RICE UNIVERSITY

Complexity and Structural Heuristics for Propositional and Quantified Satisfiability

by

Guoqiang Pan

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

Doctor of Philosophy

Approved, Thesis Committee:

Dr. Moshe Y. Vardi (Chair),
Karen Ostrum George Professor,
Computer Science

Dr. Devika S. Subramanian,
Professor,
Computer Science

Dr. Walid Taha,
Assistant Professor,
Computer Science

Dr. Kartik Mohanram,
Assistant Professor,
Electric and Computer Engineering

Houston, Texas

August, 2006

Complexity and Structural Heuristics for Propositional and Quantified Satisfiability

Guoqiang Pan

Abstract

Decision procedures for various logics are used as general-purpose solvers in computer science. A particularly popular choice is propositional logic, which is simultaneously powerful enough to model problems in many application domains, including formal verification and planning, while at the same time simple enough to be efficiently solved for many practical cases. Similarly, there are also recent interests in using QBF, an extension of propositional logic, as a modeling language to be used in a similar fashion. The hope is that QBF, being a more powerful language, can compactly encode, and in turn, be used to solve, a larger range of applications. Still, propositional logic and QBF are respectively complete for the complexity classes NP and PSPACE, thus, both can be theoretically considered intractable. A popular hypothesis is that real-world problems contain underlying structure that can be exploited by the decision procedures. In this dissertation, we study the impact of structural constraints (in the form of bounded width) and heuristics on the performance of propositional and QBF decision procedures.

The results presented in this dissertation can be seen as a contrast on how bounded-width impacts propositional and quantified problems differently. Starting with a size bound on BDDs under bounded width, we proceed to compare symbolic decision procedures against the standard DPLL search-based approach for propositional logic, as well as compare different width-based heuristics for the symbolic approaches. In general, symbolic approaches for propositional satisfiability are only competitive for a small range of problems, and the theoretical tractability for the bounded-width case rarely applies in practice. However, the picture is very different for quantified satisfiability. To that end, we start with a series of “intractability in tractability” results which shows that although the complexity of QBF with constant width and alternation is tractable, there is an inherent non-elementary blowup in the width and alternation depth such that a width-bound that is slightly above constant leads to intractability. To contrast the theoretical intractability, we apply structural heuristics to a symbolic decision procedure of QBF and show that symbolic approaches complement search-based approaches quite well for QBF.

Acknowledgement

First of all, I would like to thank my advisor Prof. Moshe Y. Vardi for the support and guidance provided through out my six years at Rice. You have advised me on not just the technical developments and approaches, but also the methodology, presentation, and insight needed to do research. And of course, the financial support too.

Also, I would like to thank the other members of my defense committee, Professors Devika Subramanian, Walid Taha, and Kartik Mohanram for their help. I have taken classes under all of you and I learned much, especially on looking at the area of computer science at a wider angle than what I would originally do.

I would also thank the people at NEC research labs, Pranav Ashar, Aarti Gupta, Malay Ganai, Chao Wang, Franjo Ivancic, Zijiang Yang, and others. I learned a lot in my two internships done at NEC research labs, and the experience helped me broaden my views to

Next, I would also like to thank my fellow students and visitors to our group, including Andrew Ladd, Brian Chen, Ben McMahan, Algys Rudys, Deian Tabakov, Armando Tacchella, Kristin Rozier, Kostas Bekris, Davis Schwarz, Ning Song, Christian Coarfa, Hernan Stamati, Andrea Ferrara, Kostantinos Tsavadis, Sumit Nain, Seth Fogarty, and others. All the discussions in the corridors, no matter how side-tracking they seemed, have been thought-provoking.

Last but certainly not least, I would also like to thank my parents for their emotional support in all these years.

Contents

Illustrations

Chapter 1

Introduction

The study of decision procedures for various logics has always been a central theme in the development of computer science. While there are many motivations, one of the most important reasons is that logic is the *natural* language of computer science. It is of no surprise that the canonical (and often, the first) problems shown to be complete for a complexity class are usually the decision problems for some classes of logic [Coo71, Sto76, BAHP82]. There are typically two problems that decision procedures for a class of logic needs to solve. The first is the *model-checking* problem, where the procedure decides whether the formula is true on *some given* model; the second is the *satisfiability* problem, where the procedure decides whether the formula is true on *at least one* model[1]. A decision procedure for satisfiability can often be used as the *universal* decision procedure for the complexity class by using the logic as a modeling language, where problem instances in that complexity class are modeled with a polynomially-sized formula. Practically, we have seen many examples of such applications, where many NP-complete problems (for example, planning [KS92], bounded reachability [BAC$^+$99, BCCZ99], and scheduling [CB94]) are solved using a decision procedure for the satisfiability of propositional logic (usually called a SAT solver). Model checking, while often an easier problem than satisfiability, is actually quite expressive (and in turn, quite complex) on many logics, for example, temporal logics, where many problems in formal verification are decided using *model-checkers*, in which the desired properties are presented as formulas in one of the temporal logics

[1]For logics that admit quantification, these two problems are typically inter-reducible from each other.

(CTL [CEA86, BCM+92, McM93], LTL[GPVW95], Forspec [AFF+02], and others), and the system to be verified as the model. Checking that the model satisfies the formula will verify that the system would exhibit the desired properties.

Still, using logic solvers as universal decision procedures is of no advantage in improving tractability, since the model-checking or satisfiability problems are just as hard as the problems we reduced from. Instead, the improvement is in the practical realm of solver development. Without using the logic solver as a universal decision procedure, each decision procedure for individual applications have to be optimized separately. In contrast, by using a logic solver, much of the optimization only need to take part inside the logic solver and not individual applications. In turn, the development of decision procedures for a large range of application domains is sped up. This is seen in the recent synergistic co-development of propositional satisfiability solvers (for example, SATO [Zha97], ZChaff [MMZ+01, ZM02a], BerkMin [GN02], etc.) along with the prosperity of SAT-solver-driven applications.

While there has been no final verdict on either the bridging or classifying the gap between deterministic and non-deterministic polynomial time (the “P vs. NP” problem), for most practical purposes it is believed that the containment P$\subset$NP is strict, i.e., that we cannot solve NP-complete problems in polynomial time, and theorems that can be proved using the assumption P$\neq$NP are very likely to hold in the real world. In fact, no sub-exponential algorithm[2] is known for NP-complete problems. However, actual implementations of algorithms do not always consume the worst-case time for all problem instances. In many cases, *heuristics* allows shortcuts to be taken in the execution of the algorithm. Since it is unlikely that P=NP, for an intractable class, heuristics are expected to only solve only a subset of the instances in polynomial time.

Classical complexity theory provides a tool to separate the complexity classes into the tractable and the intractable, but the development of heuristical solvers showed

[2]On a reasonable size metric.

that there is much more to the practical solvability of a problem than its classical complexity class. *Parameterized complexity* [DF99], on the other hand, is based on the insight that problems in a class are not uniform, instead, they have a *parameter*, controlling some aspect of the problem. Since the choice of the parameter can be essential to the difficulty of the problem, the complexity of the problem is described as a function on two variables, that of the size of the problem n, and the parameter k. One way to distinguish k and n is that each value of the parameter corresponds to a specialized algorithm which solves the problem for the particular parameter value. This can also be seen as a stratification of the original complexity class into a family of classes based on the parameter. One parameterized complexity class of particular interest if that of *fixed-parameter tractable* (FPT) class, in which the time complexity is in $O(f(k)poly(n))$, where f can be an arbitrary computable function. In other words, in an FPT class for every parameter k, there is a polynomial algorithm whose power does not depend in k. Surprisingly many problems that are classically intractable are FPT on naturally-occurring parameters. In particular, many problems on graphs can be solved efficiently if the graph is *tree-like*. This leads to the area of structural heuristics, where the structure of the problem is analyzed to reduce the effort needed to solve the problem. Beside many other interpretations, the distinct classes induced by distinct parameters on an FPT problem can be seen as a way to formalize the effects of heuristics, where the heuristics takes advantage of the feature defined by the parameter in the problem.

In this dissertation, we concern ourselves with symbolic algorithms (where the representation used is based on decision diagrams (DDs) [Bry86]) that attempt to exploit the structural properties of the problem instance, and compare them against classical search-based algorithms. Instead of restricting our attention to propositional satisfiability, we also study quantified satisfiability [CKS81], the canonical PSPACE-complete problem, for two reasons. First, quantified satisfiability is more useful than propositional satisfiability in encoding many problems because of highly compact encoding allowed by quantified satisfiability. Second, it is a more natural match to the semantics of DD-based approaches, since decision diagrams, but not propositional satisfiability, can handle both existential and universal quantification symmetrically. A bonus on studying both classes of problems is that we can gain insights on why PSPACE-complete problems are usually much harder than NP-complete problems to solve in practice, even though the best-known run-time upper-bound for both complexity classes are exponential.

The contrast between propositional satisfiability and quantified satisfiability is also exhibited in recent developments into new model-checking techniques. Much research into propositional satisfiability solvers is done for the purpose of improving bounded model checking (BMC) tools based on propositional satisfiability [BCCZ99]. While BMC tools based on propositional satisfiability solvers have been quite successful in practice, they still suffers from the fact that BMC has to tackle problem instances that are linear in the number of time steps unrolled; for some properties, this can lead to a serious strain on the resources consumed. On the other hand, classical BDD-based (unbounded) model checking algorithms often suffers from the size-explosion problem. Attempts to bridge the unbounded nature of model checking and the BMC-based approach took many directions, including k-induction, abstraction-refinement, SAT-based image computation, etc. (For a survey of these approaches, see [PBG05].) One of the directions uses QBF because many of the optimizations developed for SAT can be applied to a QBF solver, while at the same time QBF can represent unbounded model checking compactly. Still, in spite of the growing sophistication of QBF solvers, it is fair to say that they have shown no where near the effectiveness of SAT solvers [LST03]. We hope the theoretical and experimental developments we present in this dissertation can provide some insights on future development of QBF solvers.

1.1 Contributions

The contributions of this dissertation can be summarized in three aspects. First, we studied the effect of bounded pathwidth and bounded treewidth on the size of BDDs for CNF and existentially quantified CNF formulas. While these bounds have been studied for weaker representations (for example, Boolean circuits in [McM93], which can be seen as quantified formulas in a specific quantification scheme) in previous works, our investigation showed that the fixed-parameter tractability (for bounded pathwidth) or polynomial-ness (for bounded treewidth) of the size of the BDD can be maintained even when the formula is existentially quantified. With quantification, the effect from the width takes an exponential blowup compared to the non-quantified version. Second, we studied the application of structural heuristics on DD-based propositional and quantified decision procedures. We applied a number of previously developed heuristic schemes for clustering schemes [DP87, SV01], variable order [KBv01, Dec03], and symbolic representation [Bry86, Min96], first for propositional satisfiability, then for quantified satisfiability. While applying the heuristics mentioned for propositional satisfiability does not develop competitive symbolic solvers (vs. DPLL-based solvers), we learned that the performance of a BDD-based decision procedure is dependent on many factors in addition to the bounds induced by the structural heuristics. In contrast, applying the same heuristics for quantified satisfiability does lead to a competitive solver. Third, we develop a fixed-paremeter hierarchy in PSPACE based on studying powerful second-order model-checking problems on words in the spirit of [FG02]. The techinque used is called "telescoping", which allows very small formulas to be used to check complex properties on word models. Our improvement compared to [FG02] is three-fold. By replacing the logic used form monadic second-order logic as in [FG02] to quantified propositional temporal logic, we developed a tight lower-bound for each alternation-bounded fragment of the logic. Using temporal logic also allows us to use an unrolling approach to connect the complexity of model-checking problems with the complexity of bounded-width

QBF problems, and develop lower bounds that is tight against the upper bound of [Che04]. We also study the problem of small-width quantified satisfiability back in a classical complexity setting, and proved the NP-hardness (and through extending model checking to model-checking games, PSPACE-hardness) of $\log^*$-width QBF problems.

1.2 Outline

The dissertation starts with the formal definitions of the problems and logics in Chapter 2 (background). In Chapter 3 (bounded treewidth and BDD size), we investigate the impact of structural bounds on the size of BDDs, and also consider the complexity of quantification, and study the impact on BDD size from quantifying structural-bounded formulas. In Chapter 4 (search vs. symbolic decision procedures for propositional logic), we compare a range of heuristics for a symbolic solver of propositional logic. In Chapter 5 (parameterized complexity of bounded-width QBF), we generalize the lower bound from Chapter 3 and show a non-elementary lower bound on the parametric blowup of width and alternation depth parameter QBF. In Chapter 6 (hardness of small-width QBF), we use the techniques developed in Chapter 5 to prove the complexity of $\log^*$-width QBF. In Chapter 7 (a symbolic decision procedure for QBF), we return to a practical setting, and present a symbolic QBF decision procedure. Finally, in Chapter 8 (conclusion), we would conclude the dissertation and discuss possible future works.

Portions of this dissertation have been presented in conferences. A preliminary version of Chapter 3 have appeared in LPAR'2005 as [FPV05], Chapter 4 in SAT'2004 as [PV04a], Chapter 5 and 6 in LICS'2006 as [PV06], and Chapter 7 in CP'2004 as [PV04b].

Chapter 2

Background

2.1 Decision Diagrams

Binary decision diagrams (BDDs) [Bry86], also called *function graphs*, is a graph representation used to encode models of Boolean functions (which can be equivalently viewed as set of assignments to a set of binary variables, or set of bit vectors of a specific length) symbolically. As a restricted form of *branching programs* [Mei89], BDDs are rooted directed acyclic graphs (DAGs) with two terminal nodes labeled **0** and **1**. Each node in the graph is labeled with a variable and has two distinguished outgoing edges labeled **0** and **1**, indicating respectively the variable to check at this decision node and the successors (children) for each case. In other words, BDDs represent the *Shannon decomposition* of a Boolean function, where a node b labeled with variable i and with successors b', b'' represents the function $f_b = \textsc{ite}(v_i, f_{b'}, f_{b''})$, where $f_{b'}$ and $f_{b''}$ are functions represented by the nodes b' and b''. Here, $\textsc{ite}(a, b, c)$ is the *if-then-else* function, where $\textsc{ite}(1, b, c) = b$ and $\textsc{ite}(0, b, c) = c$. Compared against regular branching programs, BDDs also have the *read-once* property, where each variable can occur at most once in each path from the root to a terminal. Usually, we use the term BDDs to mean *reduced-ordered* BDDs (ROBDDs). In ordered BDDs, the set of variables is ordered and every path from the root to a terminal checks the variables in ascending order; in reduced BDDs, each node represents a distinct function. In turn, given an order of variables, the BDD for every Boolean function is *canonical*, i.e., the functions represented by two BDDs are equal iff the two BDDs are identical. As long as nodes can be shared between BDDs, equal functions would use the same BDD node. This allowed a range of properties on Boolean functions to be

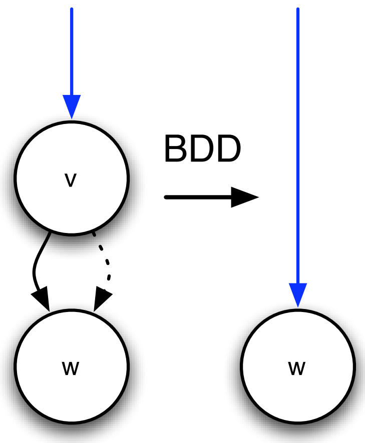


Figure 2.1 : Node elimination rule for BDDs

checked easily if they are in BDD representation, for example, a function is satisfiable iff its BDD is not the terminal node **0**. We define the *support set* of a BDD as the set of variables that label its internal nodes.

The node elimination rule needed to ensure canonicity is easy: Any node whose both children are the same node is redundant and can be removed, since $\text{ITE}(v_i, f, f) = f$. An illustration of this rule is shown in Figure 2.1.

Most Boolean operations on BDDs (including conjunction, disjunction, and complementation) can be performed in time quadratic in the size of both input BDDs, as long as both BDDs have the same variable order. This is because the Boolean operations on the Shannon decompositions can be recursively distributed. For example:

$\text{ITE}(v_i, b, b') \wedge \text{ITE}(v_i, c, c') = \text{ITE}(v_i, b \wedge c, b' \wedge c')$

One important thing to note is since BDD is a compressed data structure, there might be an exponential number of paths from the root to terminals in a single BDD, so a naive implementation of the above distribution routine is not going to finish in quadratic time in the size of the BDD. Practical implementations of BDD algorithms use *memorization*, where a “computed” table is kept for results of operations on

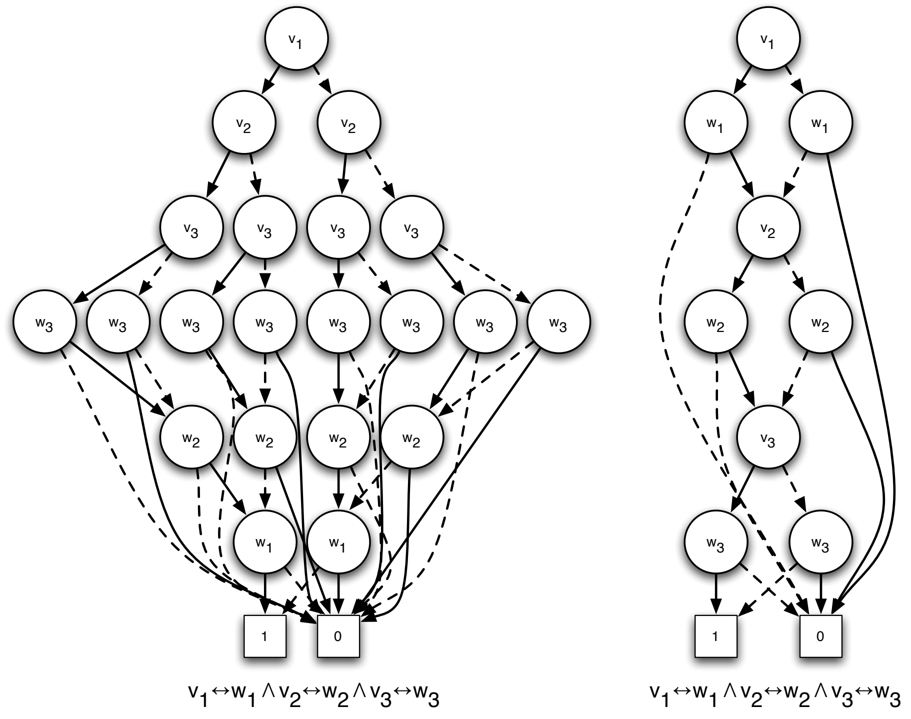


Figure 2.2 : Different variable orders for BDDs

nodes, so operations on internal nodes are not repeated, to maintain the quadratic time bound.

The main factor in the performance of BDD-based algorithms is the choice of the variable order. Different variable orders can cause a possible exponential difference in the size of BDDs, as demonstrated in Figure 2.2. A good variable order for a formula would need to put variables that are closely-related close together in the order.

In the experiments done in this dissertation, we used the BDD library from University of Colorado (CUDD) developed by Fabio Somenzi [Som98].

People have also studied a family of representations similar to BDDs, often only differing in the node elimination rule. One approach that has been particularly successful is that of zero-suppressed decision diagrams (ZDDs) [Min96], whose elimination rule is optimized towards representing sparse sets. Instead of eliminating nodes

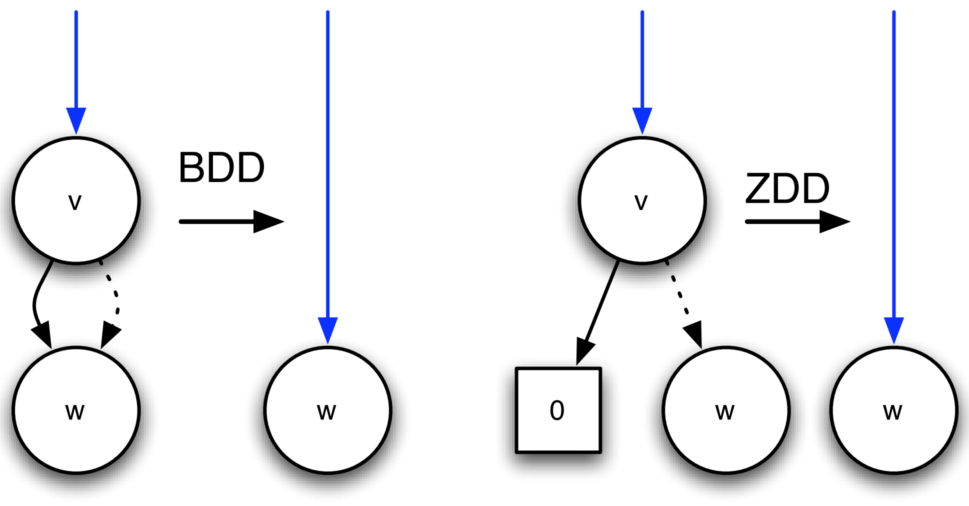


Figure 2.3 : BDD elimination rule vs. ZDD elimination rule

where both successors go to the same node, as for BDDs, ZDDs eliminate nodes where the 1 edge goes to the 0 terminal, i.e., each path to the 1 terminal represents a single distinct subset (unlike BDDs, where a single path can include many subsets) where only the 1 edges on the path represent occurring elements. See Figure 2.3 for an illustration on how BDDs are different from ZDDs.

While both BDDs and ZDDs can be used to represent similar objects, the fact that BDDs are more efficient on non-sparse sets and ZDD are more efficient on sparse sets means they are usually used for very different applications. Instead of being used to represent Boolean functions as a set of assignments, ZDDs are most often used to represent Boolean functions in CNF [CS00] or DNF [Min96] representations. A common ZDD-based representation uses two variables for the ZDD per variable of the Boolean function, representing respectively the positive and the negative literal. When ZDDs are used to represent CNF formulas, it is also advantageous to perform *subsumption-removal*, where subsumed clauses are not represented in the ZDD. For an example of using ZDDs to represent a CNF formula, see Figure 2.4. Note that the clause $\neg x_1 \vee \neg x_2$ is subsumed and does not occur in the ZDD.

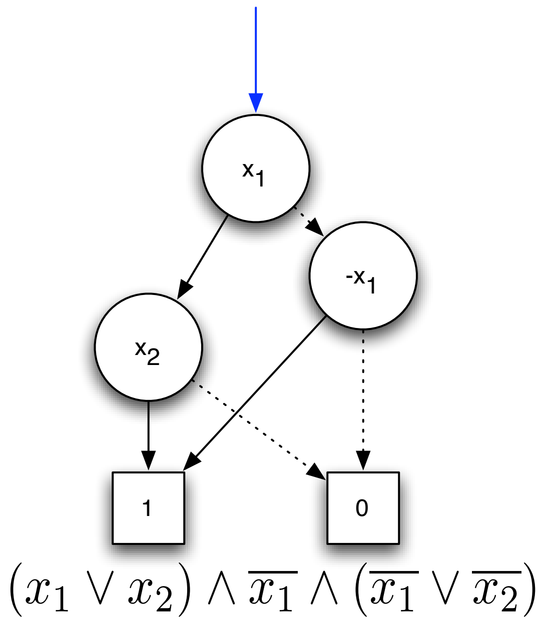

x_1
$-x_1$
x_2
1
0
$(x_1 \vee x_2) \wedge \overline{x_1} \wedge (\overline{x_1} \vee \overline{x_2})$

Figure 2.4 : ZDD representation of a clause set

In addition to the usual Boolean operations on decision diagrams, it is also possible to implement operations that are semantically quite complicated, for example, Davis-Putnam style resolution on ZDDs representing CNF formulas. An optimized version called multi-resolution is developed by Chatalic and Simon [CS00], which generates all possible resolvents on a variable while eliminating subsumed clauses. While multi-resolution is quite efficient most of the time, unlike the BDD conjunction and disjunction operators, there is no proof of polynomial time bounds on the operations.

2.2 Parameterized Complexity

Complexity theory studies the relationship between the amount of resources needed to decide a problem with respect to the size of the problem. Since the reduction between complete problems of a class is polynomial, many decision procedures for logic can be used as universal decision procedures for all problems of their characteristic complexity class.

Still, not all commonly studied logic decision problems are well-adapted to be used as a universal decision procedure. Most importantly, they may be are *too complex*. For example, the satisfiability problem of first-order logic is undecidable. One curious fact about such undecidability is that the logics themselves may not be that powerful in describing properties of models; their undecidability is due to the possible existence of unboundedly large models. This leads to a chain of research in *finite model theory*, where the size of the model is restricted to be finite. Many logics exhibit very different complexity behaviour under finite model assumption compared to the usual, unrestricted model case. One of the most important developments in finite model theory is that of *descriptive complexity*, where the question under consideration is that of the *descriptive power* of a logic, i.e., the complexity class for the set of finite models of arbitrary formulas. For example, Fagin [Fag74] showed that the descriptive power of existential second-order logic is NP-complete. In other words, for every NP-complete language, there is an existential second-order (ESO) formula that accepts exactly all strings in the language, and model checking ESO formulas on finite strings is NP-complete.

One important concept that is developed from the application of descriptive complexity theory in the area of databases is the distinction of *program complexity*, *data complexity*, and *combined complexity* [Var82]. The program complexity represents the complexity of model checking where the model is fixed, and the size of the formula is variable. The data complexity represents the opposite case, where the formula is fixed, and the model is variable. The combined complexity concerns the case where both are variable (i.e., the classical complexity). For the area of databases, the distinct complexities study how the cost scales with either the size of the query or the size of the database.

The fact that problems usually have many aspects contained in the input, and limiting the focus to the size of the input does not capture all aspects of the complexity of the problem is widely accepted in the general complexity community. In order to generalize the impact of multiple aspects on the complexity of a problem, *parameterized complexity* is developed [DF99]. In parameterized complexity, the complexity of a problem is analyzed in terms of both the problem-instance size n and a parameter k that relates to some property of the problem. Typically, this approach is used to capture the intuition that problems hard in the classical sense might have tractable classes, here represented as problems with a small k. The class of *fixed-parameter tractable (FPT)* problems are those that can be solved in time $f(k)\mathsf{poly}(n)$, where f is any computable function.

One interesting aspect of FPT classes is that many problems in classes harder than NP (for example, PSPACE, or even undecidable problems on an appropriate choice of parameter) can be stratified as a FPT class. Still, so far none of the research in FPT-based algorithms have improved the known lower bounds for intractable classes. Since we typically use parameterized complexity to study problems that are intractable in the classical setting, as long as the parameter k is no bigger than the size of the problem instance, $f(k)$ is at least exponential for known algorithms. Also, because we don't know whether P≠NP, little is known for actual lower bounds for $f(k)$.

In some cases, the FPT algorithms can actually be used as heuristics, by first attempting to classify the parameter the problem instance is in, then performing the FPT algorithm. In other cases, the FPT bound are used as a rationale on why the problem is tractable for practical cases, by showing that most practical cases do exhibit small k. Most research on parameterized complexity is in turn concentrated on parametrically intractable problems, and studies such intractability hierarchies. Still, simply because a problem is FPT does not always imply tractability. In [FG02], Frick and Grohe showed an FPT class where $f(k)$ is non-elementary (under a P≠NP assumption). In turn, given almost every k beyond a very small constant, the coeffient $f(k)$ is *astronomical*.

2.3 Propositional Logic and QBF

The satisfiability problem of propositional logic, being the canonical NP-complete problem, is a very popular tool to model and solve problems for a wide range of application domains. Problem instances in propositional logic are usually given in *conjunctive normal form* (CNF), which is a conjunction of disjunctions of literals. A satisfying solution A to a CNF formula φ is an assignment to the set of variables of φ, where every clause in φ contains a literal that is true under the assignment. This is written as $A \models \varphi$. All formulas in propositional logic can be translated to an equi-satisfiable formula in CNF through the Tseitin translation [Tse81].

Since satisfiability of propositional logic is NP-complete, it can be easily verified by presenting such an assignment A. On the other hand, unless NP=co-NP, verifying unsatisfiability would not be as easy. Currently, for powerful propositional proof systems, for example Frege or extended-Frege systems, there is no known upper bound for sizes of the proofs. Still, most automated decision procedures use a much simpler proof system called *resolution*, where two clauses $a \vee v_1$ and $\neg v_1 \vee b$ imply the consequent (also called *resolvent*) $a \vee b$. The first decision procedure for propositional logic, that of Davis and Putnam [DP60], uses resolution to generate all possible resolvents on a proposition before eliminating all clauses that contain that proposition. Thus, one proposition is eliminated from the support of the clause set at each step, while preserving the satisfiability (or unsatisfiability) of the problem. Repeated applications of this procedure will eventually eliminate all propositions until either an empty set (satisfiable case) or an empty clause (unsatisfiable case) is reached. This is, essentially, a *quantifier elimination*-based scheme.

Since the Davis and Putnam algorithm may take exponential space, in practice, most solvers for propositional logic are based on the Davis-Logemann-Loveland algorithm [DLL62] (usually called the DPLL algorithm), which attempts to search for a satisfying solution to the instance through extending partial assignments. At each step, if some literal needs to be added to the partial solution to satisfy some clause, the current partial assignment is extended onto the literal, otherwise, a currently unassigned literal is chosen. Backtracking is performed if contradicting assignments to a proposition occur.

For unsatisfiable instances, DPLL needs to cover the whole search space. To reduce the amount of backtracking needed, modern solvers use *conflict-driven learning*, where resolution is applied when a conflict is found to deduce the *reason* of the conflict. Since the search procedure essentially generates a resolution proof, problems that are proven to be *hard for resolution* (ones that require exponential-sized resolution proofs) have to consume exponential running time under DPLL. Other optimizations that are used in modern solvers include heuristically driven decision proposition choice and highly optimized Boolean constraint propagation [MMZ$^+$01].

An interesting characterization of the power of DPLL with conflict driven learning and restarts is by [BKS03], in which DPLL is shown to be as powerful as resolution. More precisely, for any unsatisfiable instance, given a resolution proof, there is a sequence of decisions, learning resolution choices, as well as restarts so the DPLL procedure simulates the resolution proof where the number of steps is polynomial in the size of the resolution proof. Of course, for satisfiable instances, the optimized decision heuristic used in modern DPLL solvers can guide the solver towards a solution very quickly, while resolution-based solvers have to generate all possible resolvents before concluding refutation is impossible.

Quantified Boolean formula (QBF) extends propositional logic by introducing quantifiers $\exists$ and $\forall$ on the propositions which range over the Boolean domain $\{0, 1\}$. We also call the propositions in QBF *variables* since their meaning is no longer fixed in the formula. We write formulas in QBF in prenex normal form $\psi = (Q_1 p_1)(Q_2 p_2) \ldots (Q_k p_k)\psi'$, where the Qs are quantifiers, the ps are propositions, and ψ' is a propositional formula, which we call the *matrix*. The sequence of quantifiers with their associated variables, are in turn, called the *prefix*. Usually, ψ' is written in CNF. The semantics of QBF is closely based on that of propositional logic, i.e., for a quantifier free formula φ, $A \models \varphi$ is defined as for propositional logic, and $A \models \exists p_i \varphi$ iff either $A[p_i \mapsto 0] \models \varphi$ or $A[p_i \mapsto 1] \models \varphi$, and $A \models \forall p_i \varphi$ iff both $A[p_i \mapsto 0] \models \varphi$ and $A[p_i \mapsto 1] \models \varphi$.

Since QBF formulas contain quantifiers, decision procedures for QBF need to do more than just search for a model, since unless NP=PSPACE, such a model cannot be verified in polynomial time. Instead, a QBF decision procedure typically searches for a *strategy* (also called a *semantic tree*), which maps every assignment to the universally quantified variables to at least one assignment to the existentially quantified variables. Based on the prefix, the assignment to any existentially quantified variable can only depend on the assignments to universally quantified variables outside it. For the worst case, the strategy is of size exponential in the number of universally quantified variables.

Since the actual "solution" to the decision problem of QBF can be exponentially big, modern QBF solvers also apply a learning procedure called *solution-driven learning* [ZM02b], in which Zhang and Malik claimed it to be symmetrical to conflict-driven learning. Solution-driven learning performs term resolution on satisfying assignments of the matrix to block out search spaces (partial assignments to universally quantified variables) that are guaranteed to contain solutions (matching assignments to existentially quantified variables).

Resolution for QBF is very similar to resolution for propositional logic, where two clauses that contains opposing literals are combined to develop a resolvent. One major difference is that we need to take the prefix into consideration when handling resolution for QBF. In particular, if two clauses $a \vee v_1$ and $\neg v_1 \vee b$ are resolved, the result is $a' \vee b'$ where a' (respectively, b') contains the set of literals in a (respectively, b) except that literals corresponding to universally quantified variables that occur after v_1 no longer occur. This is called Q-resolution and was developed by Buning, et. al. [BKF95].

By bounding the number of alternations, QBF can be stratified into classes Σ_k^{QBF}

and Π_k^{QBF}, where k is the number of alternations, and a formula is in Σ_k^{QBF} if its outermost quantifier is $\exists$ and contains k alternations and in Π_k^{QBF} if its outermost quantifier is $\forall$ and contains k alternations. The complexity hierarchy of deciding these formula classes corresponds to the *polynomial hierarchy* [Sto76]. In this dissertation, we consider the case where the matrix ψ' is restricted to CNF, so the innermost universal quantifier block introduces no additional complexity. In other words, we only consider QBF formulas in the classes Σ_k^{QBF} where k is odd and Π_k^{QBF} where k is even.

The additional descriptive power of QBF compared to propositional satisfiability is possibly useful in compressing the representation needed to describe real-world problems. Here, we show an example for bounded model checking.

In bounded model checking, variable substitutions are used to create distinct copies of the system. As an example, we present a simple case of bounded reachability, where the initial state set is represented using a formula I, and the goal state set is represented using a formula B. Given a formula f with support set $V = \{v_1, v_2, \dots v_n\}$, and a substitution variable set $V' = \{v'_1, v'_2, \dots v'_n\}$, we write $f[V/V']$ to represent a copy of f where each v_i in f is replaced with v'_i. To unroll a system to k iterations under propositional logic, we create $k+1$ copies of the state variable set V, which we call $V^0, V^1, \dots V^k$. The transition relation is a formula over $V \cup V'$, where V is the set of current state variables and V' is the set of next state variables. The BMC *unrolling* would contain $reach_{k,I,B} := I[V/V^0] \wedge B[V/V^k] \wedge \bigwedge_{0 \le i \le k-1} TR[V/V^i, V'/V^{i+1}]$. If each iteration uses a formula of size n, then k iterations would need a formula of size $O(kn)$.

Now, we see what we can do to unroll a system to $k = 2^j$ iterations under QBF. Instead of needing $k+1$ copies of the state variable set V as in propositional logic, QBF allows an iterative-squaring-like [BCM$^+$92] technique, where the midpoint of the path is guessed and two halves of the path are checked recursively as a path of length $k/2 = 2^{j-1}$. The idea behind this construction is also found in much earlier results relating alternation and space bounds, for example, [Sav70].

