Each non-terminal node also has exactly two outgoing edges, namely *positive* and *negative* edges. Source node corresponds to the function itself, and each internal node corresponds to the cofactor of the function corresponding to some of its parent nodes with respect to the associated with this parent node variable (if connecting edge is positive or negative then the cofactor is positive or negative respectively). On each path from the source to either of terminal nodes each variable may be found only once.

Example of BDD can be found to the right of the truth table in Figure 2.2. As it was mentioned BDDs allow to perform logic operations and some optimizations



quickly. The only drawback is that BDDs are still far less scalable than AIGs.

2.1.4 Conjunctive and Disjunctive Normal Forms (CNF and DNF)

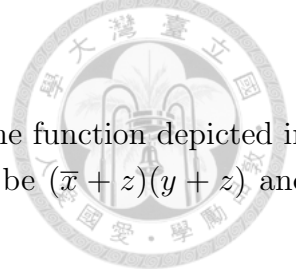
Each Boolean function can be characterized by its *onset* (i.e. assignments where function evaluates to 1) or its *offset* (i.e. assignments where function evaluates to 0). Special syntactic forms were introduced to efficiently elaborate these representations.

A *literal* in a Boolean formula is either a variable (i.e., positive-phase literal) or the negation of the variable (i.e., negative-phase literal). We denote the corresponding variable of a literal l as $var(l)$. Conjunction of a set of literals we call as a *cube*, and disjunction of a set of literals - as a *clause*. In the sequel we may alternatively specify a clause or a cube by a set of literals. If the given Boolean function is represented as a conjunction of clauses, then we say it has a *Conjunctive Normal Form*. If it is represented as a disjunction of cubes, we say it has a *Disjunctive Normal Form*.

Any Boolean function can be represented as CNF (the same holds for DNF). CNF is particularly important for Boolean logic, since its structural properties allow efficient SAT-solving algorithms to be implemented. As we shall see in Chapter 3, CNFs are crucial for effective learning during SAT solving, and also for other optimizations. Unfortunately, the worst-case time complexity for converting any given logic circuit into functionally equivalent CNF is exponential. For SAT-solving, however, as we shall also see there are several linear time complexity satisfiability (rather than functionality) preserving translations.

In the context of CNF and DNF formulas, for convenience, we might use symbol “+” instead of disjunction “ \vee ”, and omit the conjunction “ \wedge ”. If there is no ambiguity in the context we might also alternatively use “,” to separate literals both inside of a clause or cube. For example clause consisting of literals a , \bar{b} and c might interchangeably be written as $(a \vee \bar{b} \vee c)$, or $(a + \bar{b} + c)$, or (a, \bar{b}, c) . From time to time we will also use either “ \emptyset ”, or “ \square ”, or upside-down capital letter “ \top ” to denote the empty clause (i.e. false clause without any literals). To denote an empty (true) cube we will use normal capital letter “ T ”. Note that the clause containing exactly one literal is called a *unit* clause.

2.2. Satisfiability problem (SAT)



CNF and DNF representations are not canonical. For the function depicted in Figure 2.2, possible CNF and DNF representations might be $(\bar{x} + z)(y + z)$ and $\bar{x}y + z$ respectively.

2.2 Satisfiability problem (SAT)

As it was mentioned before, Boolean satisfiability (SAT) problem was introduced more than a half century ago. It was proven to be NP-complete by Cook in 1971 [12], and is one of the first problems known to be NP-complete. Nowadays SAT has not only the theoretical value (due to its direct connection to one of the most important problems in theoretical computer science, namely the “P versus NP” problem) but also due to its numerous applications in our modern digital world. Applications of satisfiability theory go far beyond its original domain, and include such major parts of mathematics as logic, graph theory, theoretical computer science, and others. A wide range of combinatorial problems can be reduced to satisfiability [12].

For the past decade a tremendous improvement was achieved in practicality of SAT solving. Most of the SAT approaches nowadays are optimized for the Boolean formulas given in CNF [12]. As we shall see in the next few subsections CNF offers a lot of structural conveniences that help much during the solving process. In this chapter we will go through basic SAT-solving algorithms, together with important for efficient solving heuristics. The resolution proof system introduced in this section will form a basis for our further discussion in the context of QBF language.

2.2.1 Preliminaries

Naturally we define truth *assignment* as a set of literals, but for convenience interchangeably might use it as a conjunction of literals. A (quantifier-free) formula ϕ over variables X subject to some truth assignment $\alpha : X' \rightarrow \{0, 1\}$ on variables $X' \subseteq X$ is denoted as $\phi|_\alpha$. $\phi|_\alpha$ is computed iteratively using Shannon expansion: $\phi_\alpha = (\phi_l)_{\{\alpha/l\}}$, where $l \in \alpha$. If α is a full assignment to variables of f , then ϕ_α is equal to either true or false.

Given a Boolean formula (or equivalently a logic circuit) ϕ over variables x_1, \dots, x_n ,

2.2. Satisfiability problem (SAT)



Boolean satisfiability problem asks whether there exists an assignment α to variables x_1, \dots, x_n such that ϕ_α evaluates to true.

Numerous SAT solvers have been introduced to the date. Some of the solvers are designed to work on logic circuits, but the absolute majority works on Boolean formulas expressed in CNF. CNF allows the use of effective data structures, and offers easier learning mechanism. In this work we will focus on the CNF-based SAT-solving algorithms.

Resolution is a basic derivation rule on CNF formulas. It works on two clauses that both contain (at least one) variable represented in opposite polarities. Formally, given two clauses C_1 and C_2 , and a *pivot variable* p with $p \in C_1$ and $\bar{p} \in C_2$, resolution produces clause

$$\text{resolve}(C_1, p, C_2) = C_1 \setminus \{p\} \cup C_2 \setminus \{\bar{p}\}.$$

Well known *modus ponens* rule can be seen as a special case of resolution of a one-literal clause and a two-literal clause.

Easy to see that resolvent is always implied from conjunction of its parents, i.e. $(C_1 \wedge C_2) \rightarrow \text{resolve}(C_1, p, C_2)$. Therefore if both C_1 and C_2 belong to CNF ϕ , then $\phi \wedge \text{resolve}(C_1, p, C_2)$ is functionally equivalent to ϕ , i.e. resolution offers a derivation mechanism which preserves the CNF structure of the formula.

Resolution is particularly important for SAT, as it is also a complete rule. Given an unsatisfiable CNF ϕ , a *resolution proof* Π of its unsatisfiability is a sequence of clauses C_i with $i \in [1..n]$, such that either $C_i \in \phi$ or exist such $\{j, k\} \in [1..i-1]$ that $C_i = \text{resolve}(C_j, p, C_k)$. Furthermore we require $C_n = \square$, and say that n is a *size* of the resolution proof. Some authors do not include initial clauses in the clause count for the size of the proof. Resolution proof is also often referred to as a *refutation*.

In practice resolution proof Π is usually represented as a *resolution graph*, denoted G_Π . Formally G_Π is a directed acyclic graph (DAG) such that each vertex in it corresponds to a clause in Π , and the edges show that a child vertex is a resolvent of its parent vertexes. Following is an example of a resolution proof.

Example 2.1 Given an unsatisfiable CNF $\phi = (a + b)_1(\bar{a} + b)_2(\bar{b} + c)_3(\bar{b} + \bar{c})_4$ over

2.2. Satisfiability problem (SAT)

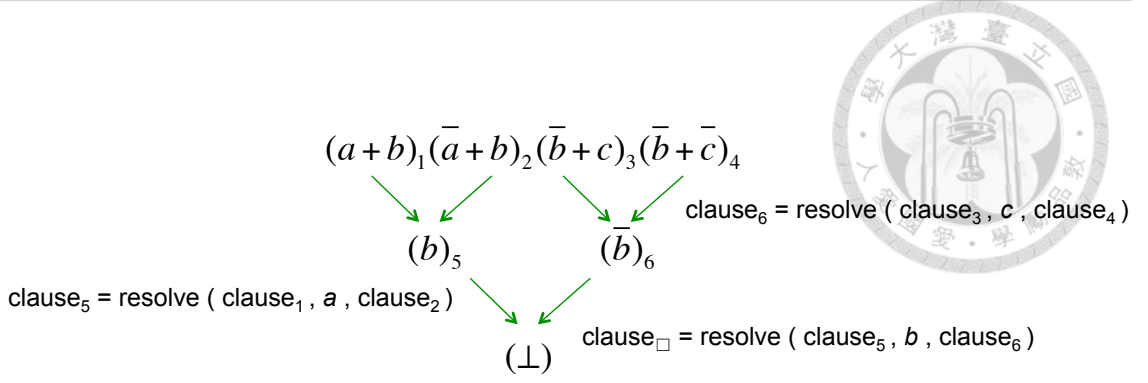


Figure 2.3: Truth table of QBF Φ .

variables a, b, c , consider the following resolution proof:

$$\Pi = \left\{ \begin{array}{l} 1. \text{ clause}_5 = \text{resolve}(\text{clause}_1, a, \text{clause}_2) = (b) \\ 2. \text{ clause}_6 = \text{resolve}(\text{clause}_3, c, \text{clause}_4) = (\bar{b}) \\ 4. \text{ clause}_\emptyset = \text{resolve}(\text{clause}_5, b, \text{clause}_6) = \square \end{array} \right\}$$

The corresponding resolution graph is depicted in Figure 2.3

Resolution alone forms a sound and complete proof system for SAT language, as the following theorem states.

Theorem 2.1 Given a CNF formula ϕ over variables x_1, \dots, x_n , it is unsatisfiable if and only if there exists a resolution proof Π proving the empty clause.

Despite its strengths, resolution has been proven to be *intractable*, i.e. there exist CNF formulas that are hard to refute using resolution rule alone (minimal resolution proofs are of exponential size with respect to the input formula size). First super-polynomial lower bound for resolution was proven by Armin Haken in 1984, and used the so-called “pigeon-hole” formulas designed by Cook and Reckhov in 1979. Pigeon-hole formulas simply encode the well-known pigeon-hole principle into a propositional CNF formula. Generally there are a lot of combinatorial problems that are known to be hard for resolution.

2.2.2 Satisfiability equivalence and Tseitin encoding

Now, as was mentioned in Section 2.1, it might be expensive to convert a logic circuit (or general Boolean formula) into a functionally equivalent CNF. It is however possible to translate the given logic circuit (or general Boolean formula) into a

2.2. Satisfiability problem (SAT)

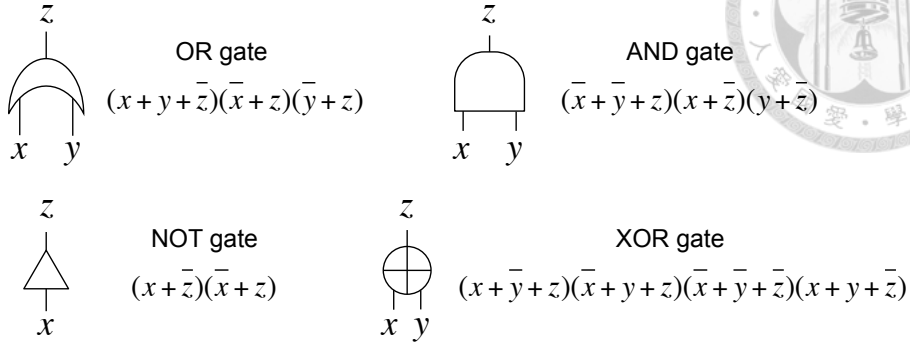


Figure 2.4: Tseitin encoding of basic logic gates.

satisfiability equivalent CNF formula. Two representations are said to be satisfiability equivalent if the functions they represent either both are SAT or both are UNSAT. One notable fact regarding satisfiability equivalence is that any CNF formula can be transformed into equisatisfiable formula that has at most 3 literals in each clause (3CNF). Furthermore it can be done in linear time with respect to the original CNF.

Another particularly interesting fact concerns resolution. Given any CNF formula ϕ and a variable x , one can rewrite the following satisfiability equivalent to ϕ formula $\phi' = \phi|_x \vee \phi|_{\bar{x}}$. This formula is no longer expressed in CNF, however curiously it is functionally equivalent to the formula $\phi \setminus \{C \in \phi | x \in \text{vars}(C)\} \cup \bigcup (\text{resolve}(C_1, x, C_2))$, i.e. formula where all possible resolvents with pivot x were added to ϕ , and then all clauses containing variable x removed from the resulting CNF. This process is called *variable elimination*. We will talk about it in more details when we discuss heuristics for SAT solving in the next few subsections.

The translation encoding of any logic circuit into satisfiability equivalent CNF was first proposed by Tseitin in 1983. Proposed encoding introduces a fresh variable for each gate and builds constraints that encode the relation imposed by the gate type on its input and output variables. The corresponding clause constraints for each elementary gate are shown in Figure 2.4.

Some improvements can be done to Tseitin transformation to get smaller CNF encodings, they however are not essential for our work. As we shall see in Section 2.3 Tseitin transformation rules can be applied in the context of QBF as well.



2.2.3 Solving SAT

In theory it is possible to eliminate all the variables by resolving them out (as was shown in previous subsection), and if at some point empty clause is produced then formula is unsatisfiable (otherwise it is satisfiable). This “symbolic” method, however, is impractical since it requires worst-case both exponential time and space. Experimentally it was proven that the best performance in SAT solving is shown by *search-based* solving algorithms.

The most naive searching approach to determine satisfiability of a formula ϕ - is to go over all possible truth assignments to its variables x_1, \dots, x_n . If under one of the assignments formula evaluates to 1, then ϕ is satisfiable (SAT). If we explored all the possible assignments and did not find a satisfying assignment then the formula is unsatisfiable (UNSAT). This simple search-based approach has a sophisticated implementation, called DPLL after Davis, Putnam, Logemann and Loveland who introduced it in 1962. Instead of choosing a complete assignment and checking if formula evaluates to true or not, DPLL assigns variables one by one in depth-first search manner. Since CNF has very special form, if under a given (partial) assignment clause has been evaluated to false, then the whole CNF evaluates to false, and there is no need to assign the rest of the variables. This “skipping” allows us to shrink the search-space very much. DPLL algorithm is depicted in Figure 2.5.

DPLL algorithm is complete and works in worst-case exponential, relative to the input formula size, time complexity. It is also sometimes called as a *core* algorithm for SAT solving, as it is used in almost every SAT solver nowadays [12].

The following example illustrates how DPLL algorithm works.

Example 2.2 Consider an unsatisfiable CNF $\phi = (a + b)_1(\bar{a} + b)_2(\bar{b} + c)_3(\bar{b} + \bar{c})_4$ from the previous subsection. DPLL search tree is shown in Figure 2.6. Note that simply trying all possible assignments to variables a , b and c would require to search through $2^3 = 8$ assignments, whereas DPLL allowed us to examine only 6 leafs of the search tree. Therefore we shrank the search space and concluded unsatisfiability of ϕ faster.

Note that the core DPLL algorithm can be applied to any Boolean function representation and is not particularly restricted to CNF formulas. CNF, however,

2.2. Satisfiability problem (SAT)



$DPLL(\phi, \alpha)$

```

input: CNF  $\phi$ , assignment  $\alpha$ 
output: FALSE or an extended SAT assignment
01  $x := \text{PickUnassignedVar}(\text{vars}(\phi), \alpha);$ 
02 if  $\phi|_{\alpha \wedge \bar{x}} = \emptyset$  then return  $\alpha \wedge \bar{x};$ 
03 if  $\square \notin \phi|_{\alpha \wedge \bar{x}}$ 
04    $\beta := DPLL(\phi, \alpha \wedge \bar{x});$ 
05   if  $(\beta)$  then return  $\beta;$ 
06 if  $\phi|_{\alpha \wedge x} = \emptyset$  then return  $\alpha \wedge x;$ 
07 if  $\square \notin \phi|_{\alpha \wedge x}$ 
08    $\beta := DPLL(\phi, \alpha \wedge x);$ 
09   if  $(\beta)$  then return  $\beta;$ 
10 return FALSE;
end
  
```

Figure 2.5: Core DPLL algorithm for SAT solving.

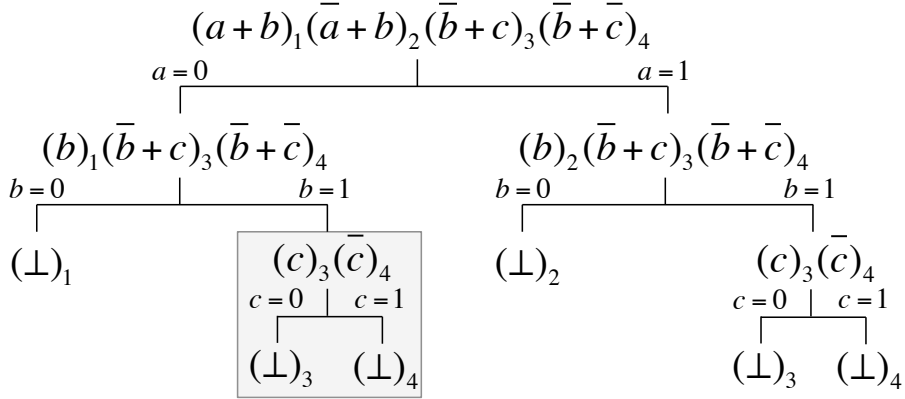


Figure 2.6: DPLL search tree for ϕ of Example 2.2.

offers a convenient way to perform *learning*. The so-called conflict-driven constraint learning algorithm (CDCL) uses DPLL as a search engine, but prunes the visited search space using conflict clauses. For example, while applying DPLL to formula ϕ from Example 2.2, we notice that the right-most branch of the search tree in fact is independent of a , and therefore we can learn clause (\bar{b}) directly, and do not go into the search branch located in the shaded square in Figure 2.6. In CDCL resolution plays the key role, as learned clauses are usually constructed by resolution from the conflict clause and some other clauses (for example from clauses responsible for heuristic assignment of some variables, e.g. *unit propagation*). In next subsection we will talk more about heuristics used in majority of the modern SAT solving engines.



2.2.4 Heuristics in SAT solving

Empirically, core CDCL algorithm was good enough for CNF formulas with thousands of variables. However in modern verification problems CNF formulas can reach hundreds of thousands of variables and millions of clauses. Since there is no polynomial algorithm known for SAT, it is necessary to utilize as many heuristics to prune the search space or simplify the formula as we can. Unit propagation or Boolean constraint propagation (BCP) is one of the major heuristics used in SAT solving. BCP has a very simple idea behind it. Whenever there is exactly one unassigned literal in the clause left (i.e. it is an effectively unit clause) - it has to be assigned true right away. This simple trick has tremendous impact on the SAT solving time [12].

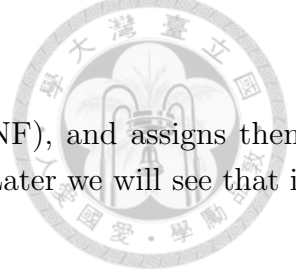
From Section 2.2 we have seen that if the formula was encoded into CNF from a logic circuit - most of its clauses would have two or three literals. Therefore during the search it in fact happens quite often that a lot of variables can be directly assigned by BCP (assigning one variable can further force other variables to be assigned by BCP as well).

One notable trick to perform BCP efficiently is the so-called *two-literal watch scheme* proposed by Moskewicz, Madigan, Zhao, Zhang and Malik in 2001, as one of the heuristics used in their SAT solver CHAFF [12].

Other major heuristics for SAT solving include different methods for choosing the next branch variable (for example based on variable occurrence frequency), learnt clauses deletion strategies (for example if learned clauses are too large they might not be useful for search space trimming).

As a separate engine, before the solving itself, many tools call the so-called *preprocessors*. Preprocessors use resolution or other techniques to simplify the original CNF formulas as much as they can before going into the actual solving. The most well known preprocessing technique (that was already mentioned previously) is variable elimination. Certain constraints are imposed on the candidate variables that one wants to eliminate. For example we do not eliminate variable if it produces too many resolvents, or produces particularly long resolvents. In general variable elimination is proven to be extremely efficient in the context of SAT solving. A special case of variable elimination, called *pure literal elimination* detects *pure literals*

2.2. Satisfiability problem (SAT)



(i.e. literals that only occur in one particular phase in CNF), and assigns them directly without waiting until their branching time comes. Later we will see that it is useful for QBFs too.

2.2.5 SAT applications

In this subsection we briefly outline the most interesting applications that use SAT-solvers either purely or incrementally [12]. We try to outline both applications of satisfiability theory in combinatorial mathematics, as well as in formal verification and synthesis.

- **Combinational equivalence checking**

Given two logic circuits ϕ and ϕ' over variables x_1, \dots, x_n and x'_1, \dots, x'_n respectively, combinational equivalence checking problem asks whether ϕ and ϕ' are functionally equivalent. This problem has direct SAT translation. We first define function $\phi_{XOR} = (\phi \oplus \phi') \wedge (x_1 \leftrightarrow x'_1) \wedge \dots \wedge (x_n \leftrightarrow x'_n)$. Then translate ϕ_{XOR} to CNF formula (for example by Tseitin transformation), and ask SAT-solver whether ϕ_{XOR} is satisfiable or not. If ϕ_{XOR} is unsatisfiable then circuits ϕ and ϕ' are functionally equivalent. Combinational equivalence is extensively used in formal verification, for example to verify that optimized logic circuit is equivalent to the original design.

- **Logic Optimization**

Given a logic circuit ϕ and two inner gates g_1 and g_2 with, say, outputs o_1 and o_2 , one could perform combinational equivalence checking on subcircuits representing logics of o_1 and o_2 . If o_1 and o_2 are functionally equivalent then gate g_2 could be eliminated from the circuit, and its output o_2 could be simply merged with o_1 , therefore simplifying the circuit ϕ .

- **Bounded model checking (BMC)**

Consider a transition system over state variables \vec{x} with transition relation $R(x, x')$ (x' is a set of next state variables), and sets $I(x)$ and $B(x)$ of initial and bad states respectively. Given a positive integer k , BMC problem asks whether there exists a k steps path from I to B in the system's state transition graph (i.e. graph representation of the transition relation R). BMC problem can be translated into satisfiability problem as follows. First construct formula

2.3. Quantified Boolean Formulae (QBF)

$\phi_k(x_0, \dots, x_k) = I(x_0) \wedge R(x_0, x_1) \wedge R(x_1, x_2) \wedge \dots \wedge R(x_{k-1}, x_k) \wedge B(x_k)$. Now if ϕ_k is unsatisfiable then B is unreachable from I in k steps. BMC is a very useful technique in general model checking, which is one of the main tasks in modern formal verification. In practice engineers start with $k = 1$ and incrementally solve satisfiability of ϕ_1, ϕ_2 , and so on, either until they could find a satisfiable ϕ_k indicating that bad state is reachable, or until ϕ_k becomes too hard for any of available SAT algorithms.

- ***Combinatorial problems***

As it was mentioned, many combinatorial problems can be reduced to propositional satisfiability. Examples of such problems include (but are not limited to) various graph coloring problems, knapsack problem, variations of bin packing problem, etc.

2.3 Quantified Boolean Formulae (QBF)

Last Section 2.2 was devoted to introduction of Boolean satisfiability testing problem, and its numerous industrial and combinatorial applications. Furthermore SAT reached a great success in terms of solving improvement through the past decade. Unfortunately propositional satisfiability has its limitations. With the recent rapid growth of the designs some of the formal verification and synthesis problems go beyond its reach. Hard combinatorial problems confirm this issue too. Examples of SAT disadvantages include inability to work with quantifiers, large formula size after translation of the original problem to SAT, poor support for functional synthesis problems, and limitations imposed by its NP complexity class.

Satisfiability of quantified Boolean formulas (QSAT) belongs to PSPACE-complete complexity class, and therefore potentially offers greater expressive power. Quantifiers natively exist in QBF, and as we shall see in the next few subsections some functional synthesis problems have natural encoding into QBF. QBF also offers more succinct encoding for some problems compare to SAT.

QBF is a relatively new research area, although a lot of optimizations for solving and certifying QBF problems has been proposed to the date. Many optimizations are borrowed from SAT solving techniques, but some are unique to QBF. In this section we briefly go through the necessary background for quantified Boolean formulas,

2.3. Quantified Boolean Formulae (QBF)



describe some of the most effective QBF solving techniques and give a little of certification preliminaries.

2.3.1 Preliminaries

Original definitions of variables, literals, clauses, etc., defined for propositional satisfiability, stay intact. Now, given a set V of Boolean variables, a set $L = V \cup \{\bar{v} \mid v \in V\}$ of positive and negative literals over V , and the existential (\exists) and universal (\forall) quantifiers, a *quantified Boolean formula* (QBF) $\mathcal{P}.\phi$ in *prenex conjunctive normal form* (PCNF) consists of the prefix $\mathcal{P} = Q_1v_1 \dots Q_kv_k$ with $Q_i \in \{\exists, \forall\}$, $v_i \in V$, and $v_i \neq v_j$ if $i \neq j$ for $i, j \in [1..k]$, and the CNF matrix $\phi \subset 2^L$. All QBFs in this work are assumed to be in PCNF and to be *closed*, i.e. all literals in the matrix are quantified in the prefix and matrix itself is free of tautological clauses.

To date CNF is a prioritized form for the matrix representation due to the same reasons as it was for SAT. Recently there was some research done towards circuit-based QBF solvers, which shows benefits of using circuit information during solving for certain class of benchmarks, however it is out of scope of this work.

For each variable $v_i \in V$, its *quantifier level* $\text{lev}(v_i)$ is the number of alternations between \exists and \forall quantifiers from Q_1 to Q_i . We apply this definition also to literals, i.e., $\text{lev}(l) = \text{lev}(\text{var}(l))$. The *quantifier index* of v_i is $\text{idx}(v_i) = i$. Similarly, for literal l , $\text{idx}(l) = \text{idx}(\text{var}(l))$. The set V of variables is partitioned into the set $V_\exists = \{v_i \in V \mid Q_i = \exists\}$ of *existential variables* and the set $V_\forall = \{v_i \in V \mid Q_i = \forall\}$ of *universal variables*. We use letters from the beginning of the Latin alphabet for existential variables/literals, letters from the end for universal variables/literals, and v for either.

For example consider formula

$$\Phi = \exists a \exists b \forall x \exists c \forall y. (a + b + x + y)(\bar{b} + \bar{x} + \bar{c} + \bar{y})(\bar{a} + \bar{x})(\bar{b} + x + c).$$

CNF $\phi = (a + b + x + y)(\bar{b} + \bar{x} + \bar{c} + \bar{y})(\bar{a} + \bar{x})(\bar{b} + x + c)$ is a matrix of Φ . Please note that there is no need to separate variables a and b in the prefix as they are adjacent and have the same quantifier. Therefore for convenience we just write $\Phi = \exists ab \forall x \exists c \forall y. \phi$. Also note that QBF definition allows us to rewrite QBF Φ iteratively as $\Phi = \exists a \Phi'$,

2.3. Quantified Boolean Formulae (QBF)

$x \backslash ab$	00	01	10	11																																				
0	<table><tr><td>$c \backslash y$</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr></table>	$c \backslash y$	0	1	0	0	1	1	0	1	<table><tr><td>$c \backslash y$</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	$c \backslash y$	0	1	0	0	0	1	1	1	<table><tr><td>$c \backslash y$</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	$c \backslash y$	0	1	0	1	1	1	1	1	<table><tr><td>$c \backslash y$</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	$c \backslash y$	0	1	0	0	0	1	1	1
$c \backslash y$	0	1																																						
0	0	1																																						
1	0	1																																						
$c \backslash y$	0	1																																						
0	0	0																																						
1	1	1																																						
$c \backslash y$	0	1																																						
0	1	1																																						
1	1	1																																						
$c \backslash y$	0	1																																						
0	0	0																																						
1	1	1																																						
1	<table><tr><td>$c \backslash y$</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	$c \backslash y$	0	1	0	1	1	1	1	1	<table><tr><td>$c \backslash y$</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	$c \backslash y$	0	1	0	1	1	1	1	0	<table><tr><td>$c \backslash y$</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr></table>	$c \backslash y$	0	1	0	0	0	1	0	0	<table><tr><td>$c \backslash y$</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr></table>	$c \backslash y$	0	1	0	0	0	1	0	0
$c \backslash y$	0	1																																						
0	1	1																																						
1	1	1																																						
$c \backslash y$	0	1																																						
0	1	1																																						
1	1	0																																						
$c \backslash y$	0	1																																						
0	0	0																																						
1	0	0																																						
$c \backslash y$	0	1																																						
0	0	0																																						
1	0	0																																						

Figure 2.7: Truth table of QBF Φ .

where $\Phi' = \exists b \forall x \exists c \forall y. \phi$. In our example variable a is a free variable relatively to Φ' as its value will be determined before actually going into Φ' . Below is the formal definition of this phenomena.

A (partial) *assignment* to a QBF $\Phi = \mathcal{P}. \phi$ over the set L of literals, similar to propositional definition, is a set $\sigma \subset L$ where it holds that if $l \in \sigma$ then $\bar{l} \notin \sigma$. The *assignment condition* $\text{cond}(\sigma)$ is the conjunction $(\bigwedge_{l \in \sigma} l)$ of literals in σ . Recall that a clause C is evaluated under assignment σ to $C_{\upharpoonright \sigma}$ such that $C_{\upharpoonright \sigma} = \top$ if $C \cap \sigma \neq \emptyset$, $C_{\upharpoonright \sigma} = \perp$ if $C \setminus \{v \mid \bar{v} \in \sigma\} = \emptyset$, and $C_{\upharpoonright \sigma} = C \setminus \{v \mid \bar{v} \in \sigma\}$ otherwise. A QBF Φ is evaluated under an assignment σ to $\Phi_{\upharpoonright \sigma}$ by replacing each $C \in \phi$ by $C_{\upharpoonright \sigma}$. The QBF $\forall x \mathcal{P}. \phi$ is true if and only if $\mathcal{P}. \phi_{\upharpoonright \{x\}}$ and $\mathcal{P}. \phi_{\upharpoonright \{\bar{x}\}}$ are true. The QBF $\exists e \mathcal{P}. \phi$ is true if and only if $\mathcal{P}. \phi_{\upharpoonright \{e\}}$ or $\mathcal{P}. \phi_{\upharpoonright \{\bar{e}\}}$ is true.

Let us examine if formula

$\Phi = \exists ab \forall x \exists c \forall y. \phi$, with

$$\phi = (a + b + x + y)(\bar{b} + \bar{x} + \bar{c} + \bar{y})(\bar{a} + \bar{x})(\bar{b} + x + c)$$

is true or false using the above definition. In Figure 2.7 we have a conveniently rewritten truth table of the matrix of Φ . Shaded rectangles show a *winning region* for existential variables. Taking a closer look to the shaded region of the truth table that corresponds to QBF $\Phi' = \forall x \exists c \forall y. \phi|_{\{a=0, b=1\}}$ we can see that for both choices of assignment to x there is a row in a small square-table with “1” in each column. Formally, both QBFs $\exists c \forall y \phi|_{\{a=0, b=1, x=0\}}$ and $\exists c \forall y \phi|_{\{a=0, b=1, x=1\}}$ are true. Therefore Φ' is true and hence Φ is also true.

In the next subsection we will go deeper into QBF semantics, and give a more

2.3. Quantified Boolean Formulae (QBF)



intuitive explanation to the above example.

2.3.2 QBF semantics

Validly any QBF formula can be viewed as a two player game. Existential (“ \exists ”) player assigns existential variables and tends to satisfy the matrix, while universal (“ \forall ”) player assigns universal variables and tends to falsify the matrix. Both players take turns according to the prefix of a given quantified Boolean formula.

As this “game” point of view suggests, one possible way to determine QBF’s answer is to look for a winning strategy for one of the players. Formally, winning strategy is a set of functions, which, depending on all the opponent’s previous moves, determines the player’s move, and eventually guarantees the player’s victory.

Definition 2.1 Let $\Phi = \dots \exists x_i \dots \forall y_j \dots [\phi(\vec{x}, \vec{y})]$ be a *true* quantified Boolean formula, over existential variables \vec{x} , and universal variables \vec{y} . A *model* (or a set of *Skolem* functions) for Φ is a mapping of each *existentially* quantified variable x_i with a function F_i (called the *Skolem-function* [39] of the corresponding variable), depending only on *universal* variables y_j with $\text{lev}(y_j) < \text{lev}(x_i)$, such that substituting each existential x_i with F_i in ϕ forces ϕ to become *tautological*, i.e. it evaluates to *true* independent of the assignment to \vec{y} .

In the same way we could define the winning strategy for the universal player in case of false QBFs.

Definition 2.2 Let $\Phi = \dots \exists x_i \dots \forall y_j \dots [\phi(\vec{x}, \vec{y})]$ be a *false* QBF, over existential variables \vec{x} , and universal variables \vec{y} . A *countermodel* (or a set of *Herbrand* functions) of Φ is a mapping of each *universally* quantified variable y_j with a function H_j (called the *Herbrand-function* of the corresponding variable), depending only on *existential* variables x_i with $\text{lev}(x_i) < \text{lev}(y_j)$, such that substituting y_j with H_j in ϕ forces ϕ to become *unsatisfiable*, i.e. it evaluates to *false* independent of the assignment of \vec{x} .

It should be mentioned that in most of the cases, a model (resp. countermodel) of a given QBF may not be unique and in fact there might exist many valid Skolem (resp. Herbrand) functions for each variable. Consider the formula Φ from previous

2.3. Quantified Boolean Formulae (QBF)



subsection:

$\Phi = \exists ab \forall x \exists c \forall y. \phi$, where

$$\phi = (a + b + x + y)(\bar{b} + \bar{x} + \bar{c} + \bar{y})(\bar{a} + \bar{x})(\bar{b} + x + c).$$

As we already established QBF Φ evaluates to true. We may witness this fact by providing a winning strategy for existential player: $\{F_a = 0 \wedge F_b = 1 \wedge F_c = \bar{x}\}$. To verify their correctness we evaluate the formula

$(F_a + F_b + x + y)(\bar{F}_b + \bar{x} + \bar{F}_c + \bar{y})(\bar{F}_a + \bar{x})(\bar{F}_b + x + F_c)$, which, as expected, is a tautology.

Please note that if we reorder quantifiers in the prefix of Φ to obtain QBF $\Phi' = \exists ab \exists c \forall x \forall y. \phi$ the functions we have found do not form a valid model any longer, since variable c now cannot be dependent on x . QBF Φ' is false in fact, which could be witnessed by countermodel $\{H_x = (a + b)(\bar{b} + c) \wedge H_y = (a + b)(\bar{b} + c)\}$ (observe that the Herbrand function H_x was not possible in Φ due to the different variable ordering in the prefix). Found countermodel is particularly interesting because it could be also expressed as $\{H_x = (a + b)(\bar{b} + c) \wedge H_y = H_x\}$. This property is also true in general: inner-more quantified universal (resp. existential) variables may be dependent on the outer-more quantified universal (resp. existential) variables. It is clear that allowing such a dependency does not bring any extra freedom for \forall -player (resp. \exists -player). These dependencies could be always eliminated by simple substitution (e.g. $H_y = H_x = (a + b)(\bar{b} + c)$), and therefore result into *logically equivalent* formula. The formal definition of logical equivalence is given below.

Definition 2.3 Two QBF formulas Φ_1 and Φ_2 over the same variables are said to be logically equivalent if $\Phi_1 = \Phi_2$ and the set of valid countermodels (resp. models) for Φ_1 is exactly the same as the set of valid countermodels (resp. models) for Φ_2 .

As propositional SAT is a special case of QBF, QBF semantics can be applied to propositional formulas too: Skolem-function model for the satisfiable propositional formula is a set of constant functions that satisfy the formula (i.e. a satisfiable assignment), and Herbrand-functions countermodel for unsatisfiable formula is empty set of functions (since we do not have any universal variables). Note that by definition verifying a QBF countermodel (resp. model) is equivalent to checking satisfiability (resp. unsatisfiability) of the substituted formula. In practice substitution itself is a relatively complicated process which can be avoided by the propositional Tseitin transformation. Let us recall false QBF $\Phi' = \exists ab \exists c \forall x \forall y. \phi$ that we used previously as our example. Instead of firstly substituting x and y variables with their respective

2.3. Quantified Boolean Formulae (QBF)

Herbrand functions in ϕ and then applying Tseitin transformation to get it into CNF form, we skip the former step and directly conjunct ϕ with constraints representing corresponding definitions. Resulting will be propositional formula $\phi \wedge (x \leftrightarrow H_x) \wedge (y \leftrightarrow H_y)$ (where equivalences would be translated into CNF). Note that this formula depends on all variables a, b, c, x, y , rather than just a, b and c as would be in case of actual substitution. New formula, however, is satisfiability equivalent to the original one but is much more convenient to work with.

Skolem model and Herbrand countermodel are also said to be *semantic certificates* for truth and falsity of QBF respectively. Next we are going to formalize Tseitin transformation over QBF language, and then look into Q-resolution which is a basic rule for the so-called *syntactic certificates* for QBF.

2.3.3 Tseitin transformation for QBF

Earlier in Section 2.2 it was mentioned that given a Boolean formula in circuit format it then could be translated into satisfiability equivalent CNF in time linear with respect to the original formula size. This procedure is called the Tseitin transformation and not surprisingly it could also be applied in the context of QBFs as Theorem 2.2 shows below.

Theorem 2.2 Consider a QBF $\Phi = \mathcal{P}.\phi$, where ϕ is expressed as a logic circuit (or alternatively as a non-cnf formula). Further, denote $\Phi_t = \mathcal{P}\exists t_1..t_n.\phi_t$, where ϕ_t is a Tseiting encoding of ϕ which introduces fresh (Tseitin) variables t_1, \dots, t_n . Then $\Phi = \Phi_t$, and furthermore Φ and Φ_t are logically equivalent with respect to (counter)models of variables from ϕ .

Proof Assume without loss of generality that $\Phi = 0$. Further consider logic gates g_1, \dots, g_n present in the circuit format of ϕ with their respective output variables t_1, \dots, t_n . Note that substitution of any countermodel of Φ into ϕ results into unsatisfiable formula by definition. Since ϕ_t is equisatisfiable to ϕ it also should be unsatisfiable under analogical substitution. The same argument works in the opposite direction too, therefore we conclude that $\Phi_t = 0$, and sets of countermodels of Φ and Φ_t are equivalent. ■

2.3. Quantified Boolean Formulae (QBF)

Theorem 2.2 intuitively states that QBF always can be represented in PCNF (although it loses circuit structure that might be useful in some special cases), by applying propositional Tseitin transformation to the matrix itself, and then quantifying all the fresh variables in the innermost scope. PCNF representation is further equivalent to the original formula in a sense that all the models or countermodels (depending if formula is true or false respectively) are preserved in the transformation. More elaborate placement of fresh variables is also possible, and might be useful for certain applications, it however is out of the scope for this work.

2.3.4 Q-resolution

Yet resolution was a complete proof system for propositional satisfiability, in QBF language its completeness requires an addition of a new syntactic rule. A clause C with literals $\{l_1, \dots, l_j\}$ in QBF Φ over variables X is called *minimal* if

$$\max_{l_i \in C, \text{var}(l_i) \in X_\forall} \{\ell(l_i)\} < \max_{l_i \in C} \{\ell(l_i)\}.$$

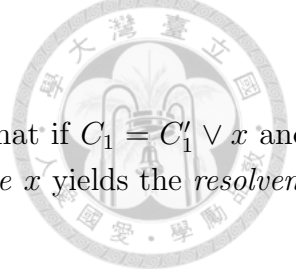
Otherwise, it is *non-minimal*. A non-minimal clause C can be minimized to a minimal clause C' by removing the literals

$$\{l \in C \mid \text{var}(l) \in X_\forall \text{ and } \ell(l) = \max_{l_i \in C} \{\ell(l_i)\}\}$$

from C . This process is called \forall -*reduction*. For any clause C we denote its \forall -reduced minimal clause as $\text{MIN}(C)$. As we shall see in Chapter 3 replacing C with $\text{MIN}(C)$ in a QBF does not change satisfiability of the formula. Consider the QBF Φ from the last subsection and take clause $(a + b + x + y)$ as an example. According to \forall -reduction rule minimal form of this clause is just $(a + b)$ since both literals x and y have greater quantification level than that of either of a and b .

A clause is *tautological* if it contains both literals x and $\neg x$ of some variable x . Two non-tautological clauses C_1 and C_2 are of *distance* k if there are k variables $\{x_1, \dots, x_k\}$ appearing in both clauses but with opposite phases. Resolution rule for QBF is defined similarly to propositional CNF formulas. However on contrary to SAT distinction has to be made for different resolution conditions, as in the future it will lead us to distinctive proof systems for QBF language. An *ordinary resolution*

2.3. Quantified Boolean Formulae (QBF)



is defined on two clauses C_1 and C_2 of distance 1. Recall that if $C_1 = C'_1 \vee x$ and $C_2 = C'_2 \vee \neg x$, then resolving C_1 and C_2 on the *pivot variable* x yields the *resolvent* $C'_1 \vee C'_2$.

