Chapter 7
Analysis of Incomplete Circuits Using Dependency Quantified Boolean Formulas

Ralf Wimmer, Karina Wimmer, Christoph Scholl, and Bernd Becker

1 Introduction

Solver-based techniques have proven to be successful in many areas in computer-aided design, ranging from formal verification of digital circuits [1, 3, 9, 29] over automatic test pattern generation [11, 13] to circuit synthesis [4, 5]. While research on solving quantifier-free Boolean formulas (the famous SAT-problem [10]) has reached a certain level of maturity, designing and improving algorithms for quantified Boolean formulas (QBFs) is one focus of active research. However, there are applications like the verification of partial circuits [18, 19, 29], the synthesis of safe controllers [4], and the analysis of games with incomplete information [26] for which QBF is not expressive enough to provide a compact and natural formulation. The reason is that QBF requires linearly ordered dependencies of the existential variables on the universal ones: Each existential variable implicitly depends on all universal variables in whose scope it is. Relaxing this condition yields so-called *dependency quantified Boolean formulas (DQBFs)*. DQBFs are strictly more expressive than QBFs in the sense that an equivalent QBF formulation can be exponentially larger than a DQBF formulation. This comes at the price of a higher complexity of the decision problem: DQBF is NEXPTIME-complete [26], compared to QBF, which is "only" PSPACE-complete. Encouraged by the success of SAT and QBF solvers and driven by the mentioned applications, research on solving DQBFs has started during the last few years [16, 17, 20, 33], yielding first prototypic solvers like iDQ [17] and HQS [20].

R. Wimmer (✉) • K. Wimmer • C. Scholl • B. Becker
Institute of Computer Science, Albert-Ludwigs-Universität Freiburg, Freiburg im Breisgau, Germany
e-mail: wimmer@informatik.uni-freiburg.de; wimmerka@informatik.uni-freiburg.de; scholl@informatik.uni-freiburg.de; becker@informatik.uni-freiburg.de

A.I. Reis, R. Drechsler (eds.), *Advanced Logic Synthesis*,
https://doi.org/10.1007/978-3-319-67295-3_7

In this paper, we focus on the application of DQBF for analyzing *incomplete combinational and sequential circuits*. Such incomplete circuits appear in early design stages, when only a subset of the system's modules has already been implemented and verification is applied in order to find errors in the available parts as early as possible. Incomplete circuits also result if the complexity of the verification task is too high and therefore some parts, which are supposed not to influence the validity of some properties, e. g., multiplier or memory modules, have been removed to make verification feasible. Analyzing incomplete circuits is also useful if a designer wants to localize errors (then one can remove parts of the design and if for all possible implementations of the removed parts the error does not disappear, the remaining parts must be erroneous). Therefore this problem has received considerable attention in the research community during the last 15 years, see, e. g., [12, 14, 21, 25, 29–31]. All solver-based approaches are restricted in the sense that they can either only handle a single black box or do not take the interfaces of the black boxes into account, allowing the black boxes to read signals which are not available to them in the actual design.

We show how the realizability problem for incomplete combinational and sequential circuits with an arbitrary number of combinational or bounded-memory black boxes can be expressed as a DQBF. Here we show for the first time a DQBF-based solution for sequential circuits with several bounded-memory black boxes where the exact interface of the black boxes, i.e., the signals entering and leaving the black boxes, can be taken into account. We also show that solving a DQBF has the same complexity as deciding realizability. We do not only sketch how DQBFs are solved in our DQBF solver HQS [20, 33], but also how so-called Skolem functions can be obtained from the solution process, provided that the formula is satisfied [34]. These Skolem functions can directly serve as an implementation of the black boxes.

This paper builds on different sources: [18, 19] applies DQBF-based methods to incomplete combinational circuits with combinational black boxes. SAT- and QBF-based techniques for controller synthesis are considered in [4]; there a footnote gives hints how a DQBF formulation can be used for that purpose. Due to the lack of efficient DQBF solvers at that time, this idea was not investigated further. However the method described there considers only a single black box which can read all primary inputs and the complete state information. The basic techniques implemented in our DQBF solver HQS have been described in [20], and [33] defines preprocessing techniques for DQBF, which speed up the solution process considerably.

1.1 Structure of the Paper

In the next section, we introduce dependency quantified Boolean formulas (DQBFs). In Sect. 3, we describe how realizability of incomplete combinational and sequential circuits can be formulated as a DQBF. Section 4 presents a method to solve DQBFs and to obtain Skolem functions for satisfied DQBFs. In Sect. 5 we give preliminary experimental results, and we conclude the paper in Sect. 6, pointing out challenges which need to be solved.

2 Foundations

Let φ and κ be quantifier-free Boolean formulas over the set V of Boolean variables and $v \in V$. We denote by $\varphi[\kappa/v]$ the Boolean formula which results from φ by replacing all occurrences of v (simultaneously) by κ. For a set $V' \subseteq V$, we denote by $\mathcal{A}(V')$ the set of Boolean assignments for V', i.e., $\mathcal{A}(V') = \{\nu \mid \nu : V' \to \{0,1\}\}$. For each quantifier-free formula φ over V, a variable assignment ν to the variables in V induces a truth value 0 or 1 of φ, which we call $\nu(\varphi)$.

Definition 1 (Syntax of DQBF) Let $V = \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$ be a set of Boolean variables. A *dependency quantified Boolean formula* (DQBF) ψ over V has the form $\psi := \forall x_1 \ldots \forall x_n \exists y_1(D_{y_1}) \ldots \exists y_m(D_{y_m}) : \varphi$, where $D_{y_i} \subseteq \{x_1, \ldots, x_n\}$ for $i = 1, \ldots, m$ is the *dependency set* of y_i, and φ is a quantifier-free Boolean formula over V, called the *matrix* of ψ.

$V_\psi^\forall = \{x_1, \ldots, x_n\}$ denotes the set of universal and $V_\psi^\exists = \{y_1, \ldots, y_m\}$ the set of existential variables. We often write $\psi = Q : \varphi$ with the quantifier prefix Q and the matrix φ. $Q \setminus \{v\}$ denotes the prefix that results from removing a variable $v \in V$ from Q together with its quantifier. If v is existential, then its dependency set is removed as well; if v is universal, then all occurrences of v in the dependency sets of existential variables are removed. Similarly we use $Q \cup \{\exists y(D_y)\}$ to add existential variables to the prefix. We sometimes assume that a DQBF $\psi = Q : \varphi$ as in Definition 1 with φ in conjunctive normal form (CNF) is given. A formula is in CNF if it is a conjunction of (non-tautological) *clauses*; a clause is a disjunction of *literals*, and a literal is either a variable v or its negation $\neg v$. As usual, we identify a formula in CNF with its set of clauses and a clause with its set of literals. For a formula φ (resp. clause C, literal ℓ), $\mathrm{var}(\varphi)$ (resp. $\mathrm{var}(C)$, $\mathrm{var}(\ell)$) means the set of variables occurring in φ (resp. C, ℓ), $\mathrm{lit}(\varphi)$ ($\mathrm{lit}(C)$) means the set of literals occurring in φ (C).