$reach_{2^j,I,B} := \exists W^0 \forall Y^1 \forall Z^1 \exists W^1 \forall Y^2 \forall Z^2 \ldots \exists W^{j-1} \forall Y^j \forall Z^j I(V/Y^0) \wedge B(V/Z^0) \wedge (\bigwedge_{0 \le i < j} (Y^{i+1} = Y^i \wedge Z^{i+1} = W^i) \vee (Y^{i+1} = W^i \wedge Z^{i+1} = Z^i)) \to TR[V/Y^j, V'/Z^j]$

This construction uses an $O(j+n)$-sized formula to encode 2^j step reachability. In a finite state system, for example, a symbolically encoded system with d bits, paths in the system cannot exceed 2^d in length, so QBF can be used to encode essentially *unbounded reachability*. There is also research in using similar QBF representations since QBF representations use only a single copy of the transition relation, in hope of reducing the number of learned clauses [DHK05].

2.4 Structural Heuristics and Width

Many decision problem on graphs are tractable when the structure of the graph satisfies certain constraints. A popular parameter for structural problems is that of *treewidth*, which measures how close a structure is to being a tree (trees have treewidth 1) [RS86]. We use the definition of treewidth as follows:

Definition 2.1. *(Robertson and Seymour* [RS86]*) Let $G = (V, E)$ be a graph. A* tree (path)-decomposition *of G is a pair $(T, \mathcal{X})$ where $T = (I, F)$ is a tree (path) with node set I and edge set F, and $\mathcal{X} = \{X_i \mid i \in I\}$ is a family of subsets of V, one for each node of T, such that*

- $\bigcup_{i \in I} X_i = V$,
- *for every edge $(v, w) \in E$, there is an $i \in I$ with $\{v, w\} \subseteq X_i$, and*
- *for all $i, j, k \in I$, if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.*

The width *of a tree (path)-decomposition is $\mathsf{max}_{i \in I} |X_i| - 1$. The* treewidth (respectively, pathwidth) *of a graph G, denoted by $tw(G)$ (respectively $pw(G)$), is the minimum width over all possible tree decompositions (respectively, path decompositions) of G.*

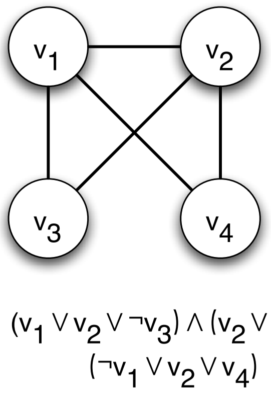


Figure 2.5 : A propositional formula and its interaction graph

When we want to use the label set as a function from nodes to vertex sets, we also write $\mathcal{X}(i)$ for X_i. For some cases, we also consider the pathwidth of a graph. Bounded pathwidth is clearly a more restrictive concept than bounded treewidth, since a path-decomposition can be used without change as a tree-decomposition. In other words, for all graphs G, $tw(G) \leq pw(G)$. Thus, any lower-bound result proved for bounded pathwidth also applies to bounded treewidth. Usually, we use w to denote treewidth of graphs and q to denote pathwidth.

The *width* of a formula is defined as the width of the *interaction graph* of its CNF form. The interaction graph is defined with the set of propositions as vertices, and the co-occurrence (in the same clause) relation between propositions as edges. An example is shown in Figure 2.5. For QBF, the interaction graph is built solely from the matrix, ignoring the prefix.

Many NP-complete problems, for example, Boolean satisfiability, are FPT with respect to the treewidth w on their structure, specifically $2^w n$ [DP89, Fre]. One way to rationalize the impact of treewidth of a system with its complexity is through the theory of communication complexity, which considers the amount of communication needed to solve a partitioned problem. For the case of bounded treewidth, if the communication needed between two halves of the tree is bounded by a function on

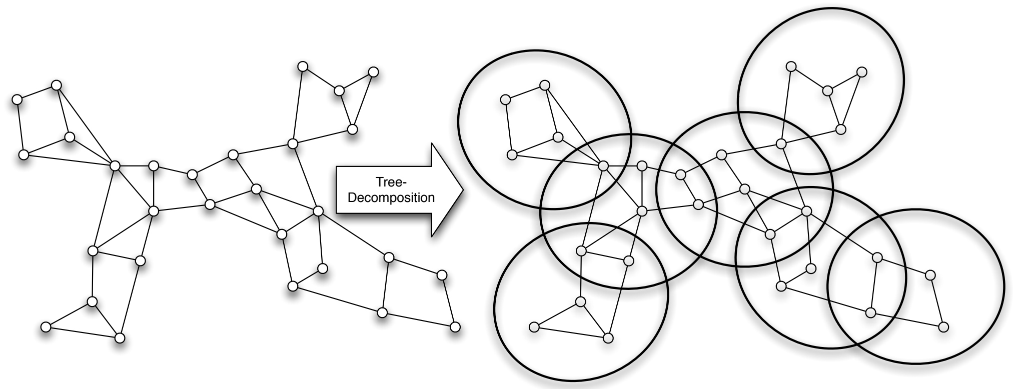


Figure 2.6 : A graph and its tree decomposition

the size of the label set of the tree node, then there exists a FPT algorithm if the problem restricted to each node can be solved polynomially.

Chapter 3

Bounded Treewidth and BDD size

Building the BDD of a propositional formula is a fundamental operation in BDD-based symbolic model checking. For applications using BDDs, the biggest challenge is to avoid a *space blowup*, where a very large number of nodes are generated to represent a certain function. This is hardly a suprising fact, since satisfiability is easy on the BDD representation, while NP-complete on a CNF representation. Thus, going from a CNF representation to a BDD representation can incur a possible exponential blow-up [BBG$^+$94]. We now show that this is not the case when the CNF formula has bounded treewidth through a detour of pathwidth. In this chapter, we will prove both upper and lower bounds for the BDD size of bounded pathwidth CNF formulas, as well as upper bounds for the BDD size of bounded treewidth CNF formula. For the bounded pathwidth case, the size of the BDD is linear in the number of propositions, while the coefficient is coefficient is exponential in the pathwidth for unquantified CNF formulas, and is double exponential in the pathwidth for existentially quantified CNF formulas. The double exponential blowup for the quantified case is *tight*. For the bounded treewidth case, the size of the BDD is polynomial in the number of propositions, where the power of the polynomial is the treewidth for the unquantified case, and exponential in the treewidth for the existentially quantified case. In this chapter, we will also point out how our results relate to previous results on the size of BDDs for bounded-width *circuits*.

We bound the number of nodes needed for a certain BDD through bounding the number of nodes *at each level*, i.e., labelled with a certain variable. Since BDDs are canonical representations of Boolean functions, the number of BDD nodes needed at

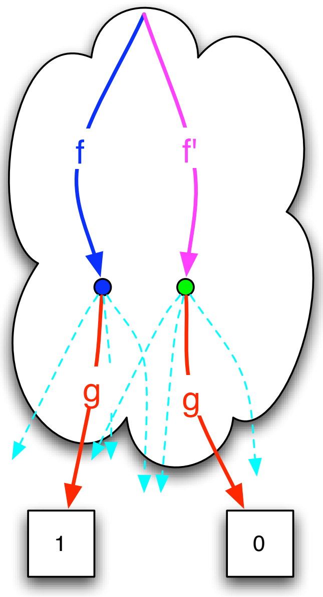


Figure 3.1 : Distinct nodes in a BDD

each level is the number of distinct Boolean functions needed to cofactor the Boolean function onto partial assignments on *visited* propositions. For two paths f and f' from the root of the BDD to lead to different nodes, there needs to be a *witness* for the fact. As shown in Figure 3.1, the nodes are on some level i with variable v_i. Both f and f' are (possibly partial) assignments to variables v_1 to v_{i-1}. For the two nodes to be distinct, the witness is an assignment g to the variables v_i to v_n where the path $f \circ g$ leads to a different terminal from $f' \circ g$.

Theorem 3.1. *A CNF formula C with n variables and pathwidth q has a BDD of size $O(n2^q)$.*

Proof. Let the path decomposition of C be (P, L). Assume without loss of generality that $P = \{1, \ldots, k\}$. We construct a variable order from the path decomposition as follows: Define $First(x) = \min(\{p \in P \mid x \in L(p)\})$ and $Last(x) = \max(\{p \in P \mid x \in L(p)\})$. Now sort the variables in increasing lexicographic order according to

$(First(x), Last(x))$; that is, define the variable order so that if $x < y$, then either $First(x) < First(y)$ or $First(x) = First(y)$ and $Last(x) < Last(y)$. We show that, using this variable order, there are at most 2^q nodes per level. The claim then follows.

For each clause c, we define $\min(c)$ as the index of the lowest ordered variable in c and correspondingly for $\max(c)$. Consider level i of the BDD, corresponding to the variable x_i. The clause set C can be partitioned into three classes with respect to level i, $C_{ended} = \{c \mid \max(c) < i\}$, $C_{cur} = \{c \mid \min(c) \leq i < \max(c)\}$, and $C_{untouched} = \{c \mid i < \min(c)\}$.

A node u at level i corresponds to a set A_u of partial assignments to variables, where each partial assignment $a \in A_u$ is an element in $2^{\{x_1 \dots x_{i-1}\}}$. For a partial assignment a and a clause set D, we write $a \models D$ if a is a model of D, i.e, for each clause $c \in D$, a satisfies some literal in c. From the semantics of BDDs, we know that all partial assignments a in A_u are equivalent with respect to extensions, i.e., given $a' \in 2^{\{x_i, \dots, x_n\}}$ and $a \in A_u$, we have that $a \cup a' \models C$ iff for every $a'' \in A_u$, $a'' \cup a' \models C$. If for $a \in A_u$, $a \not\models C_{ended}$, then we know that for every extension $a \cup a'$ of a we have that $a \cup a' \not\models C_{ended}$, so $a \cup a' \not\models C$. Thus, the node u is identical to Boolean 0 and should not exist at level i. It follows that for every $a \in A_u$, $a \models C_{ended}$. We also know that all clauses in $C_{untouched}$ have none of their variables assigned by $a \in A_u$.

Each partial assignment a at level i can be associated with a subset $M_a \subseteq C_{cur}$ where $M_a = \{c \mid c \in C_{cur}, a \models c\}$, i.e., the clauses in C_{cur} that are already satisfied by a before reading the variable x_i. We know that none of the clauses in C_{cur} have failed (all literals assigned to false) so far, since by definition of C_{cur} all such clauses have literals with variables beyond x_{i-1}. Suppose that for two distinct nodes u and v at level i there exists $a_u \in A_u$ and $a_v \in A_v$ such that $M_{a_u} = M_{a_v}$. Since u and v are distinct, there is a partial assignment $a \in 2^{\{x_i, \dots, x_n\}}$ that distinguishes between u and v; say, $a_u \cup a \models C$ and $a_v \cup a \not\models C$. Since a_u and a_v, however, both satisfy C_{ended}, both are undefined on the variables of $C_{untouched}$, and we also have, by assumption, that $M_{a_u} = M_{a_v}$, we must have that $a_u \cup a \models C$ iff $a_v \cup a \models C$ – a contradiction. It follows that $M_{a_u} \neq M_{a_v}$.

Let $j = First(x_i)$. We know that $L(j)$ contains at most $q+1$ variables, including x_i. Let $Var_i = L(j) \cap \{x_1, \ldots, x_{i-1}\}$, then Var_i has at most q variables. Suppose that u and v are two nodes at level i such that there exists $a_u \in A_u$ and $a_v \in A_v$ where a_u and a_v agree on Var_i. We show then $M_{a_u} = M_{a_v}$. Consider a clause $c \in C_{cur}$. We know that all the variables of c occur in $L(k)$ for some k. We cannot have $k < j$, since then we would have $c \in C_{ended}$, so $k \geq j$. If x_h occurs in c for some $h < i$, then by construction $x_h \in L(j')$ for some $j' \leq j$. By the property of path decompositions it follows that $x_h \in L(j)$. Since a_u and a_v agree on Var_i, it follows that they agree on c. We showed that if u and v are distinct, then for every $a_u \in A_u$ and $a_v \in A_v$, $M_{a_u} \neq M_{a_v}$. It follows that a_u and a_v cannot agree on Var_i. Since Var_i has at most q variables, there can be at most 2^q nodes at level i. The claim follows since the BDD has n levels. □

The relationship described in Theorem 3.1 between pathwidth and BDD size was first shown in [HD04]. The proof there goes via a variant of a DPLL-based satisfiability algorithm. Our argument here is direct and show how to obtain a BDD variable order from a path decomposition.

We know that for a graph G that contains n vertices we have that $pw(G) = O(tw(G) \cdot \log n)$ (For example, through the algorithm presented in [BK96]).

Corollary 3.2. *A CNF formula C with n variables and treewidth width q has a BDD of size polynomial in n and exponential in q.*

While Theorem 3.1 suggests that BDD-based algorithms are tractable on bounded width problems, typical model-checking algorithms need to do more than just build BDDs that correspond to CNF formulas. BDDs are often used to perform symbolic image operations, which requires applying existential quantification to BDDs [McM93]. While the theory of fixed-parameter tractability is built upon the premise that when the parameter is bounded, the problem becomes tractable, the constant coefficient that comes from the $f(k)$ needs to be considered on a case-by-case basis.

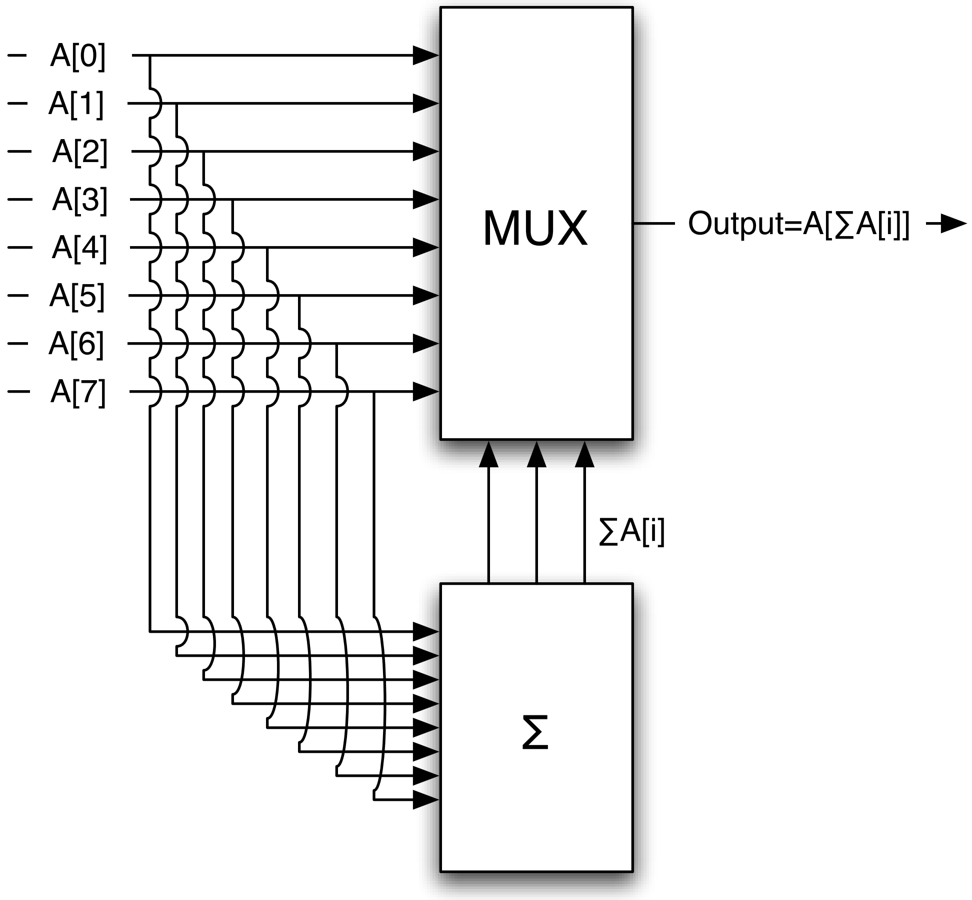


Figure 3.2 : The Hidden Bit Value Function

Often, super-exponential blowups in the parameter result in astronomical coefficients that greatly overshadow the polynomial nature of the algorithm, and makes most instances practically unsolvable. The following theorem shows that Theorem 3.1 is not likely to be useful in model checking, since using quantification on bounded-width formulas leads to such a super-exponential blowup from the parameter to the constant coefficient of the cost of the algorithm.

Theorem 3.3. *There exists a formula C in CNF with n variables and pathwidth q, and a subset of variables X of C such that $(\exists X)C$ under every variable order does* *not have a BDD of size $n2^{f(q)}$, such that f is a sub-exponential function.*

Proof. We consider the hidden-weighted bit (HWB) function, which is shown by Bryant in [Bry91] to have a BDD size of $\Omega(1.14^m)$ under arbitrary variable order, where m is the number of input bits. The HWB function (see Figure 3.2) is a Boolean function $2^m \to \{0, 1\}$, where for an m-bit input vector A, the output is the wth bit of A, w being the number of 1s in A (the *bit count* of A). The BDD is defined on the set of variables $A[0]$ to $A[m-1]$.

We consider the case where $m = 2^k$, $k > 3$, and construct a CNF formula to represent the HWB function. Clearly, from the upper bounds shown in Corollary 3.2, a direct translation cannot result in bounded pathwidth, that would imply a polynomial BDD size; we use $(m+1)k+1$ additional existentially quantified variables to facilitate the CNF encoding. In the additional variables, there are $m+1$ counters (at k bits each), which we call $X_0, \ldots X_m$, and a single bit witness w. Each X_i is used to guess the number of 1s occurring after $A[i]$. The bit witness w guesses the value of $A[X_0]$. We use CNF constraints to check the correctness of our guesses. The CNF formula C is the conjunction of all the following constraints. ($=$ and $+$ are short hand of the usual meaning defined on bit vectors of size k representing natural numbers up to $2^k - 1$):

- For each $0 \leq i < m$, we define $C_i^1 := (A[i] \to X_i = X_{i+1} + 1) \wedge (\neg A[i] \to X_i = X_{i+1})$. This asserts that if X_i is a correct guess iff X_{i+1} is a correct guess.

- For each $0 \leq i < m$, we define $C_i^2 := (X_0 = i) \to (A[i] \leftrightarrow w)$. This asserts that w is a correct guess if X_0 is a correct guess.

- $C^g := w$. Since we are building the BDD representing inputs where the HWB function returns 1, w is asserted to true.

- The well-formedness constraint is $C^{wf} := X_m = 0$. This asserts the guessed X_m is correct. Combined with the C_i^1s, they assert that all X_is are correct guesses.

The only shorthand we use above is $=$ and $+$ on bit vectors of length k, both of which can be written out in CNF with no additional variables and $O(k^2)$ clauses. Now, $(\exists X_0)\ldots(\exists X_m)(\exists w)C$ characterizes the HWB function.

Next we show that there is a path decomposition of C of width $3k+1$. There is one node per bit in A, ordered from 0 to $m-1$. Each node contains the support variables for the constraints C_i^1 and C_i^2 (the last node also contains C^g and C^{wf} with no additional variables). In turn, each node i contains the variables $A[i]$, w, X_0, X_i, and X_{i+1}, which gives a pathwidth upper bound of $3k+1$.

Consider the relationship between the size of the BDD and the pathwidth. Assume we have a BDD of size $n2^{f(q)}$, where the pathwidth is q, the number of variables is n, and f is a sub-exponential function. Here, $q \leq 3k+1$ and $n = (m+1)k+1+m = (2^k+1)(k+1)$. The size of the BDD S is then $|S| \leq ((2^k+1)(k+1))2^{f(3k+1)} < 2^{(k+3)}2^{f(3k+1)} = 2^{f(3k+1)+k+3} = 2^{g(k)}$. Since f is sub-exponential, g is sub-exponential as well. But from [Bry91], the lower bound for the size of such BDDs is $\Omega(1.14^m) = \Omega(2^{2^k \times \log\ 1.14})$, which contradicts with g being sub-exponential. So such small BDDs cannot exist.

$\square$

The use of HWB to prove blowups in bounded-width problems have been done in [McM93], where BDDs for bounded-treewidth circuit graphs are shown to incur a double-exponential blowup on *backward width*. It is quite easy to see the result here is a consequence of McMillan's lower bounds construction, because a bounded-width circuit can be easily written as a quantified bounded-width formula by introducing a new variable for each gate and quantifying all variables that corresponds to gates, only keeping those that correspond to input wires.

Next we show that our construction is almost worst case, i.e., there is a closely related upper-bound. While the theorem below is also analogous to McMillan's result on bounded pathwidth circuits, the construction we used is new, because quantified CNF is a more powerful representation model than circuits.

Theorem 3.4. *For a CNF formula C on n variables with pathwidth q and a subset of variables X, the formula $(\exists X)C$ has a BDD of size $O((n-|X|)2^{2^q})$.*

Proof. To get the upper bound, we use the same approach as the Theorem 3.1, i.e., we show an upper bound of 2^{2^q} nodes for nodes at each level i by counting the number of equivalence classes.

We use $supp(C)$ to denote the support of C, and define $Y = supp(C) - X$ as the set of *free* variables in $(\exists X)C$. We use the same variable order as Theorem 3.1, and name the variables in Y as $y_1, y_2 \ldots y_m$ according to the variable order. While the variables in X does not appear in the BDD, for the purpose of the variable order, we pretend they are interspersed with the Y variables. For a set $Z \subseteq supp(C)$, we use $Z_{<i}$ to denote the subset that appears before y_i in the variable order. Also, Z_j is used to denote the subset of Z that occurs in path-decomposition node j. Each node u corresponding to a variable y_i represents a set of assignments A_u to $Y_{<i}$, which is encoded by the paths to the node from the root of the BDD. Consider an assignment $a \in A_u$. For each assignment $b \in 2^{X_{<i}}$ to the quantified variables occurring before y_i, we have a corresponding set of clauses in C that are satisfied by $a \cup b$. Assume that y_i occurs in node k of the path decomposition of C. Recall that C can be partitioned into C_{ended}, C_{cur}, and $C_{untouched}$ based on the variable y_i. Define the function $F_a : 2^{X_{k,<i}} \to \{\bot\} \cup 2^{C_{cur}}$ such that for each assignment b to $X_{k,<i}$, $F_a(b) = \bot$ if there is no extension b' (on $X_{<i}$) of b such that $a \cup b' \models C_{ended}$; otherwise, $F_a(b) = S$ where $S \subseteq C_{cur}$ is the clauses in C_{cur} satisfied by $a \cup b$. Now, we show that two distinct nodes u and v corresponding to y_i do not contain assignments a_u in A_u and a_v in A_v such that $F_{a_u} = F_{a_v}$. Assume the contrary. Since u and v are distinct, w.l.o.g., there is an assignment a to $Y_{\geq i}$ such that $a_u \cup a \models (\exists X)C$ and $a_v \cup a \not\models (\exists X)C$. Take an assignment b on X where $a_u \cup a \cup b \models C$. Let b' be the restriction of b to the variables in $X_k \cup X_{\geq i}$, and let b'' be the restriction of b to the variables in $X_{k,<i}$. It is clear that $a \cup b' \models C_{untouched}$. We know that $F_{a_u}(b'') \neq \bot$, since b restricted to $X_{<i}$, which we call b_{a_u}, satisfies $a_u \cup b_{a_u} \models C_{ended}$. Since $F_{a_u} = F_{a_v}$,

$F_{a_v}(b'') = F_{a_u}(b'') \neq \bot$. Again, we have an extension b_{a_v} (from the definition of F_{a_v}) of b'' to $X_{<i}$ where $a_v \cup b_{a_v} \models C_{ended}$. For a clause $c \in C_{cur}$, if $c \in F_{a_u}(b'')$, then $c \in F_{a_v}(b'')$, so $a_v \cup b_{a_v} \models c$. Otherwise, $a \cup b' \models c$, since $a_u \cup a \cup b \models c$ and $a_u \cup b'' \not\models c$. So, $a_v \cup b_{a_v} \cup a \cup b' \models C_{cur}$. In summary, $a_v \cup a \cup b_{a_v} \cup b' \models C$, which contradicts with $a_v \cup a \not\models (\exists X)C$.

Now we count the number of possible functions for F_a. For each $b \in 2^{X_{k,<i}}$, the number of possible choices of $F_a(b)$ is $1 + 2^{|Y_{k,<i}|}$ since the satisfaction of clauses in C_{cur} depends only on b and assignments to $Y_{k,<i}$. Thus, the number of possible such F_as is $(1 + 2^{|Y_{k,<i}|})^{2^{|X_{k,<i}|}} \leq (2^{|Y_{k,<i}|+1})^{2^{|X_{k,<i}|}} = 2^{(|Y_{k,<i}|+1)2^{|X_{k,<i}|}} \leq 2^{2^q}$ since $q \geq |X_{k,<i}| + |Y_{k,<i}|$.

The combination of the possible numbers of F_as and the fact that distinct nodes induce distinct F_as gives us a bound of 2^{2^q} nodes at each level, i.e., a size bound of $(n - |X|)2^{2^q}$ for the whole BDD.

□

The double exponential blowup for the BDD size of quantified bounded pathwidth formulas on the pathwidth prevents us from using the property $pw(G) = O(tw(G)\log n)$ to achieve a polynomial size BDD for quantified bounded treewidth formulas. Still, it is possible to prove a similar bound as in [McM93] based on Theorem 3.4.

Theorem 3.5. *For a CNF formula $C = \bigwedge c$ on n variables with treewidth w and a subset of variables X, the formula $(\exists X)C$ has a BDD of size $O((n - |X|)n^{2^{w+1}})$.*

Proof. In this proof, we use T to denote the tree decomposition for C that have width w. $|T|$ is used to denote the number of nodes in T. Based on the concept of *nice* tree decomposition in [Klo94], $|T|$ can always be bounded to $O(n)$.

We follow the outline of the previous proof, where the only difference is in the analysis of the number of possible choices for F_as under the bounded treewidth assumption.

In the previous proof, we count the number of possible choices of a function $F : D \to R$ where D is the domain and R is the range as R^D. Because the variable support of the clause set C_{cur} is limited to the label set of one single path decomposition node, The number of possible choices of F_a is bounded by 2^{2^q} where q is the pathwidth. Given a tree decomposition of width w and size $|T|$, we can "flatten" it into a path decomposition of width $w \log |T|$ through the procedure described in [McM93] that flattens tree decompositions to path decompositions. One point to note about the resulting path decomposition is that the label set for each path decomposition node contains labels from at most $\log |T|$ tree decomposition nodes, and if we define a direction on the path decomposition (the direction we will process variables in), we can construct the flattening in a way that for each pair of each tree decomposition nodes in same path decomposition node their common parent can only appear in that path decomposition node or later.

After observing that there are $q = w \log |T|$ variables that contribute to the number of choices for F_a, we now show because the variables came from $\log |T|$ tree decomposition nodes, the number of possible choices can be reduced. To that end, we define the concept of decomposability for functions on the same domain and range:

Definition 3.6. *A function $f(\vec{v'} \cup \vec{v''})$ is decomposible into two functions $f'(\vec{v'})$ and $f''(\vec{v''})$ iff for all assignments a to $\vec{v'} \cup \vec{v''}$, $f(a) = f'(a|_{\vec{v'}}) \circ f''(a|_{\vec{v''}})$.*

Here, $\circ$ is any binary (combining) function. If a function is decomposed into k functions, then we would need to use a k-ary function. One important thing to note is that given a domain, range, and a fixed combining function, the possible number of functions that have a specific decomposition is much smaller than the number of all functions:

Claim 3.7. *If a function $f : D \to R$ can be decomposed into k functions $f_1 : D_1 \to R_1$, $\ldots$, $f_k : D_k \to R_k$, then the possible choices of such decomposible fs is less than $\Pi_{1 \leq i \leq k} |R_i|^{|D_i|}$.*

Now we show that the F_a we need is decomposible into $\log |T|$ functions, one for each tree decomposition node that is relevant. Here, instead of viewing F_a as a unary function on assignments (bit vectors) of some length i, we treat it equivalently as an i-ary function on Booleans solely for the purpose of allowing each decomposed function to only use a subset of the variables in its domain.

Claim 3.8. *F_a can be decomposed into $\log |T|$ functions, where each of the functions have up to $2^{2^{w+1}}$ possibilities.*

Proof. Recall from the definition of F_a, the function maps each assignment from the visited quantified variables in the current path decomposition node either to a subset of clauses in C_{cur}, or $\perp$, which denote some clause in C_{ended} is not satisfiable on the current assignment a and the given assignment of the visited quantified variables. Each of the clauses in C_{cur} is supported by one or more of the relevent tree decomposition nodes $t_1, \ldots t_{\log |T|}$ (that is part of the current path decomposition node). Now, we define the decomposed functions $F_a^1, \ldots F_a^{\log |T|}$ as follows: For each $1 \le j \le \log |T|$, F_a^j is a function from assignments on $X_{k,<i} \cap \mathcal{X}(t_j)$ the subset $C_{cur}^{t_j}$ of C_{cur} whose variable support is entirely contained in $\mathcal{X}(t_j)$. Each of the F_a^j is defined to have the same semantics as F_a. We also define the $\log |T|$-ary combining function $\circ(r_1, \ldots r_{\log |T|})$ as: $\circ(r_1, \ldots r_{\log |T|}) = \perp$ iff any r_j is $\perp$, otherwise for a clause $c \in C_{cur}$, $c \in \circ(r_1, \ldots r_{\log |T|})$ iff there exists $1 \le j \le \log |T|$ such that $c \in r_j$.

We now show that the decomposition is correct, i.e. $F_a(b) = \circ(F_a^1(b|_{\mathcal{X}(t_1)}), \ldots, F_a^{\log |T|}(b|_{\mathcal{X}(t_{\log |T|})}))$.

First, consider the case $F_a(b) = \perp$. Assume the decomposition is incorrect, i.e., $\circ(F_a^1(b|_{\mathcal{X}(t_1)}), \ldots, F_a^{\log |T|}(b|_{\mathcal{X}(t_{\log |T|})})) \neq \perp$. This means for all j, $F_a^j(b|_{\mathcal{X}(t_j)}) \neq \perp$. From the definition of F_a, $F_a(b) = \perp$ means that there are no extensions b' of b onto $X_{<i}$ where $a \cup b' \models C_{ended}$, but for every j, since $F_a^j(b|_{\mathcal{X}(t_j)}) \neq \perp$, there are extensions $b|'_{\mathcal{X}(t_j)}$ of $b|_{\mathcal{X}(t_j)}$ where $a \cup b|'_{\mathcal{X}(t_j)} \models C_{ended}$. Next, we partition C_{ended} into $\log |T|$ subsets based on the tree decomposition T. First, for each tree decomposition node t_j that occurs in the current path decomposition node, we consider the sub-tree T_j

of the tree decomposition T rooted at t_j (toward the direction of "processed" tree decomposition nodes). The set C^j_{ended} of clauses is the subset of C_{ended} where every clause in C^j_{ended} have all its variables contained in the union of all the label set of nodes in T_j. Since T is a tree decomposition, we know that (1) each of the clauses in C_{ended} must occur in at least one of the C^j_{ended}s. Also, we have that (2) if any variable x that is in the support set of C_{ended} occurs in the support set of more than one of the C^j_{ended}s, that variable must occur in $\bigcap_{\{j|x \in supp(C^j_{ended})\}} \mathcal{X}(t_j)$ because each of the sub-tree T_js are disjoint. Now, we consider the restriction $b|''_{\mathcal{X}(t_j)}$ of $b|'_{\mathcal{X}(t_j)}$ to the variables that occurs in the label set of nodes in T_j. Clearly, $a \cup b|''_{\mathcal{X}(t_j)} \models C^j_{ended}$. Next, we show that the $b|''_{\mathcal{X}(t_j)}$s can be combined to give a b' such that b' is an extension of b and $a \cup b' \models C_{ended}$. This is done by the simple union of the assignments in each of $b|''_{\mathcal{X}(t_j)}$s since from (1), all the clauses in C_{ended} would be witnessed by such an union. What is left is to show that such an union is consistent, i.e., that the different $b|''_{\mathcal{X}(t_j)}$s do not contain contradictory assignment on variables. From (2), we know that a contradiction cannot occur, since a variable assigned in $b|''_{\mathcal{X}(t_j)}$ but not $b|_{\mathcal{X}(t_j)}$ is not assigned any other $b|''_{\mathcal{X}(t'_j)}$s because of the support set needed in each case. Thus, $F_a(b) \neq \perp$, and through contradiction, we know that there exists some j in $1 \leq j \leq \log |T|$ where $F^j_a(b|_{\mathcal{X}(t_1)}) = \perp$.

Now, consider the case where $F_a(b) \neq \perp$. In this case, there exists an extension b' of b on quantified variables such that $a \cup b' \models C_{ended}$. Clearly, such a b' is also an extension of $b|'_{\mathcal{X}(t_j)}$, so for all j, $F^j_a(b|'_{\mathcal{X}(t_j)}) \neq \perp$. Next we consider the clause set that is $F_a(b)$. For each clauses $c \in C_{cur}$ where $c \in F_a(b)$, we know $a \cup b \models c$. From the property of the tree decomposition, we know that the support set of c is in some $\mathcal{X}(t_j)$ for some node t_j. Thus, $a \cup b|_{\mathcal{X}(t_j)} \models c$, which means $c \in F^j_a(b|_{\mathcal{X}(t_j)})$. For each clause $c \in C_{cur}$ where $c \notin F_a(b)$, we know $a \cup b \not\models c$. Thus, for any restriction $b|_{\mathcal{X}(t_j)}$, $a \cup b|_{\mathcal{X}(t_j)} \not\models c$, i.e., $c \notin F^j_a(b|_{\mathcal{X}(t_j)})$.

