Quantifier Elimination via Functional Composition

Jie-Hong R. Jiang

Department of Electrical Engineering / Graduate Institute of Electronics Engineering
National Taiwan University, Taipei 10617, Taiwan
jhjiang@cc.ee.ntu.edu.tw

Abstract. This paper poses the following basic question: Given a quantified Boolean formula $\exists x.\varphi$, what should a function/formula f be such that substituting f for x in φ yields a logically equivalent quantifier-free formula? Its answer leads to a solution to quantifier elimination in the Boolean domain, alternative to the conventional approach based on formula expansion. Such a composite function can be effectively derived using symbolic techniques and further simplified for practical applications. In particular, we explore Craig interpolation for scalable computation. This compositional approach to quantifier elimination is analyzably superior to the conventional one under certain practical assumptions. Experiments demonstrate the scalability of the approach. Several large problem instances unsolvable before can now be resolved effectively. A generalization to first-order logic characterizes a composite function's complete flexibility, which awaits further exploitation to simplify quantifier elimination beyond the propositional case.

1 Introduction

Quantifier elimination is a way of transforming a formula with quantifiers to an equivalent one without quantifiers. Eliminating quantified variables often yields desirable reduction of some sort, and applies to constraint solving, e.g., Gauss elimination for solving systems of linear equations, Fourier-Motzkin elimination for systems of linear inequalities, cylindrical algebraic decomposition [4, 2] for systems of polynomial inequalities, and so on. It plays important roles in computation theory, mathematical logic, mathematical programming, scientific computing, and other fields. Its applications are pervasive and profound. This paper is concerned about quantifier elimination in propositional logic as well as first-order logic.

Quantifier elimination in propositional logic augmented with quantifiers over propositional variables is a well-studied subject. There are several approaches to this problem:

Formula expansion. A conventional approach to quantifier elimination is based on formula expansion, $\exists x.\varphi = \varphi[x/0] \vee \varphi[x/1]$, where formula φ is expanded under all truth assignments on x by substituting 0 and 1 for x in φ. Binary decision diagrams (BDDs), and-inverter graphs (AIGs), and other data structures for Boolean function representation and manipulation can be adopted for the computation. BDDs tended to be a popular approach to such computation, the so-called image computation [3]. BDD-based computation however has its intrinsic memory-explosion limitation. On the other hand, recent progress in AIG packages [14, 13] has made AIG-based quantification a viable alternative to BDD-based one. For example, AIGs have been directly used in unbounded model checking [17]. Our approach to quantifier elimination also uses AIGs extensively.

Normal-form conversion. Eliminating the existential (respectively universal) quantifier of formula $\exists x.\varphi$ (respectively $\forall x.\varphi$) is easy[1] when φ is in disjunctive normal form (DNF) (respectively conjunctive normal form (CNF)). Thereby the normal-form conversion between CNF and DNF can be exploited for quantifier elimination, as was suggested in [15] in application to unbounded model checking.

Satisfiability solving. Using a decision procedure, the quantifier-free equivalent of $\exists x.\varphi$ can be generated by searching through all satisfying assignments to the non-quantified variables. By collecting these satisfying assignments, one can construct an equivalent quantifier-free formula. A detailed exposition of this method can be found, e.g., in [7] and the references therein.

Despite these existing approaches, there is not a single best one to quantifier elimination for all problem instances. Different approaches may have their own strengths and weaknesses.

This paper adds to the above list a new item, a compositional approach to quantifier elimination, which is by nature closer to the formula-expansion approach. We ask, given a quantified Boolean formula $\exists x.\varphi$, what should a function/formula f be such that substituting f for x in φ, denoted as $\varphi[x/f]$, yields a logically equivalent quantifier-free formula. This paper characterizes the complete flexibility of such a composite function f. Furthermore, an effective and scalable derivation of f with reasonable quality is proposed using Craig interpolation. An analysis shows that, under the sparsity assumption of φ (which is common in certain practical applications), the new compositional approach is superior to the conventional one based on formula expansion. Practical experience suggests that the new approach often yields much more compact AIGs than the conventional one when the sparsity condition holds. Several problem instances that suffer from exponential blow-up by formula expansion are effectively resolvable by functional composition.

Quantifier elimination in first-order logic is much more complicated and relatively less explored. In fact, exhaustive formula expansion does not work in first-order logic as variables can take on infinite values. Moreover, not every

[1] When φ is in DNF and CNF, respectively, removing every appearance of literals x, $\neg x$ and the so-induced illegal logic connectives from φ yields a quantifier-free equivalent of $\exists x.\varphi$ and $\forall x.\varphi$, respectively. So the computation is achievable in linear time and the size of the resultant quantifier-free formula is non-increasing compared to that of φ.

first-order theory allows quantifier elimination. One of the earliest attempts at quantifier elimination in first-order logic is Tarski's work [21], where the first quantifier-elimination procedure of real closed fields was demonstrated. Since then, algorithmic improvements have been achieved, see, e.g., [4, 2]. Also quantifier elimination has been shown possible in other first-order theories, such as term algebras, Presburger arithmetic, and other theories [20].

Extending the results of propositional logic, this paper characterizes the complete flexibility of composite functions for quantifier elimination in first-order logic. Unlike most prior efforts, which gave concrete quantifier-elimination procedures for some specific theories, we rather present a generic viewpoint and show the potential usefulness of the complete flexibility in simplifying quantifier elimination.

One of the common practices to quantifier elimination of first-order theories is based on the principle of *virtual substitution* with *elimination sets* [23]. To eliminate a quantifier, a finite set of solution terms with their validity conditions is identified for substitution. Our characterized complete flexibility may be exploited to reduce elimination sets.

This paper is organized as follows. After preliminaries are given in Section 2, Section 3 presents the main results on quantifier elimination in propositional logic. Section 4 extends the results to first-order logic. Experimental results and discussions are given in Section 5. Section 6 compares our results with some related work. Finally, Section 7 concludes this paper and outlines future work.

