Leveraging Dominators for Preprocessing QBF

Hratch Mangassarian[1] Bao Le[1] Alexandra Goultiaeva[2] Andreas Veneris[1,2] Fahiem Bacchus[2]

Abstract— **Many CAD for VLSI problems can be naturally encoded as Quantified Boolean Formulas (QBFs) and solved with QBF solvers. Furthermore, such problems often contain circuit-based information that is lost during the translation to Conjunctive Normal Form (CNF), the format accepted by most modern solvers. In this work, a novel preprocessing framework for circuit-based QBF problems is presented. It leverages structural circuit dominators to reduce the problem size and expedite the solving process. Our circuit-based QBF preprocessor PReDom recursively reduces dominated subcircuits to return a simpler but equisatisfiable QBF instance. A rigorous proof is given for eliminating subcircuits dominated by single outputs, irrespective of input quantifiers. Experimental results are presented for circuit diameter computation problems. With preprocessing times of at most five seconds using PReDom, three state-of-the-art QBF solvers can solve 27% to 45% of our problem instances, compared to none without preprocessing.**

I. Introduction

Quantified Boolean Formula (QBF) satisfiability is a PSPACE-complete generalization of Boolean satisfiability (SAT). Numerous CAD for VLSI problems (e.g., model checking [1], sequential equivalence checking [2], sequential power estimation [3]) are known to be PSPACE-complete and can be naturally encoded as QBF instances. As such, there has been a surging interest in the use of QBF in CAD, particularly by the formal community [4]–[6]. With the growing size and complexity of VLSI designs, the advancement of QBF solving procedures is necessary in order to robustly handle QBF encodings of increasingly challenging CAD tasks.

State-of-the-art QBF solvers (e.g. [7]–[10]) normally operate on QBF problems with propositional formulas given in Conjunctive Normal Form (CNF). The standardization of this format has carried over from SAT to QBF due to the efficient data-structures and solving strategies developed for CNF-based SAT solvers, which are also used in many QBF solvers. However, QBF instances originating from CAD problems usually have a circuit structure which is lost during the conversion into CNF. Recently, a number of papers have shown that this structure can be effectively exploited in QBF solvers in different ways. In particular, circuit observability don't-cares can be used to improve the performance of QBF solvers [11], [12] just as they did for SAT [13], [14]. Several papers [15], [16] analyze the limitations of CNF for QBF and offer alternative representations addressing some of these issues. More recently, QBF solvers that handle problems directly in a circuit-based format have shown promising results using various solving methods [12], [17], [18].

In this work, a novel preprocessing framework for exploiting the circuit structure of QBF problems using *structural dominators* is presented. A node in a circuit is said to dominate another node if every path from the second node to a primary output passes through the first. Dominator-based search-space pruning techniques are used in many CAD for VLSI applications, such as test pattern generation [19] and technology mapping [20], among others. In a circuit-based QBF setting, the aim is to simplify the problem by searching for nodes that dominate their fanin cones and subsequently removing the dominated subcircuits using proven theoretical reduction rules. It should be noted that the contribution of this paper is orthogonal to existing CNF-based QBF preprocessors (e.g. [21]).

A rigorous proof is given for the reduction of subcircuits dominated by single outputs in a circuit-based QBF, irrespective of the subcircuit input quantifiers or the structure of the remaining circuit. More precisely, the dominator of a subcircuit is shown to be replaceable by an appropriately computed constant truth value or a quantified input variable, without affecting the satisfiability of the original QBF. We introduce the circuit-based QBF preprocessor PReDom: PRe(process) and ReD(uce) Dom(inators). PReDom is independent of any particular QBF solver and efficiently automates the process of recursively reducing dominated subcircuits.

Experimental results are shown on circuit state-space diameter computation problems [22] for a distributed mutual exclusion protocol from NuSMV [23]. The run-time overhead of preprocessing these benchmarks using PReDom is at most five seconds and the resulting equisatisfiable QBF problems can be given to any QBF solver. PReDom reduces the number of clauses by 47% on average compared to the original instances and 19% after standard simplifications. Three state-of-the-art QBF solvers solve 27% to 45% of the resulting instances, compared to *none* after standard simplifications. Admittedly, these results encourage further research in new strategies that exploit the circuit structure of QBFs to increase performance.

The paper is organized as follows. Section II contains background. Section III presents the formal theory for the reduction of single-output dominated subcircuits in a circuit-based QBF. Section IV describes the PReDom algorithm. Section V gives experimental results and Section VI concludes the paper and discusses future work.

II. Background

A. Quantified Boolean Formulas

A QBF in *prenex form* is written as $Q.\phi$, where Q is called the *prefix* and ϕ is called the *matrix*. The matrix is a propositional logic formula over a set of Boolean variables $\{x_1, \dots, x_n\}$ using Boolean connectives such as conjunction ($\wedge$), disjunction ($\vee$) and inversion($\bar{\ }$). The prefix $Q = q_1V_1\ q_2V_2\ \cdots\ q_rV_r$ consists of *quantifiers* $q_i \in \{\exists, \forall\}$, such that $q_i \neq q_{i+1}$, and mutually disjoint variable sets V_i, called *scopes*, which partition $\{x_1, \dots, x_n\}$. A variable $x \in V_i$ is labeled as an *existential* (*universal*) variable if $q_i = \exists$ ($q_i = \forall$). A scope V_i or variable $x \in V_i$ is said to be *wider* (*narrower*) than a scope V_j or variable $y \in V_j$ if $i < j$ ($i > j$). This order of scopes imposes a partial order on the variables $\{x_1, \dots, x_n\}$, such that $x < y$ if and only if the scope of x is wider than the scope of y. In order to simplify the presentation, we establish a total order on the variables, which respects this partial order and arbitrarily orders variables that are in the same scope.

A literal l_i is an occurrence of a variable x_i or its negation $\bar{x}_i$. In the *reduction* of $Q.\phi$ by l_i, denoted $Q.\phi|_{l_i}$, if $l_i = x_i$ (if $l_i = \bar{x}_i$) then occurrences of x_i are replaced by $\mathtt{1}$ ($\mathtt{0}$), and those of $\bar{x}_i$ are replaced by $\mathtt{0}$ ($\mathtt{1}$). More generally, if π is a truth assignment over a subset of $\{x_1, \dots, x_n\}$, then $Q.\phi|_\pi$ denotes the formula $Q.\phi$ after assigning these variables to their truth values in π. The notations $\pi = \{x_i = \mathtt{1}, x_j = \mathtt{0}, \dots\}$ and $\pi = \{x_i, \bar{x}_j, \dots\}$ are equivalent and used interchangeably.