Q-resolution [29] extends the ordinary resolution on CNF to PCNF formulae with two rules: First, only existential variables can be the pivot variables for resolution. Second, \forall -reduction is applied whenever possible. Unless otherwise said, “Q-resolution” is shortened to “resolution” in the sequel. In fact (Q-)resolution is a sound and complete approach to QBF evaluation as the following theorem states (we shall give a proof in Chapter 3).

Theorem 2.3 ([29]) A QBF is false (unsatisfiable) if and only if there exists a clause resolution sequence leading to an empty clause.

Restriction of (clause) resolution rule to be done only on existential pivots comes from a minimalistic desire to have as small set of derivation rules in the proof system as possible. Recently it however was proven that elimination of this restriction leads to interesting results regarding its complexity effectiveness. More details will be given in Section 3.1.

Resolution on clauses of distance greater than 1 is called *long-distance* resolution. It produces tautological clauses, and intuitively tautological resolvents are useless for propositional SAT. On contrary to propositional satisfiability, tautological clauses might be useful under certain conditions for QBF solving. More details will be given in Section 3.2 but for the moment we focus on the ordinary Q-resolution.

Although almost all of the discussions later would be dedicated to false QBFs in PCNF form and syntactic certificates in form of clause Q-resolution, we mention that cube resolution can be defined in a very similar manner. All the definitions (including pivot, resolvent, etc.) can be directly extended to work on cubes (recall that both cubes and clauses represent sets of literals). \forall -reduction in cube Q-resolution (or also called *Q-consensus*) is switched with \exists -reduction, resolution is allowed only on universal pivots, and empty clause (falsified) is substituted with an empty cube (tautologous). Please notice that cube resolution works on formulas expressed in DNF rather than CNF. Equisatisfiable DNF form can be obtained in exactly the same manner as CNF form by Tseitin encoding. Cube Q-resolution is also sound and complete for QBF evaluation.

2.3. Quantified Boolean Formulae (QBF)



Theorem 2.4 ([23]) A QBF is true (satisfiable) if and only if there exists a cube resolution sequence leading to an empty cube.

2.3.5 Solving QBF

On the same page with SAT, there are two groups of algorithms for QBF solving, namely symbolic and search-based. Symbolic methods involve Q-resolution and the so-called *skolemization*, which we will discuss more in Section 4.1. Search based methods include extension of (already known from propositional satisfiability) DPLL(CDCL) algorithm, and also unique to QBF *abstraction-refinement methods*. In this work we bias our focus towards search-based methods, as they experimentally were proven to be more robust compare to symbolic methods, and further, as we shall later see from Section 3.1, there is a direct correlation between QBF resolution-based syntactic certification and the search procedure itself. The core search algorithm (QDPLL) is shown in Figure 2.8. Although we will not go into the details of the learning mechanism for QBF (i.e. QCDCL), we mention that both clauses and cubes can be learned during the search.

Please note that algorithm QDPLL depicted in Figure 2.8 is somewhat similar to its propositional counterpart. If we apply it to a propositional formula, lines 6-8 and 13-15 would disappear, and what left is a simpler version of DPLL algorithm compare to what we had in Figure 2.5. The only difference is that current algorithm is not constructing a SAT-assignment, but simply searching for an answer. If universal variables are present in the formula (i.e. formula is not propositional), then search becomes more elaborate compare to SAT: finding a satisfying branch no longer represents an algorithm termination condition.

Algorithm on Figure 2.8 is complete and has worst case linear *space complexity* with respect to the input formula size. Therefore it serves as a proof that QBF \in PSPACE, as the following Theorem 2.5 states.

Theorem 2.5 Deciding QBF belongs to PSPACE.

Proof Given a QBF Φ , consider the search tree constructed by QDPLL algorithm depicted in Figure 2.8 while deciding Φ . Note that for each recursive call of the algorithm only the index and the value of the current branching variable have to

2.3. Quantified Boolean Formulae (QBF)



$QDPLL(\Phi)$

```

input: QBF  $\Phi = \mathcal{P}.\phi$ 
output: TRUE or FALSE
01   $x :=$  outer most variable in  $\mathcal{P}$ ;
02   $\Phi' := \mathcal{P} \setminus \{x\}.\phi|_{\bar{x}}$ ;
03  if  $x$  is existential
04    if  $\phi|_{\bar{x}} = \emptyset$  then return TRUE;
05    if  $QDPLL(\Phi')$  then return TRUE;
06  if  $x$  is universal
07    if  $\square \in \phi|_{\bar{x}}$  then return FALSE;
08    if  $\neg QDPLL(\Phi')$  then return FALSE;
09   $\Phi' := \mathcal{P} \setminus \{x\}.\phi|_x$ ;
10  if  $x$  is existential
11    if  $\phi|_x = \emptyset$  then return TRUE;
12    if  $QDPLL(\Phi')$  then return TRUE;
13  if  $x$  is universal
14    if  $\square \in \phi|_x$  then return FALSE;
15    if  $\neg QDPLL(\Phi')$  then return FALSE;
end

```

Figure 2.8: Core QDPLL algorithm for QBF solving.

be stored (Φ' in fact does not need to be explicitly computed in lines 2 and 9 in Figure 2.8). Therefore only constant additional space is required at each recursive call of QDPLL. Recursive depth of QDPLL can take the value of at most the number of variables in Φ , i.e. the whole decision procedure takes at most linear space with respect to the original formula size. ■

Modern search-based QBF solvers are also equipped with the unit propagation and pure-literal elimination heuristics (which work exactly the same as in propositional SAT), and with universal (\forall) reduction. Recall that \forall -reduction is applied to any clause whenever possible in Q-resolution. Similarly to propositional formulas variable x is called *pure* if it occurs only in positive or negative phase in CNF formula. Obviously if x is pure and existentially quantified then we can eliminate x together with all clauses where it occurs (since we can always choose such value of x to satisfy all the clauses it is present in). On the other hand if x is universally quantified then we can simply remove it from all the clauses (since we can always choose x 's value to make no clauses to be satisfied by it). Definition of *unit* variable slightly differs from that in SAT. A variable l is called *unit*, if it is existentially quantified, and there

2.3. Quantified Boolean Formulae (QBF)

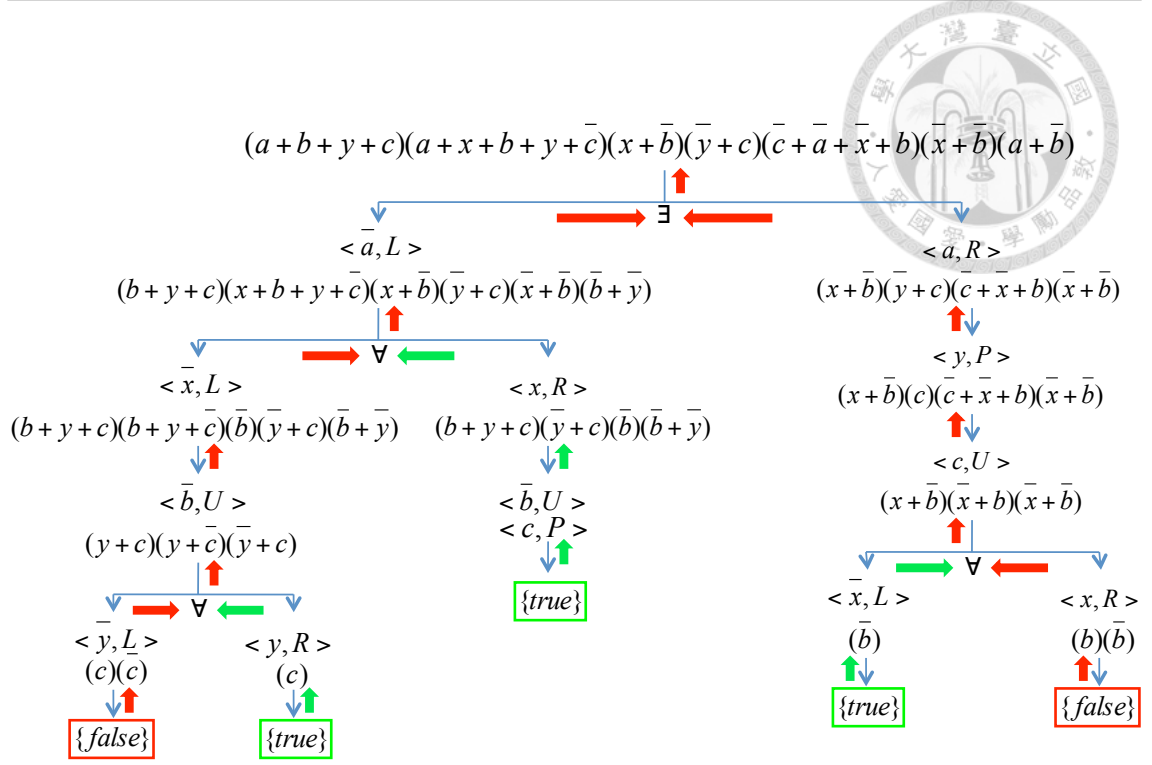


Figure 2.9: A complete search tree obtained by applying QDPLL procedure to Φ .

exists a clause containing l , such that all other literals l_i in the clause are universally quantified and have quantification level larger than that of l : $\forall i[\text{lev}(l) \leq \text{lev}(l_i)]$. As a consequence, if literal l is unit, it has to be assigned a value such that it satisfies the clause which makes it unit.

Example 2.3 shows the application of core QDPLL algorithm enhanced by three aforementioned improvements.

Example 2.3 Consider the following QBF Φ with prefix

$\Phi_{\text{prefix}} = \exists a \forall x \exists b \forall y \exists c$, and matrix

$\Phi_{\text{matrix}} = (a+b+y+c)(a+x+b+y+\bar{c})(x+\bar{b})(\bar{y}+c)(\bar{c}+\bar{a}+\bar{x}+b)(\bar{x}+\bar{b})(a+\bar{b}+\bar{y})$.

Its complete search tree is shown in Figure 2.9. $\langle *, L \rangle$ and $\langle *, R \rangle$ show the corresponding branching assignments. $\langle *, U \rangle$, $\langle *, P \rangle$ and $\langle *, R \rangle$ stand for an out-of-order assignment of the variable due to the unit propagation, pure literal elimination or \forall -reduction respectively. Red arrows stand for the false valuation under an assignment, green arrows - for the true valuation. During the search we assume the worst scenario, i.e. that all the branches have to be explored to determine the value at a given tree node.

In Section 3.1, after proving the soundness and completeness of Q-resolution

2.3. Quantified Boolean Formulae (QBF)



proof system, we will show how Q-resolution proofs can be constructed from the QBF search tree.

2.3.6 QBF certificates applications

Usually to validate results of a QBF solver, both Q-resolution proofs (syntactic certificates) and Skolem functions (semantic certificates) are commonly accepted certificates [33]. As it was stated in previous subsections, for true QBFs either cube Q-resolution proofs or Skolem-function models can certify satisfiability. For false QBFs - Herbrand-function countermodels or clause Q-resolution proofs can certify unsatisfiability. In theory, a false QBF can be negated to a true QBF, whose Skolem model can then be used as a Herbrand countermodel to the original false QBF.

Q-refutations are not directly useful for applications besides the solver answer verification. Skolem and Herbrand functions on the other hand are useful for synthesis applications. It is important to mention that search-based solvers are not able to produce (counter)models directly, they, however, can easily produce Q-refutations. In the next chapter we will go through Q-resolution proofs construction in the context of search-based QBF-solvers. Further, resolution proofs are directly related to Skolem/Herbrand functions as will be evidenced from Section 4.2. We will also introduce algorithms for Skolem/Herbrand function extraction from various derivatives of Q-resolution proofs.

We first show applications of semantic QBF certificates to the problem of *Boolean relation determinization*, which is useful in logic and property synthesis [27, 28]. A *Boolean relation* over *input variables* X and *output variables* Y is a characteristic function $R : \{0, 1\}^{|X|+|Y|} \rightarrow \{0, 1\}$ such that assignments $a \in \{0, 1\}^{|X|}$ and $b \in \{0, 1\}^{|Y|}$ make $R(a, b) = 1$ if and only if (a, b) is in the relation. Relations can be exploited to specify the permissible (non-deterministic) behavior of a system, by restricting its allowed input X and output Y combinations. To be implemented with circuits, a relation has to be *determinized* in the sense that each output variable $y_i \in Y$ can be expressed by some function $f_i : \{0, 1\}^{|X|} \rightarrow \{0, 1\}$. Formally it can be written as a QBF $\forall X, \exists Y. R(X, Y)$, and the determinization problem corresponds to finding the Skolem functions for Y variables.

Often formula $R(X, Y)$ is not represented in CNF but rather in some circuit

2.3. Quantified Boolean Formulae (QBF)

structure. By previously mentioned Tseitin transformation, it can be rewritten in CNF $\phi_R(X, Y, Z)$ with the cost of introducing some new intermediate variables Z . Therefore QBF is rewritten as $\forall X, \exists Y, \exists Z. \phi_R(X, Y, Z)$. Hence the determinized relation could be found as the Skolem functions of the above formula. Alternatively, we may compute the Skolem functions by finding the countermodel of negated QBF $\exists X, \forall Y. \neg R(X, Y)$, which can be again by Tseitin transformation translated into PCNF $\exists X, \forall Y, \exists Z'. \phi_{\neg R}(X, Y, Z')$ with Z' being the newly introduced intermediate variables in the circuit of $\neg R(X, Y)$. These two approaches to relation determinization are to be studied in the experiments part of Section 4.2.

QBF semantic certificates have other applications, for example a (counter)model could represent a concrete plan leading to a goal state in a planning problem, a circuit in system design, or a winning strategy in a two player game. We, however, do not focus on these applications in this work.



Chapter 3

Syntactic certificates and QBF

Various types of QBF certificates were introduced in Section 2.3. Semantic certificates are usually represented in the form of Skolem-function models and Herbrand-function countermodels, while syntactic certificates are mostly based on Q-resolution rule. In this chapter we are going to give a deeper discussion for syntactic QBF certificates. In the following Section 3.1 we provide a full review of Q-resolution proof system, formally prove its completeness, soundness and intractability. Further show how Q-resolution proofs are constructed during search in search-based QBF solvers. Section 3.2 gives an overview of existing extensions of Q-resolution proof systems (LQ- and QU-resolution proof systems), where we also briefly discuss their complexity properties, benefits and limitations. We conclude this chapter by introduction of new extended resolution proof systems in Section 3.3 (LQU and LQU+proof systems).

3.1 Q-resolution proof system

Essentially, the most commonly used syntactic certificates for false QBF formulas nowadays are those based on Q-resolution (Q-consensus for true QBFs). In this section we briefly overview Q-resolution and show on Example 2.3 from Section 2.3 how Q-resolution proofs are constructed in QDPLL based QBF solvers.



3.1.1 Introduction revisited

Resolution reinforced with universal reduction (i.e. Q-resolution) forms a proof system for QBF language. Below we give a formal proof to the soundness and completeness of clause Q-resolution, defined in Section 2.3 (the proof for soundness and completeness of cube Q-resolution is very similar, so we omit it).

Theorem 3.1 [30] Given QBF Φ is false, if and only if there exists a clause Q-resolution proof, leading to an empty clause: $\Phi \xrightarrow[q-res]{} \square$.

Proof We prove the statement by induction on the number of quantifiers k in prefix of Φ .

If $k = 1$ and $\Phi = \exists x\phi$, then Q-resolution equals ordinary resolution, and since ordinary resolution is complete, we always can derive an empty clause, and vise-versa.

If $k = 1$ and $\Phi = \forall x\phi$, then taking in account that $\Phi = 0$, there exists at least one clause in ϕ consisting only of variable x , which after universal reduction becomes an empty clause.

Now we assume that for all formulas with number of quantifiers $k \leq n$ the statement is true. If $k = n + 1$ and $\Phi = \forall x \vec{Q} \vec{y} \phi$, let ϕ_0 and ϕ_1 be cofactors of ϕ with respect to variable x (since ϕ is in CNF form, ϕ_0 can be obtained by removing literals x and clauses with \bar{x} , and ϕ_1 by removing literals \bar{x} and clauses with x). Since $\Phi = \Phi_0 \wedge \Phi_1$, where $\Phi_0 = \vec{Q} \vec{y} \phi_0$ and $\Phi_1 = \vec{Q} \vec{y} \phi_1$, either $\Phi_0 = 0$ or $\Phi_1 = 0$. Without loss of generality let's assume $\Phi_0 = 0$. By inductive hypothesis $\Phi_0 \xrightarrow[q-res]{} \square$, and its Q-resolution proof can be used to conclude $\Phi \xrightarrow[q-res]{} x$ or $\Phi \xrightarrow[q-res]{} \square$, since after returning literal x to all clauses it has been removed from, at most x will be in the final clause of the proof. Clearly, $x \xrightarrow[q-res]{} \square$, and thus $\Phi \xrightarrow[q-res]{} \square$.

The last case is when $k = n + 1$ and $\Phi = \exists x \vec{Q} \vec{y} \phi$. Let all the definitions be as above. Now, $\Phi = \Phi_0 \vee \Phi_1$. By inductive assumption, $\Phi_0 \xrightarrow[q-res]{} \square$ and $\Phi_1 \xrightarrow[q-res]{} \square$. So $\Phi \xrightarrow[q-res]{} \square$ or $\Phi \xrightarrow[q-res]{} x$ must be true, and also $\Phi \xrightarrow[q-res]{} \square$ or $\Phi \xrightarrow[q-res]{} \bar{x}$ must be true. Since $x\bar{x} \xrightarrow[q-res]{} \square$, we can conclude $\Phi \xrightarrow[q-res]{} \square$.

So, by the principle of induction theorem statement follows. ■

3.1. Q-resolution proof system

In the context we shall think about Q-resolution proof Π of the falsity of a QBF $\Phi = \mathcal{P}.\phi$ as about a directed acyclic graph (DAG) representing clauses derived from ϕ by repeated applications of the respective rules in process of deriving \square . If it only contains such clauses that contribute to the derivation of \square we call it a *core* proof. Otherwise we say that the proof contains redundant steps. The operation **reduce** is applied to any clause in Π from which it can remove a literal. We call application of either resolution or reduction as a *step*. The *size* of Π is the number of clauses in Π that are derived by resolution (not by reduction). By *topological order* we refer to any order following the derivation steps in Π from the clauses in ϕ to \square .

Proof construction in search-based QBF solvers is similar to the regular resolution proof construction in SAT solving. The approach we are going to show here is similar to that used in the state-of-the-art QBF solvers QUBE-CERT [23] and DEPQBF [31] that both use QDPLL as a core search algorithm, with some improvements, including unit propagation, and pure-literal elimination.

According to algorithm *QDPLL* in Figure 2.8 variables are going to be assigned in accordance with the quantification order, i.e. from outer to inner.

Recall the example used in Section 2.3

$$\begin{aligned}\Phi_{prefix} &= \exists a \forall x \exists b \forall y \exists c, \\ \Phi_{matrix} &= (a + b + y + c)(a + x + b + y + \bar{c})(x + \bar{b})(\bar{y} + c) \\ &\quad (\bar{c} + \bar{a} + \bar{x} + b)(\bar{x} + \bar{b})(a + \bar{b} + \bar{y}).\end{aligned}$$

Its complete search tree was shown in Figure 2.9.

Now, as we have the search tree, we might go on to the clause-resolution (since QBF is false) proof construction. What we have to do, basically, is to look into one of the minimal sets of branches which is responsible for the false answer (in our case all false branches). These branches are shown in Figure 3.1.

When the false branch is encountered it means that there is a clause in the original formula which is falsified by the current assignment. These clauses are shown to the right of the “false” rectangles in Figure 3.1. The clauses, written to the left of unit implication steps, are the clauses from the original CNF, responsible for the unit implication to happen. Next we traverse the chosen part of the tree in a

3.1. Q-resolution proof system

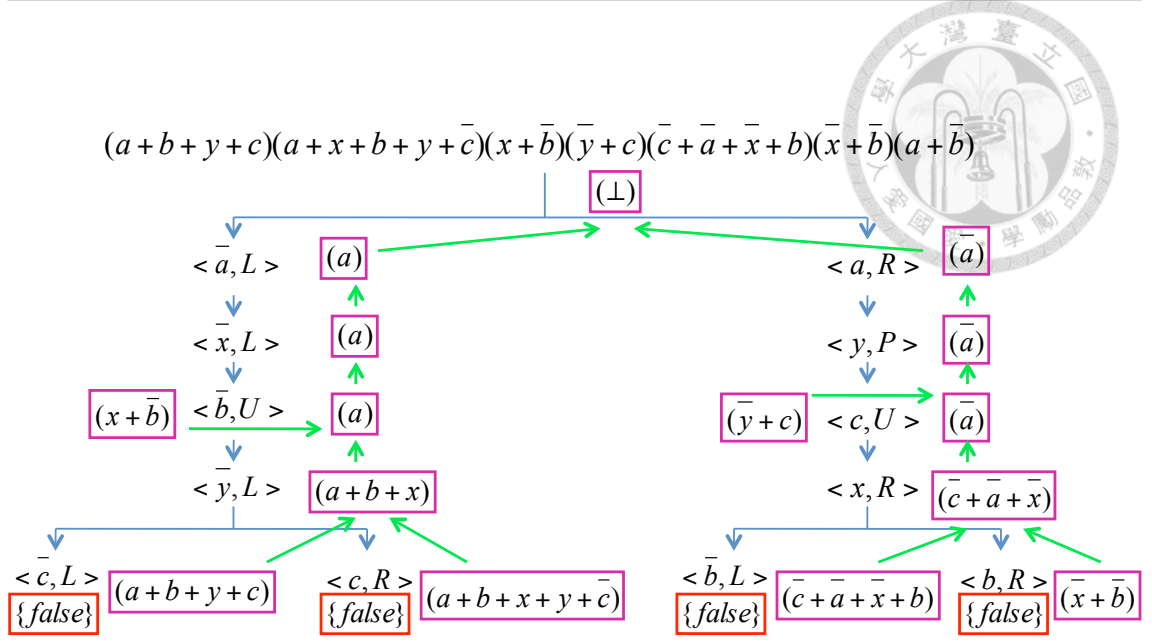


Figure 3.1: Q-resolution proof construction from the search tree.

breadth-first manner, starting from the bottom of each branch and following to the top. For each encountered step we assign a clause and write it to the right of the step. For steps, followed after variable splitting, we assign a resolvent of the clauses assigned to the splitting steps. For pure elimination steps we just copy the clause assigned from the previous step. For unit propagation steps we assign a resolvent of the clause, standing to the left of it, and the clause, standing to the right of the bottom neighbor step. In our example all the assigned clauses together with their resolution derivation are shown in Figure 3.1.

As a consequence, resulting Q-resolution proof is shown in Figure 3.2.

The complete proof that above method always gives correct resolution proofs (although in some rare cases they might involve long-distance resolution, and should be rearranged in a special manner that we will show later) can be found in [23], as well as examples for cube Q-resolution proof construction (which is identical to clause Q-resolution proofs construction).

3.1.2 Intractability of Q-resolution

Years earlier resolution was proven to be intractable for SAT. As propositional satisfiability is a special case of QSAT, propositional resolution is just a special case of Q-resolution when all the variables are existential. Therefore an exponential lower

3.1. Q-resolution proof system

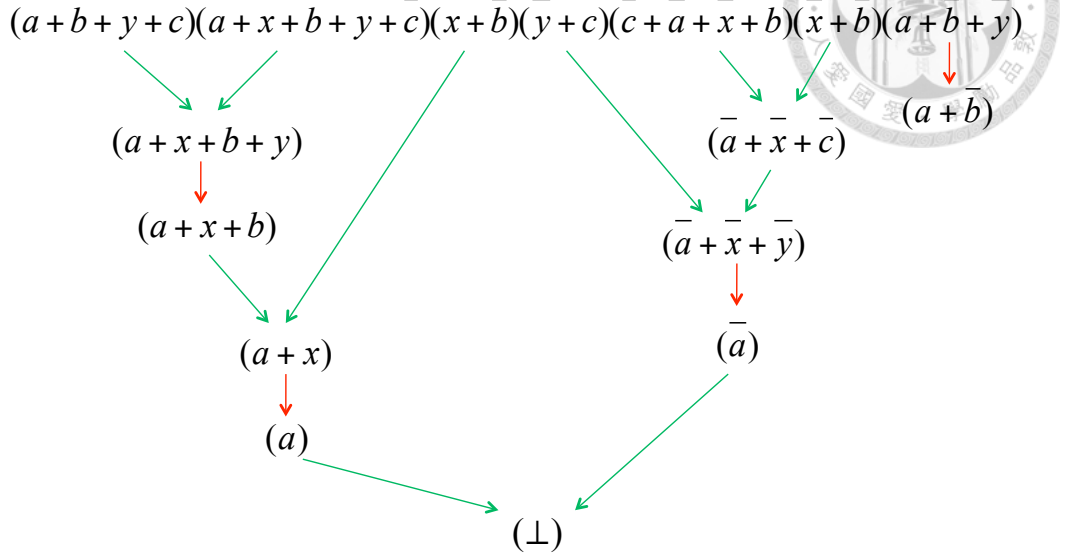


Figure 3.2: DAG for Q-resolution proof of Φ , constructed by algorithm QDPLL.

bound on resolution applies to Q-resolution as well. In this subsection we will define a unique to QBF family of formulas that not only prove intractability of Q-resolution for QBF, but also allows us to introduce extensions of Q-resolution (for example long distance resolution was already informally introduced) that will be useful in practice.

Definition 3.1 reproduces the definition of the KBKF family [29] of formulas.

Definition 3.1 (KBKF family [29]) For $t > 1$, the t^{th} member KBKF[t] of the KBKF family consists of the following prefix and clauses:

$$\exists d_1 e_1 \forall x_1 \exists d_2 e_2 \forall x_2 \dots \exists d_t e_t \forall x_t \exists f_1 \dots f_t$$

$$\begin{aligned} B &= (\bar{d}_1, \bar{e}_1) \\ D_i &= (d_i, x_i, \bar{d}_{i+1}, \bar{e}_{i+1}) & E_i &= (e_i, \bar{x}_i, \bar{d}_{i+1}, \bar{e}_{i+1}) & \text{for } i \in [1..t-1] \\ D_t &= (d_t, x_t, \bar{f}_1, \dots, \bar{f}_t) & E_t &= (e_t, \bar{x}_t, \bar{f}_1, \dots, \bar{f}_t) \\ F_i &= (x_i, f_i) & F'_i &= (\bar{x}_i, f_i) & \text{for } i \in [1..t] \end{aligned}$$

Observe that every member of KBKF-family is false. It could be both verified using Q-resolution or from game theoretic point of view. For the latter case reader could check that functions $H_{x_i} = d_i$ for $i \in [1..t]$ form a valid Herbrand countermodel for QBF KBKF[t]. As we shall see through this subsection, formulas from KBKF family are hard for Q-resolution. In Figure 3.3 you could see example of a Q-resolution proof

3.1. Q-resolution proof system

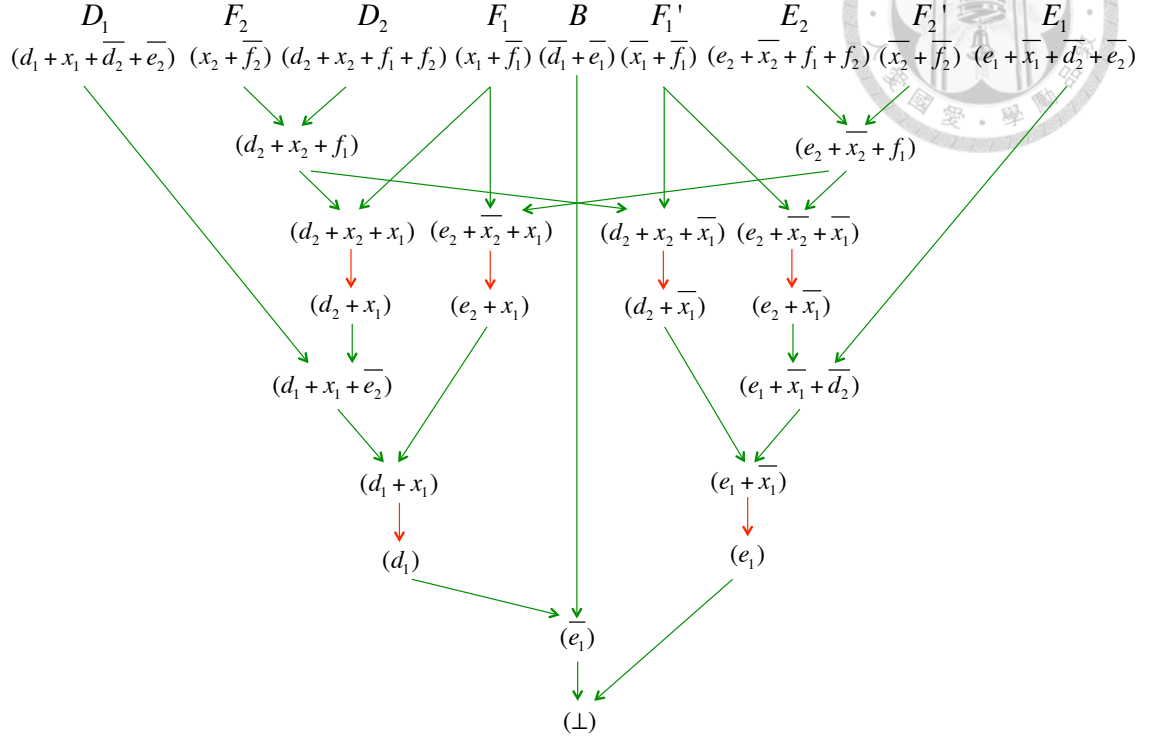


Figure 3.3: Q-resolution proof for QBF KBKF[t] with $t = 2$.

for QBF KBKF[t] with $t = 2$. In Section 3.2 we will show that with simple changes to Q-resolution corresponding refutations could become exponentially shorter.

Theorem 3.2 in [29] claims that any Q-resolution proof for members of KBKF family is of size exponential in t [29], but its proof is not completely given. Theorem 3.2 restates the claim of Theorem 3.2 in [29], and an alternative rigorous proof is given. First we propose the following Lemmas 3.1 and 3.2 that we use for the proof of Theorem 3.2 which then follows.

Lemma 3.1 Given a false QBF $\Phi = \mathcal{P}(\phi \wedge C)$ and its corresponding Q-resolution proof Π , denote C_1, C_2, \dots, C_n as the set of all immediate successors of C in Π . Then QBF $\Phi^* = \mathcal{P}(\phi \wedge C_1 \wedge C_2 \wedge \dots \wedge C_n)$ is false, and there exists a Q-resolution proof Π^* of its unsatisfiability with $|\Pi^*| \leq |\Pi|$.

Proof Q-refutation Π^* for Φ^* can be chosen as a subproof of Π , excluding those steps which derive C_1, \dots, C_n from C . Π^* both certifies the falsity of Φ^* and satisfies $|\Pi^*| \leq |\Pi|$. ■

Definition 3.2 Given an arbitrary index t , consider KBKF[t]. Denote an f -diminished

3.1. Q-resolution proof system



expansion of $KBKF[t]$ the following formula

$$\exists d_1 e_1 \forall x_1 \exists d_2 e_2 \forall x_2 \dots \exists d_t e_t$$

$$\begin{aligned} B &= (\bar{d}_1, \bar{e}_1) \\ D_i &= (d_i, x_i, \bar{d}_{i+1}, \bar{e}_{i+1}) \quad E_i = (e_i, \bar{x}_i, \bar{d}_{i+1}, \bar{e}_{i+1}) \quad \text{for } i \in [1..t-1] \\ D^* &= (d_t, x_1^-, \dots, x_{t-1}^-) \quad E^* = (e_t, x_1^-, \dots, x_{t-1}^-) \end{aligned}$$

In Definition 3.2 x_i^- stands for any (positive or negative) literal of variable x_i , and D^* (respectively E^*) represents a set of clauses going through all possible combinations of phases of variables x_1, \dots, x_{t-1} disjuncted with d_t (respectively e_t).

Lemma 3.2 f -diminished expansion of $KBKF[t]$ is false. Furthermore, removing at least one clause from set D^* or E^* makes it true.

Proof The falsity of f -diminished expansion of $KBKF[t]$ directly follows from Lemma 3.1. For the second part of the lemma, w.l.o.g. assume that clause $C = (d_t, x_1, \dots, x_{t-1})$ was removed from D^* . The resulting QBF is true since now it has Skolem functions $f_{e_i} = 1$ for $i \in [1..t]$, $f_{d_i} = 0$ for $i \in [1..t-1]$, and $f_{d_t} = x_1 \vee x_2 \vee \dots \vee x_{t-1}$. ■

Theorem 3.2 (Theorem 3.2 in [29]) Given a $KBKF$ family of QBFs, as defined by Definition 3.1 and an index t , then the smallest Q-refutation for $KBKF[t]$ is exponential in t .

Proof Prove by contradiction. Assume there exists a family of Q-resolution proofs $\Pi[t]$ for the corresponding $KBKF[t]$, bounded by some polynomial function of t .

Let us denote $\Pi = \Pi[t]$. Call the clause $C = \text{resolve}(C_1, f_i, C_2)$ an f -tailing if $\text{var}(f_i) \notin \text{vars}(C)$ for any $i \in [1..t]$. Intuitively, f -tailing clause is one of the places in the proof Π where all f_i variables have just been resolved out. Note that if C is an f -tailing clause, then $\text{var}(x_i) \in \text{vars}(C)$ for all $i \in [1..t]$ (elimination of each \bar{f}_i introduces $\text{var}(x_i)$, and none of them could be reduced until C due to the presence of at least one \bar{f}_j).

Now consider clause $D_t = (d_t, x_t, \bar{f}_1, \dots, \bar{f}_t)$. Denote Σ_D as a set of all f -tailing descendants C of D_t in Π , such that no other f -tailing clause can be found on some

3.1. Q-resolution proof system

path from D_t to C . Note that Σ_D is a proper cut separating D_t from \square in Π . By our assumption Σ_D has a polynomial size in t . Furthermore, the longest path from D_t to any clause in Σ_D also has a polynomial in t size. Note that for any clause C' on any path from D_t to C (including C itself) we have $x_t \in C'$, since no reductions was possible prior to derivation of C .

Next for each $C \in \Sigma_D$ copy the part of Π that derives C from the matrix clause set. In this way we separate the derivation of clauses in Σ_D , i.e. they do not share any ancestors except of the clauses in the matrix. Call the resulting proof as Π' . By our assumption Π' still has polynomial size in t . Take any clause $C \in \Sigma_D$. Without loss of generality assume $x_i \in C$ for all $i \in [1..t-1]$ (recall that $x_t \in C$, and for any other combination of the phases of the rest universal variables discussion is absolutely the same). Please note two things. First, that along any path from D_t to C no resolutions with clauses $F'_i = (\bar{x}_i, f_i)$ is possible (since otherwise literal \bar{x}_i must be present in C , violating assumption $x_i \in C$ for all $i \in [1..t]$). Second, clause $E_t = (e_t, \bar{x}_t, \bar{f}_1, \dots, \bar{f}_t)$ can not be responsible for presence of any of \bar{f}_i literals anywhere on the path from D_t to C ($\bar{x}_t \in E_t$ forbids on resolving with any clause on the path from D_t to C , and the necessary condition for elimination of \bar{x}_t is absence of any of \bar{f}_i literals).

Now consider the following procedure: perform resolutions of D_t with all the clauses $F_i = (x_i, f_i)$ one by one, and then universally reduce x_t to produce clause $D_{x_1, \dots, x_{t-1}}^* = (d_t, x_1, \dots, x_{t-1})$ (index x_1, \dots, x_{t-1} corresponds to the phases of literals in C , which in this case are all positive). Now use clause $D_{x_1, \dots, x_{t-1}}^*$ instead of D_t in derivation of C , disregarding any future resolutions on f_i variables on any path from $D_{x_1, \dots, x_{t-1}}^*$ to C . By all the previous discussions, the new derivation of C is sound in Q-resolution proof system.

Note that we did not increase the size of Π' by this procedure. Intuitively, we just shifted all the resolutions on f_i variables topologically closer to the matrix clauses. Perform above procedure for all the clauses in Σ_D and denote set $D^* = \bigcup_{\Sigma_D} (D_{x_1^-, \dots, x_{t-1}^-}^*)$, where subscript x_1^-, \dots, x_{t-1}^- indicates all combinations of phases of variables x_1, \dots, x_{t-1} that are present in clauses Σ_D .

Next repeat the same procedure for clause E_t to obtain clauses $E^* = \bigcup_{\Sigma_E} (E_{x_1^-, \dots, x_{t-1}^-}^*)$, and denote the resulting proof as Π'' . After application of Lemma 3.1 for clauses D_t

3.2. Extensions of Q-resolution proof system



and E_t in Π'' , we conclude that formula

$$\Phi^* = \exists d_1 e_1 \forall x_1 \exists d_2 e_2 \forall x_2 \dots \exists d_t e_t (B \bigcup_{i=1..t-1} (D_i \cup E_i) \cup D^* \cup E^*)$$

must be false, and has a polynomial in t proof Π^* . On the other hand, this means that sets D^* and E^* have to be polynomial in t , i.e. there are some combinations of phases of variables x_1, \dots, x_{t-1} that are not present in D^* and E^* . This however contradicts with Lemma 3.2, which in this case states that Φ^* must be true. ■

3.2 Extensions of Q-resolution proof system

Modern search-based QBF solvers are equipped with conflict-driven learning, which, as we have seen from Section 3.1 performs resolution in essence. Tautological clause containing both positive and negative literals of a (universal) variable may result from resolution [42]. Therefore in QBF reasoning the derivation of such clauses may be useful under certain conditions. Since the clause is resolved from two clauses with distance greater than 1, it is referred to as *long-distance resolution*. Unlike the case in propositional satisfiability, such a clause is not totally redundant as it facilitates implication in QBF evaluation. In this section we explore the influence of allowing tautological clauses in addition to Q-resolution derivation rules, and also see how restrictions on pivot variables in Q-resolution influence the strengths of the proof system.

3.2.1 LQ- and QU-resolution proof systems

Universal variable x contained in a clause C as both x and \bar{x} is called a *merged variable*. A *merged literal* l^* is used to replace both literals l and \bar{l} in C . We define $\text{var}(l^*) = \text{var}(l)$, $\text{lev}(l^*) = \text{lev}(l)$, and $\text{idx}(l^*) = \text{idx}(l)$.

Recall that given two clauses C_1 and C_2 , and a pivot variable p with $p \in C_1$, $\bar{p} \in C_2$, resolution produces the clause $\text{resolve}(C_1, p, C_2) = C_1 \setminus \{p\} \cup C_2 \setminus \{\bar{p}\}$. Since we defined a new syntactic element, namely - a merged literal, the definition of *ordinary* resolution is updated accordingly:

Resolution step is called regular if for all (merged or regular) literals $l_1 \in C_1 \setminus \{p\}$

3.2. Extensions of Q-resolution proof system

and $l_2 \in C_2 \setminus \{\bar{p}\}$ it holds that if $\text{var}(l_1) = \text{var}(l_2)$ then $l_1 = l_2$ and l_1 is not merged. Otherwise we refer to it as *long-distance* resolution. We further distinguish ordinary resolution into **resolve_∃** if $p \in V_∃$ and **resolve_∀** if $p \in V_∀$. Recall that Q-resolution only allowed existential pivots $p \in V_∃$. We call long-distance resolution over pivot $p \in V_∃$ *proper* and denote it by **resolve_{∃L}** if the following *index restriction* holds:

For all (merged or regular) literals $l_1 \in C_1 \setminus \{p\}$ and $l_2 \in C_2 \setminus \{\bar{p}\}$ it holds that if $\text{var}(l_1) = \text{var}(l_2)$ and either $l_1 \neq l_2$ or l_1 is merged, then $\text{var}(l_1) \in V_∀$ and $\text{idx}(l_1) = \text{idx}(l_2) > \text{idx}(p)$. Note that since $p \in V_∃$ and $l_1, l_2 \in V_∀$, **lev** can be used instead of **idx**. Further recall from Section 2.3 that universal reduction rule (**reduce**) removes from C all universal variables whose quantifier levels are greater than the largest level of any existential variable in C . We define **reduce** to be applied to merged literals from C in the same way as it applies to regular literals.

Besides Q-resolution that contains the derivation rules **reduce** and **resolve_∃**, there are two its sound and complete derivatives: QU-resolution [40] and LQ-resolution [5] where rules **reduce** and **resolve_∃** are extended by the rules **resolve_∀** and **resolve_{∃L}**, respectively. LQ and QU resolution proofs possess exactly the same structure as we defined for Q-resolution proofs.

Both QU and LQ resolution proof systems strictly contain Q-resolution, i.e. they both polynomially simulate Q-resolution (obviously since they strictly extend it), but Q-resolution simulates neither of them. Latter fact was established in [40] for QU-resolution and in [21] for LQ. To prove it [40] and [21] showed short (polynomial) proofs for KBKF family of formulas. We illustrate the idea behind short proofs construction for our example of the second member of KBKF family. Figures 3.4 and 3.5 show QU and LQ resolution proofs for KBKF[2] respectively.