A quantified Boolean formula (QBF) (in prenex normal form) is a DQBF such that $D_y \subseteq D_{y'}$ or $D_{y'} \subseteq D_y$ holds for any two existential variables $y, y' \in V_\psi^\exists$. Then the variables in V can be ordered resulting in a linear quantifier prefix, such that for each $y \in V_\psi^\exists$, D_y equals the set of universal variables which are to the left of y.

The semantics of a DQBF is usually defined by so-called Skolem functions.

Definition 2 (Semantics of DQBF) Let ψ be a DQBF as above. It is *satisfiable*, iff there are functions $s_y : \mathcal{A}(D_y) \to \mathbb{B}$ for $y \in V_\psi^\exists$ such that replacing each $y \in V_\psi^\exists$ by (a Boolean expression for) s_y turns φ into a tautology. The functions $(s_y)_{y \in V_\psi^\exists}$ are called *Skolem functions* for ψ.

Example 1 Consider the following DQBF:

$$\forall x_1 \forall x_2 \exists y_1(x_1) \exists y_2(x_2) : (x_1 \vee \neg y_1) \wedge (x_2 \vee y_1 \vee y_2)$$

Here the variable y_1 depends only on x_1, but not on x_2; y_2 depends only on x_2, but not on x_1. It is satisfied by using the Skolem functions $s_{y_1}(x_1) = x_1$ and $s_{y_2}(x_2) = \neg x_2$. Replacing y_1 and y_2 by their Skolem functions yields $(x_1 \vee \neg x_1) \wedge (x_2 \vee x_1 \vee \neg x_2)$, which is obviously a tautology.

3 Analysis of Incomplete Circuits

In this section, we show how DQBFs can be used to analyze incomplete combinational and sequential circuits. In both cases we ask for realizability: Are there implementations of the missing parts ("black boxes") such that the complete circuit satisfies its specification.

We assume that the missing parts are either combinational or contain only a bounded amount of memory. In the latter case, we can put the flipflops of the black boxes into the available circuit part such that the incoming and outgoing signals of these flipflops are written and read only by the black boxes as sketched in Fig. 7.1.

Then the black boxes themselves are purely combinational. Note that the case of several black boxes with an *unbounded* amount of memory is undecidable [28].

We use the notation for incomplete sequential circuits as sketched in Fig. 7.2. The primary inputs are denoted by $\mathbf{x}$, the current state by $\mathbf{s}$, and the next state by $\mathbf{s}'$. The missing parts are $\mathrm{BB}_1, \ldots, \mathrm{BB}_n$, whose interfaces, i.e., the signals entering and leaving the black boxes, are known. The input signals of black box BB_i are denoted by $\mathbf{I}_i$, its output signals by $\mathbf{y}_i$. The input cone of black box BB_i ensures the constraint $\mathbf{I}_i \equiv \mathbf{F}_i(\mathbf{s}, \mathbf{x}, \mathbf{y}_1, \ldots, \mathbf{y}_{i-1})$, the next state is described by trans $:=$

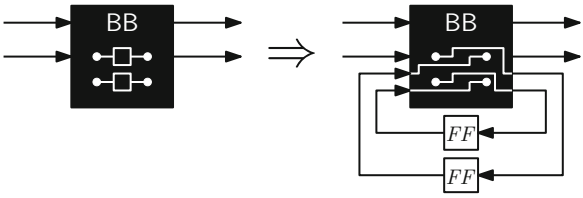

Fig. 7.1 Sequential circuits with extracted memory

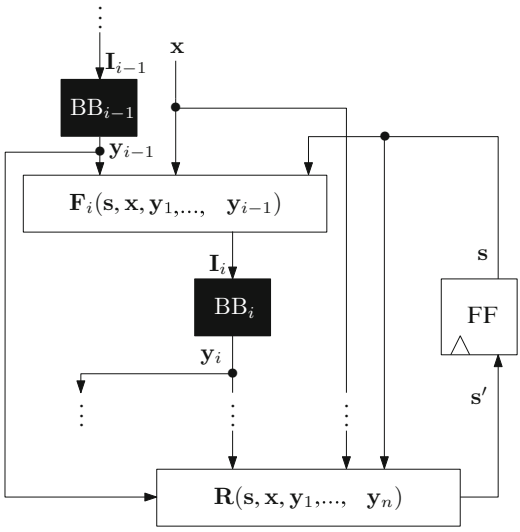

Fig. 7.2 Notation for incomplete sequential circuits

$\left(\mathbf{s}' \equiv \mathbf{R}(\mathbf{s}, \mathbf{x}, \mathbf{y}_1, \ldots, \mathbf{y}_n)\right)$. We assume w. l. o. g. that no black box output is directly connected to an input of another black box or a flipflop, i.e., $\mathbf{y}_i \cap \mathbf{I}_j = \emptyset$ for all i, j and $\mathbf{s}' \cap \mathbf{y}_j = \emptyset$ for all j. Otherwise a buffer is inserted between the two black boxes without changing the functionality of the circuit. Additionally we assume that there are no cyclic dependencies between the combinational black boxes, i.e., that BB_i only depends on the outputs of $\mathrm{BB}_1, \ldots, \mathrm{BB}_{i-1}$. Otherwise, implementing the black boxes with combinational circuits can lead to cycles in the combinational part of the circuit which do not run through memory elements. This can cause undefined behavior of the circuit.

We consider invariant properties $\mathrm{inv}(\mathbf{s}, \mathbf{x}, \mathbf{y}_1, \ldots, \mathbf{y}_n)$, defined over the primary inputs $\mathbf{x}$, the current state $\mathbf{s}$ and the black box outputs $\mathbf{y}_1, \ldots, \mathbf{y}_n$, which are required to hold at any time.

3.1 Combinational Circuits

The same notation as introduced above is also used for combinational circuits. Here, the state and next state signals $\mathbf{s}$ and $\mathbf{s}'$ as well as the memory elements are omitted.

Definition 3 The *partial equivalence checking problem (PEC)* is defined as follows: Given an incomplete circuit C_{impl} and a (complete) specification C_{spec}, are there implementations of the black boxes in C_{impl} such that C_{impl} and C_{spec} become equivalent?

In the following, we assume that (incomplete) implementation C_{impl} and specification C_{spec} are combined into a single circuit using a miter construction: Corresponding primary inputs are connected, corresponding outputs are connected via XOR gates. The outputs of the XOR gates are combined via OR gates into a single output signal. This output signal is constantly one iff, for some implementation of the black boxes, the two circuits are equivalent. This can be considered as a kind of invariant property, valid at the primary output of the combined circuit.

We now show how a DQBF formulation can be used to decide PEC.