Formally, the value of a QBF can be determined by the recursive application of the following two rules: (1) $\exists x Q.\phi = Q.\phi|_x \vee Q.\phi|_{\bar{x}}$,

[1]University of Toronto, ECE Department, Toronto, ON M5S 3G4 ({hratch, veneris}@eecg.toronto.edu, bao.le@utoronto.ca)

[2]University of Toronto, CS Department, Toronto, ON M5S 3G4 ({alexia, fbacchus}@cs.toronto.edu)

and (2) $\forall x Q.\phi = Q.\phi|_x \wedge Q.\phi|_{\bar{x}}$. A *Q-model* is a tree of truth assignments, where each existential variable is a function of wider universal variables, such that the matrix is satisfied for all universal variable assignments. If a QBF has a Q-model, it is true or QSAT, otherwise it is false or UNQSAT.

For example, the QBF problem: $\exists x_1\ \forall x_2\ \exists x_3\ .\ (x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_1)$ is QSAT because when $x_1 = 1$, for all values of x_2, there exists an assignment to x_3 ($x_3 = 1$ when $x_2 = 0$ and $x_3 = 0$ when $x_2 = 1$) that satisfies the matrix. The QBF in this example is in *prenex normal form*, where the matrix ϕ is in *Conjunctive Normal Form* (CNF). CNF is a conjunction of *clauses*, where each clause is a disjunction of literals. Most state-of-the-art QBF solvers accept QBFs in prenex normal form.

In circuit application domains, the QBF matrix can be given as a logic circuit C, consisting of interconnected logic gates, such as NOT, AND, OR and XOR, and a primary output which must be satisfied. C is a directed graph (V, E, β) with $|V|$ nodes (primary inputs and internal gates), $|E|$ edges and output $\beta \in V$. Such a circuit-based QBF can be written as $Q.\beta$, where the variables in Q are in a one-to-one correspondence with the primary inputs of the circuit C whose output is β. In order to pass $Q.\beta$ to a CNF-based QBF solver, a CNF formula expressing C can be constructed in linear time [24] by adding an existentially quantified auxiliary variable for each internal gate in a narrower scope compared to its inputs.

Let f be a function which transforms an arbitrary QBF Φ into another QBF $f(\Phi)$. f is said to preserve *equisatisfiability*, or equivalently, Φ and $f(\Phi)$ are said to be *equisatisfiable*, if:

$$\Phi \text{ is QSAT} \Leftrightarrow f(\Phi) \text{ is QSAT}.$$

The primary inputs in C (or variables in the matrix) which are quantified in the prefix are said to be *bound*. A circuit-based (or any prenex form) QBF can also have some primary inputs in C (or variables in the matrix) that are *free*, i.e. are not quantified in Q. For QBFs with free variables, the concept of equisatisfiability is extended to *logical equivalence*. Two QBFs with the same free variables are said to be *logically equivalent*, if for every assignment to the free variables, the QBFs are equisatisfiable.

B. Structural Dominators

In a directed graph $C = (V, E, \beta)$, a node $u \in V$ *dominates* node $v \in V$ ($v \neq u$), if every path from v to the output β passes through u. The set $dom(v) = \{u \in V | u \text{ dominates } v\}$ consists of nodes that dominate v. The inverse set $dom^{-1}(v) = \{u \in V | v \text{ dominates } u\}$ consists of nodes dominated by v.

Let the set $fanin(v) = \{u \in V | (u, v) \in E\}$ denote the fanins of v, the set $fanin^*(v) = \{u \in V | \exists \text{ a path } u \rightsquigarrow v \text{ in } C\}$ denote the transitive fanin cone of v, and the set $faninPI^*(v) = \{u \in fanin^*(v) | \forall w \in V. \nexists (w, u) \in E\}$ denote the primary inputs in the transitive fanin cone of v. We call a node v a *complete dominator* if it dominates every node in $fanin^*(v)$, i.e. if $dom^{-1}(v) = fanin^*(v)$.

Consider the circuit in Figure 1. Here, $dom(x_2) = \{\alpha, \beta\}$, $dom^{-1}(g_3) = \{x_1, x_3, g_1, g_2\}$ and $faninPI^*(\alpha) = \{x_1, x_2, x_3\}$. Note that $dom^{-1}(\alpha) = fanin^*(\alpha) = \{x_1, x_2, x_3, g_1, g_2, g_3\}$, hence α is a complete dominator. In this work, the transitive fanin cone $fanin^*$ of a complete dominator α is referred to as a Single-Output Dominated Subcircuit, or SODS.

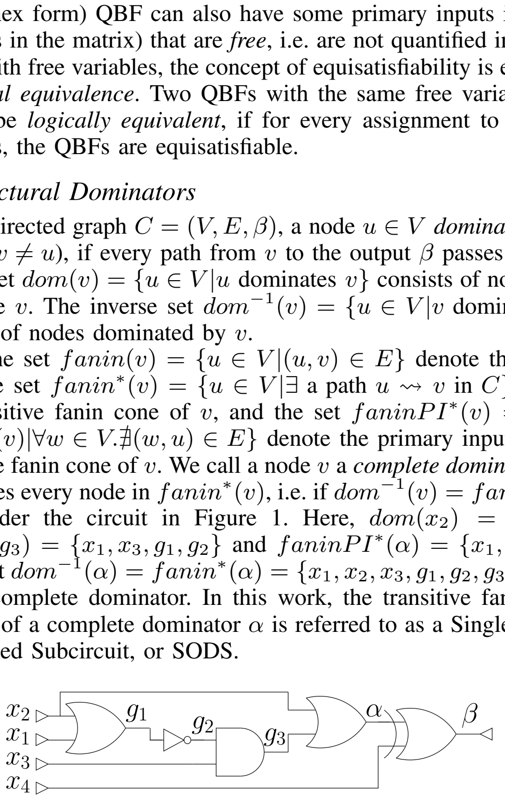

Fig. 1. A complete dominator α

Many algorithms have been developed for constructing the set $dom(v)$ for each $v \in V$. The run-times of these algorithms have improved from $O(|V|^2)$ [25] to $O(|E| + |V|)$ [26].

III. Reducing SODSes in a circuit-based QBF

This work presents a preprocessing step for the simplification of a circuit-based QBF, using structural dominators. The aim is to obtain a smaller QBF which is equisatisfiable to the original, and therefore easier to solve by a QBF solver. For instance, we will show that in the QBF $\exists x_1 \forall x_2 \exists x_3 \forall x_4.\beta$, where the circuit producing β is given in Figure 1, we can set the complete dominator $\alpha = 1$ and therefore remove $fanin^*(\alpha)$, while preserving equisatisfiability.