In next subsection we formally prove soundness of LQ-resolution proof system, by showing that any LQ-resolution proof could be transformed into an ordinary Q-resolution proof.

3.2.2 Transformation of LQ-resolution proofs to Q-resolution proofs

Certain applications might restrict us from using LQ-resolution (for example semantic certificates extraction algorithm is easier for proofs without long-distance step).

3.2. Extensions of Q-resolution proof system

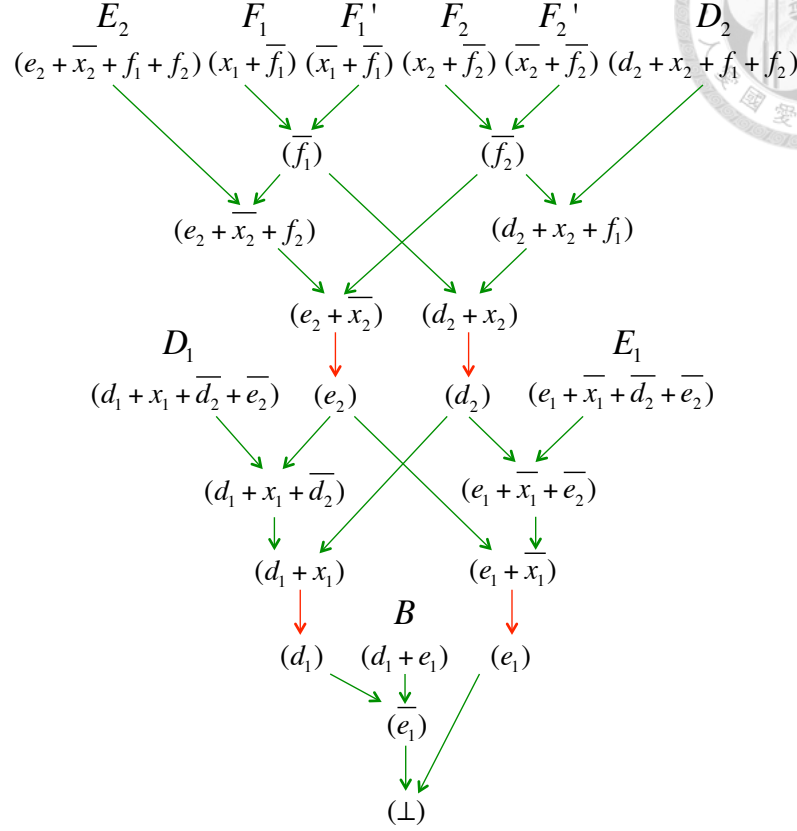


Figure 3.4: QU-resolution proof for QBF KBKF[t] with $t = 2$.

Therefore it might be desirable to convert given LQ-resolution proof into Q-refutation. Please recall that special *merged* literal notation l^* was defined to represent $l \vee \neg l$ for some literal l . LQ-resolution, formally defined in the last subsection, could be represented in form of the following three axioms:

$$\frac{C_1 \vee p \vee l \quad C_2 \vee \neg p \vee \neg l}{\text{MIN}^*(C_1 \vee C_2 \vee l^*)}$$

$$\frac{C_1 \vee p \vee l \quad C_2 \vee \neg p \vee l^*}{\text{MIN}^*(C_1 \vee C_2 \vee l^*)}$$

$$\frac{C_1 \vee p \vee l^* \quad C_2 \vee \neg p \vee l^*}{\text{MIN}^*(C_1 \vee C_2 \vee l^*)}$$

where p and l are existential and universal literals, respectively, with $\ell(p) < \ell(l)$. Recall that MIN^* extends the usual \forall -reduction MIN to reduce also l^* literal in the

3.2. Extensions of Q-resolution proof system

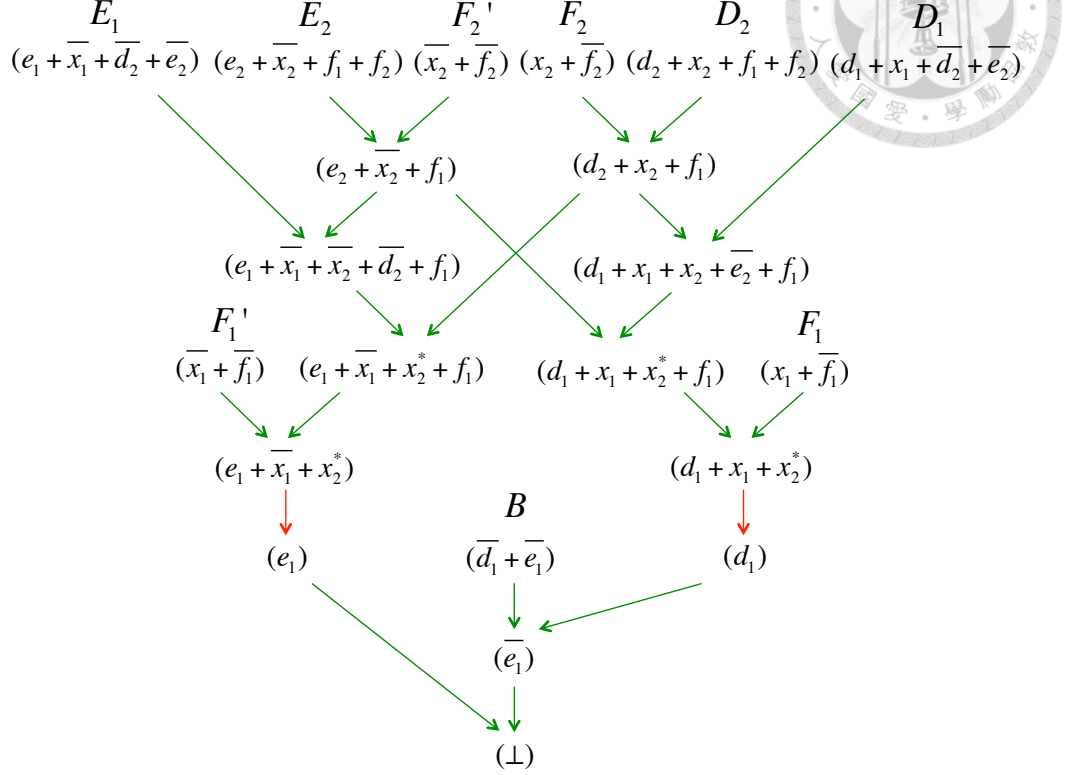


Figure 3.5: LQ-resolution proof for QBF KBKF[t] with $t = 2$.

same way as the l and $\neg l$ literals. Notice that above quantification level restriction applies to other merged variables in C_1 and C_2 as well. Otherwise long-distance resolution would be unsound. Also note that the resolution

$$\frac{C_1 \vee p \quad C_2 \vee \neg p \vee l^*}{\text{MIN}^*(C_1 \vee C_2 \vee l^*)}$$

is not considered as long-distance, since no merging occurs in this step. Therefore there is no quantification level restriction imposed on l .

As the following example shows, LQ-resolution proofs might be more condense compare to regular Q-resolution proofs for the same QBF.

Example 3.1 Consider the following QBF, whose unsatisfiability is shown by the

3.2. Extensions of Q-resolution proof system

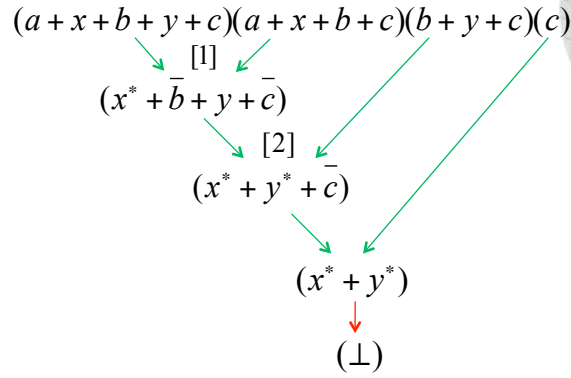


Figure 3.6: DAG of resolution proof Π^* .

resolution proof Π^* and depicted in Figure 3.6.

$$\begin{aligned}
 \Phi_{\text{pfx}} &= \exists a \forall x \exists b \forall y \exists c \\
 \Phi_{\text{mtx}} &= (a \vee x \vee \neg b \vee y \vee \neg c)(\neg a \vee \neg x \vee \neg b \vee \neg c)(b \vee \neg y \vee \neg c)(c) \\
 \Pi^* &= \left\{ \begin{array}{l} 1. \text{ clause}_5 = \text{resolve}(\text{clause}_1, \text{clause}_2) \\ 2. \text{ clause}_6 = \text{resolve}(\text{clause}_3, \text{clause}_5) \\ 3. \text{ clause}_\square = \text{resolve}(\text{clause}_4, \text{clause}_6) \end{array} \right\}
 \end{aligned}$$

As shown, the presence of long-distance resolution makes the proof more succinct.

Above resolution rules in addition to Q-resolution form a sound and complete proof system (LQ-resolution) for QBF evaluation as stated below.

Proposition 3.1 Let C_3 be the resolvent under (possibly long-distance) resolution of clauses C_1 and C_2 . Let C'_1 and C'_2 be the sub-clauses of C_1 and C_2 , respectively, (with $l \subseteq l^*$ and $\neg l \subseteq l^*$). Then the resolvent C'_3 of C'_1 and C'_2 under the same pivot variable must be a sub-clause of C_3 .

By previous proposition the following claim can be concluded.

Proposition 3.2 Let Π be a (possibly long-distance) resolution proof of a false QBF $\Phi = Q_1 x_1 \cdots Q_n x_n \cdot \phi$. For an existential variable x_i , the induced proof $\Pi_{\neg x_i}$ (respectively Π_{x_i}), derived from Π with variable x_i being assigned constant 0 (respectively constant 1) forms a legitimate proof of QBF $Q_1 x_1 \cdots Q_{i-1} x_{i-1} Q_{i+1} x_{i+1} \cdots Q_n x_n \cdot \phi|_{\neg x_i}$ (respectively $Q_1 x_1 \cdots Q_{i-1} x_{i-1} Q_{i+1} x_{i+1} \cdots Q_n x_n \cdot \phi|_{x_i}$).

3.2. Extensions of Q-resolution proof system



Theorem 3.3 LQ-resolution proof system is sound and complete proof system for QBF evaluation. That is, a QBF Φ is false if and only if there exists an LQ-resolution sequence leading to the empty clause.

Proof Completeness follows directly from the fact that Q-resolution, which is a special case of LQ-resolution, is complete for QBF evaluation [29].

We prove its soundness by induction on the number k of quantifiers in the prefix of Φ . Let Π be a resolution proof of the falsity of Φ . For $k = 1$, $\Phi = \exists x.\phi$ or $\forall x.\phi$. Since there is no long-distance resolution involved in Π , Π must be sound (following from the fact that distance-1 resolution is sound).

Assume the soundness holds for every LQ-resolution proof of Φ with number of variables $k \leq n$. For $k = n + 1$ and $\Phi = \forall x_{n+1} Q_n x_n \cdots Q_1 x_1. \phi$, variable x_{n+1} can be \forall -reduced at most once in Π (namely in the last step of Π). If x_{n+1} in Π is not \forall -reduced at all, it means all the clauses with the occurrence of variable x_{n+1} are not involved in the resolution proof, and thus can be removed from the matrix. By induction hypothesis the soundness holds. If x_{n+1} in Π is \forall -reduced, it must be reduced in the form of literal x_{n+1} or literal $\neg x_{n+1}$ (cannot be x_{n+1}^* since long-distance resolution allows the quantification level of a pivot variable less than that of a merged variable). Without loss of generality, assume literal x_{n+1} is \forall -reduced. Since no steps except for the last one in Π involves reduction of variable x_{n+1} , the ancestor clauses of the empty clause cannot include literal $\neg x_{n+1}$. So if x_{n+1} is assigned as constant 0 (and thus removed from the prefix), the induced proof will still be a legitimate proof.

On the other hand, for $k = n + 1$ and $\Phi = \exists x_{n+1} Q_n x_{n+1} \cdots Q_1 x_1. \phi$, according to Proposition 3.2 the induced proofs $\Pi_{x_{n+1}}$ and $\Pi_{\neg x_{n+1}}$ are both legitimate proofs leading to the empty clause. So the unsatisfiability proof of $Q_n x_n \cdots Q_1 x_1. \phi|_{\neg x_{n+1}}$ and that of $Q_n x_n \cdots Q_1 x_1. \phi|_{x_{n+1}}$ establish the falsity of Φ . Therefore by induction hypothesis soundness holds. ■

The following two rewriting rules can be applied to eliminate long-distance resolution steps from LQ-refutation (i.e. to convert LQ-refutation into valid Q-refutation).

3.2. Extensions of Q-resolution proof system



$$\frac{\frac{C_1 \vee p \vee q \vee l_1 \quad C_2 \vee \neg p \vee l_2}{C_1 \vee C_2 \vee q \vee l^*} \quad \neg q \vee C_3}{C_1 \vee C_2 \vee C_3 \vee l^*}$$

$$\frac{\frac{C_1 \vee p \vee q \vee l_1 \quad \neg q \vee C_3}{C_1 \vee C_3 \vee p \vee l_1} \quad C_2 \vee \neg p \vee l_2}{C_1 \vee C_2 \vee C_3 \vee l^*}$$

$$\frac{\frac{C_1 \vee p \vee q \vee l_1 \quad C_2 \vee \neg p \vee q \vee l_2}{C_1 \vee C_2 \vee q \vee l^*} \quad \neg q \vee C_3}{C_1 \vee C_2 \vee C_3 \vee l^*} \longrightarrow$$

$$\frac{\frac{C_1 \vee p \vee q \vee l_1 \quad \neg q \vee C_3}{C_1 \vee C_3 \vee p \vee l_1} \quad \frac{C_2 \vee \neg p \vee q \vee l_2 \quad \neg q \vee C_3}{C_2 \vee C_3 \vee \neg p \vee l_2}}{C_1 \vee C_2 \vee C_3 \vee l^*}$$

where p, q, l_1 , and l_2 are literals with $l_1, l_2 \in \{l, \neg l, l^*\}$, and $l_1 \neq l_2$ or $l_1 = l_2 = l^*$. As stated in the following theorem, above rewriting rules, when applied in a proper order are complete in removing long-distance resolution steps from any LQ resolution proof.

Theorem 3.4 Given an LQ-resolution proof Π of a false QBF Φ , above two rewriting rules when applied on the long-distance resolution steps in a reverse topological order eventually yield a new proof without long-distance resolution.

Proof Consider the last (i.e., in the reverse topological order, the first encountered) long-distance resolution step S of Π . Let C'_1 and C'_2 be the resolving clauses at S . Then there must exist another clause C'_3 such that these three clauses match the three top-level clauses of either the first or the second rewriting rules above. Since no descendants of S are long-distance resolution steps, clause C'_3 cannot include variable $var(l)$. After applying the corresponding rewriting rule on these clauses, the new last long-distance resolution step S is moved one step closer to the empty clause without introducing any new long-distance resolution steps to the proof. Since S is again the closest to the empty clause, rewriting rules can be applied to it again and again until all the existential variables in its C'_1 and C'_2 clauses disappear and

3.2. Extensions of Q-resolution proof system

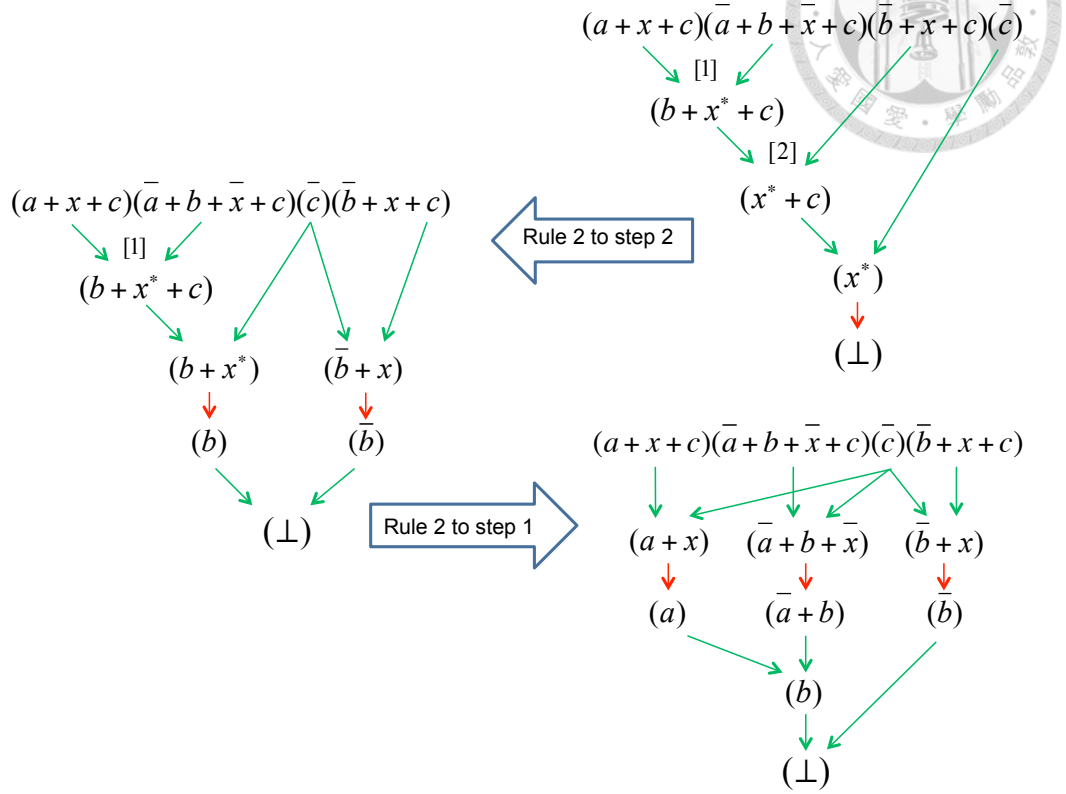


Figure 3.7: Conversion of LQ-resolution proof into Q-resolution.

$var(l)$ is eventually \forall -reduced. Consequently every last long-distance resolution step will be eventually removed. After this process is repeated for a sufficient number of iterations all long-distance resolution steps will be eliminated and resulting proof will only contain distance-1 resolutions (i.e. it is a Q-resolution proof). ■

Example 3.2 Consider the following QBF Φ and its LQ-resolution proof Π .

$$\begin{aligned}
 \Phi_{\text{pfx}} &= \exists a \exists b \forall x \exists c \\
 \Phi_{\text{mtx}} &= (a \vee x \vee c)(\neg a \vee b \vee \neg x \vee c)(\neg b \vee x \vee c)(\neg c) \\
 \Pi &= \left\{ \begin{array}{l} 1. \text{ clause}_5 = \text{resolve}(\text{clause}_1, \text{clause}_2) \\ 2. \text{ clause}_6 = \text{resolve}(\text{clause}_3, \text{clause}_5) \\ 3. \text{ clause}_\square = \text{resolve}(\text{clause}_4, \text{clause}_5) \end{array} \right\}
 \end{aligned}$$

Figure 3.7 shows the rewriting process converting Π to a new proof without long-distance resolution steps.



3.2.3 Incomparability of LQ- and QU-resolutions

Here we first show an exponential gap between the proof complexities of LQ-resolution and QU-resolution with respect to two families of QBFs obtained by modification to KBKF family of QBFs.

We first give an intuition of how to engineer a false QBF that inhibits resolve_\forall and $\text{resolve}_{\exists\text{L}}$ steps in any of its resolution proofs. Example 3.3 shows a false QBF for which any resolution proof cannot contain resolve_\forall or $\text{resolve}_{\exists\text{L}}$ steps.

Example 3.3 Consider the false QBF

$$\Phi = \exists a \forall x \forall y \exists b. (a, x, y, b)(\bar{a}, \bar{x}, \bar{y}, b)(x, y, \bar{b})(\bar{x}, \bar{y}, \bar{b}).$$

The falsity of Φ is shown by Herbrand functions $h_y = h_x = a$. Let Π be a QU-resolution proof of Φ . Since $\text{lev}(x) = \text{lev}(y)$, the universal reduction reduce always removes both x and y at once. Hence any clause in Π either contains both x and y in the same polarity, or neither x nor y in any polarity. It follows that Π cannot contain any clause derived by resolve_\forall . Alternatively, let Π be an LQ-resolution proof of Φ . Due to the level restriction, any $\text{resolve}_{\exists\text{L}}$ step must have a as pivot variable, so $\text{resolve}_{\exists\text{L}}((a, x, y, b), a, (\bar{a}, \bar{x}, \bar{y}, b)) = (x^*, y^*, b)$ is the only possible such step. However, this resolvent can never be used in a derivation of \square , because the necessary pivot literal \bar{b} always occurs in clauses together with literals of x and y , which forbids any further resolution.

For the remainder of this section it is important to keep in mind that for all $i \in [1..t]$, $\text{lev}(e_i) = \text{lev}(d_i) < \text{lev}(x_i)$ and $\text{lev}(x_t) < \text{lev}(f_i)$.

We now apply ideas from Example 3.3 to transform the KBKF family into the family KBKF-qu, for which, based on Theorem 3.2 in [29] (and Theorem 3.2 in our work), the smallest QU-refutations are of exponential size but there exist LQ-refutations of size polynomial in t . It follows from the existence of these proofs that the members of this family are false. For $t > 1$, KBKF-qu[t] is obtained from KBKF[t] by adding fresh universal variables y_i to some clauses.

Definition 3.3 (KBKF-qu family) For $t > 1$, the t^{th} member KBKF-qu[t] of the KBKF-qu family consists of the following prefix and clauses:

3.2. Extensions of Q-resolution proof system



$$\exists d_1 e_1 \forall x_1 y_1 \exists d_2 e_2 \forall x_2 y_2 \dots \exists d_t e_t \forall x_t y_t \exists f_1 \dots f_t$$

$$\begin{aligned} B &= (\bar{d}_1, \bar{e}_1) \\ D_i &= (d_i, x_i, y_i, \bar{d}_{i+1}, \bar{e}_{i+1}) & E_i &= (e_i, \bar{x}_i, \bar{y}_i, \bar{d}_{i+1}, \bar{e}_{i+1}) \quad i \in [1..t-1] \\ D_t &= (d_t, x_t, y_t, \bar{f}_1, \dots, \bar{f}_t) & E_t &= (e_t, \bar{x}_t, \bar{y}_t, \bar{f}_1, \dots, \bar{f}_t) \\ F_i &= (x_i, y_i, f_i) & F'_i &= (\bar{x}_i, \bar{y}_i, f_i) \quad i \in [1..t] \end{aligned}$$

The following proposition shows that the shortest Q-refutation for KBKF-qu[t] is at least as long as the shortest Q-refutation for KBKF[t].

Proposition 3.3 Given a false QBF $\Phi = Q_1 v_1 \dots Q_k v_k. C_1 \wedge C_2 \wedge \dots \wedge C_n$ over the set V of variables, it holds that for any variable $v \in V$, if $\Phi^* = Q_1 v_1 \dots Q_k v_k. C_1 \wedge \dots \wedge (C_j \cup \{\text{lit}(v)\}) \wedge \dots \wedge C_n$ is false, then the smallest {Q,QU,LQ}-resolution proof for Φ^* is at least as large as that for Φ .

Validity of Proposition 3.3 can be understood by the fact that removing the literal $\text{lit}(v)$ from some clause $(C_j \cup \{\text{lit}(v)\})$ can only decrease the proof size of Φ^* . Note that adding a fresh variable $v \notin V$ to \mathcal{P} influences neither the satisfiability of Φ , nor the validity of any of its Q-resolution proofs. Therefore Proposition 3.3 can be extended for addition of fresh variables to \mathcal{P} and their literals to ϕ .

Theorem 3.5 For $t > 1$ there exists an LQ-refutation of polynomial size for KBKF-qu[t], but any QU-refutation for KBKF-qu[t] is of exponential size in t (based on Theorem 3.2 in [29]).

Proof Except for clause B , each clause of KBKF-qu[t] contains two universal variables x, y with the same level and the same polarity. For any QU-refutation, in order to have a resolve_\forall step over two clauses C_1 and C_2 with x (respectively y) as pivot, y (x) must be removed from one of the clauses, which can only be done by reduce . Whenever y (x) is reduced, so is x (y). Therefore, any QU-refutation will be a Q-refutation, and by Theorem 3.2 in [29] and Proposition 3.3, the shortest Q-refutation for KBKF-qu is exponential. On the other hand, following the method proposed in Proposition 1 of [21], a polynomial LQ-refutation can be obtained. ■

We continue with the following modification of the KBKF family that inhibits $\text{resolve}_{\exists\text{L}}$ steps but allows polynomial QU-refutations. For $t > 1$, KBKF-lq[t] is retrieved from KBKF[t] by adding literals $\bar{f}_1, \dots, \bar{f}_t$ to clauses B, D_i and E_i , and literals $\bar{f}_{i+1}, \dots, \bar{f}_t$ to clauses F_i and F'_i , for all $i \in [1..t-1]$.

3.2. Extensions of Q-resolution proof system



Definition 3.4 (KBKF-lq family) For $t > 1$, the t^{th} member KBKF-lq[t] of the KBKF-lq family consists of the following prefix and clauses:

$$\exists d_1 e_1 \forall x_1 \exists d_2 e_2 \forall x_2 \dots \exists d_t e_t \forall x_t \exists f_1 \dots f_t$$

$$\begin{aligned} B &= (\bar{d}_1, \bar{e}_1, \bar{f}_1, \dots, \bar{f}_t) \\ D_i &= (d_i, x_i, \bar{d}_{i+1}, \bar{e}_{i+1}, \bar{f}_1, \dots, \bar{f}_t) & E_i &= (e_i, \bar{x}_i, \bar{d}_{i+1}, \bar{e}_{i+1}, \bar{f}_1, \dots, \bar{f}_t) & i &\in [1..t-1] \\ D_t &= (d_t, x_t, \bar{f}_1, \dots, \bar{f}_t) & E_t &= (e_t, \bar{x}_t, \bar{f}_1, \dots, \bar{f}_t) \\ F_i &= (x_i, f_i, \bar{f}_{i+1}, \dots, \bar{f}_t) & F'_i &= (\bar{x}_i, f_i, \bar{f}_{i+1}, \dots, \bar{f}_t) & i &\in [1..t-1] \\ F_t &= (x_t, f_t) & F'_t &= (\bar{x}_t, f_t) \end{aligned}$$

Observation 3.1 For $t > 1$ any member KBKF-lq[t] of the KBKF-lq family is an extended quantified Horn (QE-Horn) formula [29] and QE-Horn formulas are closed under LQ-resolution.

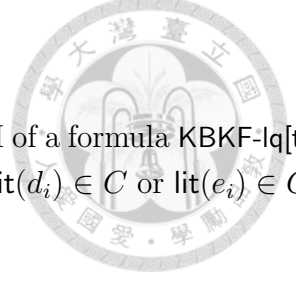
The closure of QE-Horn formulas under LQ-resolution directly follows from their closure under Q-resolution (observe that the $\text{resolve}_{\exists\text{L}}$ rule does not influence existential literals in the clauses). On the other hand, note that QE-Horn formulas are not closed under QU-resolution. Further, the following three invariants hold for any member of KBKF-lq family.

Lemma 3.3 (Invariant 1) Given any LQ-resolution proof Π of a formula KBKF-lq[t], the following holds for any clause $C \in \Pi$: For all $i \in [1..t]$, if $f_i \in C$ then $\text{lit}(x_i) \in C$, and if $\bar{f}_i \in C$ then for any $j \in [i..t]$ either $\bar{f}_j \in C$ or $\text{lit}(x_j) \in C$.

Proof First, observe that the invariant holds for any clause in the original clause set of KBKF-lq[t]. Let C be a clause derived from C' by exactly one derivation step, such that $f_i \in C$ and $f_i \in C'$. If $\text{lit}(x_i) \in C'$ then it must hold that $\text{lit}(x_i) \in C$, because resolution on universal variables is forbidden and the presence of f_i disallows the universal reduction of $\text{lit}(x_i)$ in both C' and C . Thus by induction it holds for any clause C that if $f_i \in C$ then $\text{lit}(x_i) \in C$.

Now let C be a clause derived from C' by exactly one derivation step, such that $\bar{f}_i \in C$ and $\bar{f}_i \in C'$. If $\text{lit}(x_j) \in C'$ for some $j \in [i..t]$, then $\text{lit}(x_j) \in C$ for the same reasons as above. If $\bar{f}_j \in C'$ for some $j \in [i..t]$, then either $\text{lit}(x_j) \in C$ (in the case where f_j is the pivot variable, i.e., $C = \text{resolve}(C', f_j, C'')$ with $f_j, \text{lit}(x_j) \in C''$ by the above discussion), or $\bar{f}_j \in C$ (in any other case). Thus by induction it holds for any clause C that if $\bar{f}_i \in C$ then for any $j \in [i..t]$ either $\bar{f}_j \in C$ or $\text{lit}(x_j) \in C$. ■

3.2. Extensions of Q-resolution proof system



Lemma 3.4 (Invariant 2) Given any LQ-resolution proof Π of a formula KBKF-lq[t] the following holds for any clause $C \in \Pi$: For all $i \in [1..t]$, if $\text{lit}(d_i) \in C$ or $\text{lit}(e_i) \in C$ then $f_j \notin C$ for any $j \in [1..t]$.

Proof First, the invariant holds for any clause in the original clause set of KBKF-lq[t]. Now let $C = \text{resolve}(C_1, p, C_2)$, where $\text{lit}(e_i) \in C$ or $\text{lit}(d_i) \in C$, and $\text{lit}(e_i) \in C_1$ or $\text{lit}(d_i) \in C_1$ for some $i \in [1..t]$.

If $\text{lit}(e_k) \in C_2$ or $\text{lit}(d_k) \in C_2$ for some $k \in [1..t]$, then by inductive hypothesis it holds that $f_j \notin C_1$ and $f_j \notin C_2$ for all $j \in [1..t]$. Therefore, by the definition of *resolve*, it holds that $f_j \notin C$ for all $j \in [1..t]$.

Else, $\text{lit}(e_i) \notin C_2$ and $\text{lit}(d_i) \notin C_2$, thus we are left with $p = f_k$ for some $k \in [1..t]$. By inductive hypothesis, $f_j \notin C_1$ for all $j \in [1..t]$, therefore $\bar{f}_k \in C_1$ and $f_k \in C_2$. By Observation 3.1 it holds that $f_j \notin C_2$ for all $j \in [1..t]$ with $j \neq k$. Thus for all $j \in [1..t]$ it holds that $f_j \notin C$.

Therefore, by induction it holds for any clause C and for all $i \in [1..t]$ that if $\text{lit}(d_i) \in C$ or $\text{lit}(e_i) \in C$ then $f_j \notin C$ for any $j \in [1..t]$. ■

Lemma 3.5 (Invariant 3) Given any LQ-resolution proof Π of a formula KBKF-lq[t] the following holds for any clause $C \in \Pi$: For all $i \in [1..t]$ it holds that if $\text{lit}(d_i) \in C$ or $\text{lit}(e_i) \in C$ then for any $j \in [1..i-1]$ either $\bar{f}_j \in C$ or $\text{lit}(x_j) \in C$.

Proof First, note that the invariant holds for any clause of the original clause set of KBKF-lq[t]. Now, let C be a clause retrieved from C' by one derivation step, such that $\text{lit}(e_i) \in C'$ or $\text{lit}(d_i) \in C'$, and $\text{lit}(e_i) \in C$ or $\text{lit}(d_i) \in C$. If for some $j \in [1..i-1]$ it holds that $\text{lit}(x_j) \in C'$, then $\text{lit}(x_j) \in C$ for the same reasons as in the proof of Invariant 1 (recall that $\text{lev}(e_i) = \text{lev}(d_i) > \text{lev}(x_j)$ for $j \in [1..i-1]$, therefore disallowing universal reduction of $\text{lit}(x_j)$ in the presence of either $\text{lit}(e_i)$ or $\text{lit}(d_i)$). If $\bar{f}_j \in C'$ for some $j \in [1..i-1]$, then either $\text{lit}(x_j) \in C$ (in the case where f_j is the pivot variable, i.e., $C = \text{resolve}(C', f_j, C'')$ with $\{f_j, \text{lit}(x_j)\} \in C''$ by Invariant 1), or $\bar{f}_j \in C$ (in any other case).

Therefore by induction it holds for any clause C and for all $i \in [1..t]$ that if $\text{lit}(d_i) \in C$ or $\text{lit}(e_i) \in C$ then for any $j \in [1..i-1]$ either $\bar{f}_j \in C$ or $\text{lit}(x_j) \in C$. ■

Theorem 3.6 For $t > 1$ there exists a QU-resolution proof of polynomial size for

3.2. Extensions of Q-resolution proof system



KBKF-lq[t], but any LQ-resolution proof for KBKF-lq[t] is of exponential size in t (based on Theorem 3.2 in [29]).

Proof For $t > 1$, a QU-refutation of polynomial size in t for KBKF-lq[t] can be constructed as follows: The unit clause (f_t) is obtained by the resolution step $\text{resolve}_\forall(F_t, x_t, F'_t)$. Then, for each $i \in [1..t-1]$, the unit clause (f_i) is obtained by recursively resolving all previous units $(f_{i+1})..(f_t)$ with the resolvent $\text{resolve}_\forall(F_i, x_i, F'_i)$. For $i \in [1..t]$ these unit clauses are used to remove all \bar{f}_i from the clauses D_i , E_i , and B , and the existential literals e_i and d_i are removed one after another by resolve_\exists over the remaining clauses.

For the remainder of this proof, let Π be an LQ-resolution proof for KBKF-lq[t]. Let the three clauses $C_1 = (A_1, p, X, R_1)$, $C_2 = (A_2, \bar{p}, \bar{X}, R_2)$, and $C = (A, X^*, R)$ be parts of a $\text{resolve}_{\exists\text{L}}$ step in Π , where X is a set of universal literals, $\bar{X} = \{\bar{x} \mid x \in X\}$, $X^* = \{x^* \mid x \in X\}$, $C = \text{resolve}_{\exists\text{L}}(C_1, p, C_2)$ is the resolvent of C_1 and C_2 , $A = A_1 \cup A_2$, and $R = R_1 \cup R_2$. Let x_m and x_n be the variables with the lowest, respectively the highest, level among the variables in X . By definition of $\text{resolve}_{\exists\text{L}}$ it holds that $\text{lev}(p) < \text{lev}(x_m)$. Without loss of generality, for $i \in \{1, 2\}$ let $R_i = \{v \in C_i \mid v \notin X \wedge \text{lev}(v) > \text{lev}(x_m)\}$ and $A_i = \{v \in C_i \mid v \notin (X \cup R_i \cup \{p\})\}$. Therefore, $R = \{v \in C \mid v \notin X^* \wedge \text{lev}(v) > \text{lev}(x_m)\}$ and $A = \{v \in C \mid v \notin (X^* \cup R)\}$. It is important to notice that the existential literals in R have to be removed from successors of C before X^* can be reduced. Further, $f_i \notin R$ for all $i \in [1..t]$ by Invariant 2, and $R_1, R_2 \neq \emptyset$ because otherwise x_m would be reduced before deriving C . Hence $R \subset \{\text{lit}(e_i), \text{lit}(d_i), \text{lit}(x_i) \mid m < i \leq t\} \cup \{\bar{f}_i \mid 1 \leq i \leq t\}$.

We now show by case distinction on the existential variables in R that the clause C can either not contribute to the derivation of \square in Π because at least one of the merged variables can never be reduced, or that the subclause A can be retrieved from C_1 , C_2 , and the input clauses in a polynomial number of derivation steps in the Q-resolution calculus. Under the assumption that Π is of polynomial size, its polynomial transformation into a Q-resolution proof contradicts with Proposition 3.3 and Theorem 3.2 in [29], stating that any Q-resolution for any member of KBKF-lq is exponential. Therefore, Π must be exponential.

Case 1. $\bar{f}_n \in R$. To remove \bar{f}_n , C has to be resolved with a clause C' containing f_n . By Invariant 1, C' contains $\text{lit}(x_n)$. Thus \bar{f}_n cannot be removed from R due to the level restriction on $\text{resolve}_{\exists\text{L}}$ steps. Therefore, $C \notin \Pi$.

3.2. Extensions of Q-resolution proof system

Case 2. $\text{lit}(d_i) \in R$ or $\text{lit}(e_i) \in R$. Recall that $i > m$, and without loss of generality, let $d_i \in R$. To remove d_i , C has to be resolved with a clause C' containing \bar{d}_i . By Invariant 3, C' either contains $\text{lit}(x_m)$ or \bar{f}_m . In the first case, the level restriction on $\text{resolve}_{\exists\mathbb{L}}$ steps forbids the resolution, and in the latter case, Invariant 1 applies to the resolvent similarly as in Case 1. Therefore, $C \notin \Pi$.

Case 3. $\bar{f}_i \in R$ and $i < n$. Similarly to Case 2, to remove \bar{f}_i , C has to be resolved with a clause C' containing f_i . By Invariant 1, C' either contains $\text{lit}(x_n)$, which blocks the resolution as in Case 1 and Case 2, or it contains \bar{f}_n and therefore to its resolvent, Case 1 applies. Therefore, $C \notin \Pi$.

Case 4. $\bar{f}_i \in R$ and $i > n$. Assume without loss of generality that $\bar{f}_i \in R_1$. For $j \in [i..t]$ let the set X' contain x_j if $x_j \in R_1$ and contain \bar{x}_j if $x_j \notin R_1$. By applying resolve_{\exists} over an adequate subset of the clauses $\{F_j, F'_j \mid i \leq j \leq t\}$, the clause (f_i, X') can be obtained in a polynomial number of steps and be resolved with C to eliminate \bar{f}_i . This procedure can be applied to eliminate all \bar{f}_i literals from R_1 and thus enable reduce on the variables in X . By applying the same rewriting to C_2 eventually $\text{resolve}_{\exists\mathbb{L}}(C_1, p, C_2)$ transforms into $\text{resolve}_{\exists}(C_1 \setminus \{X, F_1\}, p, C_2 \setminus \{\bar{X}, F_2\})$, where $F_1 = \{\bar{f}_i \mid i > n \wedge \bar{f}_i \in C_1\}$ and $F_2 = \{\bar{f}_i \mid i > n \wedge \bar{f}_i \in C_2\}$. ■

For $\{\text{Q}, \text{QU}, \text{LQ}\}$ -resolution, we follow the assumption that universal reduction is performed whenever possible. If one allows postponing the reduction arbitrarily (as in the definition of QU-resolution in [40]), it will generalize the aforementioned proof systems and allow a larger number of sound refutations. In the sequel we call a refutation where the reduction of at least one universal variable has been postponed a *postponed refutation* and a clause that contains a universal variable which could be universally reduced, but is still present in at least one of its child clauses, a *postponed clause*. The following corollary from Proposition 3.3 shows that postponing cannot lead to shorter refutations in terms of the number of resolutions for any of the $\{\text{Q}, \text{QU}, \text{LQ}\}$ -resolution proof systems.

Corollary 3.1 Given a false QBF Φ , let Π be its shortest $\{\text{Q}, \text{QU}, \text{LQ}\}$ -refutation, and let Π^* be its shortest postponed $\{\text{Q}, \text{QU}, \text{LQ}\}$ -refutation. Then $|\Pi^*| \geq |\Pi|$.

Proposition 3.3 can be applied to all topologically first postponed clauses in a postponed refutation and therefore, the corollary follows. By Corollary 3.1, Theorems 3.5 and 3.6 hold for postponed QU-refutations as well.



3.3 New resolution proof systems

We saw in Section 3.2 that removing restriction imposed on quantification of pivot variables in Q-resolution lead to a stronger proof system. Allowing tautological clauses derivation and extending universal reduction rule from Q-resolution to apply for merged literals results into a stronger proof system as well. In this section we try to combine both of aforementioned techniques to obtain a yet stronger proof system for QBF.

3.3.1 Introducing LQU and LQU+

In this work we propose two additional resolution systems for QBF. The first, LQU-resolution, is defined as an extension of Q-resolution by adding both the **resolve_∀** and the **resolve_{∃L}** derivation rules. The second, LQU+-resolution, extends LQU-resolution by the new derivation rule **resolve_{∀L}** that allows proper long-distance resolutions under universally quantified pivots. The proof for soundness of **resolve_{∀L}** is similar to that of **resolve_{∃L}** rule stated by Theorem 3.3. Note that the index restriction imposed on **resolve_{∀L}** cannot be simplified to level restriction as for **resolve_{∃L}**, since the universal pivot may have the same level as a merged literal in the same proof step. The following example shows that relaxing the index restriction to the level restriction is unsound for **resolve_{∀L}**.