Consider a PEC problem with black boxes $\mathrm{BB}_1, \ldots, \mathrm{BB}_n$. We first construct the quantifier prefix of the DQBF. The primary inputs $\mathbf{x}$ and the black box inputs $\mathbf{I}_1, \ldots, \mathbf{I}_n$ are universally quantified, all other variables are existentially quantified. The dependency set of black box outputs $\mathbf{y}_i$ contains exactly the inputs $\mathbf{I}_i$ of BB_i. Hence the quantifier prefix is

$$\forall \mathbf{x} \forall \mathbf{I}_1 \ldots \forall \mathbf{I}_n \exists \mathbf{y}_1(\mathbf{I}_1) \ldots \exists \mathbf{y}_m(\mathbf{I}_m) \,.$$

If the black boxes are not directly connected to the primary inputs but to internal signals, we have to take into account that not all possible combinations of values may arrive at the inputs of the black boxes. Since we use a universal quantification for the black box inputs we have to ensure that our formula is satisfied if the

value of the black box inputs $\mathbf{I}_i$ deviates from the values obtained as a function $F_i(\mathbf{x}, \mathbf{I}_1, \ldots, \mathbf{I}_{i-1})$.

$$\varphi(\mathbf{x}, \mathbf{I}_1, \ldots, \mathbf{I}_n, \mathbf{y}_1, \ldots, \mathbf{y}_n) := \left(\bigwedge_{i=1}^{n} \mathbf{I}_i \equiv \mathbf{F}_i \right) \Rightarrow \text{inv}(\mathbf{x}, \mathbf{y}_1, \ldots, \mathbf{y}_n) \,.$$

This formula is not necessarily given in CNF. By applying Tseitin transformation [32], which is essentially introducing auxiliary variables for the internal signals of the circuit, one can obtain a CNF φ' that is equisatisfiable to φ and whose size is linear in the size of φ. Let $\mathbf{a}$ be the vector of these auxiliary variables, which are existentially quantified in the quantifier prefix. As their values are implied by the values of the variables in their input cone, we can use as their dependency sets either the universal variables in their input cone or the set of all universal variables (or any set in between). We prefer making the Tseitin variables depend on all universal variables, because this typically leads to DQBFs that are easier to solve.

The resulting DQBF is:

$$\psi := \forall \mathbf{x} \forall \mathbf{I}_1 \ldots \forall \mathbf{I}_n \exists \mathbf{y}_1(\mathbf{I}_1) \ldots \exists \mathbf{y}_n(\mathbf{I}_n) \exists \mathbf{a}(\mathbf{x}, \mathbf{I}_1, \ldots, \mathbf{I}_n) : \varphi'(\mathbf{x}, \mathbf{I}_1, \ldots, \mathbf{I}_n, \mathbf{y}_1, \ldots, \mathbf{y}_n, \mathbf{a}) \,.$$

This formula ψ is satisfied iff we can replace all $\mathbf{y}_i(\mathbf{I}_i)$ with Skolem functions $s_i(\mathbf{I}_i)$ such that φ' becomes a tautology. The Skolem functions s_i exist if and only if there are implementations for the black boxes BB_i of the PEC, such that the PEC is satisfied. Therefore any PEC can be translated with linear effort into a DQBF and the PEC is satisfied iff the DQBF is satisfied.

It is easy to see that there is also the converse transformation [19]: Each DQBF can be turned into a PEC, having one black box for each existential variable such that the PEC is realizable iff the DQBF is satisfiable. This implies that PEC is NEXPTIME-complete.

3.2 Sequential Circuits

For incomplete sequential circuits with multiple combinational or bounded-memory black boxes, we investigate the following problem:

Definition 4 The *realizability problem for incomplete sequential circuits (RISC)* is defined as follows: Given an incomplete sequential circuit with multiple combinational (or bounded-memory) black boxes and an invariant property, are there implementations of the black boxes such that in the complete circuit the invariant holds at all times?

To decide RISC, one can apply a generalization of ideas described in [4] for the synthesis of controllers (which are in fact single black boxes with access to all state bits and all primary circuit inputs).

According to the notations introduced in the previous section, let $\mathbf{s}$ denote the variables encoding the current state of the circuit, $\mathbf{s}'$ the next state, and $\mathbf{x}$ the primary inputs. The formulas $F_1, \ldots, F_n$ describe the input cones of the black boxes, $\mathbf{I}_1, \ldots, \mathbf{I}_n$ their inputs, $\mathbf{y}_1, \ldots, \mathbf{y}_n$ their outputs, and R is the next state function of the circuit. Additionally we assume that init represents the circuit's initial state(s) and inv its states that satisfy the invariant.

Definition 5 A subset $W \subseteq S$ is a *winning set* if all states in W satisfy the invariant and, for all values of the primary inputs, the black boxes can ensure (by computing appropriate values) that the successor state is again in W.

A given RISC is realizable if there is a winning set that includes the initial state of the circuit. This can be formulated as a DQBF. To encode the winning sets, we introduce two existential variables w and w'; w depends on the current state and is supposed to be true for a state $\mathbf{s}$ if $\mathbf{s}$ is in the winning set. The variable w' depends on the next state variables $\mathbf{s}'$ and has the same Skolem function as w (but defined over $\mathbf{s}'$ instead of $\mathbf{s}$). We ensure that w and w' have the same semantics by the condition $\big(\mathbf{s} \equiv \mathbf{s}' \Rightarrow w \equiv w'\big)$.

Similar to the combinational case, we have to take into account that the black boxes are typically not directly connected to the primary inputs, but to internal signals. This is done by restricting the requirement that the successor state is again a winning state to the case when the black box inputs are assigned consistently with the values computed by their input cones.

Theorem 1 *Given a RISC as defined above, the following DQBF is satisfied if and only if the RISC is realizable:*

$$\begin{aligned}\forall \mathbf{s}\, \forall \mathbf{s}' \forall \mathbf{x}\, \forall \mathbf{I}_1 \ldots \forall \mathbf{I}_n\, \exists \mathbf{y}_1(\mathbf{I}_1) \ldots \exists \mathbf{y}_n(\mathbf{I}_n)\, \exists w(\mathbf{s})\, \exists w'(\mathbf{s}') :\\ \big(\text{init} \Rightarrow w\big) \wedge \big(w \Rightarrow \text{inv}\big) \wedge \big(\mathbf{s} \equiv \mathbf{s}' \Rightarrow w \equiv w'\big)\\ \wedge \Big(\big(w \wedge \bigwedge_{i=1}^{n} \mathbf{I}_i \equiv \mathbf{F}_i \wedge \mathbf{s}' \equiv R\big) \Rightarrow w'\Big).\end{aligned}$$

Theorem 2 *RISC is NEXPTIME-complete.*

Proof The reduction to DQBF above shows that RISC is in NEXPTIME. To show the hardness, we give a reduction from DQBF to RISC. First we transform the DQBF into an incomplete combinational circuit as shown in [18] such that the output of the circuit is constantly 1 iff the DQBF is satisfied. We now turn this combinational circuit into a sequential circuit with two states by storing the output of the combinational circuit in a 1-bit flipflop s. The initial state is $s \equiv 1$, the invariant is given by $s \equiv 1$. The original DQBF is satisfied iff the unsafe state 0 can be made unreachable by appropriate black box implementations. □