In the following subsection, we give a motivating example to illustrate the basic idea, where a single gate can be removed and replaced by a constant. Then we consider the general case, and give a formal proof showing how to appropriately reduce any SODS in a circuit-based QBF and replace its output by a constant truth value or a quantified primary input, all while preserving equisatisfiability.

A. A Motivating Example

Before discussing the example, we must introduce some notation. Let $\mathbb{F} = Q.\beta$ denote the original circuit-based QBF and $f_\alpha(\mathbb{F})$ denote the reduced QBF after the elimination of the SODS of the complete dominator α. We call $f_\alpha(\mathbb{F})$ the *α-reduced* QBF of $\mathbb{F}$. The function $f_\alpha(\mathbb{F})$ can yield one of the following four types of reductions:

- $\mathbb{F}^0$ (or $\mathbb{F}^1$) denotes the formula $\mathbb{F}$ where α has been replaced by 0 (or 1) and $fanin^*(\alpha)$ has been removed.
- $\mathbb{F}^{\exists_z}$ ($\mathbb{F}^{\forall_z}$), where z is a variable in the quantifier prefix of $\mathbb{F}$, denotes the formula $\mathbb{F}$ where α has been replaced by a new existential (universal) primary input $\tilde{\alpha}$ introduced in the same scope as z. Again, $fanin^*(\alpha)$ has been removed.

Now consider the circuit-based QBF in Figure 2. The circuit C contains the AND gate α, which is a complete dominator. Note that α may fan out to any number of other gates, and C may contain any number of other inputs, arbitrarily quantified in Q. Let $\mathbb{F} = \forall x_1 \exists x_2 Q.\beta$ denote this circuit-based QBF. We will show that setting $\alpha = 0$ and removing $fanin^*(\alpha)$ in $\mathbb{F}$ yields an equisatisfiable QBF.

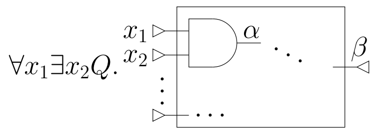

Fig. 2. A circuit-based QBF

Claim $\mathbb{F} = \forall x_1 \exists x_2 Q.\beta$ and $\mathbb{F}^0 = Q.\beta^0$ are equisatisfiable.

Proof: We must prove that: (a) $\mathbb{F}$ is true $\Rightarrow \mathbb{F}^0$ is true, and (b) $\mathbb{F}$ is false $\Rightarrow \mathbb{F}^0$ is false.

(a) By definition, $\mathbb{F} = \forall x_1 \exists x_2 Q.\beta = \exists x_2 Q.\beta|_{\bar{x}_1} \wedge \exists x_2 Q.\beta|_{x_1}$. So $\mathbb{F}$ is true $\Rightarrow \exists x_2 Q.\beta|_{\bar{x}_1}$ is true. But $x_1 = 0 \Rightarrow \alpha = 0$ and x_1 can affect β only through its dominator α. Therefore, after setting $\alpha = 0$, we can disregard x_1, and $\exists x_2 Q.\beta^0$ is true. Here, x_2 is a dummy variable, hence $Q.\beta^0$ is true.

(b) By definition, $\mathbb{F}$ is false $\Rightarrow \exists x_2 Q.\beta|_{\bar{x}_1} \wedge \exists x_2 Q.\beta|_{x_1}$ is false. So either $(b.1)$ $\exists x_2 Q.\beta|_{\bar{x}_1}$ is false or $(b.2)$ $\exists x_2 Q.\beta|_{x_1}$ is false. We show that each of $(b.1)$ and $(b.2)$ implies that $\mathbb{F}^0$ is false.

- $(b.1)$ Similarly to (a), $(b.1) \Rightarrow \exists x_2 Q.\beta^0$ is false $\Rightarrow Q.\beta^0$ is false.
- $(b.2)$ By definition, $\exists x_2 Q.\beta|_{x_1} = Q.\beta|_{x_1,\bar{x}_2} \vee Q.\beta|_{x_1,x_2}$. Therefore, $(b.2) \Rightarrow Q.\beta|_{x_1,\bar{x}_2}$ is false and $Q.\beta|_{x_1,x_2}$ is false. But $Q.\beta|_{x_1,\bar{x}_2}$ is false $\Rightarrow Q.\beta^0$ is false, since x_1 and x_2 can affect β only through α.

■

Note that $\mathbb{F}$ and $\mathbb{F}^0$ being equisatisfiable does not mean that they are structurally equivalent. In fact, setting $x_1 = 1$ and $x_2 = 1$ in $\mathbb{F}$ makes $\alpha = 1$, whereas α is a constant 0 in $\mathbb{F}^0$. On the other hand, $\mathbb{F}$ and $\mathbb{F}^0$ are equivalent with respect to satisfiability, i.e. a QBF solver will evaluate both QBFs to the same value (true or false).

B. Reducing SODSes

Given a complete dominator α, the aim of this section is to show how to construct $f_\alpha(\mathbb{F})$ in general and prove that it produces a QBF equisatisfiable to the original circuit-based QBF $\mathbb{F}$. In order to do so, we will carefully define the function $f_\alpha(\Phi, \pi)$, which produces a QBF equisatisfiable to formula Φ *under the truth assignment* π. As such, by definition, $f_\alpha(\mathbb{F}, \emptyset) = f_\alpha(\mathbb{F})$.

At the base cases, under a truth assignment π setting all variables of $faninPI^*(\alpha)$, the α-reduced QBF of $\mathbb{F}$ is trivially obtained using circuit simulation by replacing α with its value under π. We will show how to use these base cases for every truth assignment to $faninPI^*(\alpha)$ to ultimately construct the α-reduced QBF of $\mathbb{F}$ under the empty assignment, i.e. $f_\alpha(\mathbb{F}, \emptyset)$.

Let Φ be a circuit-based QBF with the same matrix (i.e. circuit) as the original QBF $\mathbb{F}$, but such that the widest k variables ($0 \le k \le n$) in the total order of its variables $\{x_1, \dots, x_n\}$ can be free (i.e. unquantified) in Φ. Also, let π be any truth assignment setting exactly the variables in $faninPI^*(\alpha)$ that are free in Φ.