The number of choices for each of the F^j_as is $(1 + 2^{|Y_{k,<i} \cap \mathcal{X}(t_j)|})^{2^{|X_{k,<i} \cap \mathcal{X}(t_j)|}} \leq 2^{(|Y_{k,<i} \cap \mathcal{X}(t_j)|+1)2^{|X_{k,<i} \cap \mathcal{X}(t_j)|}} \leq 2^{2^{w+1}}$ since $w + 1 \geq |X_{k,<i} \cap \mathcal{X}(t_j)| + |Y_{k,<i} \cap \mathcal{X}(t_j)|$. $\square$

From the above decomposition result, and the size bound of $2^{2^{w+1}}$ for the number of choices of each decomposed function (as shown in the proof for the previous theorem), we get a $(2^{2^{w+1}})^{\log|T|}$ bound on the number of choices of F_a, which is $= |T|^{2^{w+1}}$. Since $|T|$ in $O(n)$, the bound for the number of choices for F_a is in $O(n^{2^{w+1}})$, i.e., polynomial in n for constant treewidth. In turn, the size of the BDD is in $O((n-|X|)n^{2^{w+1}})$. $\square$

Chapter 4

Search vs. Symbolic decision procedures for propositional logic

In the previous chapter, we studied how one class of structural constraints, namely bounded pathwidth or bounded treewidth on the interaction graph of propositional formulas, relates to the size of BDDs that represents those formulas. Still, building the BDD of a formula is a problem of *enumeration* instead of *satisfiability solving*. Recent work has shown how to use BDDs for satisfiability solving rather than enumeration [SV01]. The idea of this approach, which we call *symbolic quantifier elimination*, is to view an instance of propositional satisfiability as an existentially quantified propositional formula. Satisfiability solving then amounts to quantifier elimination; once all quantifiers have been eliminated we are left with either **1** or **0**. This enables us to apply ideas about existential quantifier elimination from model checking [RAB$^+$95] and constraint satisfaction [DP87]. The focus in [SV01] is on studying the relation between running time with the density and size of random 3-SAT instances. Only a minimal effort is made there to optimize the approach and no comparison to search methods is reported. Nevertheless, the results in [SV01] show that BDD-based algorithms behave quite differently than search-based algorithms. While both BDD-based and search-based algorithms exhibit an easy-hard-less-hard running time pattern as density scales, they peak at different densities. More study is in turn needed to understand the performance patterns of BDD-based satisfiability algorithms.

The goal in this chapter is to study the complexity and practical performance of symbolic quantifier elimination as an approach to satisfiability solving. To this end, we conduct a direct comparison with the DPLL-based ZChaff. In comparing the symbolic approach to ZChaff we use a variety of classes of formulas. Unlike the standard practice of comparing solver performance on benchmark suites [LS03], we focus here on *scalability*. That is, we evaluate how performance scales with formula size and limit us to benchmarks where we can generate successively bigger formulas. Similar to the comparison of BDD and search on solution enumeration done by Uribe and Stickel [US94], we also find that no approach dominates across all classes. While ZChaff dominates for many classes of formulas, the symbolic approach is superior for other classes of formulas. We also apply a number of optimizations and variants of the BDD-based solver to investigate the effects of the optimizations.

4.1 An algorithm for BDD-based propositional satisfiability

In [CDS+03, US94], BDDs are used to construct a compact representation of the set of all satisfying truth assignments of CNF formulas. The input formula φ is a conjunction $c_1 \wedge \ldots \wedge c_m$ of clauses. The algorithm constructs a BDD A_i for each clause c_i. The BDDs A_i have size linear in the size of the clause since only one *cube* is excluded. A BDD for the set of satisfying truth assignment is then constructed incrementally; B_1 is A_1, while B_{i+1} is the result of $\textsc{apply}(B_i, A_i, \wedge)$, where $\textsc{apply}(A, B, \circ)$ is the result of applying a Boolean operator $\circ$ to two BDDs A and B. Finally, the resulting BDD B_m represents all satisfying assignments of the input formula.

The satisfiability problem is to determine whether a given formula $c_1 \wedge \ldots \wedge c_m$ is satisfiable. In other words, the problem is to determine whether the existential formula $(\exists x_1) \ldots (\exists x_n)(c_1 \wedge \ldots \wedge c_m)$ is true. We can apply existential quantification to a BDD B by performing:

$$(\exists x)B = \textsc{apply}(B|_{x \leftarrow 1}, B|_{x \leftarrow 0}, \vee),$$

where $B|_{x \leftarrow c}$ restricts B to truth assignments that assign the value c to the variable x. Note that quantifying x existentially eliminates it from the support set of B. Since checking whether the final BDD B_m is equal to $\mathbf{0}$ can already be done in constant time (by checking whether it is the same node as $\mathbf{0}$), it makes little sense to apply existential quantification to B_m. Suppose, however, that a variable x_j does not occur in the clauses $c_{i+1}, \ldots, c_m$. Then the existential formula can be rewritten as

$$(\exists x_1) \ldots (\exists x_{j-1})(\exists x_{j+1}) \ldots (\exists x_n)((\exists x_j)(c_1 \wedge \ldots \wedge c_i) \wedge (c_{i+1} \wedge \ldots \wedge c_m)).$$

Pursuing this rewriting strategy as aggressively as possible, we process the clauses in the order $c_1, \ldots, c_n$, quantifing variables existentially as soon as possible (that is, a variable is quantified as soon as it does not occur anymore in the unprocesses clauses). We refer to this as *early quantification* of variables. Note that different clause orders may induce different orders of variable quantification. Finding a good clause order is a major focus of this chapter.

This motivates the following change in the earlier BDD-based satisfiability-solving algorithm [SV01]: after constructing the BDD B_i, we existentially quantify variables that do not occur in the clauses $c_{i+1}, \ldots, c_m$. In this case we say the quantifier $\exists x$ has been *eliminated*. The computational advantage of quantifier elimination stems from the fact that reducing the size of the support set of a BDD typically (though not necessarily) results in a reduction of its size; that is, the size of $(\exists x)B$ is typically smaller than that of B. In a nutshell, this method, which we describe as *symbolic quantifier elimination*, eliminates all quantifiers until we are left with the constant BDD 1 or 0. Symbolic quantifier elimination was first applied to SAT solving in [Gro96] (under the name of *hiding functions*) and tried on random 3-SAT instances. The work in [SV01] studied this method further, and considered various optimizations. The main interest there, however, is in the behavior of the method on random 3-SAT instances, rather in its comparison to search-based methods based on the DPLL algorithm.[1]

So far we processed the clauses of the input formula in a linear fashion. Since the main point of quantifier elimination is to eliminate variables as early as possible,

[1]Note that symbolic quantifier elimination provides *pure* satisfiability solving; the algorithm returns 0 or 1. To find a satisfying truth assignment when the formula is satisfiable, the technique of self-reducibility can be used, cf. [Bal90].

reordering the clauses may enable us to do more aggressive quantification. That is, instead of processing the clauses in the order $c_1, \dots, c_m$, we can apply a permutation π and process the clauses in the order $c_{\pi(1)}, \dots, c_{\pi(m)}$. The permutation π should be chosen so as to minimize the number of variables in the support sets of the intermediates BDDs. This observation was first made in the context of symbolic model checking, cf. [BCL91, BGP$^+$97, GB94, HKB96]. Unfortunately, finding an optimal permutation π is by itself a difficult optimization problem, motivating heuristic approaches.

A particular heuristic that was proposed in the context of symbolic model checking is that of *clustering* [RAB$^+$95]. In this approach, the clauses are not processed one at a time, but the clauses are first partitioned into several clusters. For each cluster C we first apply conjunction to all the BDDs of the clauses in the C to obtain a BDD B_C. The clusters are then combined, together with quantifier elimination, as described earlier. Heuristics are required both for clustering the clauses and ordering the clusters. Bouquet [Bou99] proposed the following heuristic (the focus there is on enumerating prime implicants). Consider some order of the variables. Let the *rank* (from 1 to n) of a variable x be $rank(x)$, let the rank $rank(\ell)$ of a literal ℓ be the rank of its underlying variable, and let the rank $rank(c)$ of a clause c be the maximum rank of its literals. The clusters are the equivalence classes on clauses of equal rank. The rank of a cluster is the rank of its clauses. The clusters are then ordered according to increasing rank. For example, given the set of clauses $\{x_1 \vee \neg x_2, x_1 \vee x_3, \neg x_2 \vee x_3, x_3 \vee x_4\}$ with the propositions ordered by their subscript, the clusters are $C_1 = \{\}, C_2 = \{x_1 \vee \neg x_2\}, C_3 = \{x_1 \vee x_3, \neg x_2 \vee x_3\}, C_4 = \{x_3 \vee x_4\}$.

Satisfiability solving using symbolic quantifier elimination is a combination of clustering and early quantification. We keep a set of *active* variables as we conjoin clusters, in the order $C_1, \dots, C_n$. Starting from an empty set, after each cluster C_i is processed, we add all the variables that occur in C_i to the active set. Then, a variable that does not occur in all C_js where $j > i$ can be removed from the active set and eliminated via early quantification. So, we are computing $\exists X_n \dots (\exists X_2(((\exists X_1)C_1) \wedge$

$C_2) \ldots \wedge C_n)$, where the quantified variable set X_i consists of the active variables that can be quantified early after C_i is processes. (The BDD package we are using (CUDD) allows quantifying several variables in function call.) When we use Bouquet's clustering, the method is referred to as *Bouquet's Method*, which we abbreviate here is as BM. For the example above, the BM quantification schedule is $\exists x_3 x_4((\exists x_1 x_2((C_1 \wedge C_2) \wedge C_3)) \wedge C_4)$.

We still have to chose a variable order. An order that is often used in constraint satisfaction [Dec03] is the "maximum cardinality search" (MCS) order [TY84], which is based on the graph-theoretic structure of the formula. We use the interaction graph for the formula as defined in Chapter 2.4. MCS ranks the vertices from 1 to n in the following way: as the next vertex to rank, select the vertex adjacent to the largest number of previously ranked vertices (ties can be broken in various ways). The variable order used for the BDDs in the comparisons unless otherwise mentioned is the inverse of the MCS order[2].

4.2 Experimental setup

We compare symbolic quantifier elimination to search (with the DPLL-based solver ZChaff [ZM02a]) across a variety of classes of formulas. Unlike the standard practice of comparing solver performance on benchmark suites [LS03], our focus here is not on simple time comparison, but rather on *scalability*. That is, we focus on scalable classes of formulas and evaluate how performance *scales* with formula size. We are interested in seeing which method scales better, i.e., polynomial vs. exponential scalability, or different degrees of exponential or polynomial scalability. Our test suite includes both random and nonrandom formulas (for random formulas we took 60 samples per case and reported median time, since median is less sensitive to outliers than mean).

[2]Using the MCS order or its inverse as the BDD variable order exhibits little performance difference, so the inverse is preferred because the BE approach presented in Section 4.4.1 is easier to implement on the inverse order.

Experiments were performed using x86 emulation on the Rice Terascale Cluster[3], which is a large Linux cluster of Itanium II processors with 4GB of memory each.

Our test suite includes the following classes of formulas:

- Random 3-CNF: We chose uniformly k 3-clauses over n variables. The *density* of an instance is defined as k/n. We generate instances at densities 1.5, 6, 10, and 15, with up to 200 variables, to allow comparison for both under-constrained and over-constrained cases. (It is known that the satisfiability threshold of such formulas is around 4.25 [SML96]).

- Random affine 3-CNF: Affine 3-CNF formulas belongs to a polynomial class as classified by Schaefer [Sch78]. Here, they are generated in the same way as random 3-CNF formulas, except that the constraints are not 3-clauses, but parity equations in the form of $l_1 \oplus l_2 \oplus l_3 = 1$, where $\oplus$ is the exclusive-or operator[4]. Each constraint is then converted into four clauses $l_1 \vee l_2 \vee l_3, \neg l_1 \vee \neg l_2 \vee l_3, \neg l_1 \vee l_2 \vee \neg l_3, l_1 \vee \neg l_2 \vee \neg l_3$, yielding CNF formulas. The satisfiability threshold of such formula is found empirically to be around density (number of equations divided by number of variables) 0.95. We generate instances of density 0.5 and 1.5, with up to 400 variables.

- Random biconditionals: Biconditional formulas, also known as Urquhart formulas, form a class of affine formulas that have resolution proofs that are proven to have an exponential lower bound on size. A biconditional formula has the form $l_1 \leftrightarrow (l_2 \leftrightarrow (\ldots (l_{k-1} \leftrightarrow l_k) \ldots))$, where each l_i is a positive literal. Such a formula is valid if either all variables occur an even number of times or all variables occur an odd number of times [Urq95]. Their negation, in turn, is unsatisfiable and have exponential lower bound on the size of resolution proofs.

[3] `http://www.citi.rice.edu/rtc/`

[4] This is equivalent to just choosing three variables and generate $x_1 \oplus x_2 \oplus x_3 = p$ where $p = 0$ or $p = 1$ with equal probability.

We generate such unsatisfiable formulas with up to 100 variables, where each variable occurs 3 times on average.

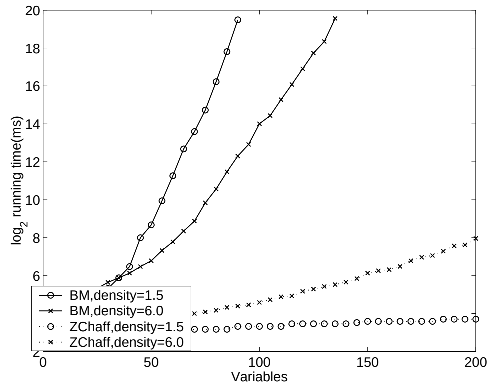
- Random chains: The classes described so far all have an essentially uniform random interaction graph, with no underlying structure. To extend our comparison to structured formulas, we generate random chains [DR94]. In a random chain, we form a long chain of random 3-CNF formulas, called *subtheories*. (The chain structure is reminiscent to the structure typically seen in satisfiability instances obtained from bounded model checking [BAC+99] and planning [KS92].) We use a similar generation parameters as in [DR94], where there are 5 variables per sub-theory and 5-23 clauses per sub-theory, but that we generate instances with a much bigger number of sub-theories, scaling up to > 20000 variables and > 4000 sub-theories.

- Nonrandom formulas: As in [US94], we considered a variety of formulas with very specific scalable structure:
  - The n-Rooks problem (satisfiable).
  - The n-Queens problem (satisfiable for $n > 3$).
  - The pigeon-hole problem with $n + 1$ pigeons and n holes (unsatisfiable).
  - The mutilated-checkerboard problem, where an $n \times n$ board with two diagonal corner tiles removed is to be tiled with 1×2 tiles (unsatisfiable).

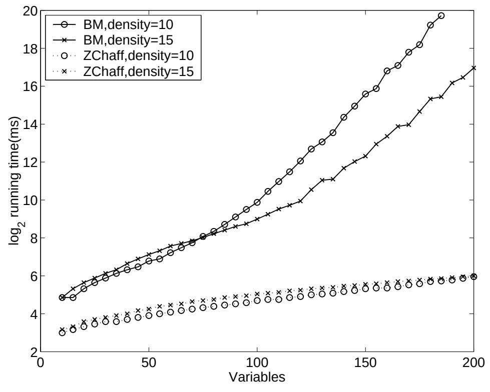
4.3 Search vs. symbolic results

Our goal in this section is to address the viability of symbolic quantifier elimination. To this end we compare the performance of BM against ZChaff[5], a leading DPLL-based solver across the classes of formulas described above, with a focus on scalability. For now, we use the MCS variable order.

[5]ZChaff version 2004.5.13



A.



B.

Figure 4.1 : Random 3-CNF

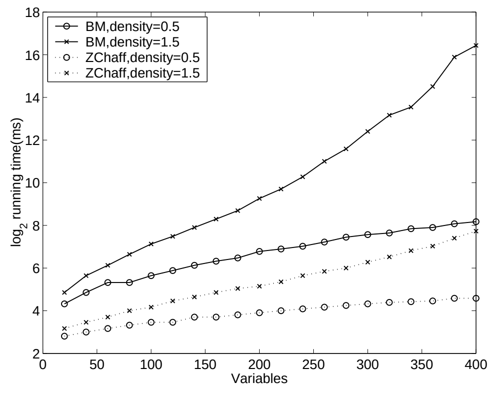
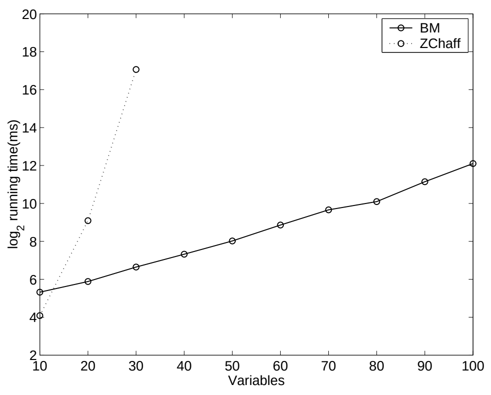
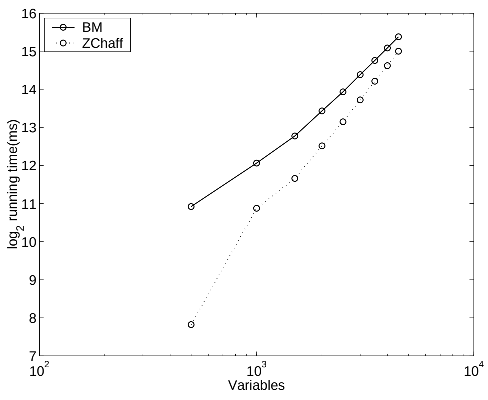


Figure 4.2 : Random 3-Affine

In Figure 4.1A and 4.1B, we can see that BM is not very competitive for random 3-CNF formulas. At density 1.5, ZChaff scales polynomially, while BM scales exponentially. At density 6.0 and at higher densities, both methods scale exponentially, but ZChaff scales exponentially better. (Note that above density 6.0 both methods scale better as the density increases. This is consistent with the experimental results in [CDS$^+$03] and [SV01].) A similar pattern emerges for random affine formulas, see Figure 4.2. Again, ZChaff scales exponentially better than BM. (Note that both methods scale exponentially at the higher density, while it is known that affine satisfiability can be determined in polytime using Gaussian elimination [Sch78].)



A.



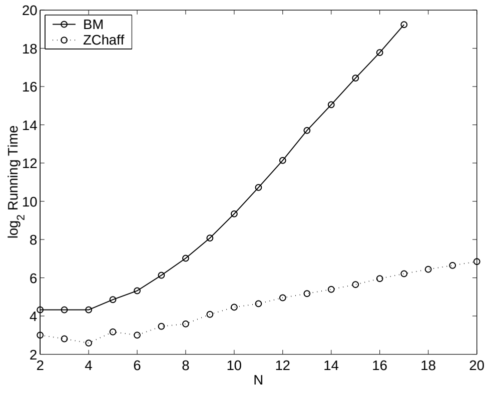
B.

Figure 4.3 : A) Random Biconditionals B) Random Chains

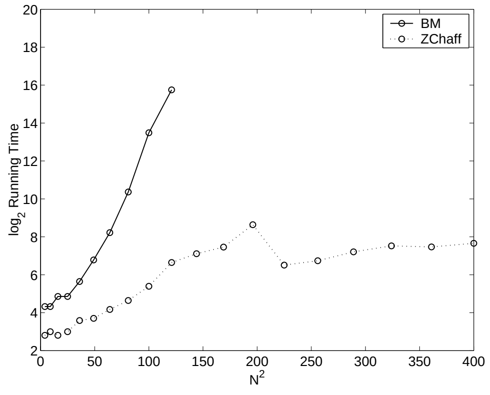
The picture changes for biconditional formulas, as shown in Figure 4.3A. Again, both methods are exponential, but BM scales exponentially better than ZChaff. (This result is consistent with the finding in [CS00], which compares search-based methods to ZDD-based multi-resolution.)

For random chains, see Figure 4.3B, which uses a log-log scale. Both methods scale polynomially on random chains. (Because density for the most difficult problems change as the size of the chains scales, we selected here the hardest density for each problem size.) Here BM scales polynomially better than ZChaff. Note that for smaller instances ZChaff outperforms BM, but the performance gap closes near the end, which justifies our focus on scalability rather than on straightforward benchmarking.

Finally, we compare BM with ZChaff on the non-random formulas of [US94]. The n-Rooks problem is a simpler version of n-Queens problem, where the diagonal constraints are not used. For n-Rooks, the results are as in Figure 4.4A. This problem has the property of being *globally consistent*, i.e., any consistent partial solution can be extended to a solution [Dec03]. Thus, the problem is trivial for search-based solvers, as no backtracking is need. In contrast BM scales exponentially on this problem. This shows that quantification alone does not eliminate all the overhead of enumeration, even when it is only able to perform pure satisfiability solving. For

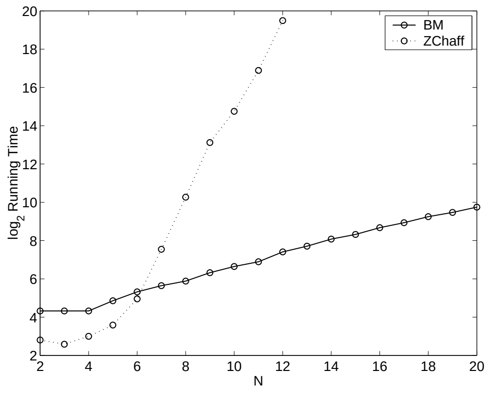


A.

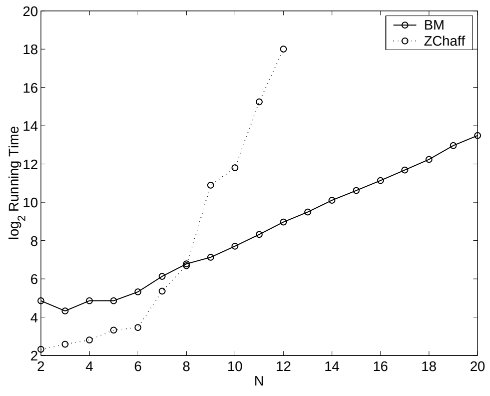


B.

Figure 4.4 : A) n-Rooks B) n-Queens



A.



B.

Figure 4.5 : A) Pigeon Hole B) Mutilated Checkerboard

n-Queens, see Figure 4.4B, BM scale exponentially in n^2, while ZChaff seems to have better scalability. Again, this should be contrasted against the results for the pigeon-hole problem and the mutilated-checkerboard problem, see Figure 4.5A and Figure 4.5B. On both problems both BM and ZChaff scale exponentially, but BM scales exponentially better than ZChaff.

4.4 Optimizing the BDD-based approach

BM is only one approach to symbolic quantifier elimination. There are, however, many choices one needs to make to guide an implementation. The order of variables is used both to guide clustering and to perform quantifier elimination, as well as to order the variables in the underlying BDDs. Both clustering and cluster processing can be performed in several ways. In this section, we investigate the impact of choices in clustering, representation, variable order, and quantifier elimination in the implementation of symbolic algorithms. Our focus here is on measuring the impact of variable order on BDD-based SAT solving; thus, the running time for variable ordering, which is polynomial for all algorithms, is not counted in our figures.

4.4.1 Cluster Ordering

As argued earlier, the purpose of quantifier elimination is to reduce support-set size of intermediate BDDs. What is the best reduction one can hope for? This question has been studied in the context of constraint satisfaction. It turns out that the optimal schedule of conjunctions and quantifier eliminations reduces the support-set size to one plus the *treewidth* of the interaction graph of the input formula [DKV02]. Computing the treewidth of a graph is known to be NP-hard, which is why heuristic approaches are employed [KBv01]. It turns out that by processing clusters in a different order we can attain the optimal support-set size. Recall that BM processes the clusters in order of increasing ranks. *Bucket elimination* (BE), on the other hand, processes clusters in order of decreasing ranks [DP87]. Maximal support-size set of BE with

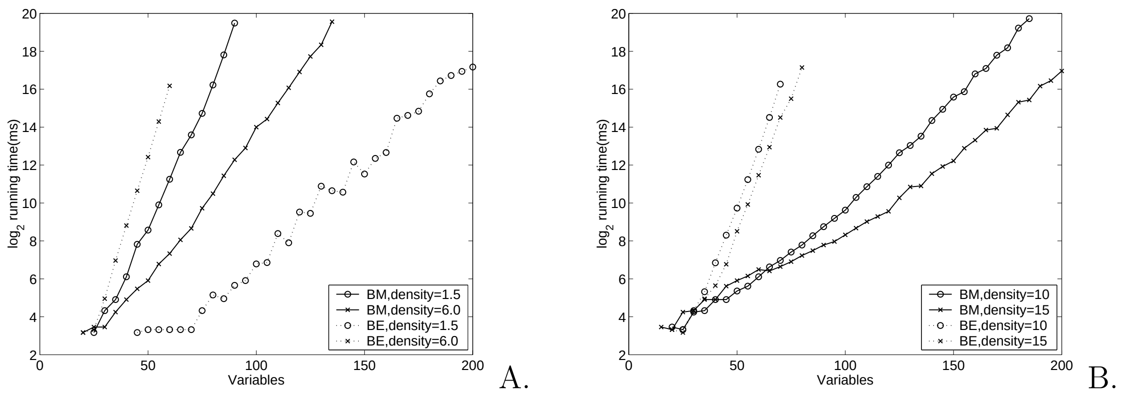


Figure 4.6 : Clustering Algorithms - Random 3-CNF

respect to optimal variable order is defined as the *induced width* of the input instance, and the induced width is known to be equal to the treewidth [DP87, Fre]. Thus, BE with respect to optimal variable order is guaranteed to have polynomial running time for input instances of logarithmic treewidth, since this guarantees a polynomial upper bound on BDD size. For BE, since the maximum-ranked variable in each cluster can not occur in any lower-ranked clusters, computing a quantification schedule from the contents of the clusters is not necessary. As each cluster is processed, the maximum-ranked variable is eliminated. For example, for the formula presented in Section 4.1, the quantification schedule would be $\exists x_1((\exists x_2((\exists x_3((\exists x_4 C_4) \wedge C_3)) \wedge C_2)) \wedge C_1)$, with one variable eliminated per cluster processed. As shown, using the inverse of variable rank as the BDD variable order allows us to always eliminate the top variable in the BDD.

We now compare BM and BE with respect to variable order built through the MCS heuristic (MCS is the preferred variable order also for BE).

The results for the comparison on random 3-CNF formulas is plotted in Figure 4.6A and 4.6B. We see that the difference between BM and BE is density dependent, where BE excels in the low-density case, which have low treewidth, and BM excels in the high-density cases, which has high treewidth. A similar density

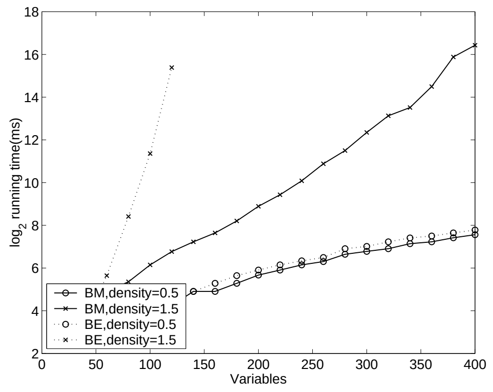
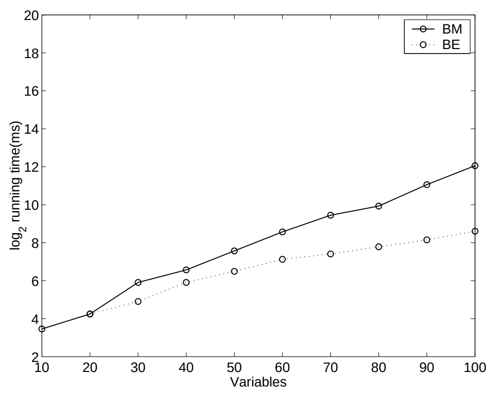
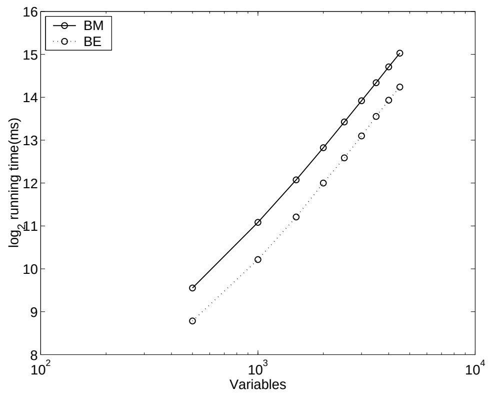


Figure 4.7 : Clustering Algorithms - Random Affine

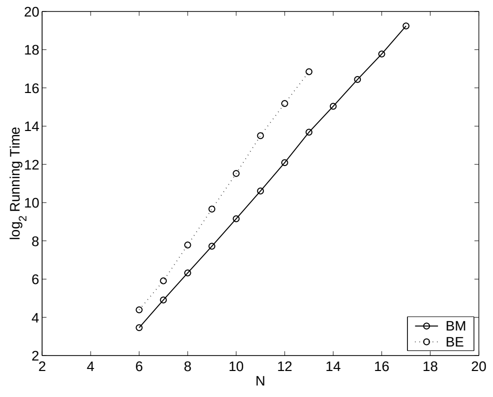


A.

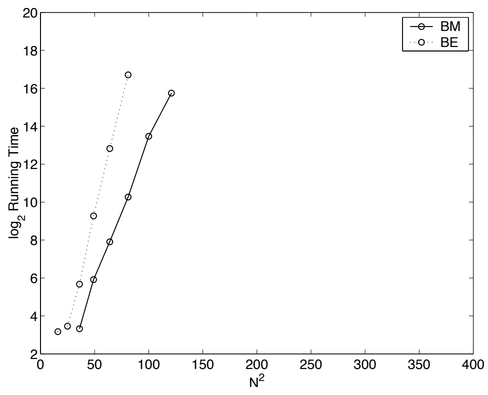


B.

Figure 4.8 : Clustering Algorithms - A) Random Biconditionals B) Random Chains



A.



B.

Figure 4.9 : Clustering Algorithms - A) n-Rooks B) n-Queens

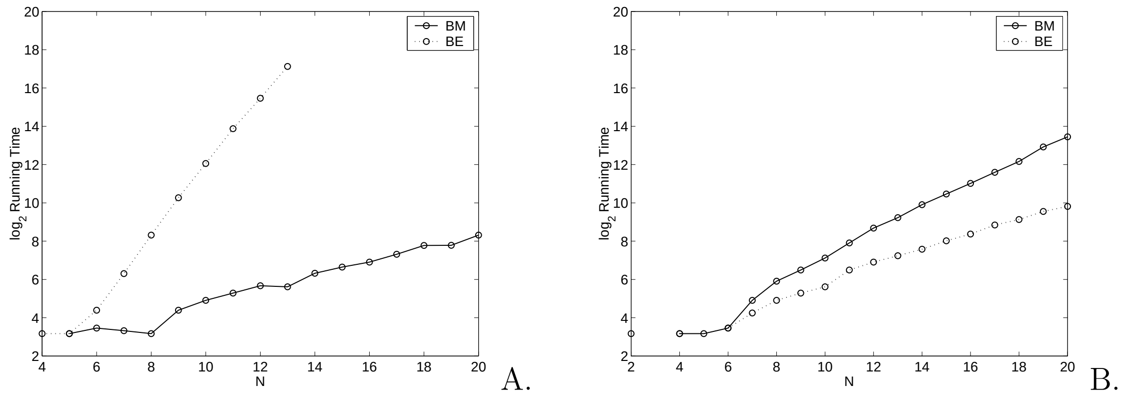
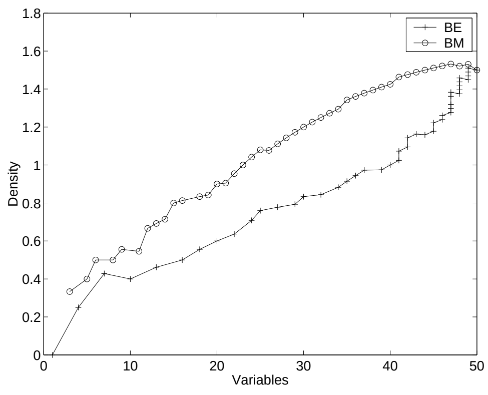


Figure 4.10 : Clustering Algorithms - A) Pigeon Hole B) Mutilated Checkerboard

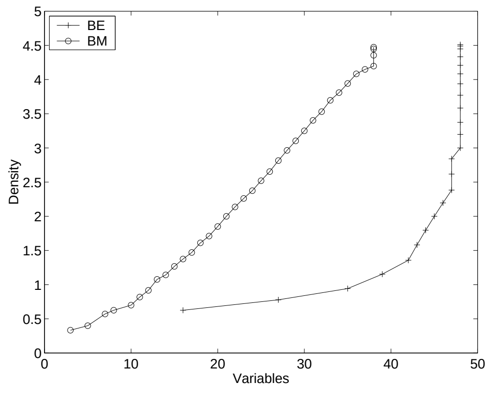
dependent behavior is shown for the affine case in Figure 4.7. The difference of the two schemes on biconditional formulas is quite small as shown in Figure 4.8A. For chains, see Figure 4.8B. Because the number of variables for these formulas are large, the cost of computing the quantification schedule gives BE an edge over BM.

On most constructed formulas, the picture is similar to the high-density random cases, where BM dominates, except for mutilated-checkerboard formulas, where BE has a slight edge. (Note that treewidth for mutilated checkerboard problems grows only at $O(n)$ compared to $O(n^2)$ for other constructed problems.) We plot the performance comparison for n-rook formulas in Figure 4.9A, n-queens formulas in Figure 4.9B, pigeon-hole formulas in Figure 4.10A, and mutilated checkerboard problems in Figure 4.10B.

To understand the difference in performance between BM and BE, we study their effect on intermediate BDD size. BDD size for a random 3-CNF instance depends crucially on both the number of variables and the density of the instance. Thus, we compare the effect of BM and BE in terms of these measures for the intermediate BDDs. We apply BM and BE to random 3-CNF formulas with 50 variables and densities 1.5 and 6.0. We then plot the density vs. the number of variables for the intermediate BDDs generated by the two cluster-processing schemes. The results are



A.



B.

Figure 4.11 : Clustering Algorithms A) Density=1.5 B) Density=6.0

plotted in in Figure 4.11A and Figure 4.11B. Each plotted point corresponds to an intermediate BDD, which reflects the clusters processed so far.

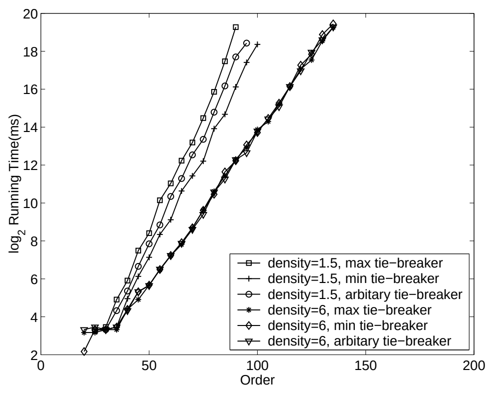
As can be noted from the figures, BM increases the density of intermediate results much faster than BE. This difference is quite dramatic for high-density formulas. The relation between density of random 3-CNF instance and BDD size has been studied in [CDS$^+$03], where it is shown that BDD size peaks at around density 2.0, and is lowest when the density is close to 0 or the satisfiability threshold. This enables us to offer a possible explanation to the superiority of BE for low-density instances and the superiority of BM for high-density instances. For formulas of density 1.5, the density of intermediate results is smaller than 2.0 and BM's increased density results in larger BDDs. For formulas of density 6.0, BM crosses the threshold density 2.0 using a smaller number of variables, and then BM's increased density results in smaller BDDs.