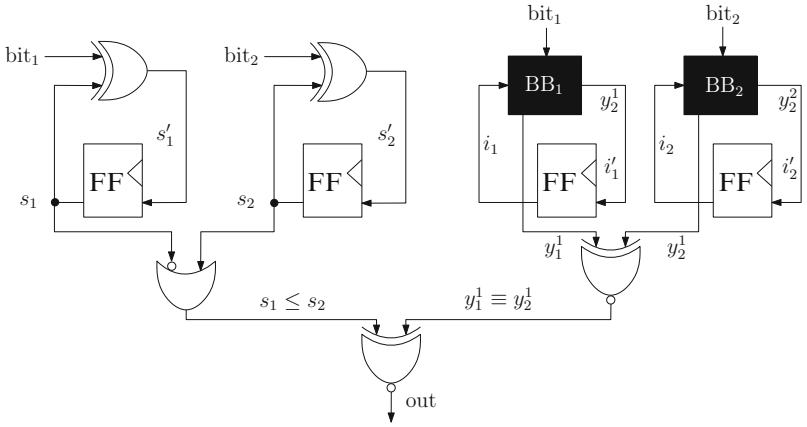

Fig. 7.3 Sequential circuit with *two black boxes*

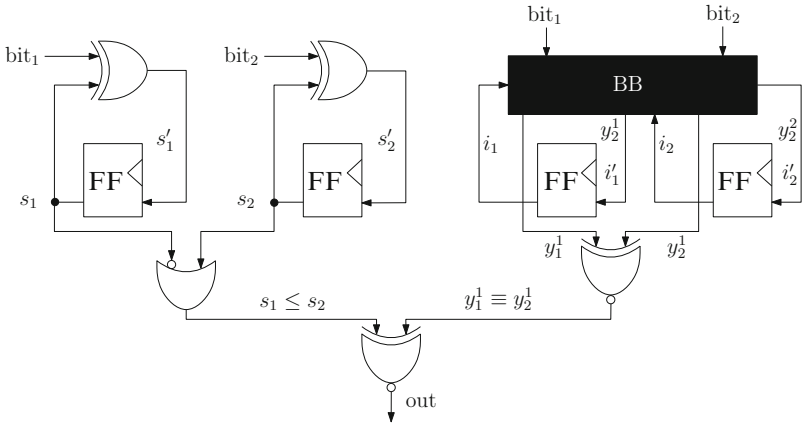

Fig. 7.4 The same sequential circuit as in Fig. 7.3, but with a *single black box*

Example 2 We illustrate the solution of RISC using two incomplete circuits in Figs. 7.3 and 7.4. The circuits are simple, but still illustrate the basic idea. We first start with the circuit in Fig. 7.3. The sequential circuit in Fig. 7.3 consists of two parts. The first part on the left can be seen as the specification for a simple sequential circuit: There are two bit streams applied to the inputs bit_1 and bit_2. The circuit computes the parities of the bit streams applied to bit_1 and bit_2 and outputs 1 iff the parity for bit stream bit_1 is smaller or equal to the parity for bit stream bit_2. The right-hand side shows a given architecture for an implementation with two black boxes, one reading bit stream bit_1 and the other reading bit stream bit_2. The outputs of the black boxes are connected by an equivalence gate. Then the output of the overall circuit is computed by an equivalence gate connecting the outputs of specification and incomplete implementation. We require the invariant property that the output

of the overall circuit is 1 at all times, i.e., that the black boxes are implemented in a way such that the implementation part agrees with the specification part. For this simple example it is easy to see that a corresponding implementation does not exist, even for black boxes with unbounded memory. Here we use our method where the number of flip flops for each black box is restricted to one. Figure 7.3 already shows the transformed circuit where the memory is extracted from the black boxes.

Applying Theorem 1, we obtain the following formula parts:

• initial state:

$$\text{init} := (\neg s_1 \wedge \neg s_2 \wedge \neg i_1 \wedge \neg i_2)$$

• transition relation:

$$\begin{aligned}\text{trans} := (s_1' \equiv s_1 \oplus \text{bit}_1) \wedge (s_2' \equiv s_2 \oplus \text{bit}_1)\\ \wedge (i_1' \equiv y_2^1) \wedge (i_2' \equiv y_2^2)\end{aligned}$$

• invariant:

$$\text{inv} := \big(\neg s_1 \vee s_2) \equiv (y_1^1 \equiv y_2^1)\big)$$

Putting these parts together yields the following DQBF:

$$\begin{aligned}&\forall \text{bit}_1 \forall \text{bit}_2 \forall s_1 \forall s_1' \forall s_2 \forall s_2' \forall i_1 \forall i_1' \forall i_2 \forall i_2'\\ &\qquad \exists y_1^1(i_1, \text{bit}_1) \exists y_2^1(i_1, \text{bit}_1) \exists y_1^2(i_2, \text{bit}_2) \exists y_2^2(i_2, \text{bit}_2)\\ &\qquad\qquad \exists w(s_1, s_2, i_1, i_2) \exists w'(s_1', s_2', i_1', i_2') :\\ &\qquad (\text{init} \Rightarrow w) \wedge (w \Rightarrow \text{inv}) \wedge \big((w \wedge \text{trans}) \Rightarrow w'\big)\\ &\qquad \wedge \big(((s_1 \equiv s_1') \wedge (s_2 \equiv s_2') \wedge (i_1 \equiv i_1') \wedge (i_2 \equiv i_2')) \Rightarrow (w \equiv w')\big)\end{aligned}$$

By applying a DQBF solver, one can verify that this formula is unsatisfiable, meaning that the design in Fig. 7.3 is not realizable.

Now consider the circuit in Fig. 7.4. It differs from the design in Fig. 7.3 only in the black boxes: Both black boxes can read both input signals bit_1 and bit_2. Thus, the black boxes can equivalently be merged into one as shown in Fig. 7.4. It is easy to see that this implementation, which does not pay attention to the exact architecture by disregarding the interface of the black boxes, is now realizable. More precisely, it is realizable if we assume that the black box with bounded memory has at least 2 memory cells at its disposal. In Fig. 7.4 we depict the incomplete circuit with two memory cells extracted from the black box. Using our approach we can indeed

prove realizability. The formula differs only in the dependency sets of the black box outputs: $D_{y_1^1} = D_{y_2^1} = D_{y_1^2} = D_{y_2^2} = \{\text{bit}_1, \text{bit}_2, i_1, i_2\}$. Now the formula is satisfiable. The following Skolem functions turn the matrix into a tautology:

Variable	y_1^1	y_2^1	y_1^2	y_2^2	w	w'
Skolem function	1	$\text{bit}_1 \oplus i_1$	$\neg i_1 \vee i_2$	$\text{bit}_2 \oplus i_2$	1	1

Using these Skolem functions, the two flipflops in the right half store the same values as the two flipflops in the left half, i.e., $s_1 = i_1$ and $s_2 = i_2$. The equivalence $y_1^1 \equiv y_2^1$ corresponds to $1 \equiv (\neg i_1 \vee i_2)$, which is the same as $i_1 \leq i_2$. Therefore the design is realizable.

