Preprint from Proceedings of
14th International Workshop on Microprocessor Test and Verification, Austin, TX, USA, 2013

Proving QBF-Hardness in Bounded Model Checking for Incomplete Designs

Christian Miller and Christoph Scholl and Bernd Becker
Institute of Computer Science
University of Freiburg, Germany
millerc|scholl|becker@informatik.uni-freiburg.de

***Abstract*—Bounded Model Checking (BMC) is a major verification technique for finding errors in sequential circuits by unfolding the design iteratively and converting the BMC instances into Boolean satisfiability (SAT) formulas. Here, we consider *incomplete* designs (i.e. those containing so-called black boxes) where the verification task is to prove unrealizability of a property. A property is called unrealizable by an incomplete design, if there is an error which can not be compensated by any implementation of the black boxes. While 01X-modeling of the unknown behavior of the black boxes yields easy-to-solve SAT problems, the logic of quantified Boolean formulas (QBF) is needed for 01X-hard problems to obtain a more precise modeling. However, QBF-modeling does not guarantee success in proving unrealizability. To this purpose, we introduce the concept of QBF-hardness in this paper, a classification of problems for which the QBF-based modeling does not provide a result. Furthermore, we present an iterative method to prove the QBF-hardness. We provide a first practical example (a parameterized incomplete arbiter bus system) to demonstrate the concept.**

I. Introduction

Bounded Model Checking (BMC) is a verification method for finding errors in sequential circuits [1], [2] and one of the most successful applications of the Boolean satisfiability problem (SAT). Starting with the initial state, the BMC procedure iteratively unfolds a circuit k times, connects the negated property to the last time frame, and finally converts the BMC instance into a SAT formula. If a SAT solver finds the k-th problem instance satisfiable, a path of length k violating the property has been found.

In this paper we focus on BMC for *incomplete* designs, meaning that certain parts of the circuit (combined into a so-called black box) are not specified. The purpose is to add a layer of abstraction if a design is too large to verify in its entirety, or allow to start the verification process earlier when certain components of the design are only partially completed. Verification of incomplete designs is emerging as larger system-on-chip (SoC) designs, containing multiple black box IP cores, are becoming more prevalent. The aim of solving black box BMC problems is to address the question of *unrealizability* of a property, where finding a path of length k proves that the property is violated regardless of the implementation of the black box. One option to model the unknown behavior of the black box is to extend Boolean logic by a third value X which then is applied to all black box outputs. Using a two-valued encoding, the BMC problems still yield SAT formulas, however, this so-called 01X-modeling might be too coarse for some problems consecutively producing unsatisfiable BMC instances. To this purpose the concept of 01X-hardness was introduced in [3]. A fully automatic procedure based on Craig interpolation was proposed which allows the classification of problems for which 01X-modeling does not lead to success. For those 01X-hard designs we make use of universally quantified variables to model the unknown behavior of the black box outputs, since this modeling allows for enough precision to prove unrealizability even if the counterexamples depend on the behavior of the black box [3]–[5]. BMC using the so-called QBF-modeling yields quantified Boolean formulas (QBF), and therefore necessitates the use of a QBF solver.

Under certain conditions even QBF-modeling is not accurate enough to prove unrealizability. In those cases (and, of course, in cases when the property is indeed realizable) the BMC procedure naturally fails and continuously produces unsatisfiable QBF formulas. To this purpose we introduce the idea of QBF-hardness in this paper as the final step in our verification workflow which we presented in [3]. We provide an iterative method to prove QBF-hardness. By a proof of QBF-hardness we prevent the QBF-based BMC procedure from running into unsatisfiable unfoldings forever. To show the applicability of the concept we consider a parametrized incomplete arbiter bus system where the implementation of the arbiter component is left open.

The paper is structured as follows. Section II will first introduce the reader to QBF logic and BMC for incomplete designs. The contribution of this paper is presented in Section III. Here, we give a definition of QBF-hardness and provide a QBF-based method to prove this property of an incomplete design. Section IV will then give preliminary results demonstrating the applicability of our concept. Finally, Section V concludes the paper.

II. Preliminaries

This section will give a brief introduction to the notation and the formalisms used throughout this paper. It will start by introducing quantified Boolean formulas, and then move on to QBF-based BMC of incomplete designs.

A. QBF Logic

QBF formulas are a generalization of pure propositional Boolean formulas where variables are either existentially or

This work has been partially funded by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS, www.avacs.org).