2 Preliminaries

Predicate logic. We closely follow the definitions and notation of [6] about first-order logic. A *first-order language* may consist of logical symbols (including parentheses, sentential connectives, variables, and the (optional) equality symbol) and parameters (including quantifier symbols, constant symbols, predicate symbols, and function symbols). Given a language, *terms* are finite expressions representing names of objects, whereas *(well-formed) formulas* are finite expressions representing assertions about objects. Given a formula, variables not in the scope of any quantifier are called *free variables*, otherwise *bound variables.* Formulas without free variables are called *sentences.*

As a notational convention, substituting a term t for some variable x in a formula φ is denoted as $\varphi[x/t]$. We say that t is *substitutable* for x in φ if every variable y in t is not captured by some quantifier $\forall y$ or $\exists y$ in φ. Substitutability can be achieved through proper renaming of bound variables. By writing $\varphi[x/t]$, this paper assumes that t is substitutable for x in φ.

A *structure* (or called an *interpretation*) $\mathfrak{A}$ of some first-order language $\mathcal{L}$ is a tuple specifying the *domain* (or *universe*), denoted $|\mathfrak{A}|$, of the variables, and associating the constant, predicate, and function symbols with meanings. A sentence σ of $\mathcal{L}$ is true in structure $\mathfrak{A}$ is denoted as $\models_{\mathfrak{A}} \sigma$.

Propositional logic can be seen as a special case of first-order logic, where functions and predicates are interchangeable, so are terms and formulas. Also there is a unique structure $\mathfrak{B}$ with $|\mathfrak{B}| = \{0, 1\}$. In propositional logic, the *positive* and *negative cofactors* of formula φ with respect to variable x are $\varphi[x/1]$ and $\varphi[x/0]$, respectively.

Propositional satisfiability and Craig interpolation. A brief introduction to SAT solving and circuit-to-CNF conversion, essential to our development, can be found in [16]. To introduce terminology and convention for later use, we restate the following theorem.

Theorem 1 (Craig Interpolation Theorem). *[5]*
Given two Boolean formulas ϕ_A and ϕ_B, with $\phi_A \wedge \phi_B$ unsatisfiable, then there exists a Boolean formula ψ_A referring only to the common variables of ϕ_A and ϕ_B such that $\phi_A \rightarrow \psi_A$ and $\psi_A \wedge \phi_B$ remains unsatisfiable.

The Boolean formula ψ_A is referred to as the *interpolant* of ϕ_A with respect to ϕ_B. Modern SAT solvers can be extended to construct interpolants from resolution refutations [16]. In the sequel, we shall assume that Boolean functions, circuits, and interpolants are represented using AIGs, which can be converted to CNF formulas in polynomial time.

3 Propositional Logic

In this section we consider quantifier elimination of propositional logic augmented with quantifiers over propositional variables.

3.1 Composability for Quantifier Elimination

Putting propositional logic in the context of first-order logic, we note that it has a unique structure/interpretation. Under this unique structure, terms, functions, predicates, and formulas all coincide. This simplicity is crucial to the following development for propositional logic, and will become apparent when we encounter first-order logic.

Theorem 2. *A quantified Boolean formula $\exists y.\varphi(\boldsymbol{x}, y)$ is logically equivalent to the quantifier-free formula $\varphi(\boldsymbol{x}, f(\boldsymbol{x}))$ for some function $f : \mathbb{B}^n \rightarrow \mathbb{B}$ if and only if every pair $a \in \mathbb{B}^n, b \in \mathbb{B}$ with $f(a) = b$ satisfies $\varphi(a, b) \vee \forall y.\neg\varphi(a, y)$.*

Proof. ($\Longrightarrow$) For $f(a) = b$, then $\varphi(a, f(a)) = \varphi(a, b)$. If $\varphi(a, b)$ is true, then $\exists y.\varphi(a, y)$ is true. On the other hand, if $\neg\varphi(a, b)$ is true, then $\neg\exists y.\varphi(a, y)$ is true due to the logical equivalence between $\varphi(\boldsymbol{x}, f(\boldsymbol{x}))$ and $\exists y.\varphi(\boldsymbol{x}, y)$. Hence every pair $a \in \mathbb{B}^n, b \in \mathbb{B}$ with $f(a) = b$ satisfies $\varphi(a, b) \vee \forall y.\neg\varphi(a, y)$.

($\Longleftarrow$) For the sake of contradiction, assume $\exists y.\varphi(\boldsymbol{x}, y)$ and $\varphi(\boldsymbol{x}, f(\boldsymbol{x}))$ are not logically equivalent. Then there exists some $a \in \mathbb{B}^n$ such that $\neg\exists y.\varphi(a, y) \wedge \varphi(a, f(a))$ or $\exists y.\varphi(a, y) \wedge \neg\varphi(a, f(a))$. The former is trivially unsatisfiable; the latter contradicts with the premise, $\varphi(a, f(a)) \vee \forall y.\neg\varphi(a, y)$. ■

In essence the above theorem answers the following question: Given a quantified Boolean formula $\exists y.\varphi(\boldsymbol{x}, y)$, what should a function f be such that the composition $\varphi(\boldsymbol{x}, f(\boldsymbol{x}))$ equals $\exists y.\varphi$? It also implies the existence of such a function.

Proposition 1. *Given a quantified Boolean formula $\exists y.\varphi(\boldsymbol{x}, y)$, there always exists a function $f(\boldsymbol{x})$ such that $\exists y.\varphi(\boldsymbol{x}, y) = \varphi(\boldsymbol{x}, f(\boldsymbol{x}))$.*

Proof. The proposition follows from the fact that, for every $a \in \mathbb{B}^n$, there always exists some $b \in \mathbb{B}$ such that $\varphi(a, b) = 1$ if $\exists y.\varphi(a, y)$ is true. ■

The following proposition characterizes the complete flexibility of a composite function for quantifier elimination.