The greater range of applicability of BM over BE suggests that minimizing support-set size ought not to be the dominant concern. BDD size is correlated with, but not dependent on, support-set size. More work is required in order to understand the good performance of BM. Our explanation argues that, as in [AM01], BM deals first with the most constrained subproblems, therefore reducing BDD-size of intermediate results. While the performance of BE can be understood in terms of treewidth, we still lack, however, a fundamental theory to explain the performance of BM.

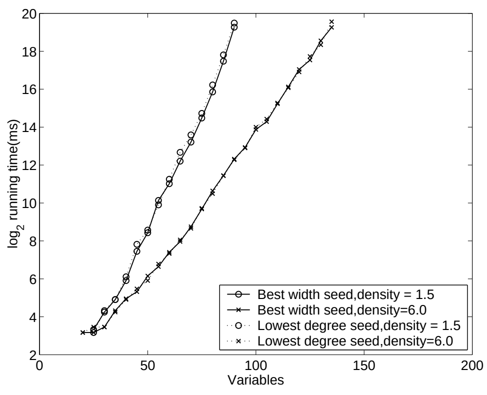
4.4.2 Variable Ordering

In this section, we study the effects of the variable order on the performance of symbolic algorithms. We only present results for BM; the relative performance between the different variable orders are the same for BE. The variable order for the BDD representation is again the inverse of the variable order for clustering. As mentioned earlier, when selecting variables, MCS has to break ties, which happens quite often. One can break ties by choosing (from those variables that have the maximum cardinality to ranked variables as MCS requires) the variable with minimal degree to unselected variables [SV01] or the variable with the maximal degree to unselected variables [BB94]. (Another choice to break ties uniformly at random, but this choice is expensive to implement, since it is difficult to choose an element uniformly at random from a heap.) We compare these two heuristics with an arbitrary tie-breaking heuristic, in which we simply select the top variable in the heap. The results are shown in Figure 4.12A for random 3-CNF formulas. For high density formulas, tie breaking made no significant difference, but least-degree tie breaking is markedly better for the low density formulas. This seems to be applicable across a variety of class of formulas and even for different orders and algorithms.

MCS typically has many choices for the lowest-rank variable. Koster et. al. [KBv01] recommended starting from every vertex in the graph and choosing the variable order that leads to the lowest treewidth. This is easily done for instances of small size, i.e. random 3-CNF or affine problems; but for structured problems, which could be much larger, the overhead is too expensive. Since min-degree tie-breaking worked quite well, we used the same idea for initial variable choice. In Figure 4.12B, we see that our assumption is well-founded, that is, the benefit of choosing the best initial variable compared to choosing a min-degree variable is negligible. For larger



A.



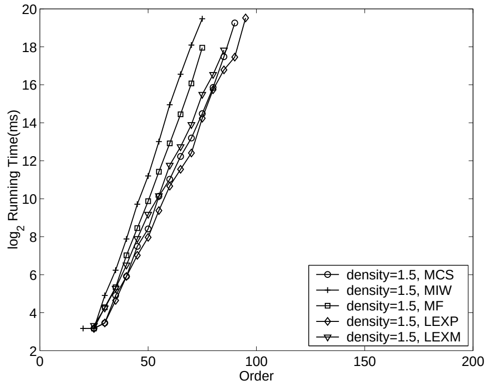
B.

Figure 4.12 : A) Variable Ordering Tie-breakers B) Initial Variable Choice

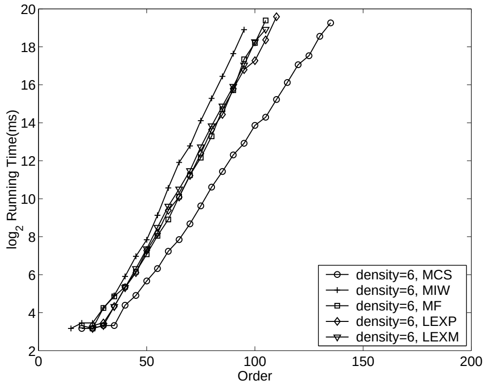
problems like the chains or the bigger constructed problems, the additional overhead of trying every initial variable would be prohibitive, so we used the low-degree seed in all cases.

Algorithms for BDD variable ordering in the model-checking systems are often based on circuit structures, for example, some form of circuit traversal [FFK88, MWBSV88] or graph evaluation [CHP93]. These techniques are not applicable here, since the formulas are provided in CNF and the original circuit structure is lost.

MCS is just one possible vertex-ordering heuristics. Other heuristics have been studied in the context of treewidth approximation. Koster et. al. [KBv01] studied two other vertex-ordering heuristics that are based on local search: LEXP and LEXM. Both LEXP and LEXM are based on *lexicographic breadth-first search*, where candidate variables are lexicographically ordered with a set of labels, and the labels are either the set of already chosen neighbors (LEXP), or the set of already chosen vertices reachable through lower-ordered vertices (LEXM). Both algorithms try to generate vertex orders where a triangulation would add a small amount of edges, thus reducing treewidth. In [Dec03], Dechter also studied heuristics like Min-Induced-Width (MIW) or Min-Fill (MF), which are greedy, non-local heuristics based on choosing the vertex that have the least number of induced neighbors (MIW) or the vertex that would add

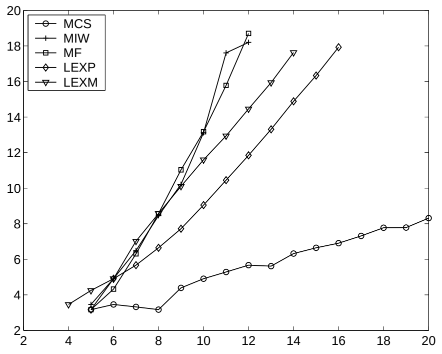


A.

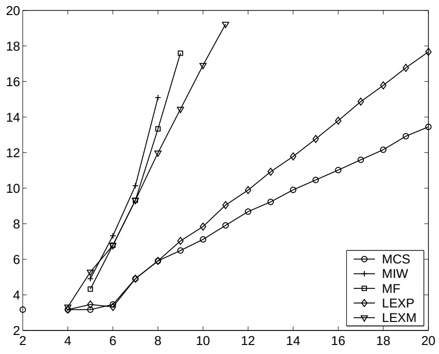


B.

Figure 4.13 : Vertex Order Heuristics - Random 3-CNF A) Density=1.5 B) Density=6



A.

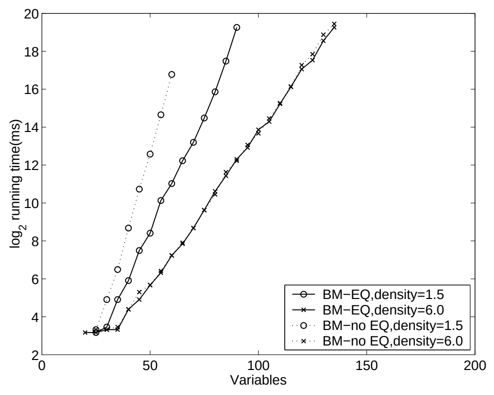


B.

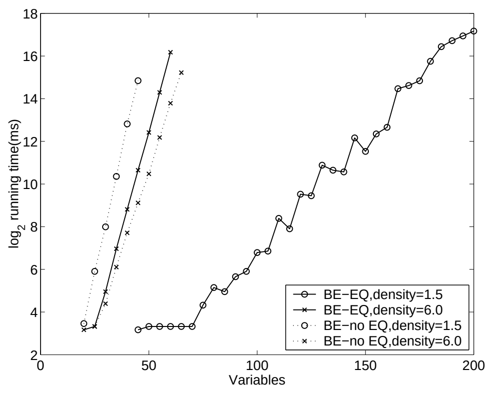
Figure 4.14 : Vertex Order Heuristics - A) Pigeon Hole B) Mutilated Checkerboard

the least number of induced edges (MF).

In Figure 4.13A and 4.13B, we compare variable orders constructed from MCS, LEXP, LEXM, MIW, and MF for random 3-CNF formulas. For high-density cases, MCS is clearly superior. For low-density formulas, LEXP has a small edge, although the difference is quite minimal. Across the other problem classes (for example, pigeon-hole formulas as in Figure 4.14A and mutilated checkerboard as in Figure 4.14B), MCS uniformly appears to be the best order, being the most consistent and generally the top performer. Interestingly, while other heuristics like MF often yield better treewidth, MCS still yields better runtime performance. This indicates that



A.



B.

Figure 4.15 : Quantifier Elimination-Random 3-CNF

minimizing treewidth need not be the dominant concern; the dominant concern is minimizing BDD size. BDD size are more closely related to *pathwidth* instead of treewidth, and local search heuristics, like MCS, produces tree decompositions that are more path-like than greedy heuristics like MF.

4.4.3 Quantifier Elimination

So far we argued that quantifier elimination is the key to the performance of the symbolic approach. In general, reducing support-set size does result in smaller BDDs. It is known, however, that quantifier elimination may incur non-negligible overhead and may not always reduce BDD size [Bry86]. To understand the role of quantifier elimination in the symbolic approach, we reimplemented BM and BE without quantifier elimination. Thus, we do construct a BDD that represent all satisfying truth assignments, but we do that according to the clustering and cluster processing order of BM and BE.

In Figure 4.15A and 4.15B, we plotted the running time of both BM and BE, with and without quantifier elimination on random 3-CNF formulas. We see that there is a trade-off between the cost and benefit of quantifier elimination. For low-density instances, where there are many solutions, the improvement from quantifier

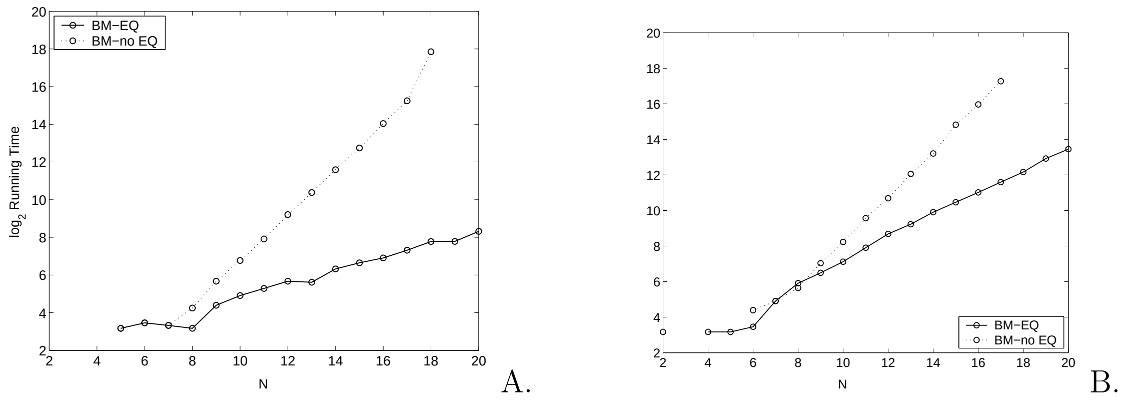


Figure 4.16 : Quantifier Elimination - A) Pigeon Hole B) Mutilated Checkerboard

elimination is clear, but for high-density instances, quantifier elimination results in no improvement (while not reducing BDD size). For BE, where the overhead of quantifier elimination is lower, quantifier elimination improves performance very significantly at low density, although at high density there is a slight slow down. On the other hand, quantifier elimination is important for the constructed formulas, for example, for the pigeon-hole formulas in Figure 4.16A and the mutilated checkerboard formulas in Figure 4.16B.

4.5 Comparison with other approaches

In the previous section, we conducted a comprehensive comparison of the impact of different parameters on the BDD-based symbolic approach. Next, we expand our focus to alternate approaches, first by comparing the BDD-based symbolic quantifier elimination with ZDD-based multi-resolution, then compare the structural variable order we used with the default dynamic variable order in the context of ZChaff.

4.5.1 BDDs vs. ZDDs

So far we used symbolically represented sets of truth assignments. An alternate approach is to use decision diagrams to represent sets of clauses instead of sets of assignments. ZRes [CS00] is a symbolic implementation of the directional resolution algorithm in [DP60, DR94]. The approach is also referred to as *multi-resolution*, since the algorithm carries out all resolutions over a variable in one symbolic step. Since individual clauses are usually sparse with respect to the set of variables, ZRes [CS00] used ZDDs [Min96], which typically offer a higher compression ratio then BDDs for the sparse spaces. Each propositional literal ℓ is represented by a ZDD variable v_ℓ (thus a propositional variable can be represented by two ZDD variables), and clause sets are represented as follows:

- The empty clause ϵ is represented by the terminal node 1.
- The empty set $\emptyset$ is represented by the terminal node 0.
- Given a set C of clauses and a literal ℓ whose ZDD variable v_ℓ is lowest in a given variable order, we split C into two subsets: $C_\ell = \{c \mid c \in C, \ell \in c\}$ and $C' = C - C_\ell$. Given ZDDs representing $C'' = \{c \mid c \vee \ell \in C_\ell\}$ and C', a ZDD representing C would be rooted at v_ℓ and have ZDDs for C'' and C' as its left and right children.

This representation is the dual of using ZDDs to represent Irredundant Sum of Products (ISOPs) of Boolean functions [Min96].

We use two set operations on sets of clauses: (1) $\times$ is the crossproduct operator, where for two clause sets C and D, $C \times D = \{c \mid \exists c' \in C, \exists c'' \in D, c = c' \cup c''\}$, and (2) $+$ is subsumption-free union, so if both C and D are subsumption free, and $c \in C + D$, then there is no $c' \in C + D$ where $c' \subset c$. Multi-resolution is implemented using $\times$ on cofactors: given a ZDD f, f_{x^+} (resp. f_{x^-}) is the ZDDs corresponding to the positive cofactor on the ZDD variable v_x (resp., $v_{\neg x}$, so $f_{x^+} = \{a \mid a \vee x \in f\}$ and $f_{x^-} = \{a \mid a \vee \neg x \in f\}$. Now $f_{x^+} \times f_{x^-}$ (after removing tautologies) represents the set of all resolvents of f on x, which has to be combined using $+$ with $f_{x'}$, which is the ZDD for the clauses not containing x. ZRes eliminates variables using multi-resolution one by one until either the empty clause is generated, in which case

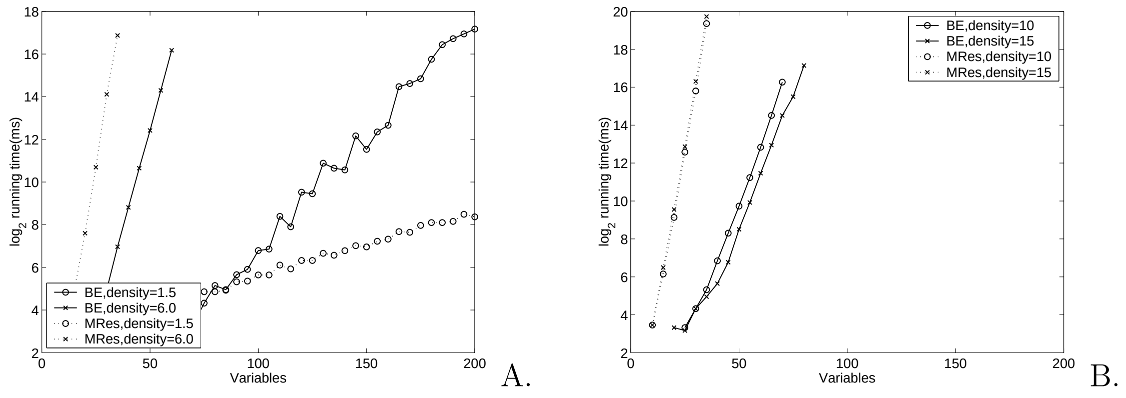


Figure 4.17 : Random 3-CNF

the formula is unsatisfiable, or all variables have been eliminated, in which case the formula is satisfiable.

To facilitate a fair comparison between ZRes and our BDD-based solver, we used the multi-resolution code used in [CS00] under our bucket-elimination framework and used the same variable and elimination order as the BDD-based algorithms. This can be seen as a comparison of the compression capability of ZDD-based clause sets versus BDD-based solutions sets representations, since at comparable stages of the two algorithms (say, before variable x_i is eliminated), the data structures represents the same Boolean function. As an optimization, a simple form of unit preference is implemented for the ZDD-based multi-resolution, since unit clauses can be easily detected in the ZDD-based clause set representation and resolved out-of-order.

The results for the 3-CNF and affine satisfiability cases are plotted in Figures 4.17A, 4.17B, and 4.18. We see that the differences between the two approaches are again density dependent. Just like the differences between BE and BM, ZDD-based multi-resolution is more efficient at low density and less efficient at high density. This can be related to the compression ratio achieved by the two representations at different densities, where the clause set representation is far more efficient at low densities. For the high-density case, the clause set representation starts to show its

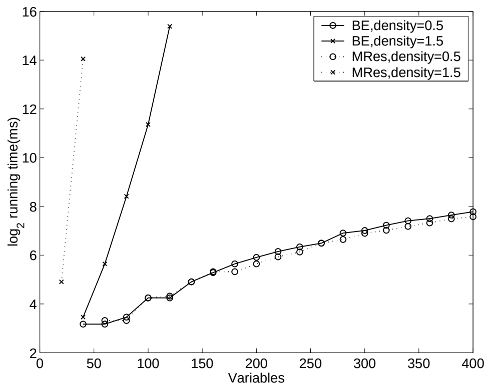


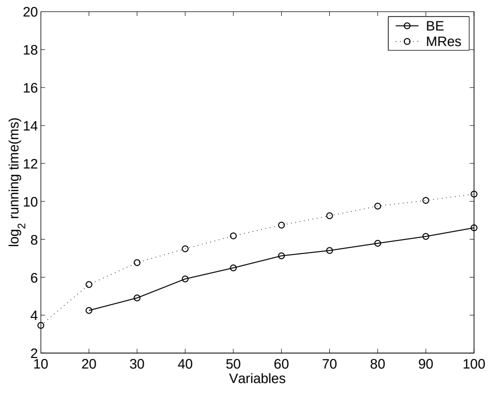
Figure 4.18 : Random 3-Affine

shortcomings. High-density problems typically have a large number of clauses and few solutions, clause-set representation is less efficient in this case. This is especially evident for the unsatisfiable case, where if BDDs are used, unsatisfiability can be detected immediately, but if clause sets are used, detection is delayed until an empty clause is generated.

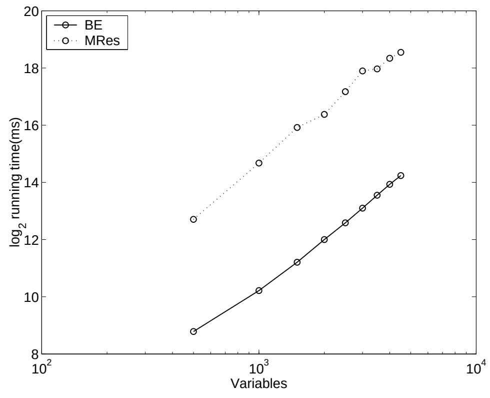
Next we examine the other classes of formulas in Figure 4.19A, 4.19B, 4.20A, 4.20B, 4.21A, and 4.21B. In all cases, the BDD-based approach is superior to the ZDD-based approach.[6]

An explanation for the superiority of the BDD-based approach can be provided in terms of the cost of the quantifier-elimination operation. Complexity of decision-diagram algorithms can be measured in the number of cache look-ups that the algorithm performs. Quantifying out a single variable uses the BDD "or" operation, which has a proven $O(n^2)$ upper bound on the number of cache look-ups [Bry86]. The same cannot be said for the ZDD multi-resolution operation used to quantify out a single variable, where the number of cache look-ups can be exponential in the width

[6]There exists other ZDD-based approaches for hard-for-resolution problems, for example, CASSAT [MM02a], which exhibits polynomial running time on pigeon-hole formulas [MM02b]. A comparison against these approaches would be a future direction of this research.

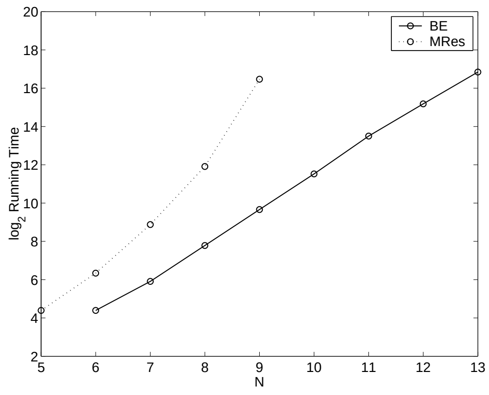


A.

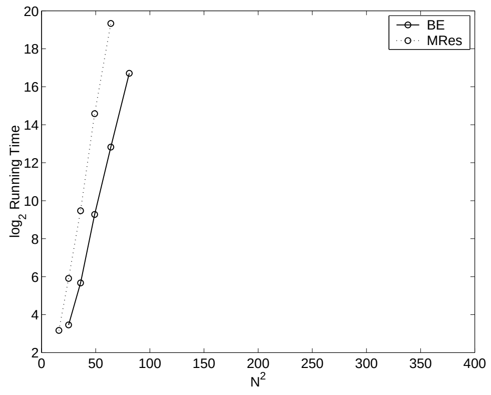


B.

Figure 4.19 : A) Random Biconditionals B) Random Chains

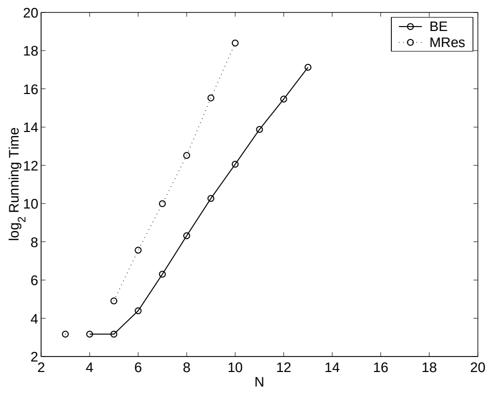


A.

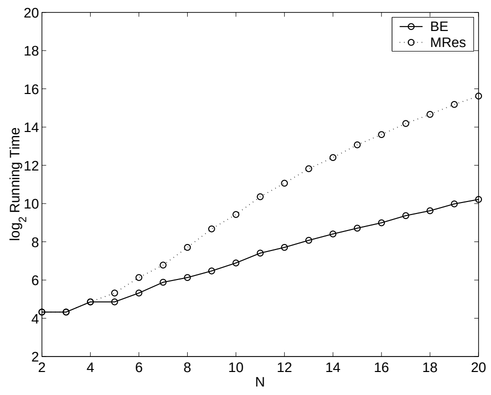


B.

Figure 4.20 : A) n-Rooks B) n-Queens



A.



B.

Figure 4.21 : A) Pigeon Hole B) Mutilated Checkerboard

of the input ZDDs. Empirically, the number of cache lookups can be 1-2 orders of magnitude larger than the size of the output ZDD. This is the main contribution to the performance hit taken by the ZDD-based algorithm.

Still, the ZDD-based approach should not be entirely discredited. Since the ZDD-based approach is superior for problems that are heavily under-constrained, we will show in Chapter 7 that is matches naturally with quantified satisfiability solving, whose matrix is under-constrained. Otherwise they would be easily unsatisfiable due to the universal quantifiers. The extra compression of the ZDD-based clause-set representation would apply, explaining the superiority of the ZDD-based approach.

4.5.2 Structure-Guided Variable Order for Search

In Section 4.4, we showed that the choice of variable order is important to the performance of BDD-based satisfiability solvers. We showed that MCS variable order offers good algorithmic performance across a variety of input formulas. In contrast, most search-based algorithm use a dynamic variable order, based on the clauses visited or generated during the search procedure, for example, the VSIDS heuristic used in ZChaff [MMZ+01]. To offer a more direct comparison between search-based and symbolic methods, we re-implemented ZChaff with the MCS variable order and compared its performance with ZChaff and with the symbolic solvers. (See [AMS01, HD03] for earlier work on structure-guided variable order for search-based methods.) We compared here the performance of ZChaff with the default (VSIDS) variable order, ZChaff with MCS variable order, and the BDD-based solvers (for each formula class we chose the best solver between BM and BE).

The results for random formulas are shown in Figures 4.22A, 4.22B, 4.23A, 4.23B, 4.24A, 4.24B, 4.25A, and 4.25B, and the results for constructed formulas are shown in Figures 4.26A, 4.26B, 4.27A, and 4.27B. In general, the structure-guided variable order is inferior in terms of performance to dynamic variable order (VSIDS). For easy problems, the overhead of pre-computing the variable order is quite significant.

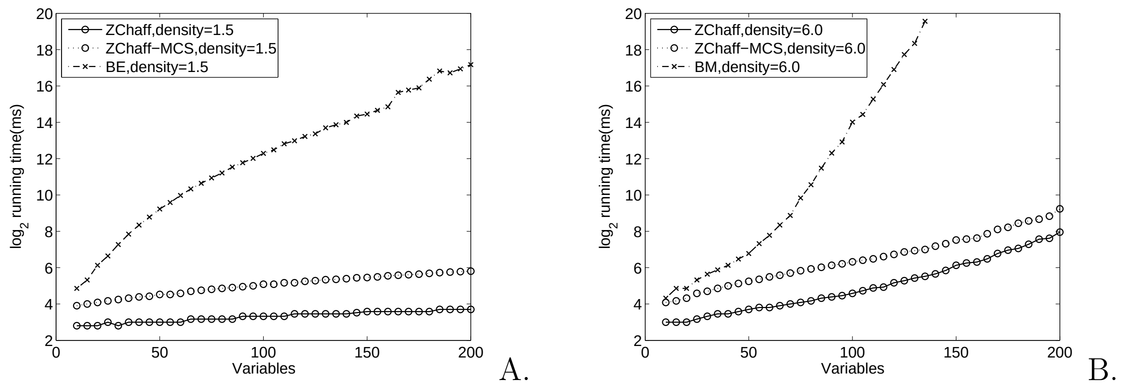


Figure 4.22 : Variable Order - Random 3-CNF (1)

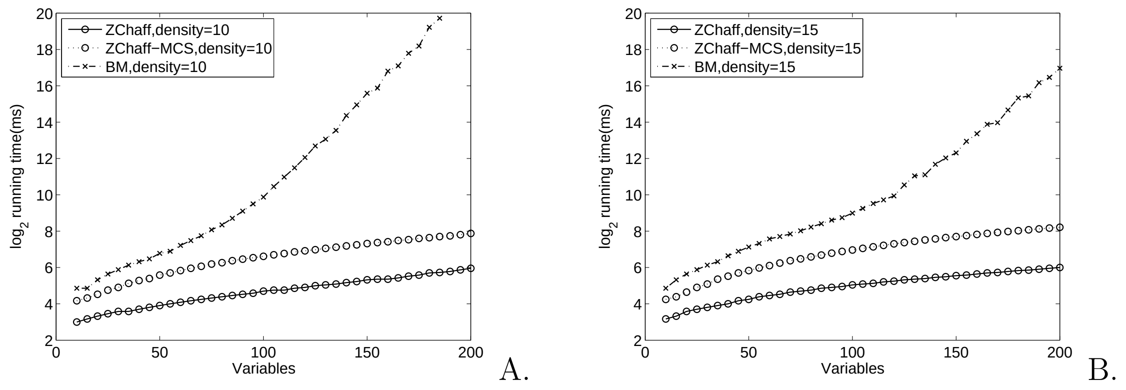


Figure 4.23 : Variable Order - Random 3-CNF (2)

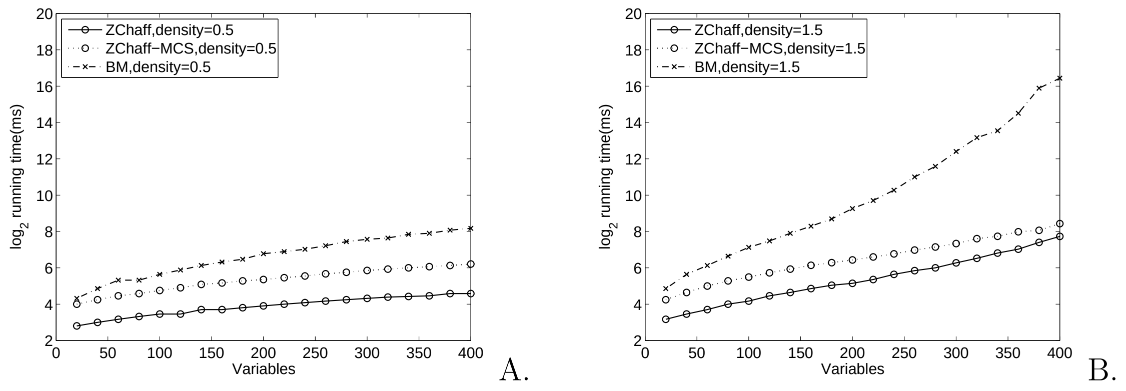
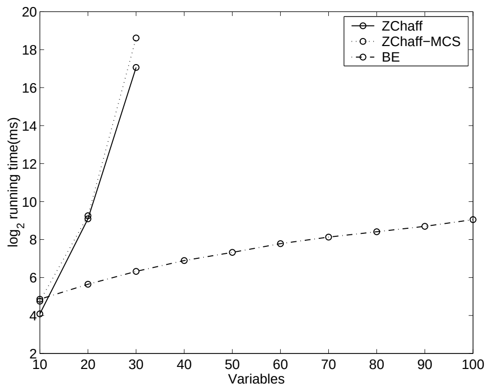
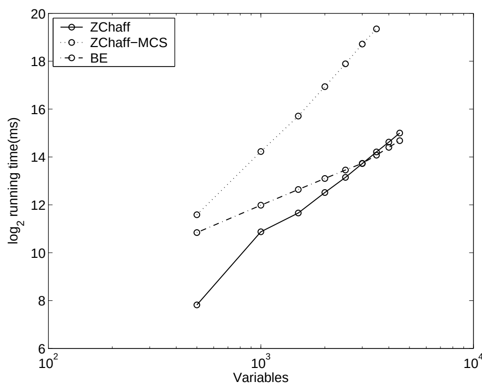


Figure 4.24 : Variable Order - Random Affine

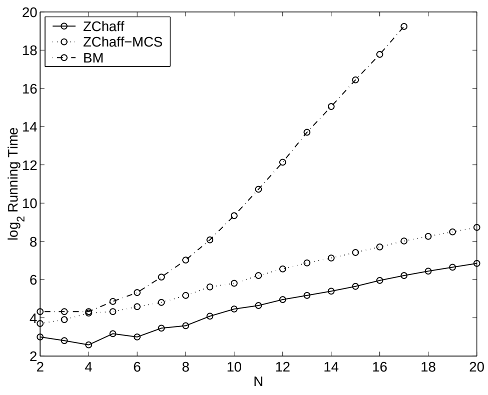


A.

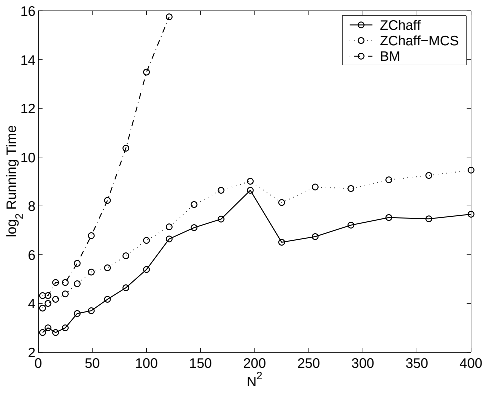


B.

Figure 4.25 : Variable Order - A) Random Biconditional B) Random Chains

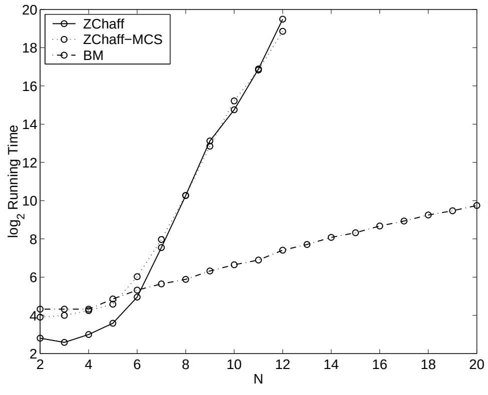


A.

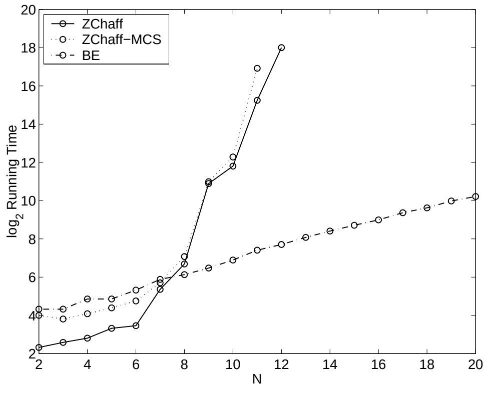


B.

Figure 4.26 : Variable Order - A) n-Rooks B) n-Queens



A.



B.

Figure 4.27 : Variable Order - A) Pigeon Hole B) Mutilated Checkerboard

The performance loss should not be entirely attributed to the overhead though, since we also observed an increase in the number of implications performed. Thus, dynamic variable order is, in general, a better algorithmic choice.[7] Nevertheless, for most formulas, there is no exponential gap in scaling between the two variable-order heuristics.

Also, replacing VSIDS by MCS did not change the relationship between ZChaff and the BDD-based solvers. The difference in performance between search-based and symbolic approaches is larger than the difference between static and dynamic decision order for ZChaff. In none of the cases did the static variable order change the relative picture between search and symbolic approaches. This shows the general superiority of search-based vs. symbolic techniques cannot be attributed to the use of dynamic variable order.

4.6 Summary

In this chapter, we investigated a large number of possible optimizations to the BDD-based symbolic approach to satisfiability, we did not manage to develop a symbolic solver that is competitive with search on a large range of problem classes. Still, as we will see later in this dissertation, this is more of a mismatch between the tool (BDD is naturally a solution enumerator) and the problem (propositional logic satisfiability solving), and not a fundamental lacking of BDD-based approaches. Similar heuristics as studied in this chapter will be successfully applied quantified satisfiability later. Some of the heuristics, for example the use of ZDDs and bucket elimination, while poor for propositional satisfiability, fares much better for quantified satisfiability. In the rest of the dissertation, we will investigate the impact of quantification on the complexity of the problems and the performance of the approaches.

[7]Except for formulas with very specific structure, for example, Figure 4.25B.