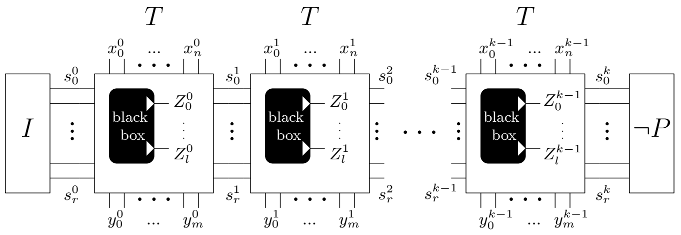


Fig. 1. Encoding of the BMC Problem for Incomplete Designs.

universally quantified. Most modern QBF solvers require the problem to be formatted in *Quantified Conjunctive Normal Form* (QCNF):

$$\psi = Q_1 x_1 . Q_2 x_2 . \dots Q_n x_n . \phi \tag{1}$$

where $n \in \mathbb{N}$, $Q_i \in \{\forall, \exists\}$ for all $1 \leq i \leq n$ is the *prefix*, and ϕ, named *matrix*, is the propositional part in conjunctive normal form (CNF) over $\{x_1, \dots, x_n\}$; this is a conjunction of *clauses*, which are in turn disjunctions of literals. A *literal* is a variable or its negation. For our QBF problems, we expect the formula to be closed, meaning that each variable is quantified in the prefix.

$$\exists x_1 \forall x_2 \exists x_3 : (x_1 \lor x_2) \land (\neg x_1 \lor x_2 \lor x_3) \tag{2}$$

Eqn. (2) shows a small illustration of a QCNF formula which can be read as follows:

> Does there exist an assignment x_1, such that for all the possible assignments to x_2 there is also an assignment to x_3 such that the formula is always satisfied?

This means that if a simple search based solver started with $x_1 = 1$, it would have to prove that a solution is possible for both $x_2 = 1$ and $x_2 = 0$ (which requires the solver to find two separate solutions). In this case, the formula is satisfiable as the two solutions are $(x_1 = 1, x_2 = 1, x_3 = 0)$ and $(x_1 = 1, x_2 = 0, x_3 = 1)$. For more details on QBF logic, semantics and solving techniques, the interested reader is referred to [6].

B. BMC for Incomplete Designs

BMC for incomplete designs aims to refute the realizability of a property, that is, it tells the designer, no matter how the unknown parts of the system will be implemented, the property will always fail. To put it in other words, the error is already in the implemented system. If this is the case, then we call the property P *unrealizable*. In this paper we restrict the properties to *invariants* and we make use of QBF modeling [3]–[5] where the variables representing the black box outputs are universally quantified. We allow black box replacements to have arbitrary sequential behavior, that is, the black box can produce different output values for the same input values at different time steps. To encode the BMC problem of incomplete designs we are naming the variables as shown in Figure 1. We use an upper index to specify the time instance of a variable. s_i^j denotes the i-th state bit in the j-th unfolding (let $\vec{s}^j = s_0^j, \dots, s_r^j$). The same holds for the primary inputs $\vec{x}^j = x_0^j, \dots, x_n^j$, the primary outputs $\vec{y}^j = y_0^j, \dots, y_m^j$, and the black box outputs $\vec{Z}^j = Z_0^j, \dots, Z_l^j$. The next state variables $\vec{s}^{j+1}$ depend on the current state, the primary inputs and the black box outputs. The whole circuit is transformed according to [7] using additional auxiliary variables $\vec{H}^j$ for each unfolding depth j. The predicate describing the initial states is given by $I(\vec{s}^0)$. Since we assume a single initial state in this paper, the initial state $I(\vec{s}^0)$ is encoded by unit clauses, setting the respective state bits to their initial value. The transition relation of time frame i is given by $T(\vec{s}^{i-1}, \vec{x}^{i-1}, \vec{Z}^{i-1}, \vec{s}^i)$. The invariant $P(\vec{s}^k)$ is a Boolean expression over the state variables [1] of the k-th unfolding. Using this information, the quantifier prefix (and the matrix) for the unrealizability problem results in the QBF formula (3) as proposed in [3]. For the sake of simplicity we include the variables representing the primary outputs of unfolding depth j into $\vec{H}^j$.

$$\begin{array}{llll} \mathit{BMC}(k) := & \exists\, \vec{s}^0\ \vec{x}^0 & \forall\, \vec{Z}^0 & \exists\, \vec{H}^0 \\ & \exists\, \vec{s}^1\ \vec{x}^1 & \forall\, \vec{Z}^1 & \exists\, \vec{H}^1 \\ & & \vdots & \\ & \exists\, \vec{s}^{k-1}\ \vec{x}^{k-1} & \forall\, \vec{Z}^{k-1} & \exists\, \vec{H}^{k-1} \\ & \exists\, \vec{s}^k & & \end{array}$$

$$I(\vec{s}^0) \land \bigwedge_{i=1}^{k} T(\vec{s}^{i-1}, \vec{x}^{i-1}, \vec{Z}^{i-1}, \vec{s}^i) \land \neg P(\vec{s}^k) \tag{3}$$

The semantics following from the prefix corresponds to the following question:

> Does there exist a state $\vec{s}^0 = s_0^0, \dots, s_r^0$ and an input vector $\vec{x}^0 = x_0^0, \dots, x_n^0$ at depth 0 such that for all possible values of the black box outputs $\vec{Z}^0 = Z_0^0, \dots, Z_m^0$ there exists an assignment to all auxiliary variables $\vec{H}^0$ (resulting in a next state $\vec{s}^1 = s_0^1, \dots, s_r^1$) and an input vector $\vec{x}^1 = x_0^1, ..., x_n^1$ at depth 1, etc. such that the property is violated?