Proposition 2. *The equality $\exists y.\varphi(\boldsymbol{x}, y) = \varphi(\boldsymbol{x}, f(\boldsymbol{x}))$ holds if and only if the composite function f satisfies $(\neg\varphi[y/0] \wedge \varphi[y/1]) \rightarrow f$ and $f \rightarrow \neg(\neg\varphi[y/1] \wedge \varphi[y/0])$. That is, $\neg\varphi[y/0] \wedge \varphi[y/1]$ and $\neg\varphi[y/1] \wedge \varphi[y/0]$ are the tightest onset and offset of f, respectively.*

Proof. There are four possible valuations of $\varphi(a, 0)$ and $\varphi(a, 1)$ for every $a \in \mathbb{B}^n$.

For $(\varphi(a, 0), \varphi(a, 1)) = (0, 0)$, a is a don't-care minterm of f because $\varphi(a, f(a)) = 0$ independent of the value of $f(a)$.
For $(\varphi(a, 0), \varphi(a, 1)) = (0, 1)$, a is an onset minterm of f because $\exists y.\varphi(a, y)$ is true and $f(a) = 1$ is the only way to make $\varphi(a, f(a))$ true.
For $(\varphi(a, 0), \varphi(a, 1)) = (1, 0)$, a is an offset minterm of f for reason similar to that of case (0, 1).
For $(\varphi(a, 0), \varphi(a, 1)) = (1, 1)$, a is a don't-care minterm of f for reason similar to that of case (0, 0).

Hence $\neg\varphi[y/0] \wedge \varphi[y/1]$ and $\neg\varphi[y/1] \wedge \varphi[y/0]$ are the tightest onset and offset of f, respectively. ■

Therefore the composite function f can be minimized using the don't-care condition $(\varphi[y/1] \wedge \varphi[y/0]) \vee (\neg\varphi[y/1] \wedge \neg\varphi[y/0])$.

Since universal quantification can be converted to existential quantification by the equality $\forall x.\varphi = \neg\exists x.\neg\varphi$, the compositional approach can be used in general quantifier elimination of quantified Boolean formulas (QBFs). The quantifiers of a QBF can be removed from inside out.

3.2 Interpolation of Composite Function

By Proposition 2, the composite function f can be obtained using symbolic methods. Finding a simple implementation of f under the don't-care flexibility hopefully makes $\varphi(\boldsymbol{x}, f(\boldsymbol{x}))$ simple and facilitates quantifier elimination. We exploit Craig interpolation for scalable computation. It relies on the following proposition, whose correctness is immediate by Theorem 1.

Proposition 3. *The interpolant with respect to*

$$\phi_A = \neg\varphi[y/0] \wedge \varphi[y/1] \text{ and} \tag{1}$$

$$\phi_B = \neg\varphi[y/1] \wedge \varphi[y/0] \tag{2}$$

is a valid implementation of a composite function f satisfying $\exists y.\varphi = \varphi[y/f]$.

Interpolation can be seen as a way to derive simple functions as long as the don't-care set is reasonably large. When the don't-care set is large, proving the unsatisfiability of $\phi_A \wedge \phi_B$ is easy and the corresponding refutation proof is simple. So the interpolant size (in term of AIG nodes) is likely to be small.

3.3 Analysis of Applicability

We compare expansion- and composition-based quantifier-elimination procedures assuming that AIGs are the underlying data structure. The AIG sizes of $\varphi[y/0] \vee \varphi[y/1]$ and $\varphi[y/f(\boldsymbol{x})]$ are analyzed.

The AIGs of $\varphi[y/0]$ and $\varphi[y/1]$ are obtained from φ through constant propagation. From practical experience, the sizes of $\varphi[y/0]$ and $\varphi[y/1]$ are rarely reduced much, especially for large AIGs. It is possible to apply aggressive minimization using don't cares. Specifically, in building $\varphi[y/0] \vee \varphi[y/1]$, $\varphi[y/0]$ can be used as the don't-care condition to minimize $\varphi[y/1]$, or vice versa (but simultaneous minimization of $\varphi[y/0]$ and $\varphi[y/1]$ is forbidden). For instance, in minimizing $\varphi[y/1]$ using don't-care condition $\varphi[y/0]$, the optimization is constrained by

$$\phi_A = \varphi[y/1] \wedge \neg\varphi[y/0] \text{ and} \tag{3}$$
$$\phi_B = \neg\varphi[y/1] \wedge \neg\varphi[y/0] \tag{4}$$

being the tightest onset and offset, respectively. Notice that interpolation or other symbolic techniques can be applied here to extract a function, say f', hopefully simpler than $\varphi[y/1]$. So $\varphi[y/0] \vee f'$ can be simpler than $\varphi[y/0] \vee \varphi[y/1]$ for quantifier elimination. (With interpolation, such simplification however was not empirically observed in our experiments. It may be due to the small size of the don't-care set, which results in ineffective interpolation.)[2]

Observe that Equations (1) and (3) are identical, whereas Equations (2) and (4) differ only in the second term. This slight difference in fact makes substantial impact on interpolation. When φ is a sparse function (with relatively few onset minterms), $\varphi[y/0]$ is very likely a sparse function as well. In this case, the offset corresponding to ϕ_B of Equation (2) is much smaller than that of Equation (4). Accordingly, proving the unsatisfiability of $\phi_A \wedge \phi_B$ of Equations (1) and (2) is easier to establish than that of Equations (3) and (4). The derived interpolant with respect to Equations (1) and (2) can be much smaller. On the contrary, for dense function φ the derived interpolant with respect to Equations (3) and (4) can be smaller.