Chapter 5

Parametrized Complexity of Bounded-Width QBF

In Chapter 3, we developed BDD size bounds for both bounded-width CNF formulas and bounded-width existentially-quantified CNF formulas. The size bounds are, in turn, lower bounds for the time needed to construct BDDs for such formulas. When using pathwidth as the measure of width, the quantified version of the problem is superficially similar to the unquantified case; both can be built in space that is $f(w)\mathsf{poly}(n)$ where n is the number of variables and w is the width. But when we look more carefully on the $f(w)$ for each case, we can see the actual impact of the width on practical performance would be very different. For the unquantified case, $f(w)$ is singly exponential, while for the quantified case, $f(w)$ is doubly exponential. We would like to know whether such an impact applies not just to a single implementation, but instead, to all algorithms. In other words, whether the use of quantifiers make bounded width problems harder. We would like to draw some insights from the classical complexity case. The polynomial hierarchy is characterized by using alternating quantifiers to stratify the complexity between P and PSPACE. There are no formal non-collapsing results for the polynomial hierarchy, and most people believe the hierarchy is strict. And in turn, increasing the alternation depth do, in some way, make a problem "harder" to solve. In the parameterized complexity setting, we want to know whether there is a analogous hierarchy that scales with the alternation depth while the width is bounded with a parameter. In other words, whether increasing the alternation depth of a problem while keeping all other parameters the same makes it harder to solve. To show non-collapsing results for such a *fixed-parameter* hierarchy, we must not be able to trade complexity between $f(w)$ part and the $poly(n)$ part.

In other words, if some problem is hard for $f(w)h(n)$ with some polynomial h, we should not be able to solve it by using some larger polynomial h' in time $f'(w)h'(n)$ where f' is smaller than f. Unfortunately, since our hardness results would need to come from the reduction of a classically intractable problem in NP, and there are no known proofs for either P≠NP, we would need to base our hierarchy on the assumption P≠NP, as under the case where P=NP, we cannot prevent the problem being solved (with a classical algorithm) in time $h(n)$ with a polynomial h.

An alternative to building such a hierarchy despite the P vs. NP difficulty is based on Frick and Grohe's technique of *telescoping* [FG02], where they generated intractable word model-checking problems with a very small monadic second-order formulas. In order to understand telescoping, it is useful to compare the case of monadic second-order logic model checking on words with other second-order fragments and other model types. For most syntactical restrictions of second order logic under an unrestricted vocabulary (usually represented as labelled graphs or hypergraphs), model checking even with a fixed formula size is still intractable in a classical sense. For different second-order fragments, the study of the classes of structures they can capture, and in turn, the study of the complexity classes of their model checking problem with a fixed formula size formed the area of *Descriptive Complexity.* For example, existential second-order logic on graph models is NP-complete for model checking [Fag74]. Model checking for the general case of second-order logic in turn characterizes the *polynomial hierarchy* [Imm99]. With a graph model, second-order logic allows one to construct through alternation (for the case of ESO, guess) an accepting run of a Turing machine and check its correctness, thus, its complexity is complete for the same complexity class as the type of the Turing machine. A run for a Turing machine is the concatenation of a sequence of instantaneous descriptions (state, symbols on tape, and head position) of the Turing machine, each representing a time step in the run. Each position on the tape is a *cell.* There are, however, two key relations between cells in a run of the Turing machine. The first is the neighbor relation on the tape i.e., the two cells are neighboring cells in the same time step. The second is the time successor relation, i.e., the two cells are the same position on the tape and represents successive time steps. Checking the correctness of a run need to use both of these relations. Of course, if the vocabulary of the model contains "user-definable" binary relations, then we can represent both relations explicitly. But, for many second-order logic fragments, the power of second order quantifiers allows us to restrict the model to a very restrictive case, that of words, where the only binary relation defined on the model is the successor relation in the word. The idea used is that of *embedding*, which is a mapping from arbitrary models to word models. It is far from inconceivable that we can write any model into a word. For example, a graph can be written into a word as the concatenation of a list of vertices followed by a list of edges (written as pairs of vertices). All vertices, of course, would need to have a distinct string representation in the word in order to distinguish between them. The difficulty would be to recover the original structure when needed, say, to check whether there is an edge between two vertices. When we have quantifiers on binary predicates, this is easily done, where [EGG] showed that ESO on words remains NP-complete when restricted to a single binary quantifier by using simple binary-encoded tags. In general, such restriction in frame types are possible because we can use a formula which characterizes the relation that we are implicitly encoding using tags, i.e., the formula is true iff the free variable assignments satisfies the relation we are defining (i.e., the formula *interprets* the relation [Leu02]). A formula (characterizing some property) on the arbitrary model can be translated to a formula on a word model where all occurrences of the relation are substituted with the interpreting formula wherever the relation occurred (Some quantifier relativization is also needed to preserve the semantics of monadic quantifiers).

When we restrict ourselves to monadic second-order logics on word models, using a fixed formula to interpret relations no longer work. The main reason of this is the tractability gap, since a fixed MSO formula can only define regular languages on words. [FG02] performs a limited kind of interpretation, where the domain size of the graph-like models are held under a bound. Telescoping is in turn a technique to maximize the size of model that can be encoded (or conversely, reduce the size of the formula needed to interpret) by exploiting the relation between satisfiability and model checking. For second-order logics, there is a very distinct difference in complexity of satisfiability and finite model checking. For monadic second-order logic on finite words, satisfiability is non-elementary, while model checking is in PSPACE. This necessitates that some formula have non-elementarily large models. While the idea of embedding graphs into word models are the same, the design of tags used in telescoping is much more elaborate. By tapping the power of alternation, the edge relation on the graph can be interpreted using a formula in MSO with $O(\log^* n)$ size.

In this dissertation, we improve upon the approach in [FG02] and apply it to the parameterized complexity of bounded-width QBF. The main difference is that our result is *alternation-tight*. Tightness in measured as the relation between the upper bound and the lower bound for function relating the width and alternation depth with the coefficient on running time. To get a tight characterization of the exact blowup, we use QPTL, a quantified temporal logic instead of using monadic second-order logic on words. QPTL has a tight bound on its classical complexity for satisfiability [SVW87]. We will show parametrically, under reasonable assumptions about intractability, there is a similar right bound on the parametric blowup. Also, we present a translation from QPTL model checking to QBF satisfiability in that is *parameter preserving*, where the width of the QBF is linearly related to the size of the QPTL formula to be model checked. The combination of the telescoping technique and translation allows us to characterize the parameterized complexity of QBF problems of bounded width and alternation, culminating in a hierarchy of fixed-parameter tractable classes. In this chapter and the following chapter, the constructions we use allow us to bound the pathwidth of the QBF formula. In turn, all lower bounds can be trivially applied to bounded-treewidth case. In contrast, the upper bound for bounded width QBF developed by Chen [Che04] is already based on treewidth.

5.1 Background

5.1.1 QPTL

Definition 5.1. *Given a set P of propositions, the set of formulas in QPTL is the smallest set satisfying the following:*

- *Every proposition in P is a formula in QPTL.*
- *If φ and ψ are formulas in QPTL, then $(\varphi \wedge \psi)$ and $(\neg\varphi)$ are formulas in QPTL. $(\varphi \vee \psi)$ is shorthand for $(\neg((\neg\varphi) \wedge (\neg\psi)))$, and $(\varphi \rightarrow \psi)$ is shorthand for $(\neg(\varphi \vee \psi))^1$.*
- *If φ is a formula in QPTL, then $X\varphi$ and $F\varphi$ are formulas in QPTL. We use X^n as shorthand for a sequence of n X operators. $G\varphi$ is shorthand for $\neg F \neg\varphi$.*
- *If φ is a formula in QPTL, then for $p \in P$, $(\exists p)\varphi$ is a formula in QPTL. $(\forall p)\varphi$ is shorthand for $\neg(\exists p)\neg\varphi$.*

We interpret QPTL formulas on finite words. A finite word model π for QPTL is a word of length n over 2^P, that is, a mapping $\pi : [0 \ldots n-1] \rightarrow 2^P$, where n is the size of the model, and P is a set of propositions (which needs to contain the free propositions in the formula we are interpreting). We write $\pi[q \mapsto Q]$ to indicate the overiding of the meaning of a proposition q in the model by a subset Q of positions (i.e., $Q \subseteq [0 \ldots n-1]$), and $p \in \pi[q \mapsto Q]$ iff either $p \neq q$ and $p \in \pi(i)$, or $p = q$ and $i \in Q$. All propositional connectives have their standard meaning. Since we are considering semantics for finite-word models, the temporal operators X and F need to consider the finiteness of the model, and so do the quantifiers. We use $\pi, i \models \varphi$ to mean the word model π at position i satisfies the formula φ. (The standard notation

[1]We often omit parentheses, when the standard precedence order $\neg > \wedge > \vee > \rightarrow$ is used.

of $\pi \models \varphi$ now becomes a shorthand for $\pi, 0 \models \varphi$.) Of course, $\models$ is only defined when $0 \leq i < |\pi|$. The formal semantics of QPTL on a finite word model follows:

- $\pi, i \models p$ iff $p \in \pi(i)$,
- $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$,
- $\pi, i \models \varphi \wedge \psi$ iff $\pi, i \models \varphi$ and $\pi, i \models \psi$,
- $\pi, i \models X\varphi$ iff $i < |\pi| - 1$ and $\pi, i+1 \models \varphi$,
- $\pi, i \models F\varphi$ iff there exists some j where $i \leq j < |\pi|$, and $\pi, j \models \varphi$,
- $\pi, i \models (\exists q)\varphi$ iff there is a set $Q \subseteq [0 \ldots |\pi| - 1]$ of positions, such that $\pi[q \mapsto Q], i \models \varphi$.

Each QPTL formula φ defines a language which is the set of all πs where $\pi, 0 \models \varphi$.

Definition 5.2. *The finite model-checking problem for QPTL is defined as: given a model π and a formula φ, where π is defined on a vocabulary that includes the free variables of φ, does $\pi \models \varphi$ hold?*

Here, we only consider QPTL formulas in prenex normal form, where a formula can be written as $(Q_1 p_1)(Q_2 p_2) \ldots (Q_k p_k)\varphi$, each Q_i is $\exists$ or $\forall$, and φ is quantifier free. The logic can be stratified into bounded-alternation layers as follows:

1. The set $\Sigma_0^{QPTL} = \Pi_0^{QPTL}$ includes the quantifier-free formulas in QPTL.
2. If φ is in Σ_k^{QPTL} or Π_{k-1}^{QPTL}, then $(\exists p)\varphi$ is in Σ_k^{QPTL}.
3. If φ is in Π_k^{QPTL} or Σ_{k-1}^{QPTL}, then $(\forall p)\varphi$ is in Π_k^{QPTL}.

The sets Σ_k^{QPTL} and Π_k^{QPTL} split QPTL into fragments where Σ_k^{QPTL} (Π_k^{QPTL}) represents the existential (universal) formulas with *alternation depth k*.

While all the QPTL formulas constructed in this dissertation should be in prenex normal form, we do, for simplicity of presentation, apply quantifiers to sub-formulas and later compose them with other sub-formulas. Still, we never put quantified sub-formulas under temporal operators, which allows all formulas we build to be convertable to prenex normal form with a simple renaming and lifting of quantifiers without increase in either size or alternation depth.

5.1.2 Tiling Systems

Definition 5.3. *A* tiling system *is a tuple $T = (D, H, V)$, where D is a finite set of* tiles types*, $H \subseteq D \times D$ and $V \subseteq D \times D$ are, respectively, the* horizontal *and* vertical *adjacency relations. A* tiling problem *is the tuple (T, w, h, I), where w and h are natural numbers, and I is an* initial constraint *(see below). The tiling problem (T, w, h, I) has a solution iff there exists a mapping $F : [1 \dots w] \times [1 \dots h] \rightarrow D$, where the following requirements are satisfied:*

- *For all $1 \leq i < w$ and $1 \leq j \leq h$, $(F(i,j), F(i+1,j)) \in H$, and for all $1 \leq i \leq w$ and $1 \leq j < h$, $(F(i,j), F(i,j+1)) \in V$.*
- *F satisfies the initial constraint I. The nature of the constraint I depends on the particular variant of tiling problem used.*

Each position (i, j), where $i \in [1..w]$, $j \in [1..h]$, is called a cell *with row i and column j.*

The tiling problem is useful in representing a number of natural complexity classes. The most well known of which is the case where $h = w$ (*square tiling*), w is written in unary (note the impact on the size of the input), and the initial condition I is empty. This version has been shown to be NP-complete by Lewis [Lew78]. Here, we consider a fixed tiling system T. This is possible because tiling systems encode runs of Turing machines, so a T that corresponds to a universal Turing machine can be used. Still, to maintain NP-completeness where we use a fixed T, we need to use a different initial condition, namely, the first row needs to be fixed to a particular sequence of tiles I given as part of the input, where $F(1, j) = I(j)$ for $1 \leq j \leq w$

(which encodes the initial tape of the Turing machine). This allows polynomial-sized computations of Turing machines to be encoded with the tiling problem.

Theorem 5.4. *[Lew78] There is a tiling system T (corresponding to universal Turing machines) such that The tiling-problem (T, w, h, I), where $h = w$, and I defines the first row of the tiling, is NP-complete.*

In the following, we concern ourselves only with this NP-complete variant, which we denote as (T, w, I). Given a tiling F, we name the corresponding solution checking problem, in absence of the initial condition, as the *tiling check* problem, where we check whether F satisfies the horizontal and vertical relations. We also assume we can separate the tile set into two parts D' and D'', where D' does not appear on the first row, and D'' only appears on the first row. Only tiles in D'' can appear in I. In the usual manner of exploiting non-determinism, once we have a formula that can perform tiling check, we can use existential quantifiers to guess the non-initial part of the tiling in order to solve the tiling problem. Since $w = |I|$, and T is fixed, the size of the tiling problem is defined as $|I|$, which is the same as the size of w in unary.

5.1.3 Non-elementary growth and some properties of the tower function $g(k, n)$

In the following, we use the terms polynomial, sub-polynomial, exponential, sub-exponential, etc. to classify functions. A function $f(\cdot)$ is *polynomially bounded* (usually just called polynomial) if there is some constant $k > 0$ such that $f(n)$ is in $O(n^k)$. Here, we write $f(\cdot)$ is in $\mathsf{poly}(\cdot)$. A function is *sub-polynomial* if for all constant $k > 0$, we have $f(n)$ in $o(n^k)$. Notice here all sub-polynomial functions are also considered to be polynomial, i.e., the set $\mathsf{poly}(\cdot)$ contains more functions than those that are strictly polynomial (i.e., polynomial, but not sub-polynomial). A function $f(\cdot)$ is *sub-exponential* if for all strictly polynomial functions $h(\cdot)$, $h(n)$ is in $o(2^{h(n)})$. Here, we denote the set of sub-exponential functions as $\mathsf{subexp}(\cdot)$. Note that our definition of sub-exponential function is more restrictive than the usual definition (where a function $f(\cdot)$ is sub-exponential if for every $c > 1$, $f(x)$ is $o(c^x)$).

In this section, we work with the growth speed of the non-elementary tower function $g(\cdot, \cdot)$ defined as: $g(0, n) = n$, $g(k+1, n) = 2^{g(k,n)}$. The inverse of g is the $\log^k$ (repeated log) function: $\log^1(n) = \log n$, $\log^{k+1}(n) = \log(\log^k(n))^2$. Finally, the $\log^*$ (iterated log) function is defined by: $k = \log^*(n)$ iff k is the least natural number such that $g(k, 1) \geq n$ (or, analogously, the least k such that $1 \geq \log^k n$).

The function $g(k, n)$ and $\log^k(n)$ are defineable as inverses of each other. In other words, $g(k, \log^k(n)) = n$. In this chapter, much of our results depend on compressing some aspect of the input through the $\log^k$ function, which is later re-expended using the g function. So, we would like to study the impact of the compression-re-expansion loop. Unfortunately, because of the extreme blowup of the function g, extremely small changes in the second parameter of g causes extremely big changes in the output. Consider the function $h_{f,k}(n) = g(k, f(\log^k(n))$ where k is an integer and f is a function:

Lemma 5.5. *Given a function $f(m)$, if there exists constants $c < 1$ and m_f such that for all $m > m_f$, $f(m) \leq c\dot{m}$, then for all $k \geq 2$, $h_{f,k}(n)$ is sub-polynomial in n.*

Proof. When $k = 2$, consider all $n > n_f = g(2, m_f)$, $h_{f,2}(n) \leq g(2, c\log^2(n)) = 2^{\log(n)^c}$. Assume we have a strictly polynomial function $p(n)$ such that for all large enough n, $h_{f,2}(n) \geq p(n)$. We know there exists c' and a constant $n_{c'}$ where for all $n > n_{c'}$, $p(n) \geq n^{c'} = 2^{c' \log(n)}$. But we know for any $c < 1$, there exists $n_{c,c'}$ such that for all $n > n_{c,c'}$, $\log(n)^c < c' \log(n)$. So for all $n > \max(n_f, n_{c'}, n_{c,c'})$, $h_{f,2}(n) < p(n)$, contradicting our assumption. Thus, $h_{f,k}(n)$ is sub-polynomial in n.

For $k > 2$, assume inductively this lemma is proved for $k - 1$. Now $h_{f,k}(n) = 2^{h_{f,k-1}(\log(n))}$. From the induction hypothesis, we know that $h_{f,k-1}$ is sub-polynomial, so we can take $p(n) = n^{1/2}$ and we know for large enough n, $p(n) > h_{f,k-1}(n)$. But

[2]All log functions we use in this dissertation use base 2.

here, $2^{(\log(n))^{1/2}} = h_{f',2}(n)$ where $f'(m) = m/2$. So $h_{f,k}(n) \leq h_{f',2}(n)$ where $h_{f',2}(n)$ is sub-polynomial from the base case of the proof. In turn, $h_{f,k}(n)$ is sub-polynomial.

One direct consequence is that such a $2^{h_{f,k}}(n)$ is sub-exponential, which we use later.

5.2 Complexity of finite model-checking problem for QPTL

In this section, we study the complexity of the finite model-checking problem for QPTL. While we start with classical complexity results that are essentially "re-hashing" of well-known techniques, the development of the parameterized complexity results requires careful constructions and leads to the first tight characterization of a hierarchy inside fixed-parameter tractability.

5.2.1 Upper Bounds

Sistla, Vardi, and Wolper [SVW87] studied the satisfiability problem of QPTL, where a given QPTL formula φ, determines whether there exists a model π such that $\pi \models \varphi$. While their analysis of QPTL is over infinite words, the automata-theoretic approach can be used to analyze QPTL also over finite words (in fact, automata on infinite words was developed as an extension to automata on finite words, cf. [Büc60]). The following lemma characterizes QPTL over finite words.

Lemma 5.6. *Given a QPTL formula φ in Σ_k^{QPTL}, there exists a non-deterministic finite automaton of size $g(k, |\varphi|)$ that accepts the same language as φ.*

Proof. We use the standard automata-theoretic technique [Büc60, SVW87, VW94], applied to nondeterminsitic automata on finite words. The induction is on the structure of the formula.

For the base case, where ψ is in Σ_1^{QPTL}, we build a finite automaton with size $g(1, |\psi|)$ as follows. For the quantifier-free part of ψ, which we call ψ', we can build a (non-deterministic) finite automaton of size $2^{|\psi'|} \leq g(1, |\psi|)$ that accepts the same language as ψ' [SVW87]. Note that transitions in the NFA are labeled with *assignments* on the variables. We can now eliminate the quantified variables of ψ from the labeling of the transitions; in other words, the quantified variables have been *projected out*. This does not increase the size of the automaton.

For the inductive case, given a formula ψ in Σ_k^{QPTL}, it is in the form $(\exists p_1)\ldots(\exists p_n)\neg\psi'$, where ψ' is a formula in Σ_{k-1}^{QPTL}. The induction hypothesis states that the language accepted by ψ' can be accepted by an automaton A' of size $g(k-1,|\psi'|)$. We build an automaton A that accepts the same language as ψ by complementing A' and projecting the result onto the free variables of ψ. Complementing A' requires an exponential blowup in the size of A', and projection can be applied with no additional size increase. Thus, the size of A is in $2^{g(k-1,|\psi'|)} \leq g(k,|\psi|)$.

One point to note about the construction used is that the automata transformations can be performed in time linear in their size, which we will use later. $\square$

Theorem 5.7. *(Analogous to Theorem 4.1,* [SVW87]*) The satisfiability problem for Σ_k^{QPTL}, where $k \geq 1$, is complete for $NSPACE(g(k-1,|\varphi|))$.*

Proof. The automaton in Lemma 5.6 can be built "on-the-fly", and emptiness can be checked in non-deterministic logspace in the size of the automaton. The lower bound can be proved as in [SVW87], as the proof there uses only finite prefixes of infinite words to describe Turing-machine computations. $\square$

In contrast to the satisfiability problem, under a finite semantics, the model-checking problem has elementary complexity.

Theorem 5.8. *The finite model-checking problem for QPTL is PSPACE-complete.*

Proof. The PSPACE upper-bound is a direct corollary of the PSPACE-completeness of monadic second-order logic (MSO) model checking on words [FG02], since the temporal operators of QPTL can be syntactically interpreted in MSO. PSPACE-hardness comes from the fact that the QBF decision problem is a special case of QPTL finite model-checking problem on a model with only one state. $\square$

Having established the classical complexity of QPTL finite model-checking problem, we investigate its parameterized complexity, where the size and alternation depth of the formula are taken as the parameters, and the size of the model is taken as the size of the problem.

Theorem 5.9. *The finite model-checking problem for a formula φ in Σ_k^{QPTL} on a finite word model of size n has time complexity $O(g(k, |\varphi|)n)$.*

Proof. Remember that all automaton transformations used in Lemma 5.6 can be performed in the same time bound as their corresponding space bound. As a result, the automaton that accepts the same language as φ have size $g(k, |\varphi|)$ and can be built in time $O(g(k, |\varphi|))$. Checking whether an NFA of size m accepts a word of length n can be performed in time $O(m \cdot n)$. Thus, model checking of QPTL can be performed in time $O(g(k, |\varphi|)n)$. □

5.2.2 Lower Bounds

For the lower bound, we proceed by connecting the complexity of QPTL model checking on finite word models to the NP-completeness of the tiling problem.

Theorem 5.10. *Assuming $P \neq NP$, there exists a constant $c > 0$ such that model checking for Σ_k^{QPTL} on models of size $|\pi|$, where $k > 2$, cannot be done in time $g(k-1, c|\varphi|)h(|\pi|)$ where $h(\cdot)$ is in $\mathsf{poly}(\cdot)$*[3].

The rest of this section is dedicated to the proof of Theorem 5.10. We start with an overview of the proof.

For a tiling problem (T, w, I), if we are given a tiling f satisfying I, we can reduce the tiling-check problem to QPTL model checking as follows: The tiling F is converted to a word F_w, and the tiling problem (T, w, I) is converted to a QPTL formula $\varphi'_{T,w}$.

[3]In the proof of Theorem 5.23, we need to refer to the actual value of this constant, and call it c_{QPTL}

Note that the initial condition I is not checked by the formula, but instead, encoded in the model. In the following, when our concern is the formula, we use (T, w) to denote the relevent part of the tiling problem. We want $F_w \models \varphi'_{T,w}$ iff F is a solution for (T, w). Writing F in word form requires concatenating the rows of F into a word of length w^2. We order the rows from w to 1 for technical reasons. To convert (T, w) to $\varphi'_{T,w}$, we need to state row and column constraints using small QPTL formulas. Once we reduced the tiling-check problem to the finite model-checking problem for QPTL, going to the tiling (decision) problem is just a small step, using existential quantifiers to state the existence of a tiling. We add an additional existential quantifier block on the outside of $\varphi'_{T,w}$ for all the propositions that actually represent non-initial tiles, and call the resulting formula $\varphi_{T,w}$. Since now the actual tiling is quantified out in the formula, the interpretation of the propositions that correspond to non-initial tiles in the word model no longer affect the decision procedure, so the word model can contain an arbitrary mapping from cells after the first row to non-initial tiles, and the mapping from cells in the first row to initial tiles is based on I. We call such a model $\pi_{w,I}$. Thus, $\pi_{w,I} \models \varphi_{T,w}$ iff (T, w, I) has a solution. Through developing a polynomial reduction, the particular model-checking problem would be NP-hard, while at the same time fixed-parameter tractable. Assuming $P \neq NP$, a lower-bound can be shown for the function in the parameter. The quality of the lower-bound would depend on how small we can make the formula to be.

We give here the proof for Theorem 5.10, under the following assumptions on the reduction. Later, we describe in detail each step in the construction of $\pi_{w,I}$ and $\varphi_{T,w}$ and show that the assumptions hold.

1. $\pi_{w,I} \models \varphi_{T,w}$ holds iff (T, w, I) have a solution.

2. The word $\pi_{w,I}$ has size polynomial in w.

3. The formula $\varphi_{T,w}$ is in Σ_k^{QPTL} and is small enough that $g(k-1, c|\varphi_{T,w}|)$ is in $\mathsf{poly}(w)$ for some $c > 0$.

4. The construction of $\pi_{w,I}$ and $\varphi_{T,w}$ from (T, w, I) can be performed in time polynomial in w.

Proof. Choose $h(\cdot)$ in $\mathsf{poly}(\cdot)$. Assume that we have a model-checking algorithm with time complexity $g(k-1, c|\varphi|)h(|\pi|)$ for Σ_k^{QPTL} with a constant $c > 0$ that is small enough for Assumption 3. We use this algorithm to give a polynomial algorithm for tiling, which by Theorem 5.4, is NP-complete. We are given I in the input, and we know $w = |I|$. Based on Assumptions 2 and 4, we can construct $\pi_{w,I}$ in time and space polynomial in w. Based on Assumption 3, we can construct a formula $\varphi_{T,w} \in \Sigma_k^{QPTL}$, where $g(k-1, c|\varphi_{T,w}|)$ is in $\mathsf{poly}(w)$, and by Assumption 4, the time taken for the construction is in $\mathsf{poly}(w)$. Then, by Assumption 1, the model-checking algorithm can be applied, which takes time $g(k-1, c|\varphi_{T,w}|)h(|\pi_{w,I}|)$, which by Assumptions 2 and 3, is in $\mathsf{poly}(w)$. So, if we have an algorithm for the finite model-checking problem that does better than the lower-bound, then there exists a polytime algorithm for an NP-complete problem. $\square$

There is a gap of a little higher than an exponential between our upper and lower bounds. A major contributor to this gap is that NP might require strict exponential time to solve. In other words, the assumption $P \neq NP$ is an exponential too strong. That exponential gap can in fact be closed by using a stronger assumption.

Corollary 5.11. *Assuming NP cannot be solved in sub-exponential time, (i.e., there does not exist a sub-polynomial function $f(n)$ where NP can be solved in time $2^{f(n)}$), then there exists a constant $c > 0$ such that model checking for Σ_k^{QPTL} on models of size $|\pi|$, where $k > 2$, cannot be done in time $g(k, c|\varphi|)h(|\pi|)$ where $h(\cdot)$ is in $\mathsf{poly}(\cdot)$.*

Proof. The proof is analogous to Theorem 5.10, except our bounds are one exponential looser. Choose $h(\cdot)$ in $\mathsf{poly}(\cdot)$. Assume that we have a model-checking algorithm with time complexity $g(k, c|\varphi|)h(|\pi|)$ for Σ_k^{QPTL} with a constant $c > 0$ that is as small as we need. (Obviously, $c < 1$ from our upper bound.) We use this algorithm to give a sub-exponential algorithm for tiling, which by Theorem 5.4, is NP-complete. We are given inputs w and I (where $w = |I|$). Based on Assumptions 2 and 4, we can construct $\pi_{w,I}$ in time and space polynomial in w. If we are proceeding as the last theorem above, based on Assumption 3, there is some c such that we can construct a formula $\varphi_{T,w} \in \Sigma_k^{QPTL}$, where $g(k-1, c|\varphi_{T,w}|)$ is in $\mathsf{poly}(w)$. But, in order to improve our bounds, we need to look at the proof of Lemma 5.21. We know that $|\varphi_{T,w}|$ is in $O(\log^{k-1}(w))$, so by Lemma 5.5, there exists some $c > 0$ such that $g(k-1, c|\varphi_{T,w}|)$ is sub-polynomial in w. In turn, $g(k, c|\varphi_{T,w}|)$ is sub-exponential in w by definition. By Assumption 4, the time taken for the construction is in $\mathsf{poly}(w)$. By Assumption 1, the model-checking algorithm can be applied, and it takes time $g(k, c|\varphi_{T,w}|)h(|\pi_{w,I}|)$. By all the above arguments, the time needed to solve tiling would be sub-exponential in w.

5.2.2.1 Constructing the Tagged Model $\pi_{w,I}$

In this section, we detail the construction of the model $\pi_{w,I}$. Naively, constructing the model shall not take much effort, since all we need is a row-by-row concatnation of the cells. Still, we need to keep in mind that we need to check certain properties on the resulting word. In particular, we need to check the tiles satisfy both the horizontal and vertical constraint. When we use a row-major concatnation, checking the horizontal constraint is not difficult; the neighboring tiles are adjacent to each other in the word. Checking the vertical constraint is a little more involved. Naively, we need to measure distances of exactly w on the word. On the surface, this look amendable to the use of “yardsticks” (Originally from [Sto74], and the QPTL version from [SVW87]). Still, when the distance w to be measured is not an exact exponential, attempting to bound the size of the yardstick formulas becomes impossible. Since we are performing model checking instead of satisfiability, we can build the yardstick into the model instead of entirely depending on the formula to measure distances. To that end, we add annotation in the form of tags marking the column index of each cell. Our technique is essentially a hybrid between the yardsticks of [SVW87] and the telescoping of [FG02]. The approach in [SVW87] used counters that are implicitly encoded using alternating yardstick formulas; in contrast, the approach in [FG02] used explicit tags[4]. The combination of the two approaches allows us to both measure arbitrary distances as well as keep the alternation depth low.

The tags we use are a form of annotated binary numbers, where each bit is annotated by its offset. This allows the use of smaller formulas to compare such binary numbers, since inspection that are more "local" would be sufficient. Since the annotated offset have to be themselves written as binary numbers, the annotation process is repeated to construct *nested* tags. First, we give a formal definition of the syntax and semantics of the tags we are using:

Definition 5.12. *A* tag *of level h is defined with the alphabet $\Sigma = \{0, 1, \langle 1\rangle, \langle /1\rangle, \ldots, \langle h\rangle, \langle /h\rangle\}$. We denote the set of tags of level h with parameter n by $\mathsf{tags}_h(n)$. Each tag represents a mapping that can be inductively defined as follows: A tag $t \in \mathsf{tags}_1(n)$ is a mapping $\{0 \ldots n-1\} \to \{0,1\}$. A tag $t \in \mathsf{tags}_{h+1}(n)$ is a mapping $\mathsf{tags}_h(n) \to \{0,1\}$. Tags are written as words on Σ. For $t \in \mathsf{tags}_1(n)$, t is a bracketed n-bit sequence, i.e., a word of form $\langle 1\rangle\{0|1\}^n\langle /1\rangle$, where $t(i)$ is the $(i+1)$-th bit after $\langle 1\rangle$. For $t \in \mathsf{tags}_{h+1}(n)$, t is a bracketed sequence of tags in $\mathsf{tags}_h(n)$ and a corresponding bit for each tag, i.e., $\langle h+1\rangle(t'\{0|1\})^*\langle /h+1\rangle$, where each t' is a tag in $\mathsf{tags}_h(n)$. Since t is a total function, every possible tag of level h appears exactly once in a tag of level $h+1$. Every tag $t' \in \mathsf{tags}_h(n)$ that appears inside t is called a* sub-tag *of t. For every sub-tag $t' \in \mathsf{tags}_h(n)$ in a tag $t \in \mathsf{tags}_{h+1}(n)$, the bit that appears directly after the t' is the mapped value of the t'. Equality on tags is defined as the (unordered) equality on the underlying mappings.*

For example, a tag in $\mathsf{tags}_2(1)$ is $\langle 2\rangle\langle 1\rangle 0\langle /1\rangle 0\langle 1\rangle 1\langle /1\rangle 1\langle /2\rangle$, which maps the sub-tag $\langle 1\rangle 0\langle /1\rangle \in \mathsf{tags}_1(1)$ to 0 and the sub-tag $\langle 1\rangle 1\langle /1\rangle \in \mathsf{tags}_1(1)$ to 1. For an overview of the structure of tags, see Figure 5.1.

[4]Tags can be seen as a form of unique indices.

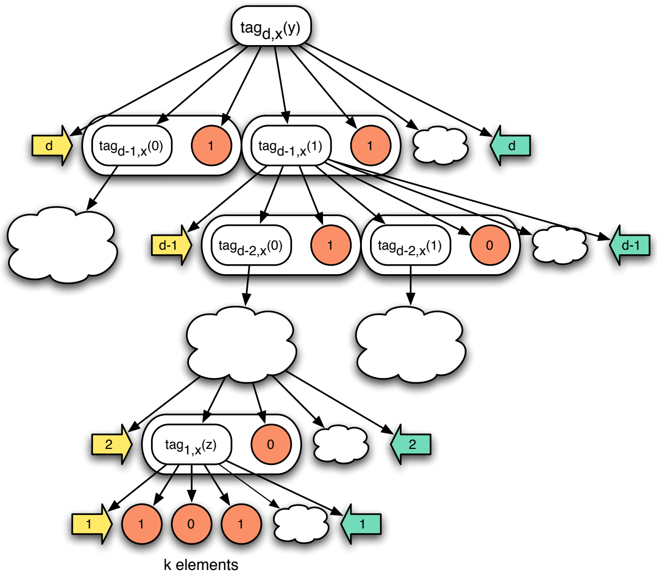


Figure 5.1 : Visualization of a tag

We now consider the number and size of tags.

Lemma 5.13. *There are $g(h, n)$ different tags in* $\mathsf{tags}_h(n)$.

Proof. We proceed by induction on the level h. For the base case, tags of level 1 are bit strings of length n, so there are $2^n = g(1, n)$ different tags in $\mathsf{tags}_1(n)$. For the inductive case, $\mathsf{tags}_{i+1}(n)$ is isomorphic to the power set $\mathcal{P}(\mathsf{tags}_i(n))$, so by the inductive hypothesis, the number of tags is $2^{g(i,n)} = g(i + 1, n)$. □

The construction we use for tags is similar to that used in [FG02]. One important difference is that our construction requires each tag in $\mathsf{tags}_h(n)$ to be a complete mapping, where the one in [FG02] allowed partial maps. Our approach trades a larger tag size for better alternation efficiency.