Example 3.4 Consider the true QBF

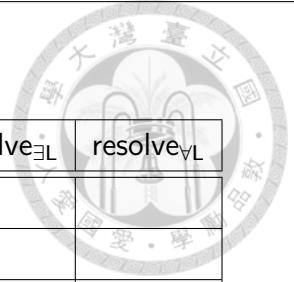
$$\Phi = \forall x \forall y \exists a. (x, y, a)_1 (\bar{x}, \bar{y}, a)_2 (x, \bar{y}, \bar{a})_3 (\bar{x}, y, \bar{a})_4.$$

The Skolem function $s_a = (x \leftrightarrow y)$ shows that Φ is true. Note that $\text{lev}(x) = \text{lev}(y)$, but $\text{idx}(x) < \text{idx}(y)$. If the index restriction is neglected, then the following unsound proof Π can be built.

$$\Pi = \left\{ \begin{array}{l} 1. \text{ clause}_5 = \text{resolve}_{\forall L}(\text{clause}_1, x, \text{clause}_2) = (y^*, a) \\ 2. \text{ clause}_6 = \text{resolve}_{\forall L}(\text{clause}_3, y, \text{clause}_4) = (x^*, \bar{a}) \\ 3. \text{ clause}_7 = \text{resolve}_{\exists}(\text{clause}_5, a, \text{clause}_6) = (x^*, y^*) \\ 4. \text{ clause}_{\square} = \text{reduce}(\text{clause}_7) = \square \end{array} \right\}$$

Note that the index restriction on x and y would disallow **resolve_{∀L}** step 2.

3.3. New resolution proof systems



	reduce	resolve \exists	resolve \forall	resolve \exists_L	resolve \forall_L
Q-resolution [29]	x	x			
QU-resolution [40]	x	x	x		
LQ-resolution [5]	x	x		x	
LQU-resolution	x	x	x	x	
LQU+-resolution	x	x	x	x	x

Table 3.1: Summary of Proof System Rules.

Table 3.1 compares the five proof systems discussed in this section by listing their derivation rules. In each line, the derivation rules for each proof system are marked by “x”. All proof systems are sound and refutationally complete for QBF. The completeness of LQU-resolution and LQU+-resolution follows from the completeness of Q-resolution. The soundness of LQU-resolution and LQU+-resolution can be proved similarly to soundness of LQ-resolution. We extend the definition of a $\{Q, QU, LQ\}$ -resolution proof Π to $\{LQU, LQU+\}$ -resolution by adding the corresponding derivation rules.

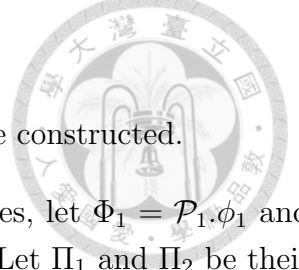
3.3.2 Superiority and Limitation of LQU and LQU+

Limitations of both KBKF-qu and KBKF-lq families can be established from a combined family KBKF-lqu that has exponential smallest proofs in both QU-resolution and LQ-resolution, but polynomial proofs in LQU-resolution. Proposition 3.4 and Theorem 3.7 below demonstrate bounds on the shortest proofs for combinations of QBF formulas.

Proposition 3.4 (cf. [21], Proposition 5) Given a false QBF $\Phi = \mathcal{P}.\phi$, a literal $e \in V_{\exists}$, an LQU-resolution proof Π for Φ , both $\Phi_{\upharpoonright\{e\}}$ and $\Phi_{\upharpoonright\{\bar{e}\}}$ are false QBFs and Π can be modified in polynomial time with respect to its size to obtain a new proof $\Pi_{\upharpoonright e}$ (respectively $\Pi_{\upharpoonright \bar{e}}$) deriving \square from $\Phi_{\upharpoonright\{e\}}$ (respectively $\Phi_{\upharpoonright\{\bar{e}\}}$).

This proposition extends Proposition 5 in [21] by allowing e to have an arbitrary quantifier level and allowing the proof to contain **resolve \forall** steps. The extension is sound, since the proof in [21] is independent of the quantifier levels of existential variables and can also be incorporated with **resolve \forall** and **resolve \exists_L** rules. The same result for Q-resolution has been proposed in [24]. By Proposition 3.4 also $\Phi_{\upharpoonright\sigma}$ and

3.3. New resolution proof systems



Π_{σ} for any assignment σ to existential variables of Φ can be constructed.

Theorem 3.7 Given two disjoint sets V_1 and V_2 of variables, let $\Phi_1 = \mathcal{P}_1.\phi_1$ and $\Phi_2 = \mathcal{P}_2.\phi_2$ be two false QBFs over V_1 and V_2 , respectively. Let Π_1 and Π_2 be their respective shortest LQU-resolution proofs. Let $\Phi = \exists a \mathcal{P}_1 \mathcal{P}_2.(\phi_1 \vee a) \wedge (\phi_2 \vee \bar{a})$, where $a \notin V_1 \cup V_2$, and for $i \in \{1, 2\}$, $(\phi_i \vee a)$ stands for $\{C \cup \{a\} \mid C \in \phi_i\}$. Then Φ is false and the size of its shortest LQU-refutation is $|\Pi_1| + |\Pi_2| + 1$.

Proof By following the resolution steps of Π_1 on the clauses of $(\phi_1 \vee a)$ we retrieve the clause (a) , by following the resolution steps of Π_2 , on $(\phi_2 \vee \bar{a})$ we retrieve (\bar{a}) , and resolving the two unit clauses results in \square . Thus an LQU-resolution proof Π with $|\Pi| = |\Pi_1| + |\Pi_2| + 1$ is constructed. Let Π be any LQU-resolution proof for Φ and let $\Pi_{\{a\}}$ be the proof generated for $\Phi_{\{a\}}$ as by Proposition 3.4 and let n_1 (resp. n_2) be the number of resolution steps in Π under any pivot $p \in V_1$ (resp. any pivot $p \in V_2$). By construction, $\Phi_{\{a\}} = \Phi_2$ and therefore $n_2 \geq |\Pi_{\{a\}}| \geq |\Pi_2|$. The dual case holds for $\Pi_{\{\bar{a}\}}$, resulting in $n_1 \geq |\Pi_{\{\bar{a}\}}| \geq |\Pi_1|$. Finally, in a derivation of \square from Φ , there must be at least one **resolve** _{\exists} with pivot variable a . As $V_1 \cap V_2 = \emptyset$ and $a \notin V_1 \cup V_2$ we conclude $|\Pi| \geq |\Pi_1| + |\Pi_2| + 1$. Note that if V_1 and V_2 are not disjoint, then in a similar way we can only prove a weaker bound $|\Pi| \geq \max(|\Pi_1|, |\Pi_2|) + 1$. ■

Definition 3.5 (KBKF-lqu family) For $t > 1$, let $\mathcal{P}^q.\phi^q$ be the t^{th} member of the KBKF-qu family over variable set V^q , and let $\mathcal{P}^l.\phi^l$ be the t^{th} member of the KBKF-lq family over variable set V^l , where $V^q \cap V^l = \emptyset$. Let a be a fresh variable with $a \notin V^q \cup V^l$. The t^{th} member KBKF-lqu[t] in the KBKF-lqu family is defined as $\exists a \mathcal{P}^q \mathcal{P}^l.(\phi^q \vee a) \wedge (\phi^l \vee \bar{a})$.

Corollary 3.2 For $t > 1$, the smallest proofs for KBKF-lqu[t] are polynomial for LQU-resolution, but are exponential for LQ-resolution and exponential for QU-resolution (based on Theorem 3.2 in [29]).

Whether the LQU+-resolution calculus has an exponential separation with respect to LQU remains an open problem. The following example, however, shows how LQU+-resolution can be more beneficial than LQU-resolution in some cases.

Example 3.5 Consider the false QBF

$$\Phi = \exists a \forall x \forall y \exists b.(a, x, b)_1(\bar{a}, \bar{x}, b)_2(x, y, \bar{b})_3(\bar{x}, \bar{y}, \bar{b})_4.$$

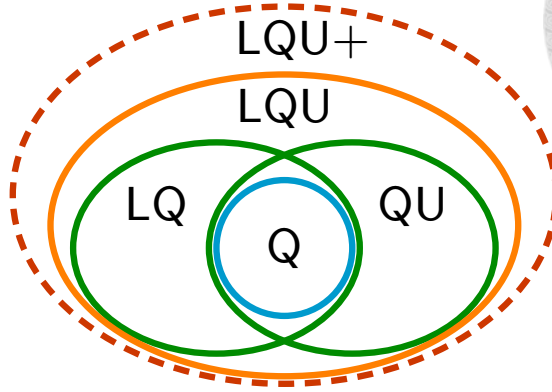


Figure 3.8: The diagram for Resolution Proof systems relationship.

Notice that Φ is similar to QBF in Example 3.3, and has Herbrand functions $h_y = h_x = a$. Further, any LQU-resolution proof of the falsity of Φ cannot contain any of the steps resolve_{\forall} and $\text{resolve}_{\exists\text{L}}$ (relevant to the derivation of an empty clause). There exists, however, an LQU+-resolution proof Π , which contains a $\text{resolve}_{\forall\text{L}}$ step.

$$\Pi = \left\{ \begin{array}{l} 1. \text{ clause}_5 = \text{resolve}_{\exists\text{L}}(\text{clause}_1, a, \text{clause}_2) = (x^*, b) \\ 2. \text{ clause}_6 = \text{resolve}_{\forall\text{L}}(\text{clause}_3, x, \text{clause}_4) = (y^*, \bar{b}) \\ 3. \text{ clause}_7 = \text{resolve}_{\exists}(\text{clause}_5, b, \text{clause}_6) = (x^*, y^*) \\ 4. \text{ clause}_{\square} = \text{reduce}(\text{clause}_7) = \square \end{array} \right\}$$

The complete picture of known relationships between Q-resolution proof system and its derivatives is shown in Figure 3.8.

Finally, it is interesting to see by the following theorem that suitable Herbrand functions can exponentially reduce a resolution proof for a false QBF.

Theorem 3.8 Let $\Phi = Q_1 v_1 \dots Q_{i-1} v_{i-1} \forall v_i Q_{i+1} v_{i+1} \dots Q_k v_k. \phi$, with $Q_i \in \{\exists, \forall\}$ for $i \in [1..k]$, be an arbitrary QBF. If there exists a function $\mathcal{F}(v_1, \dots, v_{i-1})$ such that the QBF $\Phi' = Q_1 v_1 \dots Q_{i-1} v_{i-1} \exists v_i Q_{i+1} v_{i+1} \dots Q_k v_k. \phi \wedge (v_i \leftrightarrow \mathcal{F})$ is false then also Φ is false.

Proof Let $\phi' = \phi \wedge (v_i \leftrightarrow \mathcal{F})$, let $\mathcal{H}' = \{h'_{vj} \mid Q_j = \forall \wedge 1 < j < k \wedge j \neq i\}$ be the set of Herbrand functions for Φ' and let $\mathcal{H} = \mathcal{H}' \cup h_{vi}$, with $h_{vi} = \mathcal{F}$. Then $\phi'[\mathcal{H}'] = \phi[\mathcal{H}] \wedge (\mathcal{F} \leftrightarrow \mathcal{F}) = \phi[\mathcal{H}]$. Since $\phi'[\mathcal{H}']$ is unsatisfiable, so must be $\phi[\mathcal{H}]$. Thus by definition \mathcal{H} is a Herbrand model for Φ . ■

3.3. New resolution proof systems

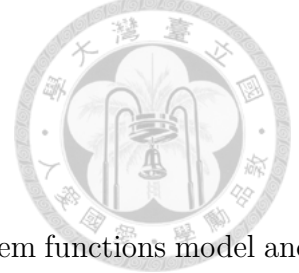
Corollary 3.3 Consider QBF KBKF-exp[t] = $\mathcal{P}\phi$. For $i \in [1..t]$ change quantifiers of y_i variables to \exists in \mathcal{P} and add the clauses $(y_i, \overline{x_i}) \wedge (\overline{y_i}, x_i)$ to ϕ . All y_i variables can be now removed by corresponding resolve_{\exists} steps from ϕ . To retrieve \square , the same polynomial QU-resolution as for the KBKF-qu[t] can be used.



Chapter 4

Semantic certificates and QBF

Limitedly syntactic QBF certificates are not directly useful for synthesis applications, while semantic certificates are. Search-based QBF solvers are not capable however to produce Skolem/Herbrand functions directly, and can only output resolution-based refutations to certify their answer. In this chapter we give an overview of various scenarios of semantic certificates derivation. In Section 4.1 we illustrate some of the QBF solving techniques that are based on direct construction of Skolem functions as a part of the solving process. These solving methods are not as well scalable as search-based (QDPLL) algorithms. Furthermore, despite the symmetry between true and false QBFs, up to the date there are no tools that could directly produce Herbrand functions for false QBFs. To resolve the aforementioned problems we establish the relation between semantic certificates and syntactic certificates. In Section 4.2 we propose a linear time complexity approach (with respect to the resolution proof size) of Skolem and Herbrand functions extraction from cube and clause Q-resolution proofs, respectively, and experimentally prove that introduced approach outperforms direct construction methods. In Section 4.3 we characterize the flexibilities of the proposed construction method that could be utilized to optimize the extracted semantic certificate quality for synthesis applications, and confirm the usefulness of the proposed techniques with fluent experimental evaluation. Finally in Section 4.4 we extend the construction algorithm proposed in Section 4.2 to a polynomial approach for (counter)model construction from LQU extension of Q-resolution proof system.



4.1 Models and countermodels

Yielding the definitions of semantic certificates in form of Skolem functions model and Herbrand functions countermodel, we illustrate some of the existing direct methods for their construction in this section.

4.1.1 Skolemization

On one hand QBF is a special case of an *effective propositional logic*, where quantification over function symbols is not allowed. On the other, however, we could degrade its complexity and substitute each existential variable with unique fresh function symbol, quantified outermost. The resulting formula, which consists of only two quantification levels (first existential and second universal) and has existential variables being replaced by their respective function symbols, is said to be in a *Skolem normal form* [39]. The process of converting QBF into a Skolem normal form is called *Skolemization*. True QBF is logically equivalent to its representation in Skolem normal form in a sense that quantified function symbols correspond to Skolem functions of the original QBF. Therefore truth of a QBF can be established by truth of its Skolem normal form representation. Please note that Skolemization procedure for false QBFs does not result into logically equivalent formula (rather just satisfiability preserving), since the notion of Herbrand functions is not very well defined in this case as shall be evidenced by Chapter 5. In Chapter 5 we will also see that Skolemization is very related to Henkin quantifiers and dependency quantified Boolean formulas. In the next subsection we shall discuss how Skolemization can be used for QBF solving and Skolem functions derivation.

4.1.2 Model generation using Skolemization

Using an example we elaborate the details of Skolemization procedure. Consider the true QBF from Subsection 2.3.1

$$\begin{aligned}\Phi &= \exists a b \forall x \exists c \forall y. \phi(a, b, c, x, y), \\ \phi(a, b, c, x, y) &= (a + b + x + y)(\bar{b} + \bar{x} + \bar{c} + \bar{y})(\bar{a} + \bar{x})(\bar{b} + x + c)\end{aligned}$$

4.1. Models and countermodels



Its corresponding Skolem normal form is

$$\Phi_S = \exists f_a f_b f_c(x) \forall xy. \phi(f_a, f_b, f_c, x, y).$$

Semantically Φ_S is precisely the same as Φ (taking into account that we already know that Φ is true). In order to decide Φ_S [9] proposed to use a variant of Shannon expansion theorem introduced in Section 2.1. Consider Φ_S and note that functions f_a and f_b do not depend on any universal variables (i.e. they are just constants), therefore we can substitute them with constant auxiliary variables v_a and v_b respectively. Function f_c is only a function of x , therefore we could represent it as $f_c = x \wedge v'_c \vee \bar{x} \wedge v''_c = (\bar{x} + v'_c)(x + v''_c)$, where v'_c and v''_c are auxiliary propositional variables as well. Now Φ_S could be transformed into an equisatisfiable QBF formula

$$\begin{aligned} \Phi_{sk} &= \exists v_a v_b v'_c v''_c \forall xy. \phi(v_a, v_b, x \wedge v'_c \vee \bar{x} \wedge v''_c, x, y) \\ &= (v_a + v_b + x + y)(\bar{v}_b + \bar{x} + x\bar{v}'_c + \bar{x}\bar{v}''_c + \bar{y})(\bar{v}_a + \bar{x})(\bar{v}_b + x + xv'_c + \bar{x}v''_c) \\ &= (v_a + v_b + x + y)(\bar{v}_b + \bar{x} + \bar{v}'_c + \bar{y})(\bar{v}_a + \bar{x})(\bar{v}_b + x + v''_c). \end{aligned}$$

Note that variables x and y are inner most quantified variables now and therefore can be universally reduced, resulting into a propositional formula

$$\Phi_{prop} = [\exists v_a v_b v'_c v''_c].(v_a + v_b)(\bar{v}_b + \bar{v}'_c)(\bar{v}_a)(\bar{v}_b + v''_c).$$

It is important to mention that formula Φ_{prop} is propositional by no coincidence. Propositionality of Φ_{prop} will be the case for any QBF Φ transformed into Φ_{prop} by aforementioned procedure, as was shown in Section 3.1 of [9].

In our example Φ_{prop} is satisfiable, with the unique satisfying assignments $\{v_a = 0; v_b = 1; v'_c = 0; v''_c = 1\}$. Therefore QBF Φ evaluates to true, and corresponding to variables a , b and c Skolem functions are $F_a = 0$, $F_b = 1$ and $F_c = x \wedge 0 \vee \bar{x} \wedge 1 = \bar{x}$, respectively. Note that obtained Skolem functions are exactly the same as those found in Subsection 2.3.2.

Skolemization approach is useful in practice since it could directly generate Skolem functions certificate, it however is less scalable than search-based solving methods.

4.2. Extraction of Skolem and Herbrand functions from Q-resolution proofs

Scalability issues of Skolemization rise from the step of reducing Φ_{sk} to Φ_{prop} . If number of universal variables is large in QBF, it is problematic (in fact worst case exponential) to write Shannon expansion of a particular existential variable. We shall confirm this in experimental Subsection 4.2.3.

Equivalent (symmetrical to Skolemization) procedure is possible for Herbrand function derivation for false QBFs, it however, up to the authors knowledge, was not implemented in any existing QBF solver. This is also a drawback for Skolemization based solvers as they can not provide semantic certificates for false QBFs (issue is also discussed in Subsection 4.2.3).

4.2 Extraction of Skolem and Herbrand functions from Q-resolution proofs

Modern QBF solvers evaluation showed that Skolemization-based QBF solvers are less efficient than search-based solvers. As it was mentioned in the beginning of this chapter search-based solvers, however, could not produce semantic certificates. On the other hand they can certify their answer with Q-resolution proofs by construction approach that was discussed in Section 3.1. In this section we introduce a linear approach to extract Skolem and Hebrand functions from cube and clause Q-resolution proofs respectively. By the end of the section we provide a full experimental evaluation of the proposed method and discuss results in details.

4.2.1 Counter-model Construction from Clause-Resolution Proofs

Aim of this section is to construct (counter)models from Q-resolution proofs, therefore we do not permit long-distance resolution in the discussed below Q-refutations.

Before delving into the main construction, we first define the following formula structure.

4.2. Extraction of Skolem and Herbrand functions from Q-resolution proofs

Definition 4.1 A *Right-First-And-Or (RFAO)* formula φ is recursively defined by

$$\varphi ::= clause \mid cube \mid clause \wedge \varphi \mid cube \vee \varphi,$$

where the symbol “ $::=$ ” is read as “can be” and symbol “ \mid ” as “or”.

Note that the formula is constructed in order from left to right. Due to the particular building rule of an RFAO formula with priority going to the right, we save on parentheses to enhance readability. For example, formula

$$\begin{aligned} \varphi &= clause_1 \wedge clause_2 \wedge cube_3 \vee clause_4 \wedge cube_5 \vee cube_6 \\ &= (clause_1 \wedge (clause_2 \wedge (cube_3 \vee (clause_4 \wedge (cube_5 \vee cube_6))))) \end{aligned}$$

Recall, that we sometimes omit expressing the conjunction symbol “ \wedge ” and interchangeably use “ $+$ ” for “ \vee ” in a formula.

In our discussion we shall call a clause/cube in an RFAO formula as a *node* of the formula, and omit a node’s subsequent operator, which can be uniquely determined. Note that the ambiguity between a single-literal clause and a single-literal cube does not occur in an RFAO formula as the clause-cube attributes are well specified in our construction.

The RFAO formula has two important properties (which will be crucial in proving Theorem 4.3):

1. If $node_i$ under some (partial) assignment of variables becomes a validated clause (denoted 1-clause) or falsified cube (denoted 0-cube), then we can effectively remove $node_i$ (if it is not the last) from the formula without further valuating it.
2. If $node_i$ becomes a falsified clause (denoted 0-clause) or validated cube (denoted 1-cube), then we need not further valuate (namely, can remove) all other nodes with index greater than i .

Below we elaborate how to construct the countermodel expressed by the RFAO formula from a clause-resolution proof Π of a false QBF Φ . Along with a proof Π , we consider its graph (DAG) representation $G_\Pi(V_\Pi, E_\Pi)$, where a vertex $v \in V_\Pi$ corresponds to a clause $v.clause$ obtained in the resolution steps of Π and a directed

4.2. Extraction of Skolem and Herbrand functions from Q-resolution proofs

edge $(u, v) \in E_\Pi \subseteq V_\Pi \times V_\Pi$ from the *parent* u to the *child* v indicates that $v.clause$ results from $u.clause$ through either resolution or \forall -reduction. The clauses of Π can be partitioned into three subsets: those in Φ_{mtx} , those resulting from resolution, and those from \forall -reduction. Let V_M , V_S , and V_D denote their respective corresponding vertex sets. So $V_\Pi = V_M \cup V_S \cup V_D$. Note that in G_Π a vertex in V_M has no incoming edges and is a *source* vertex; a vertex in V_S has two incoming edges from its two parent vertices; a vertex in V_D has one incoming edge from its parent vertex. On the other hand, there can be one or more *sink* vertices, which have no outgoing edges. Since the final clause of Π is an empty clause, the graph G_Π must have the corresponding sink vertex.

Intuition behind the construction approach stems from the following observations. First, if $V_D = \emptyset$, then the quantifier-free formula Φ_{mtx} is unsatisfiable by itself, and so is Φ . Since there exists an ordinary resolution proof, which involves no \forall -reduction, any functional interpretation on the universal variables forms a legitimate countermodel to Φ .

Second, if $V_S = \emptyset$, then Φ_{mtx} must contain a clause consisting of only universal variables. With only \forall -reduction, Φ can be falsified. Without loss of generality, assume this clause is $(l_1 \vee \dots \vee l_k)$. Then letting the Herbrand function of $\text{var}(l_i)$ be

$$F_H[\text{var}(l_i)] = \begin{cases} 0 & \text{if } l_i = \text{var}(l_i), \text{ and} \\ 1 & \text{if } l_i = \neg \text{var}(l_i), \end{cases}$$

for $i = [1, \dots, k]$, forms a countermodel of Φ . (Herbrand functions for universal variables not in the clause are unconstrained.)

Finally, we discuss the general case where V_D and V_S are non-empty. Every clause $w.clause$ of Π with $w \in V_S$ is implied by the conjunction $u.clause \wedge v.clause$ with $(u, w), (v, w) \in E_\Pi$. (That is, the clause resulting from resolution is *unconditionally* implied by the conjunction of its parent clauses.) Even if pivot variable of the corresponding resolution was universally quantified, the implication would still hold. So the implication preserves regardless of Φ_{pfx} . On the other hand, a clause $v.clause$ of Π with $v \in V_D$ is not directly implied by $u.clause$ with $(u, v) \in E_\Pi$. (That is, the clause resulting from \forall -reduction is *conditionally* implied by its parent clause.) Nevertheless Φ_{mtx} and $\Phi_{\text{mtx}} \wedge v.clause$ are equisatisfiable under Φ_{pfx} .

4.2. Extraction of Skolem and Herbrand functions from Q-resolution proofs

To characterize the conditions for an implication (especially between two clauses involved in a \forall -reduction step) to hold, we give the following definition.

Definition 4.2 Let $\alpha : X \rightarrow \{0, 1\}$ be a full assignment on variables X . Given two (quantifier-free) formulae ϕ_1 and ϕ_2 over variables X , if the implication $\phi_1 \rightarrow \phi_2$ holds under α , then we say that ϕ_2 is α -*implied* by ϕ_1 , and ϕ_1 α -*implies* ϕ_2 .

For a resolution proof of a false QBF Φ , when we say a clause is α -*implied*, we shall mean it is α -implied by its parent clause or by the conjunction of its parent clauses depending on whether the clause results from \forall -reduction or resolution. A clause resulting from resolution is surely α -implied for every α , but a clause resulting from \forall -reduction may not be α -implied for some α . We further say that a clause C is α -*inherited* if all of its ancestor clauses (except for the clauses of the source vertexes, which have no parent clauses and are undefined under α -implication) and itself are α -implied. Clearly, if C is α -inherited, then $\Phi_{\text{mtx}}|_{\alpha} = (\Phi_{\text{mtx}} \wedge C)|_{\alpha}$.

For a false QBF Φ over variables $X = X_{\exists} \cup X_{\forall}$, let the assignment $\alpha : X \rightarrow \{0, 1\}$ be divided into $\alpha_{\exists} : X_{\exists} \rightarrow \{0, 1\}$ and $\alpha_{\forall} : X_{\forall} \rightarrow \{0, 1\}$. To construct the Herbrand-function countermodel, our goal is to determine α_{\forall} for every α_{\exists} such that the empty clause of the resolution proof is α -inherited, or there exists an α -inherited clause C with $C|_{\alpha} = 0$. Therefore, for every assignment α_{\exists} , Φ implies false. That is, such α_{\forall} provides a countermodel to Φ .

Figure 4.1 sketches the countermodel construction algorithm, where the Herbrand functions are computed in RFAO formulae, each of which is stored as an (ordered) array of nodes. Before proving the correctness of the algorithm, we take the following example to illustrate the computation.

Example 4.1 Let Φ be a false QBF and Π be its resolution proof of unsatisfiability

4.2. Extraction of Skolem and Herbrand functions from Q-resolution proofs

Countermodel-Construct($\Phi, G_\Pi(V_\Pi, E_\Pi)$)

input: a false QBF Φ and its clause-resolution DAG $G_\Pi(V_\Pi, E_\Pi)$
output: a countermodel in RFAO formulae
begin
01 **foreach** universal variable x of Φ
02 RFAO_node_array[x] := \emptyset ;
03 **foreach** vertex v of G_Π in topological order
04 **if** $v.clause$ resulting from \forall -reduction on $u.clause$, i.e., $(u, v) \in E_\Pi$
05 $v.cube := \neg(v.clause)$;
06 **foreach** universal variable x reduced from $u.clause$ to get $v.clause$
07 **if** x appears as positive literal in $u.clause$
08 **push** $v.clause$ to RFAO_node_array[x];
09 **else if** x appears as negative literal in $u.clause$
10 **push** $v.cube$ to RFAO_node_array[x];
11 **if** $v.clause$ is the empty clause
12 **foreach** universal variable x of Φ
13 simplify RFAO_node_array[x];
14 **return** RFAO_node_array's;
end

Figure 4.1: Algorithm: Countermodel Construction from Q-refutations.

as below.

$$\begin{aligned}
\Phi_{\text{pfx}} &= \exists a \forall x \exists b \forall y \exists c \\
\Phi_{\text{mtx}} &= (a + b + y + c)(a + x + b + y + \bar{c})(x + \bar{b})(\bar{y} + c) \\
&\quad (\bar{a} + \bar{x} + b + \bar{c})(\bar{x} + \bar{b})(a + \bar{b} + \bar{y}) \\
\Pi &= \left\{ \begin{array}{l} 1. \text{ clause}_8 = \text{resolve}(\text{clause}_1, \text{clause}_2) \\ 2. \text{ clause}_9 = \text{resolve}(\text{clause}_3, \text{clause}_8) \\ 3. \text{ clause}_{10} = \text{resolve}(\text{clause}_4, \text{clause}_5) \\ 4. \text{ clause}_{11} = \text{resolve}(\text{clause}_{10}, \text{clause}_6) \\ 5. \text{ clause}_\square = \text{resolve}(\text{clause}_{11}, \text{clause}_9) \end{array} \right\}
\end{aligned}$$

Note that the \forall -reduction steps are omitted in Π for notation simplicity. They are shown in Figure 4.2, where clauses are indexed by the subscript numbers and the \forall -reduction steps are indexed by the parenthesized numbers indicating their derivation order.

4.2. Extraction of Skolem and Herbrand functions from Q-resolution proofs

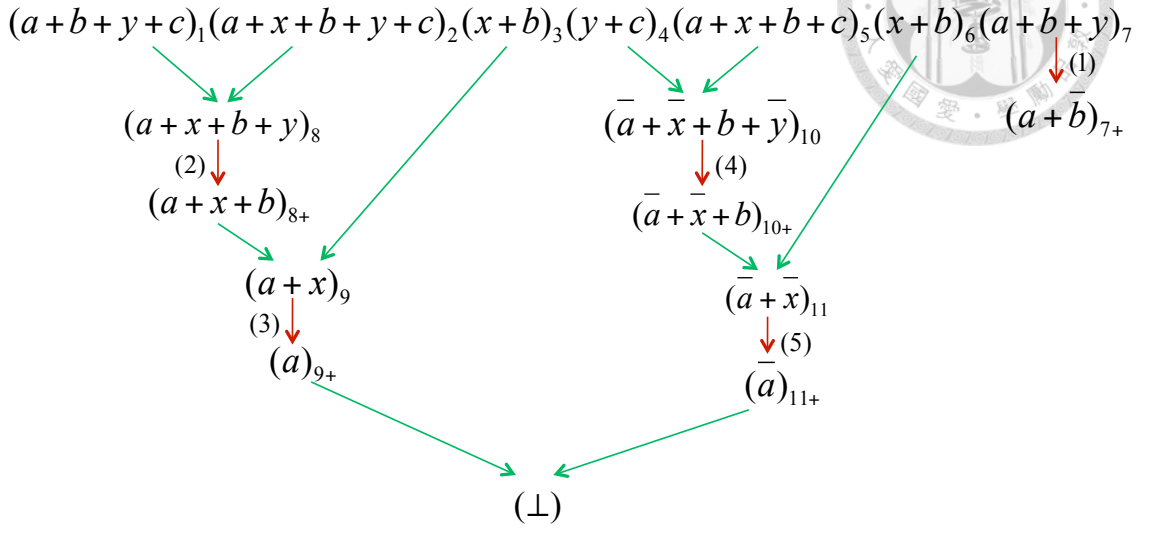


Figure 4.2: DAG of resolution proof Π .

Following the steps of algorithm *Countermodel-Construct* in Figure 4.1, the RFAO node-array contents after each \forall -reduction step of the proof in Figure 4.2 are listed in order of appearance in Figure 4.3. Resultant Herbrand functions for variables x and y are


$$\begin{aligned} F_H[x] &= (a) \wedge (a) = a, \text{ and} \\ F_H[y] &= (\neg ab) \vee ((a \vee x \vee b) \wedge (ax \neg b)), \end{aligned}$$

respectively. Note that computed $F_H[y]$ depends on variable x , which can always be eliminated by substituting $F_H[x]$ for x in $F_H[y]$. In fact, keeping such dependency may be beneficial as the countermodel can be represented in a multi-level circuit format with shared logic structures. Moreover, observe that clause 7, namely $(a \vee \neg b \vee \neg y)$, is not involved in the resolution steps leading to the empty clause. Its existence is optional in constructing the countermodel, and can be treated as *don't care* for countermodel simplification. It can be verified that, for every assignment to variables a , b , and c , formula Φ_{mtx} with variables x and y substituted with $F_H[x]$ and $F_H[y]$, respectively, is false.

The correctness of algorithm *Countermodel-Construct* of Figure 4.1 is asserted below.

Theorem 4.1 Given a false QBF Φ and a DAG G_Π corresponding to its resolution

4.2. Extraction of Skolem and Herbrand functions from Q-resolution proofs



0.	$x :$	$\left[\right]$	$y :$	$\left[\right]$
1.	$x :$	$\left[\right]$	$y :$	$\left[\text{cube}(\neg ab) \right]$
2.	$x :$	$\left[\right]$	$y :$	$\left[\begin{array}{l} \text{cube}(\neg ab), \\ \text{clause}(a \vee x \vee b) \end{array} \right]$
3.	$x :$	$\left[\text{clause}(a) \right]$	$y :$	$\left[\begin{array}{l} \text{cube}(\neg ab), \\ \text{clause}(a \vee x \vee b) \end{array} \right]$
4.	$x :$	$\left[\text{clause}(a) \right]$	$y :$	$\left[\begin{array}{l} \text{cube}(\neg ab), \\ \text{clause}(a \vee x \vee b), \\ \text{cube}(ax \neg b) \end{array} \right]$
5.	$x :$	$\left[\begin{array}{l} \text{clause}(a), \\ \text{cube}(a) \end{array} \right]$	$y :$	$\left[\begin{array}{l} \text{cube}(\neg ab), \\ \text{clause}(a \vee x \vee b), \\ \text{cube}(ax \neg b) \end{array} \right]$

Figure 4.3: Contents of RFAO node arrays.

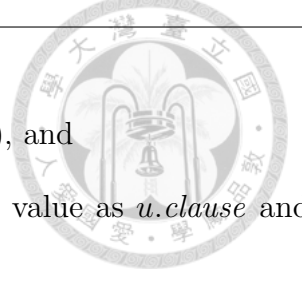
proof Π of unsatisfiability, algorithm *Countermodel-Construct* (Φ, G_Π) produces a correct countermodel for universal variables of Φ .

Proof We show that under every assignment α_\exists to existential variables of Φ , our constructed countermodel always induces some α_\forall such that $\Phi_{\text{mtx}}|_\alpha = 0$. There are two possible cases under every such α .

First, assume every clause $v.\text{clause}$ with $v \in V_D$ is α -implied. Then the empty clause must be α -inherited because other clauses resulting from resolution are always α -implied. Therefore $\Phi_{\text{mtx}}|_\alpha = 0$.

Second, assume not every clause $v.\text{clause}$ with $v \in V_D$ is α -implied. Let $C_{\alpha.\text{violate}}$ be the set of all such clauses violating α -implication. Suppose $v.\text{clause} \in C_{\alpha.\text{violate}}$ is obtained by \forall -reduction from $u.\text{clause}$ with $(u, v) \in E_\Pi$ on some universal variables. Let $C_{u \setminus v}$ denote the subclause of $u.\text{clause}$ consisting of exactly the reduced literals in the \forall -reduction leading to $v.\text{clause}$. Then $v.\text{clause}$ must satisfy the criteria

4.2. Extraction of Skolem and Herbrand functions from Q-resolution proofs



1. $v.\text{clause}|_\alpha = 0$ (otherwise $v.\text{clause}$ would be α -implied), and
2. $C_{u \setminus v}|_{\alpha_v} = 1$ (otherwise $v.\text{clause}$ would have the same value as $u.\text{clause}$ and thus be α -implied).

It remains to show that, even if $C_{\alpha.\text{violate}}$ is non-empty, there still exists some α -inherited clause C with $C|_\alpha = 0$, i.e., an induced empty clause under α .

Notice that algorithm *Countermodel-Construct* processes G_Π in a topological order, meaning that a clause in the resolution proof is processed only after all of its resolution ancestor clauses are processed. Now we consider all clauses $v.\text{clause}$ with $v \in V_D$ in the topological order under the assignment α . Let $v'.$ clause be the first clause encountered with $v'.$ clause $|_\alpha = 0$. (If there is no such $v'.$ clause under α , then it corresponds to the situation analyzed in the first case.) For every universal variable x being reduced from the parent clause $u'.$ clause of $v'.$ clause, i.e., $(u', v') \in E_\Pi$, we examine its corresponding `RFAO_node_array` $[x]$. Suppose v' is the i th enumerated vertex that results from \forall -reduction involving reducing variable x . By the aforementioned two properties of the RFAO formula and by the way how `RFAO_node_array` $[x]$ is constructed, we know that the Herbrand function value of $F_H[x]$ under α is not determined by the first $i - 1$ nodes, but by the i th node of `RFAO_node_array` $[x]$. In addition, the function value $F_H[x]$ makes the literal of variable x in clause $C_{u' \setminus v'}$ evaluate to false. Because every literal in $C_{u' \setminus v'}$ is evaluated to false, we have $u'.$ clause $|_\alpha = 0$ and thus $v'.$ clause is α -implied. Moreover, since $v'.$ clause is the first clause encountered with $v'.$ clause $|_\alpha = 0$, all its ancestor clauses must be α -implied. So $v'.$ clause is α -inherited, and thus $\Phi_{\text{mtx}}|_\alpha = 0$.

Because every assignment α_\exists together with the corresponding induced assignment α_\forall makes $\Phi_{\text{mtx}}|_\alpha = 0$, Herbrand functions computed by algorithm *Countermodel-Construct* form a correct countermodel to Φ . ■

Proposition 4.1 Given a false QBF Φ and its resolution proof of unsatisfiability, let $F_H[x]$ be the Herbrand function computed by algorithm *Countermodel-Construct* for universal variable x in Φ . Then $F_H[x]$ refers to some variable y in Φ only if $\ell(y) < \ell(x)$.

Note that by above strict inequality, Proposition 4.1 asserts that no cyclic dependency arises among the computed Herbrand functions.

4.2. Extraction of Skolem and Herbrand functions from Q-resolution proofs

Proposition 4.2 Given a false QBF and its resolution proof of unsatisfiability, algorithm *Countermodel-Construct* computes the countermodel in time linear with respect to the proof size.

Proposition 4.3 The RFAO formula size (in terms of nodes) for each universal variable computed by algorithm *Countermodel-Construct* is upper bounded by the number of \forall -reduction steps in the resolution proof.

Resolution proofs provided by search-based QBF solvers often contain (redundant) resolution steps unrelated to yielding the final empty clause. Algorithm *Countermodel-Construct* works for resolution proofs with and without redundant steps. Since a highly redundant proof may degrade the performance of the algorithm, it may be desirable to trim away redundant parts before countermodel construction. On the other hand, as illustrated in Example 4.1, it may be possible to exploit the redundancy for countermodel simplification.

4.2.2 Model Construction from Cube-Resolution Proofs

Key ideas from previous sections, shown for clause Q-resolution proofs for false QBFs, could be similarly applied to cube Q-resolution proofs for true QBFs. Example 4.2 and Figure 4.4 show QBF expressed in PDNF form and corresponding cube Q-resolution proof.

Example 4.2 Consider the following QBF in PDNF $\Phi = \forall x \exists a \forall y \exists b. (xayb + xa\bar{y} + \bar{x}a)$. It is true, as is evidenced by Skolem-function model $\{F_a = x, F_b = 1\}$, or its cube Q-resolution proof depicted in Figure 4.4.

The discussion about countermodel construction can be easily extended under the duality principle to model construction of a true QBF from its cube-resolution proof of satisfiability as shown below.

Recall, that cube resolution can be defined as dual to the clause resolution. In contrast to clause resolution, where a resolvent is implied by the conjunction of its parent resolving clauses, a resolvent in cube resolution implies the disjunction of its parent resolving cubes. So if the empty cube is deduced through a sequence of cube resolutions, then original formula is implied to be true. In fact, complementing

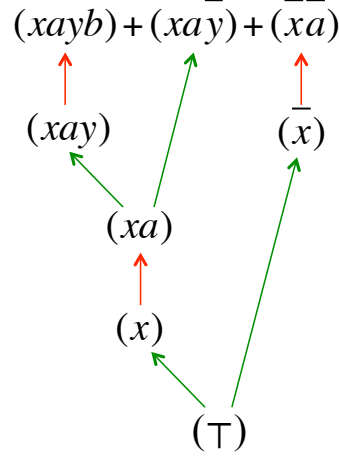


Figure 4.4: Cube Q-resolution proof for QBF Φ in PDNF form from Example 4.2.

a cube-resolution proof of a true QBF by negating each cube in the proof to a corresponding clause yields a clause-resolution proof of its negated false QBF, and vice versa.

Following a similar strategy to algorithm *Countermodel-Construct*, algorithm *Model-Construct* in Figure 4.5 computes Skolem functions from the resolution proof Π of a true QBF Φ by making the empty cube α -imply Φ_{mtx} under every α . Notice that it processes G_Π in a topological order, meaning that a cube in the resolution proof is processed only after all of its parent resolving cubes are processed. The correctness of the algorithm can be established in a way similar to Theorem 4.3 and its proof is omitted.

The following example illustrates how algorithm *Model-Construct* works.