The above sparsity condition commonly holds in practical applications. For instance, the transition relation built up from a set of transition functions of a sequential system appears to be sparse. In fact, the more the transition functions

[2] Practical experience suggests that the size of an interpolant can be sensitive to the amount of don't cares. It was observed that, for a function f, the AIG size of the interpolant of $\phi_A = f$ and $\phi_B = \neg f$ (i.e., interpolation without don't cares) is typically much (e.g., two orders of magnitude) larger than that of f. Therefore, quantifier elimination of $\exists \boldsymbol{y}.\varphi(\boldsymbol{x}, \boldsymbol{y})$ by interpolating $\phi_A = \varphi(\boldsymbol{x}, \boldsymbol{y})$ and $\phi_B = \neg\exists \boldsymbol{y}.\varphi(\boldsymbol{x}, \boldsymbol{y})$ is ineffective.

are, the sparser the transition relation is. Under this sparsity assumption, quantifier elimination using functional composition is superior to that using formula expansion.

By a qualitative comparison, expansion- and composition-based quantifier-elimination methods show different characteristics:

Manipulation complexity. The former requires cofactoring and disjunction operations; the latter requires interpolation (which invokes SAT solving) and composition operations. In addition to the above operations, for both methods AIG minimization also plays an important role in the entire quantification effort.

Circuit level. The circuit depth of a resultant AIG is shallower for the former and deeper for the latter. On the other hand, the circuit width of a resultant AIG is thicker for the former and thinner for the latter.

Circuit size. The AIG resulted from the former is often larger than that of the latter in certain applications. This phenomenon can be related to the sparsity assumption and due to the amount of achieved logic sharing.

3.4 Application to Circuit Optimization

In addition to quantifier elimination, a potential application of (the "if"-part of) Theorem 2 is to reduce circuit levels. Consider a circuit C implementing some function $f(X)$. Suppose t is an intermediate signal in the circuit with function $g(X)$ and $f(X) = h(X, t) = h(X, g(X))$. If $g(X)$ satisfies Theorem 2, then the circuit can be reexpressed by $h[t/0] \vee h[t/1]$, whose circuit level can be potentially smaller than that of $h[t/g(X)]$.

4 Predicate Logic

We study quantifier elimination in predicate logic with the following principle.

Proposition 4. *Given a language $\mathcal{L}$ in predicate logic and a structure $\mathfrak{A}$, then*

$$\models_{\mathfrak{A}} \forall \boldsymbol{x}.(\exists y.\varphi(\boldsymbol{x}, y) = \exists F.\varphi(\boldsymbol{x}, F(\boldsymbol{x}))),$$

where φ is a formula, F is an n-place function symbol, and $\boldsymbol{x} = (x_1, \ldots, x_n)$ and y are variable symbols of $\mathcal{L}$.

Proof. By the axiom of choice, such a function can be obtained by letting $f(a) = b$ for every a with some b satisfying $\varphi(a, b)$ or some arbitrary b if $\forall y.\neg\varphi(a, y)$. ■

Note that the above proof characterizes the complete flexibility of the composite function in predicate logic.

The equality of Proposition 4 suggests an equivalence-preserving transformation (with respect to some structure), and should be distinguished from the satisfiability-preserving Skolemization [18] of

$$\forall \boldsymbol{x}, \exists y.\varphi(\boldsymbol{x}, y) \models\!=\!| \exists F, \forall \boldsymbol{x}.\varphi(\boldsymbol{x}, F\boldsymbol{x}).$$

Unlike propositional logic, the nice coincidence of terms, functions, predicates, and formulas no longer holds in predicate logic. In fact all of them are distinct. Terms are built up from constant symbols, variables, and function symbols. They represent names of objects and should be distinguished from functions. Substituting terms for variables is legitimate, but substituting functions or formulas for variables is not. Quantifier elimination through substitution is achievable only when a function can be conditionally expressed by a finite set of terms. Therefore quantifier elimination by exhaustive formula expansion does not work in predicate logic. Different from propositional logic, a function in predicate logic may not be always expressible with a single term, and sometimes not even finitely expressible. With these differences in mind, we generalize Theorem 2 in the context of predicate logic as Theorems 3 and 4.

Theorem 3. *Given a first-order language $\mathcal{L}$ and a structure $\mathfrak{A}$, if a formula $\exists y.\varphi(\boldsymbol{x}, y)$ is equivalent to*

$$\varphi[y_i/t_f] = \varphi(\boldsymbol{x}, t_f),$$

by substituting for y some term t_f (finitely expressible in the language) that represents $f(\boldsymbol{x})$ for some function $f : |\mathfrak{A}|^n \to |\mathfrak{A}|$, then $\varphi(a, b) \vee \neg\exists y.\varphi(a, y)$ is satisfied for any $a \in |\mathfrak{A}|^n, b \in |\mathfrak{A}|$ with $f(a) = b$.

Unlike the necessary and sufficient condition of Theorem 2, the converse of Theorem 3 is not true because in general the composite function f may not be finitely expressible in the language. For finitely expressible f, however, the converse holds by Theorem 4.

Theorem 4. *Given a formula $\varphi(\boldsymbol{x}, y)$ of some first-order language $\mathcal{L}$ and a structure $\mathfrak{A}$, if a function $f : |\mathfrak{A}|^n \to |\mathfrak{A}|$ with $f(a) = b$ satisfying $\varphi(a, b) \vee \neg\exists y.\varphi(a, y)$ is finitely expressible in the language with*

$$f = \begin{cases} f_1 & \text{if } \gamma_1 \\ & \vdots \\ f_m & \text{if } \gamma_m \end{cases}$$

such that each f_i can be expressed with some term t_{f_i}, where guard γ_i is the predicate defining the applicability of f_i over $|\mathfrak{A}|^n$, then the quantified formula $\exists y.\varphi(\boldsymbol{x}, y)$ is equivalent to

$$\bigvee_i \gamma_i \wedge \varphi(\boldsymbol{x}, t_{f_i}).$$

The above theorems can be applied for universal quantifier elimination by $\forall x.\varphi = \neg\exists x.\neg\varphi$, and thus work for nested quantifier elimination.