First, we consider the size blowup in tags for our construction. The following relaxed bound is sufficient for our final result:

Lemma 5.14. *The size $s_h(n)$ of a tag in* $\mathsf{tags}_h(n)$*, for $n > 2$, is at most $g(h - 1, 2n)$.*

Proof. We proceed by induction on the level h. For the base case of $h = 1$, we know the size of the tag $s_1(n)$ is $n + 2 \leq 2n = g(0, 2n)$. For the inductive case, we use the properties of g where for $h \geq 1$, $g(h, n)^2 \leq g(h, 2n)$, and for each $c > 0$, $g(h, n) + c \leq g(h, n + c)$. Both can be proved by a simple induction on h. Now $s_h(n) = g(h-1, n)(s_{h-1}(n)+1)+2 \leq g(h-1, n)(g(h-2, 2n)+1)+2 \leq g(h-1, n)(g(h-2, 2n) + 2)$. Since $h \geq 2$ and $n > 2$, we have $g(h - 2, 2n) + 2 \leq g(h - 2, 2n + 2) \leq g(h - 2, 2^n) \leq g(h - 1, n)$. Thus, $s_h(n) \leq g(h - 1, n)^2 \leq g(h - 1, 2n)$. □

Our goal is to use tags to mark the column index for each tile. This introduces a blowup in the size of $\pi_{w,I}$ and we want to maintain polynomial size in w. Thus, the size of the tags need to be polynomial in w. We need to show there is at least w distinct tags in size polynomial in w:

Lemma 5.15. *For a given w and $h \leq \log^*(w) - 2$, there exists $n > 2$ such that:*

- *There are at least w distinct tags in $\mathsf{tags}_h(n)$.*
- *The size of each individual tag in $\mathsf{tags}_h(n)$ is at most w.*

Proof. We choose n such that $g(h, n - 1) < w \leq g(h, n)$, so, by Lemma 5.13, we have at least w distinct tags. Since $h < \log^*(w) - 1$, we have $g(h, 2) \leq g(\log^*(w) - 2, 2) = g(\log^*(w) - 1, 1) \leq w$, so $n > 2$. By Lemma 5.14, each tag in $\mathsf{tags}_h(n)$ is of size at most $g(h - 1, 2n) \leq g(h - 1, 2^{n-1}) = g(h, n - 1) < w$. □

In addition to having enough tags, we also need to show the tags can be efficiently generated:

Definition 5.16. *We define a mapping $\mathsf{tag}_{h,n} : N \rightarrow \mathsf{tag}_h(n)$ by induction. For the base case, $\mathsf{tag}_{1,n}(x) := \langle 1\rangle bit_{n-1}(x) \dots bit_0(x)\langle /1\rangle$, where $bit_i(x)$ extracts the value of the ith bit of x. For the inductive case, $\mathsf{tag}_{h,n}(x) := \langle h\rangle \mathsf{tag}_{h-1,n}(g(h - 1, n) - 1)bit_{g(h-1,n)-1}(x) \dots \mathsf{tag}_{h-1,n}(0)bit_0(x)\langle /h\rangle$. In other words, $\mathsf{tag}_{h,n}(x)$ is a tagged binary number, which maps every $\mathsf{tag}_{h-1,n}(y)$ to the yth bit of x.*

Next, we describe the tagged word model $\pi_{w,I}$. The construction is with respect to a parameter $k > 2$ (which is the k of Theorem 5.10). We build the word using tags of level $h = k - 1$. The vocabulary of the word model is $\Sigma = \Sigma_h \cup D \cup \{\mathsf{r}\}$, where D is the set of tiles of T. We use r as a row marker. The purpose of the row marker is to ensure we don't mistake the cell at the end of one row to be horizontally consecutive to the cell at the beginning of the next row. Given a tiling problem of size w, choose n where $g(h, n-1) < w \leq g(h, n)$. Now $\pi_{w,I}$ is the concatenation of $w - 1$ blank rows and one initial row. The blank rows use an arbitrary non-initial tile $d_0 \in D$ as a placeholder (The actually tiling will be generated through quantification in the formula we will be model checking on this model.):

$$row_{blank} := \mathsf{tag}_{h,n}(0) d_0 \mathsf{tag}_{h,n}(1) d_0 \ldots \mathsf{tag}_{h,n}(w-1) d_0 \mathsf{r}$$

The initial row is the same as defined by I:

$$row_{init} := \mathsf{tag}_{h,n}(0) I(1) \mathsf{tag}_{h,n}(1) I(2) \ldots \mathsf{tag}_{h,n}(w-1) I(w) \mathsf{r}$$

Finally $\pi_{w,I} := (row_{blank})^{w-1} row_{init}$.

For ease of the construction we use in Chapter 6, we concatenate the rows in inverted order, where the first row is at the end of the word.

Lemma 5.17. *For every k, the length of the word $\pi_{w,I}$ is polynomial in w, and $\pi_{w,I}$ can be generated in time polynomial in w. In addition, the polynomial functions do not depend on k.*

Proof. We look at instance sizes $w > g(k, 1)$. Since $h = k - 1 \leq \log^*(w) - 2$, we can apply Lemma 5.15 to show that every tag has size less than w. So, $|\pi_{w,I}| \leq (w(w + 1) + 1)w$, which is in $O(w^3)$, and the coefficient is independent from k. Intuitively, the tagged word $\pi_{w,I}$ is obtained by tagging each position in a tiling with its column address in binary, tagging each address bit with its address in binary, and so on, iterated up to h levels. Each level adds a logarithmic term of overhead (i.e., $\log(w)$, $\log(\log(w))$, and so on) in the time needed to construct the label. Since $\pi_{w,I}$ is of length $O(w^3)$, we can generate it in time $O(w^4)$. $\square$

Lemma 5.17 shows that Assumption 2 and 4 for Theorem 5.10 above hold.

5.2.2.2 Tag Comparison Using QPTL

Next we describe how to use QPTL formulas to check equality of tags, which we use in order to check the vertical constraints of tilings. The formula we build uses a set P of propositions described below. For any letter $d \in \Sigma$, we have a corresponding proposition p_d, where in the word model π, we have $\pi, i \models \mathsf{p}_d$ iff d is the ith letter in π. Other propositions in P are used to mark sets of locations in the word model; we call them *markers*. Often, instead of concerning ourselves with sets of positions, we want to denote individual positions instead. This can be easily done by asserting the sets of positions to be *singleton* sets[5]; marker propositions that are restricted to singleton sets are called *singleton markers*. A pair of such singleton markers outline a subword by pointing to the beginning and ending positions of the subword. Given two markers p and q, the subword is denoted by $\pi_{[p,q]}$.

In the following construction, we use formula templates as follows: $\psi(p) := \psi'$ defines a template ψ with parameter p based on ψ', which is a QPTL formula. An instantiation $\psi(q)$ is a copy of ψ', where all free occurrences of p are replaced with q.

Lemma 5.18. *Given $h \geq 1$ and $n \geq 1$, one can construct a formula $\varphi_{h,n}(p, p', q, q') \in \Sigma_h^{QPTL}$ of length linear in $h + n$ such that if $\pi \models \varphi_{h,n}(p, p', q, q')$, then*

1. *p, p', q, and q' are each true at exactly one position in the model;*
2. *The subwords $\pi_{[p,p']}$ and $\pi_{[q,q']}$ are equal tags in $\mathsf{tags}_h(n)$;*
3. *The subword $\pi_{[p,p']}$ appears before the subword $\pi_{[q,q']}$.*

Proof. In the following, we first construct the formula $\varphi_{h,n}$, and then show that it satisfies the requirements in the lemma. The construction is inductive on the tag depth h. Before we go into the construction itself, we present some basic templates to state some commonly used relationships on propositions. First, we define $\mathsf{sing}(p)$

[5]When we have asserted such a set to be a singleton, we ignore the difference between such singleton sets and their sole member.

to assert that the proposition p is a singleton where $\mathsf{sing}(p) := Fp \wedge G(p \rightarrow XG\neg p)$. We also need to state that a proposition r is never true strictly inside a subword $\pi_{[p,q]}$ where $\mathsf{notbetween}(p, q, r) := G(p \rightarrow XG(XFq \rightarrow \neg r))$. We also need to state that two positions, both i letters away from singleton markers p and q, assert a proposition r where $\mathsf{same}^i(p, q, r) := G(p \rightarrow X^i r) \wedge G(q \rightarrow X^i r)$.

Now we construct the base case, where $\pi_{[p,p']}$ and $\pi_{[q,q']}$ mark tags in $\mathsf{tags}_1(n)$. In order to check equality, we need to state the following:

- The markers only appear once: $\mathsf{sing}(p) \wedge \mathsf{sing}(p') \wedge \mathsf{sing}(q) \wedge \mathsf{sing}(q')$.
- The ranges defined by the markers do not contain multiple tags at level 1: $\mathsf{notbetween}(p, p', \mathsf{p}_{\langle 1 \rangle}) \wedge \mathsf{notbetween}(q, q', \mathsf{p}_{\langle 1 \rangle})$.
- The markers point to the correct symbols: $G(p \rightarrow \mathsf{p}_{\langle 1 \rangle}) \wedge G(p' \rightarrow \mathsf{p}_{\langle /1 \rangle}) \wedge G(q \rightarrow \mathsf{p}_{\langle 1 \rangle}) \wedge G(q \rightarrow \mathsf{p}_{\langle /1 \rangle})$.
- The markers appear in the correct order: $G(p \rightarrow XFp') \wedge G(p' \rightarrow XFq) \wedge G(q \rightarrow XFq')$ (Figure 5.3).
- The bit strings are the same: $\bigwedge_{1 \leq i \leq n}(\mathsf{same}^i(p, q, \mathsf{p}_1) \vee \mathsf{same}^i(p, q, \mathsf{p}_0))$ (Figure 5.4).
- $\varphi_{1,n}(p, p', q, q')$ is the conjunction of all the formulas above.

By construction, $\varphi_{1,n}(p, p', q, q')$ is a formula in Σ_1^{QPTL} with size linear in n. To ease the understanding the nature of the formulas, please also see the figures 5.2 to 5.4.

For the inductive case, we need to check whether two tags $\pi_{[p,p']}$ and $\pi_{[q,q']}$ in $\mathsf{tags}_h(n)$ are equal. To that end, we need to check that the pair of tags maps all enclosing tags to matching values. In other words, for every pair of tags in $\mathsf{tags}_{h-1}(n)$, if the first one is inside $\pi_{[p,p']}$ and the second one is inside $\pi_{[q,q']}$, and the two tags are equal, their mapped values must be equal. We state the following:

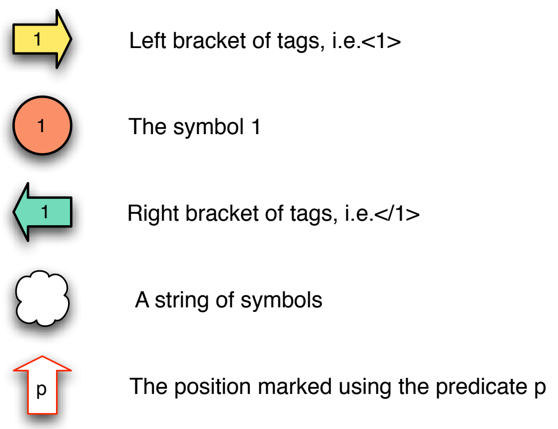

1
Left bracket of tags, i.e.<1>
1
The symbol 1
1
Right bracket of tags, i.e.</1>
A string of symbols
p
The position marked using the predicate p

Figure 5.2 : Legend for visualization of $\varphi_{k,n}(p, p', q, q')$

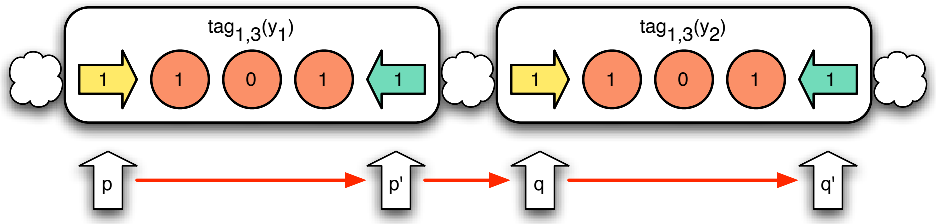


Figure 5.3 : Order check for the markers

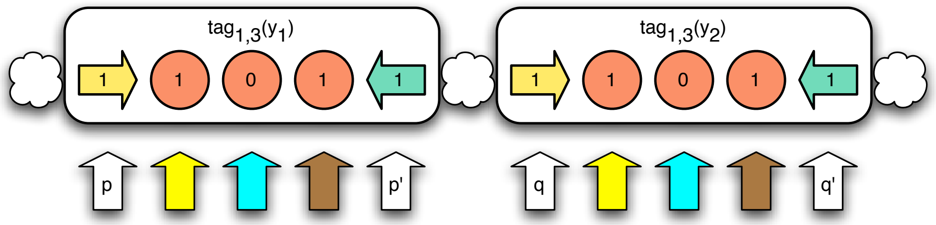


Figure 5.4 : Equality check for level 1

- The markers only appear once: $\mathsf{sing}(p) \wedge \mathsf{sing}(p') \wedge \mathsf{sing}(q) \wedge \mathsf{sing}(q')$.
- The ranges defined by the markers do not contain multiple tags at level h: $\mathsf{notbetween}(p, p', \mathsf{p}_{\langle h\rangle}) \wedge \mathsf{notbetween}(q, q', \mathsf{p}_{\langle h\rangle})$ (Figure 5.5).
- The markers point to the correct symbols: $G(p \to \mathsf{p}_{\langle h\rangle}) \wedge G(p' \to \mathsf{p}_{\langle/h\rangle}) \wedge G(q \to \mathsf{p}_{\langle h\rangle}) \wedge G(q \to \mathsf{p}_{\langle/h\rangle})$ (Figure 5.5).
- The markers appear in the correct order: $G(p \to XFp') \wedge G(p' \to XFq) \wedge G(q \to XFq')$ (Figure 5.5).
- We call the conjunction of above $\mathsf{format}_h(p, p', q, q')$.
- We use a proposition b to mark all starting positions of sub-tags in $\mathsf{tags}_{h-1}(n)$ that appear in $\pi_{[t,t']}$. $\mathsf{insidemarker}_h(t, t', b) := G(Ft \to \neg b) \wedge G(t' \to G\neg b) \wedge G(\neg \mathsf{p}_{\langle h-1\rangle} \to \neg b) \wedge G(t \to G((Ft' \wedge \mathsf{p}_{\langle h-1\rangle}) \to b))$
- Given (singleton) markers r, r', s, s', we can state that the pair of tags they point to map to the same value: $\mathsf{idmap}^h(r, r', s, s') := \varphi_{h-1,n}(r, r', s, s') \to (\mathsf{same}^1(r', s', \mathsf{p}_0) \vee \mathsf{same}^1(r', s', \mathsf{p}_1))$.
- We now define a formula to assert the equivalence of tags $\pi_{[p,p']}$ and $\pi_{[q,q']}$. For the meaning of the formula defined as $\mathsf{checktags}$, see proof below. $\mathsf{checktags}_h(p, p', q, q') := (\exists b, b')(\forall r, r', s, s')(\mathsf{insidemarker}_h(p, p', b) \wedge \mathsf{insidemarker}_h(q, q', b') \wedge ((G(r \to b) \wedge G(s \to b')) \to \mathsf{idmap}_h(r, r', s, s')))$. Note $\mathsf{idmap}_h(r, r', s, s')$ is true whenever r' or s' does not point to the end of a tag in $\mathsf{tags}_h(n)$. Also, see Figure 5.6 for a visual guide.
- $\varphi_{h,n}(p, p', q'q') := \mathsf{format}_h(p, p', q, q') \wedge \mathsf{checktags}_h(p, p', q, q')$.

Since in $\varphi_{h,n}$ the iterative part $\varphi_{h-1,n}$ appears under negation, and the quantifier prefix for the $\varphi_{h,n}$ outside $\varphi_{h-1,n}$ is $\exists\forall$, $\varphi_{h,n}$ in Σ_h^{QPTL} and of size linear in $h + n$.

Now we proceed to show that the construction is correct:

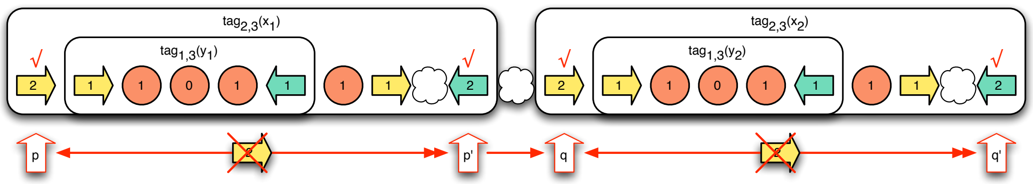


Figure 5.5 : Wellformedness check

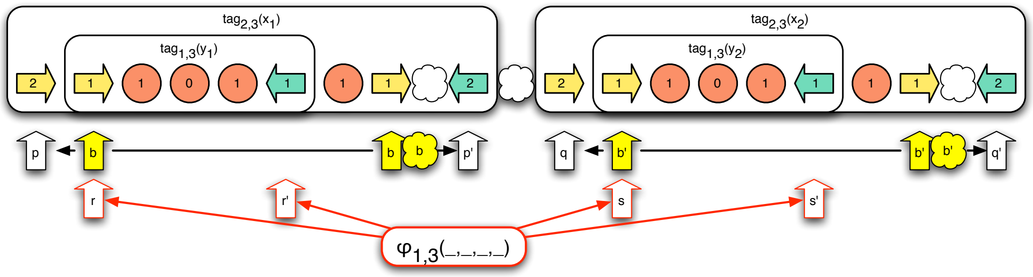


Figure 5.6 : Equality check for level above 1

Claim 5.19. *$\pi_{[p,p']}$ and $\pi_{[q,q']}$ are identical tags iff $\varphi_{h,n}(p,p',q,q')$.*

Proof. Again, we proceed by induction. The base case for $\varphi_{1,n}$ is quite simple. The fact that the singletons p, p', q, q' point to the correct position and are in the correct order is asserted by the formula. For well-formedness, we also need to check that the subwords $\pi_{[p,p']}$ and $\pi_{[q,q']}$ form actual tags, instead of being fragments that only mark part of a tag or fragments that cross multiple tags. Since the input word is constructed such that all tag boundary pairs must contain complete tags, $\pi_{[p,p']}$ or $\pi_{[q,q']}$ cannot be too short. And for $\pi_{[p,p']}$ or $\pi_{[q,q']}$ to be too long, they have to contain multiple tags, but we have asserted that tag markers $\langle 1 \rangle$ does not appear inside the subword. The equality of the subwords $\pi_{[p,p']}$ and $\pi_{[q,q']}$ is also directly asserted by the formula.

The inductive case is slightly more involved. Assuming that the construction is correct for level $h-1$, we show that it is correct for level h. format_h asserts that both tags are well formed exactly like the base case. In $\mathsf{checktags}_h$, we use the quantifier on b and b' to choose a set of positions that marks tags in $\mathsf{tags}_{h-1}(n)$, which we need to compare equality on. The set of positions b marks all starting positions of tags in $\mathsf{tags}_{h-1}(n)$ inside the subword $\pi_{[p,p']}$ (which is a tag in $\mathsf{tags}_h(n)$), and b' marks all starting positions of tags in $\mathsf{tags}_{h-1}(n)$ inside the subword $\pi_{[q,q']}$. The variables r, r', s, and s' are universally quantified. If any of them are not a singleton, or does not outline tags in $\mathsf{tags}_{h-1}(n)$, or are in the wrong order, $\mathsf{idmap}_h(r, r', s, s')$ returns true. If $\pi_{[r,r']}$ and $\pi_{[s,s']}$ do point to tags of the right type, but they are not tags we are actively checking (as marked by b and b'), the antecedent $(G(r \to b) \wedge G(s \to b'))$ part fails, skipping the check. Otherwise, idmap_h is used to check that for all pair of tags in $\mathsf{tags}_{h-1}(n)$ between $\pi_{[p,p']}$ and $\pi_{[q,q']}$, if they are the same, they map to the same value. Since the tags $\pi_{[p,p']}$ and $\pi_{[q,q']}$ are well formed by construction, this proves $\pi_{[p,p']}$ and $\pi_{[q,q']}$ are the same tag in $\mathsf{tags}_h(n)$ iff $\varphi_{h,n}(p, p', q, q')$. □ □

5.2.2.3 Constructing $\varphi_{T,w}$

Next, by using $\varphi_{k-1,n}(p, p', q, q')$ we construct a formula $\varphi'_{T,w}$ in Σ_k^{QPTL} to check that an annotated model is a tiling for the tiling problem (T, w) with size $w \leq g(k-1, n)$.

- Every position in the word has exactly one symbol: $\varphi_D := G(\bigwedge_{s\in\Sigma}(\mathsf{p}_s \to \bigwedge_{t\neq s} \neg\mathsf{p}_t)) \wedge G(\bigvee_{s\in\Sigma} \mathsf{p}_s)$.

- Horizontal constraints are observed: Define $\varphi_H := (\forall s, s', t, t')((\mathsf{sing}(s) \wedge \mathsf{sing}(s') \wedge \mathsf{sing}(t) \wedge \mathsf{sing}(t') \wedge G(s \to \mathsf{p}_{\langle k-1\rangle}) \wedge G(s' \to \mathsf{p}_{\langle /k-1\rangle}) \wedge G(t \to \mathsf{p}_{\langle k-1\rangle}) \wedge G(t' \to \mathsf{p}_{\langle /k-1\rangle}) \wedge G(s \to Fs') \wedge G(s' \to Ft) \wedge G(t \to Ft') \wedge \mathsf{notbetween}(s, s', \mathsf{p}_{\langle k-1\rangle}) \wedge \mathsf{notbetween}(s', t, \mathsf{p}_{\langle k-1\rangle}) \wedge \mathsf{notbetween}(s', t, \mathsf{p}_{\mathsf{r}}) \wedge \mathsf{notbetween}(t, t', \mathsf{p}_{\langle k-1\rangle})) \to \bigvee_{\langle d,d'\rangle\in H}(G(s' \to X\mathsf{p}_d) \wedge G(t' \to X\mathsf{p}_{d'})))$. This checks for every pair of tags $\pi_{[s,s']}$ and $\pi_{[t,t']}$, if $\pi_{[s,s']}$ appears before $\pi_{[t,t']}$, and there is no row marker or other tag markers that appear between $\pi_{[s,s']}$ and $\pi_{[t,t']}$, then the symbol that appears after $\pi_{[s,s']}$ and the symbol that appears

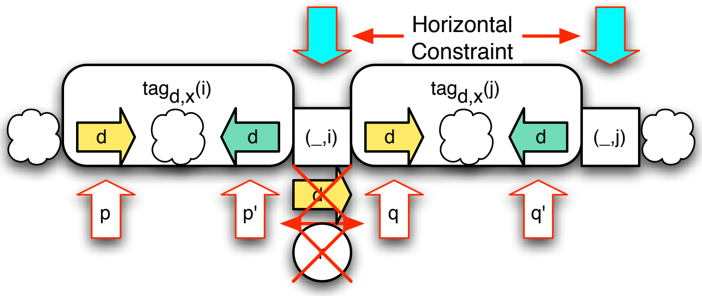


Figure 5.7 : Check for horizontal constraints

after $\pi_{[t,t']}$ are consecutive symbols on the same row, and need to satisfy the horizontal constraint. For a visualization, see Figure 5.7.

- Vertical constraints are observed: Define $\varphi_V := (\forall s, s', t, t')((\varphi_{k-1,n}(s, s', t, t') \wedge G(s' \to (F(\mathsf{p_r} \wedge Ft) \wedge \neg F(\mathsf{p_r} \wedge XF(\mathsf{p_r} \wedge Ft))))) \to \bigvee_{\langle d,d'\rangle \in V}(G(s' \to X\mathsf{p}_{d'}) \wedge G(t' \to X\mathsf{p}_d)))$. We know from Lemma 5.18 that $\varphi_{k-1,n}(s, s', t, t')$ checks whether $\pi_{[s,s']}$ and $\pi_{[t,t']}$ are identical tags in $\mathsf{tags}_{k-1}(n)$. In φ_V, we first check that the pair of tags are identical, i.e., they mark the same column; then we check that the pair of tags has exactly one row marker appearing between them, i.e., they are in consecutive rows; finally, we check that the tiles that appear after the pair of tags satisfy the vertical constraint. For a visualization, see Figure 5.8.

- The formula for checking whether the model is a solution is $\varphi'_{T,w} := \varphi_D \wedge \varphi_H \wedge \varphi_V$.

For $k \geq 2$, $\varphi'_{T,w}$ is a formula in Π^{QPTL}_{k-1}, since the occurrence of $\varphi_{k-1,n}$ in $\varphi'_{T,w}$ is negative, and the quantifier block of $\varphi'_{T,w}$ outside $\varphi_{k-1,n}$ is universal.

Given a tiling problem T and a mapping F, F is a solution for (T, w) iff $F_w \models \varphi'_{T,w}$. To check whether (T, w, I) has a solution, we take $\varphi_{T,w} = (\exists \mathsf{p}_{d_0})(\exists \mathsf{p}_{d_1}) \ldots (\exists \mathsf{p}_{d_m})\varphi'_{T,w}$.

Lemma 5.20. *Given a tiling instance (T, w, I), and an integer $k \geq 2$, we can generate in time $O(k + \log^{k-1}(w))$ a Σ^{QPTL}_k formula $\varphi_{T,w}$ of size $O(k + \log^{k-1}(w))$ such that*

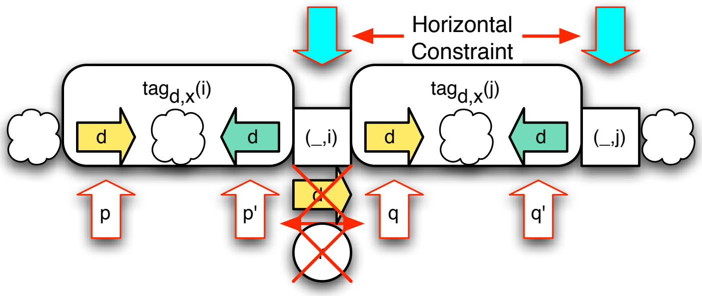


Figure 5.8 : Check for vertical constraints

$\pi_{w,I} \models \varphi_{T,w}$ *iff* (T, w, I) *have a solution, and the coefficient in* $O(k + \log^{k-1}(w))$ *is independent from* k*.*

Proof. Since $\varphi_{T,w}$ guesses a tiling non-deterministically and checks it through $\varphi'_{T,w}$, we have $\pi_{w,I} \models \varphi_{T,w}$ iff (T, w, I) has a solution, satisfying Assumption 1 of Theorem 5.10. Our choice of n such that $g(k-1, n-1) < w \leq g(k-1, n)$ means $n = \lceil \log^{k-1}(w) \rceil$, so $|\varphi'_{T,w}|$ and, in turn, $|\varphi_{T,w}|$ are in $O(k + n) = O(k + \log^{k-1}(w))$. The coefficient in $O(k + \log^{k-1}(w))$ is a constant based on our formula schemas. The formula can be generated in time linear in its size. □

Lemma 5.20 allows us to meet Assumption 1 needed for Theorem 5.10. We now show that we can also satisfy Assumption 3 and 4 of Theorem 5.10.

Lemma 5.21. *There exists a constant* $c > 0$ *such that for every* $k > 2$*, the formula* $\varphi_{T,w}$ *is small enough so that* $g(k-1, c|\varphi_{T,w}|)$ *is in* $\mathsf{poly}(w)$*, and* $\varphi_{T,w}$ *can be constructed in time polynomial in* w*.*

Proof. By Lemma 5.20, $|\varphi_{T,w}|$ is in $O(k + \log^{k-1}(w))$. So, there is a constant c and w_c such that for all $w > w_c$, $c|\varphi_{T,w}| < \log^{k-1}(w)$. Thus, $g(k-1, c|\varphi_{T,w}|) < g(k-1, \log^{k-1}(w)) = w$. We can construct $\varphi_{T,w}$ in time $O(\log^{k-1}(w))$, which is much smaller than $O(w)$. □

5.3 Parametric complexity of bounded-width QBF

We now come back to the decision problem of bounded-width QBF. A variant of this problem is studied in [Che04] in the context of quantified constraint satisfaction and an FPT algorithm is given, where both the width and the alternation depth are taken as parameters. The function f given in the algorithm is non-elementary. We prove here that, under complexity-theoretic assumptions, the function in the parameter is indeed non-elementary.

Theorem 5.22. *Satisfiability for Σ_k^{QBF} formulas of treewidth at most w can be solved in time $O(g(k,w)|\varphi|)$.*

Proof. The result follows by a careful analysis of the algorithm in [Che04], counting the number of distinct trees (called *choice constraints* in [Che04]) generated by the algorithm. □

The algorithm presented in [Che04] is an extension of the decision procedure for bounded-width propositional formulas, which can be decided in $O(2^w|\varphi|)$ time [DP89, Fre]. Unfortunately, it is unlikely that bounded-width QBF enjoys the same kind of tractability:

Theorem 5.23. *Assuming $P \neq NP$, there exists a constant $c > 0$ such that for every $h(\cdot)$ in $\mathsf{poly}(\cdot)$, satisfiability for bounded-pathwidth formula ψ in Σ_k^{QBF}, where k is an odd number > 2, of pathwidth at most w cannot be solved in time $g(k-1, c\cdot w)h(|\psi|)$.*

Proof. We combine Theorem 5.10 above with a translation from QPTL finite model checking to QBF satisfiability to show the lower bound. In Theorem 5.25 that follows, we show that for every QPTL formula $\varphi \in \Sigma_k^{QPTL}$, with odd k, and every word model π, we can construct a QBF formula $\psi = \varphi_{Q,\pi}$ of pathwidth at most $2|\varphi| - 1$ and alternation depth k such that $\models \psi$ iff $\pi \models \varphi$. Also, ψ is of size $O(|\varphi||\pi|)$, so there is a constant c' such that the construction takes time at most $c'|\varphi||\pi|$ and $|\psi| \leq c'|\varphi||\pi|$. Consider $c = c_{QPTL}/2$. For every $k > 2$ and for every polynomially bounded function $f'(\cdot)$, we have that $g(k-1, c(2|\varphi|-1))f'(|\varphi|) < g(k-1, c_{QPTL}|\varphi|)$ for large enough $|\varphi|$.

This is because $g(k-1, c(2|\varphi|-1)) = g(k-1, c_{QPTL}|\varphi|-c_{QPTL}/2)$, which is far smaller than $g(k-1, c_{QPTL}|\varphi|)$. For example, for every positive constant d, $g(2,x)/g(2,x-d)$ is not polynomially bounded on x. Now, suppose we can decide the satisfiability of ψ in time $g(k-1, c \cdot w)h(|\psi|)$, we can use it to decide the truth of $\pi \models \varphi$ through deciding ψ in time $g(k-1, c_{QPTL}|\varphi|)h'(|\pi|)$, with a polynomial $h'(\cdot)$. First, the time taken to solve the model checking of φ on π through QBF is $g(k-1, c(2|\varphi|-1))h(c'|\varphi||\pi|) + c'|\varphi||\pi| \leq g(k-1, c(2|\varphi|-1))(h(c'|\varphi||\pi|) + c'|\varphi||\pi|)$. Because $h(\cdot)$ is polynomial, there exist polynomial functions $f'(\cdot)$ and $h'(\cdot)$ such that $h(c'|\varphi||\pi|) + c'|\varphi||\pi| \leq f'(|\varphi|)h'(|\pi|)$. Also, from our assumption, $g(k-1, c(2|\varphi|-1))f'(\varphi) \leq g(k-1, c_{QPTL}(|\varphi|))$ for large enough $|\varphi|$. Thus, $g(k-1, c(2|\varphi|-1))h(c'|\varphi||\pi|) + c'|\varphi||\pi| \leq g(k-1, c_{QPTL}|\varphi|)h'(|\pi|)$ for large enough $|\varphi|$. This contradicts with Theorem 5.10 under the same $P \neq NP$ assumption. $\square$

A direct consequence is that bounded-width QBF can not be solved with an FPT algorithm with an elementary blowup in the parameters k and w, where k is the alternating depth and w is the width.

In the same way where the one exponential gap between the upper and lower bound QPTL model checking can be closed by using a stronger assumption, the same strengthening also works for bounded-width QBF:

Corollary 5.24. *Assuming NP cannot be solved in sub-exponential time, there exists a constant $c > 0$ such that for every $h(\cdot)$ in $\mathsf{poly}(\cdot)$, satisfaibility for bounded-pathwidth formula ψ in Σ_k^{QBF}, where k is an odd number > 2, of pathwidth at most w cannot be solved in time $g(k, c \cdot w)h(|\psi|)$.*

5.3.1 Translating QPTL finite model checking to QBF

Theorem 5.25. *Given a formula φ in Σ_k^{QPTL} (for odd k), and a finite word model π, we can generate in time $O(|\varphi||\pi|)$ a formula $\varphi_{Q,\pi}$ in Σ_k^{QBF} of size $O(|\varphi||\pi|)$ and pathwidth at most $2|\varphi|-1$, such that $\pi \models \varphi$ iff $\models \varphi_{Q,\pi}$.*

Proof. We perform the following translation from QPTL to QBF. Consider the QPTL formula $\varphi = (\exists x_{1,1}) \ldots (\forall x_{2,1}) \ldots (\exists x_{k,1}) \ldots \varphi'$. We first construct a propositional model π_Q such that $\pi_Q \models \varphi_Q$ iff $\pi \models \varphi$. We can then eliminate the propositional model by describing it as a conjunction of unit literals and conjoining with φ_Q (this does not increase the width of the formula). We establish the correspondence between $\pi \models \varphi$ and $\pi_Q \models \varphi_Q$ inductively.

The base case is when φ is quantifier free. A propositional model π_Q can capture all information encoded in the word model π: For each free proposition q of φ we create $|\pi|$ Boolean propositions $p_{q,i}$, $0 \leq i < |\pi|$, defined by $\pi_Q(p_{q,i}) = 1$ iff $q \in \pi(i)$. We refer to this set of Boolean propositions as P_Q.

Next, given a formula in LTL(X,F) (the propositional part of QPTL), without loss of generality, assume it to be in negation normal form, where negations only appear before propositions. Formulas in LTL(X,F) can be converted to negation normal form (by using also the temporal operator G) by pushing negations inward.

We write $\mathsf{sub}(\varphi)$ for the set of sub-formulas of φ, and $\mathsf{sub}'(\varphi)$ for the set of non-atomic sub-formulas of φ, in other words, $\mathsf{sub}'(\varphi) = \mathsf{sub}(\varphi) - \{q, \neg q | q \text{ is a proposition defined in } \varphi\}$. With each ψ in $\mathsf{sub}'(\varphi)$, we associate $|\pi|$ new propositions $p_{\psi,i}$. Essentially, we use a propositional encoding of accepting runs of the automaton constructed in [VW94] (adapted to finite words). The formula φ_Q can be encoded in CNF form by unrolling the formula onto the structure as follows. (For ease of understanding, clause groups for closely related propositions are written using the $\leftrightarrow$ operator, where $a \leftrightarrow (b \wedge c)$ represents the three clauses $\neg a \vee b$, $\neg a \vee c$, $a \vee \neg b \vee \neg c$, and similarly for $a \leftrightarrow (b \vee c)$ and $a \leftrightarrow b$.)