Definition 1 *We define $f_\alpha(\Phi, \pi)$ recursively as follows:*

(a) *Base case. If all the variables in Φ are free (i.e. $\Phi = \beta$) and consequently π assigns all the variables in $faninPI^*(\alpha)$, then $f_\alpha(\Phi, \pi) = \Phi^0$ (or Φ^1) if $\alpha = 0$ (or 1) under π.*

(b) *If $\Phi = qx\Psi$, where $x \notin faninPI^*(\alpha)$, $q \in \{\exists, \forall\}$ and Ψ is Φ with x free, then $f_\alpha(\Phi, \pi) = qxf_\alpha(\Psi, \pi)$.*

(c) *If $\Phi = qx\Psi$, where $x \in faninPI^*(\alpha)$, $q \in \{\exists, \forall\}$ and Ψ is Φ with x free, then $f_\alpha(\Phi, \pi) = f_\alpha(\Psi, \pi \cup \{x = 0\}) \odot f_\alpha(\Psi, \pi \cup \{x = 1\})$, where $\odot = \vee(\wedge)$ if $q = \exists(\forall)$, and the result of this function is given in Table I.*

We use the notation $a \uparrow b$ ($a \downarrow b$) for two variables a and b to denote the one with the widest (narrowest) scope.

TABLE I
$f_\alpha(\Phi, \pi)$ WHEN $\Phi = qx\Psi$ AND $x \in faninPI^*(\alpha)$

Case	$f_\alpha(\Psi, \pi \cup \{x = 0\})$ and $f_\alpha(\Psi, \pi \cup \{x = 1\})$		$f_\alpha(\Phi, \pi)$	
			$q = \exists$	$q = \forall$
1	Ψ^0	Ψ^0	Φ^0	Φ^0
2	Ψ^0	Ψ^1	$\Phi^{\exists x}$	$\Phi^{\forall x}$
3	Ψ^0	$\Psi^{\exists z}$	$\Phi^{\exists z}$	Φ^0
4	Ψ^0	$\Psi^{\forall z}$	Φ^0	$\Phi^{\forall z}$
5	Ψ^1	Ψ^1	Φ^1	Φ^1
6	Ψ^1	$\Psi^{\exists z}$	$\Phi^{\exists z}$	Φ^1
7	Ψ^1	$\Psi^{\forall z}$	Φ^1	$\Phi^{\forall z}$
8	$\Psi^{\exists z_1}$	$\Psi^{\exists z_2}$	$\Phi^{\exists z_1 \downarrow z_2}$	$\Phi^{\exists z_1 \uparrow z_2}$
9	$\Psi^{\exists z_1}$	$\Psi^{\forall z_2}$	$\Phi^{\exists z_1}$	$\Phi^{\forall z_2}$
10	$\Psi^{\forall z_1}$	$\Psi^{\forall z_2}$	$\Phi^{\forall z_1 \uparrow z_2}$	$\Phi^{\forall z_1 \downarrow z_2}$

In order to construct $f_\alpha(\mathbb{F}, \emptyset)$ using Definition 1, one has to proceed in a bottom-up recursive fashion, starting from all the base cases, where $\Phi = \beta$ and π assigns every combination of $faninPI^*(\alpha)$, and adding quantification scopes to β as described in Definition 1, until $f_\alpha(\mathbb{F}, \emptyset)$ is calculated. This will be illustrated in detail later using an example.

Theorem 1 $\Phi|_\pi$ *and* $f_\alpha(\Phi, \pi)$ *are logically equivalent.*

Proof: The proof is by induction over the variables in the prefix of Φ, and follows the recursive definition of the function $f_\alpha(\Phi, \pi)$.

(a) Base case. Here π assigns all variables in $faninPI^*(\alpha)$, so $f_\alpha(\Phi, \pi) \in \{\Phi^0, \Phi^1\}$ simply replaces α by its value under π.

(b) $\Phi = qx\Psi$, with $x \notin faninPI^*(\alpha)$. By the inductive hypothesis, assume $f_\alpha(\Psi, \pi)$ and $\Psi|_\pi$ are logically equivalent. Since $x \notin faninPI^*(\alpha)$, it does not participate in the transformation $f_\alpha(\Psi, \pi)$, so $qxf_\alpha(\Psi, \pi)$ and $qx\Psi|_\pi$ are logically equivalent. Hence, by definition, $f_\alpha(\Phi, \pi)$ and $\Phi|_\pi$ are logically equivalent.

(c) $\Phi = qx\Psi$, with $x \in faninPI^*(\alpha)$. Recall the definition of quantification: $\exists x\Psi|_\pi = \Psi|_{\pi,\bar{x}} \vee \Psi|_{\pi,x}$ and $\forall x\Psi|_\pi = \Psi|_{\pi,\bar{x}} \wedge \Psi|_{\pi,x}$. So, $\Phi|_\pi = qx\Psi|_\pi = \Psi|_{\pi,\bar{x}} \odot \Psi|_{\pi,x}$, where $\odot = \vee(\wedge)$ if $q = \exists(\forall)$. By the inductive hypothesis, $\Psi|_{\pi,\bar{x}}$ and $f_\alpha(\Psi, \pi \cup \{x = 0\})$ are logically equivalent, and $\Psi|_{\pi,x}$ and $f_\alpha(\Psi, \pi \cup \{x = 1\})$ are logically equivalent. So $\Phi|_\pi$ and $f_\alpha(\Psi, \pi \cup \{x = 0\}) \odot f_\alpha(\Psi, \pi \cup \{x = 1\})$ are logically equivalent. It remains to show that this latter function is equivalent to the results displayed in the last two columns of Table I, for every $f_\alpha(\Psi, \pi \cup \{x = 0\})$ and $f_\alpha(\Psi, \pi \cup \{x = 1\})$.

- Cases 1,5. Here, $f_\alpha(\Psi, \pi \cup \{x = 0\}) = f_\alpha(\Psi, \pi \cup \{x = 1\})$, so $f_\alpha(\Psi, \pi \cup \{x = 0\}) \odot f_\alpha(\Psi, \pi \cup \{x = 1\}) = f_\alpha(\Psi, \pi \cup \{x = 0\})$, which is Ψ^0 or Ψ^1.
- Case 2. Here, $f_\alpha(\Psi, \pi \cup \{x = 0\}) \odot f_\alpha(\Psi, \pi \cup \{x = 1\}) = \Psi^0 \odot \Psi^1$. We can apply the definition of quantification to create a new variable $\tilde{\alpha}$ that will replace α, such that $\Psi^0 \odot \Psi^1 = q\tilde{\alpha}\Psi|_{\alpha=\tilde{\alpha}}$, where q is the quantifier type of x, and the new input variable $\tilde{\alpha}$ is placed in the same scope as x. This is also denoted as Ψ^{qx}. Hence $f_\alpha(\Phi, \pi) = \Psi^{qx}$.
- Cases 3,4,6,7,8,9,10. Recall from formal logic that if $a \Rightarrow b$, then $a \wedge b = a$ and $a \vee b = b$. As such:
  - If $q = \exists$ and $f_\alpha(\Psi, \pi \cup \{x = 0\}) \Rightarrow f_\alpha(\Psi, \pi \cup \{x = 1\})$ (or vice-versa), then $f_\alpha(\Phi, \pi) = f_\alpha(\Psi, \pi \cup \{x = 1\})$ (or $f_\alpha(\Psi, \pi \cup \{x = 0\})$).
  - If $q = \forall$ and $f_\alpha(\Psi, \pi \cup \{x = 0\}) \Rightarrow f_\alpha(\Psi, \pi \cup \{x = 1\})$ (or vice-versa), then $f_\alpha(\Phi, \pi) = f_\alpha(\Psi, \pi \cup \{x = 0\})$ (or $f_\alpha(\Psi, \pi \cup \{x = 1\})$).