With the following examples, we illustrate the usefulness of the complete flexibility of a composite function to simplify quantifier elimination.

Example 1. Consider the first-order language $\mathcal{L}_G$ with equality, 1-place function symbol S, 2-place predicate symbol R. Let structure $\mathfrak{A} = (\{0, \ldots, 4\}; S$ (successor

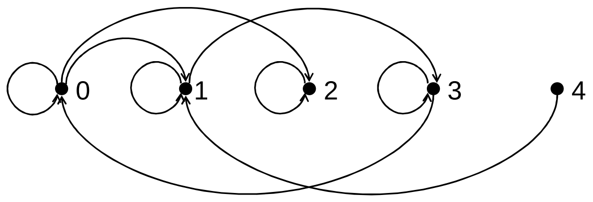

Fig. 1. Graph defined by the first-order language $\mathcal{L}_G$

function modulo 5), $R = \{(0,0),(1,1),(2,2),(3,3),(0,1),(0,2),(1,3),(3,0),(4,1)\})$. The graph induced by the structure $\mathfrak{A}$ is shown in Figure 1, where every element of the universe $\{0,\dots,4\}$ is represented as a vertex, and every $(u,v) \in R$ is represented as a directed edge from u to v. Let φ be

$$\exists y.((y \neq x) \wedge R(x,y) \wedge R(y,y)).$$

Then the (x,y)-values satisfying $((y \neq x) \wedge R(x,y) \wedge R(y,y))$ have the property that vertex x connects to a different vertex y that has a self-loop. So (x,y)-values are as follows.

x	y
0	1, 2
1	3
2	$\emptyset$
3	0
4	1

By Theorem 4, a solution function f can be as follows.

x	$f(x)$
0	2
1	3
2	4
3	0
4	1

In this case, f can be expressed in terms of S as $f(x) = S(S(x))$. So φ can be transformed into the equivalent quantifier-free formula

$$(S(S(x)) \neq x) \wedge R(x, S(S(x))) \wedge R(S(S(x)), S(S(x))).$$

Example 2. Consider the following formula φ in the language of number theory under structure $\mathfrak{N} = (\mathbb{N}; 0, S, S^{-1}, <, +, \cdot, E)$, where S^{-1} is the inverse of the successor function S with $S^{-1}(0) = 0$.

$$\begin{aligned}(S(0) < x) \wedge (x < y) \wedge (\forall a, \forall b.(y = a \cdot b \rightarrow (a = S(0) \vee b = S(0)))\\ \rightarrow \forall a, \forall b.(x = a \cdot b \rightarrow (a = S(0) \vee b = S(0))))\end{aligned}$$

φ asserts that x is greater than 1, y is greater than x, and, if y is prime, then x is prime. By the property of Mersenne numbers, the quantification of $\exists y.\varphi$ can be eliminated by substituting $2^x - 1$, shorthand for $S^{-1}(S(S(0))Ex)$, for y in φ as

$$\begin{aligned}(S(0) < x) \wedge (x < (2^x - 1)) \wedge (\forall a, \forall b.((2^x - 1) = a \cdot b \rightarrow (a = S(0) \vee b = S(0))) \\ \rightarrow \forall a, \forall b.(x = a \cdot b \rightarrow (a = S(0) \vee b = S(0)))),\end{aligned}$$

which can be further simplified to

$$\begin{aligned}(S(0) < x) \wedge (\forall a, \forall b.((2^x - 1) = a \cdot b \rightarrow (a = S(0) \vee b = S(0))) \\ \rightarrow \forall a, \forall b.(x = a \cdot b \rightarrow (a = S(0) \vee b = S(0)))).\end{aligned}$$

Example 3. Consider the following first-order formula over real closed fields. (The notation and symbols are used in the conventional arithmetic sense.)

$$\exists x.(a \cdot x^2 + c = 0) \tag{5}$$

Let

$$f(a,c) = \begin{cases} \sqrt{-c/a} & \text{if } c/a < 0 \text{ or } c/a = 0 \\ 0 & \text{if } 0 < c/a \end{cases}.$$

(For $0 < c/a$, the value of $f(a,c)$ can be arbitrary and is set to 0.) Quantifier elimination by substituting $f(a,c)$ for x in Equation (5) yields

$$((c/a < 0 \vee c/a = 0) \wedge a \cdot (\sqrt{-c/a})^2 + c = 0) \vee (0 < c/a \wedge a \cdot 0^2 + c = 0),$$

which can be simplified to $c/a < 0 \vee c/a = 0$. Alternatively, let

$$f(a,c) = \sqrt{\sqrt{(-c/a)^2}}.$$

The corresponding quantifier-free formula becomes

$$a \cdot \left(\sqrt{\sqrt{(-c/a)^2}}\right)^2 + c = 0.$$

As this paper focuses on the characterization of the complete flexibility of a composite function, how to effectively exploit such flexibility in simplifying quantifier elimination in predicate logic remains an open problem. In fact interpolation is not directly extensible to generate composite functions due to the difference between formulas and functions in predicate logic.

5 Experimental Results

The proposed compositional approach to quantifier elimination was implemented in the ABC package [1]; the experiments were conducted on a Linux machine with Xeon 3.4GHz CPU and 6Gb RAM.

We showed the effectiveness of quantifier elimination by constructing the transition relations of circuits taken from ISCAS and ITC benchmark suites. In the transition relation of a circuit, its primary-input variables were existentially quantified.