The BMC procedure iteratively unfolds the incomplete circuit for $k = 0, \dots, K$ until a predefined maximal unfolding depth K is reached. If a QBF solver finds $\mathit{BMC}(k)$ satisfiable, the unrealizability of the property P has been proven. In that case the resulting system can reach a "bad state" after k steps, no matter how the black box is implemented.

We can prove that, whenever $\mathit{BMC}(k)$ is *unsatisfiable*, there is an implementation of the black box which is able to avoid error paths of length k *as long as the black box is allowed to read all primary inputs*. However, if the black box in the design at hand is not directly connected to all primary inputs, (i.e. if the black box does not have "complete information"), such an implementation does not need to exist. Thus, for black boxes having "incomplete information" the property may be unrealizable although BMC with QBF modeling is not able to prove this.

Example 1. Consider the incomplete circuit shown in Figure 2. The state bits s_0 and s_1 depend on the current state, the primary input x, and the black box outputs Z_0 and Z_1, respectively, and are computed by the transition functions $s_0' = x + Z_0$ and $s_1' = x \cdot Z_1 + s_0 \cdot \neg Z_1$. Let the invariant property $P = \neg(s_0 \land s_1)$ state that s_0 and s_1 must never be 1

[1]In general the property can also check the primary outputs, but for sake of convenience, we omit details in the paper.

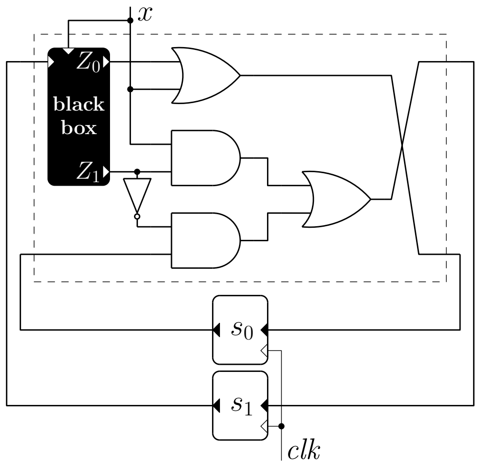
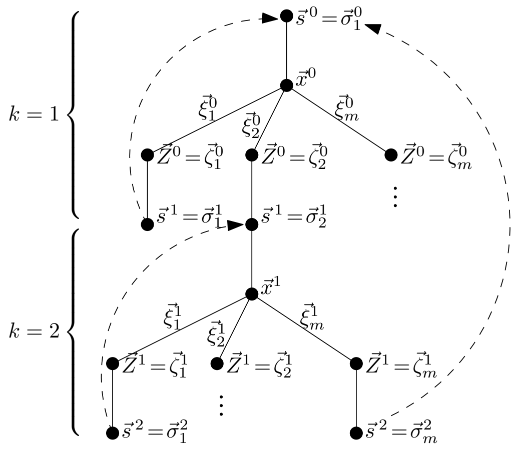


Fig. 2. Example Incomplete Design.



$k=1$
$k=2$
$\vec{s}^0=\vec{\sigma}_1^0$
$\vec{x}^0$
$\vec{\xi}_1^0$
$\vec{\xi}_2^0$
$\vec{\xi}_m^0$
$\vec{Z}^0=\vec{\zeta}_1^0$
$\vec{Z}^0=\vec{\zeta}_2^0$
$\vec{Z}^0=\vec{\zeta}_m^0$
$\vec{s}^1=\vec{\sigma}_1^1$
$\vec{s}^1=\vec{\sigma}_2^1$
$\vec{x}^1$
$\vec{\xi}_1^1$
$\vec{\xi}_2^1$
$\vec{\xi}_m^1$
$\vec{Z}^1=\vec{\zeta}_1^1$
$\vec{Z}^1=\vec{\zeta}_2^1$
$\vec{Z}^1=\vec{\zeta}_m^1$
$\vec{s}^2=\vec{\sigma}_1^2$
$\vec{s}^2=\vec{\sigma}_m^2$

Fig. 3. QBF-Hardness Graph.

at the same time. Let the initial state of the system be defined as $s_0^0 = s_1^0 = 0$. After checking for an initial violation of the property, the BMC procedure unfolds the system once, and tries to find an assignment to x^0 such that for all possible assignments to Z_0^0 and Z_1^0 the state $(1,1)$ can be reached. Indeed, $x^0 = 1$ implies $s_0 = 1$ for all assignments to the black box outputs, however, for $Z_1^0 = 0$ $s_1 = 1$ can not be obtained (neither by setting $x^0 = 0$ nor $x^0 = 1$). Thus, *BMC*(1) is unsatisfiable and BMC continues by adding a second copy of the transition relation to the problem. If $x^0 = 1$, the current state bit s_0^1 at the second unfolding evaluates to 1 as well. Furthermore, if x^1 is set to 1, the next state bits s_0^2 and s_1^2 evaluate to 1 for all values of Z_0^1 and Z_1^1. Hence, when applying the input pattern $x^0 = x^1 = 1$, a state violating P can be reached after two steps for all actions of the black box and thus, *BMC*(2) is satisfiable and P is unrealizable.