  Hence, it suffices to prove the implication $f_\alpha(\Psi, \pi \cup \{x = 0\}) \Rightarrow f_\alpha(\Psi, \pi \cup \{x = 1\})$ (or vice-versa) for each of these cases.

  3 $\Psi^0 \Rightarrow \Psi^{\exists z}$. If Ψ^0 is true, we can make $\Psi^{\exists z}$ true by setting $\tilde{\alpha} = 0$.

  4 $\Psi^{\forall z} \Rightarrow \Psi^0$. By definition, if $\Psi^{\forall z}$ is true, it must be true for both $\tilde{\alpha} = 0$ and $\tilde{\alpha} = 1$. In particular, it must be true when $\tilde{\alpha} = 0$.

  6 $\Psi^1 \Rightarrow \Psi^{\exists z}$. Analogous to case 3.

  7 $\Psi^{\forall z} \Rightarrow \Psi^1$. Analogous to case 4.

  8 $\Psi^{\exists z_1 \uparrow z_2} \Rightarrow \Psi^{\exists z_1 \downarrow z_2}$. Without loss of generality, assume that $z_1 \uparrow z_2 = z_1$ and $z_1 \downarrow z_2 = z_2$. If $\Psi^{\exists z_1}$ is true, then it has a Q-model in which the value of $\tilde{\alpha}$ is a function of all universal variables with a wider scope than z_1. Since the scope of z_2 is narrower, in $\Psi^{\exists z_2}$ the value of $\tilde{\alpha}$ is a function of all those and possibly additional universal variables. Therefore, the value of $\tilde{\alpha}$ in a Q-model of $\Psi^{\exists z_2}$ can emulate $\tilde{\alpha}$ in $\Psi^{\exists z_1}$ by ignoring the remaining universal variables with scopes between z_1 and z_2, which it is a function of.

  9 $\Psi^{\forall z_2} \Rightarrow \Psi^{\exists z_1}$. The proof can be composed from Cases 3 and 4 as follows: $\Psi^{\forall z_2} \Rightarrow \Psi^0 \Rightarrow \Psi^{\exists z_1}$.

  10 $\Psi^{\forall z_1 \downarrow z_2} \Rightarrow \Psi^{\forall z_1 \uparrow z_2}$. Widening the scope of the universal $\tilde{\alpha}$ from $z_1 \downarrow z_2$ to $z_1 \uparrow z_2$ will make existential variables between those scopes also a function of $\tilde{\alpha}$. Since $\Psi^{\forall z_1 \downarrow z_2}$ is true, to produce a Q-model for $\Psi^{\forall z_1 \uparrow z_2}$ the existential variables between the scopes of z_1 and z_2 can emulate their values in a Q-model of $\Psi^{\forall z_1 \downarrow z_2}$, ignoring $\tilde{\alpha}$.

Finally, in all the sub-cases of (c), we can respectively replace each of $\Psi^0, \Psi^1, \Psi^{\exists z}, \Psi^{\forall z}$ by $\Phi^0, \Phi^1, \Phi^{\exists z}, \Phi^{\forall z}$, by adding a dummy quantification for x without affecting satisfiability, since x does not occur in any of them. In other terms, $\Psi^0 = qx\Psi^0 = \Phi^0$, $\Psi^1 = qx\Psi^1 = \Phi^1$, and so on.

■

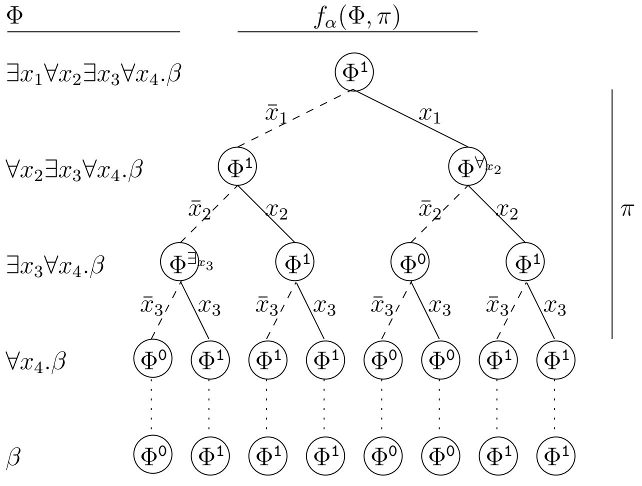

Fig. 3. Constructing $f_\alpha(\mathbb{F})$

Corollary 1 *$\mathbb{F}$ and $f_\alpha(\mathbb{F})$ are equisatisfiable.*

Proof: In Theorem 1, replacing $\Phi = \mathbb{F}$ and $\pi = \emptyset$ since all variables of $\mathbb{F}$ are bound, we get that $\mathbb{F}|_\emptyset = \mathbb{F}$ and $f_\alpha(\mathbb{F},\emptyset) = f_\alpha(\mathbb{F})$ are logically equivalent, and hence equisatisfiable, since they have no free variables. ■

The following example demonstrates the construction of $f_\alpha(\mathbb{F})$.

Example 1 *Consider the circuit-based QBF:*

$$\mathbb{F} = \exists x_1 \forall x_2 \exists x_3 \forall x_4.\beta(\alpha(x_1,x_2,x_3),x_4)$$

where the circuit C producing the primary output β is given in Figure 1. Given the complete dominator α, Figure 3 illustrates the construction of the equisatisfiable α-reduced QBF $f_\alpha(\mathbb{F})$ in a bottom-up recursive fashion. Each node in the tree gives the value of $f_\alpha(\Phi,\pi)$, where Φ is shown at the same level of the node and to the left of the tree, whereas π assigns all the literals shown on the edges of the path from the root to that node.

The construction starts with the base cases (i.e. case (a) in Definition 1), occurring at the leaves of the tree in Figure 3, where $\Phi = \beta$ has only free variables. At each leaf, π fully assigns $faninPI^*(\alpha) = \{x_1,x_2,x_3\}$, and $f_\alpha(\beta,\pi)$ sets α to the value it assumes under π. For example, in the second leaf from the left, $\pi = \{\bar{x}_1,\bar{x}_2,x_3\}$, which yields $\alpha = 1$, and hence $f_\alpha(\beta,\{\bar{x}_1,\bar{x}_2,x_3\}) = \Phi^1$. Notice that the base cases are constructed by simulating α for all combinations of $\{x_1,x_2,x_3\}$.