Example 4.3 Consider true QBF $\Psi = \neg\Phi$ for Φ being the QBF of Example 4.1 and its resolution proof Π' shown below with \exists -reduction steps omitted.

$$\begin{aligned}\Psi_{\text{pfx}} &= \forall a \exists x \forall b \exists y \forall c \\ \Psi_{\text{mtx}} &= (\bar{a}\bar{b}\bar{y}\bar{c}) + (\bar{a}x\bar{b}\bar{y}c) + (\bar{x}b) + (y\bar{c}) + (ax\bar{b}c) + (xb) + (\bar{a}by)\end{aligned}$$

4.2. Extraction of Skolem and Herbrand functions from Q-resolution proofs

Model-Construct($\Phi, G_{\Pi}(V_{\Pi}, E_{\Pi})$)

input: a true QBF Φ and its cube-resolution DAG $G_{\Pi}(V_{\Pi}, E_{\Pi})$
output: a model in RFAO formulae

begin

```

01  foreach existential variable  $x$  of  $\Phi$ 
02    RFAO_node_array[ $x$ ] :=  $\emptyset$ ;
03  foreach vertex  $v$  of  $G_{\Pi}$  in topological order
04    if  $v.cube$  resulted from  $\exists$ -reduction on  $u.cube$ , i.e.,  $(u, v) \in E_{\Pi}$ 
05       $v.clause := \neg(v.cube)$ ;
06      foreach existential variable  $x$  reduced from  $u.cube$  to get  $v.cube$ 
07        if  $x$  appears as positive literal in  $u.cube$ 
08          push  $v.cube$  to RFAO_node_array[ $x$ ];
09        else if  $x$  appears as negative literal in  $u.cube$ 
10          push  $v.clause$  to RFAO_node_array[ $x$ ];
11  if  $v.cube$  is the empty cube
12    foreach existential variable  $x$  of  $\Phi$ 
13      simplify RFAO_node_array[ $x$ ];
14  return RFAO_node_array's;
end

```

Figure 4.5: Algorithm: Model Construction from Q-consensus proofs.

$$\Pi' = \left\{ \begin{array}{l} 1. \text{cube}_8 = \text{resolve}(\text{cube}_1, \text{cube}_2) \\ 2. \text{cube}_9 = \text{resolve}(\text{cube}_3, \text{cube}_8) \\ 3. \text{cube}_{10} = \text{resolve}(\text{cube}_4, \text{cube}_5) \\ 4. \text{cube}_{11} = \text{resolve}(\text{cube}_{10}, \text{cube}_6) \\ 5. \text{cube}_{\square} = \text{resolve}(\text{cube}_{11}, \text{cube}_9) \end{array} \right\}$$

The DAG of Π' is shown in Figure 4.6, where the arrow direction signifies the implication direction. Note that Ψ above is not expressed in PCNF for ease of illustration. Realistically Ψ should be in PCNF, and in this case cubes of Ψ_{mtx} above can be considered as cubes learnt from solving Ψ .

The corresponding RFAO node-array content after each \exists -reduction step in the DAG of Figure 4.6 is the same as that of Figure 4.3. Consequently the obtained Skolem functions $F_S[x]$ and $F_S[y]$ for Ψ equal the previous Herbrand functions $F_H[x]$

4.2. Extraction of Skolem and Herbrand functions from Q-resolution proofs

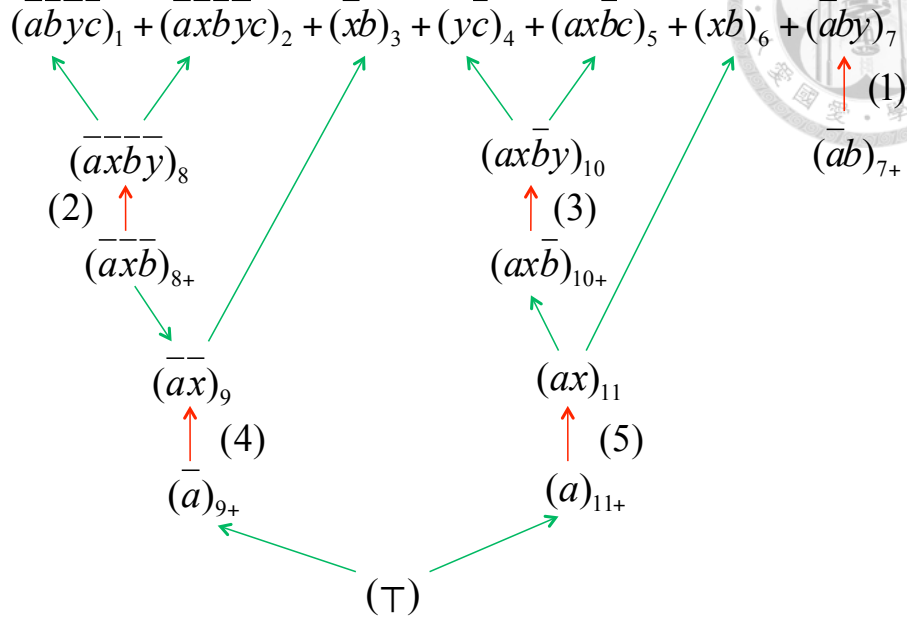


Figure 4.6: DAG of resolution proof Π' .

and $F_H[y]$ for Φ , respectively. In fact, equivalences are not a mere coincidence because the Skolem-function model for a true QBF Φ is also the Herbrand-function countermodel for its negated false QBF $\neg\Phi$, and vice versa.

Note that algorithm *Countermodel-Construct* (*Model-Construct*) can be easily modified to compute the Herbrand (Skolem) functions for a subset of the universal (existential) variables of interest for a given QBF. Let k be the maximal quantification level among the universal (existential) variables whose Herbrand (Skolem) functions are of interest. Algorithm *Countermodel-Construct* (*Model-Construct*) only needs to maintain RFAO node arrays for universal (existential) variables with quantification level no greater than k . For Skolemization-based solvers, this partial derivation is not possible because Skolem functions are constructed on-the-fly during QBF solving, whereas our construction is performed after the entire proof is done.

4.2.3 Experimental evaluation

Earlier proposed method was implemented into a tool called RESQU using C++ language. The experiments were conducted on a Linux machine with a Xeon 2.53 GHz CPU and 48 GB RAM for two sets of test cases: the QBF evaluation benchmarks

4.2. Extraction of Skolem and Herbrand functions from Q-resolution proofs

Table 4.1: Summary for QBFEVAL Benchmarks.

		all	sKizzo		SQUOLEM+RESQU			QUBE-CERT+RESQU		
		#sv	#sv	time (sv)	#sv	time (sv)	time (md)	#sv/#pg	time (sv)	time (md)
true	'05	84	69	1707	50	1491	—	19/19	415	55
	'06	48	29	295	25	200	—	44/44	860	152
	Σ	132	98	2002	75	1691	—	63/63	1274	207
false	'05	77	0	0	42	1467	13	46/25	2370	13
	'06	29	0	0	9	86	1	28/22	917	2
	Σ	106	0	0	51	1553	13	74/47	3286	15

#sv: number of instances solved; #pg: number of proofs involving no long-distance resolution; time (sv/md): CPU time in seconds for QBF evaluation/(counter)model generation; —: data not available due to inapplicability of ResQu

downloaded from [37] and relation determinization ones modified from [14].

We compared different Skolem-function derivation scenarios using QBF solvers sKizzo [10], SQUOLEM [28], and QUBE-CERT, which are equipped with certification capabilities.¹ For true QBF instances, sKizzo and SQUOLEM were applied to obtain Skolem-function models whereas the cube-resolution proofs produced by QUBE-CERT were converted to Skolem-function models by RESQU. For false QBF instances, sKizzo was applied on the negated formulae to obtain Skolem-function countermodels whereas the clause-resolution proofs produced by SQUOLEM and QUBE-CERT were converted to Skolem-function countermodels by RESQU.

Table 4.1 summarizes the results of our first experiment on the QBFEVAL'05 and QBFEVAL'06 test sets, which contain 211 and 216 instances, respectively. In the experiment, all the QBF solvers, including sKizzo, SQUOLEM, and QUBE-CERT, are given a 600-second time limit and a 1-GB memory limit for solving each instance. Under the given resource limits, all solvers, together, solved 132 true and 106 false instances. All the (counter)models produced by RESQU were verified using MINISAT [20] while the models produced by sKizzo and SQUOLEM were assumed correct without verification.

¹We did not experiment with EBDDRES [28] and YQUAFFLE [41] as the former tends to generate larger certificates for false QBF compared to SQUOLEM, and the latter has characteristics similar to QUBE-CERT.

4.2. Extraction of Skolem and Herbrand functions from Q-resolution proofs

It should be mentioned that the resolution proofs produced by QUBE-CERT were not simplified, that is, including resolution steps unrelated to producing the final empty clause (or empty cube). The unrelated resolution steps were first removed (with runtime omitted) before the (counter)model construction of RESQU. On the other hand, the clause-resolution proofs produced by SQUOLEM were simplified already and involved no long-distance resolution. Hence RESQU had no problems constructing their countermodels.

We compared the numbers of instances whose (counter)models generated by RESQU and by other tools. When models are concerned, RESQU (via the proofs from QUBE-CERT) covered 63 (19 in QBFEVAL'05 and 44 in QBFEVAL'06), whereas SKIZZO and SQUOLEM in combination covered 105 (75 in QBFEVAL'05 and 30 in QBFEVAL'06). When countermodels are concerned, RESQU (via the proofs from SQUOLEM and QUBE-CERT) covered 83 (60 in QBFEVAL'05 and 23 in QBFEVAL'06), whereas SKIZZO covered 0.² Notably, RESQU circumvents the DNF-to-CNF conversion problem and is unique in generating countermodels.

While all the (counter)models can be constructed efficiently (for proofs without long-distance resolution), some of them can be hard to verify. In fact, about 84% of the 161 (counter)models constructed by RESQU were verified within 1 second using MINISAT; there are 5 models of the true instances in QBFEVAL'06 that remained unverifiable within 1000 seconds.

Table 4.1 also reveals that for the QBFEVAL'05 true instances SKIZZO and SQUOLEM outperformed QUBE-CERT in solving more instances, while the situation reversed for the QBFEVAL'06 true cases. Investigating the reasons, we noted that, for the solved QBFEVAL'05 true instances, the average numbers of quantification levels, existential variables, universal variables, and clauses are 22, 1393, 80, and 8028, respectively; for QBFEVAL'06, they are 3, 672, 78, and 1690, respectively. The statistics might suggest that SKIZZO and SQUOLEM could work better for true QBF with fewer existential variables (since there are fewer Skolem functions to derive), whereas search-based solvers could work better for cases with fewer quantification

²In addition to SKIZZO, in theory SQUOLEM can also compute Skolem-function countermodels of false QBF instances by formula negation. We only experimented with SKIZZO, which can read in DNF formulae and thus requires no external DNF-to-CNF conversion, arising due to formula negation. Although SQUOLEM is not experimented in direct countermodel generation by formula negation, prior experience [28] suggested that it might be unlikely to cover much more cases than SKIZZO.

4.2. Extraction of Skolem and Herbrand functions from Q-resolution proofs

Table 4.2: Summary for Relation Determinization Benchmarks.

	all #sv	sKIZZO		SQUOLEM+RESQU			QUBE-CERT+RESQU		
		#sv	time (sv)	#sv	time (sv)	time (md)	#sv/#pg	time (sv)	time (md)
true	17	2	1.0	3	2.1	—	17/17	51.3	8.8
false	22	0	0	21	1175.8	5.2	22/0	1254.3	242.9

(Legend same as in Table 4.1)

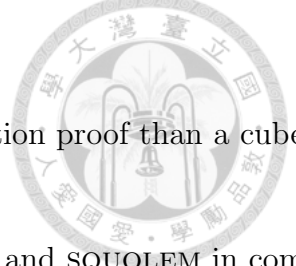
levels (since resolution depths are shallower and \exists -reduction can be more effective).

We also compared the sizes of (counter)models in terms of and-inverter graph (AIG) nodes. For models, we noticed that those constructed by RESQU from the proofs of QUBE-CERT can be up to a few orders of magnitude larger than those from sKIZZO and SQUOLEM. Similarly for countermodels, those constructed by RESQU from the proofs of QUBE-CERT tend to be much larger than those from the proofs of SQUOLEM. These AIGs can be very redundant and can be substantially simplified by ABC [11] synthesis commands, e.g., `balance`, `dc2`, `collapse`, etc. If heavy synthesis commands such as `collapse` succeeded, all the (counter)models derived in different ways for the same instance can be of comparable sizes.

Table 4.3 shows the results of our second experiment on 22 relation determinization benchmarks. All the original 22 instances are true (satisfiable). We compared their models obtained in two ways: by direct model construction from the satisfiability proofs of the original formulae and by indirect model construction from the unsatisfiability proofs of their negations. Unlike the QBFEVAL cases, negating these formulae by Tseitin’s transformation does not result in variable- and clause-increase, as was discussed in Section 2.3. The experiment was conducted under the resource limit same as before. For the original instances, RESQU could have generated Skolem functions only for the existential variables of interests (namely, the output variables of a Boolean relation rather than the intermediate variables), but it generated all for the verification purpose.

It was observed that SQUOLEM is easier to produce a clause-resolution proof than Skolem functions for a given relation determinization instance. This phenomenon might be explained by the common large number of inner-most existential variables, each of which requires building a Skolem function. It was similarly observed that

4.3. Flexibilities and Optimization of Skolem and Herbrand Functions



QUBE-CERT is relatively easier to produce a clause-resolution proof than a cube-resolution one.

As summarized in Table 4.2, for the true cases, SKIZZO and SQUOLEM in combination can construct models for only 3 instances, whereas from the 17 proofs of QUBE-CERT, RESQU can generate (and verify) all models. For the negated cases, all the proofs provided by QUBE-CERT involved long-distance resolution. RESQU internally applied the aforementioned rewriting rules to convert them into new proofs without long-distance resolution and constructed their countermodels. All the instances were successfully solved and verified. SQUOLEM solved 21 out of 22 instances, and RESQU can generate (and verify) all their countermodels (i.e., models for the original QBF). It is interesting to see that, in the relation determinization application, countermodel generation for the negated formulae can be much easier than model generation for the original formulae. It reveals the essential value of RESQU.

4.3 Flexibilities and Optimization of Skolem and Herbrand Functions

Main approach for Skolem/Herbrand functions derivation from Q-resolution proofs, proposed in Section 4.2, has linear time complexity with respect to the size of corresponding Q-refutation (Q-consensus). Derived functional certificates, however, might be too large for practical applications. In this section we characterize five types of flexibilities under the RESQU construction procedure and their use for Skolem/Herbrand function optimization. Although our discussion focuses on Herbrand functions, similar results can be obtained for Skolem functions. Experimental results show certificate simplification with up to orders of magnitude reduction in circuit size and depth.

4.3.1 Effective pure literals detection

Essentially *Countermodel-Construct* algorithm ensures that under every truth assignment to the set of existential variables the universal variables will receive logic values specified by their respective Herbrand functions such that some clause in the

4.3. Flexibilities and Optimization of Skolem and Herbrand Functions

Table 4.3: Results for Relation Determinization Benchmarks.

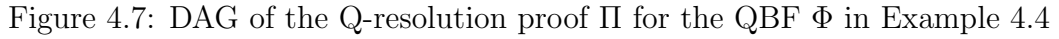
	(#i, #o, #e, #C)	sKIZZO		SQUOLEM+RESQU		QUBE+RESQU	
		time (sv)	size	time (sv/md/vf)	size	time (sv/md/vf)	size
22 original (true) instances	(7, 3, 55, 322)	0.1	377	(0.1, —, —)	134	(0.0, 0.0, 0.0)	28
	(20, 10, 1k, 6k)	0.9	1.3k	(0.8, —, —)	1.4k	(0.1, 0.0, 0.0)	118
	(21, 9, 1k, 8k)	NA		(NA, —, —)		(5.3, 1.7, 1.2)	149k
	(24, 12, 2k, 11k)	NA		(1.2, —, —)	179	(1.0, 0.1, 0.0)	1.9k
	(28, 14, 2k, 11k)	NA		(NA, —, —)		(0.4, 0.1, 0.0)	61
	(32, 16, 3k, 20k)	NA		(NA, —, —)		(1.2, 0.2, 0.0)	1.2k
	(36, 18, 6k, 35k)	NA		(NA, —, —)		(0.2, 0.2, 0.0)	91
	(42, 20, 7k, 42k)	NA		(NA, —, —)		(0.3, 0.1, 0.0)	3
	(39, 19, 10k, 59k)	NA		(NA, —, —)		(3.1, 0.5, 0.0)	307
	(46, 22, 11k, 63k)	NA		(NA, —, —)		(1.9, 0.3, 0.0)	58
	(49, 19, 11k, 68k)	NA		(NA, —, —)		(NA, NA, NA)	
	(32, 18, 13k, 81k)	NA		(NA, —, —)		(NA, NA, NA)	
	(50, 24, 15k, 89k)	NA		(NA, —, —)		(3.1, 0.8, 0.1)	3.5k
	(53, 25, 16k, 96k)	NA		(NA, —, —)		(3.4, 0.4, 0.0)	283
	(56, 26, 20k, 118k)	NA		(NA, —, —)		(8.1, 1.1, 0.1)	905
	(59, 27, 26k, 157k)	NA		(NA, —, —)		(3.9, 0.6, 0.0)	187
	(65, 29, 29k, 174k)	NA		(NA, —, —)		(7.2, 0.9, 0.1)	232
	(62, 28, 30k, 182k)	NA		(NA, —, —)		(9.3, 1.3, 0.1)	731
	(72, 32, 36k, 215k)	NA		(NA, —, —)		(NA, NA, NA)	
	(68, 30, 51k, 303k)	NA		(NA, —, —)		(3.0, 0.6, 0.1)	11
	(95, 35, 58k, 346k)	NA		(NA, —, —)		(NA, NA, NA)	
	(41, 23, 90k, 538k)	NA		(NA, —, —)		(NA, NA, NA)	
22 complemented (false) instances	(7, 3, 55, 322)	NA		(0.0, 0.0, 0.0)	6	(0.1, 0.0, 0.0)	17
	(20, 10, 1k, 6k)	NA		(1.1, 0.0, 0.0)	53	(0.1, 0.1, 0.0)	61
	(21, 9, 1k, 8k)	NA		(0.2, 0.0, 0.0)	4	(1.2, 0.1, 0.0)	5
	(24, 12, 2k, 11k)	NA		(0.3, 0.0, 0.0)	0	(0.3, 0.3, 0.0)	30
	(28, 14, 2k, 11k)	NA		(0.3, 0.0, 0.0)	3	(1.0, 0.5, 0.0)	150
	(32, 16, 3k, 20k)	NA		(2.0, 0.0, 0.0)	3	(1.0, 0.2, 0.0)	3
	(36, 18, 6k, 35k)	NA		(3.1, 0.1, 0.1)	3	(4.2, 1.3, 0.0)	11
	(42, 20, 7k, 42k)	NA		(3.2, 0.1, 0.1)	3	(9.2, 3.6, 0.0)	1.2k
	(39, 19, 10k, 59k)	NA		(9.4, 0.1, 0.1)	5	(10.0, 1.6, 0.0)	201
	(46, 22, 11k, 63k)	NA		(9.9, 0.2, 0.1)	3	(3.6, 0.6, 0.0)	31
	(49, 19, 11k, 68k)	NA		(8.3, 0.2, 0.1)	3	(14.1, 0.9, 0.0)	1
	(32, 18, 13k, 81k)	NA		(10.4, 0.2, 0.1)	3	(10.4, 1.3, 0.0)	0
	(50, 24, 15k, 89k)	NA		(15.8, 0.3, 0.1)	4	(510, 89.9, 0.0)	20k
	(53, 25, 16k, 96k)	NA		(23.7, 0.3, 0.1)	5	(7.2, 0.9, 0.0)	4
	(56, 26, 20k, 118k)	NA		(30.2, 0.4, 0.1)	3	(25.3, 2.3, 0.0)	93
	(59, 27, 26k, 157k)	NA		(74.2, 0.4, 0.1)	3	(203, 122, 0.0)	12k
	(65, 29, 29k, 174k)	NA		(46.9, 0.4, 0.2)	0	(24.5, 1.5, 0.0)	5
	(62, 28, 30k, 182k)	NA		(84.5, 0.5, 0.3)	4	(94.9, 3.3, 0.0)	570
	(72, 32, 36k, 215k)	NA		(130, 0.4, 0.2)	3	(80.1, 2.6, 0.0)	662
	(68, 30, 51k, 303k)	NA		(363, 0.7, 7.3)	3	(26.1, 2.3, 0.0)	0
	(95, 35, 58k, 346k)	NA		(359, 1.0, 8.2)	2	(86.1, 2.5, 1.2)	6
	(41, 23, 90k, 538k)	NA		(NA, NA, NA)		(142, 5.1, 0.0)	170

#i: number of input variables in a relation; #o: number of output variables in a relation; #e: number of innermost existential variables added due to circuit-to-CNF conversion; #C: number of clauses in final CNF; size: number of AIG nodes after performing ABC command `dc2` with negligible runtime not shown; time (sv/md/vf): CPU time in seconds for QBF evaluation/(counter)model generation/verification; NA: data not available due to computation out of resource limit; —: data not available due to inapplicability of ResQu

$$\overline{x+b+z}_6(a+b+y+z)$$

$$\downarrow \quad \quad \quad \downarrow$$

$$(\overline{x+b})_{6+} \cdot (a+b)_{7+}$$



We use the following example as a working example throughout our discussion.

$$\begin{aligned}\Phi_{\text{pfx}} &= \exists a \forall x \exists b \forall y \exists c \forall z \\ \Phi_{\text{mtx}} &= (a + b + y + c)(a + x + b + y + \bar{c})(x + \bar{b} + z) \\ &\quad (\bar{y} + c)(\bar{a} + \bar{x} + b + \bar{c})(\bar{x} + \bar{b} + z) \\ &\quad (a + \bar{b} + \bar{y} + \bar{z})\end{aligned}$$
$$\begin{aligned} F[x] &= (a) \wedge (a) = a, \\ F[y] &= (\neg ab) \vee ((a \vee x \vee b) \wedge (ax \neg b)), \text{ and} \\ F[z] &= (x \vee \neg b) \wedge ((\neg x \vee \neg b) \wedge (\neg ab)), \end{aligned}$$

79

4.3. Flexibilities and Optimization of Skolem and Herbrand Functions



The property below is related to QBF logical implication.

Proposition 4.4 Given two false QBFs Φ_1 and Φ_2 with the same prefix, let $\Phi_{1_{\text{mtx}}} \subseteq \Phi_{2_{\text{mtx}}}$. Then every Herbrand countermodel of Φ_1 is also a countermodel of Φ_2 .

By above proposition, for a false QBF Φ its unsatisfiable core of clauses in Φ_{mtx} can be useful in guiding the search for more desirable certificates. For the portion of a proof irrelevant to the deduction of the empty clause, it can be optionally taken or not taken into account for the Herbrand function construction without affecting the validity of the constructed Herbrand countermodel.

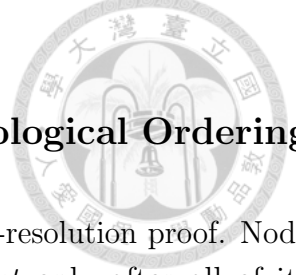
Example 4.5 Consider the Q-resolution proof of Figure 4.7. Clause 7 does not belong to the unsatisfiable core of the proof, and thus it can be optionally added in or removed from RFAO arrays. In Example 4.4, Clause 7 was taken into account. If we omit it, the Herbrand functions of y and z become $F[y] = (a \vee x \vee b) \wedge (ax \neg b)$ and $F[z] = (x \vee \neg b) \wedge (\neg x \vee \neg b)$, respectively.

In practice it is not uncommon that unsatisfiable core may contain *effectively pure literals*, which are pure (i.e., appearing only in either positive or negative phase) only in the unsatisfiable core but not in the original matrix. They can be exploited for the following constant simplification.

Proposition 4.5 Given a Q-resolution proof Π of a false QBF Φ , if the \forall -reduced literals of a universal variable x are of positive (negative) phase everywhere in the proof relevant to empty clause derivation, then the Herbrand function of x can be directly simplified to a constant 0 (constant 1) function.

Our implementation detects effectively pure literals in an unsatisfiable core for constant propagation.

Example 4.6 Consider the Q-resolution proof of Figure 4.7. Since z is an effectively pure literal in the unsatisfiable core (Clauses 1 \sim 6), its Herbrand function can be constructed as $F[z] = 0$.



4.3.2 Optimization with Flexibility of Topological Ordering

Some pre-specified topological order is imposed on a given Q-resolution proof. Node of a graph is visited by algorithm *Countermodel-Construct* only after all of its predecessor nodes with incidence edges to it have been visited. Given a proof DAG, many topological orders can be defined, and any topological order uniquely determines RFAO formula construction of Herbrand functions. The following proposition is immediate from the correctness of algorithm *Countermodel-Construct*.

Proposition 4.6 Given a proof DAG $G_{\Pi}(V, E)$ of a QBF Φ , let $O : V \rightarrow \mathbb{N}$ be an ordering on the vertices V satisfying the topological property $O(u) < O(v)$ for any edge $(u, v) \in E$. Then the RFAO formulas constructed by the *ResQu* procedure correspond to a correct Skolem and Herbrand certificate of Φ .

Proof Statement is immediate from the correctness of the *ResQu* procedure. ■

Clearly the set of topological orderings of a proof DAG can be exploited as flexibility to search for good circuit realization of Herbrand functions.

The flexibility of topological ordering has an intriguing connection to the *circuit depth complexity* [1], where the number of alternations between AND and OR gates is of main concern. Note that using the number of AND-OR alternations as a complexity measure is particularly effective for RFAO formulas. The flexibility can be naturally exploited to minimize the number of AND-OR alternations and thus circuit depths.

Example 4.7 Continue Example 4.4. Under the original clause reduction ordering 3, 6, 7, 8, 9, 10, 11, we have

$$F[y] = (\neg ab) \vee ((a \vee x \vee b) \wedge (ax \neg b)).$$

Under another ordering 3, 6, 10, 11, 8, 9, 7, we get

$$F[y] = (ax \neg b) \vee ((a \vee x \vee b) \wedge (\neg ab)).$$

Under the ordering 3, 6, 7, 10, 11, 8, 9, we obtain

$$F[y] = (\neg ab) \vee ((ax \neg b) \vee (a \vee x \vee b)).$$

4.3. Flexibilities and Optimization of Skolem and Herbrand Functions

Note that the above three orderings satisfy the topological precedence constraint and are legitimate orderings. Whereas the first two orderings contain one alternation in the RFAO formula construction, the third ordering contains no alternation. To realize these RFAO formulas using AND and OR gates with unbounded fanins, the first two formulas requires OR-AND-OR three logic layers, and the third formula requires AND-OR two logic layers (from inputs to the output).

Algorithm *Alternation-Minimization* in Figure 4.8 shows our implementation of the alternation minimization procedure. It is based on a greedy strategy to minimize the number of alternations in RFAO formula construction. Given a Q-resolution proof DAG G_Π as input, it returns RFAO arrays for Herbrand functions. In lines 3-8 of the algorithm we traverse the proof G_Π and mark as *red* the nodes where \forall -reduction occurs (corresponding set is denoted as a **Red_Set**). Further, at step 8 the list of all topologically unconstrained red nodes is built (denoted as a **Free_Queue**). Next, by one-step look-ahead, we iteratively select the best (i.e., incurring the least increase of AND-OR alternations) \forall -reduction step from the **Free_Queue**. The corresponding scores are computed in lines 10-21. Necessary RFAO nodes are added according to the chosen greedy \forall -reduction step in lines 22-28 (equivalent to steps 5-10 of algorithm *Countermodel-Construct* on Figure 4.1). Finally, the visited reduction step is unmarked, and both **Red_Set** and **Free_Queue** are updated accordingly in lines 29-33. The overall time complexity of the process is quadratic in the number of \forall -reduction steps in G_Π . Theorem 4.2 further states the correctness of the algorithm *Alternation-Minimization*.

Theorem 4.2 Algorithm *Alternation-Minimization* returns correct Herbrand functions.

Proof Note that lines 9-22 in Figure 4.8 do not change the input Q-refutation, but rather impose a different order in which we process the proof. It means that RFAO construction procedure in lines 24-28 (that corresponds to the original *Countermodel-Construct* construction algorithm) returns the correct countermodel. ■

4.3.3 Postponing forall-reduction

On contrary to the previous ordering flexibility, which confines to a fixed proof, one may structurally rewrite the given resolution proof for Skolem and Herbrand function

4.3. Flexibilities and Optimization of Skolem and Herbrand Functions



Alternation-Minimization($\Phi, G_{\Pi}(V, E)$)

input: a false QBF Φ and its Q-resolution DAG $G_{\Pi}(V, E)$;
output: Herbrand functions in RFAO formulas;
begin
01 **foreach** $x \in X_{\forall}$ of Φ
02 RFAO_array[x] := \emptyset ;
03 **foreach** vertex $v \in V$ in a topological order
04 **if** $v.clause$ is \forall -reduced from $u.clause$
05 **mark** u as *red* vertex;
06 **push** u into Red_Set;
07 **if** u has no *red* ancestors in $G_{\Pi}(V, E)$
08 **push** u into Free_queue;
09 **while** Free_queue is not empty
10 $minScore = inf$;
11 **foreach** vertex u in the Free_queue
12 **let** v be the child of u in $G_{\Pi}(V, E)$;
13 $score = 0$;
14 **foreach** literal $\{l | l \in u.clause \wedge l \notin v.clause\}$
15 $C :=$ the last node of RFAO_array[$var(l)$];
16 **if** C is a clause and l is negative
17 $score = score + 1$;
18 **else if** C is a cube and l is positive
19 $score = score + 1$;
20 **if** ($score < minScore$)
21 $minScore = score$; $z := u$;
22 **let** v be the child of z in $G_{\Pi}(V, E)$;
23 $v.cube := \neg(v.clause)$;
24 **foreach** literal $\{l | l \in z.clause \wedge l \notin v.clause\}$
25 **if** l is a positive literal
26 **push** $v.clause$ to RFAO_array[$var(l)$];
27 **else if** l is a negative literal
28 **push** $v.cube$ to RFAO_array[$var(l)$];
29 **delete** u from the Free_queue;
30 **unmark** u and **delete** u from the Red_set;
31 **foreach** u in Red_set
32 **if** u has no *red* ancestors in $G_{\Pi}(V, E)$
33 **push** u into Free_queue;
34 **return** RFAO_array's;
end

Figure 4.8: Algorithm: Greedy Alternation Minimization.



minimization.

Proposition 4.7 In a Q-resolution proof Π of a QBF Φ over variables X , let C'_1 be a clause \forall -reduced from C_1 such that literal $l \in C_1$ and $l \notin C'_1$ (inferring $\text{var}(l) \in X_\forall$). Suppose C_1 and C_2 are resolved in Π yielding a resolvent clause C_3 . If 1) literal $\neg l \notin C_2$ and 2) there exists no literal $l' \in C_2$ with $\text{var}(l') \in X_\exists$ and $\ell(l') > \ell(l)$, then the \forall -reduction of l in C_1 can be postponed and performed in C_3 .

Proof Observe that postponing the \forall -reduction operation of C_1 to C_3 yields clause C'_3 subsuming C_3 , that is, every literal of C'_3 is also a literal of C_3 . Since the derivation of C'_3 is sound and the implication $C'_3 \rightarrow C_3$ holds, the correctness of the rewriting is established. ■

Thereby the \forall -reduction steps of a proof can be postponed and exploited as flexibility in realizing Herbrand functions. Postponing \forall -reductions may potentially reduce the number of \forall -reductions in a Q-resolution proof. This situation may happen if reduction postponing is simultaneously applied on both clauses C_1 and C_2 for the same universal variable in the same phase. Also if C_3 is simpler than C_1 , the RFAO formula of variable $\text{var}(l)$ can be simpler.

Example 4.8 Consider the Q-resolution proof of Figure 4.7. The \forall -reduction of literal y in Clause 8 can be postponed and only performed in Clause 9. Also the \forall -reduction of literal $\neg y$ in Clause 10 can be postponed and performed in Clause 11. By the *Countermodel-Construct* procedure, the Herbrand function of variable y is derived as $F[y] = (\neg ab) \vee ((a) \wedge (a)) = a \vee b$, which is simpler than before.

It is important to notice that application of algorithm *Countermodel-Construct* depicted on Figure 4.1 for countermodel extraction from the postponed Q-resolution proofs is only sound under certain restrictions. Example 4.9 illustrates the issue of circular dependencies that might arise if algorithm *Countermodel-Construct* is applied to an arbitrary postponed Q-refutation.

Example 4.9 Consider the following false QBF Φ :

$$\Phi = \exists a \forall x \exists b \forall y \exists c (a \vee x \vee y)_1 (\neg a \vee \neg x \vee b)_2 (\neg b \vee \neg y)_3,$$

and its postponed Q-refutation Π :



Postponing-Reduction($\Phi, G_{\Pi}(V, E)$)

input: a false QBF Φ and its Q-resolution DAG $G_{\Pi}(V, E)$
output: postponed Q-resolution DAG $G_{\Pi}(V, E)$;
begin
 01 **foreach** vertex $v \in V$ in a topological order
 02 **if** $v.clause$ is \forall -reduced from $u.clause$
 03 **foreach** literal $\{l | l \in u.clause \wedge l \notin v.clause\}$
 04 **let** z be the closest descendant of v such that:
 05 $\{\exists l' | l' \in z.clause \wedge$
 $\wedge var(l') \in X_{\exists} \wedge lvl(l') > lvl(l)\}$;
 06 **let** $l.bound :=$ distance from z to v in G_{Π} ;
 07 **set** the *score* for each universal variable x such that:
 08 **if** $lvl(x) > lvl(y)$
 09 $score(x) < score(y)$;
 10 **else if** $\sum_{var(l)=x} (l.bound) > \sum_{var(l)=y} (l.bound)$
 11 $score(x) > score(y)$;
 12 **foreach** vertex u in topological order $G_{\Pi}(V, E)$
 13 **foreach** universal literal l that can be reduced from
 $u.clause$ in the *score*-descending order
 14 **if** no any child of u has literal \bar{l} and $l.bound > 1$
 15 **postpone** reduction of l ;
 16 **decrease** $l.bound$ by 1;
 17 **else**
 18 **continue** to the next vertex;
 19 **return** $G_{\Pi}(V, E)$;
end

Figure 4.9: Algorithm: Postponing \forall -reduction.

$$\Pi = \left\{ \begin{array}{l} 1. C_4 = reduce(C_1, x); \\ 2. C_5 = resolve(C_2, a, C_4); \\ 3. C_6 = reduce(C_5, y); \\ 4. C_7 = resolve(C_6, b, C_3); \\ 5. C_8 = reduce(C_7, \neg x, \neg y). \end{array} \right\}$$

According to algorithm *Countermodel-Construct* extracted Herbrand functions are $F[x] = (a \vee y)$ and $F[y] = (\neg x \vee b)$. This set of functions, however, does not form a proper countermodel.

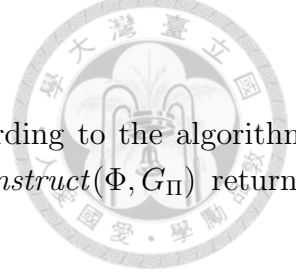
4.3. Flexibilities and Optimization of Skolem and Herbrand Functions

As Example 4.9 shows, universal reduction has to be done strictly according to a specific order. If one universal variable can not be postponed further, all other variables with higher quantification order have to be reduced simultaneously with it. Proposition 4.8 below states that the strict order condition is also a sufficient condition for sound application of algorithm *Countermodel-Construct* for postponed proofs.

Proposition 4.8 Consider a false QBF Φ , and its postponed Q-resolution proof Π , such that for any two universal reduction steps $C_1 = \text{reduce}(D_1)$ and $C_2 = \text{reduce}(D_2)$ in Π the following holds: if $x \in D_1/C_1$ (i.e. x is universally reduced from D_1) and $y \in C_1$, then either $x \notin C_2$ or $y \notin D_2/C_2$.

Proof First observe, that constructed by *Countermodel-Construct* algorithm from Π Herbrand functions respect the quantification order of Φ . They also falsify the matrix, which can be proven identically to the proof of correctness of the *Countermodel-Construct* on unpostponed proofs. ■

Implemented algorithm *Postponing-Reduction* is depicted in Figure 4.9. Our implementation postpones a \forall -reduction step as late (i.e. as close to the empty clause in the proof) as possible. It is achieved by a two-pass proof traversal: In the first pass, the proof is traversed from root clauses to the empty clause and determines the maximum postponing *depth* of a given clause (i.e. the shortest postponing distance among all the longest legitimate paths starting at the given clause). Application of rules 1 and 2 of Proposition 4.7 corresponds to the lines 3-5 of the *Postponing-Reduction* procedure. Second pass is necessary to avoid cyclic variable dependencies arising in RFAO formula construction. Cyclic dependency issue requires the reduction of all universal variables to be done strictly according to some specific order through the whole resolution proof. In lines 7-11 we heuristically impose a strict order on all the universal variables according to their quantification level and the cumulative postponing ability. Intuitively, variables with lower postponing score might enforce reduction of variables with higher postponing score earlier than expected. Therefore in our imposed strict order variables with lower scores are considered last. Lines 14-18 make sure that no violation of rules 1 and 2 of Proposition 4.7 occurs, and if it does, current universal literal (as well as all other universal literals that are quantified inner-more according to our strict order) is not postponed anymore (i.e. is reduced). Correctness of the presented postponing algorithm is formally stated by the Corollary 4.1, which directly follows from Proposition 4.8.



Corollary 4.1 Given a false QBF Φ and postponed according to the algorithm *Postponing-Reduction* Q-refutation G_Π , *Countermodel-Construct*(Φ, G_Π) returns correct countermodel.

Note that algorithm *Alternation-Minimization* can be also soundly applied to the postponed by procedure *Postponing-Reduction* Q-refutation.

4.3.4 Don't care minimization

Here we propose a heuristic syntactic characteristics of don't care conditions of some (universal) variable. Recall that $\phi|_\alpha$ denotes the induced formula of ϕ under (partial) assignment α and let $\Phi|_\alpha$ to denote QBF with prefix Φ_{pfx} and matrix $\Phi_{\text{mtx}}|_\alpha$. (Assume the variables with values being assigned by α vanish in Φ_{pfx} .) Definition 4.3 extends the notion of the QBF under a given assignment to Q-resolution proofs.

Definition 4.3 Let Π be a Q-refutation for false QBF Φ over existential and universal variables X and Y respectively. Further, let assignment $\alpha : X' \rightarrow \{0, 1\}$ be an arbitrary assignment on variables $X' \subseteq X$. $\Pi|_\alpha$ denotes the induced under α Q-refutation, and is computed from Π using (whenever possible, until convergence, priority is given with respect to the specified order) the following processing rules: Consider a resolution step $C = \text{resolve}(D_1, p, D_2)$, where $C, D_1, D_2 \in \Pi$ are clauses, p is a pivot literal such that $p \in D_1$ and $\bar{p} \in D_2$.

1. If D_1 (resp. D_2) is missing p (resp. \bar{p}) after rewriting, then $C \rightarrow D_2$ (resp. $C \rightarrow D_1$).
2. For any $C \in \Pi$, if $C \cap \alpha \neq \emptyset$ then $C \rightarrow \text{true}$;
3. If $D_1 = \text{true}$ (resp. $D_2 = \text{true}$) then $C \rightarrow D_2$ (resp. $C \rightarrow D_1$). Similarly for the reduction step $C = \text{reduce}(D)$, if $D = \text{true}$ then $C \rightarrow \text{true}$.

Proposition 4.9 For a false QBF Φ over variables X , let Π be a Q-resolution proof of its falsity and $\alpha : X' \rightarrow \{0, 1\}$ be an assignment on variables $X' \subseteq X_\exists$. Then $\Pi|_\alpha$ is a Q-resolution proof of $\Phi|_\alpha$.

Proof First, consider an arbitrary resolution of clauses $C_1 \vee p$ and $C_2 \vee \bar{p}$ under the pivot p producing resolvent $C = C_1 \vee C_2$. Condition $p \in \alpha$ (resp. $\bar{p} \in \alpha$) results

4.3. Flexibilities and Optimization of Skolem and Herbrand Functions

into simplification $C = C_2$ (resp. $C = C_1$) which is a subclause of $C_1 \vee C_2$. Second, observe that removing some existential literals from some clauses does not prevent any \forall -reductions. Therefore every clause in $\Pi|_\alpha$ corresponds to a stronger subclause (or at least to the same clause) in Π . Consequently $\Pi|_\alpha$ deduces the empty clause (proving also that $\Phi|_\alpha$ is false). ■

It is important to notice that condition $X' \subseteq X_\exists$ is crucial for the soundness of Proposition 4.9. Intuitively, proposition states that if the existential player originally did not have a winning strategy, then the restriction of his solution space by an assignment α (i.e., some of the existential variables effectively receive constant Skolem-functions) could not possibly allow him to win the game. Furthermore, the induced proof of unsatisfiability remains valid. Assignment of universal variables, on the other hand, might lead to the unsound induced Q-resolution proof.

By the above property the following don't care conditions for Herbrand functions of universal variables can be characterized.

Proposition 4.10 Given a QBF Φ over variables X and a Q-resolution proof Π of its falsity, if an assignment α to the existential variables X_\exists is such that the proof $\Pi|_\alpha$ for the falsity of $\Phi|_\alpha$ does not include any steps involving \forall -reduction on a universal variable $x \in X_\forall$, then α is a don't care condition of the Herbrand function $F[x]$ of x , that is, the function value of $F[x]$ under assignment α can be either 0 or 1.