III. QBF-Hardness

As already discussed in Section II-B, under certain conditions (black boxes having "incomplete information") the BMC procedure using a QBF formulation is not able to prove unrealizability even if the property is indeed unrealizable. In this sense the QBF formulation is a sound but incomplete approximation (just as 01X-modeling which is also an approximation, but is strictly coarser). If unrealizability can not be proven due to the approximative nature of the method or if the property is really realizable, then the BMC procedure as described in Section II-B would produce unsatisfiable QBF formulas for all unfoldings and would never return a result. This motivates the following definition of *QBF-hardness*:

Definition 1. An incomplete design with an invariant P is *QBF-hard* iff the QBF-modeled BMC problem is unsatisfiable for every unfolding.

A. Proving QBF-Hardness

The idea of proving QBF-hardness is as follows: The QBF-based BMC procedure classifies a property as unrealizable, iff there exist input sequences of some length k such that independently from the black box actions the property will be violated after k steps. Conversely, the QBF-based BMC procedure is not able to prove unrealizability with an unfolding of length k or smaller, if for each input valuation in each time frame there is an action of the black box such that the property is fulfilled after k steps, and additionally all states on these paths also fulfill the property. Furthermore, if we can prove for this scenario that after at most k steps every state has already been visited before, we can be sure that the QBF-based BMC procedure will *never* produce a satisfiable instance, since for every input pattern it is possible to determine at least one realization of the black box leading to a state which does not violate the property, independently from the length of the unfolding.

This concept is illustrated in Figure 3. Let $\vec{s}^0 = \vec{\sigma}_1^0$ be the initial state which fulfills P. Next, the graph branches for all possible assignments $\vec{\xi}_1^0, \ldots, \vec{\xi}_m^0$ to the primary inputs $\vec{x}^0$. For each of these values $\vec{\xi}_i^0$ there exists an action of the black box outputs $\vec{Z}^0 = \vec{\zeta}_i^0$ leading to next states $\vec{s}^1 = \vec{\sigma}_i^1$ which all fulfill P. Once a state is equivalent to a state which was visited before (which is indicated by a dashed backward arrow in Figure 3 stating that $\vec{\sigma}_1^1 = \vec{\sigma}_1^0$, $\vec{\sigma}_1^2 = \vec{\sigma}_2^1$, $\vec{\sigma}_m^2 = \vec{\sigma}_1^0$, respectively), this branch does not need to be further explored. If at some depth all so far explored states point back to already visited states, then the black box outputs are set in a way that the system remains in "good states" forever, i.e., we are in the situation sketched above and we can be sure that the QBF-based BMC procedure will never produce a satisfiable instance, independently from the length of the unfolding. Thus, determining whether a graph fulfilling the aforementioned properties exists answers the question of whether a design is QBF-hard.

Example 2. The incomplete design in Figure 4 is a slight modification of the circuit in Figure 2 where the types of the gate computing s_0 and the gate having s_0 and $\neg Z_1$ as inputs have switched. Hence the transition functions changed to $s_0' = x \cdot Z_0$ and $s_1' = x \cdot Z_1 + \neg Z_1 + s_0$. Again, the invariant $P = \neg(s_0 \wedge s_1)$ states that s_0 and s_1 must never be 1 at the same time. BMC for incomplete designs fails to prove unrealizability, since it is impossible to simultaneously assign the value 1 to both state bits for all possible values of Z_0 and Z_1. Rather, the QBF-hardness procedure finds the QBF-hardness graph depicted in Figure 5 at the second iteration step. To put it in words, the black box can be chosen in such a way that the system – for all traces of the system implied by

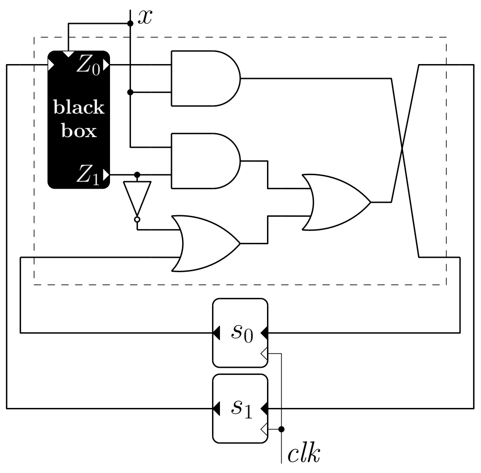


Fig. 4. QBF-Hard Incomplete Design

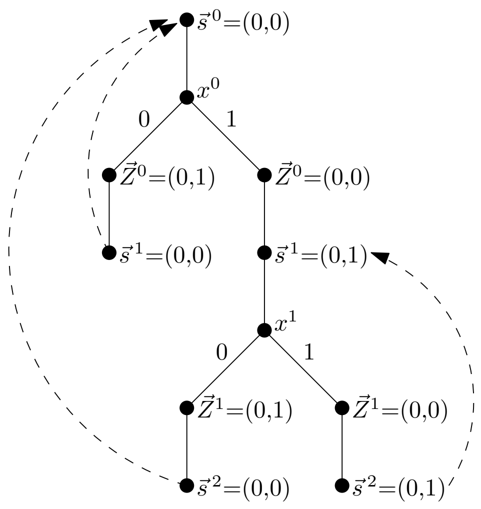


Fig. 5. Example QBF-Hardness Graph.

different valuations of input x – only reaches the states (0,0) and (0,1), and thus, P is never violated.