Once all the base cases are computed, we consider $\Phi = \forall x_4.\beta$. Using the notation of Definition 1, we can write $\Phi = qx_4\Psi$, with $\Psi = \beta$ and $q = \forall$. Since $x_4 \notin faninPI^*(\alpha)$, by case (b) of Definition 1, we get $f_\alpha(\Phi,\pi) = \forall x_4 f_\alpha(\Psi,\pi)$. So if $f_\alpha(\Psi,\pi) = \Psi^0$ (or Ψ^1), then $f_\alpha(\Phi,\pi) = \Phi^0$ (or Φ^1). As such, the reduction at α remains the same at the level of variables in the prefix which are not in $faninPI^*(\alpha)$, such as x_4. The dotted lines above each leaf of the tree in Figure 3 demonstrate this.

Next, each of $x_1,x_2,x_3 \in faninPI^*(\alpha)$, and therefore they follow case (c) of Definition 1. Here, a node $f_\alpha(\Phi,\pi)$ is a function of its two children $f_\alpha(\Psi,\pi \cup \{x_i = 0\})$ and $f_\alpha(\Psi,\pi \cup \{x_i = 1\})$, where $\Phi = qx_i\Psi$, as specified in Table I. For instance, consider $\Phi = \forall x_2 \exists x_3 \forall x_4.\beta$ (so $\Psi = \exists x_3 \forall x_4.\beta$) and $\pi = \{x_1\}$, which corresponds to the right child of the root in Figure 3. We get $f_\alpha(\Phi,\pi) = f_\alpha(\Psi,\pi \cup \{x_2 = 0\}) \wedge f_\alpha(\Psi,\pi \cup \{x_2 = 1\}) = \Psi^0 \wedge \Psi^1 = \Phi^{\forall x_2}$, applying case 2 in Table I.

The final result is given at the root of the tree, where $\Phi = \mathbb{F}$, $\pi = \emptyset$ and $f_\alpha(\mathbb{F},\emptyset) = f_\alpha(\mathbb{F}) = \Phi^1$. In other terms, we can set $\alpha = 1$ and remove its fanin cone without affecting the satisfiability of the original QBF. Clearly, this produces a smaller QBF, which should be easier to solve.

Algorithm 1 PReDom algorithm
```
1:  procedure PReDom($Q$,$C$)
2:      Simplify($Q$,$C$)
3:      $dom \leftarrow$ComputeDom($C$)
4:      $dom^{-1} \leftarrow$ComputeDom$^{-1}$($dom$)
5:      $\forall_{v \in V}.ignore[v] \leftarrow$ false
6:      for all $v \in V$ in topological order $\wedge\ \neg ignore[v]$ do
7:          if $v \in PI$ then
8:              $faninPI^*[v] \leftarrow \{v\}$
9:          else
10:             $faninPI^*[v] \leftarrow \bigcup_{u \in faninPI^*[v]} faninPI^*[u]$
11:         end if
12:         if $|faninPI^*[v]| < MAX$ then
13:             if $faninPI^*[v] = dom^{-1}[v] \cap PI$ then
14:                 $v \leftarrow$ Reduce($v$,$faninPI^*[v]$,$Q$,$C$)
15:                 $faninPI^*[v] \leftarrow \{v\}$
16:             end if
17:         else
18:             $ignore[v] \leftarrow$ true
19:             $\forall_{u \in fanout[v]}.ignore[u] \leftarrow$ true
20:         end if
21:     end for
22: end procedure
```

Notice that during the construction of $f_\alpha(\mathbb{F})$, variables not in $faninPI^*(\alpha)$, such as x_4 in Example 1, can be disregarded in practice. As such, one only needs to examine the SODS of α and can disregard the remaining circuit when constructing $f_\alpha(\mathbb{F})$.

IV. The PReDom Algorithm

In this section, we describe our implementation of the preprocessor PReDom, which searches for complete dominators and uses the theory described in Section III-B to reduce their SODSes. In our implementation, the resulting QBF is given in prenex normal form, so that state-of-the-art CNF-based QBF solvers can take advantage of the reductions. In order to do so, PReDom also stores the CNF representation of the circuit-based problem. Actions such as eliminating a subcircuit and setting dominator outputs to constants are done by removing the clauses corresponding to the gates in that subcircuit and adding unit literals to the CNF formula, respectively.

Algorithm 1 gives a simplified view of the PReDom procedure. Before searching for complete dominators, PReDom performs several standard optimization steps, grouped in Simplify(Q,C) (line 2). Unit constraint propagation and universal reduction are performed directly in the CNF. Equivalence reduction is achieved efficiently by removing NOTs and BUFFERs in the circuit (and therefore their clauses in the CNF) and adding fanin polarity information for the remaining gates. Finally, dangling gates are also removed.

On line 3, $dom[v]$ is computed $\forall v \in V$, where V is the set of nodes in C. We use the method in [25] because it is simple and sufficiently fast in practice, but faster algorithms exist, e.g. [26]. Next, we compute the inverse map $dom^{-1}[v], \forall v \in V$, which stores the set of nodes dominated by v.

In the main loop of the preprocessor, each vertex v is traversed in topological order (line 6), and the set $faninPI^*[v]$ is computed recursively (lines 7 to 10). On line 12, the preprocessor checks that the number of primary inputs in the fanin cone of v is not more than

TABLE II

PREDOM PREPROCESSING RESULTS

Benchmark	# original clauses	PReDom						
		# clauses after SIMPLIFY	final # clauses	% reduction original-final	% reduction SIMPLIFY-final	# SODSes	COMPUTEDOM time (sec)	total time (sec)
dme1_2	20 984	11 565	10 068	52%	13%	125	0.9	1.2
dme1_3	32 888	20 488	18 991	42%	7%	125	1.4	1.8
dme1_4	44 792	29 411	27 914	38%	5%	125	1.9	2.5
dme1_5	56 696	38 334	18 173	68%	53%	76	2.4	3.2
dme1_6	68 600	47 257	45 760	33%	3%	125	2.9	3.9
dme1_7	80 504	56 180	54 683	32%	3%	125	3.4	4.6
dme1_8	92 408	65 103	63 606	31%	2%	125	4.0	5.3
dmeSmall_2	13 984	7 701	6 708	52%	13%	83	0.4	0.5
dmeSmall_4	29 856	19 595	9 136	69%	53%	73	0.9	1.1
dmeSmall_8	61 600	43 383	21 020	66%	52%	73	1.8	2.5
dmeSmall_9	69 536	49 330	48 337	30%	2%	83	2.0	2.8
Average				**47%**	**19%**			

a user-defined upper-bound MAX, which we have set to 20. This is needed to avoid memory explosion, since the number of base cases in the reduction is exponential in $|faninPI^*[v]|$, as shown in Figure 3. Otherwise, notice that β itself is always a complete dominator and could be theoretically reduced. Rather, the scheme is geared towards recursively finding intermediate complete dominators, thus iteratively reducing $|faninPI^*[v]|$ for dominators further down.