- In the following, whenever $\psi = a$ or $\psi = \neg a$, substitute $p_{a,i}$ or $\neg p_{a,i}$ for $p_{\psi,i}$.
- $\psi = X\psi'$: $C_\psi := (\neg p_{\psi,|\pi|-1}) \wedge \bigwedge_{0 \leq i < |\pi|-1} (p_{\psi,i} \leftrightarrow p_{\psi',i+1})$
- $\psi = F\psi'$: $C_\psi := (p_{\psi,|\pi|-1} \leftrightarrow p_{\psi',|\pi|-1}) \wedge \bigwedge_{0 \leq i < |\pi|-1} (p_{\psi,i} \leftrightarrow (p_{\psi,i+1} \vee p_{\psi',i}))$ (because $\pi, i \models F\psi$ iff $\pi, i \models \psi$ or $\pi, i+1 \models F\psi$.)

- $\psi = G\psi'$: $C_\psi := (p_{\psi,|\pi|-1} \leftrightarrow p_{\psi',|\pi|-1}) \wedge \bigwedge_{0\leq i<|\pi|-1}(p_{\psi,i} \leftrightarrow (p_{\psi,i+1} \wedge p_{\psi',i}))$ (Because $\pi, i \models G\psi$ iff $\pi, i \models \psi$ and $\pi, i+1 \models G\psi$.)

- $\psi = \psi' \wedge \psi''$: $C_\psi := \bigwedge_{0\leq i<|\pi|} p_{\psi,i} \leftrightarrow (p_{\psi',i} \wedge p_{\psi'',i})$

- $\psi = \psi' \vee \psi''$: $C_\psi := \bigwedge_{0\leq i<|\pi|} p_{\psi,i} \leftrightarrow (p_{\psi',i} \vee p_{\psi'',i})$

Now, we have $\varphi'_Q := p_{\varphi,0} \wedge \bigwedge_{\psi\in\mathsf{sub}'(\varphi)} C_\psi$ in CNF with $O(|\pi||\varphi|)$ clauses. We proceed to quantify out the propositions that correspond to valuations of subformulas in $\mathsf{sub}'(\varphi) = \{\psi_1, \psi_2 \ldots \psi_m\}$. Thus, π_Q define exactly the set of propositions we need to interpret $\varphi_Q := (\exists p_{\psi_1,0}) \ldots (\exists p_{\psi_m,|\pi|-1})\varphi'_Q$. φ_Q is in turn the *QBF translation* of φ.

Claim 5.26. *If φ is quantifier free, then $\pi \models \varphi$ iff $\pi_Q \models \varphi_Q$.*

Proof.

- $\Rightarrow$: We have $\pi \models \varphi$. We can extend π_Q with an assignment A on the quantified variables corresponding to subformulas in $\mathsf{sub}'(\varphi)$ such that $\pi_Q, A \models \varphi'_Q$ (thus $\pi_Q \models \varphi_Q$) by annotating the model with satisfied subformulas as follows: For every $\psi \in \mathsf{sub}'(\varphi)$, $A(p_{\psi,i}) = 1$ iff $\pi, i \models \psi$. Now we look at the clauses in φ'_Q. We have $A(p_{\varphi,0}) = 1$ since $\pi, 0 \models \varphi$. Every clause in $\bigwedge_{\psi\in\mathsf{sub}'(\varphi)} C_\psi$ holds because they encode exactly the semantics of QPTL as defined in Section 5.1. Thus, $\pi_Q, A \models \varphi'_Q$, and $\pi_Q \models (\exists p_{\psi_1,0}) \ldots (\exists p_{\psi_m,|\pi|-1})\varphi'_Q$.

- $\Leftarrow$: A similar construction can be performed as above. We have $\pi_Q \models (\exists p_{\psi_1,0}) \ldots (\exists p_{\psi_m,|\pi|-1})\varphi'_Q$. Thus, we have an assignment A on the quantified variables such that $\pi_Q, A \models \varphi'_Q$. We use induction on the structure of φ to show for any sub-formula ψ of φ, if $p_{\psi,i} \in A \cup \pi_Q$, then $\pi, i \models \psi$. For the base case of atomic sub-formulas, we have from the construction of π_Q that $\pi_Q \models p_{a,i}$ iff $\pi, i \models a$. For all inductive cases, there exist clauses in φ'_Q to ensure the connection between ψ and $\psi'(\psi'')$ for $\psi = X\psi'$, $\psi = F\psi'$, $\psi = G\psi'$, $\psi = \psi' \wedge \psi''$, $\psi = \psi' \vee \psi''$. Since there is a unit clause in φ'_Q to assert $p_{\varphi,0}$, we have $A \models p_{\varphi,0}$. From the induction, we have $\pi \models \varphi$. $\square$

We use QBF quantifiers to simulate QPTL quantifiers inductively. Assume we can translate ψ to ψ_Q in QBF. The inductive case is:

- $\varphi = (\exists x)\psi$: Take $\varphi_Q = (\exists p_{x,0}) \dots (\exists p_{x,|\pi|-1})\psi_Q$
- $\varphi = (\forall x)\psi$: Take $\varphi_Q = (\forall p_{x,0}) \dots (\forall p_{x,|\pi|-1})\psi_Q$

Clearly, φ_Q is in prenex normal form. The following claims are immediate.

Claim 5.27. $\pi \models \varphi$ *iff* $\pi_Q \models \varphi_Q$.

Claim 5.28. *The matrix of* $\varphi_{Q,\pi}$ *have pathwidth at most* $2|\varphi| - 1$.

Proof. The matrix of $\varphi_{Q,\pi}$ have clauses connecting at most two neighboring unrolling of propositions, so there is the following path decomposition $\{\{p_{\psi,0}, p_{\psi,1} \mid \psi \in \mathsf{sub}(\varphi)\}, \{p_{\psi,1}, p_{\psi,2} \mid \psi \in \mathsf{sub}(\varphi)\}, \dots, \{p_{\psi,|\pi|-2}, p_{\psi,|\pi|-1} \mid \psi \in \mathsf{sub}(\varphi)\}\}$, where the requirements of a path decomposition is met:

- The decomposition is a path.
- All constraints (clauses) occur in one path node.
- Every proposition only occurs in a sub-path (two consecutive nodes) of the decomposition.

The width of the decomposition is $2|\varphi| - 1$. □

Not only is the translation width efficient, it is also alternation efficient as well, in that

- For $\varphi \in \Sigma_i^{QPTL}$ where i is even, $\varphi_{Q,\pi} \in \Sigma_{i+1}^{QBF}$.
- For $\varphi \in \Sigma_i^{QPTL}$ where i is odd, $\varphi_{Q,\pi} \in \Sigma_i^{QBF}$.

The construction of $\varphi_{Q,\pi}$ can be performed in time linear to its size, i.e., $O(|\varphi||\pi|)$. In summary, the formula $\varphi_{Q,\pi}$ we constructed meets all the requirements posed by Theorem 5.25. □

Chapter 6

Hardness of Small-width QBF

In the last chapter, we studied the parameterized complexity of bounded-width QBF, where both the width and alternation depth of the instance is taken as parameters. Through telescoping, we got an lower bound on the parameter blowup that closely matched the upper bound. The next step, of course, is to consider the case where the alternation depth is not bounded. Unfortunately, telescoping through interpretation is no longer a useful technique for this case, since the formula we generate always have alternation depth linear in the size of the formula. Still, through telescoping, we can show something very close to the case of bounded-width and unbounded alternation, which is that of *very small* width and unbounded alternation. Here, we consider very small to be $O(\log^* n)$. Since in the last chapter, our telescoping used an embedding of a NP-complete problem, and generated a formula of $O(k+\log^k(n)$ size, the following NP-hardness result should not be a surprise.

Definition 6.1. *The class of* $\log^*$*-pathwidth QBF is the set of QBF formulas* ψ *that have pathwidth* $O(\log^*(|\psi|))$*.*

Theorem 6.2. *The class of* $\log^*$*-pathwidth QBF is NP-hard.*

Proof. NP-hardness is by a reduction from the NP-hard tiling problem. For a tiling instance (T, w, I), We first use the construction in Section 5.2 to encode it as a QPTL model-checking instance, then use the translation presented for Theorem 5.25 below to translate it to a QBF instance. For the first step of the construction, take the alternation depth k to be the least odd number $\geq \log^*(w) - 2$ (in other words, k is either $\log^*(w) - 2$ or $\log^*(w) - 1$). Because $k - 1 \geq \log^*(w) - 3$, we have that $\log^{k-1}(w) \leq 16$. By Lemma 5.20, the size of the QPTL formula $\varphi_{T,w}$ is $O(k + \log^{k-1}(w)) = O(k)$. By Lemma 5.17, the model $\pi_{w,I}$ has size $h(w)$ for some polynomial function h. Now we use Theorem 5.25 to translate the finite model-checking problem $\pi_{w,I} \models \varphi_{T,w}$ to a QBF instance $\psi = \varphi_{Q,\pi_{w,I},T,w}$ that has size $|\psi| = O(|\varphi_{T,w}||\pi_{w,I}|) = O(k \times h(w))$ and pathwidth at most $2|\varphi_{T,w}| - 1$. Since $k = O(\log^*(w))$, $O(k \times h(w))$ is polynomial in w. Similarly, the pathwidth of ψ is at most $2|\varphi_{T,w}| - 1$, which is in $O(k) = O(\log^*(w))$ and in turn, $O(\log^*(|\psi|))$ (since $|\psi|$ is polynomial in w). Thus, the family of $\log^*$-pathwidth QBF problems is $NP - hard$. $\square$

In contrast, propositional satisfiability problems that have $O(\log(|\psi|))$ pathwidth are still in P from the BDD size bound in Chapter 3.

The main result of this chapter would be on an extension of Theorem 6.2, where we would show that restricting the width of QBF problems to $\log^*$ have no impact of its classical complexity. Telescoping from QPTL model checking is unlikely to help, even through QPTL model checking is PSPACE-complete. The main reason is that the PSPACE class depend heavily on unbounded alternation [1], while restricting the formula to a small size effectively bounds the alternation on the same time.

To achieve PSPACE-hardness, we attempt to uncouple the alternation from the formula. Remember in the formula we used, only the outermost existential quantifier is used to guess part of the model, the rest of the quantifiers are only used for interpret the equality relation on tags. Instead of guessing a model, we can have two alternating players construct a model in hope of increasing the difficulty of the model checking problem. We first give an outline of the approach in following theorem, before presenting the details needed in the constriction:

Theorem 6.3. *The decision problem for the class of* $\log^*$*-pathwidth QBF is PSPACE-complete.*

[1]Unless the polynomial hierarchy collapses to some level, which would be a very unorthodox belief.

Proof. The problem is trivially in PSPACE. For PSPACE-hardness, we start from a PSPACE-complete tiling-game problem and combine two translation results to achieve a polynomial reduction. Given a tiling-game instance (T, w, I), we use Theorem 6.7 to convert it into a QPTL model-checking game $(\pi_{w,I}, \varphi_{T,w,G}, G)$. The model $\pi_{w,I}$ and the play order G have size polynomial in w, and $\varphi_{T,w,G}$ has size $O(\log^*(w))$. Then, by applying Theorem 6.5, we convert the QPTL model-checking game $(\pi_{w,I}, \varphi_{T,w,G}, G)$ to a QBF formula $\psi = \varphi_{Q,\pi_{w,I},w,G}$ that has size polynomial in $|\pi_{w,I}|$, $|G|$, and $|\varphi_{T,w,G}|$. In turn, $|\psi|$ is polynomial in w. Also, ψ have pathwidth at most $2|\varphi_{T,w,G}| - 1$, which is in $O(\log^*(w))$, and in turn, $O(\log^*(|\psi|))$. Thus, the family of $\log^*$-pathwidth QBF problems is PSPACE-complete. $\square$

6.1 Model-checking games and their relationship to QBF

First, we extend finite model checking for QPTL to a two-player game. Instead of being given a model to check, two players (the Constructor and the Spoiler) play a game to update a model for a specific formula. The Constructor's goal is to satisfy the formula in the resulting model, under opposition by the Spoiler.

Definition 6.4. *A* finite model-checking game *for QPTL is defined as follows: A finite-word model is a map $\pi : [0 \ldots n-1] \rightarrow 2^P$. The input to the game is a formula φ (which uses both the propositions in π as well as two distinguished propositions $\mathsf{p}_\exists$ and $\mathsf{p}_\forall$[2]), a word model π, the number m of rounds, and a* play order *G. The play order G labels each position $0 \ldots n-1$ in the model π either with a* round number*, which is an integer between 0 and $m-1$, or leaves it unlabelled. Thus, for each round number i, we have a set G_i of positions in the model defined by $G_i = \{d | G(d) = i\}$. We can omit m when the play order G is given. Two players, the Constructor and the Spoiler, attempt to update π under the play order G. The play order is processed*

[2]While it is possible to develop all the results here without the distinguished propositions, using them simplifies the construction needed in the reduction.

in increasing round order i. If i is even, the Constructor updates π^i to π^{i+1} (with $\pi^0 = \pi$) on every position d in G_i by choosing a new assignment $x_d \in 2^P$ and updating π^i with x_d. In summary, $\pi^{i+1} = \pi^i[d_1 \mapsto x_{d_1}][d_2 \mapsto x_{d_2}]\dots[d_k \mapsto x_{d_k}]$, where $G_i = \{d_1, \dots, d_k\}$. Here, $\pi[d \mapsto x](i) = \pi(i)$ if $d \neq i$ and $\pi[d \mapsto x](d) = x$. If i is odd, the Spoiler performs the update instead of the Constructor. The model π^m is extended onto the distinguished propositions $\mathsf{p}_\exists$ and $\mathsf{p}_\forall$ to become π', where $\mathsf{p}_\exists \in \pi'(d)$ iff $G(d)$ is even, and $\mathsf{p}_\forall \in \pi'(d)$ iff $G(d)$ is odd. π' is a winning model for the Constructor iff $\pi' \models \varphi$. The Constructor wins a model-checking game iff given π, G, m, and φ, he has a strategy to make the resulting model π' winning, and we write it as $\pi \models_G \varphi$.

The *finite model-checking game* for QPTL is a conservative extension of the finite model-checking problem on QPTL, since given an empty G, it is exactly the finite model-checking problem for QPTL. We can use a translation similar to that of Section 5.3 to convert QPTL model-checking games to QBF.

Theorem 6.5. *Given a QPTL model-checking game (π, φ, G), we can construct in polynomial time a QBF formula $\varphi_{Q,\pi,G}$ such that $\pi \models_G \varphi$ iff $\models \varphi_{Q,\pi,G}$. The size of $\varphi_{Q,\pi,G}$ is polynomial in $|\varphi|$, $|\pi|$, and $|G|$, and the pathwidth of $\varphi_{Q,\pi,G}$ is at most $2|\varphi| - 1$.*

Proof. We use the QBF translation of QPTL finite model checking presented in Theorem 5.25 as a tool. First, we construct φ_Q in QBF from φ and π. Next, we unroll the quantifiers implicit in G onto φ_Q. For each round in G where $G_i = \{d_1, \dots d_k\}$, the corresponding QBF prefix block Q_i is $(qp_{x_1,d_1})\dots(qp_{x_j,d_1})(qp_{x_1,d_2})\dots(qp_{x_j,d_k})$, where $P = \{x_1, \dots x_j\}$ is the set of propositions in φ excluding $\mathsf{p}_\exists$ and $\mathsf{p}_\forall$, and q is $\exists$ if i is even and $\forall$ if i is odd. I.e., for every round i in G, there has $|P| \times |G_i|$ additional quantifiers in the prefix to represent the choices made by the player corresponding to G_i. The formula $\varphi_{Q,G}$ is in turn $Q_1 \dots Q_m \varphi_Q$. The propositional model π_Q is generated from π in the same manner as Theorem 5.25, except that the assignment to $p_\exists$ and $p_\forall$ is defined by G instead of by π. π_Q can be written as a quantifier-free

QBF formula of unit clauses (as usual, when lifting the quantifiers in $\varphi_{Q,G}$ to the overall formula, we only need to keep the unit clauses in π_Q that correspond to the free propositions of $\varphi_{Q,G}$) and conjoined with $\varphi_{Q,G}$ to make $\varphi_{Q,\pi,G}$. Now we have $\pi \models_G \varphi$ iff $\models \varphi_{Q,\pi,G}$. Since $|\varphi_Q|$ is $O(|\pi||\varphi|)$, $|\varphi_{Q,\pi,G}|$ is in $O(|\pi||\varphi| + |G|)$. And because $\varphi_{Q,\pi,G}$ have the same matrix as $\varphi_{Q,\pi}$ except for unit clauses referring to $\mathsf{p}_\exists$ and $\mathsf{p}_\forall$, the same path decomposition can be used to show a pathwidth of at most $2|\varphi| - 1$. $\square$

6.2 Reducing tiling games to model-checking games

In Section 5.2, we showed that we can reduce the tiling problem (T, w, I) to a QPTL finite model-checking problem with model size in $\mathsf{poly}(w)$ and formula size in $O(k + \log^k(w))$. This result is combined with that of Theorem 6.2 to show that QBF formulas of $\log^*(n)$ width is NP-hard. The ideal result would be to show PSPACE-completeness, i.e., restricting QBF to $\log^*(n)$ width does not reduce its complexity. To do that, we need a tiling problem that is PSPACE-complete and has a polynomial number of cells. This requires inherent alternation in the problem in the form of a game. We use the following model of the tiling game:

Definition 6.6. *A* tiling game *uses a tiling system T and is played on a board with h rows and w columns with an initial condition I. Two players, the Constructor, and the Spoiler plays by tiling alternating rows, starting with the Constructor on row 1. After the whole board is tiled, the resulting tiling F is a winning tiling for the Constructor if one of the following holds:*

1. *The tiling F is a solution for the tiling problem (T, h, w, I).*
2. *The smallest j such that $(F(i, j-1), F(i, j)) \notin V$ or $(F(i-1, j), F(i, j)) \notin H$ is even.*

The second condition states that the Spoiler is the first to produce a failing tile. The Constructor wins the game (T, h, w, I) iff he has a strategy to force a winning tiling.

A unary, square version of the tiling game (T, w, I), where $h = w$ and I specifies the first row, like most variants of square tiling games [Chl86], is PSPACE-complete. We use finite model-checking games for QPTL to encode tiling games. The construction of the model $\pi_{w,I}$ from w is the same as in section 5.2.2.1. The play order G have w rounds, where each round G_{i-1} for $1 \leq i \leq w$ contains the set of positions in $\pi_{w,I}$ that corresponds to the cells (i, x) for $1 < x \leq w$. The formula is constructed as follows. Condition 1 is encoded by the $\varphi'_{T,w}$ of section 5.2.2.3. To encode condition 2, we guess the first position where the Spoiler made a failing play using a new variable e (and auxiliary variables e_D, e_H, and e_V). The following checks need to be performed. First, we check that e is a singleton and on a cell that is played by the Spoiler. Second, we check the tiling against the constraints. But, now the requirement for a correct tiling is reduced; the tiling is not necessarily a solution, as errors can occur on rows after the error marker e. Since we use an inverted row-major order, the correctness checks are not applied on positions before e in the word model, i.e., whenever Fe holds. Third, we check that an actual error occurred on the error marker. There are three types of errors, guessed by the variables e_D, e_H, and e_V. 1) The player chooses an assignment that does not correspond to a tile, i.e., more than one of p_ds or none of the p_ds are assigned to true. 2) The horizontal constraint is violated. 3) The vertical constraint is violated. We use the following formulas:

- $\varphi_e := \mathsf{sing}(e) \wedge G(e \rightarrow \mathsf{p}_\forall) \wedge G(e \leftrightarrow (e_D \vee e_H \vee e_V))$. This formula checks the error is performed by the Spoiler and guesses the type of the error.

- $\varphi'_D := G((\neg e_D \wedge (\mathsf{p}_\forall \vee \mathsf{p}_\exists)) \rightarrow (Fe \vee ((\bigvee_{d \in D} \mathsf{p}_d) \wedge (\bigwedge_{d,d' \in D, d \neq d'} \neg(\mathsf{p}_d \wedge \mathsf{p}_{d'})))) \wedge (e_D \rightarrow ((\bigwedge_{d \in D} \neg \mathsf{p}_d) \vee (\bigvee_{d,d' \in D, d \neq d'} \mathsf{p}_d \wedge \mathsf{p}_{d'}))))$

- $\varphi'_H := (\forall s, s', t, t')((\mathsf{sing}(s) \wedge \mathsf{sing}(s') \wedge \mathsf{sing}(t) \wedge \mathsf{sing}(t') \wedge G(s \rightarrow \mathsf{p}_{\langle k-1 \rangle}) \wedge G(s' \rightarrow \mathsf{p}_{\langle / k-1 \rangle}) \wedge G(t \rightarrow \mathsf{p}_{\langle k-1 \rangle}) \wedge G(t' \rightarrow \mathsf{p}_{\langle / k-1 \rangle}) \wedge G(s \rightarrow Fs') \wedge G(s' \rightarrow Ft) \wedge G(t \rightarrow Ft') \wedge \mathsf{notbetween}(s, s', \mathsf{p}_{\langle k-1 \rangle}) \wedge \mathsf{notbetween}(s', t, \mathsf{p}_{\langle k-1 \rangle}) \wedge \mathsf{notbetween}(s', t, \mathsf{p}_\mathsf{r}) \wedge \mathsf{notbetween}(t, t', \mathsf{p}_{\langle k-1 \rangle})) \rightarrow (((\bigvee_{\langle d,d' \rangle \in H} (G(s' \rightarrow X\mathsf{p}_d) \wedge G(t' \rightarrow X\mathsf{p}_{d'}))) \vee G(s' \rightarrow$

$Fe)) \wedge (G(s' \wedge Xe_H) \rightarrow \neg(\bigvee_{\langle d,d'\rangle \in H}(G(s' \rightarrow X\mathsf{p}_d) \wedge G(t' \rightarrow X\mathsf{p}_{d'}))))))$

- $\varphi'_V := (\forall s, s', t, t')((\varphi_{k-1,n}(s, s', t, t') \wedge G(s' \rightarrow (F(\mathsf{p_r} \wedge Ft) \wedge \neg F(\mathsf{p_r} \wedge XF(\mathsf{p_r} \wedge Ft))))) \rightarrow ((\bigvee_{\langle d,d'\rangle \in V}(G(s' \rightarrow X\mathsf{p}_{d'}) \wedge G(t' \rightarrow X\mathsf{p}_d)) \vee (s' \rightarrow Fe)) \wedge (G(s' \wedge Xe_V) \rightarrow \neg(\bigvee_{\langle d,d'\rangle \in V}(G(s' \rightarrow X\mathsf{p}_d) \wedge G(t' \rightarrow X\mathsf{p}_{d'})))))$

- Condition 2 is encoded by $\varphi_\forall := (\exists e, e_D, e_H, e_V)(\varphi_e \wedge \varphi'_H \wedge \varphi'_V \wedge \varphi'_D)$.

- The formula for the model-checking game is $\varphi_{T,w,G} := \varphi'_{T,w} \vee \varphi_\forall$.

Theorem 6.7. *Given a tiling game (T, w, I), we can construct in polynomial time a QPTL model-checking game $(\pi_{w,I}, \varphi_{T,w,G}, G)$ such that the Constructor wins the tiling game (T, w, I) iff $\pi_{w,I} \models_G \varphi_{T,w,G}$, the model $\pi_{w,I}$ and the play order G have size polynomial in w, and the formula $\varphi_{T,w,G}$ has size $O(\log^*(w))$.*

Proof. In the model-checking game, G encodes the same play order for the Constructor and the Spoiler as the tiling game. φ is a disjunction on the two winning conditions. The $\varphi'_{T,w}$ portion of $\varphi_{T,w,G}$ checks that the Constructor wins the tiling game through producing an tiling that is a solution to (T, w, I). The $\varphi_\forall$ portion of $\varphi_{T,w,G}$ checks that the Constructor wins the tiling game by guessing the position of the first failing play and checking that it is actually an error belonging to the Spoiler, as well as checking that all plays before the error are correct. So $\pi_{w,I} \models \varphi_{T,w,G}$ iff the Constructor wins (T, w, I). For the size bound, we use the same formula alternation depth k as Theorem 6.2, where k is the least odd number $\geq \log^*(w) - 2$. $\pi_{w,I}$ has the same polynomial $O(w^4)$ size as in Lemma 5.17, and G has size at most $|\pi_{w,I}|$. By construction, $\varphi_{T,w,G}$ has size $O(|\varphi_{k-1,n}|)$, which by Theorem 5.18, is $O(k - 1 + \log^{k-1}(w))$. Since $\log^{k-1}(w)$ is bounded by a constant from our choice of k, $|\varphi_{T,w,G}|$ is in $O(k)$, which is $O(\log^*(w))$. □

Chapter 7

A Symbolic Decision Procedure for QBF

The previous two chapters tackled intractability results for QBF. Our goal in this chapter is to go back to practical algorithms, by exploring an alternative approach to QBF solving based on symbolic quantifier elimination. While in Chapter 4, the symbolic quantifier elimination approach for propositional satisfiability is only shown to be effective on a small set of cases, symbolic techniques based on BDDs have been successful in various automated-reasoning applications, such as model checking [BCM$^+$92], planning [CR00], and modal satisfiability testing [PSV02, PV03]. More recent efforts focused on SAT solving using quantifier elimination, which, in essence, goes back to the original approach of [DP60], since resolution as used there can be viewed as a variable-elimination technique, ala Fourier-Motzkin. (Resolution is typically thought of as a constraint-propagation technique [Dec03], but since a variable can be eliminated once all resolutions on it have been performed [DP60], it can also be thought as a quantifier-elimination technique.) In [CS00] it is shown how *zero-suppressed decision diagrams* (ZDDs) [Min96] can offer a compact representation for sets of clauses and can support symbolic resolution (called there *multi-resolution*).

While based on the results in Chapter 4, the case for symbolic techniques in SAT solving cannot be said to be too strong, they are intriguing enough to justify investigating their applicability to QBF. On one hand, even though extending search-based technique to QBF resulted in the development of many highly-optimized solvers, QBF is still a "well-solved" problem like QBF, and all these solvers only work on a small subset of real-life problems. On the other hand, symbolic quantifier elimination handles universal quantifiers just as easily (and sometimes more easily) as it handles existential quantifiers, so extending symbolic techniques to QBF is quite natural. (Symbolic techniques have already been used to address conformant-planning problems [CR00], which can be expressed as QBF instances of low alternation depth.) In this chapter we investigate the two symbolic techniques to QBF. We extend the ZDD-based multi-resolution approach of [CS00] and the BDD-based approach of symbolic quantifier elimination of Chapter 4. We call the two approaches *QMRES* and *QBDD*. We compare these two approaches with three leading search-based QBF solvers: Quaffle and QuBE, which were mentioned earlier, and Semprop [Let02]. Unlike other comparative works [LST03], we decided to split our benchmark suite according to the provenance of the benchmarks, as our goal is to identify classes of problems for which the symbolic approaches are suited. We use a benchmark suite generated by Rintanen [Rin99], which consists of a variety of constructed formulas (we omitted the random formulas), a second generated by Ayari [AB00], which consists of scalable formulas converted from circuit descriptions and protocol descriptions, and those generated by Pan [PV03], which consist of QBF formulas translated from modal logic formulas. Our experiments reveal that QMRES is significantly superior to QBDD. In fact, QBDD does not seem to be a competitive solver. (Though we return to this point at our concluding discussion.) In contrast, QMRES is quite competitive against the search-based solvers. While it is comparable to search-based method on Rintanen's formulas, QMRES outperforms them on Ayari's and Pan's formulas. At the same time, QMRES performs abysmally on random formulas. This suggests that symbolic techniques ought to be considered as complementary to search-based techniques and should belong in the standard tool kit of QBF solver implementers.

7.1 Background

If constraints are represented by clauses rather than by BDDs, then variables can be eliminated via resolution. Given a set C of clauses and a variable x, the variable can be eliminated by adding to C all resolvents on x and then eliminating all clauses where

x occurs. Formally $(\exists x)C$ is logically equivalent to $Resolve_x(C)$, where $Resolve_x(C)$ is the set of clauses obtained from C by adding all resolvents on x and then deleting all clauses containing x. In fact, completeness of resolution is shown by eliminating all variables one by one, each time replacing a set C of clauses by $Resolve_x(C)$ [DP60]. (Eliminating variables in such a fashion is reminiscent of Fourier-Motzkin variable elimination for systems of linear inequalities and of Gaussian variable elimination for systems of linear equalities.) This approach is also referred to as *directional resolution* [DR94] (see report there on experimental comparison of directional resolution to search-based techniques). Here, for the resolution-based approach, we use the ZDD representation used by [CS00]. See Section 4.5.1 for more details.

Most SAT solvers are search-based, following the ideas of [DLL62]. QBF solvers build upon search techniques developed for SAT, forcing backtracking on universal variables and branching on variables according to alternation order [CSGG02]. A decision procedure based on an extension of resolution to QBF (called *Q-resolution*) is described in [BKF95], but, to the best of our knowledge has only been implemented as a learning technique for DPLL. We comment later on the difference between Q-resolution, DPLL, and our multi-resolution approach to QBF.

7.2 Symbolic Quantifier Elimination for QBF

The basic idea of our approach is to extend symbolic quantifier elimination from SAT to QBF. Given a QBF formula $\varphi = Q_1X_1Q_2X_2\ldots Q_nX_n\varphi'$, we eliminate the quantifiers from *inside out*, that is, in order of decreasing alternating depth, starting with the variables in X_n. At each stage, we maintain a set of constraints, represented symbolically either as a ZDD (expressing a set of clauses) or as a set of BDDs. To eliminate an existential variable from a set of clauses we perform multi-resolution, while universal variables can be eliminated by simply deleting them [BKF95]. To eliminate an existential variable x from a set of BDDs we conjoin the BDDs in whose support set x occurs and then quantify it existentially, while to eliminate a universal variable we quantify it universally. (We can apply universal quantification to an BDD B: $(\forall x)B = \textsc{apply}(B|_{x\leftarrow 1}, B|_{x\leftarrow 0}, \wedge)$.) The variables within a quantifier block Q_iX_i are unordered and can be eliminated in any order. Here we apply the heuristics described in Chapter 4. The BDD approach here can be seen as a QBF analogue of BE in Chapter 4, and similarly, the ZDD approach here is a QBF analogue of multi-resolution in Chapter 4.

We note that our resolution approach to QBF is different than that of [BKF95]. We require that quantifiers be eliminated from the inside out; thus, resolution can be performed only on the existential variables in the innermost quantifier block. In contrast, Q-resolution [BKF95] allows resolution on non-innermost existential variables. The difference stems from the fact that the focus in Q-resolution is on generating resolvents, while the focus here is on quantifier elimination. For Q-resolution to be complete, *all* resolvents need to be kept. In contrast, once we have performed multi-resolution on a variable x, all clauses containing it are deleted.

First, we describe QMRES, a multi-resolution QBF solver. For the definition of the ZDD-based representation and the semantics of the operators, please see Chapter 4.5.1. We provide pseudocode in Algorithm 1:

Note that in addition to multi-resolution the algorithm applies a naive form of unit propagation (weaker then what is described in [ZM02b]). When the clause set is represented using ZDDs, clauses with only a single literal can be easily enumerated without traversing the whole ZDD, since such clauses are represented by a path of length 2 in the ZDD. Existential unit literals can then be resolved on without regard to their alternation depth. (If a universal literal becomes unit, the formula is false.) The overhead of such a check are negligible so we applied it in all cases.

We now describe QBDD, an BDD-based QBF solver. We provide pseudocode in Algorithm 2[1]:

[1]In the implementation, we use an approximation where $\textit{choose\_bucket}(R_i) = i+1$, which avoided the overhead of traversing the BDD R_i and finding the lowest ordered variable according to v.

Algorithm 1 Multi-resolution for QBF

```
Q-Multi-Res($\varphi$, $S$, $v$)
**Require:** $S$ is the set of clauses forming matrix of $\varphi$, and $v = \langle v_1 \ldots v_n \rangle$ is an order
  of variables where $alt(v_i) \geq alt(v_{i+1})$
**Ensure:** returns **true** if $\varphi$ is valid and **false** otherwise
  **for** i=1..n **do**
    **if** $v_i$ is existential **then**
      $S \Leftarrow (S_{v_i^+} \times S_{v_i^-}) + S_{v_i'}$
    **else**
      $S \Leftarrow S_{v_i^+} + S_{v_i^-} + S_{v_i'}$
    **end if**
    $S \Leftarrow Unitprop(S)$\{Apply unit propagation\}
  **end for**
  return $S \neq \{\phi\}$
```

In Chapter 2 we described the MCS heuristics for variable ordering. MCS is only one of many variable-ordering heuristics that are used to approximate treewidth [KBv01]. We explored several other variable-ordering heuristics in [PV04a]. MCS came out as the overall best performers across a wide variety of benchmarks. It is interesting to note that MCS is not necessarily the best performer in terms of induced width. Other heuristics, such as *min-fill* [Dec03] yield better induced width. One has to remember, however, that variable order impacts not only the induced width but also decision-diagram size. In turns out that a variable that reduce the size of the support set of decision diagram does not necessarily reduces its size. While these effects may be less marked for ZDD-based clause representation, we chose to use MCS for both of our algorithms.

As described earlier, however, MCS is computed from the matrix of the QBF formula, ignoring completely the quantifier prefix. Since quantifier elimination proceed

Algorithm 2 Bucket Elimination for QBF

```
  QBDD($\varphi$, $S$, $v$)
Require: $S$, $v$ as in Q-Multi-Res
Ensure: returns true if $\varphi$ is valid and false otherwise
  Build BDD clusters $S_1 \dots S_n$, where a clause $c \in S$ is in cluster $S_i$ if $v_i$ is the lowest ordered variable in $c$
  for i=1..n do
    if $v_i$ is existential then
      $R_i = \exists x_i \bigwedge_{c \in S_i} c$
    else
      $R_i = \forall x_i \bigwedge_{c \in S_i} c$
    end if
    if $R_i = \mathbf{0}$ then
      return false
    end if
    $j = choose\_bucket(R_i)$
    $S_j = S_j \cup \{R_i\}$
  end for
  return true
```

from the inside out, we need to adapt MCS to take into account alternation depth. We first perform MCS on the matrix only, ignoring alternation. Then, variable order for quantifier elimination is generated, where at each step we choose a variable from those with the highest alternation depth that has the lowest MCS rank.