Despite recent advances in AIG packages, it was observed that AIG minimization may not be effective when AIG sizes reach tens of thousands of nodes. Hence it is beneficial to perform AIG minimization whenever possible before the sizes get too large to be reduced. So in the experiments AIGs were constantly minimized throughout the computation. Specifically, for expansion-based quantification, minimization was applied after the disjunction of two cofactored circuits; for composition-based quantification, minimization was applied before circuit-to-CNF conversion for SAT solving, at interpolant simplification, and after functional composition.[3]

As quantification scheduling is crucial to the scalability of quantifier elimination, we adopted a simple strategy: By imposing a postponement threshold (the percentage of AIG-size increase due to quantification), variables whose corresponding quantifications result in substantial increase in AIG sizes (exceeding the postponement threshold) are deferred. This threshold is lifted gradually in the iterative computation until all quantifiers are eliminated. In the following experiments, the threshold starts at and increases by 10%. The same scheduling strategy is applied for both expansion- and composition-based quantifications.

Table 1 compares the two quantification methods based on formula expansion (denoted QE-Exp) and functional composition (denoted QE-Cmp), where an entry with "—" indicates data unavailable due to time-out at the limit of 90000 seconds. The smaller number of every corresponding pair of AIG sizes (circuit depths) between QE-Exp and QE-Cmp is highlighted in bold. The ratio shown in the table is calculated under the exclusion of the 5 unsolvable circuits of QE-Exp.

As can be seen from Columns 5 and 9 of Table 1, QE-Cmp is much more scalable than QE-Exp. The AIG sizes of QE-Cmp after quantification are typically much smaller than or comparable to those of QE-Exp. Apart from the 5 unsolvable circuits of QE-Exp, `s991` is an extreme, where the final AIG size of QE-Cmp is 3 orders of magnitude smaller than that of QE-Exp. On the other hand, circuit `s1196` is an exception, where the result of QE-Cmp is 6 times larger than that of QE-Exp due to the failure to derive reasonable-sized interpolants. Despite the effectiveness of QE-Cmp, there are still cases `b14`, ..., `b22` of the ITC benchmark circuits (not shown in Table 1) unsolvable by either of QE-Exp and QE-Cmp. In these cases, the unsatisfiability of $\phi_A \wedge \phi_B$ is hard to establish. Even if an interpolation succeeds, the corresponding interpolant is too large to be useful. Further breakthroughs are needed to overcome these limitations.

In addition to circuit sizes, we examine the effects of QE-Exp and QE-Cmp on circuit depths. Columns 6 and 10 of Table 1 show the characteristics of both methods. Compared to Column 4, the transition relations before quantifier

[3] A synthesis script of commands `ifraig`, `rewrite`, `refactor`, `balance`, `rewrite`, `refactor`, `balance` of ABC was applied for AIG minimization.

Table 1. Quantifier elimination with formula expansion and functional composition.

circuit	(#in, #reg, #n, #l)	rel before QE		QE-Exp				QE-Cmp			
		#n	#l	#n	#l	time	mem	#n	#l	time	mem
prolog	(36, 136, 1656, 26)	1474	29	—	—	—	—	**1088**	**31**	6.27	38.0
s1196	(14, 18, 529, 24)	548	22	**3473**	**21**	5.15	37.3	21881	2532	123.15	37.3
s1269	(18, 37, 569, 35)	622	37	31005	**39**	59.24	37.5	**1694**	116	41.05	37.5
s13207.1	(62, 638, 8027, 59)	5272	45	—	—	—	—	**4741**	**44**	50.60	40.6
s1423	(17, 74, 657, 59)	757	63	17619	**59**	25.45	38.1	**3142**	452	6.19	38.1
s1488	(8, 6, 653, 17)	686	19	1269	**21**	2.90	38.1	**515**	48	3.82	38.1
s1494	(8, 6, 647, 17)	696	20	1261	**21**	2.98	38.1	**607**	42	2.54	38.1
s1512	(29, 57, 780, 30)	697	28	1187	**24**	2.64	37.7	**823**	53	3.78	37.7
s15850.1	(77, 534, 9786, 82)	5679	57	—	—	—	—	**180597**	**14247**	49409.27	427.4
s208.1	(10, 8, 104, 11)	103	14	65	**11**	0.08	37.4	**49**	12	0.06	37.4
s298	(3, 14, 119, 9)	157	15	**117**	**12**	0.08	37.4	122	**12**	0.23	37.4
s3271	(26, 116, 1573, 28)	1565	32	**1549**	**29**	3.08	38.0	1604	62	7.11	38.0
s3330	(40, 132, 1789, 29)	1434	29	—	—	—	—	**1029**	**28**	6.37	38.0
s3384	(43, 183, 1702, 60)	1801	63	1307	**58**	6.94	38.3	**1276**	**58**	17.29	38.3
s344	(9, 15, 160, 20)	164	19	**140**	**19**	0.33	37.1	155	**19**	0.81	37.1
s349	(9, 15, 161, 20)	168	19	**140**	**19**	0.26	37.5	155	**19**	0.82	37.5
s382	(3, 21, 158, 9)	220	19	**179**	**16**	0.10	37.7	189	**16**	0.27	37.7
s38417	(28, 1636, 22397, 47)	15762	44	**15705**	**40**	44.79	48.7	18865	106	149.13	46.8
s38584.1	(38, 1426, 19407, 56)	18094	48	57105	**45**	1382.97	71.4	**38089**	1362	268.94	46.0
s386	(7, 6, 159, 11)	189	15	217	**16**	0.62	37.8	**166**	25	0.48	37.8
s400	(3, 21, 162, 9)	228	20	**180**	**16**	0.13	37.7	190	**16**	0.27	37.7
s420.1	(18, 16, 218, 13)	223	17	137	**19**	0.03	37.4	**105**	20	0.05	37.4
s444	(3, 21, 181, 11)	234	19	**179**	**16**	0.09	37.6	191	**16**	0.24	37.6
s499	(1, 22, 152, 12)	274	25	**299**	**17**	0.10	37.4	368	31	0.22	37.4
s510	(19, 6, 211, 12)	236	16	431	21	1.08	37.7	**177**	**13**	1.49	37.7
s526	(3, 21, 193, 9)	284	16	**188**	17	0.09	37.6	210	**14**	0.27	37.6
s5378	(35, 164, 2779, 25)	1995	25	957759	**43**	63744.28	49.1	**37072**	2602	415.18	38.5
s635	(2, 32, 286, 127)	317	42	312	**35**	0.21	37.6	**280**	42	0.28	37.6
s641	(35, 19, 379, 74)	221	30	1202	**18**	4.5	37.5	**277**	27	5.82	37.5
s6669	(83, 239, 3148, 93)	3218	90	—	—	—	—	**2428**	**79**	68.58	39.0
s713	(35, 19, 393, 74)	235	30	1060	**18**	5.36	37.5	**324**	39	3.88	37.5
s820	(18, 5, 289, 10)	364	19	1821	**19**	3.71	37.9	**460**	49	2.85	37.9
s832	(18, 5, 287, 10)	374	19	1579	**20**	3.25	37.9	**419**	37	2.77	37.9
s838.1	(34, 32, 446, 17)	463	22	281	**35**	0.04	37.7	**217**	36	0.08	37.7
s9234.1	(36, 211, 5597, 58)	2790	44	109835	**45**	955.24	39.2	**18898**	653	119.83	39.2
s938	(34, 32, 446, 17)	463	22	281	**35**	0.10	37.4	**217**	36	0.08	37.4
s967	(16, 29, 394, 14)	483	20	9020	**27**	13.05	37.3	**2159**	244	12.58	37.3
s991	(65, 19, 519, 59)	372	46	3227475	**41**	32425.76	90.7	**1287**	124	33.57	37.8
sbc	(41, 27, 1023, 22)	764	21	39023	**31**	72.07	38.0	**2300**	213	21.41	38.0
b01	(2, 5, 42, 6)	59	11	**61**	**11**	0.23	37.2	75	19	0.26	37.2
b02	(1, 4, 23, 5)	36	9	**40**	**9**	0.02	37.3	**40**	12	0.07	37.3
b03	(4, 30, 127, 10)	247	17	247	**16**	0.14	37.6	**239**	**16**	0.34	37.6
b04	(11, 66, 660, 28)	809	32	33633	**46**	50.24	38.2	**5271**	302	9.83	38.2
b05	(1, 34, 963, 55)	965	39	552	37	0.06	38.1	**512**	**35**	0.13	38.1
b06	(2, 9, 46, 5)	77	10	**80**	**9**	0.11	37.4	92	17	0.29	37.4
b07	(1, 49, 391, 31)	560	35	661	28	0.14	37.6	**566**	**27**	0.13	37.6
b08	(9, 21, 153, 16)	238	27	212	**18**	0.29	37.7	**205**	**18**	0.77	37.7
b09	(1, 28, 141, 9)	247	19	**237**	**17**	0.03	37.6	**237**	**17**	0.08	37.6
b10	(11, 17, 178, 12)	247	17	1510	**26**	2.12	37.6	**353**	**26**	1.23	37.6
b11	(7, 31, 732, 34)	734	35	618	**25**	0.50	37.9	**590**	**25**	0.96	37.9
b12	(5, 121, 952, 19)	1485	26	**1740**	**24**	0.65	38.3	1908	41	2.21	38.3
b13	(10, 53, 299, 20)	472	20	435	**16**	0.49	37.6	**423**	**16**	1.15	37.6
ratio				1.000	1.000	1.000	1.000	0.036	8.064	0.013	0.952