Line 13 is the condition for v to be a complete dominator, which is reduced on line 14. Here PI denotes the set of primary inputs in C. The function REDUCE uses parallel simulation to efficiently produce all the input combinations of $faninPI^*[v]$, and applies a scheme similar to the one shown in Example 1, to reduce the SODS of v. When a complete dominator is found and reduced, it is propagated and its $faninPI^*[v]$ is reset. The $ignore$ flags are used to disregard nodes which are already known to have more than MAX primary inputs in their fanin cones.

Finally, since gates with more than two inputs can be decomposed into two-input gates, it is sufficient for a multi-input gate to completely dominate the fanin cones of just two of its inputs for a reduction to be possible. PReDom handles these cases efficiently.

V. EXPERIMENTS

We implemented our circuit-based QBF preprocessor PReDom in C++. The input format of the preprocessor is composed of three parts: (a) A circuit description in ISCAS85 format using NOT, BUFFER, AND, OR, NAND, NOR, XOR and XNOR gates, (b) the corresponding QBF in prenex normal form, and (c) a file mapping between the circuit nodes and variables/clauses in the CNF. The preprocessor first applies standard optimizations (SIMPLIFY), then searches for complete dominators and reduces their SODSes, as described in this paper. The resulting problems are then given to QBF solvers.

The benchmarks are a suite of circuit state-space diameter computation problems called *dme* [22] for a distributed mutual exclusion protocol from NUSMV [23]. They are originally given in the QBF1.0 format, and are converted in negligible time into our specified format using the converter of [12]. The results of three state-of-the-art QBF solvers, namely sKizzo [7] (version 0.10), 2clsQ [10] and quantor [8] (version 3.0, with the recommended picosat back end) are compared with and without PReDom. All experiments are conducted on a Pentium IV 2.8 GHz Linux platform with 12 GB of memory and a time limit of 5 hours.

Table II shows the preprocessing results using PReDom. The first column gives the instance name. Column *# original clauses* gives the original number of clauses in the CNF of each problem instance. Columns *# clauses after* SIMPLIFY and *final # clauses* respectively show the number of clauses after the optimizations in SIMPLIFY and after preprocessing by PReDom. Column % *reduction original-final* (% *reduction* SIMPLIFY*-final*) shows the percentage reduction in the number of clauses from the original (simplified) instance to the final preprocessed instance. Column *# SODSes* gives the total number of SODSes which were reduced. Column COMPUTEDOM *(sec)* gives the run-time to compute the sets $dom(v), \forall v \in V$, using the algorithm from [25]. Finally, column *total time (sec)* gives the total run-time of PReDom on each instance in seconds.

Notice that the average reduction in the number of clauses from the original to the final instance is 47%. This includes both standard optimizations in SIMPLIFY and dominator-based reductions. The average reduction in the number of clauses after SIMPLIFY is 19%. This number varies significantly across these benchmarks. Figure 4 illustrates the reduction percentages. For each instance, 100% represents all the original clauses. Each bar is partitioned into the clauses reduced by SIMPLIFY (top), then the dominator-based reductions (middle), and the final clauses (bottom). The most dominator-based reductions (up to 53%) occur when a complete dominator with high fanout is replaced by a constant value, resulting in the elimination of even more circuitry using unit constraint propagation. This also explains the seeming inverse relationship between the number of reduced SODSes and the dominator-based reductions, because when a dominator is replaced by a constant, propagation already eliminates other dominators which might dominate the first one. In these instances, we note that the maximum size of $faninPI^*(\alpha)$ for a found complete dominator α was 3.

Looking at the last two columns of Table II, we can see that computing the sets $dom(v)$ takes a significant portion of the preprocessing run-time. This is because we use a simple algorithm [25], which has an $O(|V|^2)$ worst-case time complexity. Once these sets are computed, the time for finding complete dominators and replacing their SODSes is small.

Table III shows the results of the application of the QBF solvers sKizzo, 2clsQ and quantor on the instances after standard sim-

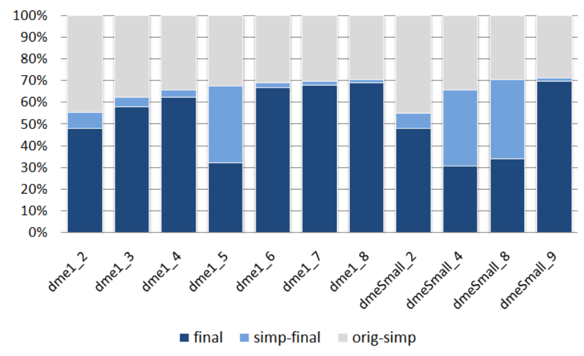

Fig. 4. Clause reduction percentages

TABLE III

QBF SOLVER EVALUATION

Benchmark	sKizzo time (s)		2clsQ time (s)		quantor time (s)	
	+sim	+dom	+sim	+dom	+sim	+dom
dme1_2	–	–	–	–	–	–
dme1_3	–	0.2	–	–	–	0.2
dme1_4	–	–	–	–	–	0.1
dme1_5	–	0.5	–	0.6	–	0.1
dme1_6	–	–	–	–	–	–
dme1_7	–	–	–	–	–	–
dme1_8	–	–	–	–	–	–
dmeSmall_2	–	16.2	–	–	–	–
dmeSmall_4	–	0.2	–	0.1	–	0.1
dmeSmall_8	–	0.6	–	0.7	–	0.1
dmeSmall_9	–	–	–	–	–	–
Summary	**0/11**	**5/11**	**0/11**	**3/11**	**0/11**	**5/11**

plifications only (column *+sim*) and after complete preprocessing using PReDom (column *+dom*). The effect of SIMPLIFY by itself is minimal because these QBF solvers already apply equivalent simplifications at the CNF level. sKizzo, 2clsQ and quantor time-out or mem-out on all instances where our techniques are not used to reduce SODSes, even with a time-out of 5 hours. On the other hand, they respectively solve $5/11$, $3/11$ and $5/11$ of the instances after preprocessing using PReDom, and the run-times are usually less than one second. Since none of the *+sim* runs terminate, our results were validated using the circuit-based QBF solver CirQit [12].