Beside testing the impact of structural heuristics, we also hope that symbolic approaches can solve problems that are otherwise intractable for search-based approaches because of their use of a compressed data structure. Such use of compression can lead to a more powerful underlying reasoning system.

In order to analyze the difference between search and symbolic-based approaches for QBF, we take an in-depth look at what modern search-based solvers do. The following Figure 7.1 is an excerpt of the algorithm presented in Figure 1 of [ZM02b]. Here, we ignore the conflict-driven learning portion and concentrate on solution-driven learning portion. In other words, we only consider the code path needed to determine that a QBF instance is satisfiable. The function deduce() performs Boolean constraint propagation on a representation they call *ACNF*, which is a disjunction of the original CNF matrix with a DNF of *learned solutions*. The function analyze_SAT() performs *term resolution*, the dual of Q-resolution that operates on terms instead of clauses.

Consider the following CNF matrix:

$$\bigwedge_{1 \leq i \leq n} (a_i \vee b_i) \wedge (\neg a_i \vee \neg b_i)$$

Here, every satisfying solution need to contain assignments to all variables. Consider the quantifier prefix $\forall a_1 \ldots \forall a_n \exists b_1 \ldots \exists b_n$ to the above matrix. Since learned solutions ignore innermost existential literals (just like learned clauses ignore innermost universal literals), each learned solution is a term on the variables a_1 to a_n. But all the solutions needed to satisfy the matrix give rise to *minterms* in a_1 to a_n, which limits their ability to prune out search space.

Theorem 7.1. *There are at least 2^n calls to deduce() in the solving of the formula above through solution-driven learning.*

```
while(1) {
  decide_next_branch();
  while(true) {
    status=deduce();
    if(status == CONFLICT){
      ...
    }
    else if (status==SATISFIABLE) {
      blevel= analyze_SAT()
      if (blevel==0)
        return SAT;
      else  backtrack(blevel);
    }
    else break;
  }
}
```

Figure 7.1 : Satisfiability searching part of Algorithm in Fig.1, [ZM02b]

Proof. In order to terminate, since the formula is satisfiable, we need to ascertain all 2^n minterms on a_is lead to satisfying assignments on the propositional matrix. This is done through deducing the empty term during term resolution. We ignore all the times when deduce() leads to conflicts, and only consider the times when it returned with SATISFIABLE. There are only two possibilities. One is that the SATISFIABLE result is because the original propositional matrix is satisfied. In which case, because the assignment is a complete assignment on a_is and b_is, we can cover at most 1 out of 2^n possible minterms. The other case is that we went into a pruned search space, i.e., one of the learned terms is satisfied. In this case, no previously unvisited minterms are covered. In addition, in the learning procedure analyze_SAT(), properties of resolution asserts no previously unvisited minterms are covered. So, in order to decude an empty term to assure satisfiability, we need to visit 2^n minterms through enumerating all 2^n satisfiable assignments to the original propositional matrix, leading to at least 2^n calls to deduce(). $\square$

One important thing to note is that for problems that are satisfiable, DPLL and resolution are no longer inter-reducible, in contrast to the polynomial simulation between DPLL and resolution for propositional logic in [BKS03]. For example, the formula above is *saturated* with respect to resolution, in other words, there are no possible clauses to be deduced through resolution. In turn, refuting the formula is not possible and satisfiability can be immediately ascertained using resolution, as opposed to the exponential number of steps we need when using DPLL where resolution is only used as a learning technique.

One important distinction to note here is that although conflict-driven learning and solution-driven learning seems to symmetric, since they are dual techniques, they are really quite different in practice. While both learning techniques perform resolution, conflict-driven learning generates *local* consequences, i.e., clauses that are implied by a subset of the propositional matrix. On the other hand, solution-driven learning generates *global* consequences, which are terms that are solutions to the whole propositional matrix. This difference is determined by the choice of ACNF representation, where clauses and terms are not symmetrically represented. In contrast, bottom-up symbolic algorithms that builds BDDs are truely symmetric, which is a single consequence generation procedure for both conflicts and solutions.

7.3 Experimental Results

We compare the symbolic approaches with three search-based solvers: QuBE [GNT01], Quaffle [ZM02b], and Semprop [Let02]. These search-based solvers use sophisticted heuristics for branch-variable selection and lemma/conflict caching. For both symbolic solvers, we used CUDD [Som98] as the underlying decision diagram engine, and for QMRES, we used the multi-resolution engine implemented by Chatalic and Simon [CS00].

We use three classes of benchmarks from the QBFLIB benchmark suites[Nar], those generated by Rintanen [Rin99], from which we omitted the random formulas but kept a variety of hand constructed formulas, those generated by Ayari [AB00], which consist of scalable formulas converted from circuit descriptions and protocol descriptions, and those generated by Pan [PV03], which consist of formulas translated from modal logic.

7.3.1 Symbolic vs. Search

A first observation, consistent with propositional logic, is that symbolic approaches typically performs very badly on random problems. For example, symbolic approaches are orders of magnitude slower for uniform propositional 3-CNF problems [PV04a]. (In our QBF experiments, the symbolic approaches completed none of the uniform random formulas in the Rintanen's benchmarks within the 1000s timeout limit.) In the following, we compare the symbolic and search approaches only on constructed formulas, ignoring the results for random problems. (In general, QBF solvers typically behave quite differently on random vs. non-random formulas, which is why

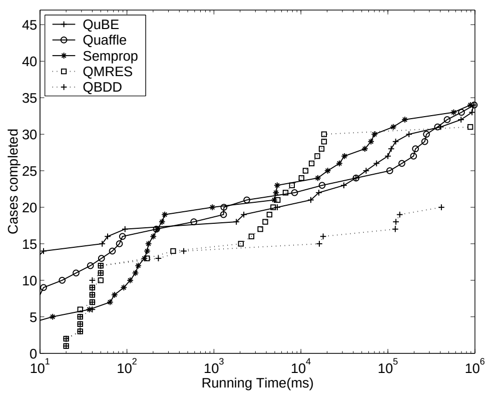


Figure 7.2 : Rintanen's Benchmarks (Non-random)

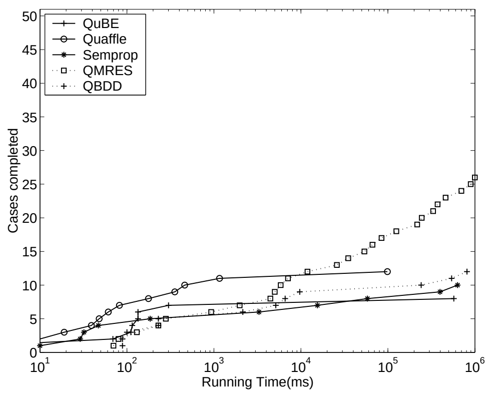


Figure 7.3 : Ayari's Benchmarks

comparative studies separate the two cases [LST03].)

First we evaluated our solvers on Rintanen's and Ayari's benchmark suites [Rin99, AB00]. Rintanen's benchmark suite is one of the first benchmark suite for QBF, including formulas constructed from planning problems, hand-constructed formulas, and randomly generated formulas (which we omitted) covering a wide range of difficulty and alternation depth. Ayari's benchmark suite is one of the first to encode bounded-model-construction problems in QBF through M2L-STR, a monadic second-order logic on finite words [AB00], with typically quite low alternation depth.

The results for Rintanen's benchmarks are plotted in Figure 7.2 and the results for

Ayari's Benchmarks are plotted in Figure 7.3. We used the plotting style presented in [GMTZ01, SS01], which plotted the number of completed cases against the time taken to run each case. A solver with a higher line dominates (i.e., is considered to be better than) one with a lower line since it solved more cases in the same amount of time.

Our first observation is that QMRES clearly dominates QBDD. This is somewhat surprising, since similar experiments we performed on propositional formulas showed that with the same variable order, the BDD-based approach dominates the ZDD-based in most cases, falling behind only for highly under-constrained problems, where the compression of ZDD-based clause set representation is greater. In contrast, ZDD-based clause sets seems to be getting better compression across the range of the QBF problems, resulting in node usage that are orders of magnitude smaller then that of QBDD

Comparing symbolic solvers against search-based solvers, the picture is somewhat mixed. For Rintanen's benchmarks, there is no clear winner, but for Ayari's benchmarks, QMRES showed a clear edge. Some of the formulas in the Rintanen's benchmarks, for example, the formulas resulted from the encoding of the blocks problems in artificial intelligence, only have a low alternation depth and small number of universal variables, allowing search to perform effectively. In essence, such problems are closer to SAT, where search-based approach typically outperform symbolic solvers [PV04a]. On the other hand, Ayari's problems are derived from circuit and protocol problems, whose symmetry favors the compression of the symbolic approach. It is interesting to note that the advantage of QMRES shows only when more difficult problems are considered. On easier problems search-based methods are faster.

Next, we come to formulas obtained by translation to QBF from modal formulas in the logic $\mathcal{K}$ [PV03]. The original modal formulas are scalable classes constructed by Heuerding and Schwendimann [HS96], where modal properties are nested to construct successively harder formulas. The resulting QBF formulas are grouped in the same

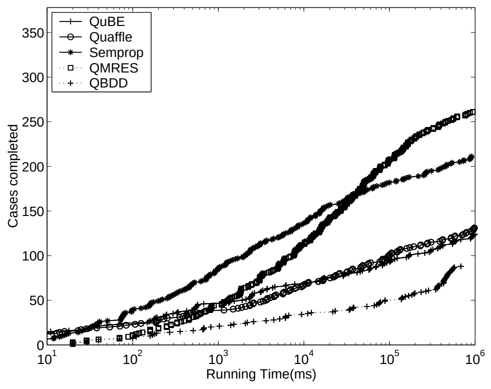


Figure 7.4 : Modal Logic Benchmarks

18 classes as the modal formulas, half satisfiable and half unsatisfiable, and each class contains 21 cases of which alternation depth scales linearly. Using translation from modal logic allowed construction of high alternation-depth problems that are non-trivial and compact. The formulas span a large range of difficulty and sizes, from hundreds to tens of thousands of variables. All the original modal formulas can be solved using solvers specific for the modal logic $\mathcal{K}$. We plotted time vs. cases solved in Figure 7.4. We see that QMRES clearly dominates the search-based methods. (This is consistent with the results described in [PV03], where symbolic modal solvers dominate search-based solvers.)

A fundamental question is why a symbolic method outperforms search-based method for QBF, while the situation is reversed for SAT [PV04a]. A possible explanation is that propositional satisfiability is in essence a search for a satisfying assignment. SAT solvers excel in quickly eliminating unsuccessful search branches, combining backjumping and conflict-based learning. In contrast, search-based QBF solvers have to deal with backtracking forced by the universal quantifers, which seems to nullify many of the advanced search heuristics of SAT solvers. Since QBF solving requires dealing with the whole space of possible assignments, symbolic methods benefit from the compression provided by decision diagrams.

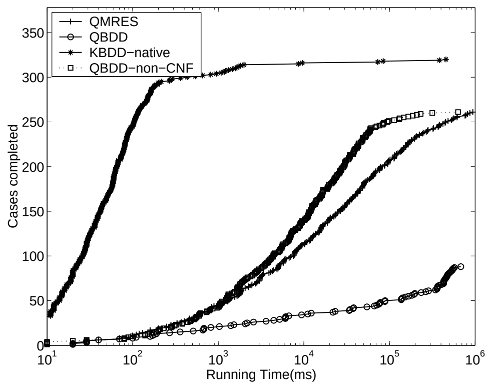


Figure 7.5 : Study of overhead on ML benchmarks

7.3.2 QMRES vs QBDD

To better understand the disappointing performance of the BDD-based approach, we take a deeper look at Pan's formulas, which are generated from the modal formulas of [HS96] through two different translation steps, first from ML to non-CNF QBF, then from non-CNF QBF to QBF. In addition to running QMRES and QBDD on the final QBF formulas, we run KBDD, an BDD-based modal solver [PV03], on the original modal formulas. We also run an ad-hoc BDD-based solver on the intermediate non-CNF QBF formulas, where we translate propositional formulas to BDDs without going through the CNF step and then apply quantifier elimination.

In Figure 7.5, we plotted the performance of the corresponding solvers. We see that the native solver KBDD performs best, with QMRES and QBDD-non-CNF very close to each other and not far behind the native solver. There is a much larger gap between these two and the performance of QBDD, resulting from the CNF translation. The gap between the performance of KBDD and QBDD-non-CNF can be attributed to the conversion of conjunction in modal formulas to universal quantification in the QBF translation. The gap between the performance of QMRES and QBDD-non-CNF to that of QBDD can be attributed to the the cost incurred by the BDD-based solver in handling the additional variables generated in the conversion to CNF and the difficulty in implementing certain optimizations under the BDD representation, for example, unit propagation. This suggests that the BDD-based approach might be more effective for problems whose natural encoding is not in CNF.

Chapter 8

Conclusion

In this dissertation, we looked at both practical and theoretical impacts of structural heuristics on symbolic decision procedures. In particular, we concerned ourselves with the structural property of *width*, which are measured on the interaction graph of a problem instance. The contrast between how bounded-width impacts theoretical complexity and how width-based heuristics perform in practice is quite remarkable. For propositional logic, bounding the width of the interaction graph induced a tractable subclass, but practical instances rarely exhibit the small-width needed. For relatively high-width problems, optimizing other factors on BDD size become as important as optimizing width. So optimizing through structural heuristics alone shall not be the primary goal. In contrast, for QBF, while bounded width of the matrix do not induce tractability for most cases beyond a small, constant width and constant alternation, applying structural heuristics to a symbolic decision procedure does lead to a competitive implementation.

There are a few reasons for this major disparity between theory and practice. Primarily, while structural heuristics is a very important factor in the efficiency of DD-based decision procedure implementations, it is not the sole factor in determining the performance of a DD-based decision procedure. Often, the nature of the algorithm used plays a bigger part. For example, the difference between search and symbolic solvers are much bigger than the effect of different heuristics on each individual approach. Search-based solvers do a good job on propositional logic through leading the search towards the solution quickly if the instance is satisfiable. This effect is particularly marked for instances that are quite under-constrained, for example, for random 3-CNF instances where the density is below 3.8, DPLL-based search procedures are polynomial [CDS+00]. On the unsatisfiable side, while random unsatisfiable instances may be hard (exponential time on DPLL [CDS+00]), many industrial unsatisfiable instances still have small resolution proofs, allowing search-based solvers to tackle them efficiently. In contrast, pure, symbolic decision procedures are model-building in nature, and performs a bottom-up construction. In turn, at each step it only looks at a portion of the whole instance. Such an algorithm is in turn not able to use the whole instance to simplify the construction process. Thus, even though the compression allowed by the BDD-based representation is significant, a significantly larger portion of solution-space need to be covered for a symbolic decision procedure in comparison to the search-based approach. Still, the symbolic solvers exhibited better performance on problems that are provably *hard for resolution*, and there are research in the development of *hybrid solvers* for propositional satisfiability [JS04, DK03], where search and DD-based compressed representations are combined to develop a solver with broader applicability.

Our comparison of different clustering and quantification scheduling for DD-based symbolic proposition decision procedures further reinforced the view that structural constraints, especially width, are not the sole indicator of performance. While BE is optimal with respect to an optimal variable order, for many over constrained instances, BM exhibited better performance. The reason is that constrainedness, an important factor in the size of BDDs, also played an important role. Balancing the different factors that impact the size of BDDs would be a major challenge in the development of symbolic and hybrid solvers.

The experimental study of symbolic decision procedures for QBF gave more promising results. For QBF, search-based solvers perform markedly worse compared to their counter-parts on propositional logic. In addition, such reduced performance for search compared to propositional logic are also seen in other PSPACE-complete logics, for example, the modal logic K. One of the most important reasons is that QBF (and PSPACE-complete problems) does not admit small solutions in general under the usual complexity assumptions. Thus, search-based solvers would need to perform an exponential number of search operations to demonstrate satisfiability. In contrast, the compression provided by the DD-based representation allowed the model building to be performed with no additional impact on the number of operations needed. This is consistent with what us learned from comparing a BDD-based decision procedure for modal logic K against search-based approaches, where the BDD-based decision procedure is also quite competitive. The structural heuristics we studied for symbolic propositional decision procedure are also successfully applied to symbolic QBF. One of the active directions in QBF solving is to extend search with methods that can reduce the amount of work needed to generate the complete strategy, either with a hybrid approach, through resolution [Bie04], or model union [Let02].

In general, while search is marked better than symbolic algorithms for propositional logic, symbolic algorithms cannot be disregarded because of its wider applicability in tackling harder logics like QBF here or modal logics [PSV02, PV03]. Some kind of structural heuristics would always be a center part of symbolic algorithms, since variable order and order of operations have a critical impact on the size of BDDs during the algorithm. But, the concern should not solely on the measure of width, as we showed that optimizing for width alone does not always give the best result.

Theoretically, we also developed a number of interesting results. The study on the relation between the size of the BDD for a formula and the width of the formula leads us to study the impact of quantification, where the impact from the width took an exponential blowup. This leads us to investigate the impact of quantification on problems under the parameterized complexity framework. Since most believe that the containment NP$\subset$PSPACE is strict, and in addition, the polynomial hierarchy does not collapse to any level, in classical complexity, quantification does make problems harder. In contrast, NP-complete problems have the same known worst-case complexity as PSPACE-complete problems, where both take exponential time. So the different complexity classes, let alone a whole hierarchy in between, do little to explain practical worst-case complexity. We gave a possible explanation based on parameterized complexity based on the width parameter, namely, a hierarchy of fixed-parameter tractable classes inside PSPACE. As long as we assume P$\neq$NP, such an hierarchy is strict, and each additional quantifier block induced an additional expoential blowup on the coefficient. Based on the hierarchy construction, we are also able to show applying an identical constraint, namely $\log^*$ width, to propositional logic and QBF resulted in drastically different behaviour. This partially contributes to the practical performance difference between NP-class and PSPACE-class problems, and suggests that the alternations in PSPACE does make problems "harder" to solve than problems in NP.

Bibliography

[AB00] A. Ayari and D. Basin. Bounded model construction for monadic second-order logics. In *Proceedings of CAV'00*, 2000.

[AFF$^+$02] Roy Armoni, Limor Fix, Alon Flaisher, Rob Gerth, Boris Ginsburg, Tomer Kanza, Avner Landver, Sela Mador-Haim, Eli Singerman, Andreas Tiemeyer, Moshe Y. Vardi, and Yael Zbar. The forspec temporal logic: A new temporal property-specification language. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 296–211, 2002.

[AM01] E. Amir and S. McIlraith. Solving satisfiability using decomposition and the most constrained subproblem. In *LICS Workshop on Theory and Applications of Satisfiability Testing (SAT 2001)*, Electronic Notes in Discrete Mathematics (Elsevier Science), June 2001.

[AMS01] F.A. Aloul, I.L. Markov, and K.A. Sakallah. MINCE: A static global variable-ordering for SAT and BDD. In *Proc. IEEE 10th International Workshop on Logic and Synthesis*, pages 281–286, June 2001.

[BAC$^+$99] A. Biere, Cimatti A, E.M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDD. In *Proc. 36th Conf. on Design Automation*, pages 317–320, 1999.

[BAHP82] M. Ben-Ari, J.Y. Halpern, and A. Pnueli. Deterministic propositional dynamic logic: Finite models, complexity, and completeness. *J. Comp. Sys. Sci.*, 25:402–417, 1982.

[Bal90] J. Balcazar. Self-reducibility. *Journal of Computer and System Sciences*, 41(3):367–388, 1990.

[BB94] D. Beatty and R. Bryant. Formally verifying a microprocessor using a simulation methodology. In *Proc. 31st Design Automation Conference*, pages 596–602. IEEE Computer Society, 1994.

[BBG$^+$94] I. Beer, S. Ben-David, D. Geist, R. Gewirtzman, and M. Yoeli. Methodology and system for practical formal verification of reactive hardware. In *Proc. 6th Conf. on CAV*, volume 818 of *LNCS*, pages 182–193, Stanford, June 1994.

[BCCZ99] A. Biere, A. Cimatti, E.M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *TACAS*, volume 1579 of *LNCS*. Springer-Verlag, 1999.

[BCL91] J.R. Burch, E.M. Clarke, and D.E. Long. Symbolic model checking with partitioned transition relations. In *VLSI 91, Proc. IFIP TC10/WG 10.5 International Conference on Very Large Scale Integration, Edinburgh, Scotland, 20-22 August, 1991*, pages 49–58, 1991.

[BCM$^+$92] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[BGP$^+$97] M. Block, C. Gröpl, H. Preuß, H. L. Proömel, and A. Srivastav. Efficient ordering of state variables and transition relation partitions in symbolic model checking. Technical report, Institute of Informatics, Humboldt University of Berlin, 1997.

[Bie04] A. Biere. Resolve and expand. In *Proc. 7th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2004)*, pages 238–246, 2004.

[BK96] H.L. Bodlaender and T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21:358–402, 1996.

[BKF95] H.K. Buning, M. Karpinski, and A. Flogel. Resolution for quantified Boolean formulas. *Inf. and Comp.*, 117(1):12–18, 1995.

[BKS03] Paul Beame, Henry A. Kautz, and Ashish Sabharwal. Understanding the power of clause learning. In *IJCAI*, pages 1194–1201, 2003.

[Bou99] F. Bouquet. *Gestion de la dynamicite et enumeration d'implicants premiers, une approche fondee sur les Diagrammes de Decision Binaire.* PhD thesis, Universite de Privence, France, 1999.

[Bry86] R.E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, Vol. C-35(8):677–691, August 1986.

[Bry91] R.E. Bryant. On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. *IEEE Transaction on Computers*, 40(2):205–213, February 1991.

[Büc60] J.R. Büchi. Weak second-order arithmetic and finite automata. *Zeit. Math. Logik und Grundl. Math.*, 6:66–92, 1960.

[CB94] J.M. Crawford and A.B. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proc. 12th Nat. Conf. on Artificial Intelligence*, volume 2, pages 1092–1097, 1994.

[CDS$^+$00] C. Coarfa, D. D. Demopoulos, A. San Miguel Aguirre, D. Subramanian, and M.Y. Vardi. Random 3-SAT: The plot thickens. In *Proc. 6th Int. Conf. Constraint Programming (CP 2000)*, 2000.

[CDS$^+$03] C. Coarfa, D. D. Demopoulos, A. San Miguel Aguirre, D. Subramanian, and M.Y. Vardi. Random 3-SAT: The plot thickens. *Constraints*, pages 243–261, 2003.

[CEA86] E.M. Clarke, E.A. Emerson, and A.P.Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Prog. Lan. Sys.*, 8:244–263, 1986.

[Che04] H. Chen. Quantified constraint satisfaction and bounded treewidth. In Ramon López de Mántaras and Lorenza Saitta, editors, *ECAI*, pages 161–165. IOS Press, 2004.

[Chl86] B. Chlebus. Domino-tiling games. *J. Comp. Sys. Sci.*, 32:374–392, 1986.

[CHP93] P. Chung, I. Hajj, and J. Patel. Efficient variable ordering heuristics for shared ROBDD. In *Proc. 1993 IEEE Int. Symp. on Circuits and Systems (ISCAS93)*, pages 1690–1693, 1993.

[CKS81] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, January 1981.

[Coo71] S. A. Cook. The complexity of theorem proving procedures. In *Proc. 3rd ACM Symp. on Theory of Computing*, pages 151–158, 1971.

[CR00] A. Cimatti and M. Roveri. Conformant planning via symbolic model checking. *J. of AI Research*, 13:305–338, 2000.

[CS00] P. Chatalic and L. Simon. Multi-Resolution on compressed sets of clauses. In *Twelfth International Conference on Tools with Artificial Intelligence (ICTAI'00)*, pages 2–10, 2000.

[CSGG02] M. Cadoli, M. Schaerf, A. Giovanardi, and M. Giovanardi. An algorithm to evaluate quantified Boolean formulae and its experimental evaluation. *J. of Automated Reasoning*, 28(2):101–142, 2002.

[Dec03] R. Dechter. *Constraint Processing*. Morgan Kaufmman, 2003.

[DF99] R.G. Downey and M.R. Fellows. *Parametrized Complexity*. Springer-Verlag, 1999.

[DHK05] N. Dershowitz, Z. Hanna, and J. Katz. Bounded model checking with QBF. pages 408–414, 2005.

[DK03] Robert F. Damiano and James H. Kukula. Checking satisfiability of a conjunction of BDDs. In *Proc. 40th Design Automation Conference (DAC 2003)*, pages 818–823, 2003.

[DKV02] V. Dalmau, P.G. Kolaitis, and M.Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *Proceedings of 8th Int. Conf. on Principles and Practice of Constraint Programming (CP 2002)*, pages 310–326, 2002.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *J. ACM*, 5:394–397, 1962.

[DP60] S. Davis and M. Putnam. A computing procedure for quantification theory. *J. ACM*, 7:201–215, 1960.

[DP87] R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34:1–38, 1987.

[DP89] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, pages 353–366, 1989.

[DR94] R. Dechter and I. Rish. Directional resolution: The Davis-Putnam procedure, revisited. In *KR'94: Principles of Knowledge Representation and Reasoning*, pages 134–145. 1994.

[EGG] T. Eiter, G. Gottlob, and Y. Gurevich. Existential second-order logic over strings.

[Fag74] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. *SIAM-AMS Proceedings*, 7:43–73, 1974.

[FFK88] M. Fujita, H. Fujisawa, and N. Kawato. Evaluation and improvements of Boolean comparison method based on binary decision disgrams. In *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD-88)*, pages 2–5, 1988.

[FG02] M. Frick and M. Grohe. The complexity of first-order and monatic second-order logic revisited. In *LICS'02*, pages 215–224, 2002.

[FPV05] A. Ferrara, G. Pan, and M.Y. Vardi. Treewidth in verification: Local vs. global. In *LPAR 2005*, pages 489–503, 2005.

[Fre] E.C Freuder. Complexity of k-tree structured constraint satisfaction problems.

[GB94] D. Geist and H. Beer. Efficient model checking by automated ordering of transition relation partitions. In *Proc. 6th Int. Conf. on Computer Aided Verification (CAV 1994)*, pages 299–310, 1994.

[GMTZ01] E. Giunchiglia, M. Maratea, A. Tacchella, and D. Zambonin. Evaluating search heuristics and optimization techniques in propositional satisfiability. *Lecture Notes in Computer Science*, 2083, 2001.

[GN02] E. Goldberg and Y. Novikov. BerkMin: A fast and robust SAT solver. In *Proc. Design Automation and Test in Europe (DATE 2002)*, pages 142–149, 2002.

[GNT01] E. Giunchiglia, M. Narizzano, and A. Tacchella. QuBE, a system for deciding quantified Boolean formulae satisfiability. In *IJCAR'01*, 2001.

[GPVW95] R. Gerth, D. Peled, M.Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification Testing and Verification*, pages 3–18, 1995.

[Gro96] J. F. Groote. Hiding propositional constants in BDDs. *FMSD*, 8:91–96, 1996.

[HD03] Jinbo Huang and Adnan Darwiche. A structure-based variable ordering heuristic for SAT. In *Proc. 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, pages 1167–1172, 2003.

[HD04] J. Huang and A. Darwiche. Using DPLL for efficient OBDD construction. In *Proc. 7th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2004)*, 2004.

[HKB96] R. Hojati, S. C. Krishnan, and R. K. Brayton. Early quantification and partitioned transition relations. In *Proc. 1996 Int. Conf. on Computer Design (ICCD '96)*, pages 12–19, 1996.

[HS96] A. Heuerding and S. Schwendimann. A benchmark method for the propositional modal logics K, KT, S4. Technical report, Universität Bern, Switzerland, 1996.

[Imm99] N. Immerman. *Descriptive Complexity.* Graduate Texts in Computer Science. Springer, 1999.

[JS04] H.S. Jon and F. Somenzi. CirCUs : Hybrid satifiability solver. In *Proc. of the 7th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2004)*, pages 47–55, May 2004.

[KBv01] A.M.C.A. Koster, H.L. Bodlaender, and S.P.M. van Hoesel. Treewidth: Computational experiments. Technical report, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 2001.

[Klo94] T. Kloks. Treewidth. In *Computations and Approximations*, 1994.

[KS92] H. Kautz and B. Selman. Planning as satisfiability. In *Proc. 10th Eur. Conf. on AI (ECAI 92)*, pages 359–363, 1992.

[Let02] R. Letz. Lemma and model caching in decision procedures for quantified Boolean formulas. In *TABLEAUX 2002*, pages 160–175, 2002.

[Leu02] M. Leucker. Prefix-recognizable graphs and monadic logic. In E. Grädel, W. Thomas, and T. Wilke, editors, *Automata, Logics, and Infinite Games*. Springer-Verlag, 2002.

[Lew78] H.R. Lewis. Complexity of solvable cases of the decision problem for the predicate calculus. In *FOCS 1978*, pages 35–47, 1978.

[LS03] D. Le Berre and L. Simon. The essentials of the SAT'03 competition. In *Proc. 6th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2003)*, pages 452–467, 2003.

[LST03] D. Le Berre, L. Simon, and A. Tacchella. Challenges in the QBF arena: the SAT'03 evaluation of QBF solvers. In *Proc. 6th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, pages 468–485, 2003.

[McM93] K.L. McMillan. *Symbolic Model Checking*. Kluwer Acad. Pub., 1993.

[Mei89] C. Meinel. *Modified Branching Programs and Their Computational Power*. Springer-Verlag, 1989.

[Min96] S. Minato. *Binary Decision Diagrams and Applications to VLSI CAD*. Kluwer, 1996.

[MM02a] D. B. Motter and I. L. Markov. A compressed breadth-first search for satisfiability. In *Proc. 4th Int. Workshop on Algorithm Engineering and Experiments (ALENEX 2002)*, volume 2409 of *Lecture Notes in Computer Science*, pages 29–42, 2002.

[MM02b] D. B. Motter and I. L. Markov. On proof systems behind efficient SAT solvers. In *Proc. of 5th Int. Symp. on the Theory and Applications of Satisfiability Testing (SAT 2002)*, pages 206–213, May 2002.

[MMZ$^+$01] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proc. of 39th Design Automation Conference (DAC 2001)*, pages 530–535, June 2001.

[MWBSV88] S. Malik, A. Wang, R. Brayton, and A. Sangiovanni Vincentelli. Logic verification using binary decision diagrams in a logic synthesis environment. In *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD-88)*, pages 6–9, 1988.

[Nar] M. Narizzano. QBFLIB, the quantified Boolean formulas satisfiability library. http://www.qbflib.org.

[PBG05] M.R. Prasad, A. Biere, and A. Gupta. A survey of recent advances in sat-based formal verification. *STTT*, 7:156–173, 2005.

[PSV02] G. Pan, U. Sattler, and M.Y. Vardi. BDD-based decision procedures for K. In *Proc. of CADE 2002*, volume 2392 of *LNAI*, pages 16–30, 2002.

[PV03] G. Pan and M.Y. Vardi. Optimizing a symbolic modal solver. In *Proc. of CADE 2003*, 2003.

[PV04a] G. Pan and M.Y. Vardi. Search vs. symbolic techniques in satisfiability solving. In *Proc. 7th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2004)*, pages 137–146, 2004.

[PV04b] Guoqiang Pan and Moshe Y. Vardi. Symbolic decision procedures for QBF. In *Proceedings of 10th Int. Conf. on Principles and Practice of Constraint Programming (CP 2004)*, pages 453–467, 2004.

[PV06] G. Pan and M.Y. Vardi. Fixed-parameter hierarchies in PSPACE. In *LICS*, 2006.

[RAB$^+$95] R. Ranjan, A. Aziz, R. Brayton, B. Plessier, and C. Pixley. Efficient BDD algorithms for FSM synthesis and verification. In *Proc. of IEEE/ACM Int. Workshop on Logic Synthesis*, 1995.

[Rin99] J. Rintanen. Constructing conditional plans by a theorem-prover. *J. of A. I. Res.*, 10:323–352, 1999.

[RS86] N. Robertson and P.D. Seymour. Graph minors. ii. algorithmic aspects of treewidth. *J. of Algorithms*, 7:309–322, 1986.

[Sav70] W.J. Savitch. Relationship between nondeterministic and deterministic tape complexities. *J. Comp. Sys. Sci.*, 4:177–192, 1970.

[Sch78] T. Schaefer. The complexity of satisfiability problems. In *Proc. of the 10th annual ACM symposium on Theory of computing(STOC'78)*, pages 216–226, 1978.

[SML96] B. Selman, D. G. Mitchell, and H. J. Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81(1-2):17–29, 1996.

[Som98] F. Somenzi. CUDD: CU decision diagram package. http://vlsi.colorado.edu/~fabio/CUDD/, 1998.

[SS01] G. Sutcliffe and C. Suttner. Evaluating general purpose automated theorem proving systems. *Artificial intelligence*, 131:39–54, 2001.

[Sto74] L.J. Stockmeyer. *The complexity of decision problems in automate theory and logic*. PhD thesis, Dept. of Elec. Eng., MIT, 1974.

[Sto76] L. Stockmeyer. The polynomial-time hierarchy. *Theo. Comp. Sci.*, 3:1 – 22, 1976.

[SV01] A. San Miguel Aguirre and M. Y. Vardi. Random 3-SAT and BDDs: The plot thickens further. In *Proc. of the 7th Int. Conf. Principles and Practice of Constraint Programming (CP 2001)*, pages 121–136, 2001.

[SVW87] A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automana with applications to temporal logic. *Theo. Comp. Sci.*, 49:217–237, 1987.

[Tse81] G.S. Tseitin. *On the complexity of derivation of propositional calculus*, pages 466–483. Springer, 1981.

[TY84] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to tests chordality of graphs, tests acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, 1984.

[Urq95] A. Urquhart. The complexity of propositional proofs. *the Bulletin of Symbolic Logic*, 1:425–467, 1995.

[US94] T. E. Uribe and M. E. Stickel. Ordered binary decision diagrams and the Davis-Putnam procedure. In *1st Int. Conf. on Constraints in Computational Logics*, pages 34–49, 1994.

[Var82] M.Y. Vardi. The complexity of relational query languages. In *Proc. Sym. Theory. of Comp.*, 1982.

[VW94] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 15:1 – 37, 1994.

[Zha97] H. Zhang. SATO: An efficient propositional prover. In *Proc. of International Conference on Automated Deduction (CADE-97)*, pages 272–275, 1997.

[ZM02a] L. Zhang and S. Malik. The quest for efficient Boolean satisfiability solvers. In *Proc. 14th Int. Conf. on Computer Aided Verification (CAV 2002)*, pages 17–36, 2002.

[ZM02b] L. Zhang and S. Malik. Towards symmetric treatment of conflicts and satisfaction in quantified Boolean satisfiability solver. In *Proceedings of 8th Int. Conf. on Principles and Practice of Constraint Programming (CP 2002)*, pages 200–215, 2002.