In fact, we can prove the following theorem:

Theorem 1. For black boxes with complete information (i.e. black boxes which are connected to all primary inputs) QBF-hardness coincides with realizability.

Proof: (Sketch) Consider the QBF-hardness graph sketched in Figure 3. All valuations of states $\vec{s}^{\,i}$ are implied by the valuations for the variables $\vec{x}^0, \ldots, \vec{x}^{i-1}$, $\vec{Z}^0, \ldots, \vec{Z}^{i-1}$ before. The valuation of $\vec{s}^{\,0}$ is fixed, since we assume a single initial state. Thus, the $\vec{s}^{\,i}$-nodes have exactly one outgoing edge and edges leading to them can be replaced by edges to their unique successor. Afterwards all $\vec{s}^{\,i}$-nodes are removed. The resulting graph fixes reactions of the black box outputs to primary inputs which guarantee correctness. The graph can be seen as a finite program with cycles which a black box monitoring all primary inputs may execute in order to realize the property. ■

- $k = 0:$
  $\exists \vec{s}^0\ I(\vec{s}^0) \cdot P(\vec{s}^0)$

- $k = 1:$
  $\exists \vec{s}^0\ (I(\vec{s}^0) \cdot P(\vec{s}^0) \cdot$
  $\forall \vec{x}^0 \exists \vec{Z}^0 \vec{H}^0 \vec{s}^1\ T(\vec{s}^0, \vec{x}^0, \vec{Z}^0, \vec{s}^1) \cdot \mathit{BL}(\vec{s}^0, \vec{s}^1))$

- $k = 2:$
  $\exists \vec{s}^0\ (I(\vec{s}^0) \cdot P(\vec{s}^0) \cdot$
  $\forall \vec{x}^0 \exists \vec{Z}^0 \vec{H}^0 \vec{s}^1\ T(\vec{s}^0, \vec{x}^0, \vec{Z}^0, \vec{s}^1) \cdot (\mathit{BL}(\vec{s}^0, \vec{s}^1) + P(\vec{s}^1) \cdot$
  $\forall \vec{x}^1 \exists \vec{Z}^1 \vec{H}^1 \vec{s}^2\ T(\vec{s}^1, \vec{x}^1, \vec{Z}^1, \vec{s}^2) \cdot \mathit{BL}(\vec{s}^0, \vec{s}^1, \vec{s}^2)))$

- $k = 3:$
  $\exists \vec{s}^0\ (I(\vec{s}^0) \cdot P(\vec{s}^0) \cdot$
  $\forall \vec{x}^0 \exists \vec{Z}^0 \vec{H}^0 \vec{s}^1\ T(\vec{s}^0, \vec{x}^0, \vec{Z}^0, \vec{s}^1) \cdot (\mathit{BL}(\vec{s}^0, \vec{s}^1) + P(\vec{s}^1) \cdot$
  $\forall \vec{x}^1 \exists \vec{Z}^1 \vec{H}^1 \vec{s}^2\ T(\vec{s}^1, \vec{x}^1, \vec{Z}^1, \vec{s}^2) \cdot (\mathit{BL}(\vec{s}^0, \vec{s}^1, \vec{s}^2) + P(\vec{s}^2) \cdot$
  $\forall \vec{x}^2 \exists \vec{Z}^2 \vec{H}^2 \vec{s}^3\ \ T(\vec{s}^2, \vec{x}^2, \vec{Z}^2, \vec{s}^3)\ \cdot\ \mathit{BL}(\vec{s}^0, \vec{s}^1, \vec{s}^2, \vec{s}^3))))$

$$\vdots$$

Fig. 6. Iterative Procedure to Prove QBF-Hardness.

B. Generating QBF Formulas for the QBF-Hardness Procedure

The next step is to formulate the existence of such graphs as QBF problems (Figure 6 lists the QBF formulas of the first four iterations of our QBF-hardness procedure). First ($k = 0$) we test if all initial states fulfill P.[2] If not, the property is unrealizable and the procedure for QBF-hardness stops. Otherwise, we check in the next iteration ($k = 1$) whether for every input pattern there exists a black box output pattern such that after one transition the resulting state is the initial state again. Since at the beginning we identified the initial state as a "good state", satisfiability of this QBF formula implies QBF-hardness of the incomplete design. However, for those states $\vec{s}^{\,1}$ which are not equivalent to the initial state, we continue to check the properties of the QBF-hardness graph for the next unfolding depth, that is $\vec{s}^{\,1}$ has to fulfill P and for all input patterns there has to be at least one black box action such that the resulting state $\vec{s}^{\,2}$ either is equivalent to $\vec{s}^{\,0}$ or to $\vec{s}^{\,1}$, etc. This iterative procedure continues until a satisfiable instance could be found. Note, in Figure 6 the backloop (*BL*) predicate determines whether a state $\vec{s}^{\,k}$ has already been explored before:

$$\mathit{BL}(\vec{s}^{\,0}, ..., \vec{s}^{\,k}) := \bigvee_{i=0}^{k-1} (\vec{s}^{\,i} \equiv \vec{s}^{\,k})$$