“#in”: number of primary inputs; “#reg”: number of registers; “#n”: number of AIG nodes; “#l”: AIG circuit depth; “time”: CPU time (sec); “mem”: memory (Mb) elimination, QE-Exp yielded circuits with comparable logic levels as shown in Column 6, whereas QE-Cmp produced circuits with many more logic levels as shown in Column 10. QE-Exp (respectively QE-Cmp) can be seen in a sense as quantification with restricted (respectively unrestricted) increase in logic levels. QE-Exp and QE-Cmp are analogous to two-level and multi-level logic minimization, respectively. QE-Cmp therefore can potentially achieve more logic sharing and generate smaller circuits. Since logic-level increase is not immaterial in every application, in this case heavy logic synthesis, e.g., with collapse operation, can be adopted to reduce logic levels. On the other hand, these extreme characteristics about logic levels might hint at the potential reduction power of the proposed optimization method discussed in Section 3.4.

The proposed method can be easily integrated into the framework of unbounded model checking as suggested in [17]. Our preliminary experiments on reachability analysis suggested that sparsity is an important factor for QE-Cmp to be effective. Without taking advantage of sparsity in reachability analysis, QE-Cmp may not be as good as QE-Exp. How to enforce sparsity in reachability analysis using QE-Cmp remains to be done.

6 Prior Work

Propositional logic. There have been intensive efforts on BDD-based image computation based on the principle of formula expansion, e.g., [22], and efforts on SAT-based computation with solution enumeration, e.g., [7]. The closest to ours, however, is AIG-based formula expansion [17].

This paper proposes a compositional approach to quantifier elimination. The complete flexibility of the composite function is characterized. Although symbolic techniques, such as BDDs, can be applied, we use SAT solving and Craig interpolation for scalable derivation of the composite function.

Craig interpolation was adopted in [16] to approximate image computation and in [8] to approximate transition relation. This paper uses interpolation to compute exact image and exact transition relation.

Under the virtual substitution principle of [23], quantifier elimination by formula expansion can be considered as virtual substitution with two terms [19]; quantifier elimination by functional composition can be considered as virtual substitution with a single term.

Predicate logic. When generalized to predicate logic, our composite function is similar to the Skolem function [18] with the following differences: Firstly, the former is used for quantifier elimination with term substitution, whereas the latter is used to rewrite formulas in Skolem normal form using Skolem function symbols. Secondly, quantifier elimination with composite functions is structure-specific (i.e., with respect to some structure/interpretation), whereas normal form conversion with Skolem functions is structure-independent (i.e., universal to all structures/interpretations). Thirdly, quantifier elimination with composite functions is equivalence preserving, whereas normal form conversion with Skolem functions is only satisfiability preserving.