Proof The fact that $\Pi|_\alpha$ does not involve any \forall -reduction on variable x implies x does not belong to $\Pi|_\alpha$ at all, and the value of x under α is irrelevant to the proof of falsity. Thus any function value of $F[x]$ under α is a correct Herbrand function value. $\Pi|_\alpha$, proving that under assignment α $F[x]$ can be either 0 or 1. ■

Characterizing all don't care conditions specified by above proposition is hard however. Nevertheless, for false QBFs, it is easy to specify a portion of these conditions.

Proposition 4.11 Given a false QBF Φ , let ψ be a conjunction of all clauses in Φ_{mtx} that only consist of existential variables. Then $\neg\psi$ characterizes a don't care set for all the Herbrand functions of Φ .

Above proposition holds due to the fact that under every assignment α to the existential variables that makes ψ to be false, proof $\Pi|_\alpha$ is directly reduced to just a

4.3. Flexibilities and Optimization of Skolem and Herbrand Functions

single empty clause. Unfortunately this technique cannot be applied to true QBFs in PCNF because for a true QBF we have to collect cubes, instead of clauses, which are not present in the original formula.

The following special set of don't cares can be easily computed, too.

Proposition 4.12 For a false QBF Φ over variables X and its Q-resolution proof Π , let $D[x]$ be the conjunction of the clauses in Π that result from \forall -reduction on literals of universal variable $x \in X_\forall$. Then $D[x]$ characterizes the don't care conditions in realizing the Herbrand function $F[x]$ of x .

Proof Observe that for every assignment α to variables in X_\exists with $D[x]|_\alpha = 1$, the induced proof $\Pi|_\alpha$ becomes free of \forall -reduction on variable x . By Proposition 4.10 we conclude that $D[x]$ characterizes a don't care set for the Herbrand function of x . ■

In our implementation, the above don't cares are supplied to the Berkeley logic synthesis tool ABC [11] for logic resubstitution and minimization.

Example 4.10 Consider the Q-resolution proof of Figure 4.7. For variable x , we have $D[x] = (a) \wedge (\neg a)$, which corresponds to an empty don't care set. For variable y , we get $D[y] = (a \vee x \vee b) \wedge (\neg a \vee \neg x \vee b)$ (here we consider only the clauses in the unsatisfiable core of the proof), which can be used to simplify $F[y]$ from the complex formula $(a \vee x \vee b) \wedge (ax\neg b)$ to the simple formula a .

4.3.5 Effects of variable sorting on certificate size

As was stated in Section 2.1 and-inverter graphs (AIGs) form an efficient data structure for Boolean manipulations. Structural hashing, balancing, rewriting and refactoring are common optimization techniques in AIG synthesis [32]. Empirical experience suggests that synthesis tools are sensitive to the variable naming, and even to the order of fanin variables of a multi-input gate.

Certificates produced by RESQU approach have very specific structure. They consist primarily of large multi-input OR and AND gates, which originate from nodes in corresponding RFAO arrays. When transforming a general Boolean network to an AIG, multi-input gates shall be split into 2-input AND gates. Different orderings of fanin variables may lead to different AIG structures.

4.3. Flexibilities and Optimization of Skolem and Herbrand Functions

Example 4.11 Consider 3-input AND gate $x = (a \wedge b \wedge c)$. It can be split into $y = a \wedge b$ and $x = y \wedge c$, as well as into $y = a \wedge c$ and $x = y \wedge b$. Note that the maximum structural sharing for gates $x_1 = (a \wedge b \wedge c)$ and $x_2 = (a \wedge b \wedge d)$ occurs when a and b are grouped together, yielding a common intermediate gate $y = a \wedge b$.

Now consider Q-resolution proof Π for a false QBF. Suppose two clauses C_1 and C_2 are obtained by universal reduction such that C_1 topologically precedes C_2 in Π (i.e., C_2 is a not immediate successor of C_1). It is empirically observed that for most of the literals $l \in C_2$ we also have $l \in C_1$. Furthermore, occasionally $C_1 \in C_2$. This observation is supported by Q-resolution proofs generated by search-based solvers. The resolution subproof on the path from C_1 to C_2 usually looks like a chain of eliminations of existential variables from C_1 , caused by unit propagations, in order to allow universal reduction that occurs at C_2 .

Example 4.12 Consider the Q-resolution proof of Figure 4.7. Observe that clause $C_1 = (a)_{9+}$ is a topological successor of $C_2 = (a \vee x \vee b)_{8+}$, and $C_2 \in C_1$. Furthermore clause $C_1 = (\neg a)_{11+}$ is a topological successor of $C_2 = (\neg a \vee \neg x \vee b)_{10+}$, and again $C_2 \in C_1$.

Compared to inner quantified variables, outer quantified ones occur more often in clauses that are topologically closer to the empty clause. This phenomenon suggests the following heuristics for variable ordering to increase structural sharing, while converting RFAO circuit into AIG.

Observation 4.1 When splitting multi-input gates, following the order of variables imposed by the quantification prefix can increase overall sharing in AIGs constructed from RFAO arrays.

This observation is confirmed by empirical statistics.

4.3.6 Experimental evaluation

Presented minimization methods were integrated into tool RESQU, for Skolem and Herbrand function derivation. All experiments were conducted on a Linux machine with a Xeon 2.53GHz CPU and 48GB RAM. Two benchmark test sets were

4.3. Flexibilities and Optimization of Skolem and Herbrand Functions

selected, namely the QBFEVAL'10 test set and a set of selected QBF families from QBFLIB [37].³

To generate Q-consensus/resolution proofs, we used the leading QBF solver DEPQBF [31, 34] for QBF evaluation. DEPQBF comes with an internal tool, QRPCERT [34], to generate Skolem and Herbrand certificates from Q-consensus and Q-resolution proofs. We intend to evaluate the certificate quality of QRPCERT and RESQU. Notice that, the Skolem and Herbrand certificates produced by QRPCERT were validated with its own internal checker CERTCHECK running PICOSAT [34], whereas those produced by RESQU were validated with ABC [11]. Although the validation time is not directly comparable, the SAT solver in ABC is not as advanced as PICOSAT. Thus the differences in validation setup are not favorable to RESQU.

In the experiments, a time limit of 1000 seconds was set for QBF solving. For the QBFEVAL'10 benchmark suite (including 568 formulas in total, 130 true have been solved by DEPQBF), a memory limit of 2GB was imposed on DEPQBF for proof logging. For the selected QBFLIB families (including 90 formulas in total, 66 have been solved by DEPQBF), a memory limit of 1GB was imposed for proof logging. Moreover, a time limit of 2000 (respectively 1000) seconds is set for Skolem (respectively Herbrand) certificate generation and validation of true (respectively false) QBFs while no memory limit is imposed.⁴

The plots of Figures 4.10 and 4.11 compare, in terms of circuit size and depth, Skolem and Herbrand certificates generated under different scenarios. Figures 4.10 and 4.11 show the results of true and false instances, respectively. All the certificates were transformed to and-inverter graph (AIG) circuits in ABC and further optimized under command `balance`. QRPCERT tool implements the original certificate extraction algorithm similar to RESQU, and is not equipped with any optimization. As the variable sorting heuristics gives a large improvement in certificate size, and does not require much additional computation resources, we have it in RESQU and by default use it together with other optimization techniques. The impact of the variable sorting heuristics can be seen by comparing results between QRPCERT and

³Since many false instances from the QBFEVAL'10 test set have relatively small certificate sizes, the full optimization power of presented techniques cannot be seen easily. Hence some instance families in QBFLIB that tend to have more complicated Herbrand-certificates were selected. They include the `ToiletC`, `k-ph-p`, and `Blocks` families.

⁴The time limit difference is due to the fact that in general true instances tend to be harder to generate and validate their certificates than false instances.

4.3. Flexibilities and Optimization of Skolem and Herbrand Functions

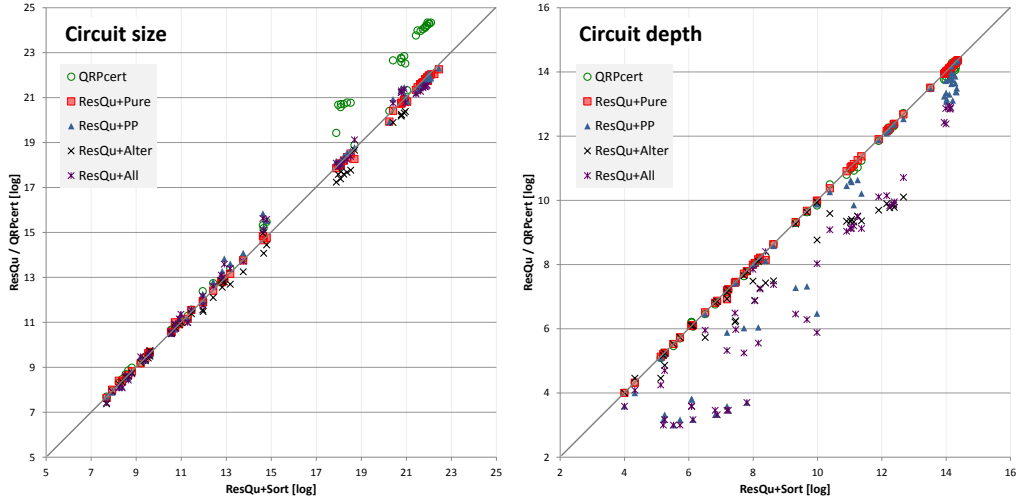


Figure 4.10: Certificate size and depth statistics for true instances.

RESQU+SORT. Furthermore, all the data has been normalized to RESQU+SORT with only sorting heuristics on, in order to clearly see the improvements yielded by other heuristics. RESQU+PURE denotes results by applying the effectively pure literal simplification; RESQU+PP denotes results by applying reduction postponing; RESQU+ALTER denotes results by applying alternation minimization; RESQU+ALL denotes results yielded by all heuristics combined. In these figures, the x -axis gives the data produced by RESQU+SORT and the y -axis gives the data yielded from QRPCERT and RESQU enhanced further by different optimization techniques. Both axes are in the \log_2 scale.

Table 4.4 summarizes the impacts of the proposed enhancements on the generated certificates. Comparison is made in terms of the numbers of generated (denoted $\#e$) and verified (denoted $\#v$) certificates, as well as of geometric ratios for the certificate generation time (denoted T_e), certificate validation time (denoted T_v), circuit size in AIG nodes (denoted $\#aig$), and circuit depth (denoted $\#lvl$). Geometric mean is more favorable in our case compared to the arithmetic mean because it treats improvement and degradation unbiasedly.

When solving and verification times are of concern, despite computational overhead, alternation minimization and postponing reduction heuristics substantially speed up verification process. In fact, more instances can be successfully verified in a shorter time when using all of the heuristics together, compare to validation of the certificates produced by RESQU+SORT and QRPCERT. When certificate quality

4.3. Flexibilities and Optimization of Skolem and Herbrand Functions

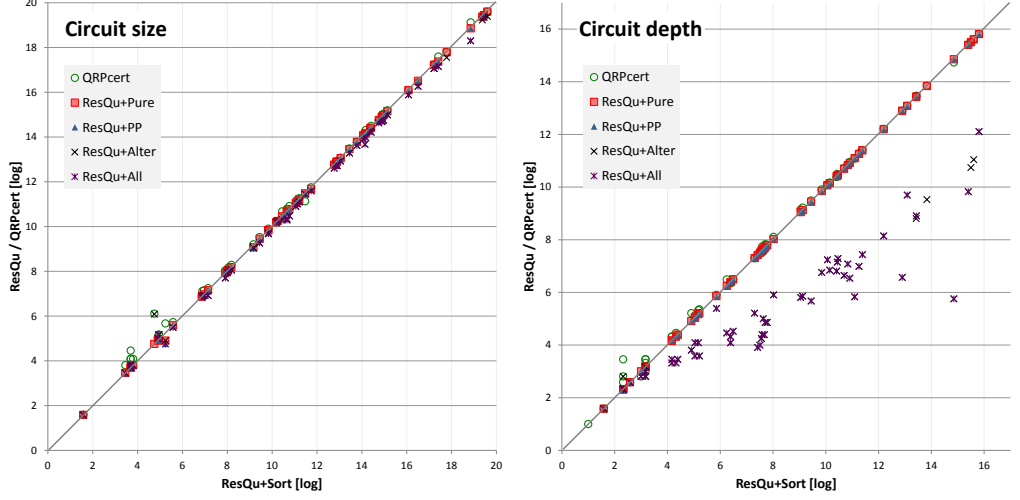


Figure 4.11: Certificate size and depth statistics for false instances.

Table 4.4: Certificate comparison summary.

		QRPCERT	RQ+SORT	RQ+PURE	RQ+PP	RQ+ALTER	RQ+ALL
True	#e/#v	130/84	130/86	130/86	124/90	98/88	105/90
	T_e	0.78	1	1	2.6	2	3.53
	T_v	1.56	1	0.99	0.5	0.91	0.48
	#aig	1.95	1	0.98	1.07	0.84	1.02
	#lvl	0.96	1	1.03	0.48	0.59	0.25
False	#e/#v	66/62	66/62	66/62	66/62	66/63	66/63
	T_e	0.88	1	1	1.1	1.19	1.19
	T_v	1.4	1	0.99	0.99	0.88	0.88
	#aig	1.03	1	0.99	1	0.89	0.89
	#lvl	1.03	1	1	1	0.11	0.11

RQ is short notation for RESQU; #e/#v is the number of computed/verified certificates; #aig represents AIG size ratio; #lvl represents circuit depth ratio.

is of concern, RESQU+SORT itself improves over QRPCERT significantly in circuit size (up to 6 times and 2 times for true and false formulas, respectively). Other optimization techniques show further reductions compared to RESQU+SORT. For true instances, we are able to achieve up to 2 times and 8 times improvements on circuit size and depth, respectively. For false cases, certificate depth can be reduced by up to 2 orders of magnitude whereas improvement on circuit size is minor.

Experiments on don't care minimization were conducted separately as it can be applied on any Skolem and Herbrand functions generated from RESQU as well as QRPCERT. The computed don't-care sets were supplied to ABC using command `mfs` for circuit minimization [11]. The improvements in both AIG size and circuit depth were on average 3% ~ 5%, compared to the same optimization procedure but

without specifying don't cares. Although the improvements were minor, don't care computation is inexpensive anyway. We note that don't-care based AIG minimization remains relatively unexplored, and certificate simplification may potentially benefit from its future improvement.

4.4 Extension of the extraction algorithm to other proof systems

Previously in Sections 4.2 and 4.3 we focused on extraction of Skolem/Herbrand functions from regular (i.e. distance-1) Q-resolution proofs. In practice, however, some QBF solvers could be equipped to produce LQ-resolution proofs [21]. Experimental evaluation shows that more benchmarks could be certified using LQ-proofs compare to regular Q-resolution. As we showed in Section 3.2, LQ-refutations can be transformed into sound Q-refutations. Therefore Skolem/Herbrand functions also could be extracted from LQ-resolution proofs, by firstly converting them to Q-refutations, and then apply the corresponding algorithm from Section 4.2. In this section we will show how to extract Skolem/Herbrand functions directly from LQ-proofs. As experiments shall show in the end of this section, direct extraction is significantly faster than extraction from converted proofs (which also confirms the theoretical analysis shown later in this section).

4.4.1 Polynomial extraction of Skolem and Herbrand functions from LQ(U+)-resolution proofs

Yet we proposed a procedure of converting LQ-refutations into Q-refutations in Section 3.2 it might be very inefficient to use it for (counter)model derivation. Below we elaborate a procedure of direct extraction of Herbrand-functions countermodel from the given LQ-refutation in time linear with respect to the proof size. By duality, a Skolem-functions model can be similarly extracted from a cube LQ-consensus for true QBF.

Similarly to Q-resolution proofs, we consider an LQ-resolution proof Π of a false QBF Φ as a directed acyclic graph (DAG) $G_\Pi(V_\Pi, E_\Pi)$, where a vertex $v \in V_\Pi$

4.4. Extension of the extraction algorithm to other proof systems

corresponds to a clause $v.\text{clause}$ in Π , and an edge $(u, v) \in E_\Pi \subseteq V_\Pi \times V_\Pi$ corresponds to the derivation of $v.\text{clause}$ from either an LQ-resolution step (i.e., $v.\text{clause} = \text{resolve}(u.\text{clause}, \cdot)$) or a universal reduction step (i.e., $v.\text{clause} = \text{reduce}(u.\text{clause})$) over $u.\text{clause}$ in Π . For $(u, v) \in E_\Pi$, we call v a *child* of u , and u a *parent* of v . To generalize, for u that reaches v through a number of connected edges, we call v a *descendant* of u , and u an *ancestor* of v .

To study the implication relations among the clauses in an LQ-resolution proof, we recall the definitions of α -implication and α -inheritance from Section 4.2.

By definition, merged literals do not follow the same semantics as ordinary literals. Given a variable x , it is not meaningful to look at a valuation of x^* under any assignment to x , as it is always true. Consequently, when forall-reduction of x^* in clause C occurs, it is semantically inconsistent to talk about α -implication property in the presence of tautological clauses, as it was done in Section 4.2. The following simple example illustrates the problem for the notion of α -implication in the presence of tautological clauses.

Example 4.13 Consider the QBF $\Phi = \exists a \forall x \exists b. (a + x + b)_1 (\bar{a} + \bar{x} + b)_2 (\bar{b})_3$ and the corresponding LQ-resolution proof Π :

$$\Pi = \left\{ \begin{array}{l} 1. C_4 = \text{resolve}(C_1, C_2) \\ 2. C_5 = \text{resolve}(C_3, C_4) \\ 3. C_\square = \text{reduce}(C_5) \end{array} \right\}$$

Note that $C_5 = \{x^*\}$, and if the semantics of a merged literal x^* is to be treated similarly to an ordinary literal, then $C_5|_\alpha = 1$ for any assignment α . Therefore C_\square cannot be α -implied. However, the empty clause is soundly deduced following the LQ-resolution proof system.

Given a false QBF Φ over variables $X = X_\exists \cup X_\forall$ and its LQ-resolution proof Π , let α_\exists be an assignment to the existential variables X_\exists . For an arbitrary vertex $v \in V_\Pi$ and an arbitrary literal $l \in v.\text{clause}$, Table 4.5 defines some additional attributes associated with v or with l , including *parent literal*, *phase function*, *effective literal*, and *shadow clause*, which are to be used in our countermodel extraction algorithm.

A *phase function* intuitively represents the induced phase of a literal in a clause under a particular assignment to the existential variables. An *effective literal* repre-

4.4. Extension of the extraction algorithm to other proof systems

Attribute	Definition
Parent literal	Literal $l' \in u.\text{clause}$ is called a <i>parent literal</i> of $l \in v.\text{clause}$, denoted $l.\text{ancestor}$, if $\text{var}(l') = \text{var}(l)$ and $(u, v) \in E_\Pi$. Note that l can only have 0, 1 or 2 parent literals.
Phase function	The <i>phase function</i> , denoted $l.\text{phase}$, of literal $l \in v.\text{clause}$ is defined as follows: <ul style="list-style-type: none"> • if l is positive, then $l.\text{phase} = 1$; • if l is negative, then $l.\text{phase} = 0$; • if l is merged and l has only one parent literal l', then $l.\text{phase} = l'.\text{phase}$; • if l is merged and l has two parent literals $l_1 \in u_1.\text{clause}$ and $l_2 \in u_2.\text{clause}$ with $(u_1, v) \in E_\Pi$ and $(u_2, v) \in E_\Pi$, then $l.\text{phase} = (l_1.\text{phase} \wedge \bar{p}) \vee (l_2.\text{phase} \wedge p)$, where pivot $p \in X_\exists$, $p \in u_1.\text{clause}$ and $\bar{p} \in u_2.\text{clause}$.
Effective literal	The <i>effective literal</i> , denoted $l.\text{elit}$, of $l \in v.\text{clause}$ is a literal that satisfies $l.\text{elit} \leftrightarrow (x \leftrightarrow l.\text{phase})$, where $x = \text{var}(l)$.
Shadow clause	The <i>shadow clause</i> , denoted $v.\text{shadcls}$, of $v \in V_\Pi$ is the clause of effective literals of v : $v.\text{shadcls} = \bigcup_{l \in v.\text{clause}} (l.\text{elit}).$

Table 4.5: Attributes of each vertex $v \in V_\Pi$ of an LQ-resolution proof Π , represented as a DAG $G_\Pi(V_\Pi, E_\Pi)$, of a false QBF Φ .

sents the induced value of its corresponding literal. Observe that, by the definition of the phase function and the effective literal, $l.\text{elit} \leftrightarrow l$ holds whenever l is not a merged literal. In essence, effective literals represent the conditional origins of merged literals in connection to ordinary literals. A *shadow clause* corresponds to the disjunction of a set of effective literals. To illustrate, consider the merged literal $x^* \in C_5$ in Example 4.13. We have $x^*.\text{elit} = (\bar{a} \leftrightarrow x)$. For partial assignment $a = 0$, we have $x^*.\text{elit} = x$ and C_2 valuating to true. Hence, proof Π can be simplified to

$$\Pi|_{\{\bar{a}\}} = \left\{ \begin{array}{l} 1. C'_4 = \text{resolve}(C_1|_{\{\bar{a}\}}, C_3|_{\{\bar{a}\}}) \\ 2. C'_\square = \text{reduce}(C'_4) \end{array} \right\}$$

Observe that the merged literal $x^* \in C_5$ in proof Π now corresponds to the ordinary

4.4. Extension of the extraction algorithm to other proof systems

Countermodel-Extract-LQ($\Phi, G_{\Pi}(V_{\Pi}, E_{\Pi})$)

input: a false QBF Φ and its LQ-res DAG $G_{\Pi}(V_{\Pi}, E_{\Pi})$
output: a countermodel in RFAO formulas

begin

```

01  foreach universal variable  $x$  of  $\Phi$ 
02    RFAO[ $x$ ] :=  $\emptyset$ ;
03  foreach vertex  $v \in V_{\Pi}$  in topological order
04    foreach merged literal  $l \in v.clause$ 
05      update  $l.phase$  from its parent literal(s);
06    if  $v.clause = reduce(u.clause)$ 
07       $C := v.shadcls$ ;
08      foreach universal literal  $l$  reduced from  $u.clause$ 
09         $x := var(l)$ ;
10        if  $x \in u.clause$ 
11          push back  $C$  to RFAO[ $x$ ];
12        else if  $\bar{x} \in u.clause$ 
13          push back  $\bar{C}$  to RFAO[ $x$ ];
14        else if  $x^* \in u.clause$ 
15          push back  $(C \vee \overline{l.phase})$  to RFAO[ $x$ ];
16          push back  $(\bar{C} \wedge \overline{l.phase})$  to RFAO[ $x$ ];
17    if  $v.clause$  is the empty clause
18      return RFAO formulas;
end

```

Figure 4.12: Algorithm extracting a countermodel from an LQ-resolution proof.

literal $x \in C'_4$, which is equivalent to $x^*.elit$ under our partial assignment $a = 0$. On the other hand, under partial assignment $a = 1$, a similar correspondence can be observed. Therefore, effective literals intuitively represent how merged literals should be interpreted under a given (partial) assignment.

The procedure, *Countermodel-Extract-LQ*, to extract Herbrand functions from a given LQ-resolution proof is outlined in Figure 4.12. It is similar to the procedure *Countermodel-Construct*, but with two main differences: First, shadow clauses, rather than ordinary clauses, are used to construct RFAO formulas. Second, merged literals are processed as in Lines 14-16. Notice that Line 5 uses the definition of phase functions given in Table 4.5.

Example 4.14 illustrates the computation steps of algorithm *Countermodel-Extract-LQ*.

4.4. Extension of the extraction algorithm to other proof systems

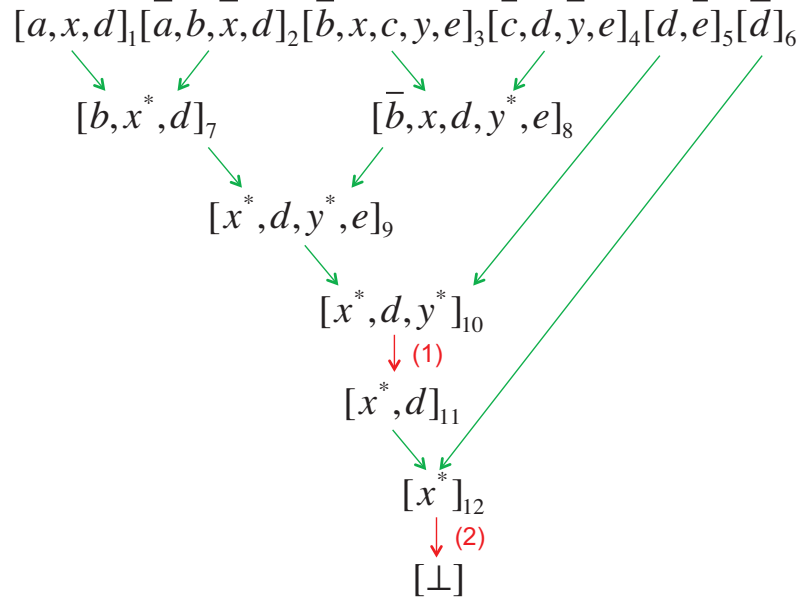


Figure 4.13: DAG of LQ-resolution proof II.

Example 4.14 Consider the false QBF Φ with its prefix and matrix defined as follows.

$$\Phi_{\text{pfx}} = \exists ab \forall x \exists cd \forall y \exists e$$

$$\Phi_{\text{mtx}} = (a, x, d)_1 (\bar{a}, b, \bar{x}, d)_2 (\bar{b}, x, c, y, e)_3 (\bar{c}, d, \bar{y}, e)_4 (d, \bar{e})_5 (\bar{d})_6$$

Its falsity can be established by the following LQ-resolution proof Π , also visualized in the DAG of Figure 4.14.

$$\Pi = \left\{ \begin{array}{l} 1. C_7 = \text{resolve}(C_1, C_2) \\ 2. C_8 = \text{resolve}(C_3, C_4) \\ 3. C_9 = \text{resolve}(C_7, C_8) \\ 4. C_{10} = \text{resolve}(C_5, C_9) \\ 5. C_{11} = \text{reduce}(C_{10}) \\ 6. C_{12} = \text{resolve}(C_{11}, C_6) \\ 7. C_{\square} = \text{reduce}(C_{12}) \end{array} \right\}$$

The phase functions f_i and g_i corresponding to the literals of variables x and y present in the clause C_i can be derived as shown in Figure 4.14, to the left and to the right, respectively. Note that, if $\text{lit}(x) \notin C_i$ (resp. $\text{lit}(y) \notin C_i$), then $f_i = \emptyset$ (resp.

4.4. Extension of the extraction algorithm to other proof systems

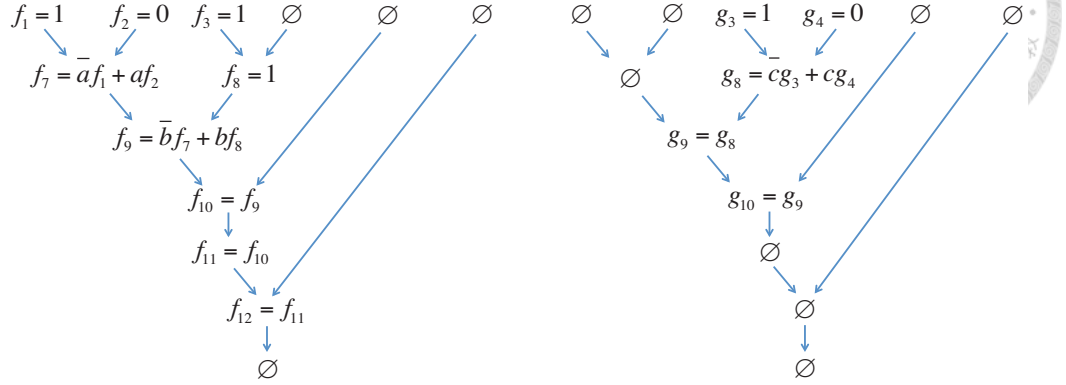


Figure 4.14: Phase functions f and g for literals of variables x and y .

$g_i = \emptyset$).

The RFAO array contents for variables x and y as generated by algorithm *Countermodel-Extract-LQ* after each \forall -reduction step in the proof of Figure 4.14 are shown below.

0.	$x :$	$\begin{bmatrix} \\ \end{bmatrix}$	$y :$	$\begin{bmatrix} \\ \end{bmatrix}$
1.	$x :$	$\begin{bmatrix} \\ \end{bmatrix}$	$y :$	$\begin{bmatrix} clause(f_{10}^e, d, \overline{g_{10}}) \\ cube(\overline{f_{10}^e}, \overline{d}, \overline{g_{10}}) \end{bmatrix}$
2.	$x :$	$\begin{bmatrix} clause(\overline{f_{12}}) \\ cube(\overline{f_{12}}) \end{bmatrix}$	$y :$	$\begin{bmatrix} clause(f_{10}^e, d, \overline{g_{10}}) \\ cube(\overline{f_{10}^e}, \overline{d}, \overline{g_{10}}) \end{bmatrix}$

Note that f_{10}^e , which equals $(x \wedge f_{10}) \vee (\overline{x} \wedge \overline{f_{10}})$, stands for the effective literal of x^* in C_{10} .

After re-expressing the RFAO formulas in terms of the existential variables, we get the Herbrand functions of x and y ,

4.4. Extension of the extraction algorithm to other proof systems



$$\begin{aligned}
 F[x] &= \overline{(f_{12})} \wedge \overline{(f_{12})} \\
 &= \overline{(\bar{b}f_7 + bf_8)} \\
 &= \overline{\bar{b}(\bar{a}f_1 + af_2) + b} \\
 &= a\bar{b} \\
 F[y] &= (f_{10}^e + d + \bar{g}_{10}) \wedge (\bar{f}_{10}^e \bar{d} \bar{g}_{10}) \\
 &= \overline{(\bar{f}_{10}^e \bar{d} \bar{g}_{10})} = \overline{(xf_{10}) + (\bar{x}\bar{f}_{10})} \bar{d}\bar{g}_8 \\
 &= \overline{(x(\bar{b}\bar{a} + b)) + (\bar{x}(\bar{b}\bar{a} + b))} \bar{d}c \\
 &= \bar{d}c
 \end{aligned}$$

It can be verified that, after substituting $F[x]$ and $F[y]$ for variables x and y in Φ_{mtx} , the obtained quantifier-free formula is indeed unsatisfiable.

The correctness of algorithm *Countermodel-Extract-LQ* of Figure 4.12 is asserted by the following theorem.

Theorem 4.3 Given a false QBF Φ and its LQ-resolution proof Π , the algorithm *Countermodel-Extract-LQ*(Φ, G_Π) produces a countermodel of Herbrand functions for the universal variables of Φ .

To prove Theorem 4.3, we need to show that 1) the Herbrand functions returned by *Countermodel-Extract-LQ* obey the prefix order dependency (i.e., the Herbrand function $F[x]$ of universal variable x only refers to the variables with quantification levels less than that of x), and 2) their substitution for corresponding universal variables indeed makes the matrix Φ_{mtx} unsatisfiable. Proposition 4.13 establishes the first part, and Lemma 4.2 the second part.

Proposition 4.13 Given a false QBF Φ and its LQ-resolution proof Π , let literal $l \in v.\text{clause}$ with $v \in V_\Pi$. If the truth or falsity of $l.\text{elit}$ refers (through recursive substitution of phase functions) to some variable x , then $lv_l(x) \leq lv_l(l)$.

Proof Observe that the proposition holds, by the definition of effective literals, for any literal l that is not a merged literal. Since the clauses in Φ_{mtx} do not involve any merged literals, the proposition holds for all the literals in the clauses of Φ_{mtx} . On

4.4. Extension of the extraction algorithm to other proof systems

the other hand, for a merged literal $l \in v.clause$, if $v.clause = reduce(u.clause)$ in Π , and l' denotes $l.ancestor$, then $l.elit = l'.elit$ according to the definition of effective literals. Now, if $v.clause = resolve(u_1.clause, u_2.clause)$ and $var(l) \notin vars(u_1.clause)$ (resp. $var(l) \notin vars(u_2.clause)$), then $l.elit = l'.elit(l)$, where l' represents $l.ancestor$. On the other hand, if $v.clause = resolve(u_1.clause, u_2.clause)$ under pivot variable p and $var(l) \in vars(u_1.clause)$ and $var(l) \in vars(u_2.clause)$, then this is an LQ-resolution step (if it is not, then $l \in v.clause$ cannot be a merged literal). By the rule of LQ-resolution, $lvl(p) < lvl(l)$. Therefore, regardless of the origin of v from either universal reduction or resolution, if the proposition holds for any literal in each parent of $v \in V_\Pi$, then it must also hold for the literals in $v.clause$. By induction, the proposition holds for any $l \in v.clause$ for any $v \in V_\Pi$. ■

Similarly to Proposition 4.13, we can prove that for any $l \in v.clause$, function $l.phase$ only refers to the variables with quantification level less than $lvl(l)$. Taking into account the construction of procedure *Countermodel-Extract-LQ*, the following corollary follows.

Corollary 4.2 Herbrand functions returned by the algorithm *Countermodel-Extract-LQ* obey the prefix order dependency.

Proof Given a universal reduction step $v.clause = reduce(u.clause)$, and a literal $l \in u.clause$ such that $l \notin v.clause$ (i.e., l is a reduced literal), it holds that $lvl(l) > lvl(l')$ for any $l' \in v.clause$ by the definition of universal reduction. Therefore, by Proposition 4.13, the truth or falsity of $v.shadcls$ only refers to the variables with quantification level less than $lvl(l)$. Similarly function $l.phase$ only refers to the variables with quantification level less than $lvl(l)$. Hence for each universal variable $var(l)$, its corresponding RFAO node array only refers to the variables with quantification level less than $lvl(l)$. ■

To prove that Herbrand functions returned by *Countermodel-Extract-LQ* form a countermodel, we follow a similar line of reasoning as for proving the correctness of *Countermodel-Construct*. However, the algorithm *Countermodel-Extract-LQ* stores shadow clauses (cubes) in RFAO arrays rather than ordinary clauses (cubes) and it considers universal reduction on merged literals. Note that, if no LQ-resolution step is present in a proof (i.e., no merged literal appears), then *Countermodel-Extract-LQ* returns exactly the same Herbrand functions as *Countermodel-Construct*.

Regardless of the change from ordinary to shadow clauses (cubes), the two RFAO

4.4. Extension of the extraction algorithm to other proof systems

formula properties listed in Section 4.2 remain intact. In the following, when we say that some shadow clause of vertex v is α -implied, we mean it is α -implied by the shadow clause (the conjunction of the shadow clauses) corresponding to the parent vertex (parent vertices) of v . Lemma 4.1 shows the properties of α -implication among shadow clauses.

Lemma 4.1 Given a false QBF Φ and its LQ-resolution proof Π , let $v \in V_\Pi$ and $v.\text{clause} = \text{resolve}(u_1.\text{clause}, u_2.\text{clause})$ in Π with pivot literals $p \in u_1.\text{clause}$ and $\bar{p} \in u_2.\text{clause}$. Then $(u_1.\text{shadcls}|_\alpha \wedge u_2.\text{shadcls}|_\alpha) \rightarrow v.\text{shadcls}|_\alpha$ under any assignment α to the variables in Φ .

Proof Let $\text{vars}(u_1.\text{clause}) = \{p\} \cup L_1 \cup M$ and $\text{vars}(u_2.\text{clause}) = \{p\} \cup L_2 \cup M$, where L_1 and L_2 are the sets of variables *local* to $u_1.\text{clause}$ and $u_2.\text{clause}$, respectively, and M is the set of their *common* variables, excluding the pivot variable p . If $l.\text{elit}|_\alpha = 1$ for some $l \in v.\text{clause}$, then by the definition of shadow clauses $v.\text{shadcls}|_\alpha = 1$. Therefore $(u_1.\text{shadcls}|_\alpha \wedge u_2.\text{shadcls}|_\alpha) \rightarrow v.\text{shadcls}|_\alpha$.

Consider the other case that $l.\text{elit}|_\alpha = 0$ for each $l \in v.\text{clause}$. Without loss of generality, assuming $\bar{p} \in \alpha$, we prove $u_1.\text{shadcls}|_\alpha = 0$ in the following. (Assuming $p \in \alpha$, $u_2.\text{shadcls}|_\alpha = 0$ can be proved similarly).

For each l_1 with $\text{var}(l_1) \in L_1$ by the definition of effective literals, it holds that $l_1.\text{elit} = l'_1.\text{elit}$, where $l'_1 \in u_1$ is a parent of l_1 . Hence if $l_1.\text{elit}|_\alpha = 0$, then $l'_1.\text{elit}|_\alpha = 0$. Further, for each literal l with $\text{var}(l) \in M$, we have $l.\text{phase}|_\alpha = ((\bar{p} \wedge l'.\text{phase}) \vee (p \wedge l''.\text{phase}))|_\alpha = l'.\text{phase}|_\alpha$, where $l' \in u_1$ and $l'' \in u_2$ are the parents of l . Therefore $l.\text{elit}|_\alpha = (x \leftrightarrow l.\text{phase})|_\alpha = (x \leftrightarrow l'.\text{phase})|_\alpha = l'.\text{elit}|_\alpha = 0$, where $x = \text{var}(l)$. Consequently, $l'.\text{elit}|_\alpha = 0$ for each l' with $\text{var}(l') \in \{p\} \cup L_1 \cup M$, and thus $u_1.\text{shadcls}|_\alpha = 0$.

Thereby $(u_1.\text{shadcls}|_\alpha \wedge u_2.\text{shadcls}|_\alpha) \rightarrow v.\text{shadcls}|_\alpha$ under any assignment α , and the lemma follows. ■

Finally, the following lemma shows that the substitution of all universal variables by their corresponding Herbrand functions returned by *Countermodel-Extract-LQ*(Φ, G_Π) indeed makes Φ_{mtx} unsatisfiable, and thus completes the proof of Theorem 4.3.

Lemma 4.2 Given a false QBF Φ and its LQ-resolution proof Π , the algorithm

4.4. Extension of the extraction algorithm to other proof systems

Countermodel-Extract-LQ(Φ, G_Π) returns Herbrand functions whose substitution for the corresponding universal variables makes the matrix Φ_{mtx} unsatisfiable.

Proof Given an assignment α_\exists to the existential variables of Φ , we show below that the constructed Herbrand functions induce an assignment α_\forall to the universal variables of Φ such that $\Phi_{\text{mtx}}|_\alpha = 0$ for $\alpha = \alpha_\exists \cup \alpha_\forall$.

Let V_D be the set of all vertices $v \in V_\Pi$ whose clauses were obtained by universal reduction in Π (i.e., $v.\text{clause} = \text{reduce}(u.\text{clause})$ for some $u \in V_\Pi$). Notice that algorithm *Countermodel-Extract-LQ* processes G_Π in a topological order, meaning that a clause in Π is processed only after all of its ancestor clauses are processed. Therefore we consider all shadow clauses $v.\text{shadcls}$ with $v \in V_D$ in the topological order under the assignment α . First, assume that for each $v \in V_D$ its corresponding shadow clause $v.\text{shadcls}$ satisfies $v.\text{shadcls}|_\alpha = 1$, and therefore is α -implied. By Lemma 4.1, we conclude that in this case $v.\text{shadcls}$ is α -implied for any $v \in V_\Pi$. Hence the empty shadow clause (that corresponds to the empty clause) is α -inherited, and thus $\Phi_{\text{mtx}}|_\alpha = 0$.

Second, assume that for some vertex $v \in V_D$, the corresponding shadow clause $v.\text{shadcls}|_\alpha = 0$. Let $v'.$ *shadcls* be the first such encountered shadow clause. Denote u' as the parent of v' . Note that $u'.$ *shadcls* and all its ancestors must be α -inherited (as all the ancestors of $u'.$ *shadcls* are α -implied). Let $C_{u' \setminus v'}$ be the set of all the universal literals being reduced from $u'.$ *clause* to get $v'.$ *clause*. By the definition of shadow clauses, we have $u'.$ *shadcls* = $v'.$ *shadcls* $\bigcup_{l \in C_{u' \setminus v'}} l.\text{elit}$. It follows that

$$u'.$$
shadcls $|_\alpha = v'.$ *shadcls* $|_\alpha \vee \bigvee_{l \in C_{u' \setminus v'}} l.\text{elit}|_\alpha = \bigvee_{l \in C_{u' \setminus v'}} l.\text{elit}|_\alpha.$

Next we show that our construction yields $l.\text{elit}|_\alpha = 0$ for any $l \in C_{u' \setminus v'}$, therefore leading to $u'.$ *shadcls* $|_\alpha = 0$. For each variable $x = \text{var}(l)$ with $l \in C_{u' \setminus v'}$, we examine its corresponding $\text{RFAO}[x]$. Since $w.\text{shadcls}|_\alpha = 1$ for any ancestor $w \in V_D$ of v' , the value of Herbrand function $F[x]$ under α is not determined by any of the RFAO nodes that were added to the RFAO array before the reduction of x happens in u' (by Property 1 of RFAO arrays mentioned in Section 4.2). We now analyze the following three cases for the reduction of x in $u'.$ *clause*:

1. For x being reduced as a positive literal l , the corresponding RFAO clause node evaluates to 0 (since $v'.$ *shadcls* $|_\alpha = 0$), and hence $F[x]|_\alpha = 0$ (by Property 2 of

4.4. Extension of the extraction algorithm to other proof systems

RFAO arrays mentioned in Section 4.2).