Algorithm 1 gives the iterative procedure for proving QBF-hardness of an incomplete design as it is currently implemented in our verification tool. It produces exactly the formulas as in Figure 6 with the only difference that the quantifiers of the QBF formulas have been moved to left yielding equivalent QBF formulas in prenex normal form. After checking for an initial violation of P in lines 1 to 4 ($k = 0$), the main loop iterates from $k = 1 \ldots K$. Starting with the subformula of time frames $k-1$ to k in line 7, the QBF formula of the k-th step is build by generating the subformulas from

[2]Note that we assume a single initial state, i.e., $\exists \vec{s}^0\ I(\vec{s}^0) \cdot P(\vec{s}^0)$ is equivalent to $\forall \vec{s}^0\ (I(\vec{s}^0) \implies P(\vec{s}^0))$.

Algorithm 1: Proving QBF-Hardness

```
Input: init. state $I$, trans. rel. $T$, property $P$, max. depth $K$
Output: QBF-hardness
1  $\Phi = \exists \vec{s^0}\ I(\vec{s^0}) \cdot P(\vec{s^0})$;
2  if QBFSolve($\Phi$) == false then
3      return P initially violated
4  end
5  $k = 1$;
6  while $k \leq K$ do
7      $\Phi = P(\vec{s^{k-1}}) \cdot T(\vec{s^{k-1}}, \vec{x^{k-1}}, \vec{Z^{k-1}}, \vec{s^k}) \cdot BL(\vec{s^0}, \ldots, \vec{s^k})$;
8      for $i = k-1, \ldots, 1$ do
9          $\Phi = P(\vec{s^{i-1}}) \cdot T(\vec{s^{i-1}}, \vec{x^{i-1}}, \vec{Z^{i-1}}, \vec{s^i}) \cdot (BL(\vec{s^0}, \ldots, \vec{s^i}) + \Phi)$;
10     end
11     $\Phi = (I(\vec{s^0}) \cdot \Phi)$;
12     $\Phi = \forall \vec{s^0} \vec{x^0} \exists \vec{Z^0} \vec{s^1} \forall \vec{x^1} \exists \vec{Z^1} \ldots \exists \vec{s^{k-1}} \forall \vec{x^{k-1}} \exists \vec{Z^{k-1}} \vec{s^k}\ \Phi$;
13     if QBFSolve($\Phi$) == true then
14         return QBF-hard
15     end
16     $k = k + 1$;
17 end
18 return Unknown
```

$k-1$ downward to 1 in lines 8 to 10. Finally, the resulting propositional formula is connected to the check for the initial state in line 11. The quantifier prefix is generated at the end in line 12 of the algorithm. Before passing the formula to an arbitrary QBF solver, the matrix of the QBF formula (i.e. its propositional part) has to be translated into CNF. Assuming that the predicates I, P and T are already given by CNFs with auxiliary variables, the main problem is to remove *disjunctions* in the QBF formulas resulting from the QBF-hardness procedure (e.g. $\ldots BL(\vec{s}^0, \ldots, \vec{s}^i) + P(\vec{s}^i) \cdot T(\vec{s}^i, \vec{x}^i, \vec{Z}^i, \vec{s}^{i+1}) \ldots$). To encode the disjunction $(\varphi + \psi)$ of two CNFs φ and ψ we add one extra literal $\neg a$ to each clause of φ yielding φ', one extra literal $\neg b$ to each clause in ψ yielding ψ' and obtain the equisatisfiable CNF $\varphi' \wedge \psi' \wedge (a \vee b)$. If the QBF solver finds the instance satisfiable the QBF-hardness of the verification problem has been proven (lines 13 to 15 of the algorithm). Otherwise k is incremented for the next iteration step.

Remark 1. An overall algorithm which alternates for each k between a QBF-based check for unrealizability and a QBF-hardness check will stop (at latest) when k exceeds the reachability recurrence diameter RRD of the design (without the black box). The reachability recurrence diameter is the longest loop-free path starting from the initial state [8]. As soon as k exceeds RRD, either all paths in the QBF-hardness graph are closed by loops or unrealizability is proven by Eqn. (3). An alternative to proving QBF-hardness would be to compute the reachability recurrence diameter RRD and solve a formula similar to Eqn. (3) until RRD is exceeded. However, RRD may be excessively large for practical designs. The advantage of Algorithm 1 is that it always computes a *minimal* k which suffices to prove QBF-Hardness. Algorithm 1 increases k step by step and, implicitly, the QBF solver computes black box actions which "close the loops" in the QBF-hardness graph *as soon as possible*.