For both incomplete circuits, our solver HQS [20] solved the DQBF in at most 0.1 s.

We can conclude that it is necessary to take the precise interfaces of the black boxes into account in order to obtain a valid answer whether the design is realizable.

4 Solving DQBFs

Elimination-based DQBF solvers like HQS [20, 33] apply a series of satisfiability-preserving transformation steps to the formula until a SAT or QBF problem results, which can be solved by an arbitrary SAT or QBF solver. As a pure yes/no answer is not satisfactory when solving analysis problems as presented in the previous section, we provide the main ideas how Skolem functions can be extracted from the solution process. More details can be found in [34, 35]. This extraction proceeds in the reverse order of transformation, starting with (constant) Skolem functions for the final SAT problem, which correspond to a satisfying assignment.

4.1 Transformation Steps

The central operation of elimination-based solvers is the elimination of existential and universal variables from the formula. QBF solvers can eliminate variables in the order given by the quantifier prefix (starting with the inner-most variable block). Because there is no linear order on the variables in a DQBF, this is typically no longer possible.

Lemma 1 *Let $\psi = Q : \phi$ be a DQBF and $y \in V_\psi^\exists$ an existential variable which depends on all universal variables. Then ψ is equisatisfiable to $\psi' := Q \setminus \{\exists y(D_y)\} : \phi[0/y] \vee \phi[1/y]$.*

If $s_z^{\psi'}$ for $z \in V_{\psi'}^{\exists}$ are Skolem functions for the formula ψ', obtained by eliminating $y \in V_{\psi}^{\exists}$, we set $s_y^{\psi} := \phi[1/y][s_z^{\psi'}/z]$ and $s_z^{\psi} := s_z^{\psi'}$ for $z \neq y$. This yields Skolem functions for ψ [34].

The elimination of universal variables in solvers like HQS [20] is done by *universal expansion* [2, 7, 8, 19]. This is applicable even if some existential variables depend on the expanded universal one.

Lemma 2 *For a DQBF* $\psi = \forall x_1 \ldots \forall x_n \exists y_1(D_{y_1}) \ldots \exists y_m(D_{y_m}) : \varphi$ *with* $E_{x_i} = \left\{ y_j \in V_{\psi}^{\exists} \,\middle|\, x_i \in D_{y_j} \right\}$, *the* universal expansion *w. r. t. variable* $x_i \in V_{\psi}^{\forall}$, *is defined by*

$$\left(Q \setminus \{x_i\}\right) \cup \left\{ \exists y_j'(D_{y_j} \setminus \{x_i\}) \,\middle|\, y_j \in E_{x_i} \right\} : \varphi[1/x_i] \wedge \varphi[0/x_i][y_j'/y_j \text{ for all } y_j \in E_{x_i}] .$$

That means, when eliminating a universal variable $x \in V_{\psi}^{\forall}$, we have to copy all existential variables $y \in V_{\psi}^{\exists}$ that depend on x.

Assume that $s_z^{\psi'}$ for $z \in V_{\psi'}^{\exists}$ are Skolem functions for ψ'. Then $s_y^{\psi} := (x \wedge s_{y'}^{\psi'}) \vee (\neg x \wedge s_y^{\psi'})$ for $y \in V_{\psi}^{\exists}$ with $x \in D_y$ and $s_z^{\psi} := s_y^{\psi'}$ for $z \in V_{\psi}^{\exists}$ with $x \notin D_z$ are Skolem functions for ψ [34].

In principle, these two operations suffice to turn each DQBF into an (exponentially larger) SAT problem. In order to reduce computation time and memory consumption, pre- and inprocessing steps have turned out to be essential.

Standard operations are the detection of unit and pure literals. A literal ℓ is unit if (ℓ) is a clause in the formula. A literal ℓ is pure if $\neg\ell$ does not appear in the formula. In both cases $\mathrm{var}(\ell)$ can be replaced by a constant (which is also the Skolem function for that variable). Further preprocessing techniques like blocked clause elimination, the identification of equivalent variables, and structure extraction have been devised for DQBF [33, 36]. All of them are supported when Skolem functions are to be computed. We refer to [34] for more details.

4.2 Elimination Sets

Since the expansion of all universal variables leads to an exponentially larger SAT instance, this is typically not feasible. Instead, the solver HQS eliminates variables only until a QBF is obtained, which can be solved by an arbitrary QBF solver. The central problem is to determine a minimum set of universal variables whose elimination turns the DQBF into a QBF [20]. To solve this, we can use the following dependency graph:

Definition 6 Let ψ be a DQBF. Its *dependency graph* $G_\psi = (V_\psi^\exists, E_\psi)$ is a directed graph with the existential variables as its nodes and edges $E_\psi = \{(y, z) \in V_\psi^\exists \times V_\psi^\exists \mid D_y \not\subseteq D_z\}$.
It can be used to recognize if a DQBF is actually a QBF:

Lemma 3 *Let ψ be a DQBF. Its dependency graph G_ψ is acyclic iff ψ has an equivalent QBF prefix.*
That means we have to find a minimum set $U \subseteq V_\psi^\forall$ of universal variables whose elimination makes G_ψ acyclic. One can show that for making the graph acyclic by eliminating universal variables, it suffices to consider the cycles of length 2. The selection of variables can be done using a MAXSAT solver: for each universal variable x a variable m_x is created in the MAXSAT solver such that $m_x = 1$ means that x needs to be eliminated. Soft clauses are used to get an assignment with a minimum number of variables assigned to 1. Hard clauses enforce that for all $y, z \in V_\psi^\exists$, $y \neq z$, either all variables in $D_y \setminus D_z$ or in $D_z \setminus D_y$ are eliminated.

The variables in U are then eliminated, ordered according to the number of existential variables that depend upon them.

For more details, including formal correctness proofs, we refer the reader to [20].