It should be noted that most of the QBF problems in the non-prenex non-CNF track of the QBFEVAL'08 competition did not have SODSes, which explains the small number of QBF instances used in our experiments. The experiments show the benefits of reducing SODSes, if they exist. On the other hand, even if no SODSes exist, the preprocessing time is negligeable. The authors are currently working on a circuit-based QBF solver which performs dominator-based reductions *on-the-fly*, rather than as a preprocessing step. This is much more powerful because variable assignments and circuit don't-cares in a search-based solver can create new SODSes at run-time.

VI. CONCLUSION AND FUTURE WORK

This work presents a novel way to exploit the circuit structure of a circuit-based QBF, using structural dominators. A rigorous proof is given for reducing subcircuits dominated by single outputs. We present our QBF preprocessor PReDom, which recursively applies this theory to return simpler but equisatisfiable QBF instances, in an effort to expedite the QBF solving process. Experimental results on circuit diameter computation problems show a significant increase in solved instances after preprocessing. In summary, we believe that this paper encourages further research in new strategies that exploit the circuit structure of QBFs to increase performance.

Directions for future work include on-the-fly dominator-based reductions in a circuit-based QBF solver, rather than as a preprocessing step. In many cases, the original circuit does not have single-output dominated subcircuits; however during a search-based QBF solving procedure, variable assignments and circuit don't-cares can produce complete dominators. The challenge there would be the efficient identification of those dominated subcircuits on-the-fly. On the other hand, a theoretical extension would be to deal with multiple-output dominators, which occur more frequently.

REFERENCES

[1] A. Sistla and E. Clarke, "The complexity of propositional linear temporal logics," *Journal of the ACM*, vol. 32, no. 3, pp. 733–749, 1985.

[2] J. Baumgartner, H. Mony, V. Paruthi, R. Kanzelman, and G. Janssen, "Scalable sequential equivalence checking across arbitrary design transformations," in *Int'l Conf. on Comp. Design*, 2006.

[3] A. T. Freitas, H. C. Neto, and A. L. Oliveira, "On the complexity of power estimation problems," in *Int'l Workshop on Logic and Synthesis*, 2000, pp. 239–244.

[4] N. Dershowitz, Z. Hanna, and J. Katz, "Bounded model checking with QBF," in *Int'l Conf. on Theory and Applications of Satisfiability Testing*, 2005, pp. 408–414.

[5] T. Jussila and A. Biere, "Compressing BMC encodings with QBF," in *Intl. Workshop on Bounded Model Checking*, 2006, pp. 1–14.

[6] H. Mangassarian, A. Veneris, S. Safarpour, M. Benedetti, and D. Smith, "A performance-driven QBF-based iterative logic array representation with applications to verification, debug and test," in *Int'l Conf. on CAD*, 2007, pp. 240–245.

[7] M. Benedetti, "sKizzo: a suite to evaluate and certify QBFs," in *Int'l Conf. on Automated Deduction*, 2005, pp. 369–376.

[8] A. Biere, "Resolve and expand," in *Int'l Conf. on Theory and Applications of Satisfiability Testing*, 2004, pp. 238–246.

[9] E. Giunchiglia, M. Narizzano, and A. Tacchella, "Qube: A system for deciding quantified boolean formulas satisfiability," in *IJCAR '01: Proceedings of the First International Joint Conference on Automated Reasoning*. London, UK: Springer-Verlag, 2001, pp. 364–369.

[10] H. Samulowitz and F. Bacchus, "Dynamically partitioning for solving QBF," in *Int'l Conf. on Theory and Applications of Satisfiability Testing*, 2007, pp. 215–229.

[11] D. Tang and S. Malik, "Solving quantified boolean formulas with circuit observability don't cares," in *9th International Conference on Theory and Applications of Satisfiability Testing (SAT), Lecture Notes in Computer Science (LNCS), Volume 4121/2006*, August 2006.

[12] A. Goultiaeva, V. Iverson, and F. Bacchus, "Beyond CNF: A circuit-based QBF solver," in *Int'l Conf. on Theory and Applications of Satisfiability Testing*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 412–426.

[13] S. Safarpour, A. Veneris, R. Drechsler, and J. Hang, "Managing don't cares in Boolean satisfiability," in *Design, Automation and Test in Europe*, 2004, pp. 260–265.

[14] Q. Zhu, N. Kitchen, A. Kuehlmann, and A. Sangiovanni-Vincentelli, "Sat sweeping with local observability don't-cares," in *Design Automation Conf.*, 2006, pp. 229–234.

[15] L. Zhang, "Solving QBF with combined conjunctive and disjunctive normal forms," in *National Conference on Artificial Intelligence (AAAI)*, 2006.

[16] A. Sabharwal, C. Ansótegui, C. P. Gomes, J. W. Hart, and B. Selman, "QBF modeling: Exploiting player symmetry for simplicity and efficiency," in *Int'l Conf. on Theory and Applications of Satisfiability Testing*, 2006, pp. 382–395.

[17] F. Lonsing and A. Biere, "Nenofex: Expanding NNF for QBF solving," in *Int'l Conf. on Theory and Applications of Satisfiability Testing*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 196–210.

[18] U. Egly, M. Seidl, and S. Woltran, "A solver for QBFs in negation normal form," *Constraints*, vol. 14, no. 1, pp. 38–79, March 2009.

[19] T. Kirkland and M. R. Mercer, "A topological search algorithm for ATPG," in *Design Automation Conf.*, 1987, pp. 502–508.

[20] J. Cong and Y. Ding, "An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," *IEEE Trans. on CAD*, vol. 13, pp. 1–12, 1994.

[21] H. Samulowitz, J. Davies, and F. Bacchus, "Preprocessing QBF," in *Int'l Conf. on Principles and Practice of Constraint Programming*, 2006, pp. 514–529.

[22] E. Giunchiglia, M. Narizzano, and A. Tacchella, "Quantifier structure in search-based procedures for QBFs," *IEEE Trans. on CAD*, vol. 26, no. 3, pp. 497–507, 2007.

[23] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV 2: An opensource tool for symbolic model checking," in *Computer Aided Verification*, 2002, pp. 359–364.

[24] G. S. Tseitin, "On the complexity of derivations in the propositional calculus," in *Studies in Constructive Mathematics and Mathematical Logic*. New York - London: Part 2. Consultants Bureau, 1968, pp. 115–125.

[25] F. E. Allen and J. Cocke, "Graph-theoretic constructs for program flow analysis," Technical Report RC 3923 (17789), IBM Thomas J. Watson Research Center, Tech. Rep., 1972.

[26] L. Georgiadis and R. E. Tarjan, "Finding dominators revisited: extended abstract," in *SODA*, 2004, pp. 869–878.