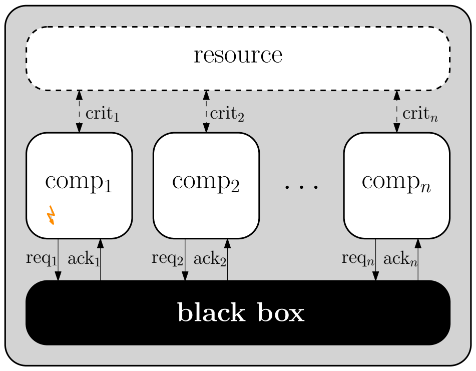


Fig. 7. Incomplete Arbiter Bus System.

IV. Experimental Results

A. Case Study

As a case study we use a parameterized arbiter bus system [9] consisting of n abstract components which access a shared resource (see Figure 7 for an illustration). A component i sends a request signal req$_i$ to the arbiter which grants access to the resource by an appropriate acknowledge signal ack$_i$. Then component i enters the resource by setting its crit$_i$ signal to 1. Additionally, the arbiter ensures that not more than one component can access the resource at the same time (mutual exclusion property). We inserted two kind of errors into comp$_1$:

- error-1: The ack$_1$ signal is incorrectly processed (in the sense that the critical section can even be entered if ack$_1$ is 0).
- error-2: The crit$_1$ signal is always 1.

Regardless of which error is inserted, comp$_1$ can enter the critical section without having the acknowledge of the arbiter. Thus, any other component could access the resource as well and as a consequence the mutual exclusion property would be violated. In the incomplete arbiter bus system the implementation of the arbiter is left open, i.e., unrealizability of the mutual exclusion property would mean that it is not possible to find an arbiter implementation guaranteeing mutual exclusion. However, since there is at least one implementation of the black box preserving the property (simply by granting access to none of the components), unrealizability of the mutual exclusion property can not be proven. Hence, this verification problem is QBF-hard. In total, for n components the instance has $15 \cdot n$ gates, $2 \cdot n$ latches, and n black box outputs.

B. Results

We integrated the QBF-hardness procedure into our QBF-based BMC tool for incomplete designs [10]–[12]. The evaluation was performed on one processor of a quad-core Intel Xeon E5-2643 processor running at 3.3 GHz with 32 GB of main memory. For each instance we set a timeout of 3600 seconds. Table I shows the results for proving QBF-hardness of the incomplete arbiter bus system and is divided into the results for error-1 and error-2, respectively. It gives the number of instantiated components (Column 1) and the depth at which QBF-hardness could be proven (Column 2). In order to have a broad variety of QBF solving techniques, we used the following QBF engines as back-end solvers:

error 01

comp.	k	QuBE	DepQBF	Quantor	AIGsolve	AQME
2	6	0.22	0.06	7.84	2.23	9.57
3	9	2.08	20.64	MO	344.89	14.89
4	12	26.57	TO	MO	MO	38.12
5	-	TO	TO	MO	MO	MO
6	-	TO	TO	MO	MO	MO

error 02

comp.	k	QuBE	DepQBF	Quantor	AIGsolve	AQME
2	3	0.01	0.00	0.02	0.23	5.24
3	4	0.06	0.04	7.24	4.77	3.91
4	5	0.42	5.85	MO	62.20	159.33
5	6	12.11	TO	MO	1968.03	9.72
6	7	2550.52	TO	MO	MO	123.02

TABLE I. Results Arbiter Bus System.

- *QuBE 7.2* [13], [14] is a search-based QBF solver that offered good performance on the black box benchmark set at the 2007 and 2008 QBF competitions.
- *DepQBF* [15] is a search-based QBF solver which won the QBFEVAL'10 competition [16].
- Quantor [17] uses Q-resolution for $\exists$-variables and expansion for $\forall$-variables until the resulting formula is purely propositional which is then solved by a SAT solver.
- *AIGsolve* [18], [19] is a QBF solver based on And-Inverter-graphs (AIGs) using quantifier elimination methods.
- AQME [20] is a self-adaptive multi-engine QBF solver which chooses the engine which is most likely to solve a given instance based on its syntactic features.

The solving times (Columns 3 to 7) are given in seconds. For error-1 the depth at which QBF hardness can be proven grows linearly in the numbers of components and using QuBE or AQME we were able to prove QBF-hardness for up to 4 components within the given timeout. A similar picture results for error-2. Here, the depth needed for proving QBF-hardness is proportional to the numbers of components as well (but it grows more slowly) and problem instantiations with up to 6 components could be solved. Generally speaking, the QBF formulas generated for proving QBF hardness seem to be harder to solve than the ones generated during the BMC process. This is not very surprising: Whereas optimizations for proving QBF-hardness are at the very beginning, we developed a series of optimizations to increase the efficiency of QBF-based BMC [10]–[12]: For instance we formulated the problem in a way that it is suitable for *incremental* QBF solving where the solution process for $\mathit{BMC}(k)$ profits from information learned during the solution of $\mathit{BMC}(i)$ with $i < k$, we extended a QBF solver to allow incremental solving, and we integrated incremental preprocessing techniques into QBF solving. At the moment it remains an open question whether QBF-hardness checks will be able to equal the success of QBF-based BMC in processing large industrial designs by modifications in the formulation of the QBF-hardness problems accounting for special strengths of QBF solvers and / or by QBF solver modifications supporting the QBF-hardness check.