2. Similarly, for x being reduced as a negative literal l , the corresponding RFAO cube node evaluates to 1 (since $(v'.shadcub)|_\alpha = 1$), and hence $F[x]|_\alpha = 1$.
3. For x being reduced as a merged literal l , by algorithm *Countermodel-Extract-LQ* two RFAO nodes, namely, clause node $Node_1 = (v'.shadcls \vee \overline{l.phase})$ and cube node $Node_2 = (v'.shadcub \wedge \overline{l.phase})$, are added to the RFAO array. Now, if $l.phase|_\alpha = 0$, then $Node_1|_\alpha = Node_2|_\alpha = 1$. Therefore $F[x]|_\alpha = 1$ (determined by the RFAO cube $Node_2|_\alpha = 1$) and $l.elit|_\alpha = ((F[x] \wedge l.phase) \vee (\overline{F[x]} \wedge \overline{l.phase}))|_\alpha = 0$. On the other hand, if $l.phase|_\alpha = 1$, then $Node_1|_\alpha = Node_2|_\alpha = 0$. Therefore $F[x]|_\alpha = 0$ (determined by the RFAO clause $Node_1|_\alpha = 0$) and $l.elit|_\alpha = 0$ similarly.

By the above analysis, it holds that $l.elit|_\alpha = 0$ for any $l \in C_{u' \setminus v'}$. Therefore $u'.shadcls|_\alpha = 0$, and taking into account that $u'.shadcls$ is α -inherited we establish $\Phi_{mtx}|_\alpha = 0$. ■

Algorithm *Countermodel-Extract-LQ* has a linear time complexity because each vertex is processed once by traversing its literals once. When a clause or cube is pushed into an RFAO array, only its ID needs to be stored. Therefore proposed algorithm has theoretical exponential outperformance compare to transformation of LQ-proofs into Q-proofs first, and then applying ordinary algorithm *Countermodel-Construct* from Section 4.2.

4.4.2 Experimental evaluation

Algorithm of converting LQ-refutations into Q-refutations was equipped into RESQU. Resulting tool we call as RESQU-LQU. The countermodel extraction algorithm of Figure 4.12, introduced in the last subsection, and its dual model extraction algorithm were also both implemented in the C++ language in the tool RESQU, and named the new implementation as RESQU-LP (or RESQU-LPOLY from “long-distance” and “polynomial”). The experiments were conducted on the same as in the last subsection Linux machine with a Xeon 2.53 GHz CPU and 48 GB RAM for two sets of test cases: the KBKF family of formulas [29] and the benchmark formulas of QBFEVAL’10 [37]. With the test cases, we evaluated the performance

4.4. Extension of the extraction algorithm to other proof systems

of RESQU-LP in terms of runtime, memory consumption, and the quality of the produced countermodels (i.e., the circuit size and depth of the constructed Herbrand functions).

DEPQBF [31] solver was used in these experiments as it has an option to produce both Q-refutations and LQ-refutations for the same QBF-formula. Therefore, we applied DEPQBF without and with long-distance resolution (command line parameter “`--long-dist-res`” [21] to produce Q- and LQ-resolution proofs for the aforementioned sets of benchmarks, respectively. Then we compared the model and countermodel extraction in three settings: RESQU for Q-resolution proofs, and RESQU-LQU and RESQU-LP for LQ-resolution proofs. Further, to validate the constructed models and countermodels, the SAT solver MINISAT [20] embedded in ABC [11] was applied.

Table 4.6 shows the runtime statistics for QBF solving, countermodel extraction, and countermodel validation for the KBKF family of QBF instances, which are all false and hard for Q-resolution, but easy for LQ-resolution, to refute [21, 29]. Column 1 lists the instances indexed by parameter t , which reflects the formula size (there are $3t + 2$ variables and $4t + 1$ clauses in the t^{th} member of the KBKF family). Columns denoted by DEPQBF and DEPQBF-L indicate QBF solver settings without and with long distance resolution, respectively. The columns “slv,” “ext,” and “vld” report the CPU runtime in seconds for QBF solving, certificate extraction, and certificate validation, respectively. An entry containing “-1” indicates that the computation was either out of the time limit of 1,000 seconds, or out of the memory limit of 25 GB. An entry containing “-” indicates that the data is not available. As evident from Table 4.6, DEPQBF required runtime (and, in fact, yielded proof size) exponential in t , whereas DEPQBF-L required runtime (and, in fact, yielded proof size) linear in t . RESQU was able to extract countermodels from all the proofs produced by DEPQBF for $t \leq 20$, but the cases with $t \geq 16$ could not be validated within the time limit. On the other hand, RESQU-LQU was only able to extract countermodels for $t \leq 20$ from the proofs produced by DEPQBF-L within the time limit, while the cases with $t \geq 18$ could not be validated within the time limit. In comparison, RESQU-LP easily accomplished every extraction task within 0.4 seconds under 6 MB memory consumption; its produced countermodels were all validated within 0.1 seconds.

Continuing the above experiments, Table 4.7 shows the circuit sizes in terms of

4.4. Extension of the extraction algorithm to other proof systems

Table 4.6: Time statistics (in seconds) for KBKF instances.

t	DEPQBF	RESQU		DEPQBF-L	RESQU-LQU		RESQU-LP	
	slv	ext	vld	slv	ext	vld	ext	vld
10	0.1	0.1	0.1	0.0	0	0.1	0.0	0.1
11	0.2	0.1	0.3	0.0	0.1	0.1	0.0	0.1
12	0.5	0.3	0.7	0.0	0.1	0.1	0.0	0.1
13	1.2	0.6	2.3	0.0	0.3	0.1	0.0	0.1
14	2.8	1.4	7.6	0.0	0.7	0.1	0.0	0.1
15	6.8	3.0	30.5	0.0	1.8	0.1	0.0	0.1
16	16.6	6.7	-1	0.0	3.9	0.8	0.0	0.1
17	41.0	15.1	-1	0.0	9.4	5.4	0.0	0.1
18	102.8	33.6	-1	0.0	20.5	40.4	0.0	0.1
19	261.5	74.1	-1	0.0	48.8	-1	0.0	0.1
20	674.2	175.7	-1	0.0	95.1	-1	0.0	0.1
30	-1	-	-	0.0	-1	-	0.0	0.1
40	-1	-	-	0.0	-1	-	0.1	0.1
50	-1	-	-	0.0	-1	-	0.1	0.1
60	-1	-	-	0.0	-1	-	0.1	0.1
70	-1	-	-	0.0	-1	-	0.2	0.1
80	-1	-	-	0.0	-1	-	0.3	0.1
90	-1	-	-	0.0	-1	-	0.3	0.1
100	-1	-	-	0.0	-1	-	0.4	0.1

“slv” stands for the CPU solving time in seconds; “ext” stands for the countermodel extraction time; “vld” stands for countermodel validation time.

the number of and-inverter graph (AIG) nodes, denoted “#AIG,” and circuit depths, denoted “#LVL,” to evaluate the quality of the extracted Herbrand functions. An entry containing “-” indicates that either the data is unavailable or ABC failed to read in the certificate due to its excessive size. Note that the simplified AIGs of the Herbrand functions of KBKF instance t produced by RESQU-LP have t AIG nodes and are significantly smaller than the corresponding functions produced by other methods.

To evaluate the performance of RESQU-LP on application benchmark formulas, we conducted the above experiments on the QBFEVAL’10 instances. Out of the 569 formulas, DEPQBF-L was able to generate resolution proofs for 177 false QBFs and 98 true QBFs within the limits of 1,000 seconds, 2 GB RAM, and 1 GB proof size. Among the 177 proofs of falsity, 144 involved only Q-resolution proofs, and

4.4. Extension of the extraction algorithm to other proof systems

Table 4.7: Certificate sizes for KBKF instances.

t	RESQU		RESQU-LQU		RESQU-LP	
	#AIG	#LVL	#AIG	#LVL	#AIG	#LVL
10	3.1k	1.6k	93	10	10	2
11	6.1k	3.1k	231	13	11	2
12	11.9k	6.1k	532	16	12	2
13	24.6k	12.3k	840	17	13	2
14	47.5k	24.5k	1.5k	19	14	2
15	95.1k	49.1k	2.7k	20	15	2
16	253.9k	82.0k	16.4k	29	16	2
17	-	-	61.6k	32	17	2
18	-	-	132.8k	33	18	2
19	-	-	-	-	19	2
20	-	-	-	-	20	2
30	-	-	-	-	30	2
40	-	-	-	-	40	2
50	-	-	-	-	50	2
60	-	-	-	-	60	2
70	-	-	-	-	70	2
80	-	-	-	-	80	2
90	-	-	-	-	90	2
100	-	-	-	-	100	2

“slv” stands for the CPU solving time in seconds; “ext” stands for the countermodel extraction time; “vld” stands for countermodel validation time.

33 involved LQ-resolution proofs; on the other hand, the 98 proofs of truth all involved only Q-resolution proofs. Therefore, we focused on the 33 instances where LQ-resolution proofs were available for comparison. To compare the quality of the extracted certificates, Figure 4.15 plots the results in terms of AIG size and circuit depth. The x axis in the figure corresponds to the results obtained by RESQU-LP, and the y axis corresponds to those obtained by RESQU and RESQU-LQU. Both axes are presented in the \log_{10} scale, and an index k indicates 10^k . If a method failed to extract a certificate, then the corresponding result was set to the upper bound of the figure. We note that DEPQBF was not able to produce Q-resolution proofs for three formulas whose LQ-resolution proofs were obtainable. Hence, extracting certificates from LQ-resolution proofs is beneficial for certain applications. As shown in Figure 4.15, certificates produced by RESQU-LP are consistently smaller than those produced by RESQU-LQU. Furthermore RESQU-LQU was not able to produce

4.4. Extension of the extraction algorithm to other proof systems

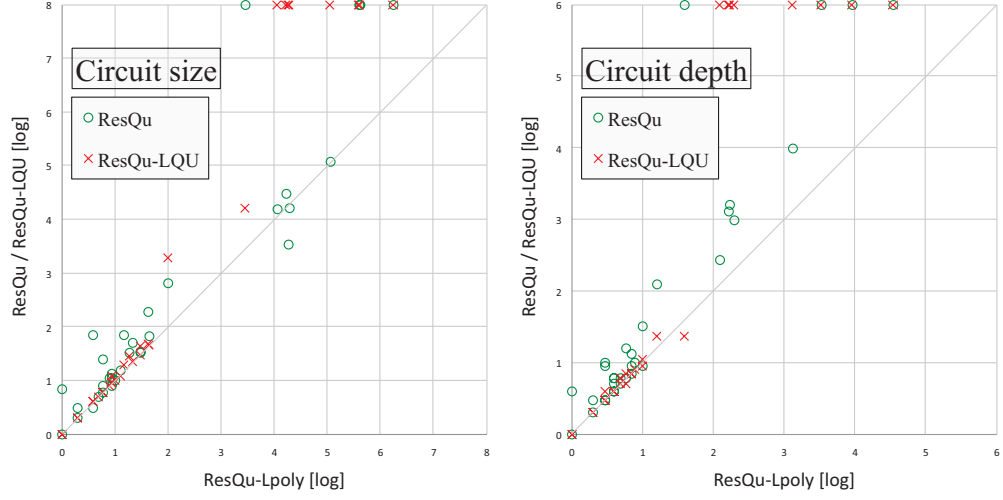


Figure 4.15: Comparison on certificate quality for application benchmarks.

certificates for eight instances, whereas RESQU-LP handled them without difficulty. The certificates extracted by RESQU-LQU are on average 16% smaller than those produced by RESQU in both AIG size and depth; in contrast, the certificates produced by RESQU-LP are on average 45% smaller in AIG size and 55% smaller in AIG depth than those extracted by RESQU. These results suggest not only the distinct value of LQ-resolution, but also the effectiveness of our algorithm in extracting high-quality certificates.



Chapter 5

Beyond Quantified Boolean Formulae

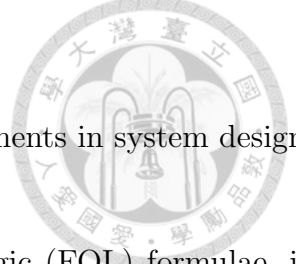
Note that in previous chapters we focused only on aspects of QBF solving and certification. *Dependency quantified Boolean formulas (DQBFs)* form a natural extension of QBF by allowing more flexible quantifier constraints. DQBF is also an emerging research area, since in addition to QBF applications some particular restrictions on desired countermodels have to be imposed by designers. We are going to devote this chapter completely for DQBF review. We shall introduce DQBF semantic classification that is inspired by QBF as its special case.

5.1 Dependency Quantified Boolean Formulas

5.1.1 Introduction to Henkin quantifiers

Dependencies in QBF followed a specific linear order. Henkin quantifiers (or branching quantifiers) [25] generalize QBF by admitting explicit specification, for an existentially quantified variable, about its dependence on universally quantified variables. In addition to mathematical logic, Henkin quantifiers appear not uncommonly in various contexts, such as natural languages [35], computation [13], game theory [36], and even system design. They permit the expression of (in)dependence in language, logic and computation, the modelling of incomplete information in noncooperative games,

5.1. Dependency Quantified Boolean Formulas



and the specification of partial dependencies among components in system design, which is the main motivation of this chapter.

When Henkin quantifiers are imposed on first-order logic (FOL) formulae, it results in the formulation of independence-friendly (IF) logic [26], which was shown to be more expressive than first-order logic and exhibit expressive power same as existential second-order logic. However one notable limitation among others of IF logic under the game-theoretical semantics is the violation of the law of the excluded middle, which states either a proposition or its negation is true. Therefore negating a formula can be problematic in terms of truth and falsity. From a game-theoretical viewpoint, it corresponds to undetermined games, where there are cases under which no player has a winning strategy. Moreover, the winning strategies of the semantic games do not exactly correspond to Skolem and Herbrand functions in synthesis applications although syntactic rules for negating IF logic formulae were suggested in [18, 19].

When Henkin quantifiers are imposed on Boolean formulae, it results in the so-called dependency quantified Boolean formulae (DQBFs), whose satisfiability lies in the complexity class of NEXPTIME-complete [36]. In contrast to QBFs, whose evaluation is PSPACE-complete, DQBFs offer more succinct descriptive power than QBFs provided that NEXPTIME is not in PSPACE. By expansion on universally quantified variables, a DQBF can be converted to a QBF with the cost of exponential blow up in formula size [15, 16].

In this section we study DQBFs from a synthesis perspective. By distinguishing formula negation and complement, the connection between Skolem and Herbrand functions is established. While the law of the excluded middle holds for negation, it does not hold for complement. The special subset of DQBFs whose truth and falsity coincide with the existence of Skolem and Herbrand functions, respectively, is characterized. Our formulation provides a unified view on DQBF models and countermodels, which encompasses QBFs as a special case. We study some fundamental properties of DQBFs in Section 5.2. In Section 5.3, the Q-resolution rule of QBFs is extended to DQBFs and the resultant resolution, called DQ-resolution, is shown to be sound but incomplete.



5.1.2 DQBF preliminaries

Main difference between *dependency quantified Boolean formula* (DQBF) versus QBF is in its allowance for explicit specification of variable dependencies. I.e. syntactically a DQBF Φ is the same as a QBF except that in Φ_{pfx} an existential variable y_i is annotated with the set $S_i \subseteq X$ of universal variables referred to by its Skolem function, denoted as $\exists y_{i(S_i)}$, or a universal variable x_j is annotated with the set $H_j \subseteq Y$ of existential variables referred to by its Herbrand function, denoted as $\forall x_{j(H_j)}$, where S_i and H_j are called the *support sets* of y_i and x_j , respectively. However, either the dependencies for the existential variables or the dependencies for the universal variables (but not both) shall be specified. That is, a prenex DQBF is in either of the two forms:

$$\text{S-form: } \forall x_1 \cdots \forall x_n \exists y_{1(S_1)} \cdots \exists y_{m(S_m)} \cdot \phi, \text{ and} \quad (5.1)$$

$$\text{H-form: } \forall x_{1(H_1)} \cdots \forall x_{n(H_n)} \exists y_1 \cdots \exists y_m \cdot \phi, \quad (5.2)$$

where ϕ is some quantifier-free formula. Note that *the syntactic quantification order in the prefix of a DQBF is immaterial and can be arbitrary* because the variable dependencies are explicitly specified by the support sets. Such quantification with dependency specification corresponds to the Henkin quantifier [25].¹

By the above syntactic extension of DQBFs, the inputs of the Skolem (respectively Herbrand) function of an existential (respectively universal) variable can be explicitly specified, rather than inferred from the syntactic quantification order. That is, an existential variable y_i (respectively universal variable x_j) can be specified to be semantically independent of a universal variable (respectively an existential variable) whose syntactic scope covers y_i (respectively x_j). Unlike the totally ordered set formed by those of a QBF, the support sets of the existential or universal variables of a DQBF form a partially ordered set in general. This extension makes DQBFs more succinct in expressive power than QBFs [36].

For the semantics, the truth and falsity of a DQBF can be interpreted by the existence of Skolem and Herbrand functions. Precisely an S-form (respectively H-form) DQBF is true (respectively false) if and only if its Skolem (respectively

¹Henkin quantifiers in their original proposal [25] specify dependencies for existential variables only. The dependencies are extended in this paper to universal variables.

5.1. Dependency Quantified Boolean Formulas

Herbrand) functions exist for the existential (respectively universal) variables while the specified variable dependencies are satisfied. Consequently, Skolem functions serve as the model to a true S-form DQBF whereas Herbrand functions serve as the countermodel to a false H-form DQBF.

Alternatively, the truth and falsity of a DQBF can be understood from a game-theoretic viewpoint. Essentially an S-form DQBF can be interpreted as a game played by one \forall -player and m noncooperative \exists -players [36]. An S-form DQBF is true if and only if the \exists -players have winning strategies, which correspond to the Skolem functions. Similarly an H-form DQBF can be interpreted as a game played by one \exists -player and n noncooperative \forall -players. An H-form DQBF is false if and only if the \forall -players have winning strategies, which correspond to the Herbrand functions.

As was shown in [15, 16], an S-form DQBF Φ can be converted to a *logically equivalent*² QBF Φ' by formula expansion on the universal variables. Assume that universal variable x_1 is to be expanded in Formula (5.1) and $x_1 \notin S_1 \cup \dots \cup S_{k-1}$ and $x_1 \in S_k \cap \dots \cap S_m$. Then Formula (5.1) can be expanded to

$$\begin{aligned} & \forall x_2 \dots \forall x_n \exists y_{1(S_1)} \dots \exists y_{k-1(S_{k-1})} \\ & \exists y_{k(S_k[x_1/0])} \exists y_{k(S_k[x_1/1])} \dots \exists y_{m(S_m[x_1/0])} \exists y_{m(S_m[x_1/1])} \cdot \phi|_{x_1=0} \wedge \phi|_{x_1=1}, \end{aligned}$$

where $S_i[x_1/v]$ denotes x_1 in S_i is substituted with logic value $v \in \{0, 1\}$, and $\phi|_{x_1=v}$ denotes all appearances of x_1 in ϕ are substituted with v including those in the support sets of variables $y_{i(S_i)}$ for $i = k, \dots, m$. (The subscript of the support set of an existential variable are helpful for tracing expansion paths. Different expansion paths of an existential variable result in distinct existential variables.) Such expansion can be repeatedly applied for every universal variables. The resultant formula after expanding all universal variables is a QBF, whose variables are all existentially quantified. As to be shown in Section 5.2, expansion can be applied also to H-form DQBFs.

²That is, Φ and Φ' characterize the same set of Skolem-function models (by properly relating the existential variables of Φ' to those of Φ).

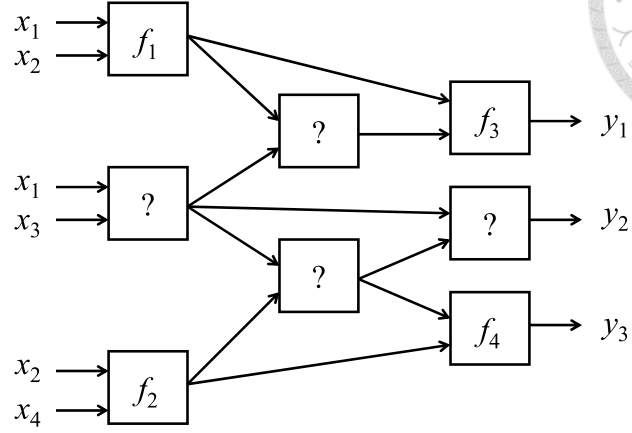


Figure 5.1: A network of known and unknown logic components

5.1.3 Applications and prior work

Although DQBF evaluation is a new and relatively unexplored field, it has potential broad variety of applications. To date there is only one search based DQBF solver [22] extended from the Q-DPLL algorithm [12,17]. We note that the framework provided by the QBF solver sKizzo [9], which is based on Skolemization, can also be naturally extended to DQBF solving. In addition to evaluation, certification of DQBFs, the focus of this work, is equally important in enabling practical applications.

One of the potential applications of DQBFs is topologically constrained logic synthesis [38], where a set of unknown components in a given Boolean network is to be synthesized such that the resultant network behavior conforms to a system specification. Figure 5.1 depicts one such example, where the network consists of four known and four unknown function components each with two inputs. Given the fixed connection of the network topology and some Boolean relation specifying the set of allowed input-output values, the Boolean functions of the four unknown components are to be synthesized.

A special problem of topologically constrained logic synthesis is shown in Figure 5.2, where m unknown functions f_1, \dots, f_m are to be synthesized from a Boolean relation specification $R(\vec{x}, \vec{y})$ with $\vec{x} = (x_1, \dots, x_n)$ and $\vec{y} = (y_1, \dots, y_m)$ such that $y_i = f_i(\vec{x}_i)$ with \vec{x}_i , a sub-vector of \vec{x} , being the pre-specified input constraint of f_i ,

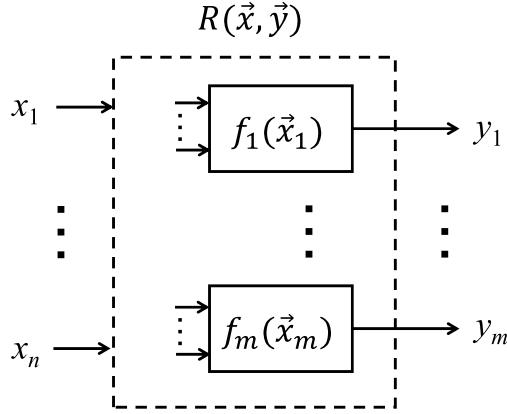


Figure 5.2: Function derivation from a Boolean relation

for $i = 1, \dots, m$. It can be naturally expressed by the S-form DQBF

$$\forall x_1 \cdots \forall x_n \exists y_1(\vec{x}_1) \cdots \exists y_m(\vec{x}_m). R(\vec{x}, \vec{y}).$$

Then the Skolem functions to the DQBF correspond to the desired synthesis solution. This synthesis problem is trivially the same as deriving Herbrand functions to the H-form DQBF

$$\exists x_1 \cdots \exists x_n \forall y_1(\vec{x}_1) \cdots \forall y_m(\vec{x}_m). \neg R(\vec{x}, \vec{y}).$$

The above problem is an extension of the (input-unconstrained) Boolean relation determinization problem considered in Section 2.3 [27].

Related to DQBF is the so-called *independence friendly* (IF) logic. According to [26], IF logic with the game-theoretical semantics is known to violate the law of the excluded middle. A simple example is the IF logic formula $\forall x \exists y_{/x}. (x = y)$ for $x, y \in \{0, 1\}$, where $y_{/x}$ indicates the independence of y on x [19]. It assumes that not only y is independent of x , but also is x independent of y . That is, it is equivalent to $\forall x_{()} \exists y_{()}.(x = y)$ in our dependency notation. In a game-theoretic viewpoint, neither the \exists -player nor the \forall -player has a winning strategy. Therefore this formula is neither true nor false, and has no equivalent DQBF since any DQBF can always be expanded into a QBF, whose truth and falsity can be fully determined.

On the other hand, the game-theoretical semantics of IF logic, when extended to DQBFs, does not provide a fully meaningful approach to synthesizing Skolem and



Herbrand functions. Unlike the unimportance of the syntactic quantification order in our formulation, the semantic game of IF logic should be played with respect to the prefix order. Since different orders correspond to different games, the semantics is not directly useful in our considered synthesis application.

Henkin quantifiers in their original form [25] specified only the dependencies of existential variables on universal variables. Such restricted dependencies were assumed in early IF logic [26] research. As was argued in [19], the dependency of universal variables on existential variables are necessary to accomplish a symmetric treatment on the falsity, in addition to truth, of an IF logic formula. With such extension, IF logic formulae can be closed under negation. However, how the dependencies of existential variables and universal variables relate to each other was not studied. The essential notion of Herbrand functions was missing. In contrast, our formulation on DQBFs treats Skolem and Herbrand functions on an equal footing. Unlike [19], we restrict a formula to be of either S-form or H-form, rather than simultaneous specification of dependencies for existential and universal variables. This restriction makes the synthesis of Skolem and Herbrand functions for DQBFs more natural.

Prior work [15, 36] assumed DQBFs are of S-form only. In [36], a DQBF was formulated as a game played by a \forall -player and multiple noncooperative \exists -players. This game formulation is fundamentally different from that of IF-logic. The winning strategies, if they exist, of the \exists -players correspond to the Skolem functions of the DQBF. This game interpretation can be naturally extended to H-form DQBFs.

5.2 DQBF properties

5.2.1 Semantic classification of DQBF

Recall that in QBF certification there always exists either a Skolem-function model or a Herbrand-function countermodel to a given QBF. One intriguing question is whether or not the same property carries to DQBFs as well. To answer this question, we distinguish two operators, *negation* (symbolized by “ \neg ”) and *complement* (by “ \sim ”), for DQBFs. Let Φ_S and Φ_H be Formulae (5.1) and (5.2), respectively. By

5.2. DQBF properties



negation, we define

$$\neg\Phi_S = \exists x_1 \cdots \exists x_n \forall y_{1(S_1)} \cdots \forall y_{m(S_m)}. \neg\phi \text{ and} \quad (5.3)$$

$$\neg\Phi_H = \exists x_{1(H_1)} \cdots \exists x_{n(H_n)} \forall y_1 \cdots \forall y_m. \neg\phi. \quad (5.4)$$

By complement, we define

$$\sim\Phi_S = \exists x_{1(H'_1)} \cdots \exists x_{n(H'_n)} \forall y_1 \cdots \forall y_m. \neg\phi \text{ and} \quad (5.5)$$

$$\sim\Phi_H = \exists x_1 \cdots \exists x_n \forall y_{1(S'_1)} \cdots \forall y_{m(S'_m)}. \neg\phi, \quad (5.6)$$

where $H'_i = \{y_j \in Y \mid x_i \notin S_j\}$ and $S'_k = \{x_l \in X \mid y_k \notin H_l\}$, which follow what we call the *complementary principle* of the Skolem and Herbrand support sets.

By the above definitions, one verifies that $\neg\neg\Phi = \Phi$, $\sim\sim\Phi = \Phi$, and $\neg\sim\Phi = \sim\neg\Phi$. Moreover, because the Skolem functions of Φ_S , if they exist, are exactly the Herbrand functions of $\neg\Phi_S$, and the Herbrand functions of Φ_H , if they exist, are exactly the Skolem functions of $\neg\Phi_H$, the following proposition holds.

Proposition 5.1 DQBFs under the negation operation obey the *law of the excluded middle*. That is, a DQBF is true if and only if its negation is false.

Since any DQBF can be converted to a logically equivalent QBF by formula expansion, it also explains that the law of the excluded middle should hold under negation for DQBFs as it holds for QBFs.

A remaining question is whether or not the complement of DQBFs obeys the law of the excluded middle. The answer to this question is in general negative as we show below. Based on the existence of Skolem and Herbrand functions, we classify DQBFs into four categories:

$$\mathcal{C}_S = \{\Phi \mid \Phi \text{ is true and } \sim\Phi \text{ is false}\},$$

$$\mathcal{C}_H = \{\Phi \mid \Phi \text{ is false and } \sim\Phi \text{ is true}\},$$

$$\mathcal{C}_{SH} = \{\Phi \mid \Phi \text{ and } \sim\Phi \text{ are true for S-form } \Phi, \text{ or false for H-form } \Phi\}, \text{ and}$$

$$\mathcal{C}_\emptyset = \{\Phi \mid \Phi \text{ and } \sim\Phi \text{ are false for S-form } \Phi, \text{ or true for H-form } \Phi\}.$$

Note that if $\Phi \in \mathcal{C}_S$, then $\sim\Phi \in \mathcal{C}_H$; if $\Phi \in \mathcal{C}_H$, then $\sim\Phi \in \mathcal{C}_S$; if $\Phi \in \mathcal{C}_{SH}$, then $\sim\Phi \in \mathcal{C}_{SH}$; if $\Phi \in \mathcal{C}_\emptyset$, then $\sim\Phi \in \mathcal{C}_\emptyset$.

5.2. DQBF properties

Under the above DQBF partition, observe that the complement of DQBFs obeys the law of the excluded middle if and only if \mathcal{C}_{SH} and \mathcal{C}_\emptyset are empty. In fact, as to be shown, for any QBF Φ , $\Phi \notin \mathcal{C}_{SH} \cup \mathcal{C}_\emptyset$. As a consequence, the complement and negation operations for any QBF Φ coincide, and thus $\neg\sim\Phi = \Phi$. However, for general DQBFs, \mathcal{C}_{SH} and \mathcal{C}_\emptyset are not empty as the following two examples show.

Example 5.1 Consider the DQBF

$$\Phi = \forall x_1 \forall x_2 \exists y_{1(x_1)} \exists y_{2(x_2)}. ((y_1 \oplus x_1) \wedge (y_2 \oplus x_2)) \vee ((y_2 \oplus x_2) \wedge (y_1 \oplus x_1)),$$

where symbols “ \oplus ” and “ \oplus ” stand for Boolean XOR and XNOR operators, respectively. Φ has Skolem functions, e.g., x_1 and $\neg x_2$ for existential variables y_1 and y_2 , respectively, and $\neg\sim\Phi$ has Herbrand functions, e.g., y_2 and y_1 for universal variables x_1 for x_2 , respectively. That is, $\Phi \in \mathcal{C}_{SH}$.

Example 5.2 Consider the DQBF

$$\Phi = \forall x_1 \forall x_2 \exists y_{1(x_1)} \exists y_{2(x_2)}. (y_1 \vee \neg x_1 \vee x_2)(y_2 \vee x_1 \vee \neg x_2)(\neg y_1 \vee \neg y_2 \vee \neg x_1 \vee \neg x_2).$$

It can be verified that Φ has no Skolem functions, and $\neg\sim\Phi$ has no Herbrand functions. That is, $\Phi \in \mathcal{C}_\emptyset$.

By these two examples, the following proposition can be concluded.

Proposition 5.2 DQBFs under the complement operation do not obey the law of the excluded middle. That is, the truth (respectively falsity) of a DQBF cannot be decided from the falsity (respectively truth) of its complement.

Nevertheless, if a DQBF $\Phi \notin \mathcal{C}_{SH} \cup \mathcal{C}_\emptyset$, then its truth and falsity can surely be certified by a Skolem-function model and a Herbrand-function countermodel, respectively.³ That is, excluding those in $\mathcal{C}_{SH} \cup \mathcal{C}_\emptyset$, a DQBF under the complement operation obeys the law of the excluded middle.

A sufficient condition for a DQBF not in \mathcal{C}_{SH} (equivalently, a necessary condition for a DQBF in \mathcal{C}_{SH}) is presented in Theorem 5.1.

³In general a false S-form DQBF has no Herbrand-function countermodel, and a true H-form DQBF has no Skolem-function model.

5.2. DQBF properties

Theorem 5.1 Let ϕ be a quantifier-free formula over variables $X \cup Y$, let $\Phi_1 = \forall x_1 \cdots \forall x_n \exists y_1(S_1) \cdots \exists y_m(S_m) \cdot \phi$ and $\Phi_2 = \forall x_1(H_1) \cdots \forall x_n(H_n) \exists y_1 \cdots \exists y_m \cdot \phi$ with $H_i = \{y_j \in Y \mid x_i \notin S_{y_j}\}$. Then there exist Skolem functions $\vec{f} = (f_1, \dots, f_m)$ for Φ_1 and Herbrand functions $\vec{g} = (g_1, \dots, g_n)$ for Φ_2 only if the composite function vector $\vec{g} \circ \vec{f}$ admits no fixed-point, that is, there exists no truth assignment α to variables $\vec{x} = (x_1, \dots, x_n)$ such that $\alpha = \vec{g}(\vec{f}(\alpha))$.

Proof Since Φ_1 is true and has Skolem functions \vec{f} , formula $\phi[\vec{y}/\vec{f}]$ must be a tautology. On the other hand, since Φ_2 is false and has Herbrand functions \vec{g} , formula $\phi[\vec{x}/\vec{g}]$ must be unsatisfiable. Suppose that the fixed-point condition $\alpha = \vec{g}(\vec{f}(\alpha))$ holds under some truth assignment α to \vec{x} . Then $\phi[\vec{y}/\vec{f}]|_\alpha = \phi[\vec{x}/\vec{g}]|_\beta$ for $\beta = \vec{f}(\alpha)$ being the truth assignment to \vec{y} . It contradicts with the fact that $\phi[\vec{y}/\vec{f}]$ must be a tautology and $\phi[\vec{x}/\vec{g}]$ must be unsatisfiable. ■

The following corollary shows that $\Phi \notin \mathcal{C}_{SH}$ for any QBF Φ .

Corollary 5.1 For any QBF Φ , the Skolem-function model and Herbrand-function countermodel cannot co-exist.

Proof If a QBF is false, its Skolem-function model does not exist and the corollary trivially holds. Without loss of generality, assume a true QBF is of the form $\Phi = \exists \vec{y}_1 \forall \vec{x}_1 \cdots \exists \vec{y}_n \forall \vec{x}_n \cdot \phi$. Let $\{\vec{y}_1 = \vec{f}_1(), \dots, \vec{y}_n = \vec{f}_n(\vec{x}_1, \dots, \vec{x}_{n-1})\}$ be a model for Φ . Further by contradiction assume there exist a countermodel $\{\vec{x}_1 = \vec{g}_1(\vec{y}_1), \dots, \vec{x}_n = \vec{g}_n(\vec{y}_1, \dots, \vec{y}_n)\}$. So the fixed-point condition is $\{\vec{x}_1 = \vec{g}_1(\vec{f}_1()), \dots, \vec{x}_n = \vec{g}_n(\vec{f}_1(), \dots, \vec{f}_n(\vec{x}_1, \dots, \vec{x}_{n-1}))\}$. Since no cyclic dependency presents in the fixed-point equations, the set of equations always has a solution. In other words, due to the complete ordering of the prefix of a QBF, a fixed-point exists. By Theorem 5.1, the Skolem-function model and Herbrand-function countermodel cannot co-exist. ■

A sufficient condition for a DQBF not in \mathcal{C}_\emptyset can be characterized by procedure *Herbrand-Construct* as shown in Figure 5.3. Note that although the algorithm computes Herbrand functions of $\neg \sim \Phi_S$ for a false S-form DQBF Φ_S , it can be used to compute Skolem functions of $\neg \sim \Phi_H$ for a true H-form DQBF Φ_H by taking as input the negation of the formula.

Given a false S-form DQBF Φ with $n \geq 1$ universal variables, procedure *Herbrand-Construct* in line 1 collects the support set H_n for universal variable x_n . Let $H_n =$



Herbrand-Construct(Φ, n)

input: a false S-form DQBF $\Phi = \forall x_1 \dots \forall x_n \exists y_{1(S_1)} \dots \exists y_{m(S_m)} \cdot \phi$,
and the number n of universal variables

output: Herbrand-functions (g_1, \dots, g_n) of $\neg \sim \Phi$

```

01  $H_n := \{y_i \in Y \mid x_n \notin S_i\}$ 
02 if ( $n > 1$ )
03    $\Phi_{\text{exp}} := \text{Formula-Expand}(\Phi, x_n)$ ;
04    $\vec{g}^\dagger := \text{Herbrand-Construct}(\Phi_{\text{exp}}, n - 1)$ ;
05   if ( $\vec{g}^\dagger = \emptyset$ ) return  $\emptyset$ ;
06    $\vec{g} := \text{Variable-Merge}(\vec{g}^\dagger)$ ;
07   foreach assignment  $\alpha$  to  $H_n$ 
08     if ( $\phi[x_1/g_1, \dots, x_{n-1}/g_{n-1}]|_{\alpha, x_n=0}$  is unsatisfiable)
09        $g_n(\alpha) = 0$ ;
10     if ( $\phi[x_1/g_1, \dots, x_{n-1}/g_{n-1}]|_{\alpha, x_n=1}$  is unsatisfiable)
11        $g_n(\alpha) = 1$ ;
12     else return  $\emptyset$ ;
13 else
14   foreach assignment  $\alpha$  to  $H_n$ 
15     if ( $\phi|_{\alpha, x_n=0}$  is unsatisfiable)
16        $g_n(\alpha) = 0$ ;
17     if ( $\phi|_{\alpha, x_n=1}$  is unsatisfiable)
18        $g_n(\alpha) = 1$ ;
19     else return  $\emptyset$ ;
20 return  $(g_1, \dots, g_n)$ ;
end

```

Figure 5.3: Algorithm: Herbrand functions construction for S-form DQBF Φ .

$\{y_{a_1}, \dots, y_{a_k}\}$ and the rest be $\{y_{a_{k+1}}, \dots, y_{a_m}\}$. It then recursively constructs the Herbrand functions of the formula expanded on x_n until $n = 1$. By formula expansion on x_n in line 3, variables $\{y_{a_{k+1}}, \dots, y_{a_m}\}$, which depend on x_n , are instantiated in Φ_{exp} into two copies, say, $\{y'_{a_{k+1}}, y''_{a_{k+1}}, \dots, y'_{a_m}, y''_{a_m}\}$. Then the *Variable-Merge* step in line 6 lets $g_i = g_i^\dagger[y'_{a_{k+1}}/y_{a_{k+1}}, y''_{a_{k+1}}/y_{a_{k+1}}, \dots, y'_{a_m}/y_{a_m}, y''_{a_m}/y_{a_m}]$.⁴ In constructing the Herbrand function g_n of x_n , each assignment α to H_n is examined. Since Herbrand function aims to falsify ϕ , the value of $g_n(\alpha)$ is set to the x_n value that makes $\phi[x_1/g_1, \dots, x_{n-1}/g_{n-1}]|_\alpha$ unsatisfiable.

⁴The method to perform *Variable-Merge* in line 6 is not unique. In theory, as long as no violation of variable dependencies is incurred, any substitution can be applied. In practice, however the choice of substitution may affect the strength of the algorithm *Herbrand-Construct* in terms of the likelihood of returning (non-empty) Herbrand functions.

5.2. DQBF properties



Theorem 5.2 Given a false S-form DQBF Φ , algorithm *Herbrand-Construct* returns either nothing or correct Herbrand functions, which falsify $\neg \sim \Phi$.

Proof Observe first that the functions returned by the algorithm satisfy the support-set dependencies for the universal variables. It remains to show that $\phi[x_1/g_1, \dots, x_n/g_n]$ is unsatisfiable. By contradiction, suppose there exists an assignment β to the existential variables Y such that $\phi[x_1/g_1, \dots, x_n/g_n]|_\beta = 1$. Let $v \in \{0, 1\}$ be the value of $g_n|_\alpha$ for α being the projection of β on $H_n \subseteq Y$. Then $\phi[x_1/g_1, \dots, x_{n-1}/g_{n-1}, x_n/v]|_\beta = 1$. However it contradicts with the way how $g_n|_\alpha$ is constructed. Hence the returned Herbrand functions (g_1, \dots, g_n) , if they are not empty, are indeed correct Herbrand functions. ■

The following corollary shows that $\Phi \notin \mathcal{C}_\emptyset$ for any QBF Φ .

Corollary 5.2 If Φ is a false QBF and its universal variables x_1, \dots, x_n follow the QBF's prefix order, algorithm *Herbrand-Construct* always returns non-empty Herbrand functions.