4.3 Solver Overview

Figure 7.5 shows the structure of the general-purpose DQBF solver HQS. The input is a DQBF in CNF. After preprocessing, which is done on the CNF, gate detection is applied, essentially undoing Tseitin transformation and removing the existential variables introduced by the CNF transformation. The result is a representation of the

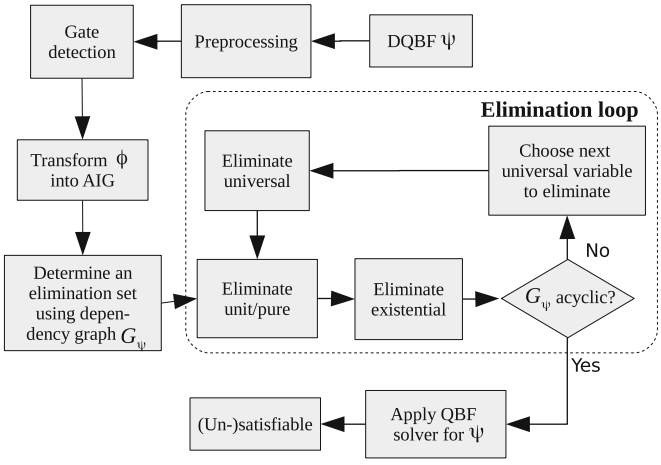

Fig. 7.5 Structure of the solver (adapted from [20])

formula as an And-Inverter graph (AIG), on which the further steps are performed. Before the actual elimination loop starts, we determine a minimum elimination set as described above.

Within the elimination loop, we check for unit and pure variables, which can be replaced by constants. This is done on the AIG using syntactic checks. Additionally, all existential variables are eliminated for which this is possible. Otherwise they would double for each eliminated universal variable. Then we check if the dependency graph has already become acyclic. If this is the case we generate the corresponding QBF prefix and solve the formula using the QBF solver AIGsolve [27], which operates directly on AIGs. Otherwise we select the next universal variable to eliminate and expand it.

5 Experimental Results

In the following, we present preliminary experimental results for incomplete combinational circuits. To solve the DQBFs, we use our elimination-based DQBF solver HQS [20], which was described briefly in the previous section.

We have extended HQS by the possibility to compute Skolem functions for satisfied DQBFs. The computation of Skolem functions works in two phases: During the solution process we collect the necessary data and store it on a stack. When the satisfiability of the formula has been determined, we free the other data structures of the solver and extract the Skolem functions from the collected data. We can apply don't-care minimization to the Skolem functions, based on Craig interpolants [24], and use the tool ABC [6] for further minimization of the Skolem functions' AIG representation.

All experiments were run on one Intel Xeon E5-2650v2 CPU core at 2.60 GHz clock frequency and 64 GB of main memory under Ubuntu Linux as operating system. We aborted all experiments which either took more than 1000 s CPU time or more than 8 GB ($= 2^{30}$ bytes) of main memory. As benchmarks we used 4318 DQBF instances from different sources. Most of them are DQBFs resulting from equivalence checking of incomplete combinational circuits [15, 17, 19]. The remaining ones are controller synthesis problems [4]. The sizes of the circuits range from a few hundred to a few thousand gates. The incomplete circuits contain one to five randomly selected black boxes. The controller synthesis problems are essentially sequential circuits with a single black box.

We first compare the efficiency of HQS with the only other available DQBF solver iDQ [17], which solves the formula by iteratively solving SAT instances generated from the DQBF. Both solvers were run after preprocessing the DQBFs. Since iDQ relies on a formula in CNF, while HQS does not, different preprocessing operations had to be applied: besides standard techniques like the detection of unit and pure literals and equivalent variables, preprocessing for HQS applies gate detection, which reverses Tseitin transformation in order to reconstruct the original circuit. This is not possible in case of iDQ. Instead, for iDQ, we apply blocked clause elimination and variable elimination by resolution, which are both not beneficial for HQS. We refer the reader to [33, 36] for more details on DQBF preprocessing.

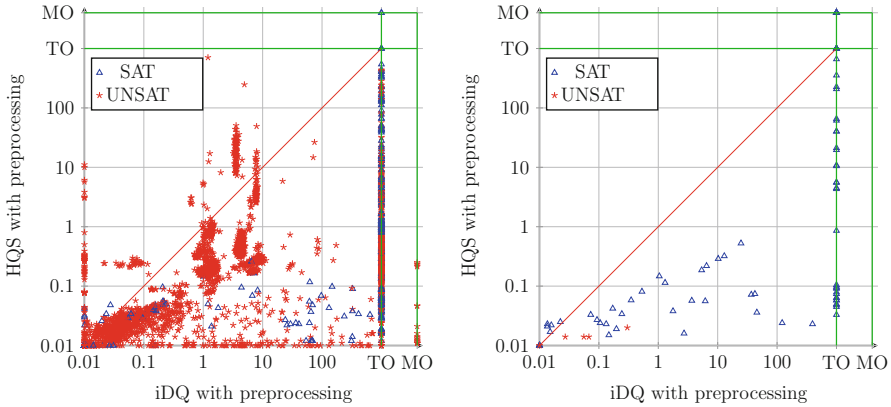

Fig. 7.6 Computation times (in seconds) of HQS and iDQ (both with preprocessing) on PEC instances [15, 17, 20] (*left*) and instances from controller synthesis [4] (*right*)

Figure 7.6 shows the results for incomplete combinational circuits (left, 3686 instances) and instances from controller synthesis (right, 89 instances) for those instances that could be solved by at least one solver; the remaining 543 instances could not be solved. The controller synthesis instances are incomplete sequential circuits with a single black box that can read all state bits and all primary inputs. We can observe in both cases, that HQS (with few exceptions) is more efficient and solves considerably more instances than iDQ. A closer look shows that the computation time and memory consumption is strongly influenced by the number of universal variables which have to be eliminated in order to obtain an equisatisfiable QBF. This is caused by the copies of the existential variables that are created when eliminating universal variables.

In spite of the improvements made during the last few years, the size of the instances that can effectively be solved is smaller by roughly one to two orders of magnitude than solvable QBF instances—strongly dependent on the number of variable copies which are created to obtain an equisatisfiable QBF.

The second set of experiments concerns the computation of Skolem functions for satisfiable DQBFs. We first measured the overhead of collecting the necessary data for computing Skolem functions during the solution process, i.e., until the truth value of the formula has been determined. The results are shown in Fig. 7.7. We can observe that the overhead is in most cases negligible—in a very few cases, the memory consumption is even reduced. The reason for this behavior is that within the AIG package different optimizations like rewriting are triggered when certain thresholds are exceeded. This can lead to smaller AIGs and thus save memory (and computation time for the subsequent operations).

For all 720 satisfiable instances we were able to solve without Skolem functions, we could also compute Skolem functions. For these instances we compare the sizes of the Skolem functions with and without optimizations using interpolation and ABC. Figure 7.8 displays the results. In many cases, we can reduce the sizes of the Skolem functions considerably, sometimes by up to two orders of magnitude.