Nevertheless, our first results show that QBF-hardness checks can be successfully applied in order to stop a potentially infinite (and thus expensive) series of QBF-based BMC checks in cases when the property is realizable or when a proof of unrealizability can not be found by QBF-based BMC due to its approximative nature.

V. Conclusion

In this paper we presented the concept of QBF-hardness to determine whether BMC for a QBF-modeled incomplete design will fail by endlessly producing unsatisfiable BMC instances. We provided a fully automatic QBF-based approach to prove QBF-hardness which was integrated into our BMC tool for incomplete designs. We showed the applicability of our approach using an incomplete arbiter bus system.

In the future we will work on further optimizations of the QBF-hardness check and we will investigate clever heuristics for tightly integrating the QBF-hardness procedure into the overall BMC procedure.

References

[1] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic Model Checking without BDDs," in *TACAS*, 1999, pp. 193–207.

[2] E. Clarke, A. Biere, R. Raimi, and Y. Zhu, "Bounded Model Checking Using Satisfiability Solving," *Formal Methods in System Design.*, pp. 7–34, 2001.

[3] C. Miller, S. Kupferschmid, M. Lewis, and B. Becker, "Encoding Techniques, Craig Interpolants and Bounded Model Checking for Incomplete Designs," in *Proc. SAT*, ser. LNCS, vol. 6175. Springer, 2010.

[4] M. Herbstritt, B. Becker, and C. Scholl, "Advanced SAT-Techniques for Bounded Model Checking of Blackbox Designs," in *Proc. Workshop on Microprocessor Test and Verification*, 2006, pp. 37–44.

[5] M. Herbstritt and B. Becker, "On Combining 01X-Logic and QBF," in *Proc. 11th Intl. Conf. on Computer Aided Systems Theory (EuroCAST)*. Springer, 2007.

[6] E. Giunchiglia, P. Marin, and M. Narizzano, *Reasoning with Quantified Boolean Formulas*, ser. Frontiers in Artificial Intelligence and Applications. IOS Press, February 2009, vol. 185, ch. 24, pp. 761–780.

[7] G. Tseitin, "On the Complexity of Proofs in Propositional Logics," *Seminars in Mathematics*, 1970.

[8] D. Kroening and O. Strichman, "Efficient computation of recurrence diameters," in *VMCAI*, ser. LNCS, vol. 2575. Springer, 2003, pp. 298–309.

[9] T. Nopper and C. Scholl, "Symbolic model checking for incomplete designs with flexible modeling of unknowns," *IEEE Transactions on Computers*, vol. 62, no. 6, pp. 1234–1254, 2013.

[10] P. Marin, C. Miller, M. Lewis, and B. Becker, "Verification of Partial Designs Using Incremental QBF Solving," in *DATE*, 2012.

[11] P. Marin, C. Miller, and B. Becker, "Incremental QBF Preprocessing for Partial Design Verification (Poster Presentation)," in *Proc. SAT*, 2012.

[12] C. Miller, P. Marin, and B. Becker, "A Dynamic QBF Preprocessing Approach for the Verification of Incomplete Designs," in *RCRA*, 2012.

[13] E. Giunchiglia, M. Paolo, and M. Narizzano, "QuBE7.0, System Description," *Journal of Satisfiability.*, vol. 7, no. 8, pp. 83–88, 2010.

[14] P. Marin, E. Giunchiglia, and M. Narizzano, "Conflict and Solution Driven Constraint Learning in QBF," in *Doctoral Program of Constraint Programming Conference*, 2010.

[15] F. Lonsing and A. Biere, "DepQBF: A Dependency-Aware QBF Solver," *JSAT*, vol. 7, no. 2-3, pp. 71–76, 2010.

[16] C. Peschiera, L. Pulina, A. Tacchella, U. Bubeck, O. Kullmann, and I. Lynce, "The Seventh QBF Solvers Evaluation (QBFEVAL'10)," in *SAT*, ser. LNCS, vol. 6175. Springer, 2010.

[17] A. Biere, "Resolve and expand." in *Proc. SAT*, 2004, pp. 59–70.

[18] F. Pigorsch and C. Scholl, "Exploiting Structure in an AIG Based QBF Solver," in *Conf. on Design, Automation and Test in Europe (DATE)*, April 2009, pp. 1596–1601.

[19] ——, "An AIG-based QBF-solver using SAT for preprocessing," in *Design Automation Conference, 2010 47th ACM/IEEE*, 2010, pp. 170 –175.

[20] L. Pulina and A. Tacchella, "A self-adaptive multi-engine solver for quantified Boolean formulas," *Constraints*, vol. 14, pp. 80–116, March 2009.