Proof We prove the statement by induction on the number of universal variables. For the base case, without loss of generality consider QBF $\Phi = \exists y_1 \dots \exists y_k \forall x \exists y_{k+1} \dots \exists y_m \cdot \phi$. After line 1, *Herbrand-Construct* enters line 14. Since the QBF is false and has only one universal variable x , expanding on x yields a purely existentially quantified unsatisfiable formula: $\exists y_1 \dots \exists y_k (\exists y'_{k+1} \dots \exists y'_m \cdot \phi|_{x=0} \wedge \exists y''_{k+1} \dots \exists y''_m \cdot \phi|_{x=1})$. By its unsatisfiability, for every assignment α to y_1, \dots, y_k , formula $\exists y'_{k+1} \dots \exists y'_m \cdot \phi|_{\alpha, x=0} \wedge \exists y''_{k+1} \dots \exists y''_m \cdot \phi|_{\alpha, x=1}$ must be unsatisfiable. Since $\exists y'_{k+1} \dots \exists y'_m \cdot \phi|_{\alpha, x=0}$ and $\exists y''_{k+1} \dots \exists y''_m \cdot \phi|_{\alpha, x=1}$ share no common variables, at least one of them must be unsatisfiable. Hence the procedure returns a non-empty Herbrand function.

For the inductive step, assume the previous recursive calls for $k = 1, \dots, n-1$ of *Herbrand-Construct* do not return \emptyset . We show that the current call for $k = n$ cannot return \emptyset . Expanding Φ on x_n yields

$$\begin{aligned} \Phi_{\text{exp}} &= \forall x_1 \dots \forall x_{n-1} \exists y_{1(S_1)} \dots \exists y_{k(S_k)} \\ &\quad (\exists y'_{k+1(S_{k+1})} \dots \exists y'_{m(S_m)} \cdot \phi|_{x_n=0} \wedge \exists y''_{k+1(S_{k+1})} \dots \exists y''_{m(S_m)} \cdot \phi|_{x_n=1}). \end{aligned}$$

By the inductive hypothesis, functions $g_1^\dagger, \dots, g_{n-1}^\dagger$ are returned. Moreover, g_i^\dagger

5.2. DQBF properties

for any $i = 1, \dots, n-1$ is independent of y'_j and y''_j for $j = k+1, \dots, m$. So we construct $g_i = g_i^\dagger$. Since g_1, \dots, g_{n-1} have been constructed in a way such that

$$\begin{aligned} & \exists y_1 \cdots \exists y_k (\exists y'_{k+1} \cdots \exists y'_m \cdot \phi[x_1/g_1, \dots, x_{n-1}/g_{n-1}]|_{x_n=0} \\ & \wedge \exists y''_{k+1} \cdots \exists y''_m \cdot \phi[x_1/g_1, \dots, x_{n-1}/g_{n-1}]|_{x_n=1}) \end{aligned}$$

is unsatisfiable, under every assignment α to y_1, \dots, y_k formula

$$\begin{aligned} & \exists y'_{k+1} \cdots \exists y'_m \cdot \phi[x_1/g_1, \dots, x_{n-1}/g_{n-1}]|_{x_n=0} \\ & \wedge \exists y''_{k+1} \cdots \exists y''_m \cdot \phi[x_1/g_1, \dots, x_{n-1}/g_{n-1}]|_{x_n=1} \end{aligned}$$

is unsatisfiable. Moreover, since $\exists y'_{k+1} \cdots \exists y'_m \cdot \phi[x_1/g_1, \dots, x_{n-1}/g_{n-1}]|_{x_n=0}$ and $\exists y''_{k+1} \cdots \exists y''_m \cdot \phi[x_1/g_1, \dots, x_{n-1}/g_{n-1}]|_{x_n=1}$ do not share any variables, at least one of them must be unsatisfiable. So g_n is returned. ■

Note that the above proof does not explicitly perform the substitution $g_i = g_i^\dagger[y'_{a_{k+1}}/y_{a_{k+1}}, y''_{a_{k+1}}/y_{a_{k+1}}, \dots, y'_{a_m}/y_{a_m}, y''_{a_m}/y_{a_m}]$ in *Variable-Merge* because all g_i in fact do not depend on primed or double-primed variables in the QBF case.

Procedure *Herbrand-Construct* is useful in deriving Herbrand functions not only for QBFs but also for general DQBFs as the following example suggests.

Example 5.3 Consider the DQBF $\Phi = \forall x_1 \forall x_2 \exists y_1(x_1) \exists y_2(x_2) \cdot \phi$ with $\phi = (y_1 \vee x_2) \wedge (y_2 \vee x_1) \wedge (\neg y_1 \vee \neg y_2 \vee \neg x_1 \vee \neg x_2)$. *Herbrand-Construct*($\Phi, 2$) computes Herbrand functions for $\neg \sim \Phi$ with the following steps. Expanding Φ on x_2 yields $\Phi_{\text{exp}} = \forall x_1 \exists y_1(x_1) \exists y'_2 \exists y''_2 \cdot \phi|_{x_2=0} \wedge \phi|_{x_2=1}$ with $\phi|_{x_2=0} = (y_1) \wedge (y'_2 \vee x_1)$ and $\phi|_{x_2=1} = (y''_2 \vee x_1) \wedge (\neg y_1 \vee \neg y''_2 \vee \neg x_1)$. The recursive call to *Herbrand-Construct*($\Phi_{\text{exp}}, 1$) determines the value of function $g_1^\dagger(y'_2, y''_2)$ under every assignment α to (y'_2, y''_2) . In particular, $g_1^\dagger(0, 0) = 0$ due to $\phi_{\text{exp}} = (y_1) \wedge (x_1) \wedge (x_1)$; $g_1^\dagger(0, 1) = 0$ (or 1) due to $\phi_{\text{exp}} = (y_1) \wedge (x_1) \wedge (\neg y_1 \vee \neg x_1)$; $g_1^\dagger(1, 0) = 0$ due to $\phi_{\text{exp}} = (y_1) \wedge (x_1)$; $g_1^\dagger(1, 1) = 1$ due to $\phi_{\text{exp}} = (y_1) \wedge (\neg y_1 \vee \neg x_1)$. So $g_1^\dagger(y'_2, y''_2) = y'_2 y''_2$ (or y''_2), and $g_1(y_2) = g_1^\dagger[y'_2/y_2, y''_2/y_2] = y_2$.

Returning to *Herbrand-Construct*($\Phi, 2$), we have $\phi[x_1/g_1] = (y_1 \vee x_2) \wedge (y_2) \wedge (\neg y_1 \vee \neg y_2 \vee \neg x_2)$. The value of function g_2 for each assignment α to y_1 can be determined with $g_2(0) = 0$ due to $\phi[x_1/g_1]|_{y_1=0} = (x_2) \wedge (y_2)$ and $g_2(1) = 1$ due to $\phi[x_1/g_1]|_{y_1=1} = (y_2) \wedge (\neg y_2 \vee \neg x_2)$. That is, $g_2(y_1) = y_1$. The computed g_1 and g_2

5.2. DQBF properties

indeed make $\phi[x_1/g_1, x_2/g_2] = (y_1) \wedge (y_2) \wedge (\neg y_1 \vee \neg y_2)$ unsatisfiable.

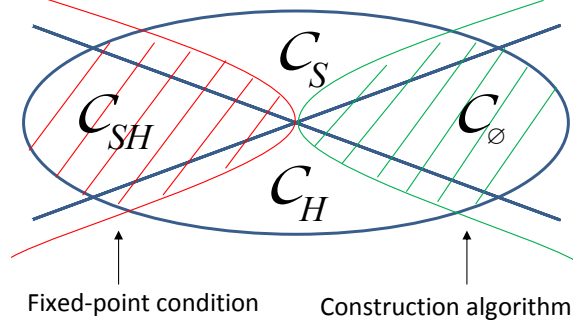


Figure 5.4: Four DQBF categories and regions characterized by Theorems 5.1 and 5.2.

Since the DQBF subset $\mathcal{C}_S \cup \mathcal{C}_H$ obeys the law of the excluded middle under the complement operation, Theorems 5.1 and 5.2 provide a tool to test whether a DQBF Φ can be equivalently expressed as $\neg \sim \Phi$, that is, whether a DQBF has either a Skolem-function model or a Herbrand-function countermodel. Figure 5.4 shows the four DQBF categories and the regions characterized by Theorems 5.1 and 5.2.

5.2.2 Formula Expansion on Existential Variables

Reasonably DQBF formula can be expanded on existential variables by firstly negating it using De Morgan's law and then expanding on universal variables. It leads to the following expansion rule, which is dual to expanding universal variables.

Proposition 5.3 Given a DQBF $\forall x_{1(H_1)} \cdots \forall x_{n(H_n)} \exists y_1 \cdots \exists y_m \cdot \phi$, assume without loss of generality that y_1 is to be expanded with $y_1 \notin H_1 \cup \cdots \cup H_{k-1}$ and $y_1 \in H_k \cap \cdots \cap H_n$. The formula can be expanded to

$$\begin{aligned} & \forall x_{1(H_1)} \cdots \forall x_{k-1(H_{k-1})} \forall x_{k(H_k[y_1/0])} \forall x_{k(H_k[y_1/1])} \cdots \forall x_{n(H_n[y_1/0])} \forall x_{n(H_n[y_1/1])} \\ & \exists y_2 \cdots \exists y_m \cdot \phi|_{y_1=0} \vee \phi|_{y_1=1}, \end{aligned}$$

where $H_i[y_1/v]$ denotes y_1 in H_i is substituted with logic value $v \in \{0, 1\}$, and $\phi|_{y_1=v}$ denotes all appearances of y_1 in ϕ are substituted with v including those in the support sets of variables $x_{i(H_i)}$ for $i = k, \dots, n$.

5.2. DQBF properties

Such expansion can be repeatedly applied for every existential variables. The resultant formula after expanding all existential variables is a QBF. Note that, when Skolem functions are concerned rather than Herbrand functions, the support sets of the existential variables should be listed and can be obtained from H_i by the aforementioned complementary principle.

Example 5.4 Consider expanding variable y_1 of DQBF

$$\Phi = \forall x_{1(y_1)} \forall x_{2(y_2)} \forall x_{3(y_3)} \exists y_1 \exists y_2 \exists y_3. \phi.$$

By De Morgan's law and expansion on a universal variable, we obtain

$$\begin{aligned} \neg \neg \Phi &= \neg \exists x_{1(y_1)} \exists x_{2(y_2)} \exists x_{3(y_3)} \forall y_1 \forall y_2 \forall y_3. \neg \phi \\ &= \neg \exists x_{1(0)} \exists x_{1(1)} \exists x_{2(y_2)} \exists x_{3(y_3)} \forall y_2 \forall y_3. \neg \phi|_{y_1=0} \wedge \neg \phi|_{y_1=1} \\ &= \forall x_{1(0)} \forall x_{1(1)} \forall x_{2(y_2)} \forall x_{3(y_3)} \exists y_2 \exists y_3. \phi|_{y_1=0} \vee \phi|_{y_1=1}. \end{aligned}$$

5.2.3 Prenex and Non-prenex Conversion

Young DQBF field has a lot of things to borrow from QBF. Below we generalize syntactic rules used in QBF for localization of quantifiers to sub-formulae, to DQBF. We focus on the truth (namely the Skolem-function model), while similar results can be concluded by duality for the falsity (namely the Herbrand-function countermodel), of a formula.

The following proposition shows the localization of existential quantifiers to the sub-formulas of a disjunction.

Proposition 5.4 The DQBF

$$\forall \vec{x} \exists y_{1(S_1)} \cdots \exists y_{m(S_m)}. \phi_A \vee \phi_B,$$

where $\forall \vec{x}$ denotes $\forall x_1 \cdots \forall x_n$, sub-formula ϕ_A (respectively ϕ_B) refers to variables $X_A \subseteq X$ and $Y_A \subseteq Y$ (respectively $X_B \subseteq X$ and $Y_B \subseteq Y$), is logically equivalent to

$$\forall \vec{x}_c \left(\forall \vec{x}_a \exists y_{a_1(S_{a_1} \cap X_A)} \cdots \exists y_{a_p(S_{a_p} \cap X_A)} \phi_A \vee \forall \vec{x}_b \exists y_{b_1(S_{b_1} \cap X_B)} \cdots \exists y_{b_q(S_{b_q} \cap X_B)} \phi_B \right),$$

where variables \vec{x}_c are in $X_A \cap X_B$, variables \vec{x}_a are in $X_A \setminus X_B$, variables \vec{x}_b are in

5.2. DQBF properties



$X_B \setminus X_A$, $y_{a_i} \in Y_A$, and $y_{b_j} \in Y_B$.

Proof A model to the former expression consists of every truth assignment to X and the induced Skolem function valuation to Y . Since every such combined assignment to $X \cup Y$ either satisfies ϕ_A or ϕ_B , by collecting those satisfying ϕ_A (respectively ϕ_B) and projecting to variables $X_A \cup Y_A$ (respectively $X_B \cup Y_B$) the model (i.e., the Skolem functions for \vec{y}_a and \vec{y}_b) to the latter expression can be constructed. (Note that, for a quantifier $\exists y_i$ splitting into two, one for ϕ_A and the other for ϕ_B , in the latter expression, they are considered distinct and have their own Skolem functions.)

In addition, the Skolem functions for $\forall \vec{x}_a \exists y_{a_1(S_{a_1} \cap X_A)} \cdots \exists y_{a_p(S_{a_p} \cap X_A)} \phi_A|_\alpha$ and those for $\forall \vec{x}_b \exists y_{b_1(S_{b_1} \cap X_B)} \cdots \exists y_{b_q(S_{b_q} \cap X_B)} \phi_B|_\alpha$ under every assignment α to \vec{x}_c can be collected and combined to form a model for the former expression. In particular the respective Skolem functions $f_{a_j}|_\alpha$ and $f_{b_k}|_\alpha$ under α for y_{a_j} and y_{b_k} originating from the same quantifier y_i in the former expression are merged into one Skolem function $f_i = \bigvee_\alpha (\chi_\alpha(f_{a_j}|_\alpha \vee f_{b_k}|_\alpha))$, where χ_α denotes the characteristic function of α , e.g., $\chi_\alpha = x_1 x_2 \neg x_3$ for $\alpha = (x_1 = 1, x_2 = 1, x_3 = 0)$. ■

Example 5.5 Consider the QBF

$$\Phi = \forall x_1 \exists y_1 \forall x_2 \exists y_2 \forall x_3 \exists y_3. \phi_A \vee \phi_B$$

with ϕ_A refers to variables x_1, x_2, y_1, y_2 and ϕ_B refers to x_2, x_3, y_2, y_3 . It has the following equivalent DQBF expressions.

$$\begin{aligned} \Phi &= \forall x_1 \forall x_2 \forall x_3 \exists y_{1(x_1)} \exists y_{2(x_1, x_2)} \exists y_{3(x_1, x_2, x_3)}. \phi_A \vee \phi_B \\ &= \forall x_1 \forall x_2 \forall x_3 (\exists y_{1(x_1)} \exists y_{2(x_1, x_2)} \phi_A \vee \exists y_{2(x_2)} \exists y_{3(x_2, x_3)} \phi_B) \\ &= \forall x_2 (\forall x_1 \exists y_{1(x_1)} \exists y_{2(x_1, x_2)} \phi_A \vee \forall x_3 \exists y_{2(x_2)} \exists y_{3(x_2, x_3)} \phi_B) \end{aligned}$$

In contrast, conventionally the quantifiers of the QBF can only be localized to

$$\forall x_1 \exists y_1 \forall x_2 \exists y_2 (\phi_A \vee \forall x_3 \exists y_3 \phi_B).$$

The following proposition shows the localization of universal quantifiers to a sub-formula of a conjunction.



Proposition 5.5 The DQBF

$$\forall \vec{x} \exists y_{1(S_1)} \cdots \exists y_{k(S_k)} \cdot \phi_A \wedge \phi_B,$$

where $\forall \vec{x}$ denotes $\forall x_1 \cdots \forall x_n$, sub-formula ϕ_A (respectively ϕ_B) refers to variables $X_A \subseteq X$ and $Y_A \subseteq Y$ (respectively $X_B \subseteq X$ and $Y_B \subseteq Y$), is logically equivalent to

$$\forall \vec{x} \exists y_{2(S_2)} \cdots \exists y_{k(S_k)} \cdot (\exists y_{1(S_1 \cap X_A)} \phi_A) \wedge \phi_B,$$

for $y_1 \notin Y_B$.

Proof The proposition follows from the fact that the Skolem function of y_1 is purely constrained by ϕ_A only, and is the same for both expressions. Note that the former formula is equivalent to $\forall \vec{x} \exists y_{1(S_1 \cap X_A)} \cdots \exists y_{k(S_k)} \cdot \phi_A \wedge \phi_B$. ■

Essentially DQBFs allow tighter localization of quantifier scopes than QBFs. On the other hand, converting a non-prenex QBF to the prenex form may incur the size increase of support sets of existential variables due to the linear (or complete order) structure of the prefix. With DQBFs, such spurious increase can be eliminated.

5.3 DQ-resolution

Methods of Q-resolution for QBFs can be naturally extended to DQBFs as follows. For an S -form DQBF $\Phi = \forall x_1 \cdots \forall x_n \exists y_{1(S_1)} \cdots \exists y_{m(S_m)} \cdot \phi$ in PCNF, a clause $C \in \phi$ is called *minimal* if, for every literal $l \in C$ with $\text{var}(l) = x_i \in X$, there exists some $l' \in C$ with $\text{var}(l') = y_j \in Y$ such that $x_i \in S_j$. Otherwise, C is *non-minimal*. A non-minimal clause C can be reduced to a minimal clause C^\dagger by removing from C its universal literals

$$\{l \in C \mid \text{var}(l) = x_i \in X \text{ and } x_i \notin S_j \text{ for all } \text{var}(l') = y_j \in Y \text{ with } l' \in C\}$$

This reduction process is called the \forall_D -reduction. Similarly, by duality in an H -form DQBF $\Phi = \forall x_{1(H_1)} \cdots \forall x_{n(H_n)} \exists y_1 \cdots \exists y_m \cdot \phi$ in PDNF, a cube $C \in \phi$ is called *minimal* if, for every literal $l \in C$ with $\text{var}(l) = y_i \in Y$, there exists some $l' \in C$ with $\text{var}(l') = x_j \in X$ such that $y_i \in H_j$. Otherwise, C is *non-minimal*. A non-minimal cube C can be reduced to a minimal cube C^\dagger by removing from C its existential

5.3. DQ-resolution



literals

$$\{l \in C \mid \text{var}(l) = y_i \in Y \text{ and } y_i \notin H_j \text{ for all } \text{var}(l') = x_j \in X \text{ with } l' \in C\}$$

This reduction process is called the \exists_D -reduction.

DQ-resolution and *DQ-consensus* of DQBFs are the same as Q-resolution and Q-consensus of QBFs, respectively, except that the \forall -reduction and \exists -reduction are replaced by \forall_D -reduction (for *S*-form DQBFs) and \exists_D -reduction (for *H*-form DQBFs). The following theorem states the soundness of DQ-resolution.

Theorem 5.3 Given an *S*-form DQBF Φ in PCNF, Φ is false if there exists a DQ-resolution sequence leading to an empty clause.

Proof Let ϕ be the matrix of Φ in CNF. Assume $\phi = \phi' \wedge C$ for some non-minimal clause $C = (l_1 \vee \dots \vee l_n \vee l^*)$ with $\text{var}(l^*)$ being a universal variable not in S_j for all existential variables $y_j = \text{var}(l_i)$, $i = 1, \dots, n$. Let Ψ be a DQBF same as Φ except for the clause C being replaced by $(l_1 \vee \dots \vee l_n)$. The logic equivalence between Φ and Ψ can be easily established by expanding Φ and Ψ on the universal variable $\text{var}(l^*)$. By the aforementioned expansion rule, it is easily seen that Φ and Ψ after expansion converge to the same formula. Consequently \forall_D -reduction is sound. That is, reducing a non-minimal clause of a DQBF to its minimal form preserves logic equivalence. On the other hand, resolution is sound regardless of the quantification prefix. Since both resolution and \forall_D -reduction of DQ-resolution are sound, the derived resolvents and their reduced clauses are logically implied by the original formula. Therefore, as long as an empty clause can be obtained through DQ-resolution, the DQBF must be false. ■

Example 5.6 Consider the DQBF

$$\forall x_1 \forall x_2 \exists y_{1(x_1)} \exists y_{2(x_2)} \cdot (y_1 \vee \neg x_1 \vee x_2)(y_2 \vee x_1 \vee \neg x_2)(\neg y_1 \vee \neg y_2 \vee \neg x_1 \vee \neg x_2),$$

whose falsity is established by the DQ-resolution proof shown in Figure 5.5.

Similar to Theorem 5.3, one can establish by duality the soundness of DQ-consensus.

Theorem 5.4 Given an *H*-form DQBF Φ in PDNF, Φ is true if there exists a DQ-consensus sequence leading to an empty cube.

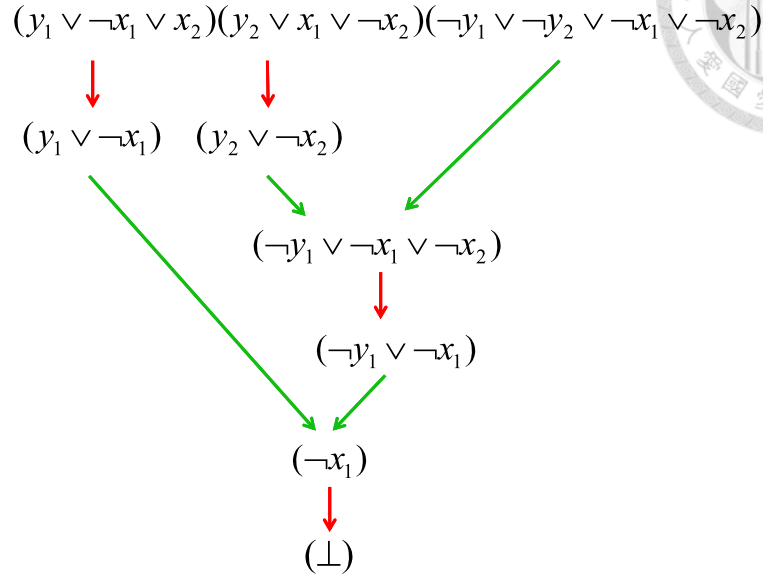


Figure 5.5: DQ-resolution proof of a false DQBF

Proof The proof is similar to that of Theorem 5.3, and is omitted. ■

Note that the proofs of Theorems 5.3 and 5.4 do not carry to H-form DQBFs in PCNF and S-form DQBFs in PDNF, respectively, because expansions on universal variables of an H-form DQBF and on existential variables of an S-form DQBF are illy defined. Moreover, although DQ-resolution can be similarly defined for H-form DQBFs in PCNF by the modified reduction rule as removing from a clause C its universal literals

$$\{l \in C \mid \text{var}(l) = x_i \in X \text{ and } y_j \in H_i \text{ for all } \text{var}(l') = y_j \in Y \text{ with } l' \in C\}$$

and DQ-consensus can be defined for S-form DQBFs in PDNF by removing from a cube C its existential literals

$$\{l \in C \mid \text{var}(l) = y_i \in Y \text{ and } x_j \in S_i \text{ for all } \text{var}(l') = x_j \in X \text{ with } l' \in C\},$$

these definitions are unsound, however, as the following example suggests.

Example 5.7 Consider the S-form DQBF

$$\Phi = \forall x_1 \forall x_2 \exists y_{1(x_1)} \exists y_{2(x_2)}. (\neg x_2 y_1 \neg y_2) \vee (\neg x_1 \neg y_1 y_2) \vee (x_1 x_2 y_1 y_2).$$

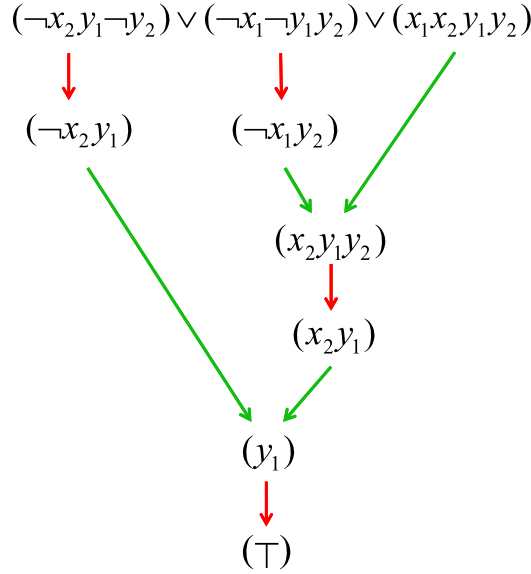


Figure 5.6: DQ-consensus proof for $\sim\neg\Phi$

As can be verified, it is false due to the absence of Skolem function models. However, the modified \exists_D -reduction may lead to a DQ-consensus proof of an empty cube as shown in Figure 5.6, which in turn asserts that the H-form $\sim\neg\Phi$ is true. Therefore the modified \exists_D -reduction is unsound for S -form DQBFs. (Similarly the modified \forall_D -reduction is unsound for H -form DQBFs.)

Although DQ-resolution and DQ-consensus are sound as shown in Theorems 5.3 and 5.4, they are unfortunately incomplete in proving the truth and falsity of DQBFs as we show below.

Theorem 5.5 DQ-resolution is incomplete in proving the truth and falsity of S -form DQBFs.

Proof The theorem can be established by the following DQBF.

$$\Phi = \forall x_1 \forall x_2 \exists y_{1(x_1)} \exists y_{2(x_2)} \cdot (y_1 \vee y_2 \vee x_1)(\neg y_1 \vee \neg y_2 \vee x_1)(y_1 \vee y_2 \vee \neg x_1 \vee \neg x_2) \\ (\neg y_1 \vee y_2 \vee \neg x_1 \vee x_2)(y_1 \vee \neg y_2 \vee \neg x_1 \vee x_2)(\neg y_1 \vee \neg y_2 \vee \neg x_1 \vee \neg x_2).$$

It can be verified that Φ is false (i.e., no Skolem function models), and yet no DQ-resolution steps can be made. ■



Similarly one can establish the following claim.

Theorem 5.6 DQ-consensus is incomplete in proving the truth and falsity of H-form DQBFs.

Since the truth of an S -form (respectively H -form) DQBF is defined by the existence of Skolem (respectively Herbrand) functions, the definition is not consistent with the existence of DQ-consensus (respectively DQ-resolution) proofs. As we have seen from Chapter 3, for the QBF case, a Q-resolution (respectively Q-consensus) proof can be both an evidence for the absence of Skolem (respectively Herbrand) functions and an evidence for the existence of Herbrand (respectively Skolem) functions. For the DQBF case, unfortunately there is no such nice property. For an S -form (respectively H -form) DQBF Φ , a DQ-resolution (respectively DQ-consensus) proof can only certify the absence of Skolem (respectively Herbrand) functions, thus soundly proving the formula is false (respectively true), but cannot guarantee the existence of Herbrand (respectively Skolem) functions for the H -form (respectively S -form) DQBF $\sim\neg\Phi$.

Although DQ-resolution and DQ-consensus are incomplete, they are useful, due to their soundness, in DQBF evaluation since resolvents and consensus terms can be used as learnt clauses for S -form DQBFs and learnt cubes for H -form DQBFs, respectively. Moreover, if for some true S -form (respectively false H -form) DQBF Φ we can prove $\Phi = \sim\neg\Phi$ (namely if $\Phi \in C_S$ or $\Phi \in C_H$), then we can soundly use a DQ-consensus (respectively DQ-resolution) proof for the H -form (respectively S -form) DQBF $\sim\neg\Phi$ as the evidence of the existence of Skolem (respectively Herbrand) functions for Φ .

In light of algorithm *Countermodel-Construct* for QBFs, presented in Section 4.2, one might hope that similar algorithms exist for DQBFs in converting DQ-resolution proofs to Herbrand functions and converting DQ-consensus proofs to Skolem functions. However it is, in general, impossible due to the non-emptiness of C_{SH} and C_\emptyset . Nevertheless, the possibility that such algorithms exist for DQBFs in $C_S \cup C_H$ is not ruled out.



Chapter 6

Conclusions and future work

Extensive study of various problems related to certification issues in quantified decision procedures was presented in this work. Contributions listed in Section 1.2 have been fully uncovered through the pages of this work. We provided an extensive study of various certificate types for quantified Boolean formulae and revealed symmetrically complete QBF certification picture.

We analyzed existing syntactic QBF certificates in form of resolution proofs, established missing links between their complexities, identified their weaknesses and proposed improved extended proof systems that, as we hope, could also find practical use in QBF solving. We also extended the ideas from QBF language to DQBF, introduced DQ-resolution and discussed some of its important properties.

Further, we characterized the connection between Q-resolution proofs and Skolem and Herbrand functions, designed an algorithm for Skolem and Herbrand functions extraction from Q-refutations (and LQU-refutations as well), and explored various flexibilities present during the extraction process for certificate optimization. From practical side, we implemented the proposed algorithm into an efficient stand-alone tool RESQU (and extensions RESQU-SORT and RESQU-LP) for (optimized) (counter)model extraction from Q-resolution proofs (also LQU+-resolution proofs), and extensively tested it on the various applications and testbenches.

Through the experiments conducted in this work we have found that our approach in general can produce more models and countermodels than prior methods. Furthermore no solvers could produce countermodels directly without formula negation

before. Experiments also showed that using our approach we can determinize more instances from relation determinization problems than just by using solvers which directly produce Skolem functions. Experimental evaluation of optimization techniques proposed in Section 4.3 showed up to two orders of magnitude improvement in the extracted (counter)models quality.

We conclude that theoretical and practical novelties introduced in this work form an important contribution to the understanding and use of quantified decision procedures for modern applications. In future we plan to continue our research of syntactic and semantic certificates for QBF and DQBF languages to make them more competitive against other approaches, and enable QBF-based approaches for more formal verification and synthesis applications.



List of Figures

1.1	Transistor count and Moore's law.	3
2.1	Truth tables representing key Boolean operations.	7
2.2	Various representations of a Boolean function.	8
2.3	Truth table of QBF Φ	13
2.4	Tseitin encoding of basic logic gates.	14
2.5	Core DPLL algorithm for SAT solving.	16
2.6	DPLL search tree for ϕ of Example 2.2.	16
2.7	Truth table of QBF Φ	21
2.8	Core QDPLL algorithm for QBF solving.	28
2.9	A complete search tree obtained by applying QDPLL procedure to Φ	29
3.1	Q-resolution proof construction from the search tree.	35
3.2	DAG for Q-resolution proof of Φ , constructed by algorithm QDPLL.	36
3.3	Q-resolution proof for QBF KBKF[t] with $t = 2$	37
3.4	QU-resolution proof for QBF KBKF[t] with $t = 2$	42
3.5	LQ-resolution proof for QBF KBKF[t] with $t = 2$	43

3.6	DAG of resolution proof Π^*	44
3.7	Conversion of LQ-resolution proof into Q-resolution.	47
3.8	The diagram for Resolution Proof systems relationship.	57
4.1	Algorithm: Countermodel Construction from Q-refutations.	66
4.2	DAG of resolution proof Π	67
4.3	Contents of RFAO node arrays.	68
4.4	Cube Q-resolution proof for QBF Φ in PDNF form from Example 4.2.	71
4.5	Algorithm: Model Construction from Q-consensus proofs.	72
4.6	DAG of resolution proof Π'	73
4.7	DAG of the Q-resolution proof Π for the QBF Φ in Example 4.4	79
4.8	Algorithm: Greedy Alternation Minimization.	83
4.9	Algorithm: Postponing \forall -reduction.	85
4.10	Certificate size and depth statistics for true instances.	92
4.11	Certificate size and depth statistics for false instances.	93
4.12	Algorithm extracting a countermodel from an LQ-resolution proof.	97
4.13	DAG of LQ-resolution proof Π	98
4.14	Phase functions f and g for literals of variables x and y	99
4.15	Comparison on certificate quality for application benchmarks.	108
5.1	A network of known and unknown logic components	113
5.2	Function derivation from a Boolean relation	114
5.3	Algorithm: Herbrand functions construction for S-form DQBF Φ	119

List of Figures

5.4	Four DQBF categories and regions characterized by Theorems 5.1 and 5.2.	122
5.5	DQ-resolution proof of a false DQBF	127
5.6	DQ-consensus proof for $\sim\neg\Phi$	128



List of Tables

3.1	Summary of Proof System Rules.	55
4.1	Summary for QBFEVAL Benchmarks.	74
4.2	Summary for Relation Determinization Benchmarks.	76
4.3	Results for Relation Determinization Benchmarks.	78
4.4	Certificate comparison summary.	93
4.5	Attributes of each vertex $v \in V_\Pi$ of an LQ-resolution proof Π , represented as a DAG $G_\Pi(V_\Pi, E_\Pi)$, of a false QBF Φ	96
4.6	Time statistics (in seconds) for KBKF instances.	106
4.7	Certificate sizes for KBKF instances.	107




Bibliography

- [1] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.
- [2] Valeriy Balabanov, Hui-Ju Katherine Chiang, and Jie-Hong R. Jiang. Henkin quantifiers and boolean formulae: A certification perspective of DQBF. *Theor. Comput. Sci.*, 523:86–100, 2014.
- [3] Valeriy Balabanov, Hui-Ju Katherine Chiang, and Jie-Hong Roland Jiang. Henkin quantifiers and boolean formulae. In *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, pages 129–142, 2012.
- [4] Valeriy Balabanov and Jie-Hong R. Jiang. Resolution proofs and skolem functions in QBF evaluation and applications. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, pages 149–164, 2011.
- [5] Valeriy Balabanov and Jie-Hong R. Jiang. Unified QBF Certification and Its Applications. *Formal Methods in System Design*, 41:45–65, 2012.
- [6] Valeriy Balabanov, Jie-Hong R. Jiang, Mikoláš Janota, and Magdalena Widl. Efficient extraction of QBF (counter)models from long-distance resolution proofs. In *To appear in proceedings of International Conference on Advancements in Artificial Intelligence - AAAI 2015*, 2015.
- [7] Valeriy Balabanov, Shuo-Ren Lin, and Jie-Hong R. Jiang. Flexibility and optimization of QBF skolem-herbrand certificates. In *Work in Progress*, 2015.
- [8] Valeriy Balabanov, Magdalena Widl, and Jie-Hong R. Jiang. QBF resolution systems and their proof complexities. In *Theory and Applications of Satisfiability*

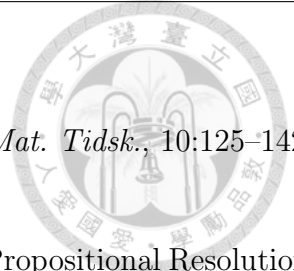


- Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, pages 154–169, 2014.
- [9] Marco Benedetti. Evaluating qbfs via symbolic skolemization. In *Logic for Programming, Artificial Intelligence, and Reasoning, 11th International Conference, LPAR 2004, Montevideo, Uruguay, March 14-18, 2005, Proceedings*, pages 285–300, 2004.
- [10] Marco Benedetti. sKizzo: A suite to evaluate and certify QBFs. In *International Conference on Automated Deduction (CADE)*, pages 369–376. Springer, 2005.
- [11] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification. [http://http://www.eecs.berkeley.edu/~alanmi/abc/](http://www.eecs.berkeley.edu/~alanmi/abc/).
- [12] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.
- [13] A. Blass and Y. Gurevich. Henkin quantifiers and complete problems. *Annals of Pure and Applied Logic*, 32:1–16, 1986.
- [14] Roderick Bloem, Stefan J. Galler, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Martin Weiglhofer. Interactive presentation: Automatic hardware synthesis from specifications: a case study. In *2007 Design, Automation and Test in Europe Conference and Exposition (DATE 2007), April 16-20, 2007, Nice, France*, pages 1188–1193, 2007.
- [15] Uwe Bubeck. *Model-based transformations for quantified boolean formulas*. PhD thesis, University of Paderborn, 2010.
- [16] Uwe Bubeck and Hans Kleine Büning. Dependency quantified horn formulas: Models and complexity. In *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, pages 198–211, 2006.
- [17] Marco Cadoli, Marco Schaerf, Andrea Giovanardi, and Massimo Giovanardi. An algorithm to evaluate quantified boolean formulae and its experimental evaluation. *J. Autom. Reasoning*, 28(2):101–142, 2002.

- 
- [18] Xavier Caicedo, Francien Dechesne, and Theo M. V. Janssen. Equivalence and quantifier rules for logic with imperfect information. *Logic Journal of the IGPL*, 17(1):91–129, 2009.
- [19] F. Dechesne. *Game, Set, Maths: Formal Investigations into Logic with Imperfect Information*. PhD thesis, Tilburg University, 2005.
- [20] Niklas Eén and Niklas Sörensson. An Extensible SAT-Solver. In *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 2919 of *LNCS*, pages 502–518. Springer, 2003.
- [21] Uwe Egly, Florian Lonsing, and Magdalena Widl. Long-distance resolution: Proof generation and strategy extraction in search-based QBF solving. In *International Conference on Logic Programming and Automated Reasoning (LPAR)*, pages 291–308. Springer, 2013.
- [22] Andreas Fröhlich, Gergely Kovásznai, and Armin Biere. A dpll algorithm for solving dqbf. In *Pragmatics of SAT*, 2012.
- [23] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Clause/term resolution and learning in the evaluation of quantified boolean formulas. *J. Artif. Intell. Res. (JAIR)*, 26:371–416, 2006.
- [24] Alexandra Goultiaeva, Allen Van Gelder, and Fahiem Bacchus. A Uniform Approach for Generating Proofs and Strategies for Both True and False QBF Formulas. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 546–553. AAAI Press, 2011.
- [25] Leon Henkin. Some remarks on infinitely long formulas. *Infinitistic Methods*, pages 167–183, 1961.
- [26] J. Hintikka and G. Sandu. Informational independence as a semantical phenomenon. In *Logic, Methodology and Philosophy of Science*, pages 571–589, 1989.
- [27] Jie-Hong R. Jiang, Hsuan-Po Lin, and Wei-Lun Hung. Interpolating Functions from Large Boolean Relations. In *Proc. International Conference on Computer-Aided Design (ICCAD)*, pages 779–784. IEEE/ACM, 2009.
- [28] Toni Jussila, Armin Biere, Carsten Sinz, Daniel Kröning, and Christoph M. Wintersteiger. A first step towards a unified proof checker for QBF. In *Theory and*

- Applications of Satisfiability Testing - SAT 2007, 10th International Conference, Lisbon, Portugal, May 28-31, 2007, Proceedings*, pages 201–214, 2007.
- [29] Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for Quantified Boolean Formulas. *Information and Computation*, 117(1):12–18, Feb. 1995.
- [30] Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified boolean formulas. *Inf. Comput.*, 117(1):12–18, 1995.
- [31] Florian Lonsing and Armin Biere. DepQBF: A Dependency-Aware QBF Solver (System Description). *Journal on Satisfiability, Boolean Modeling and Computation*, 7:71–76, 2010.
- [32] Alan Mishchenko, Satrajit Chatterjee, and Robert K. Brayton. Dag-aware AIG rewriting a fresh look at combinational logic synthesis. In *Proceedings of the 43rd Design Automation Conference, DAC 2006, San Francisco, CA, USA, July 24-28, 2006*, pages 532–535, 2006.
- [33] Massimo Narizzano, Claudia Peschiera, Luca Pulina, and Armando Tacchella. Evaluating and certifying qbfs: A comparison of state-of-the-art tools. *AI Commun.*, 22(4):191–210, 2009.
- [34] Aina Niemetz, Mathias Preiner, Florian Lonsing, Martina Seidl, and Armin Biere. Resolution-based certificate extraction for QBF - (tool presentation). In *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, pages 430–435, 2012.
- [35] S. Peters and D. Westerstahl. *Quantifiers in Language and Logic*. Oxford University Press, 2006.
- [36] G. Peterson, J. Reif, and S. Azhar. Lower bounds for multiplayer noncooperative games of incomplete information. In *Computers and Mathematics with Applications*, volume 41, pages 957–992, 2001.
- [37] QBFLIB. Online quantified boolean formulas library, and evaluation portal. <http://www.QBFlib.com>.
- [38] Subarnarekha Sinha, Alan Mishchenko, and Robert K. Brayton. Topologically constrained logic synthesis. In *Proceedings of the 2002 IEEE/ACM International Conference on Computer-aided Design, 2002, San Jose, California, USA, November 10-14, 2002*, pages 679–686, 2002.

Bibliography

- 
- [39] Thoralf Skolem. Über die mathematische logik. *Norsk. Mat. Tidsk.*, 10:125–142, 1928.
- [40] Allen Van Gelder. Input Distance and Lower Bounds for Propositional Resolution Proof Length. In *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 3569 of *LNCS*, pages 282–293. Springer, 2005.
- [41] Yinlei Yu and Sharad Malik. Validating the result of a quantified boolean formula (QBF) solver: theory and practice. In *Proceedings of the 2005 Conference on Asia South Pacific Design Automation, ASP-DAC 2005, Shanghai, China, January 18-21, 2005*, pages 1047–1051, 2005.
- [42] Lintao Zhang and Sharad Malik. Conflict Driven Learning in a Quantified Boolean Satisfiability Solver. In *Proc. International Conference on Computer-Aided Design (ICCAD)*, pages 442–449. ACM, 2002.