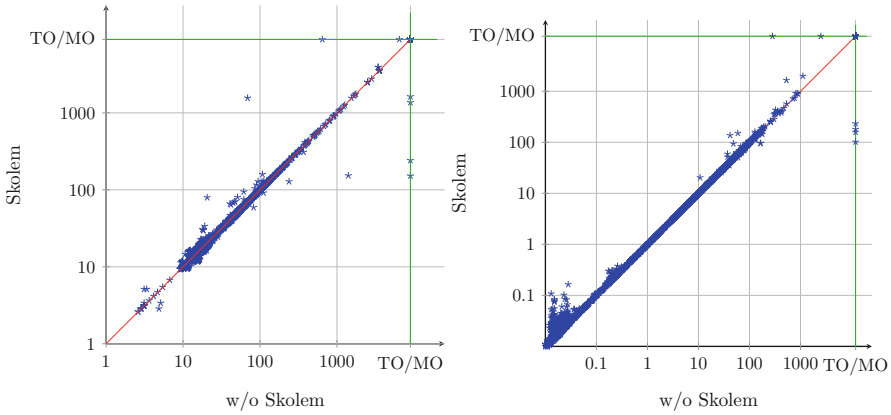

Fig. 7.7 Overhead of Skolem function computation on memory consumption (*left*) and running time (*right*)

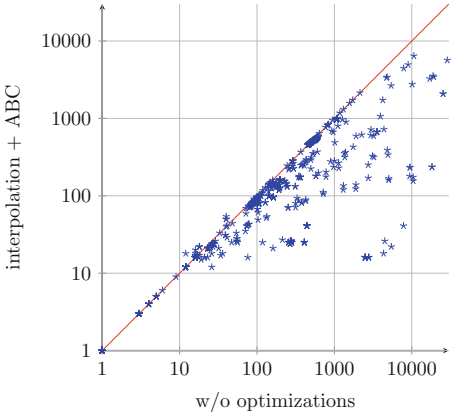

Fig. 7.8 Size of the computed Skolem functions with and without optimizations

Because QBFs are a special case of DQBFs, we can use HQS to compute Skolem functions for satisfied QBFs as well. In Fig. 7.9, we compare the sizes of the Skolem functions generated by HQS with those generated by the state-of-the-art QBF solver DepQBF 5.0 [22, 23] for a set of satisfiable QBF instances from the QBF Gallery 2013[1] and from partial equivalence checking [29] (with a single black box). Since HQS (and in particular its preprocessor) is not optimized for solving QBF instances, we abstain from a detailed comparison of the running times of HQS and DepQBF. DepQBF is often (but not always) faster than HQS. In a few cases, the generation of Skolem functions with DepQBF failed, because the necessary resolution proof became too large (we aborted DepQBF when the size of the dumped resolution proof exceeded 20 GB).

[1] See http://www.kr.tuwien.ac.at/events/qbfgallery2013/.

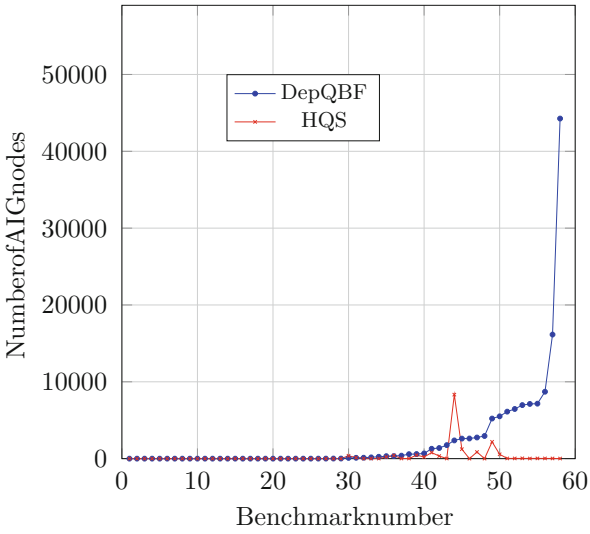

Fig. 7.9 Comparison of the sizes of Skolem functions from HQS and from DepQBF on QBF instances. The instances are ordered according to the size of DepQBF's Skolem functions

Figure 7.9 shows the sizes of the Skolem functions computed by DepQBF and by HQS (with interpolation and ABC). To enable a fair comparison, we also applied ABC with the same commands to the Skolem functions generated using DepQBF. We can observe that HQS' Skolem functions are in most cases smaller (often significantly) than those obtained from DepQBF.

In summary, the experiments show that HQS is able to solve the DQBFs resulting from small to medium-sized circuits effectively. We can not only obtain a pure yes/no answer, but also Skolem functions for the satisfiable instances without significant overhead. On satisfiable *QBF* instances, the size of the Skolem functions computed by HQS is similar, in many cases smaller in comparison to Skolem functions computed by DepQBF.

As a side remark, the example provided in Sect. 3 can easily be solved with a recent DQBF solver within fractions of a second. Benchmarks dealing with the synthesis of multiple black boxes in sequential circuits currently do not exist, but would be interesting to have.

6 Conclusion and Open Challenges

This paper has shown that DQBF formulations allow to express the realizability of invariant properties for incomplete combinational and sequential circuits with an arbitrary number of black boxes in a natural way. First prototypic solvers allow not only to solve the resulting DQBFs for small to medium-sized circuits, but also to extract Skolem functions, which can serve as implementations of the missing parts.

Still, many challenges remain: The scalability of the solvers has to be improved and might be tuned towards specific applications. More powerful preprocessing techniques are necessary as well as improvements in the solver core. We hope that with the availability of solvers more applications of these techniques become feasible (distributed controller synthesis) or are newly discovered thereby inspiring further improvements of the solvers—just as it was for propositional SAT solving and is for QBF solving.

References

1. P. Ashar, M.K. Ganai, A. Gupta, F. Ivancic, Z. Yang, Efficient SAT-based bounded model checking for software verification, in *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, ed. by T. Margaria, B. Steffen, A. Philippou, M. Reitenspieß, Technical Report, Paphos, Cyprus, vol. TR-2004-6 (Department of Computer Science, University of Cyprus, 2004), pp. 157–164
2. V. Balabanov, H.-J. Katherine Chiang, J.-H.R. Jiang, Henkin quantifiers and Boolean formulae: a certification perspective of DQBF. Theor. Comput. Sci. **523**, 86–100 (2014)
3. A. Biere, A. Cimatti, E.M. Clarke, O. Strichman, Y. Zhu, Bounded model checking. Adv. Comput. **58**, 117–148 (2003)
4. R. Bloem, R. Könighofer, M. Seidl, SAT-based synthesis methods for safety specs, in *Proceedings of VMCAI*, ed. by K.L. McMillan, X. Rival. Lecture Notes in Computer Science, San Diego, CA, vol. 8318 (Springer, Berlin, 2014), pp. 1–20
5. R. Bloem, U. Egly, P. Klampfl, R. Könighofer, F. Lonsing, M. Seidl, Satisfiability-based methods for reactive synthesis from safety specifications. CoRR, abs/1604.06204 (2016), http://arxiv.org/abs/1604.06204
6. R.K. Brayton, A. Mishchenko, ABC: an academic industrial-strength verification tool, in *Proceedings of CAV*, ed. by T. Touili, B. Cook, P. Jackson. Lecture Notes in Computer Science, Edinburgh, vol. 6174 (Springer, Berlin, 2010), pp. 24–40
7. U. Bubeck, Model-based transformations for quantified Boolean formulas. PhD thesis, University of Paderborn (2010)
8. U. Bubeck, H. Kleine Büning, Dependency quantified Horn formulas: models and complexity, in *Proceedings of SAT*, ed. by A. Biere, C.P. Gomes. Lecture Notes in Computer Science, Seattle, WA, vol. 4121 (Springer, Berlin, 2006), pp. 198–211
9. E.M. Clarke, A. Biere, R. Raimi, Y. Zhu, Bounded model checking using satisfiability solving. Formal Methods Syst. Des. **19**(1), 7–34 (2001)
10. S.A. Cook, The complexity of theorem-proving procedures, in *Proceedings of STOC* (ACM, New York, 1971), pp. 151–158
11. A. Czutro, I. Polian, M.D.T. Lewis, P. Engelke, S.M. Reddy, B. Becker, Thread-parallel integrated test pattern generator utilizing satisfiability analysis. Int. J. Parallel Prog. **38**(3–4),185–202 (2010)
12. W. Damm, B. Finkbeiner, Automatic compositional synthesis of distributed systems, in *Proceedings of FM*, ed. by C.B. Jones, P. Pihlajasaari, J. Sun. Lecture Notes in Computer Science, Singapore, vol. 8442 (Springer, Berlin, 2014), pp. 179–193
13. S. Eggersglüß, R. Drechsler, A highly fault-efficient SAT-based ATPG flow. IEEE Des. Test Comput. **29**(4), 63–70 (2012)
14. B. Finkbeiner, S. Schewe, Bounded synthesis. Int. J. Softw. Tools Technol. Transfer **15**(5–6), 519–539 (2013)
15. B. Finkbeiner, L. Tentrup, Fast DQBF refutation, in *Proceedings of SAT*, ed. by C. Sinz, U. Egly. Lecture Notes in Computer Science, Vienna, vol. 8561 (Springer, Berlin, 2014), pp. 243–251