Under predicate logic, our quantifier elimination is similar to virtual substitution [23] with the following differences: In virtual substitution, a finite elimination set of terms is identified for quantifier elimination. The notion of composite functions is not explicit in [23]. Our emphasis, on the other hand, is on the characterization of the complete flexibility of composite functions. In eliminating a quantified variable, a single composite function is characterized, rather than a set of terms of the underlying language. We do not address how to represent a composite function by terms, but suggest the usefulness of flexibility in reducing elimination sets.

7 Conclusions and Future Work

We have presented a compositional approach to quantifier elimination. The complete flexibility of composite functions was characterized; Craig interpolation was exploited for effective computation. Experiments showed promising results on extending the capacity of quantifier elimination when the sparsity assumption holds. For first-order logic, our results may shed light on elimination-set minimization as motivated by the examples of Section 4.

As this paper just showed the first step, much work remains to be done. The effectiveness of our method on unbounded model checking and on circuit optimization suggested in Section 3.4 needs further investigation. Quantification scheduling under the new compositional approach awaits improvement. Moreover, a hybrid approach to quantifier elimination by combining the expansion- and composition-based methods may be pursued to keep both AIG sizes and depths small. We anticipate applications of the new quantification method in scalable logic synthesis, where Craig interpolation is evidently gaining importance [10, 9, 11, 12].

Acknowledgments

This work was supported in part by NSC grants 95-2218-E-002-064-MY3 and 96-2221-E-002-278-MY3.

References

1. Berkeley Logic Synthesis and Verification Group. *ABC: A System for Sequential Synthesis and Verification*, 2005. `http://www.eecs.berkeley.edu/~alanmi/abc/`
2. B. F. Caviness and J. R. Johnson (editors). *Quantifier Elimination and Cylindrical Algebraic Decomposition.* Springer, 1998.
3. O. Coudert and J. C. Madre. A unified framework for the formal verification of sequential circuits. In *Proc. Int'l Conf. Computer-Aided Design*, pp. 126-129, 1990.
4. G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata theory and formal languages*, Lecture Notes in Comput. Sci., Vol. 33., Springer, pp. 134-183, 1975.

5. W. Craig. Linear reasoning: A new form of the Herbrand-Gentzen theorem. *J. Symbolic Logic*, vol. 22, no. 3, pp. 250-268, 1957.
6. H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 2nd edition, 2000.
7. M. Ganai, A. Gupta, and P. Ashar. Efficient SAT-based unbounded symbolic model checking using circuit cofactoring. In *Proc. Int'l Conf. Computer-Aided Design*, pp. 510-517, 2004.
8. R. Jhala and K. McMillan. Interpolant-based transition relation approximation. In *Proc. Computer Aided Verification*, pp. 39-51, 2005.
9. R.-R. Lee, J.-H. R. Jiang, and W.-L. Hung. Bi-decomposing large Boolean functions via interpolation and satisfiability solving. In *Proc. Design Automation Conf.*, pp. 636-641, 2008.
10. C.-C. Lee, J.-H. R. Jiang, C.-Y. Huang, and A. Mishchenko. Scalable exploration of functional dependency by interpolation and incremental SAT solving. In *Proc. Int'l Conf. on Computer-Aided Design*, pp. 227-233, 2007.
11. H.-P. Lin, J.-H. R. Jiang, and R.-R. Lee. To SAT or not to SAT: Ashenhurst decomposition in a large scale. In *Proc. Int'l Conf. Computer-Aided Design*, pp. 32-37, 2008.
12. A. Mishchenko, R. K. Brayton, J.-H. R. Jiang, S. Jang. Scalable don't-care-based logic optimization and resynthesis. In *Proc. Int'l Symp. on Field Programmable Gate Arrays*, pp. 151-160, 2009.
13. A. Mishchenko, S. Chatterjee, and R. K. Brayton. DAG-aware AIG rewriting: A fresh look at combinational logic synthesis. In *Proc. Design Automation Conference*, pp. 532-536, 2006.
14. A. Mishchenko, S. Chatterjee, J.-H. R. Jiang, and R. K. Brayton. FRAIGs: A unifying representation for logic synthesis and verification. Technical Report, EECS Dept., UC Berkeley, 2005.
15. K. McMillan. Applying SAT methods in unbounded symbolic model checking. In *Proc. Computer Aided Verification*, pp. 250-264, 2002.
16. K. McMillan. Interpolation and SAT-based model checking. In *Proc. Computer Aided Verification*, pp. 1-13, 2003.
17. F. Pigorsch, C. Scholl, and S. Disch. Advanced unbounded model checking based on AIGs, BDD sweeping, and quantifier scheduling. In *Proc. Formal Methods on Computer Aided Design*, pp. 89-96, 2006.
18. Th. Skolem. *Uber die mathematische Logik. Norsk. Mat. Tidsk.*, 10:125-142, 1928. [Translation in *From Frege to Gödel, A Source Book in Mathematical Logic*, J. van Heijenoort, Harvard Univ. Press, 1967.]
19. A. Seidl and T. Sturm. Boolean quantification in a first-order context. In *Proc. Int'l Workshop on Computer Algebra in Scientific Computing*, pp. 329-345, 2003.
20. T. Sturm. New domains for applied quantifier elimination. In *Proc. Int'l Workshop on Computer Algebra in Scientific Computing*, pp. 295-301, 2006.
21. A. Tarski. *A Decision Method for Elementary Algebra and Geometry*, University of California Press, Berkeley, 1951.
22. H. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Implicit enumeration of finite state machines using BDDs. In *Proc. Int'l Conf. on Computer-Aided Design*, pp. 130ąV133, 1990.
23. V. Weispfenning. The complexity of linear problems in fields. *Journal of Symbolic Computation*, 5:3-27, 1988.