16. A. Fröhlich, G. Kovásznai, A. Biere, A DPLL algorithm for solving DQBF, in *International Workshop on Pragmatics of SAT (POS)*, Trento (2012)
17. A. Fröhlich, G. Kovásznai, A. Biere, H. Veith, iDQ: instantiation-based DQBF solving, in *International Workshop on Pragmatics of SAT (POS)*, ed. by D. Le Berre. EPiC Series, Vienna, vol. 27 (EasyChair, 2014), pp. 103–116
18. K. Gitina, S. Reimer, M. Sauer, R. Wimmer, C. Scholl, B. Becker, Equivalence checking for partial implementations revisited, in *Proceedings of MBMV*, ed. by C. Haubelt, D. Timmermann, Rostock (Universität Rostock, ITMZ, 2013), pp. 61–70
19. K. Gitina, S. Reimer, M. Sauer, R. Wimmer, C. Scholl, B. Becker, Equivalence checking of partial designs using dependency quantified Boolean formulae, in *Proceedings of ICCD*, Asheville, NC (IEEE CS, 2013), pp. 396–403
20. K. Gitina, R. Wimmer, S. Reimer, M. Sauer, C. Scholl, B. Becker, Solving DQBF through quantifier elimination, in *Proceedings of DATE*, Grenoble (IEEE, New York, 2015)
21. M. Herbstritt, B. Becker, C. Scholl, Advanced SAT-techniques for bounded model checking of blackbox designs, in *Proceedings of MTV* (IEEE, New York, 2006), pp. 37–44
22. F. Lonsing, A. Biere, DepQBF: a dependency-aware QBF solver. J. Satisf. Boolean Model. Comput. **7**(2–3), 71–76 (2010)
23. F. Lonsing, F. Bacchus, A. Biere, U. Egly, M. Seidl, Enhancing search-based QBF solving by dynamic blocked clause elimination, in *Proceedings of LPAR*, ed. by M. Davis, A. Fehnker, A. McIver, A. Voronkov. Lecture Notes in Computer Science, Suva, vol. 9450 (Springer, Berlin, 2015), pp. 418–433
24. K.L. McMillan, Applications of Craig interpolants in model checking, in *Proceedings of TACAS*, ed. by N. Halbwachs, L.D. Zuck. Lecture Notes in Computer Science, Edinburgh, vol. 3440 (Springer, Berlin, 2005), pp. 1–12
25. T. Nopper, C. Scholl, Symbolic model checking for incomplete designs with flexible modeling of unknowns. IEEE Trans. Comput. **62**(6), 1234–1254 (2013)
26. G. Peterson, J. Reif, S. Azhar, Lower bounds for multiplayer non-cooperative games of incomplete information. Comput. Math. Appl. **41**(7–8), 957–992 (2001)
27. F. Pigorsch, C. Scholl, Exploiting structure in an AIG based QBF solver, in *Proceedings of DATE* (IEEE, New York, 2009), pp. 1596–1601
28. A. Pnueli, R. Rosner, Distributed reactive systems are hard to synthesize, in *Annual Symposium on Foundations of Computer Science*, St. Louis, MO (IEEE Computer Society, Washington, 1990), pp. 746–757
29. C. Scholl, B. Becker, Checking equivalence for partial implementations, in *Proceedings of DAC*, Las Vegas, NV (ACM, New York, 2001), pp. 238–243
30. C.-J.H. Seger, R.E. Bryant, Formal verification by symbolic evaluation of partially-ordered trajectories. Formal Methods Syst. Des. **6**(2), 147–189 (1995)
31. C.-J.H. Seger, R.B. Jones, J.W. O'Leary, T.F. Melham, M. Aagaard, C. Barrett, D. Syme, An industrially effective environment for formal hardware verification. IEEE Trans. CAD Integr. Circuits Syst. **24**(9), 1381–1405 (2005)
32. G.S. Tseitin, On the complexity of derivation in propositional calculus, in *Studies in Constructive Mathematics and Mathematical Logic Part 2* (Springer, Berlin, 1970), pp. 115–125
33. R. Wimmer, K. Gitina, J. Nist, C. Scholl, B. Becker, Preprocessing for DQBF, in *Proceedings of SAT*, ed. by M. Heule, S. Weaver. Lecture Notes in Computer Science, Austin, TX, vol. 9340 (Springer, Berlin, 2015), pp. 173–190
34. K. Wimmer, R. Wimmer, C. Scholl, B. Becker, Skolem functions for DQBF, in *Proceedings of ATVA*, Lecture Notes in Computer Science, Chiba, vol. 9938 (Springer, Berlin, 2016), pp. 395–411
35. K. Wimmer, R. Wimmer, C. Scholl, B. Becker, Skolem functions for DQBF (extended version). Technical Report, FreiDok, Freiburg im Breisgau (2016), https://www.freidok.uni-freiburg.de/data/11130
36. R. Wimmer, S. Reimer, P. Marin, B. Becker, HQSpre–an effective preprocessor for QBF and DQBF, in *Proceedings of TACAS, Part I*, ed. by A. Legay, T. Margaria. Lecture Notes in Computer Science, Uppsala, vol. 10205 (Springer, Berlin, 2017)